FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Anonymizace osobních údajů ve strukturovaných dokumentech
Student:	Bc. Jakub Doležal
Vedoucí:	Ing. Jiří Mlejnek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Seznamte se s existující implementací nástroje Winch, který slouží pro anonymizaci osobních údajů v relačních databázích. Analyzujte požadavky a navrhněte rozšíření tohoto nástroje, umožňující provádět anonymizaci osobních údajů ve strukturovaných dokumentech ve formátu XML a JSON. V rámci analytické části diskutujte možné přístupy k definici anonymizačního modelu, způsobu provádění anonymizace a vyberte nejvhodnější řešení.
Navržené řešení implementujte včetně uživatelského rozhraní, které usnadní definici anonymizačního modelu pro vybrané soubory. Množství implementovaných anonymizačních funkcí konzultujte s vedoucím práce. Připravte testovací data a celé řešení důkladně otestujte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 31. ledna 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Anonymizace osobních údajů ve strukturovaných dokumentech

Bc. Jakub Doležal

Katedra Softwarového inženýrství
Vedoucí práce: Ing. Jiří Mlejnek

3. května 2022

Poděkování

Chtěl bych poděkovat mojí rodině za podporu v době studií a při tvorbě této práce. Dále bych rád poděkoval mému vedoucímu Ing. Jiřímu Mlejnkovi za možnost vypracovat diplomovou práci pod jeho vedením.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

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

V Praze dne 3. května 2022 . . . . . .

Odkaz na tuto práci

Doležal, Jakub. *Anonymizace osobních údajů ve strukturovaných dokumentech*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato práce se zabývá návrhem a implementací rozšíření nástroje Winch pro záměnu (tzv. anonymizaci) osobních údajů. Anonymizace jako taková přináší výhodu v tom, že odstraní citlivá data a zároveň je ponechá v podobě, která je srovnatelná se skutečností. To pak umožňuje například testování s reálně vypadajícími daty, ale bez nebepečí vyzrazení osobních údajů. Toto rozšíření se pak zaměřuje na data obsažená v dokumentech ve formátu XML a JSON.

Práce nejdříve sleduje, jak anonymizace probíhá v současných částech nástroje anonymizující data v relčních databázích či v souborech tabulkových procesorů Excel. Dále probíhá analýza problému včetně obou požadovaných formátů a různých forem dotazování na data uložená v nich. Na analýzu postupně navazuje návrh, kdy je definován anonymizační model, neboli způsob určení dat k anonymizaci. Následuje návrh začlenění nových funkcí do současného řešení a poté již samotná realizace.

Výsledkem práce je pak rozšíření aplikace Winch pro souborový systém, které umožňuje nastavit anonymizační model a poté anonymizovat data v obou požadovaných formátech.

Klíčová slova anonymizace, osobní údaje, souborový systém, XML, JSON, stromový dokument

Abstract

This thesis is focused on design and implementation of the Winch extension for exchange (so-called anonymization) of personal data. Anonymization, as such, has the advantage of removing sensitive data while keeping it in a form that is comparable to reality. This then allows, for example, testing with real-looking data, but without the danger of disclosing personal information. This extension in particular is focuses on the data contained in XML and JSON documents.

The thesis first monitors how anonymization takes place in the current parts of tool anonymizing data in relational databases or in Excel spreadsheet files. Furthermore, the problem is analyzed, including both required formats and various forms of querying the data stored in them. The analysis is followed by a design of the anonymization model as the method of determining data for anonymization. Last part of the thesis is the proposal for the incorporation of new functions into the current solutions and then the implementation itself.

The result is the extension of the Winch application for the file system, which allows you to set the anonymization model and then anonymize the data in both required formats.

Keywords anonymization, personal data, file system, XML, JSON, tree document

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Projekt Winch vznikl jako nástroj pro odstranění osobních údajů (tzv. anonymizaci) v relačních databázích. Základním důvodem pro řešení anonymizace dat je splnění legislativních a regulačních opatření na ochranu dat v souladu se Zákonem 101/2000 Sb. o ochraně osobních údajů. Výhodou anonymizace například oproti šifrování je, že anonymizovaná data stále vypadají jako reálná a co je hlavní, neobsahují žádné osobní/citlivé informace. Správně provedená anonymizace neumožňuje návrat k původním datům, což je také funkce nástroje Winch.

Winch je rozdělen na dvě funkční části. První tvoří aplikace spustitelná z příkazové řádky provádějící samotné kroky anonymizace v relačních databázích. Druhou je pak doplněk (add-in) pro CASE nástroj Enterprise Architekt, kde je možné provádět nastavení anonymizace a zobrazit strukturu dat. Další funkcí nástroje Winch je schopnost provádět řezy dat, díky kterým je možné snížit objem informací či selektivně anonymizovat jen některé záznamy.

Dále vznikl nový modul aplikace Winch, a to Winch pro anonymizaci položek na souborovém systému. Jako první proběhla implementace podporující zbavení osobních údajů u souborů tabulkových procesorů formátu XLS a XLSX. Dále je v plánu podpora zpracování nestrukturovaných dokumentů jako například PDF či obrázků. Náplní této práce je pak anonymizace citlivých dat uložených ve strukturovaných dokumentech ve formátu XML a JSON.

Extensible Markup Language neboli XML je obecný značkovací jazyk, který umožňuje vytváření vlastních popisných struktur pro různé typy dat. Typicky se používá pro přenos informací po síti nebo pro popis věcného obsahu dokumentů.[11] Pro účely této práce bude však nejdůležitější možnost uložení dat jako takových.

JSON je poté zkratkou JavaScript Object Notation. Jak název napovídá, JSON vznikl jako podmnožina standardu programovacího jazyka JavaScript umožňující popisovat strukturu objektů. Vzhledem k textové charakteristice je

na něm však nezávislý. Tak jako XML se JSON nejčastěji používá pro přenos dat mezi aplikacemi.[5]

Kapitola 1

Cíl práce

Cílem práce je seznámit se s aplikací Winch a provést její rozšíření pro anonymizaci osobních údajů. Rozšíření se zaměří na umožnění odstranění citlivých dat ze strukturovaných dokumentů typu XML a JSON. Důležitým krokem k dosažení cíle bude také navržení vhodného anonymizačního modelu, neboli jak určit data k anonymizaci.

Aplikace dále vizualizuje strukturu souboru a poskytne uživatelské rozhraní, které umožní nadefinování parametrů anonymizačního modelu. Cílem je robustní a kvalitní řešení, proto ho bude také nutné řádně otestovat na rozličných typech dat.

Kapitola 2

Analýza a návrh

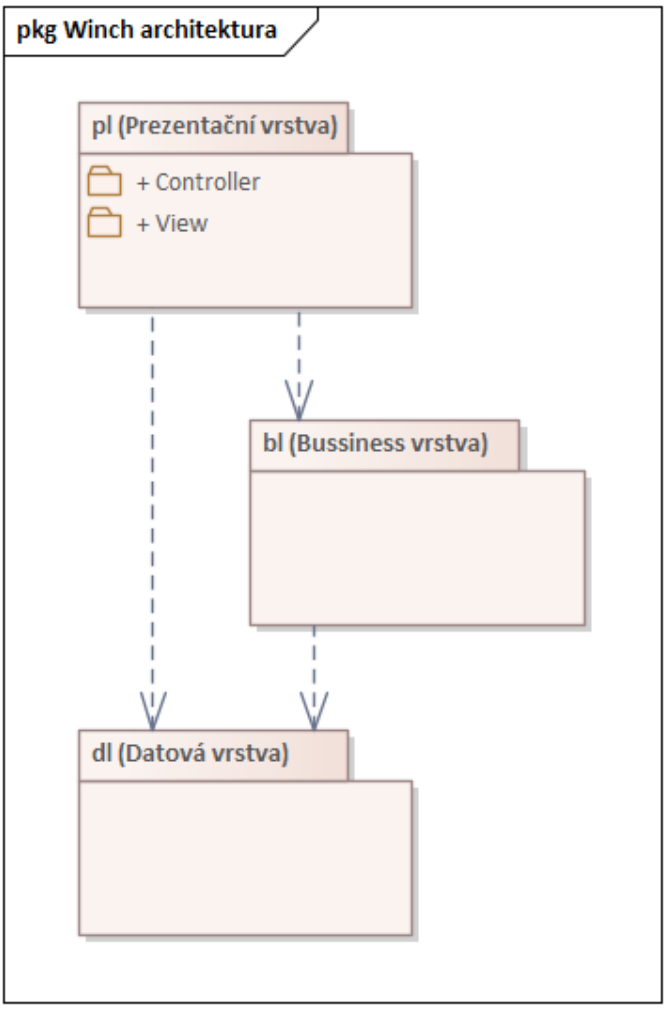
2.1 Současný stav nástroje Winch

Jak již byl zmíněno v úvodu, nástroj Winch pro anonymizaci na relačních databázích je rozdělen na dvě části. První částí je aplikace provádějící anonymizaci nazývána Actor postavená na architektuře virtuálního serveru Java, je spustitelná z příkazové řádky a napsána v jazyce Groovy (objektově orientovaný jazyk pro platformu Java). Druhou vizuální část pak tvoří add-in do nástroje Enterprise Architect napsaný v jazyce C# na architektuře .NET Framework. Tato práce se pak zaměřuje na rozšíření novější varianty aplikace Winch, a to pro anonymizaci na souborovém systému.

Tato odnož „Winche“ je analogicky s Actorem napsána v jazyce Groovy. Prvky uživatelského rozhraní využívají komponenty knihovny Java Swing. Aplikace je postavena na relaxované třívrstvé architektuře – viz obrázek 2.1, která je rozdělena na

- Prezentační vrstvu zastřešující vizuální stránku aplikace, která je dále rozdělena na
  - kontrolery, jež zpracovávají vstupy uživatele a
  - zobrazované prvky (tzv. view),
- Business vrstvu, kde je situována aplikační logika programu a
- Datovou vrstvu sloužící pro manipulaci se soubory.

Architekturu lze označit jako relaxovanou vzhledem k tomu, že prezentační vrstva přistupuje i přímo k datové. V době seznamování se s nástrojem je částečně implementovaná podpora anonymizace souborů tabulkových procesorů typu Excel ve formátech xlsx a xls. V uživatelském rozhraní není zatím realizována možnost výběru způsobu provedení anonymizace, tedy výběr tzv.



Obrázek 2.1: Winch architektura

patternu. Anonymizace je tím momentálně omezena pouze na lokální soubory. Anonymizačních funkcí je k dispozici pouze několik, nicméně jsou rychle doplňovány studenty v rámci předmětu Softwarový týmový projekt.

2.2 Aplikační požadavky

V této kapitole budou vyjmenovány požadavky na implementaci. Zaměřeno je jak na ty popisující konkrétní funkce aplikace, tak i obecné požadavky na ní kladené.

2.2.1 Funkční požadavky

FN1 Anononymizace dat v souborech formátu XML.

Aplikace umožní uživateli vybrat XML soubory, zvolit data určená k anonymizaci a nastavit model, podle kterého poté bude anonymizace vykonána. Anonymizace bude provedena buď přímo na upravovaném souboru, či se vytvoří kopie dotčeného souboru a ten bude následně pozměněn. Data mohou být

- v jednom souboru například v rámci kolekcí nebo
- rozprostřena do více (typicky) malých souborů.

FN2 Anononymizace dat v souborech formátu JSON.

Analogicky s požadavkem FN1 bude možné provést anonymizaci dat v souborech typu JSON.

FN3 Vizualizace obsahu anonymizovaného souboru.

Aplikace vizualizuje oba požadované formáty vhodným způsobem tak, aby uživatel získal přehled o obsahu a struktuře daného souboru.

FN4 Výběr a nastavení modelu anonymizace.

Aplikace poskytne grafické rozhraní, které umožní výběr dat a volbu vhodné anonymizační třídy pro jednotlivé datové prvky v souborech určených k anonymizaci.

FN5 Načtení struktury anonymizovaného dokumentu XML nebo JSON ze schéma definujícího souboru.

Řešení poskytne podporu pro formáty DTD a XSD definující struktur XML. U souborů typu JSON budou umožněny definice pomocí formátu JSON Schema. Aplikace také umožní detekovat strukturu souboru přímo z jeho obsahu a to v případech, kdy schéma není k dispozici.

FN6 Filtrace pomocí dotazovacího jazyka.

Data v anonymizovaných souborech bude možné prohledávat (filtrovat) pomocí vhodných dotazovacích jazyků. Aplikace napomůže vytvoření takového dotazu (např. pomocí výběru elementu v celkovém přehledu) a zároveň vhodně vizualizuje data, která zadanému příkazu vyhovují.

2.2.2 Obecné (nefunkční) požadavky

NF1 Integrace do aplikace implementované v jazyce Groovy s využitím knihovny Swing pro uživatelské rozhraní.

NF2 Implementace rozšíření kompatibilní s 32 bitovou verzí Java 8.

2.3 Definice typického uživatele

Typický uživatel bude datový analytik. Lze předpokládat znalost struktur souborů a do jisté míry i dotazovacích jazyků. Aplikace by však měla umožnit nastavení anonymizace i méně technicky znalému uživateli.

2.4 Doménový model

První část doménového modelu (zobrazená na snímku 2.2) zachycuje dva typy souboru, kterými se zabývá tato práce.

XML soubor Tento typ formátu je tvořen dvěma prvky, a to element a text. Dokument tohoto druhu má vždy jeden kořenový element, který může obsahovat kombinaci dalších elementů a textů. Totéž platí i o těchto „potomkových“ elementech. Důležité je taktéž pořadí prvků. Krom toho je také povolen prázdný obsah elementu. Proces tvoření elementu je možné vidět na obrázku 2.3. V doméně anonymizace je pak vhodné uvažovat o anonymizaci textových prvků a hodnot atributů.

XML elementy se zapisují pomocí
otevíracích a uzavíracích značek, které využívají znaků `<` a `>`, tzv. menšítko a většítko. Celá otevírací značka vypadá následovně `<nazev_elementu>`. Podobně pak uzavírací označení elementu `</nazev_elementu>`. Obsah prvku je poté vložen mezi obě značky. Pro element s prázdným obsahem lze použít zkrácenou verzi s pouze jedním označovačem v notaci `<prazdny_element/>`. Uvnitř otevíracího prvku (v případě zkráceného zápisu prázdného elementu tedy i uzavíracího) lze dále definovat atributy zápisem `nazev_atributu="hodnota"`. Textové prvky se nijak neuvozují a vkládají se do prvků jako běžný text.[11] Celý element pak může vypadat následovně.

```
<nazev_elementu nazev_atributu="hodnota">
    Textový prvek
    <prazdny_element/>
</nazev_elementu>
```

JSON soubor Základním prvkem JSON dokumentů je objekt. Každý JSON soubor má jeden kořenový objekt nebo pole. Uvnitř něho se pak může nacházet množina párů unikátního názvu a hodnoty. Přípustné jednoduché hodnoty jsou

- řetězce,
- čísla (s i bez desetinné čárky),
- boolovské hodnoty (true nebo false) a
- null.

Tyto typy se hodí pro anonymizaci a bude možné jim přiřadit anonymizační třídu. Dále JSON umožňuje jako hodnoty použít jiné objekty a také pole. V poli jsou povoleny všechny předchozí hodnoty včetně dalších polí.

