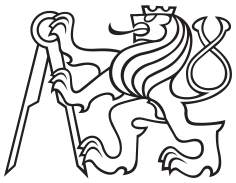
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Obousměrné generování API z modelu

Giorgi Gogatishvili

Vedoucí: Ing. David Kadleček, Ph.D.
Obor: Softwarové inženýrství a technologie
Zaměření: Business informatics
Květen 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Gogatishvili** Jméno: **Giorgi** Osobní číslo: **507287**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

Specializace: **Business informatics**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Obousměrné generování API z modelu

Název bakalářské práce anglicky:

Bidirectional generation of API from the model

Pokyny pro vypracování:

The first goal of the project is to propose and implement mechanisms for creating and managing models and subsequent generation of development artifacts in programming languages such as Java, Kotlin, Typescript and API definitions, e.g. in swagger format. The second main goal is to implement the opposite direction, thus generate model from API.
Modelling will be performed in Sparx Enterprise Architect and two
model types will be supported.
i) Class model that can be used to represent midtier domain model,
frontend model or persistence model
ii) API model that can be used to represent interfaces with operations
exposed by usually some service tier
We expect that the modelling approach will support for composability
of models and APIs from reusable attributes and attribute groups to
promote reusability and impact analysis.
The student will implement two generation pipelines on top of the
above models. The first will generate ready made developers' artifacts
such as swagger, java or kotlin classes (POPJOs) a and interface stubs
and skeletons.
The second pipeline will implement capability to reverse engineer
existing API to generate API model in Sparx Enterprise Architect.
Evaluation criteria:
1) Modelling perspective or methodology how to model class model and
API model in Sparx Enterprise Architect that can be later on used to
generate developers' artifacts
2) Example class model and API model and generated artifacts
3) Example swagger API that is imported and results in a creation of
API model in Sparx Enterprise Architect
Note: We expect that the student will utilize Swagger Open API
standards and tooling to reduce required development effort and
increase adherence to the existing standards.

Seznam doporučené literatury:

[0] Swagger supported by SmartBear, OpenAPI Specification, 2021
[1] Apache Avro, Specification, https://avro.apache.org/
[2] W3C, W3C XML Schema Definition Language (XSD)
[3] OMG, XMI, Dostupné z: https://www.omg.org/spec/XMI/2.1.1
[4] Masse, Mark. REST API design rulebook: designing consistent
RESTful web service interfaces. "O'Reilly Media, Inc.", 2011,

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Kadleček, Ph.D. Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **11.02.2024** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **21.09.2025**

Ing. David Kadleček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Poděkování

Rád bych poděkoval Ing. Davidu Kadlečkovi, Ph.D. za vedení této bakalářské práce. Dále bych chtěl poděkovat Ing. Martinu Mašatovi, který na projektu pracoval přede mnou a který byl ochotný radit s případnými technickými problémy. Dále děkuji Ing. Martinu Turynovi za pomoc s formálními náležitostmi a formátováním tohoto dokumentu. V neposlední řadě bych chtěl vyjádřit díky mé rodině a mým kamarádům, kteří se o moji práci zajímali a byli mi morální podporou.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a v souladu s Metodickým pokynem o dodržování etických principů pro vypracování závěrečných prací a že jsem uvedl všechny použité informační zdroje.

Praha 24. května 2024

. . . . . .

Abstrakt

Tento projekt navazuje na diplomovou práci Ing. Martina Mašaty, ve kterém byl vyvinut generátor schopný analyzovat diagramy v EA (Enterprise Architect). Tento generátor produkuje kódy pro Swagger, Avro schémata a Javu. Cílem aktuálního projektu je rozšířit generátor o nové funkcionality a moduly, konkrétně rozšíření funkčnosti pro již existující Swagger a Avro schémata. Dále bude projekt zaměřen na vývoj zcela nových modulů pro generování JSON schémat, XSD a XMI souborů.

Klíčová slova: Generování kódu, EA, OpenAPI, Avro schéma, JSON schéma, XSD, XMI

Vedoucí: Ing. David Kadleček, Ph.D.

Abstract

This project builds upon Ing. Martin Mašata's master's thesis, in which a generator capable of analyzing diagrams in EA (Enterprise Architect) was developed. This generator produces code for Swagger, Avro schemas, and Java. The current project aims to expand the generator with new functionalities and modules, specifically enhancing the functionality for existing Swagger and Avro schemas. Furthermore, the project will focus on developing entirely new modules for generating JSON schemas, XSD and XMI files.

Keywords: Code generation, EA, OpenAPI, Avro schema, JSON schema, XSD, XMI

Title translation: Bidirectional generation of API from the model

Obsah

Obrázky

Kapitola 1
Úvod

1.1 Motivace

Téma této práce vzniklo z potřeb reálného projektu MPSV (Ministerstvo práce a sociálních věcí). Na tomto projektu se klade velký důraz na návrh a dokumentaci API mezi různými systémy. Vznikla zde tedy potřeba pro nástroj, který by z navržené API, která je zakreslena v EA, automaticky vygeneroval soubory potřebné pro implementaci. Mezi tyto soubory patří OpenAPI, Avro schéma, JSON schéma nebo XSD. Dále by pomocí tohoto nástroje bylo možné vygenerovat diagramy pro již existující OpenAPI soubory. Zautomatizování této části vývoje softwaru zjednoduší práci programátorům, kteří se pak můžou věnovat implementaci složitějších nestandardních částí systémů.