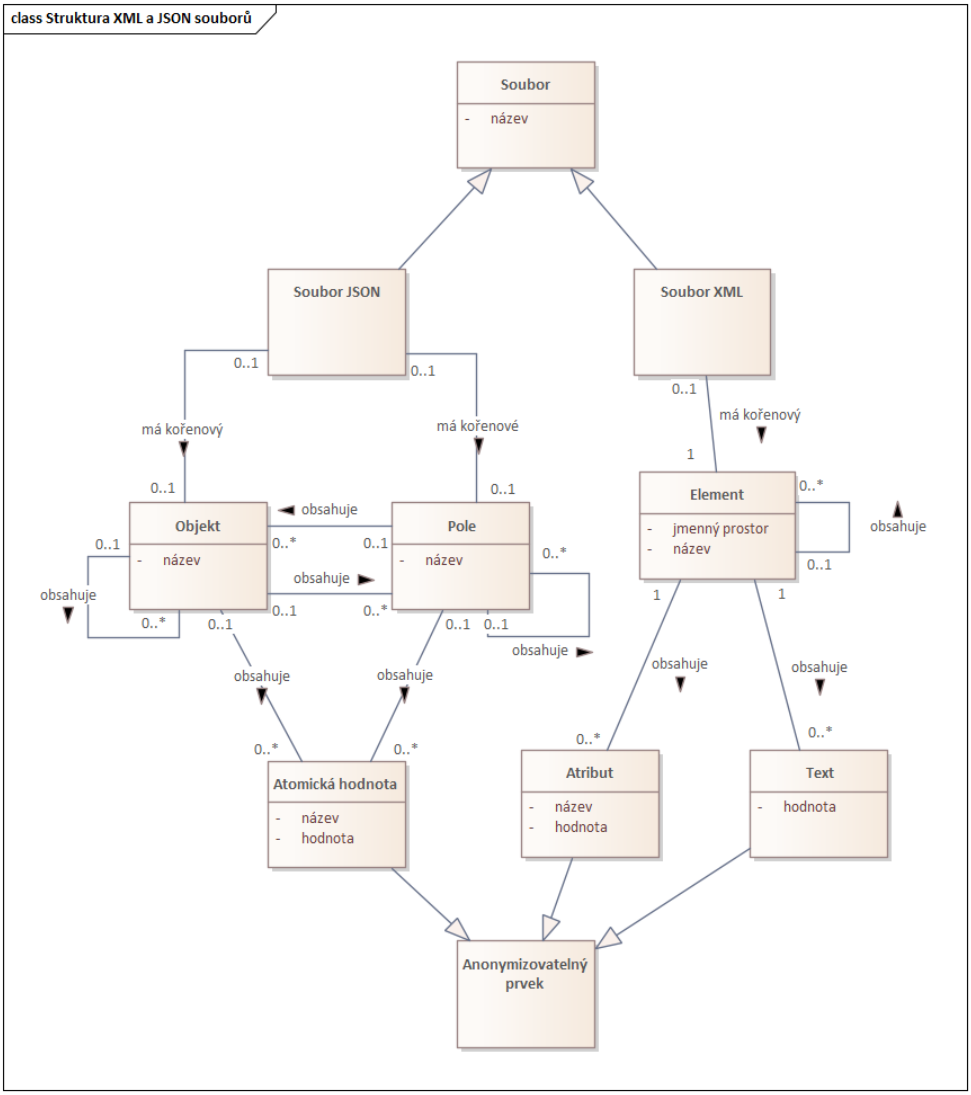
Pro zapsání JSON objektů je využito složených závorek, tedy znaků `{` a `}`. Názvy hodnot jsou ohraničeny symbolem anglických dvojitých uvozovek `"` a od hodnoty odděleny dvojtečkou. Tvorbu JSON objektů znázorňuje obrázek 2.4 Jednotlivé dvojice unikátních názvů a hodnot jsou rozděleny čárkou (na rozdíl od programovacího jazyka JavaScript v notaci JSON nemůže být za poslední dvojicí čárka). Řetězce jsou shodně s názvy ohraničeny symbolem anglických uvozovek, čísla se zapisují běžným způsobem. U desetinných čísel je, tak jak je v anglosaských zemích zvykem, použita tečka. Hodnoty true, false a null jsou „speciální“ a píší se přímo bez uvození. Pro zapsání pole je využito hranatých závorek (znaky `[` a `]`), kdy jednotlivé prvky jsou analogicky s objekty odděleny čárkou. Vše je nejlépe vidět na ukázce.[5]

```
{
    "retezec": "text",
    "cislo": 3,
    "boolean": true,
    "pole": [1, "text", false, [-1.0, 0, 1.0]],
    "objekt": {
        "cislo2": 5
    }
}
```

Oba formáty mají stromovou strukturu. U XML je tvořena elementy, které je možné vnořovat do sebe. U souborů JSON tvoření stromu umožňují, krom základních objektů, také struktury pole.

Druhá část doménového modelu na obrázku 2.5 popisuje aktuální abstrakci programu Winch společně se začleněním rozšíření pro anonymizaci souborů formátu XML a JSON. Současné prvky programu, jako je aplikace (souhrnné označení pro nastavení celé anonymizace skupiny souborů), kontext (umožňuje různá nastavení zdrojů a cílů dat pro různá prostředí, např. produkční či vývojové), adresář, soubor anebo vzor, vycházejí z bakalářské práce pana Orta[6]. V současné implementaci je pro aplikaci možno nastavit pouze jeden kontext s dvěma připojeními, to je zohledněno v doménovém modelu 2.5 této práce. Vzor neboli pattern má pak aplikace jako celek přiřazen přes kontext, ale každý jednotlivý soubor může mít použitý vlastní pattern.

Nad samotnými soubory XML a JSON je pak dále zachycena jistá abstrakce v podobě entity „Dataset“. Jedná se o vyjádření podmnožiny dat, kterou si uživatel bude nad dokumentem vytvářet. Vymezení dat bude provedeno příkazem vhodného dotazovacího jazyka. V rámci vytvořeného datasetu bude

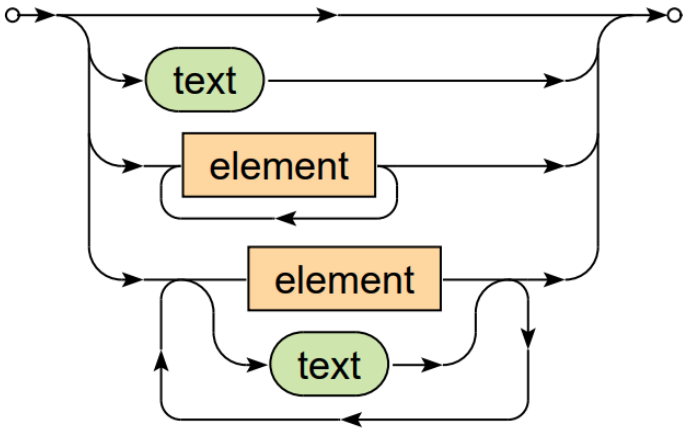


Obrázek 2.2: Struktura XML a JSON

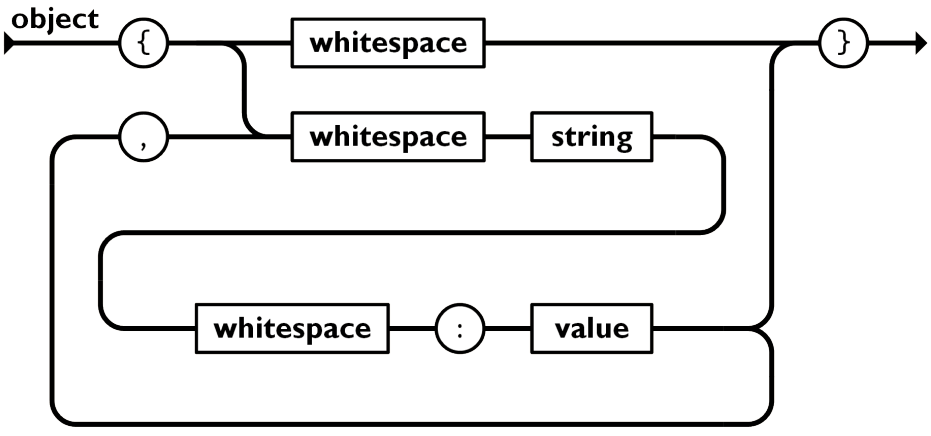
dále analytik určovat jednotlivé anonymizovatelné prvky a těm přiřazovat anonymizační třídu.

2.5 Formáty pro definici dat v XML a JSON souborech

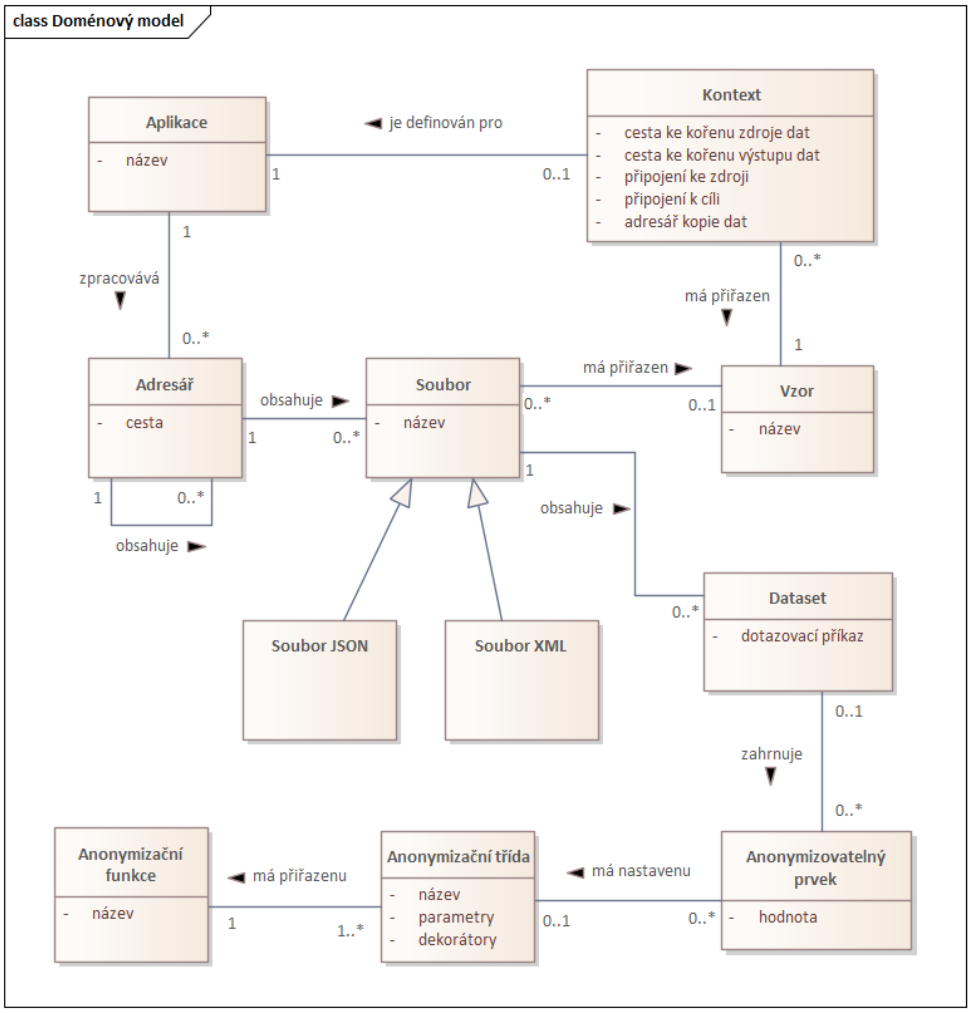
Pro oba typy dokumentů existuje několik různých typů formátů pro definici dat. Obecně popisují strukturu uložených informací a jejich datové formáty.



Obrázek 2.3: Tvoření XML elementů[4]



Obrázek 2.4: Tvoření JSON objektů[5]



Obrázek 2.5: Doménový model[6]

2.5.1 XML

Pro definici schématu dat se u XML dokumentů nejčastěji využívají formáty DTD (Document Type Definition) a XSD (XML Schema Definition).

2.5.1.1 DTD

Starší jazyk definice struktury pro XML nebo SGML (Standard Generalized Markup Language) dokumentu, používá notaci lehce odlišnou od XML. Umožňuje pojmenovat elementy (návěstí `!ELEMENT`) a definovat jejich obsah, tedy zda má obsahovat text (značí se `#PCDATA`), jiný element (udá se jméno elementu, které musí být následně definováno) nebo kombinace obojího,

kdy možnosti jsou odděleny pomocí | (vertikální čára). Možné je definovat i jakýkoliv obsah klíčovým slovem `ANY`. Notace dále umožňuje stanovit množství elementů, možnosti jsou

- libovolné množství (*),
- žádný nebo jeden element (?),
- alespoň jeden element (+) nebo
- přesně jeden (neznačí se, základní stav).

Deklarovat je možné i atributy XML elementů. Jejich definice je uvedena pomocí `!ATTLIST` a postupně obsahuje název elementu, pro který bude atribut určen, název atributu, typ atributu a jeho hodnotu. Typ může být například CDATA (představuje prostý text), unikátní identifikátor nebo seznam identifikátorů jiných elementů. Kompletní soupis je k dispozici na stránce W3 Schools[12]. Příklad DTD souboru (návěští `!DOCTYPE` definuje kořenový element dokumentu)[13]:

```
<!DOCTYPE TVSCHEDULE [

<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE,HOLIDAY*))>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>

<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
]>
```

2.5.1.2 XSD

XSD na rozdíl od DTD využívá přímo notaci XML. Každý XSD dokument je uveden kořenovým elementem `schema`, ve kterém je možno nadefinovat jeden nebo více jmenných prostorů v rámci atributů elementu. Následující ukázka `xmlns:xs="http://www.w3.org/2001/XMLSchema"` indikuje, že elementy použité ve schématu pochází z odkazované stránky a zároveň mají mít prefix `xs:`. Notace dále poskytuje možnosti k definici konkrétních elementů. Základními dvěma typy jsou jednoduchý element (Simple Element) a složený (Complex Element).

Pro jednoduché elementy (označeny `element`) platí, že mohou obsahovat pouze text a žádné atributy. Je však možné nadefinovat, o jaký typ textového obsahu se jedná. XSD poskytuje několik nejčastěji používaných typů. Příkladem buď

xs:string pro definici čistě textového obsahu,

xs:integer reprezentující celočíselné hodnoty,

xs:decimal čísla s desetinou čárkou nebo

xs:date pro datum ve formátu YYYY-MM-DD.

Podobným způsobem lze určit i atributy elementů. Ty jsou uváděny klíčovým slovem `attribute`. Pro jednoduché elementy a atributy je možné nastavit řadu omezení (návěstí `restriction`), ku příkladu výčet povolených hodnot, vzor (podobný regulárnímu výrazu) či maximální délka textu. Kompletní seznam možných omezení je k nalezení na W3 Schools[14].

Komplexní elementy jsou popsány uvnitř elementu `complexType`. Mohou být jedním ze 4 typů:

- Prázdný element (typicky obsahuje pouze definici atributů).
- Element obsahující pouze další elementy. Vnitřní elementy mohou obecně být jednoduché nebo opět komplexní a musí být uvnitř jednoho z indikátorů
  - all (všechny definované elementy musí být přítomny v libovolném pořadí),
  - choice (jeden z popsaných je nutno vybrat) nebo
  - sequence (všechny definované elementy ve specifikovaném pořadí).
- Element mající jen textový obsah. Takové elementy musí obsahovat definici `xs:simpleContent`, uvnitř které ještě musí být uvedeno buď omezení nebo rozšíření (element `extension`).
- Smíšený element, který obsahuje kombinaci elementů a textů. Popsán je stejně jako elementy s dalšími uvnitř, ale u elementu `complexType` musí být uveden atribut `mixed="true"`.

Pro všechny typy komplexních elementů platí, že mohou obsahovat atributy. Příklad XSD souboru:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="osoba">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="jmeno" type="xs:string"/>
                <xs:element name="prijmeni" type="xs:string"/>
            </xs:sequence>
        <xs:attribute name="uid" type="xs:string"/>
```

```
        </xs:complexType>
    </xs:element>
</xs:schema>
```

2.5.2 JSON

Pro dokumenty JSON je v této práci uvažována definice pomocí JSON Schema. Tato notace využívá přímo strukturu souborů JSON, tedy objekty a jejich parametry. V kořenovém objektu JSON Schema jsou definována dvě důležitá klíčová slova. Prvním je `schema` stanovující použitý standard schématu. Druhým je poté `id`, který udává URI pro vytvářené schéma. Dále se na stejné úrovni definují přímo prvky validovaného souboru. JSON Schema poskytuje tři možnosti.

- `type` – udává typ hodnoty. Pro JSON to jsou známé typy null, boolean, object, array, number nebo string. JSON Schema přidává možnost validovat i to, zda se jedné o celé číslo. Tato možnost odpovídá hodnotě integer. Je možné udat více těchto hodnot pomocí pole.
- `enum` – výčet požadovaných hodnot v neprázdném poli.
- `const` – konstantní hodnota parametru.

Pokud byl definován typ object, je možné stále na stejné úrovni JSON dokumentu stanovit vlastnosti tohoto objektu. K tomu je určen nový objekt uvozený klíčovým slovem `properties`, uvnitř kterého jsou další dvojice klíč a objekt, který opět umožňuje definovat omezení na klíčem pojmenovanou vlastnost. Pro vlastnosti objektů je také možné určit povinnost pomocí klíčového slova `required` a pole názvů povinných položek. Podobně jako pro objekty lze nastavit restrikce i na prvky pole v objektu uvedeném klíčem `items`. Zde však definuje omezení přímo na všechny prvky pole.

Omezení obecně závisí na specifikovaném typu. Pro číselné hodnoty je to například minimum či maximum, pro řetězce poté délka textu nebo regulární výraz dle ECMA-262[15]. U objektů to je ku příkladu již zmíněná povinnost vlastností či jejich maximální počet. Následuje ukázka[16] JSON Schema definice.

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "https://example.com/product.schema.json",
    "title": "Product",
    "description": "A product from Acme's catalog",
    "type": "object",
    "properties": {
        "productId": {
            "description": "The unique identifier for a product",
```

```
            "type": "integer"
        },
        "tags": {
            "description": "Tags for the product",
            "type": "array",
            "items": {
                "type": "string"
            },
            "uniqueItems": true
        },
        "dimensions": {
            "type": "object",
            "properties": {
                "length": {
                    "type": "number"
                },
                "width": {
                    "type": "number"
                }
            },
            "required": [ "length", "width" ]
        }
    },
    "required": [ "productId" ]
}
```

2.6 Analýza podobného řešení

Vzhledem k tomu, že podobný volně dostupný software se stejným zaměřením (tedy anonymizace XML nebo JSON souborů) nebyl nalezen, zaměřuje se práce alespoň na způsoby požadované vizualizace XML a JSON dokumentů. Pro oba formáty je typická stromová struktura, což se odrazuje i v jejich typickém zobrazování. V naprosté většině případů je strom rozvíjen horizontálně zleva doprava. Vizuálně velmi zdařilý je prohlížeč XML na stránce xmlviewer.org, jehož ukázka je na obrázku 2.6. Další ukázkou 2.7 je prohlížeč XML zahrnutý v Microsoft Visual Studiu. Na rozdíl xmlviewer.org Visual Studio nezakrývá přítomnost XML tagů a atributy elementů zobrazuje přímo v nich. Podobně tomu je i na vizualizéru JSON ze stránky jsonviewer.stack.hu, kde jsou viditelné2.8 prvky JSON objektů.

Jediný konceptuálně odlišný způsob zobrazení stromové struktury lze vidět2.9 v aplikaci BaseX. Tento princip vizualizace se snaží prvky potomků zanořovat do obdélníkového vyjádření rodiče. Už s malým množstvím dat se však tento způsob zobrazení stává velmi nepřehledný. I proto se při imple-

Obrázek 2.6: Prohlížeč XML xmlviewer.org[7]

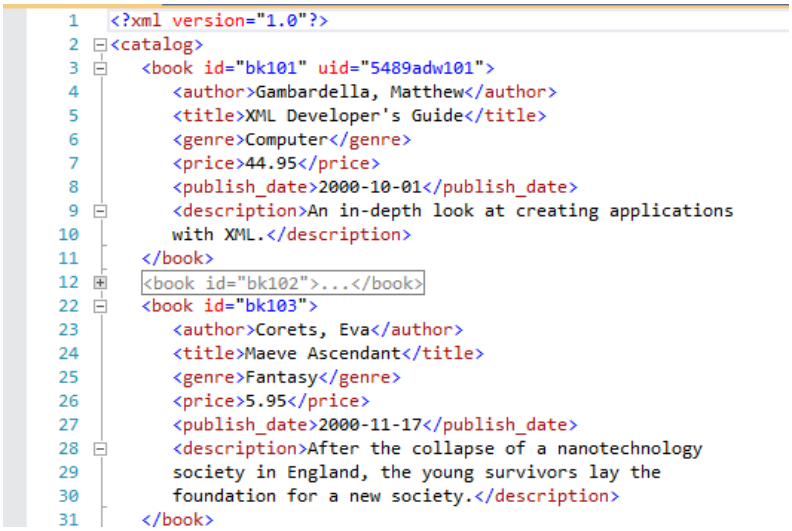
mentaci přikloním ke klasickému stromovému vyobrazení a jednotlivé prvky struktur spíše zanedbám po vzoru xmlviewer.org2.6.

2.7 Výběr technologií

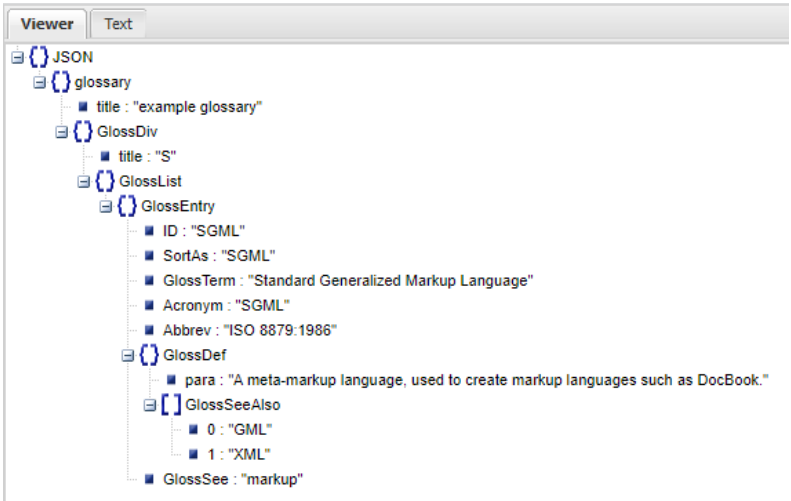
Tato kapitola je zaměřena na porovnání různých technologií či knihoven, které mohou zastat požadované funkcionality.

2.7.1 Zobrazení XML/JSON

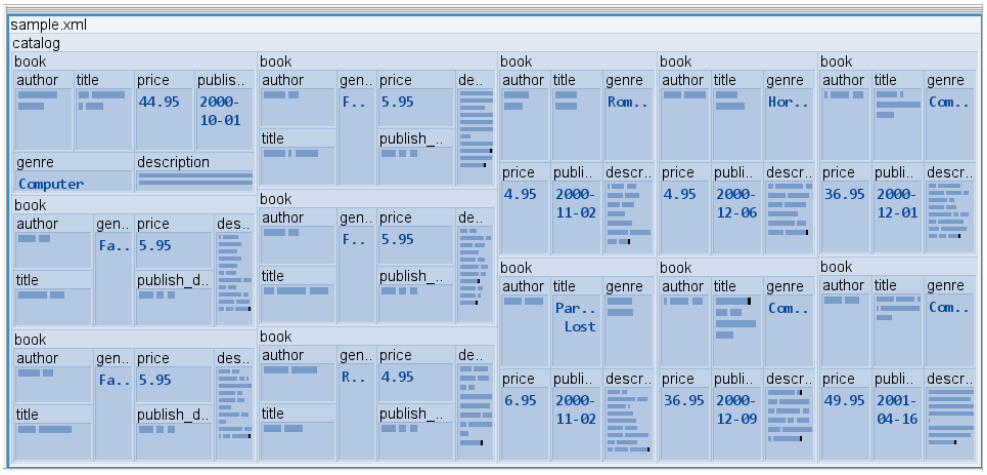
Vzhledem k současné implementaci uživatelského rozhraní aplikace v knihovně Swing se analýza zaměřuje především na řešení vycházející z ní. Vybraná technologie by měla splňovat požadavky stanovené v předchozí kapitole 2.6. Zároveň bude žádoucí možnost úpravy způsobu zobrazení jednotlivých prvků stromu dokumentu (například doplnění o jinou komponentu uživatelského rozhraní). Přehled pokrývá jak již kompletní hotové vizualizace, tak i pouze vhodné komponenty.



Obrázek 2.7: Prohlížeč XML Microsoft Visual Studio[8]



Obrázek 2.8: Prohlížeč JSON jsonviewer.stack.hu[9]



Obrázek 2.9: Zobrazení XML v Base-X[10]

2.7.1.1 XMLEditorKit[1]

Tato implementace vychází z komponenty JEditorPane, jde tedy pouze o obohacené textové pole. XMLEditorKit rozšiřuje StyledEditorKit, díky tomu jsou zvýrazněny prvky XML dokumentu a je taktéž možné otevírat a zavírat jednotlivé elementy. Implementace uvnitř textového pole samozřejmě umožňuje také editaci dokumentu. Nicméně celé řešení je poměrně komplexní a bylo by složité ho rozšířit či upravit. Navíc je silně zaměřeno pouze na XML. Ukázku je možno vidět na obrázku 2.10.

2.7.1.2 JsonBrowse[2]

Podobně jako XMLEditorKit jde o řešení zaměřené pouze na jeden typ dokumentů, v tomto případě je to JSON. JsonBrowse však vychází z komponenty JTree, tedy prvku, který je už v základu určen pro stromové struktury. Řešení však tvoří spíše jeden celek a není vhodné pro použití jako upravitelná komponenta. Aplikace je zachycena na snímku 2.11.

2.7.1.3 JTree

Jak již bylo zmíněno, tato Swing komponenta je přímo určená k zobrazení stromových struktur. Nelistové uzly stromu umožňuje otvírat a zavírat. Pro tyto operace lze nastavit posluchač (listener) a reagovat na ně. Navíc je pro jednotlivé buňky stromu možné vytvořit vlastní způsob vykreslení. Díky tomu lze do požadovaných uzlů přidat jiné komponenty. Zobrazení stromu se nijak vizuálně neliší od JsonBrowse.

Obrázek 2.10: Komponenta XmlEditorKit

2.7.1.4 Outline[3]

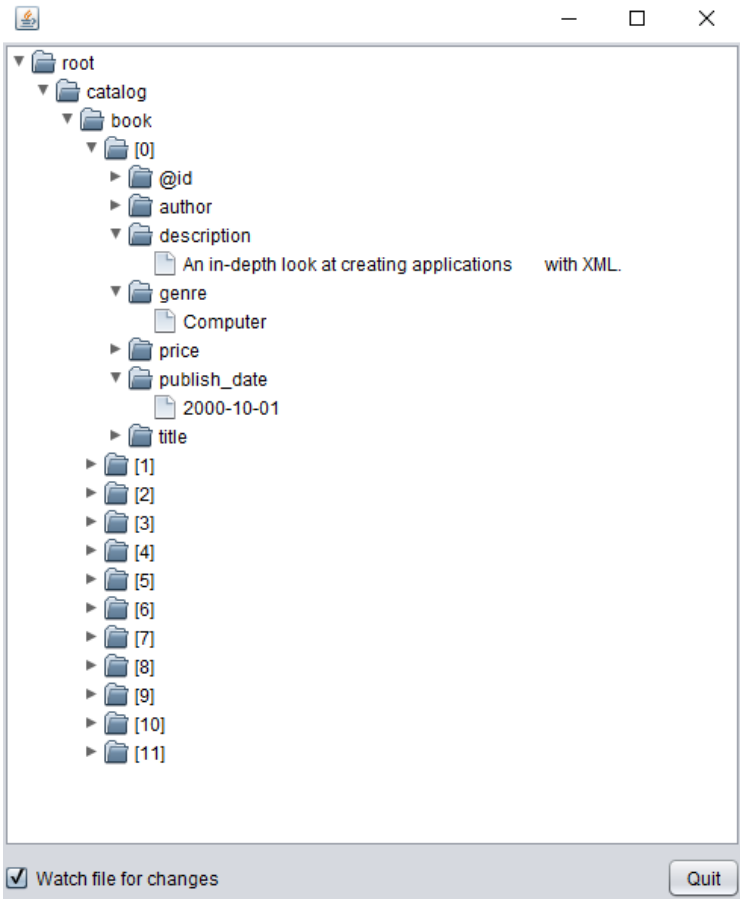
Tato komponenta z projektu NetBeans[17] rozšiřuje standardní tabulkové zobrazení JTable o vizualizaci stromu. Ten je reflektován v prvním sloupci tabulky. V ostatních sloupcích lze analogicky s JTree přidávat jiné komponenty nebo je ponechat jako prostý text. Ukázka Outline je na obrázku 2.12.

2.7.1.5 Zhodnocení

Nejvhodnější komponentou pro zobrazení XML a JSON v definovaném vizuálním provedení se jeví JTree. Je od základu navržena na zobrazování stromových struktur, navíc poskytuje možnosti individuálních úprav zobrazení. Outline může být vhodná v případech, kdy bude pro jednotlivé stromové prvky potřeba zobrazit další informace.

2.7.2 Dotazovací jazyky na XML a JSON

Hlavním účelem dotazovacího jazyka v aplikaci Winch pro anonymizaci XML a JSON dokumentů je výběr dat ze zpracovávaných souborů. Jazyk by tedy měl určit jeden nebo více prvků ze stromové struktury dokumentu, přičemž by měl umožnit filtrování na základě omezení kladených na obsah dat. Navigace ve struktuře stromu by měla být možná jak od kořene (absolutní cesta), tak i z některého vnitřního uzlu (relativní cesta). Jazyk bude taktéž k dispozici uživateli, je tedy vhodné uvažovat jazyk, který datoví analytici běžně používají.



Obrázek 2.11: Aplikace JsonBrowse

2.7.2.1 XPath

XPath je jazyk určený na navigaci v XML dokumentech. Umožňuje označení dat pomocí absolutní i relativní cesty. Navigovat lze ve všech možných směrech uvnitř stromu, a to jak nahoru nebo dolu (předci nebo potomci), tak i horizontálně mezi sourozenci a jejich potomky. Samozřejmostí je i přístup k atributům. Dále XPath poskytuje dobré možnosti filtrace například na základě textových hodnot elementů či atributů anebo pozice ve stromu dokumentu. Je také možné sdružovat (sjednocovat) výsledky několika dotazů nebo provádět průniky dat.[18]

XPath neposkytuje nástroje pro složitější podmíněné dotazování a další manipulaci s vybranými daty. Ve verzi XPath 3.1 je také umožněno zpracování JSON dokumentů. Standard přidává funkce pro práci s poli a objekty (označované jako map). Nicméně tato verze XPath poskytuje pouze omezené možnosti pro JSON. Například průchod stromem je možný pouze směrem

File System	Date	Size
winch-disl-connector	8.4.2022	4096
.git	28.4.2022	4096
.gitignore	29.1.2022	138
.gradle	29.1.2022	0
.idea	29.4.2022	4096
build.gradle	8.4.2022	7074
config	29.1.2022	0
configuration.json	29.1.2022	710
disl-winch-connector	29.1.2022	0
.gitignore	29.1.2022	36
.groovy	29.1.2022	0
build	9.2.2022	4096
lib	29.1.2022	0
Readme.md	29.1.2022	2416
src	29.1.2022	0
disl-winch-db2	29.1.2022	0
disl-winch-filesystem	29.4.2022	4096
disl-winch-generator	29.1.2022	0

Obrázek 2.12: Komponenta Outline

dolu. Možná i proto se použití XPath pro JSON data nezdá být běžnou praxí[19].

2.7.2.2 XQuery

XQuery je funkcionální dotazovací jazyk na XML data. Lze říci, že se jedná o ekvivalent jazyka SQL pro XML dokumenty. Na identifikaci dat využívá notaci XPath. Následně poskytuje další příkazy pro práci s daty. Tento systém je označován jako FLWOR, kde jednotlivá písmena znamenají:

for Výběr procházených uzlů.

let Nastavení potřebných proměnných.

where Filtrace uzlů.

order by Seřazení uzlů.

return Úprava výstupu.

Dále XQuery poskytuje podmíněné dotazy (if-then-else) nebo kvantifikované výrazy (some|every-satisfies). Vše je možné provádět uvnitř tzv. konstruktorů, s kterými se vytvářejí nové elementy. XQuery je tedy výborným nástrojem pro transformace, navigaci v datech však příliš neobohacuje. Co se týče JSON dokumentů má XQuery k dispozici stejné funkce jako XPath.[18]

2.7.2.3 JSONPath

Jak název napovídá, jde, do jisté míry, o ekvivalent XPath pro dokumenty JSON. Stejně jako XPath, JSONPath umožňuje částečně navigaci uvnitř stromové struktury souboru včetně filtrování. Samozřejmostí jsou pak nástroje pro práci s poli. Bohužel však poskytuje pouze dotazy s absolutní cestou. S tím je spojena i nemožnost procházení stromem směrem nahoru ke kořeni či do stran. Dále uživateli nedává prostředky pro vykonání množinových operací. Pro XML dokumenty využít nelze.[20]

2.7.2.4 GPath

Koncepčně podobný jazyk pro navigaci ve stromové struktuře jako jazyky XPath a JSONPath specifický pro programovací jazyk Groovy. Výhodou GPath je jeho univerzálnost, kdy umožňuje použití jak pro XML tak i JSON dokumenty. Poskytuje základní navigaci ve stromu směrem dolů a v rámci sourozenců. Sám o sobě nedává možnost filtrací.[21] Ale vzhledem k implementaci nástroje v jazyce Groovy, lze uvažovat řešení, kde bude filtrování možno definovat pomocí Groovy skriptu. Nevýhodou jazyka je však jeho poměrně malá rozšířenost[22]. To by znamenalo nutnost poskytnout uživateli komplexní uživatelské rozhraní pro sestavení takových příkazů.

2.7.2.5 Zhodnocení

Shrnutí výhod a nevýhod je možné vidět v tabulce 2.1. Pro implementaci byly vybrány jazyky XPath pro XML a JSONPath pro JSON, a to především proto, že představují standard v cílené funkčnosti. Dobře pokrývají všechny požadavky kladené na jazyk a jsou běžně používané, což by mělo uživateli vyhovovat.

2.7.3 Načítání (parsování) dokumentů XML/JSON

Pro načítání dokumentů XML nebo JSON existují tři základní přístupy. Liší se především paměťovou náročností a každý je vhodný v různých situacích.

2.7.3.1 Document Object Model

Princip Document Object Model (DOM) je založený na načtení celého dokumentu a sestavení jeho stromu do paměti. To přináší velké nároky na paměťovou složitost tohoto přístupu. Označení DOM je obvyklé především ve spojení s dokumenty XML. U struktur JSON může být pojmenováno jako Object Model a často se jedná o základní (nijak neoznačovaný) přístup.

Pro dokumenty XML je u principu DOM přímo v Javě zabudovaná reprezentace v balíčku `org.w3c`. Načítání pak provádí třídy z knihovny `javax.xml.parsers`. Jazyk Groovy doplňuje tyto balíčky podporou GPath.

Tabulka 2.1: Výhody a nevýhody dotazovacích jazyků

	Výhody	Nevýhody
XPath	Zavedený jazyk Navigace ve stromu ve všech směrech Možné použití pro oba formáty Množinové operace	Pro JSON není běžně používaný
XQuery	Všechny výhody XPath Rozšíření možností filtrování oproti XPath	Složité dohledání původních dat při úpravě výstupu dotazu pomocí return
JSONPath	Zavedený pro JSON	Chybí množinové operace Navigace pouze směrem dolu ve stromu
GPath	Použitelný pro oba formáty	Málo rozšířený

Groovy taktéž nabízí třídu `groovy.xml.XmlParser`. Ta nenačítá celý DOM jako takový, ale sama si udržuje podobnou strukturu dokumentu.