1.2 Kontext

Na tomto projektu přede mnou pracoval Ing. Martin Mašata. Bylo to tématem jeho diplomové práce, v níž byl vyvinut generátor schopný analyzovat diagramy v EA (Enterprise Architect) a následně generovat kódy pro Swagger, Avro schémata a programovací jazyk Java. Tato diplomová práce položila pevný základ pro následné rozšíření funkcionalit generátoru. [5]

1.3 Cíl

Z aktuálních požadavků MPSV vyplynuly dva hlavní cíle.

- Prvním cílem projektu je rozšířit funkčnost již existujících modulů pro Swagger a Avro schémata. Tato rozšíření hlavně doplní chybějící části specifikací a přidají další možnosti pro práci s těmito formáty.
- Druhým klíčovým cílem je vyvinout zcela nové moduly pro generování JSON schémat, XSD a XMI souborů. JSON schéma a XSD se budou generovat z již zakreslených diagramů v EA. XMI soubory budou sloužit pro opačný proces. Tedy zdokumentování již existujícího API. Pomocí XMI souborů se automaticky vytvoří diagramy v EA na základě již

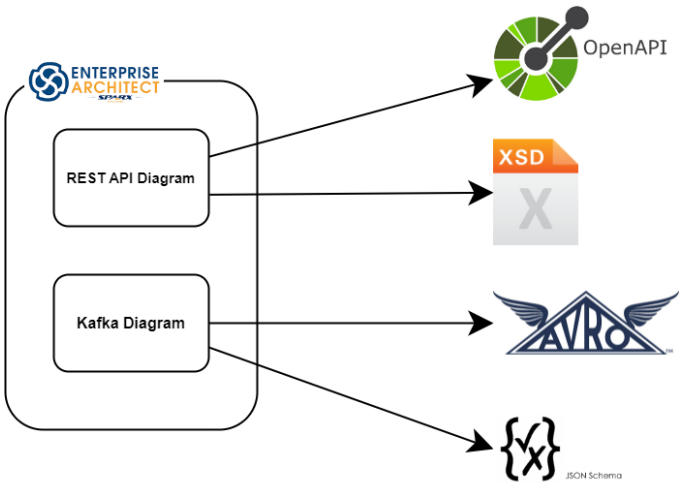
existujících souborů popisující API. Například podle dodaného Swagger dokumentu popisující REST API.

Kapitola 2
Analýza

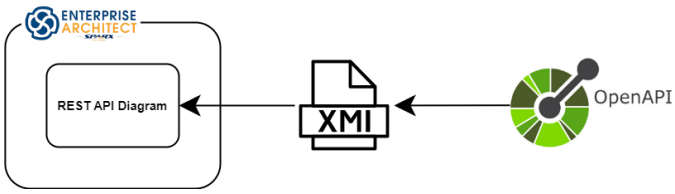
V této kapitole rozebereme hlavní funkčnost naší aplikace včetně funkčních a nefunkčních požadavků, které jsme si při analýze definovali. Dále se podrobněji podíváme na jednotlivé technologie a pojmy, jež hrají klíčovou roli v tomto projektu. Nakonec si krátce řekneme, jaký bude mít tato aplikace přínos v kontextu softwarového vývoje.

2.1 Hlavní funkčnost

V průběhu analýzy jsme si vytyčili hlavní funkcionalitu generátoru. Generátor bude schopen generovat OpenAPI a XSD soubory z diagramů týkajících se rozhraní REST API v nástroji Enterprise Architect. Dále bude generovat Avro a JSON schémata s ohledem na Kafka diagramy. Nakonec bude generátor schopen vytvářet XMI soubory z OpenAPI, které bude možné následně importovat do EA a tím vytvořit diagramy. Detailní rozbor jednotlivých technologií a funkcionalit je proveden v následujících kapitolách.



Obrázek 2.1: Generování OpenAPI, XSD, Avro a JSON schémat



Obrázek 2.2: Generování XMI

2.2 Požadavky

Při analýze jsme si vytyčili následující funkční a nefunkční požadavky. Tyto požadavky jsme formulovali podle následující šablony:
<FR/NFR s modulem a číslem požadavku> - <Název požadavku>. <Systém umožní co>

2.2.1 Funkční požadavky

Funkční požadavky jsme si rozdělili na dvě kategorie. Požadavky zabývající se rozšířením již existujících modulů a požadavky, které se týkají zcela nových modulů.