Jak již bylo zmíněno u dokumentů JSON se jedná o standardní přístup, existuje proto mnoho implementací. V modulu Winch filesystem je již použita knihovna `org.json`[23]. Dalšími zástupci jsou například [24] od společnosti Google, Jackson[25] nebo balíček `javax.json` z JEE 7. Groovy taktéž podporuje zpracování JSON dokumentů, a to zásluhou třídy `groovy.json.JsonSlurper`. Ta kupříkladu převádí JSON objekty na `java.util.LinkedHashMap` nebo JSON pole na `java.util.ArrayList`.

DOM je tedy vhodný především při opakovaném dotazování na data a rozličné manipulace s nimi. Navíc pro programové využití XPath a JsonPath představuje přístup přes kompletní DOM nutnou podmínku. Popsané implementace poskytované jazykem Groovy toto bohužel nesplňují – zaměřují se na použití GPath. Nevýhodou přístupu DOM je však vysoká paměťová náročnost, která u velmi velkých souborů může vést i na nemožnost načtení celého dokumentu do paměti programu. Proto byla provedena následující analýza, ve které byla prozkoumána paměťová náročnost jednotlivých analyzátorů.

Vytvořeny byly dva dokumenty XML. První (označen A) obsahující kolekci 5 milionů záznamů s dvěma podelementy (s texty v délce přibližně 8 a 35 znaků) a jedním číselným atributem vyjadřujícím identifikační číslo. Velikost tohoto souboru se pohybovala okolo 700 MB. Druhý (označen B) dokument poté obsahoval 4 miliony záznamů s 6 subelementy mající texty o průměrné délce deseti znaků a atributem definujícím id. Celková velikost tohoto souboru by cca 1,1 GB. Test prozkoumal dva analyzátory. V Javě zakomponovaný balíček `org.w3c` ve spolupráci s `javax.xml.parsers` a Groovy analyzátor

Tabulka 2.2: XML analyzátory (parsery)

Velikost v operační paměti (násobnost zvětšení)	A (∼700 MB)	B (∼1,1 GB)
`org.w3c` + `javax.xml.parsers`	4,2 GB (6,5×)	6,8 GB (6,2×)
`groovy.xml.XmlParser`	6,4 GB (9,1×)	9,6 GB (8,7×)

Tabulka 2.3: JSON analyzátory (parsery)

Velikost v operační paměti (násobnost zvětšení)	A (∼430 MB)	B (∼690 MB)
org.json	5,7 GB (13,3×)	6,8 GB (9,9×)
Jackson	3,9 GB (9,1×)	4,5 GB (6,5×)
GSON	4,8 GB (11,2×)	7,4 GB (10,7×)

`groovy.xml.XmlParser`. Tabulka 2.2 zobrazuje velikost po načtení souboru do operační paměti.

Analogicky byly vytvořeny dokumenty JSON s tím rozdílem, že atribut identifikátoru byl převeden na prvek klíč-hodnota. Velikost souborů se vyšplhala přibližně na 430 MB (soubor A), respektive 690 MB (soubor B). Otestovány byly načítače knihoven org.json, Jackson a GSON. Všechny tyto knihovny byly nastaveny jako poskytovatelé JSONu pro implementaci JsonPath[26] v Javě. Výsledky jsou zaznamenány v tabulce 2.3.

Dle naměřených dat je z testovaných knihoven v hledisku paměťové efektivity nejlepší org.w3c u XML dokumentů. Pro soubory JSON se jako nejefektivnější projevil Jackson.

2.7.3.2 Simple API for XML

Jak název napovídá, Simple API for XML neboli SAX (někdy také označováno jako push parser), je zaměřeno na dokumenty XML. Princip je založen na vyvolávání událostí při čtení dokumentu. Typické události jsou začátek nebo konec dokumentu, začátek nebo konec elementu či znaky (text). Na programu přijímajícím tyto události je pak jejich zpracování. Je tedy možné nepotřebné prvky dokumentu vyfiltrovat a uložit či zpracovat jen ty důležité.

Pro XML dokumenty je opět v Javě zavedena podpora pro tento přístup. Dosaženo je toho pomocí dědění třídy `org.xml.sax.helpers.DefaultHandler`, kde je poté nutné přetížit metody požadovaných událostí. Instance této třídy je pak předána analyzátoru (par-

ser). Groovy poskytuje již zmíněnou třídu `groovy.xml.XmlParser`, ta však SAX využívá pouze interně a vytvoří svoji reprezentaci DOM. Podobně funguje i třída `groovy.xml.XmlSlurper`. Ta však strukturu vytváří tzv. „líně“, tedy až když je potřeba.

Přístup SAX je tedy více flexibilní a dává programátorovi možnost volby způsobu zpracování. Potenciálně je výrazně méně paměťově náročný. Pro dokumenty JSON se pak osvojil následující podobný princip.

2.7.3.3 Streaming

Streaming také označovaný jako pull parser nebo StAX (Streaming API for XML) je princip založený na sekvenčním dotazování se na strukturu dokumentu. Na rozdíl od SAX, kdy předání informace o struktuře iniciuje analyzátor, je tomu zde naopak a programátor je nucen analyzovat další prvky struktury. V širším pojetí jsou oba principy velmi podobné, liší se pouze způsob získání informace.

Tento postup je, jak bylo zmíněno, častý pro dokumenty JSON. Již zmíněné knihovny GSON[24] a Jackson[25] poskytují rozhraní pro toto zpracování. `org.json`[23] však streaming neimplementuje. U dokumentů XML je pak možné využít třídu `javax.xml.stream.XMLEventReader`.

2.7.3.4 Zhodnocení

Načtení přes přístup DOM představuje nutnost pro použití XPath a JSONPath. To však přináší omezení na velikost zpracovávaných souborů vzhledem k tomu, že hrozí nevejití se do operační paměti. Nabízí se tedy v těchto případech přechod na zbylé dva způsoby načtení a interní konverzi dotazů např. na GPath dotazy. Nicméně soubory takto veliké by měly být spíše hraničním případem.

SAX a Streaming pak také představují potenciální řešení pro rychlou analýzu struktury dokumentu.

2.8 Anonymizační model

Anonymizační model je nastavení konkrétních prvků anonymizace a způsob jeho uložení. Část modelu je již definována v bakalářské práci pana Orta[6] nebo převzata z aktuální verze modulu aplikace Winch. Anonymizační model vychází z doménového modelu rozebraného v kapitole 2.4.

2.8.1 Převzaté vlastnosti

Jde především o prvky definující aplikaci jako celek nebo metody způsobu zapsání konfigurací.

- Uložení konfiguračních souborů ve formátu JSON.

  Anonymizační model je ukládán do dvou druhů souborů. Prvním je kontext aplikace, druhým jsou poté jednotlivé konfigurační soubory korespondující s dokumenty určenými k anonymizaci. Oba jsou shodně uloženy ve formátu JSON.

- Kontext aplikace.

  Kontext jako takový slouží pro uložení různých verzí prostředí. Ukládá informace o připojení a umístění zdroje dat a cílového úložiště. Jak již bylo zmíněno v kapitole o doménovém modelu, aplikace v současné době neumožňuje různé kontexty. Soubor kontextu dále zahrnuje použitý vzor společný pro celou aplikaci (vzor však lze předefinovat v konfiguračním souboru jednotlivých anonymizovaných souborů) a mapování mezi anonymizační třídou a anonymizační funkcí. V aktuální verzi je možné pro jednotlivé třídy definovat pouze konstantní hodnoty parametrů volaných funkcí.

- Konfigurace souboru určeného k anonymizaci.

  Každý konfigurační soubor je vždy definován pro jednu položku. Tou může být jeden nebo více souborů určených k anonymizaci. Použití pro více souborů je docíleno tak, že parametr popisující relativní umístění anonymizovaného souboru ve zdrojovém úložišti umožňuje zahrnutí zástupných znaků `*` nebo `?`. Tato funkcionalita byla však pouze navržena, implementována není. Dále je v konfiguračním souboru možno uložit již zmíněný vzor specifický pro zpracovávaný soubor či soubory. V případech, kdy vzor celé aplikace nevyžaduje kontextové informace cílového úložiště, ale pro vzor specifikovaný v konfiguračním souboru jsou nutné, je potřebné tyto informace uložit zde. V konfiguračním souboru je také uložen název výsledného anonymizovaného souboru včetně absolutní cesty k němu.

- Místo uložení konfiguračního souboru

  Konfigurační soubory je možné ukládat v kořenu cílového adresáře nebo také v jeho jakémkoli podadresáři. Aplikace poté provádí detekci všech takových souborů s validní JSON strukturou definující anonymizační model uvnitř podadresářů cílové složky.

2.8.2 Nové vlastnosti

Některé nové vlastnosti již byly definovány v doménovém modelu. Jde například o dataset nebo anonymizovatelný prvek. Zde jsou dále doplněny následující prvky, které budou v rámci anonymizačního modelu ukládány.

Strukturu definující soubor – jde o adresu souboru v rámci zdroje dat, který definuje strukturu uložených dat. Podporované formáty byly probrány v kapitole 2.5. U případů, kdy takovýto soubor není k dispozici, se využije následující prvek.

Stromové schéma je pohled na strukturu XML nebo JSON dokumentu, kde složené prvky se stejnou či skoro stejnou (některý prvek může chybět) strukturou na stejné úrovni stromu popisujícího takový dokument jsou seskupeny do jednoho. Typicky jde o prvky kolekce nějakých entit. V případě XML mají stejný název elementu a stejného rodiče (jsou uvnitř identického elementu). U souborů JSON se pak jedná o objekty uvnitř polí.

Jak již bylo zmíněno, model je ukládán do souborů formátu JSON. Nejvyšším prvkem pro definici anonymizace dokumentů XML a JSON bude pole `datasets`. To bude obsahovat objekty s textovou vlastností `query`, přestavující dotazovací příkaz, který určil daný dataset. Dalším parametrem pak bude pole `anonymizableItems` vymezující samotné anonymizovatelné prvky určené k anonymizaci. Ty, krom prvku `anonymizationClass` definujícího anon. třídu, budou obsahovat dopředný relativní dotaz (dotaz, který bude přistupovat pouze k potomkům v rámci stromové struktury) popisující cestu k prvku v rámci datasetu. Tento dotaz bude zaznamenán v atributu `relativeQuery`. Příkazy pro určení datasetu a konkrétního anonymizovatelného prvku se při vykonání anonymizace spojí v jeden. Příklad v XPATH: dataset, který je definován dotazem `//Osoba` a anon. prvek určený `KrestniJmeno/text()` budou sloučeny na dotaz `//Osoba/KrestniJmeno/text()`.

Anonymizační model také umožní nastavení parametrů typu proměnná (Groovy výraz) a anonymizovatelný prvek (odpovídá označení „sloupec“ používaný při anonymizování v relačních databázích). Dále bude upraven způsob předání konstantních parametrů. Seznam atributů se uloží u jednotlivých položek v rámci pole `anonymizableItems` a bude odpovídat vlastnosti `arguments`. Hodnotou tohoto parametru bude pole, obsahující objekty, které budou dále zahrnovat vlastnosti

- `type` definující typ parametru a mající hodnoty z množiny {constant, variable, column} a
- `value` symbolizující hodnotu samotného parametru, kdy pro jednotlivé typy bude znamenat následující:
  - konstanta – textově zapsaná hodnota parametru,
  - proměnná – výraz v jazyce Groovy a
  - anonymizovatelný prvek – dotazovací příkaz ve formě dopředného relativního dotazu v rámci datasetu.

Celé mapování může vypadat následovně (v ukázce je využit dotaz jazyka XPATH).

```
"datasets" : [ {
    "query": "//Osoba",
    "anonymizableItems": [
        {
            "relativeQuery": "KrestniJmeno/text()"
            "anonymizationClass": "First name",
            "arguments": [ {
                "value": "Pohlavi/text()"
            }]
        },
        {
            "relativeQuery": "Prijmeni/text()",
            "anonymizationClass": "Last name",
            "arguments": [],
            "decorators": "NotNulDecorator('N/A')"
        }
    ]
} ]
```

Atributy předávané anonymizačním funkcím půjde stejným způsobem definovat i pro samotnou anonymizační třídu. Uložení této vlastnosti modelu bude provedeno tak, že do jednotlivých objektů z pole atributu `anonymizationClassFunctionMapping` nacházejícího se v kontextu aplikace bude přidána položka `attributes`. Ta bude moci mít stejný obsah jako u mapování dotazů, ale typ anonymizovaný prvek (column) nebude možné použít. Případné kombinování atributů bude provedeno v pořadí

1. obecné pro anonymizační třídu (z kontextového souboru) a

2. specifické pro dotaz (definované v konfiguračním souboru).

Kombinování parametrů je však definováno spíše jako budoucí možnost, funkce momentálně mají maximálně dva parametry, kdy prvním bývá původní hodnota.

Dalším prvkem nastavení anonymizační třídy bude seznam dokorátorů. Ten bude zaznamenán v podobě textové položky identifikované klíčem `decorators`. Jednotlivé dekorátory bude možné analogicky s řešením v relačních databázích řetězit pomocí svislé čáry '|'.

Do konfiguračních souborů bude v případech, kdy nebude k dispozici schéma definující soubor, uloženo detekované stromové schéma. Popisovat ho bude atribut `treeSchema`. Pro krátký XML soubor s následujícím obsahem

```
<EntitiesA>
    <EntityA id="1">
        <param1>value</param1>
        <param2>value</param2>
        <param3>value</param3>
    </EntityA>
    <EntityA id="2">
        <param1>value</param1>
        <param2>value</param2>
    </EntityA>
</EntitiesA>
```

bude stromové schéma zaznamenáno takto.

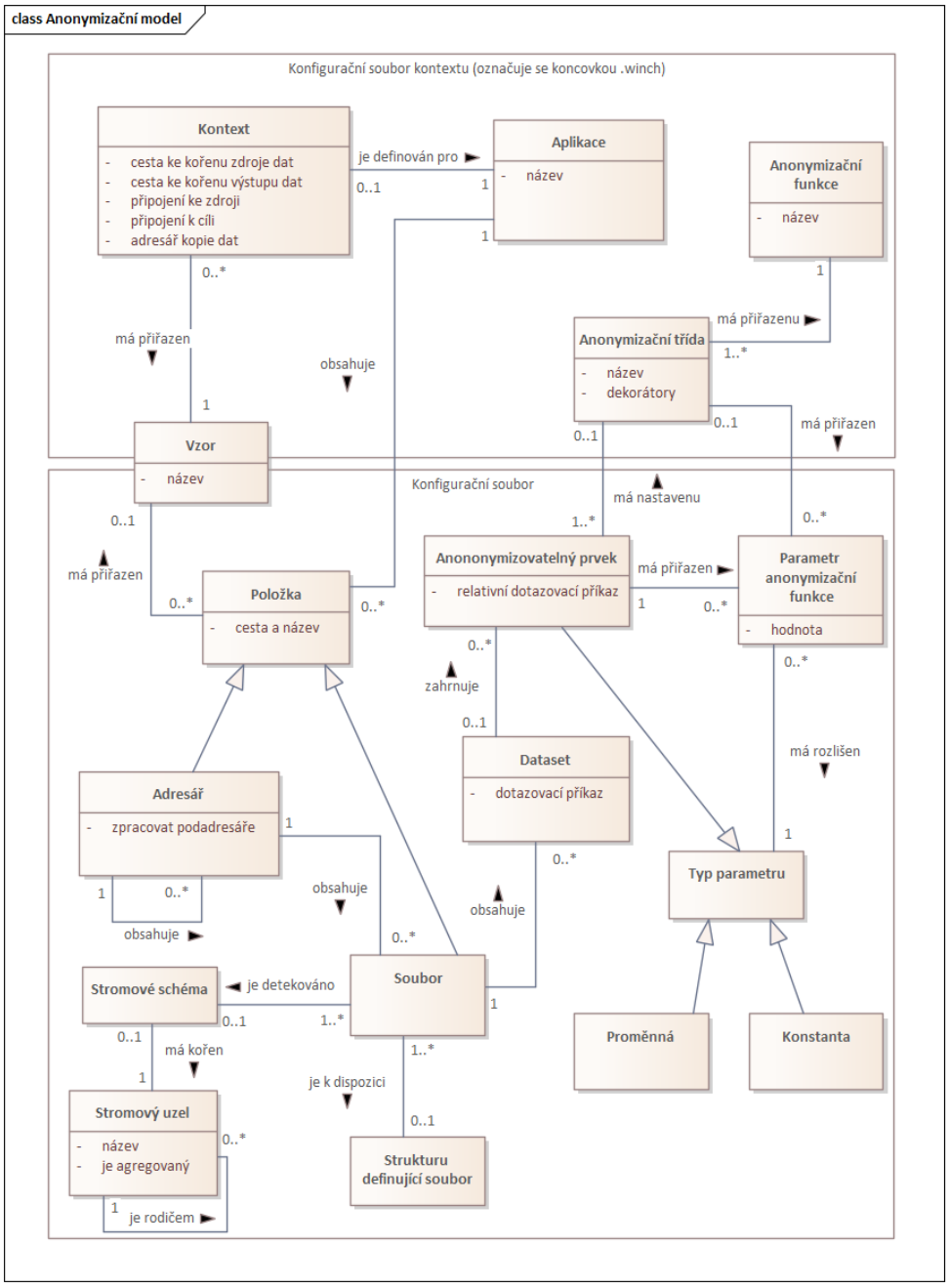
```
"treeSchema": {
    "EntitiesA": {
        "isAgregated": false,
        "EntityA": {
            "isAgregated": true,
            "<attributes>": ["id"],
            "param1": null,
            "param2": null,
            "param3": null
        }
    }
}
```

V případech, kdy bude k dispozici schéma definující soubor typu DTD nebo XSD pro XML dokumenty a JSON Schema pro soubory JSON, bude jméno a cesta k takovému souboru uloženo v atributu `schemaFile`. Uživatelské rozhraní umožní nadefinování anonymizovaných prostřednictvím náhledu na tento soubor.

Použití zástupných symbolů v definici cesty a názvu dokumentu uvnitř konfiguračního souboru umožní, že všechny nové přidané prvky mohou být společné pro více souborů. Kompletní anonymizační model je zachycen na obrázku 2.13.

2.9 Architektura aplikace

Následující kapitola popisuje strukturu aplikace. Jak již bylo zmíněno v kapitole 2.1, modul je tvořen relaxovanou 3-vrstvou architekturou. Ta se skládá ze tří balíčků, a to datový, aplikační a prezentační. V této kapitole bude nahlédnuto podrobněji na jednotlivé vrstvy. Prolínají se zde již existující balíčky a třídy s nově navrženými. Nové prvky jsou v diagramech odlišeny tmavšími barvami.



Obrázek 2.13: Anonymizační model

2.9.1 Datová vrstva

Tato vrstva obsahuje následující balíčky.

dao Tento balíček definuje pouze jedno rozhraní `ConfigurationsDAO`, které je navržené ke komunikaci s datovou vrstvou (především k práci s anonymizačním modelem). Implementuje ho třída `DataLayerAccessJsonFiles` z balíčku `dl.services.anonymizationModelManager`. Nicméně jako jediné rozhraní s vrstvou neslouží, aplikační vrstva přistupuje i k jiným třídám.

entity Zde jsou situovány třídy popisující doménový (anonymizační) model. Hlavní entitou je třída `WinchApplication`, která schraňuje všechny informace o aplikaci, jako je název aplikace, kontext, pracovní kořenový adresář nebo místo uložení aplikace. Dále si drží seznamy konfigurací strukturovaných (tabulkových) a nestrukturovaných souborů či rejstřík provedených úkolů. Strukturované soubory obsahují seznamy tabulek, které poté zahrnují sloupce. Od `WinchApplication` dále dědí třída `RWinchApplication`, kde R zde představuje recommended, tedy doporučený. Doporučenost je ve smyslu vhodnosti (nebo nevhodnosti) souboru k anononymizaci v podobě „black listu“ a „white listu“. Tyto listy obsahují položky, jenž symbolizují soubory nebo složky.

Pro reprezentaci konfigurace u XML a JSON dokumentů byla navržena třída `FileConfigutationTreeStructured` dědící od třídy `FileConfiguration`. Analogicky se strukturovanými a nestrukturovanými soubory si aplikace Winch drží seznam konfigurací stromových souborů. Jak bylo navrženo v kapitole 2.8, anonymizaci dokumentů se stromovou strukturou nejprve definuje seznam datasetů určený dotazovacím příkazem. Tyto informace schraňuje třída `Datasets`. Jednotlivé datasety poté drží seznam anonymizovatelných prvků v podobě instancí třídy `AnonymizableItem`. Ta v sobě pak ukládá informace o anonymizační třídě, parametrech předaných anonymizační funkci a dekorátorech. V případě parametrů jde o množinu prvků třídy `AnonymizationClassArgument`.

Pro stromové soubory je také nutné ukládat informace o struktuře. Ta bude buď určená proměnnou `schemaFilePath` v třídě `FileConfigutationTreeStructured`, pokud ji definuje soubor pro popis schématu. Nebo odkazem na `TreeNode` v podobě proměnné `treeSchema` u případů, kdy bude struktura získána průchodem dokumentu. Závislosti v rámci balíčku jsou zachyceny na obrázku 2.14.

exception Zahrnuje výjimky patřící pod datovou vrstvu.

services Balíček zaštiťuje třídy, které provádějí manipulace se soubory. Je rozdělen na několik další balíčků.

anonymizationModelManager Obsahuje již zmíněnou třídu `DataLayerAccessJsonFiles`, která implementuje rozhraní `ConfigurationsDAO` z dao. Dále je v balíčku třída `AnonymizationModelLoader`, jenž slouží jako obálka (wrapper) nad třídami z balíčku `services.jsonLoader`.

converter Zde je jediná třída `SpreadsheetRowColumnIndexConverter` poskytující metody na převod názvů sloupců u tabulkových souborů na jejich vnitřní reprezentaci.

fileHandling Skýtá třídu `FileOperations` sloužící k operacím se soubory, jako jsou zápis do souboru, ověření existence, vytvoření, přesunutí nebo smazání souboru. Taktéž poskytuje metody pro zpracování cest v rámci souborového systému.

itemPersist Zahrnuje rozhraní `IItemLoader` definující metody pro práci s položkami (v black a white listech) a třídu `ItemObjectLoader`, která je implementuje.

jsonParser Zaštiťuje třídy provádějící načítání konfiguračních souborů JSON. Nejvýše je v hierarchii postavena třída `ModelFileParserAnyFile` poskytující základní metody pro načítání společných atributů. Ostatní třídy `ModelFileParser*` potom vytvářejí strom tříd zobrazený na obrázku 2.15, kde každá třída zajišťuje načtení informací specifických pro její cíl (např. kontext nebo tabulkový soubor). Pro zpracování dokumentů se stromovou strukturou je přidána třída `ModelFileParserTreeStructured`.

V balíčku je také třída `jsonParser`, která poskytuje metody pro práci na úrovni jednotlivých prvků JSON souborů. Využívá k tomu knihovnu `org.json`[23].

jsonSaver Analogicky s předchozím balíčkem poskytuje třídy pro ukládání jednotlivých konfiguračních souborů. Nejvyšší třídou je zde `ModelFileSaverAnyFile`. Na rozdíl od načítání, ukládání konfiguračních souborů Csv dokumentů není odlišeno od ostatních tabulkových formátů. Uložení konfiguračních souborů stromových dokumentů bude provádět třída `ModelFileSaverTreeStructured`.

loader Zde je umístěna třída `ApplicationLoader` orchestrující kompletní načítání všech souborů s konfiguracemi. Ta dále využívá třídu `CombinedLoader` z podbalíčku `dataLoader`, jenž skládá dohromady jednotlivé části načtení aplikace, definované v metodách `load` uvnitř traitů pojmenovaných `*Loader` (např. `BlackListLoader` nebo `DestinationLoader`).

provider Balíček obsahuje pouze třídu `ApplicationProvider` vytvářející instanci `RWinchApplication` na základě dat zadaných ve formuláři pro vytvoření nové aplikace.

saver Poskytuje třídu `ApplicationSaver` zodpovědnou za uložení všech konfigurací.

2.9.2 Aplikační (business) vrstva

Tato vrstva zahrnuje logiku aplikace. Dělí se na další balíčky.

dataSelection Tento balíček je dále rozdělen na `service` a `utilities`. V „servisním“ balíčku se nalézá třída `DataSelectionService`. Ta poskytuje metody pro zpracování výběru dat (tabulek a sloupců) z tabulkových dokumentů. Pro ověření platnosti zvolené konfigurace využívá třídy `ColumnValidator` a `TableValidator` z jmenného prostoru `utilities.validator`. V balíčku `utilities` se pak ještě nachází třída `ExcelColumnConverter`, která poskytuje konverze mezi číselnou a textovou reprezentací označení sloupce tabulky. Třída se zdá být poměrně duplicitní s třídou `SpreadsheetRowColumnIndexConverter` z datové vrstvy.

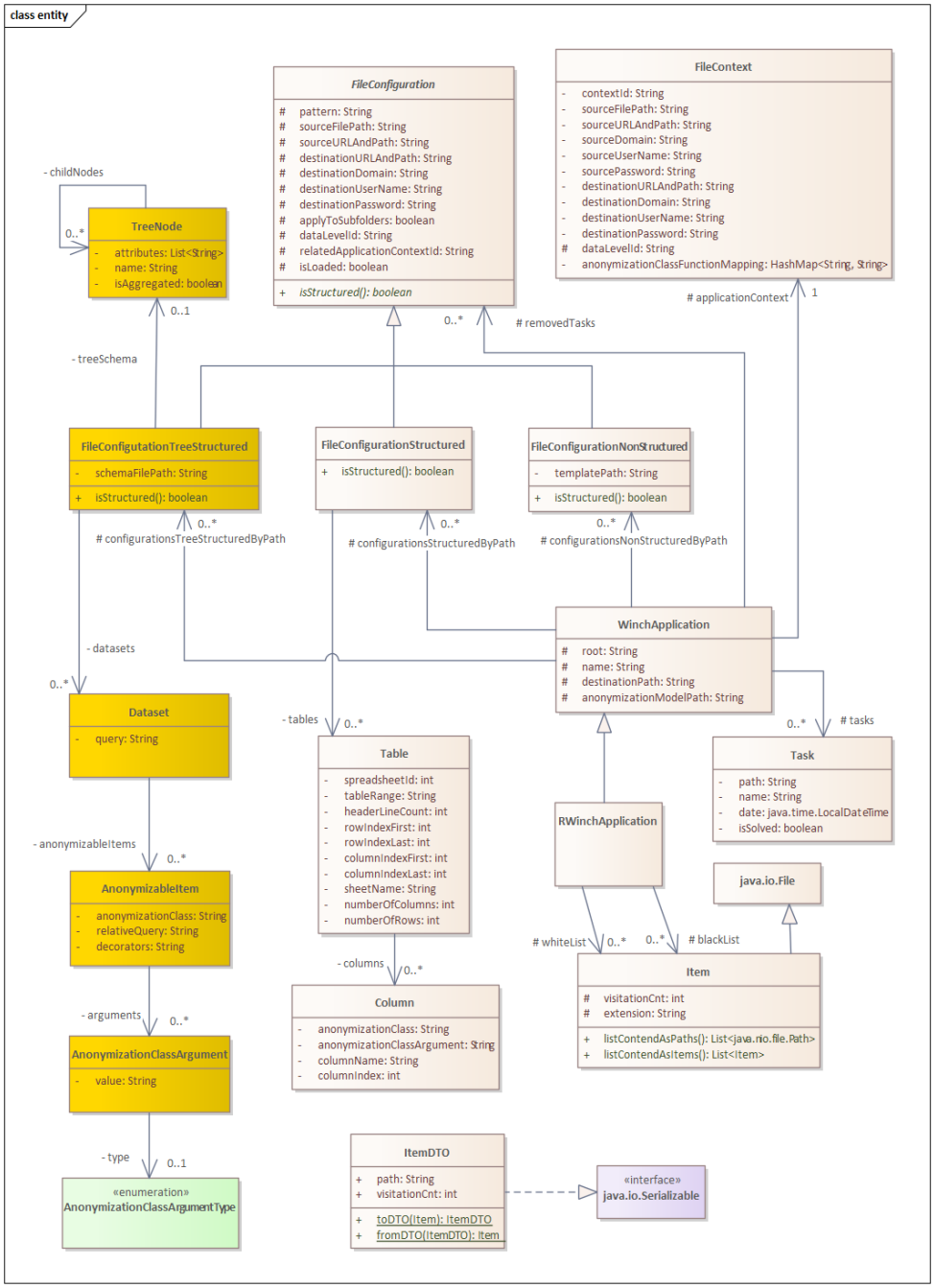
Pro stromové soubory jsou do balíčku `service` navrženy třídy `XmlTreeDataSelectionService` a `JsonTreeDataSelectionService`. Ty budou dědit od abstraktní třídy `ATreeDataSelectionService`. Hlavním úkolem bude zpracovávat výběry ze stromové struktury na potřebné dotazy v jazycích XPath a JSONPath.

exception Výjimky pro aplikační úroveň.

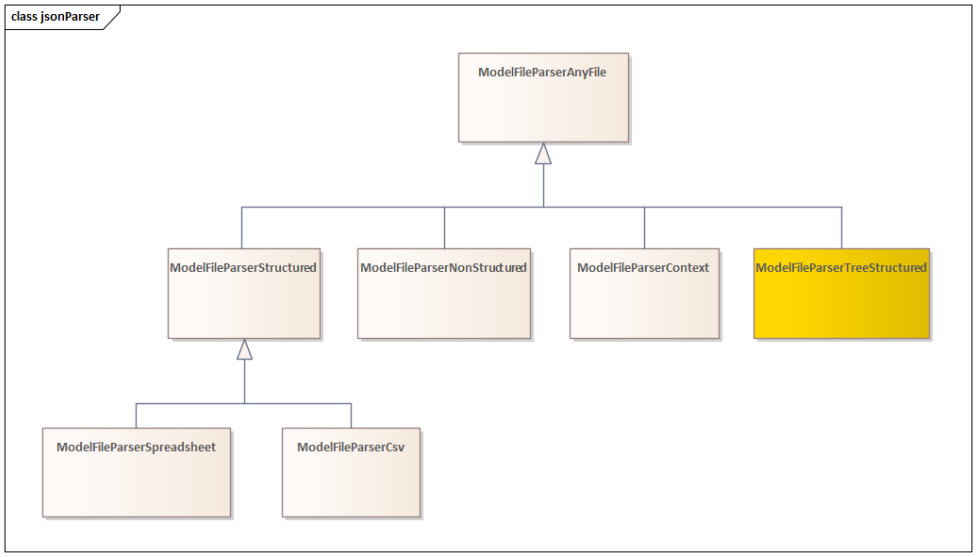
fileSelection Balíček zaštiťující proces výběru souborů určených k anonymizaci. Dále se dělí na menší balíčky.

browser Obsahuje třídu `RegexSearch`, která implementuje metody pro nalezení adresářové cesty v souborovém systému na základě regulárního výrazu. Tyto metody jsou definované v rozhraní `IRegexSearch` ze stejného balíčku. Dále se ve jmenném prostoru nachází třída `BrowserService` implementující zmíněné rozhraní, ale pouze tak, že volá řešení z `RegexSearch`. Asi i proto je třída označena jako zastaralá. Nicméně implementuje také rozhraní `IItemManager` ze sousedního balíčku `fileSection.services`, které určuje signatury metod pro práci s black a white listy a obecnými položkami.

recommendation Zde jsou zastřešeny třídy určující soubory doporučené k anonymizaci. Nalézá se zde třída `DiscoveryConfigurer`, jenž je zodpovědná za vytvoření konfigurace. Momentálně je toho docíleno tak, že je spuštěna externí aplikace *WinchUDDConfiguration.exe*. Dále se v balíčku nachází třída `GradualFileAdviser` poskytující metodu `discoverNext`. Opakované volání této metody provádí samotnou detekci citlivých dat. Vnitřně je k tomu použita knihovna `org.apache.tika.Tika`.



Obrázek 2.14: Balíček entity



Obrázek 2.15: Hierarchie parserů konfiguračních souborů JSON

services Krom již zmíněného rozhraní `IItemManager` se zde nachází další dvě, a to `IPersistentItem` a `ITaskManager`. Ty definují metody pro práci s položkami, respektive úkoly. Třídy `ItemService` a `TaskService` poté implementují rozhraní `IItemManager` a `ITaskManager`.

utilities Obsahuje dvě uvtovárny, pro vytvoření položek (Item) a úkolů (Task). Pojmenovány jsou `ItemFactory`, respektive `TaskFactory`. Dále je zde podbalíček `validators` zahrnující třídy pro ověření platnosti složky ze souborového systému. Nachází se zde také třída `CreateApplicationDialogValidatorNew` zastřešující kontrolu správného vyplnění formuláře pro vytvoření nové aplikace.

filesystem Tento balíček zastřešuje hlavní logiku anonymizace. Je rozdělen na další 4 existující balíčky a jeden nově navržený.

anonymization Obsahuje třídu `AnonymizationManager`, která poskytuje metodu `applyAnonymizationToStructuredFile`. Tato metoda řídí anonymizaci jednoho tabulkového souboru. Implementace pro konkrétní formát dokumentu je určena instancí rozhraní `StructuredFileManipulator` ze sousedního balíčku `filesystem.structuredFileProcessor`. Připraveny jsou řešení pro formáty XLS a XLSX třídami `TableManipulatorXLS` respektive `TableManipulatorXLSX`. `TableManipulatorCSV` na svojí implementaci čeká. Dále je v balíčku `anonymization` podbalíček `mapping` ob-

sahující třídu, jenž umožňuje načtení mapování mezi anonymizační třídou a anonymizační funkcí.

connectionHandling Balíček schraňuje třídy určené pro práci se soubory na lokálním či vzdáleném disku. Potřebné operace abstrahuje třída `AbstractConnectionManager`. Vytvořena je však pouze implementace pro lokální souborový systém v podobě třídy `ConnectionManagerLocalFileSystem`.

executor Třída `FileSystemExecutor` v tomto balíčku obsahuje statickou metodu `execute()`, kterou se anonymizace spouští. Implementace se však v době analýzy potýká s nefunkčním načtením konfigurace.

structuredFileProcessor Jak již bylo zmíněno u balíčku `anonymization`, zde jsou situovány třídy pro zpracování anonymizace tabulkových dokumentů.

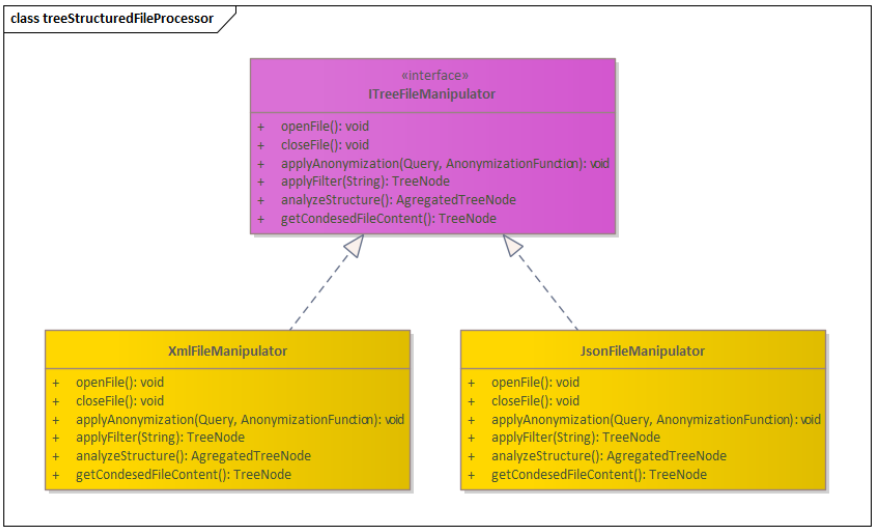
treeStructuredFileProcessor Zde je navrženo rozhraní `ITreeFileManipulator` definující několik metod pro zpracování dokumentu se stromovou strukturou. Jde především o metody provádějící anonymizaci, filtraci dle dotazu či analýzu struktury. Třídy `XmlFileManipulator` a `JsonFileManipulator` poté budou mít za úkol implementovat toto rozhraní pro jejich formáty. Toto je naznačeno na obrázku 2.16.

pattern V tomto balíčku jsou umístěny implementace vzorů (způsoby realizace anonymizace). Jednotlivé vzory dědí od abstraktní třídy `AbstractFilePattern`. K dispozici jsou následující implementace.

- `CreateAsCopyFilePattern` – pouze překopíruje soubor ze zdroje do cíle, neprovádí anonymizaci.
- `CreateAsNewFilePattern` – vytvoří kopii souboru a na ni provede odstranění citlivých dat.
- `DeleteFilePattern` – smaže anonymizovaný soubor.
- `ReplaceFilePattern` – zamění soubor ve zdrojovém úložišti za soubor šablony definovaný v konfiguraci.
- `UpdateFilePattern` – provede anonymizaci přímo na určeném souboru, nevytváří kopii.

Patterny jsou vytvářeny mechanizmem reflexe metodou `createInstanceOfPatternByName` z třídy `PatternFactory`, která se nachází ve stejnojmenného podbalíčku. Jednotlivé implementace jsou realizovány pomocí „kroků“, jenž jsou definované v následujícím balíčku.

step Kroky lze rozdělit na ty, které pracují s připojením (ať už ke zdroji nebo k cíli) či manipulují se soubory. Nejzajímavější je však krok



Obrázek 2.16: Balíček treeStructuredFileProcessor

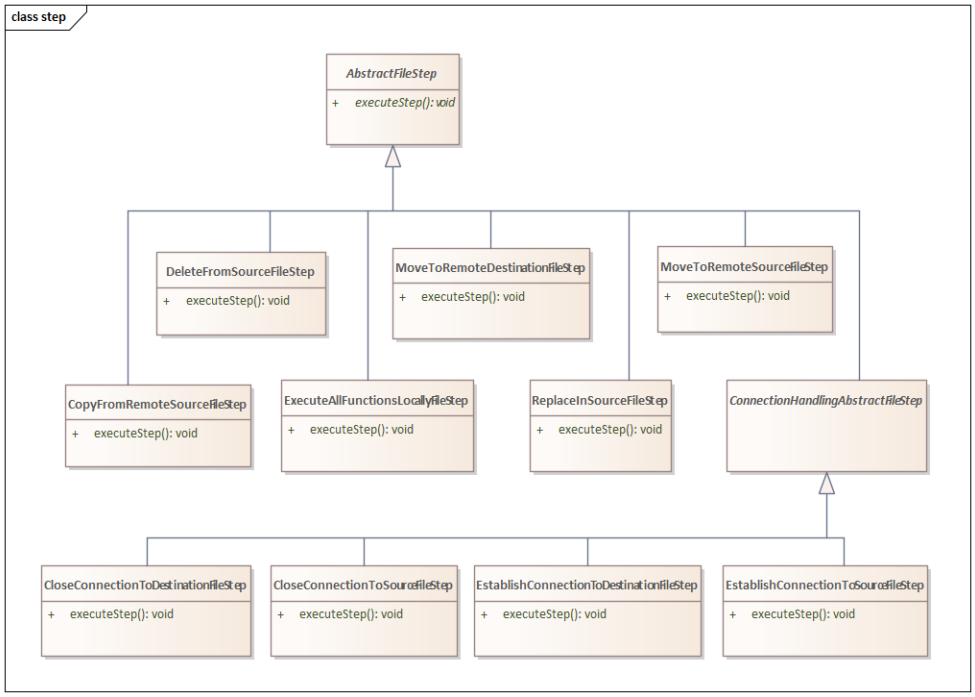
`ExecuteAllFunctionsLocallyFileStep` iniciující samotnou anonymizaci. Všechny kroky dědí od abstraktní třídy `AbstractFileStep`, která definuje klíčovou abstraktní metodu `executeStep`, jenž jednotlivé kroky musí implementovat. Pro kroky pracující s připojením je tu ještě abstraktní nadtřída `ConnectionHandlingAbstractFileStep`. Celou hierarchii je možné vidět na obrázku 2.17.

Sekvence volání při anonymizaci strukturovaných souborů v rámci aplikační vrstvy je zobrazena na snímku 2.18. Caller zde symbolizuje volání z uživatelského rozhraní.

2.9.3 Prezentační vrstva

Tato vrstva představuje uživatelské rozhraní. Jeho realizace dle názvů balíčků vychází z návrhového vzoru MVC (Model-View-Controller). Nacházejí se zde View a Controller. Model pak představuje především datová vrstva, ke které se většinou přistupuje přes aplikační. Uživatelské rozhraní je sestaveno z komponent knihovny Swing.

controllers Úkolem kontrolerů je zpracování vstupů od uživatele a jejich předání aplikační, potažmo přímo datové vrstvě. Balíček přímo obsahuje třídu `NewController`, která je zodpovědná události z obrazovky pro vytvoření nové anonymizační aplikace. Kontroler však neimplementuje reakci na nastavení mapování mezi anonymizačními třídami a anonymizačními funkcemi. Rovněž informace o zadaném adresáři pro vložení kopie anonymizovaného souboru není zpracována. V balíčku je také kon-



Obrázek 2.17: Hierarchie balíčku step

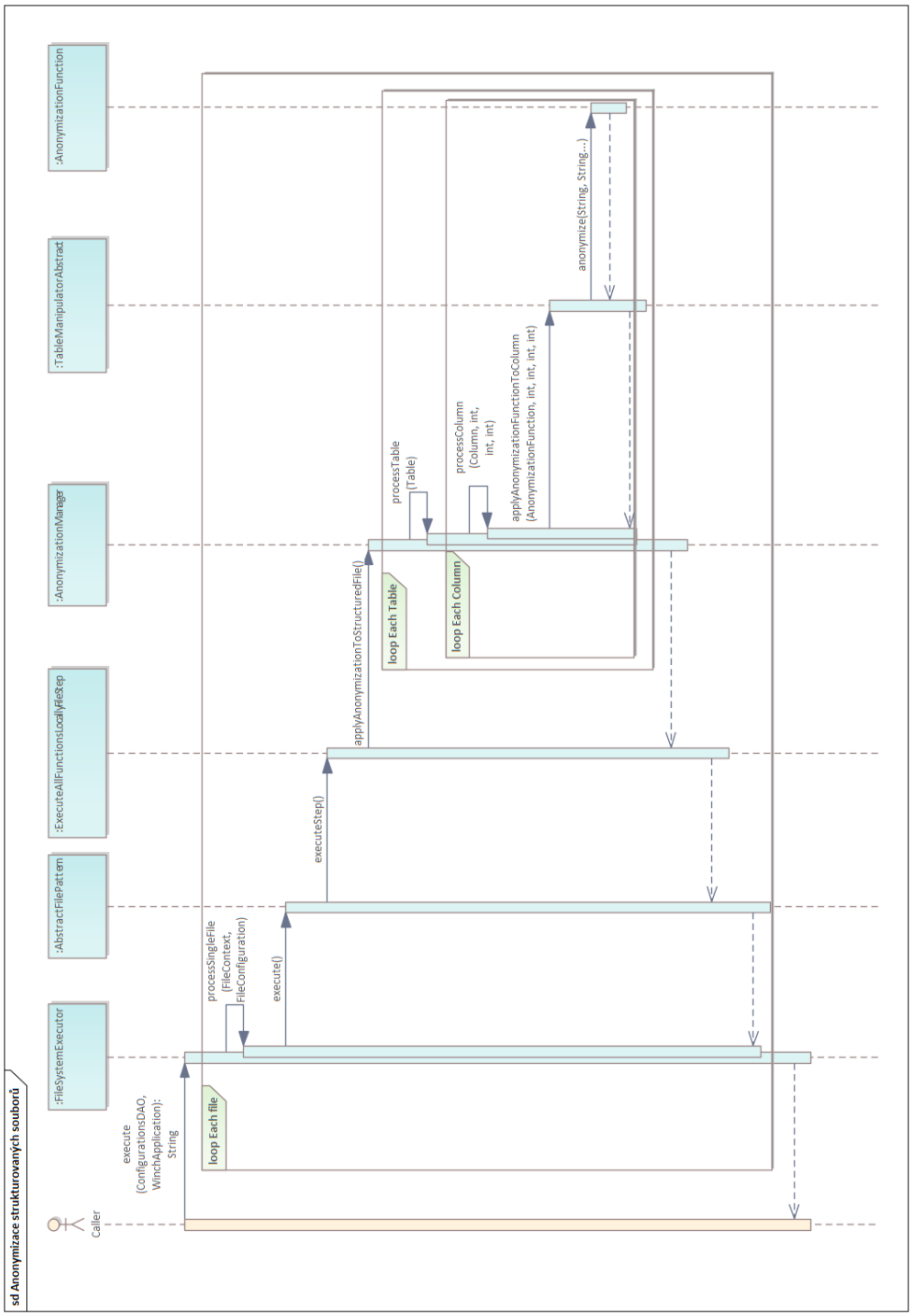
trolér `StartController` pro reakci na příkazy z úvodní obrazovky programu. Obsaženy jsou ještě dva podbalíčky.

anonymization Zde se nacházejí třídy `AnonymizationController`, `ColumnListController` a `TableListController`, které zpracovávají operace prováděné na obrazovce nastavující anonymizaci tabulkových dokumentů.

Vzhledem k implementaci dalších typů souborů navrhuji zavést další úroveň balíčků, podobně jako to je zavedeno v sousedním balíčku `view.anonymization`. Třídy zpracovávající vstupy z nastavení tabulkových dokumentů budou přesunuty do podbalíčku `strucutred`. Dále vznikne balíček `treeStructured` zpracování akcí z obrazovky pro nastavení anonymizace u XML a JSON souborů.

utils Tento balíček obsahuje třídu `FFileChooser` zaštiťující práci s komponentou `javax.swing.FFileChooser`. Nalézá se zde také balíček `crate`, jehož dvě třídy slouží jako pomocné pro obrazovku nové aplikace.

view Tento balíček by měl obsahovat pouze třídy reprezentující vizuální komponenty aplikace. Nicméně je možné sledovat, že se zde objevují mnohé metody pro zpracování vstupů. Třídy `StartWindow` a `NewWindow`



Obrázek 2.18: Sekvence volání při anonymizaci tabulkových souborů

představují úvodní okno programu, respektive formulář pro vytvoření nové anonymizační aplikace. Hlavní část uživatelského rozhraní pak tvoří tabulátorový rámec implementovaný třídou `TabWindow`. Obsahy jednotlivých záložek (tabů) jsou rozděleny do dalších balíčků.

anonymization Sdružuje záložky sloužící pro nastavení anonymizace na konkrétních souborech. Je dále rozdělen na zobrazení strukturovaných a nestrukturovaných dokumentů. Implementovány jsou však pouze strukturované soubory.

Podobně jako v balíčku `controller.anonymization` zde bude zaveden nový podbalíček `treeStructured` pro komponenty související s nastavováním anonymizačního modelu pro dokumenty se stromovou strukturou.

discovery Tento balíček zahrnuje vše spojené se zobrazením záložky pro výběr či nalezení souborů vhodných k anonymizaci.

lists Obsahuje třídy zobrazující seznamy vybraných úkolů, které prostupují napříč jednotlivými záložkami. Je zde možné vidět i další listy jako seznam odmítnutých souborů.

summary Představuje závěrečnou shrnující záložku. Jeho součástí je i seznam odmítnutých souborů v aplikaci označovaný jako black list.

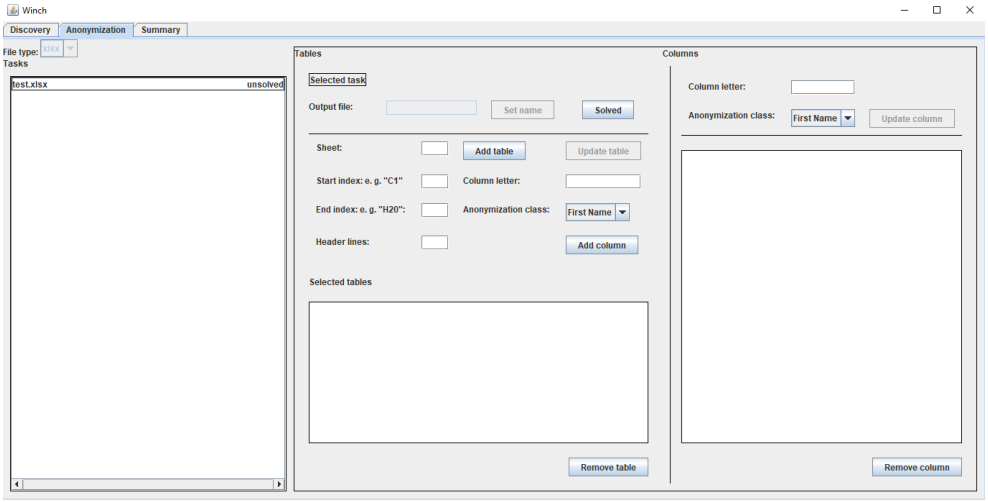
2.10 Uživatelské rozhraní

Návrh uživatelského rozhraní se soustředí na vytvoření záložky pro nastavení anonymizace dokumentů XML nebo JSON. Z aktuálního uživatelského rozhraní (obrázek 2.19) přebírá seznam úkolů (tasks) a rozdělení na jednotlivé záložky (Discovery, Anonymization a Summary). Návrh je vyobrazen na snímku 2.20.

Hlavním prvkem umístěným jako první zleva je náhled na strukturu souboru. Ta může být určena, jak je zaznamenáno ve funkčních požadavcích, pomocí strukturu definujícího dokumentu typu DTD nebo XSD pro XML soubory nebo JSONSchema u JSON dokumentů. Struktura může být také získána analýzou samotného dokumentu. Kliknutím do strukturálního pohledu se provede zápis dotazu odpovídající provedenému výběru do textového pod pohledem. Uživatel může dotaz dále upravovat. V okamžiku, kdy je s ním spokojený, provede definici datasetu pomocí tlačítka „Create Dataset“.

Vytvořený dataset se následně objeví v seznamu napravo od strukturálního pohledu v podobě dotazu, jenž ho reprezentuje. Výběrem datasetu z tohoto seznamu se umožní nastavení anonymizace pro data, která daný dataset obsahuje. Toto nastavení probíhá v tabulce pod zmíněným seznamem datasetů. Tabulka má následující 3 sloupce.

První sloupec zobrazuje stromovou strukturu v rámci datasetu.

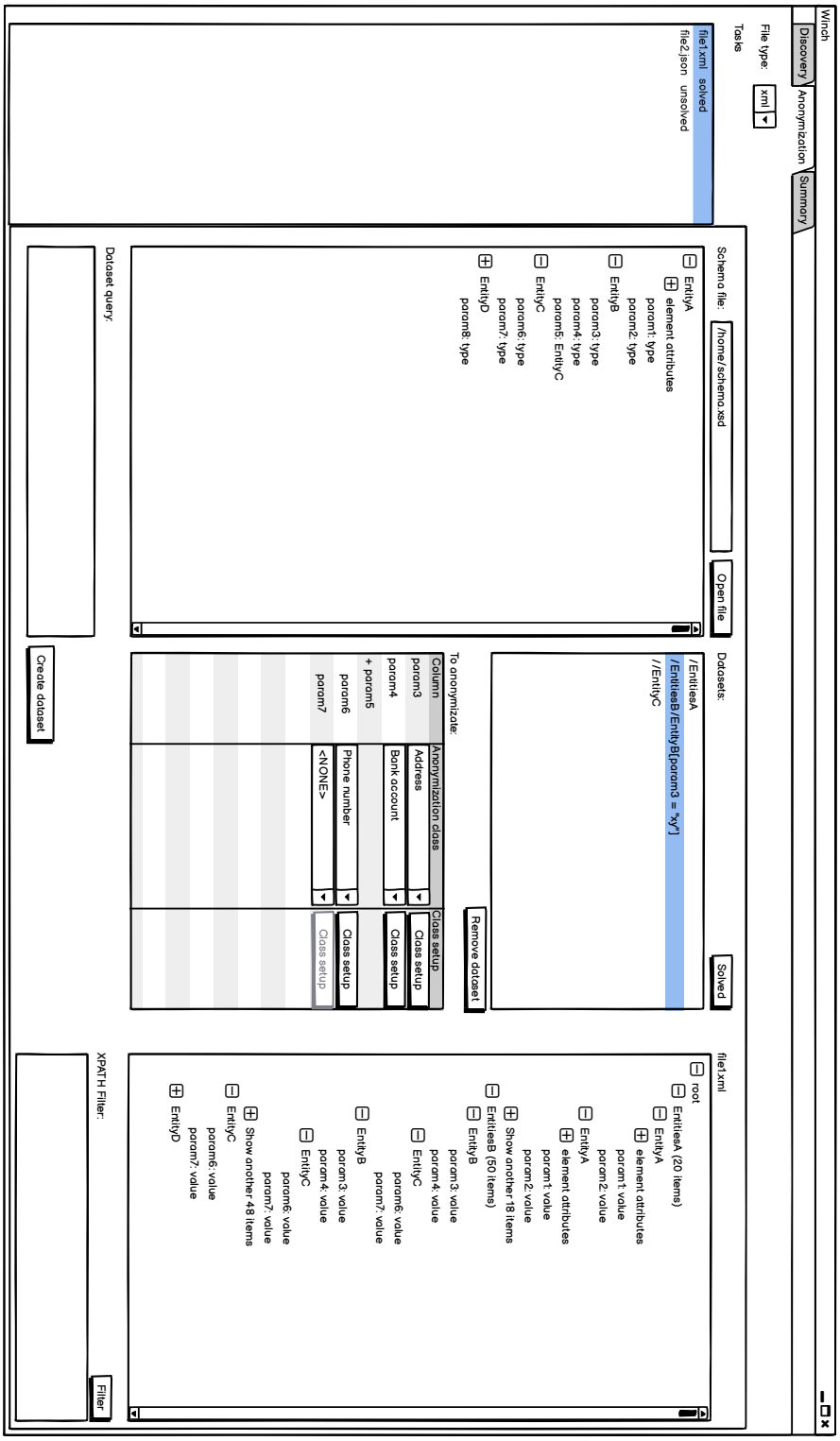


Obrázek 2.19: Existující uživatelské rozhraní pro nastavení anonymizačního modelu tabulkových dokumentů

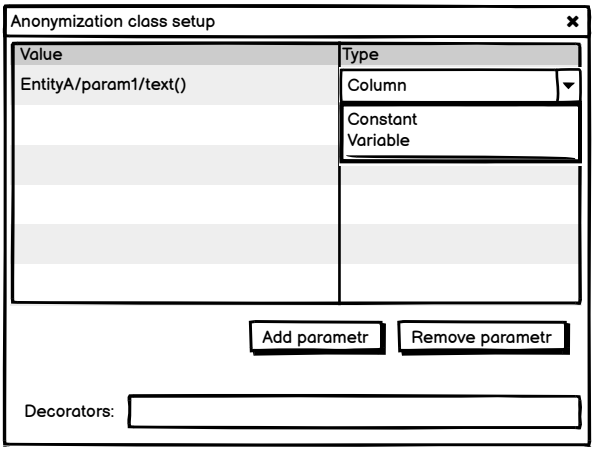
Anonymization class Vybraná anonymizační třída pro prvek v řádku. Tato možnost bude zobrazena jen u prvků identifikovaných jako anonymizovatelné.

Class setup Tlačítko, kterým se zobrazí okno2.21 pro nastavení atributů a dekorátorů specifických pro konkrétní řádek. Atributy budou následně předány anonymizační funkci. Tlačítko bude k dispozici opět jen ve vybraných řádcích.

Celkový pohled umístěný vpravo poskytne možnost zobrazení celé struktury souboru. Pod tímto pohledem je umístěna řádka pro příkazy na filtrování ze zpracovávaného souboru, jenž bude po stisknutí tlačítka „Filter“ upravovat obsah celkového pohledu. Uživatel takto bude moci vyladit svůj filtrovací výraz a poté ho použít v seznamu anonymizovaných dat.



Obrázek 2.20: Návrh uživatelského rozhraní pro nastavení anonymizačního modelu pro dokumenty XML a JSON



Obrázek 2.21: Nastavení anonymizační třídy

Kapitola 3

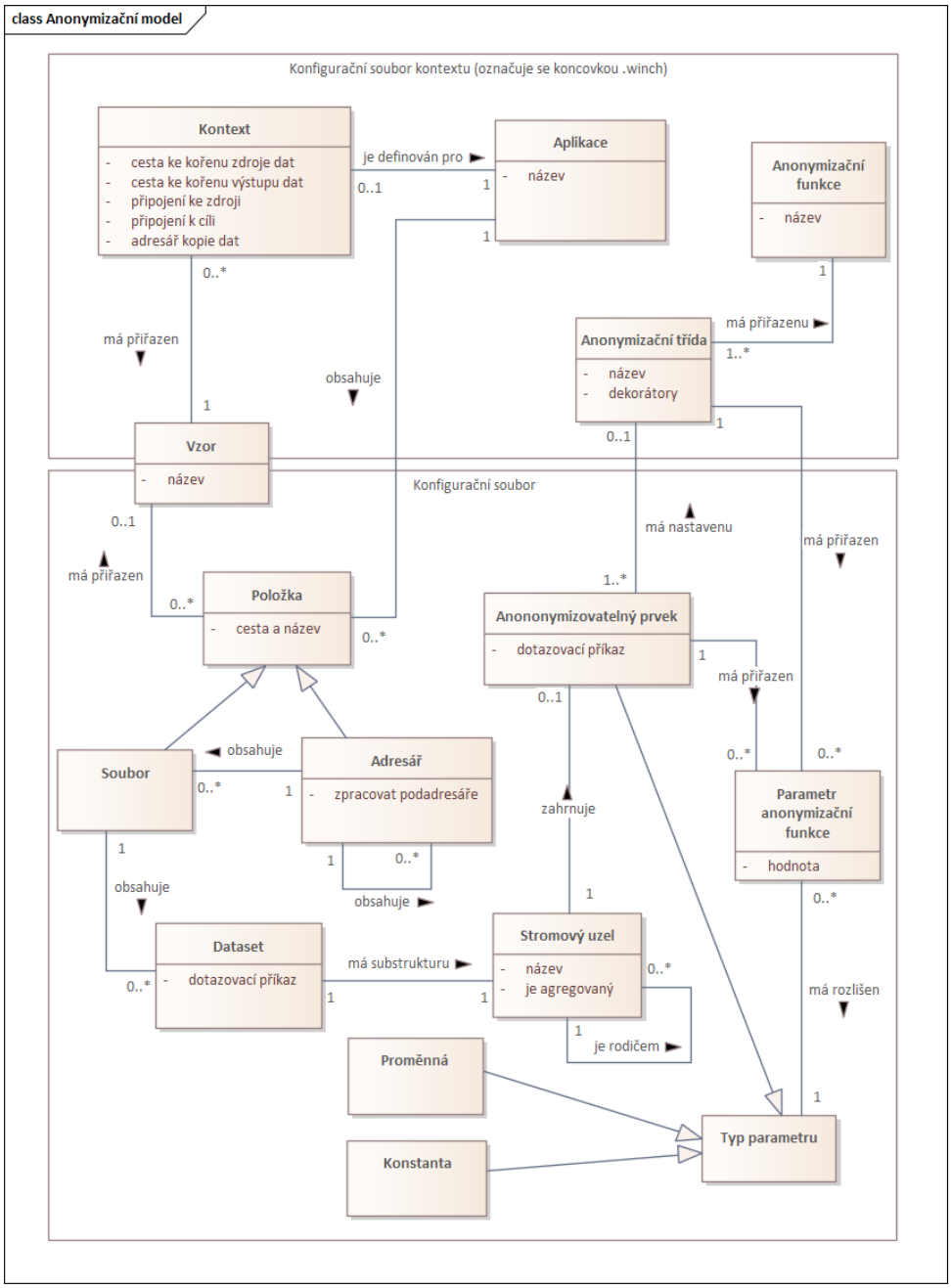
Realizace

Samotná realizace doznala menších odchylek od návrhu. Jedna úprava se týká anonymizačního modelu. Ten byl pozměněn tak, že dataset neobsahuje anonymizovatelné prvky přímo. Naopak součástí popisu datasetu se stala stromová „substruktura“ popisující data dotazem určené množiny. Popis struktury využívá stejné entity „Stromový uzel“ jako dříve zavedené stromové schéma. Do tohoto uzlu pak byla přidána možnost určit anonymizovatelný prvek. Celé toto řešení umožňuje lépe zpětně určit, jakému prvku ve struktuře datasetu odpovídá určený anonymizovatelný prvek. Realizovaný model je možné vidět na obrázku 3.1.

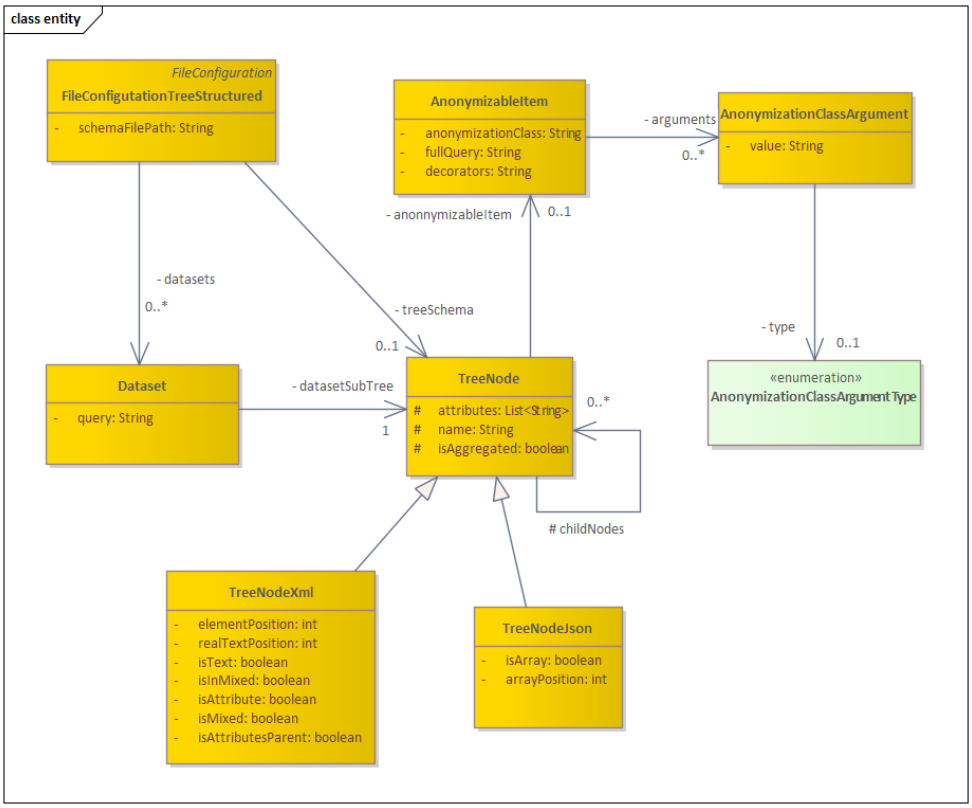
3.1 Datová vrstva

V implementaci datové vrstvy se promítly změny z anonymizačního modelu do balíčku `entity`. Úpravy kopírují právě zmíněné kroky, tedy třída `Dataset` obsahuje vlastnost `datasetSubTree` typu `TreeNode`, která reprezentuje detekovanou substrukturu. V `TreeNode` poté přibyla možnost určit anonymizovatelný prvek v podobě atributu `anonymizableItem`. Dále vznikla potřeba rozlišovat další atributy v rámci analýzy struktury. Tyto informace jsou specifické pro jednotlivé formáty, proto byly zavedeni potomci třídy `TreeNode` v podobě `TreeNodeXml` a `TreeNodeJson`. Krom toho byla z třídy `FileConfiguration` a všech jejich potomků odstraněna metoda `isStructured`. Finální stav nových prvků balíčku entity je zobrazen na snímku 3.2 (původní prvky jsou zde vynechány).

Pro načítání uložené konfigurace anonymizačního modelu byly kromě navržené třídy `ModelFileParserTreeStructured` implementovány i její specializovaní potomci. Jde o třídy `ModelFileParserXml` a `ModelFileParserJson`. Ty se starají pouze o načítání specifických atributů ze tříd `TreeNodeXml` a`TreeNodeJson`, které popisují strukturu souboru či datasetu. Společné prvky pokrývá rodičovská třída `ModelFileParserTreeStructured`.



Obrázek 3.1: Realizovaný anonymizační model



Obrázek 3.2: Implementace balíčku entity

Analogicky byly implementovány třídy zajišťující zápis konfiguračních informací pro stromové dokumenty. Jedná se o rodičovskou třídu `ModelFileSaverTreeStructured` a její potomky `ModelFileSaverXml` a `ModelFileSaverJson`. Uložení informací společných pro všechny konfigurace anonymizovaných souborů je zajištěno děděním hlavní třídy od `ModelFileSaverConfiguration` s využitím jejích metod `saveMandatoryFields` a `saveOptionalFields`. Celé řešení ukládání a načítání konfiguračních souborů se opírá o dosud k tomu používanou knihovnu `org.json`.

3.2 Aplikační (business) vrstva

Implementace logiky aplikace se v případě tříd zajišťujících provedení samotné anonymizace držela návrhu. Jde o rozhraní `ITreeFileManipulator` a jeho implementace z balíčku `filesystem.treeStrucutredFileProcessor` (možno vidět na obrázku 2.16).

Současná implementace samotné anonymizace JSON dokumentů však nese menší omezení. Použitá knihovna `com.jayway.jsonpath`[26] totiž neumožňuje

dotazem získanou množinu dat jednotlivě předat anonymizační funkci a její výsledek okamžitě zanést do načtené reprezentace souboru. Načtená reprezentace souboru také bohužel neumožňuje přepínat mezi výstupem datovým a výstupem určujícím cestu k datům. Soubor je proto načten do dvou reprezentací, jednou v datové a podruhé v reprezentaci určujícím cestu, kdy oba jsou v paměti ponechány pro opětovné použití.

Toto řešení je efektivnější v případech, kdy je určeno více anonymizovatelných položek v jednom souboru. Pokud by však byla v každém souboru pouze jedna anonymizovatelná položka, bylo by vhodnějším řešením načíst obě reprezentace postupně s tím, že reprezentaci určující cesty k datům by bylo možné po použití upustit a uvolnit tím paměť. Vhodnou optimalizací by pak byla detekce jednotlivých případů, zmapování běhového prostředí a použití vhodnější varianty pro daný případ.

Pro dokumenty XML vyvstal jiný problém. Vzhledem k možnosti anonymizovat XML dokument i bez dodání schéma popisujícího souboru, není v těchto případech možné určit, jaké bílé znaky může parser zanedbat. Při anonymizaci textu uvnitř elementu se smíšeným obsahem je tedy nutné udat pozici textu v rámci elementu a přitom brát v potaz i předchozí textové elementy, které obsahují pouze bílé znaky. Tedy například i odřádkování. Řešením celého problému by mohlo být vytvoření parseru, který by umožňoval použít filtrační rozhraní `org.w3c.dom.ls.LSParserFilter`.

Pro analýzu struktury pak byl zaveden podbalíček `helpers` obsahující třídu `AbstractNodeAnalyzer` a její potomky `JsonNodeAnalyzer` a `XmlNodeAnalyzer`. Jádrem analýzy struktur obou formátů je rekurzivní algoritmus procházející strom do hloubky. V průběhu analýzy algoritmus slučuje na každé úrovni potomky aktuálního uzlu se stejnou strukturou do jednoho. Na závěr analýzy každé úrovně jsou vyřazeni potomci, kteří mají podmnožinu vlastností jiného potomka.

V aplikační vrstvě bylo dále na místo třídy `ATreeDataSelectionService` navržené v kapitole 2.9.2 a jejích podtříd zavedeno rozhraní `IQueryGenerator`. Vytvořeny byly dvě jeho implementace, a to `XPathQueryGenerator` a `JsonPathQueryGenerator`. Nicméně funkce je podobná jako u původně navržených tříd, tedy generovaní dotazů na základě výběru ze strukturálního pohledu. Implementované třídy a rozhraní byly umístěny do balíčku `dataSelection.utilities.queryGenerator` v rámci business vrstvy.

Třída `AnonymizationManager` z balíčku `filesystem.anonymization` byla refaktorizována na dvě samostatné. Jedná se o abstraktní `AbstractAnonymizationManager` a jejího potomka `AnonymizationManagerStructured`. Do první zmíněné třídy byla přesunuta logika společná s anonymizací stromových souborů. Na to vznikla také třída `AnonymizationManagerTreeStructured` zpracovávající konkrétní konfiguraci stromového dokumentu.

Posledním přídavkem aplikační vrstvy je třída `TreeDataSelectionService` z balíčku `dataSelection.service`, jenž slouží

do jisté míry jako model pro uživatelské rozhraní při nastavování anonymizace stromových dokumentů.

3.3 Prezentační vrstva

Refaktorizace balíčku `controllers.anonymization` byla provedena dle návrhu, zavedeny byly sub-balíčky `structured` a `treeStructured`, a také zde byl vytvořen kontroler `CommonController` reagující na tlačítko „Solved“. Do balíčku `treeStructured` jsou pak umístěny všechny kontrolery zpracovávající vstupy uživatele při nastavování anonymizačního modelu stromových dokumentů.

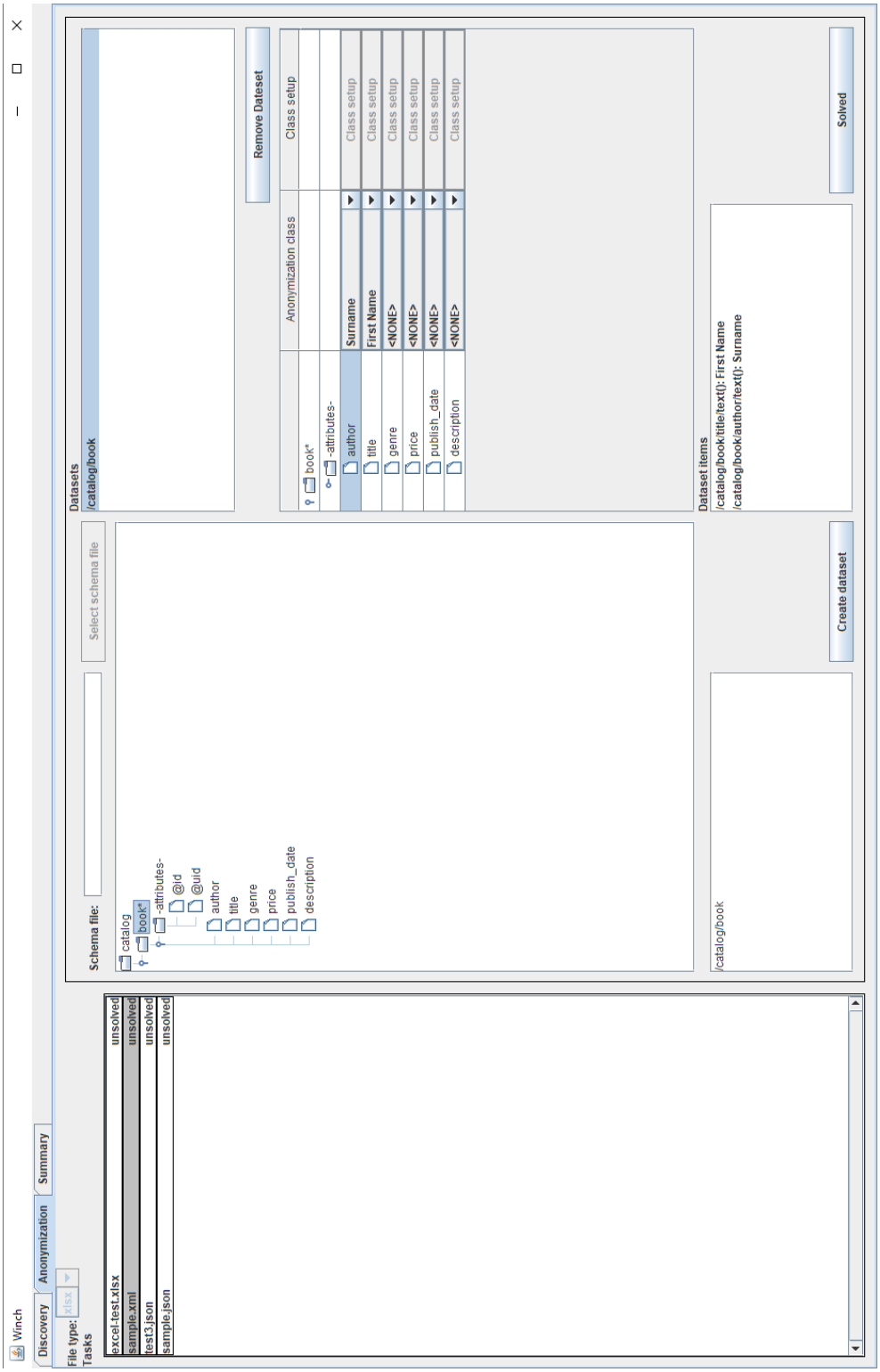
V balíčku `view.anonymization.treeStructured` jsou samotné vizuální komponenty. Uživatelské rozhraní využívá již zmíněnou knihovnu Swing. Pro zobrazení stromové struktury celého souboru byla použita komponenta JTree. K nastavení jednotlivých anonymizovatelných prvků je pak k dispozici komponenta Outline. Pro tento účel vznikla třída `ItemSetupOutline`, která Outline rozšiřuje. Všechny prvky pro nastavení modelu anonymizace jsou sdruženy v JPanelu třídy `TreeStructuredFilePanel`. Nalézají se zde i další komponenty Swing, jako jsou JList, JButton nebo JLabel, pokrývající ostatní prvky uživatelského rozhraní. Výsledná obrazovka je k vidění na obrázku 3.3.

Oproti návrhu přibyl v implementovaném rozhraní přehledový seznam položek pro vybraný dataset umístěný vpravo dole3.3. Naopak náhled na obsah celého souboru s filtračním polem pro odladění dotazu prozatím realizován nebyl. Nepřítomnost tohoto prvku však nebrání nastavení anonymizačního modelu.

3.3.1 Uživatelská příručka

V této podkapitole je krátce popsáno nastavení modelu pro stromové dokumenty.

1. Vybrat úkol (soubor) ze seznamu vlevo.
2. Nadefinovat dotaz určující množinu dat (dataset) v textovém poli dole uprostřed a potvrdit stisknutím tlačítka „Create dataset“. Výběrem ze stromového strukturálního pohledu umístěného nad textovým polem bude dotaz určující daný prvek do textového pole předvyplněn.
3. Vybrat dataset ze seznamu vpravo nahoře.
4. V tabulce pod seznamem datasetů je poté možné určit jednotlivé prvky k anonymizaci vybráním jejich anonymizační třídy v druhém sloupci tabulky. Výběrem „<NONE>“ lze určení zrušit – takto označený prvek nebude anonymizován.
5. Po zpracování souboru je možné tuto skutečnost potvrdit stisknutím tlačítka „Solved“



Obrázek 3.3: Vytvořené uživatelské rozhraní

3.4 Neimplementované funkce

Následující funkce nebyly prozatím implementovány.

- Podpora pro zobrazení schéma definujících souborů a jejich využití pro generování dotazu.
- Nastavení parametrů a dekorátorů anonymizačních tříd a jejich přiřazení anonymizovatelným prvkům.
- Podpora pro definici množiny souborů zástupnými symboly či určení celého adresáře pro anonymizaci.

Kapitola 4

Testování

Implementované části datové a aplikační vrstvy jsou pokryty jednotkovými testy. K tomuto účelu byla využita v projektu již zavedená knihovna `JUnit`. Testy povětšinou zkouší funkčnost jednotlivých tříd. Ale například pro testování zápisu a čtení anonymizačního modelu do a z konfiguračních souborů byly zavedeny testy zkoušející obě operace zároveň. Jedná se o testovací třídy `ModelFileSaverLoaderXmlTest` a `ModelFileSaverLoaderJsonTest`.

Na aplikaci bylo dále provedeno výkonové testování.

4.1 Testy výkonu aplikace

Jedním z omezení současné implementace je nutnost načtení celých souborů do paměti, což vyžadují použité knihovny na provádění dotazů na data. Aktuální realizace anonymizačního modelu vyžaduje analýzu struktury dotazem určeného datasetu. To dává omezení na maximální velikost zpracovaného dokumentu. V tomto směru platí násobnosti zjištěné v tabulkách 2.2 a 2.3 (pro implementaci byly použity knihovny `org.w3c` s využitím `javax.xml.parsers` a `Jackson`), jsou však jen orientační. Skutečná násobnost zvětšení souboru v paměti je dána „košatostí“ struktury, velikostmi textů, ale třeba také použitým formátováním souboru. Nelze tedy stanovit přesná omezení. V testovaných případech se násobné zvětšení pohybovalo okolo devíti. Dalšími omezujícími faktory jsou verze použitého Java virtuálního stroje (32 bit/64 bit), hostující operační systém a případně i velikost operační paměti.

Při samotném testování výkonu jednotlivých výkonnostně složitějších operacích byl zjištěn problém efektivnosti analýzy dokumentů XML. Z dosud nezjištěných příčin je analýza XML souborů řádově pomalejší než analýza struktur JSON, byť obě vycházejí ze stejného algoritmu. Pro testování výkonu XML byly použity 2 různé instance. V jednom případě byla použita kolekce s 100 000 elementy. Druhá instance poté obsahovala dvě kolekce po 50 000 záznamech. Měřením doby běhu analýz bylo zjištěno, že zpracování první instance trvá více než dvojnásobek času. Lze tedy soudit, že na dobu běhu má

Tabulka 4.1: Výkon XML

Doba běhu (s)	Analýza souboru	Analýza datasetu	Anonymizace
A (100k záznamů)	156	159	0,7
B (2×50k záznamů)	71	37	0,6

Tabulka 4.2: Výkon JSON

Doba běhu (s)	Analýza souboru	Analýza datasetu	Anonymizace
A (5M záznamů)	47	43	201
B (2×2M záznamů)	59	36	132

velký vliv maximální počet elementů uvnitř jiného elementu. Výsledky měření jsou zaznamenány v tabulce 4.1.

Pro otestování výkonnosti aplikace při zpracování dokumentů JSON byly použity podobně strukturované instance jako u XML, bylo však možné použít více záznamů. První instance obsahovala 5 milionů záznamů a druhá dva krát 2 miliony. Tabulka 4.2 poté zachycuje naměřené časy běhu jednotlivých operací.

Při testování anonymizace byla u obou formátů použita anonymizační funkce, která pouze generuje náhodné číslo. Běhové prostředí mělo následující parametry.

- Java 8 64bit
- Hostující OS – Windows 10 Education 64bit
- Procesor – Intel i5-9400F
- Operační paměť – 16 GB 2666 MHz

Závěr

Cílem této práce bylo vytvořit rozšíření nástroje Winch, které bude umožňovat anonymizaci osobních údajů ve strukturovaných dokumentech ve formátu XML a JSON. Součástí zadání byla také implementace uživatelského rozhraní, jenž usnadní definici anonymizačního modelu pro vybrané soubory. Návrh modelu byl také zahrnut do této práce.

Základní požadavky se podařilo splnit. Je možné provádět anonymizaci dat v obou požadovaných formátech. Pro nastavení anonymizace bylo vytvořeno uživatelské rozhraní, které je součástí modulu Winch pro anonymizaci na souborovém systému a umožňuje nastavení modelu určujícího anonymizaci. Prvkem finální implementace je také analýza struktury dokumentu a její přehledné zobrazení. Detekce struktury se taktéž stala nutnou součástí implementace poté, co byla zahrnuta jako součást popisu datasetu v rámci anonymizačního modelu. Bohužel, vzhledem k nalezenému problému s efektivitou analýzy formátu XML, je anonymizace pro tyto dokumenty mírně omezena. Nicméně odstranění tohoto problému by nemělo být příliš složité.

Jistým problémem také může být dvojité načítání souboru JSON při jeho anonymizaci. Možné řešení (detekce, kdy je vhodné načíst soubor jednou a kdy dvakrát) bylo diskutováno v kapitole 3.2. Obecně pak implementace anonymizace pro oba formáty vyžaduje načtení anonymizovaných souborů do paměti. V případech, kdy je soubor velmi velký a naopak operační paměť není dostatečně velká, to může vést i k nemožnosti anonymizace dotčeného dokumentu. Řešení, které by umožňovalo dotazování na data a zároveň by nevyžadovalo načtení souboru do paměti, se nepodařilo najít.

Na implementaci pak dále čeká podpora pro určení více souborů k anonymizaci pomocí zástupných symbolů, nastavení parametrů a dekorátorů a možnost zobrazit a využít schéma definující soubory.

Literatura

[1] Stanislav Lapitsky: *XMLEditorKit library as example of XML viewing in JEditorPane/JTextPane. [online].* [cit. 2022-03-06]. Dostupné z: http://java-sl.com/xml_editor_kit.html

[2] Tim Vaughan: *Basic graphical JSON object heirarchy browser [online].* [cit. 2022-03-06]. Dostupné z: https://github.com/tgvaughan/JsonBrowse

[3] The Apache Software Foundation: *Class Outline [online].* Dostupné z: https://bits.netbeans.org/dev/javadoc/org-netbeans-swing-outline/org/netbeans/swing/outline/Outline.html

[4] Svoboda, M.: Data Formats [online]. 2020, [cit. 2022-02-15]. Dostupné z: https://www.ksi.mff.cuni.cz/~svoboda/courses/201-MIE-PDB/lectures/MIEPDB16-Lecture-04-Formats.pdf

[5] JSON.org: *Introducing JSON [online].* [cit. 2022-02-16]. Dostupné z: https://www.json.org/json-en.html

[6] Ort, R.: Anonymizace osobních údajů v dokumentech [online]. 2020, [cit. 2022-02-15]. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/90341/F8-BP-2020-Ort-Radomir-thesis.pdf

[7] XML Viewer org.: *XML Viewer [online].* Dostupné z: https://www.xmlviewer.org/

[8] Microsoft Corporation: *MS Visual Studio [online].* Dostupné z: https://visualstudio.microsoft.com/

[9] Gabor Turi: *Online JSON Viewer [online].* Dostupné z: http://jsonviewer.stack.hu/

[10] Christian Grün: *BaseX [online].* Dostupné z: https://basex.org/

[11] Refsnes Data: *Introduction to XML [online]*. [cit. 2022-02-15]. Dostupné z: https://www.w3schools.com/xml/xml_whatis.asp

[12] Refsnes Data: *DTD - Attributes [online]*. [cit. 2022-02-27]. Dostupné z: https://www.w3schools.com/xml/xml_dtd_attributes.asp

[13] Refsnes Data: *DTD - Examples [online]*. [cit. 2022-02-27]. Dostupné z: https://www.w3schools.com/xml/xml_dtd_examples.asp

[14] Refsnes Data: *XSD Restrictions/Facets [online]*. [cit. 2022-02-28]. Dostupné z: https://www.w3schools.com/xml/schema_facets.asp

[15] Ecma International: *ECMA-262, 11th edition, June 2020 ECMAScript® 2020 Language Specification [online]*. [cit. 2022-03-01]. Dostupné z: https://262.ecma-international.org/11.0/

[16] JSON Schema: *Getting Started Step-By-Step [online]*. [cit. 2022-03-01]. Dostupné z: https://json-schema.org/learn/getting-started-step-by-step.html

[17] The Apache Software Foundation: *Apache NetBeans [online]*. Dostupné z: https://netbeans.apache.org/

[18] Svoboda, M.: XML Databases: XPath, XQuery [online]. 2020, [cit. 2022-03-08]. Dostupné z: https://www.ksi.mff.cuni.cz/~svoboda/courses/201-MIE-PDB/lectures/MIEPDB16-Lecture-05-XQuery.pdf

[19] Stack Exchange Inc: *Search xpath json [online]*. [cit. 2022-03-04]. Dostupné z: https://stackoverflow.com/questions/tagged/xpath+json

[20] Goessner, S.: JSONPath - XPath for JSON [online]. 2007, [cit. 2022-03-08]. Dostupné z: https://goessner.net/articles/JsonPath/

[21] the Apache Groovy project: *Processing XML [online]*. [cit. 2022-03-04]. Dostupné z: https://groovy-lang.org/processing-xml.html

[22] Stack Exchange Inc: *Search gpath [online]*. [cit. 2022-03-04]. Dostupné z: https://stackoverflow.com/questions/tagged/gpath

[23] Sean Leary: *JSON-java [online]*. Dostupné z: https://github.com/stleary/JSON-java

[24] Google LLC: *GSON [online]*. Dostupné z: https://github.com/google/gson

[25] FasterXML, LLC: *Jackson [online]*. Dostupné z: https://github.com/FasterXML/jackson

[26] Json-path: *Jayway JsonPath [online]*. Dostupné z: https://github.com/json-path/JsonPath

Příloha **A**

Seznam použitých zkratek

XML Extensible Markup Language

JSON JavaScript Object Notation

CASE Computer Aided Software Engineering

DTD Document Type Definition

XSD XML Schema Definition

DOM Document Object Model

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... instrukce ke spuštění aplikace
- `uzivatelska-prirucka.pdf` ..... příručka pro nastavení anonymizace tabulkových dokumentů
- `exe`
  - `disl-winch-filesystem-3.1.9.11-SNAPSHOT`
    - `bin` ..... adresář se spouštěcími skripty
    - `lib` ..... jar soubory knihoven a implementace
- `samples` ..... soubory k otestování aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Dolezal_Jakub_2022.pdf` ..... text práce ve formátu PDF