- Rozšíření existujícího OpenAPI modulu
  - **FR-OpenAPI-01 - Generování více souborů.** Systém umožní generovat více různých souborů pro různé diagramy místo jednoho velkého.
  - **FR-OpenAPI-02 - Psaní poznámek.** Systém umožní psát poznámky u každého elementu.
  - **FR-OpenAPI-03 - Formátování poznámek.** Systém umožní psát víceřádkovou poznámku s českými znaky.
  - **FR-OpenAPI-04 - Shrnutí.** Systém umožní napsat shrnutí koncového bodu.
  - **FR-OpenAPI-05 - Info objekt.** Systém umožní definovat info objekt OpenAPI souboru.
  - **FR-OpenAPI-06 - Pořadí vlastností.** Systém umožní definovat vlastní pořadí vlastností.
  - **FR-OpenAPI-07 - Kompozice.** Systém umožní definovat kompozice (allOf, anyOf, oneOf, not).
- Rozšíření existujícího Avro modulu
  - **FR-Avro-01 - Reference záznamů.** Systém umožní přidat reference na záznamy.
  - **FR-Avro-02 - Nepovinná pole.** Systém umožní přidat nepovinná pole.

- **FR-Avro-03 - Psaní poznámek.** Systém umožní přidat poznámky pro popis elementů.
- **FR-Avro-04 - Pořadí polí.** Systém umožní stanovit vlastní pořadí polí.
- **FR-Avro-05 - Jmenné prostory.** Systém umožní definovat vlastní jmenný prostor pro záznamy.
- **FR-Avro-06 - Doplnit datové typy.** Systém umožní nově používat i datové typy bytes a decimal.

- Implementace nového XSD modulu
  - **FR-XSD-01 - Generování XSD.** Systém bude generovat XSD soubor pro každý objekt posílaný v daném REST API diagramu.
  - **FR-XSD-02 - Datové struktury.** Systém ponechá stejnou datovou strukturu generovaných objektů, jako byla v OpenAPI souborech.
  - **FR-XSD-03 - Datové typy.** Systém při generování použije datové typy, které budou co nejvíce podobné těm z OpenAPI souborů.
- Implementace nového JSON schéma modulu
  - **FR-JSON-01 - Generování JSON schémat.** Systém bude generovat JSON schéma odpovídající každému Avro schématu.
  - **FR-JSON-02 - Datové struktury.** Systém ponechá stejnou datovou strukturu generovaných objektů, jako byla v Avro schématech.
  - **FR-JSON-03 - Datové typy.** Systém při generování použije datové typy, které budou co nejvíce podobné těm z Avro schémat.
  - **FR-JSON-04 - Kompozice.** Systém umožní definovat kompozice (allOf, anyOf, oneOf, not).
- Implementace nového XMI modulu
  - **FR-XMI-01 - Generování XMI.** Systém bude generovat XMI soubor pro každé vložené OpenAPI.
  - **FR-XMI-02 - Generování EA elementů.** Systém v XMI definuje EA elementy pro jednotlivé objekty, číselníky a vlastnosti z OpenAPI souboru.
  - **FR-XMI-03 - Generování EA diagramů.** Systém v XMI definuje EA diagramy odpovídající naší zvolené metodice pro zakreslení REST API.

2.2.2 Nefunkční požadavky

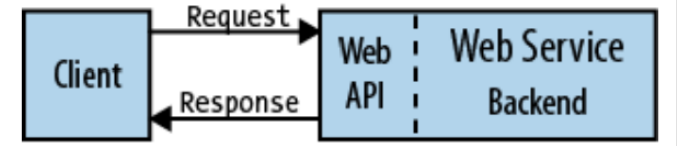
Nefunkční požadavky zůstaly z části stejné, jako byly v práci Ing. Martina Mašaty. [5] Jedná se o NFR-01 a NFR-02. Následují požadavky jsou zde uvedeny nově.

- **NFR-01 - Reportování chyb.** Systém uživateli zobrazí chyby včetně místa, kde přesně se vyskytly.
- **NFR-02 - Technická dokumentace.** Systém bude mít technickou dokumentaci pro instalaci a spuštění. (Viz příloha B.)
- **NFR-03 - Changelog.** Systém bude mít changelog popisující změny v každé nové verzi.
- **NFR-04 - Komentáře kódu.** Systém bude mít okomentovaný kód.
- **NFR-05 - Modularita kódu.** Systém bude vysoce modulární, aby bylo možné snadno upravovat jednotlivé části bez velkého dopadu na zbytek systému.
- **NFR-06 - Konzistentní výstup.** Systém bude zachovávat co nejvíce konzistentní výstup vygenerovaných souborů, aby při změnách bylo snadné porovnávat rozdílné verze dokumentů.

2.3 REST API

REST API neboli anglicky Representational State Transfer Application Programming Interface je architektonický styl webových služeb. Tyto webové služby vystavují ven svoje API, které je využíváno klientskými aplikacemi pro práci s daty. Jedním z hlavních pravidel, které by REST API mělo splňovat, je bezestavovost. To znamená, že si server neuchovává žádné informace o klientovi a veškerá potřebná data jsou obsažená v daném požadavku. Pro tuto komunikaci je využíván protokol http případně https. Metody pro práci s daty jsou GET, POST, PUT, PATCH a DELETE. [1] [6]

Diagram níže zobrazuje základní strukturu a komunikaci mezi webovou službou a klientem.



Obrázek 2.3: Web API, zdroj: [1]

Jedním z cílů tohoto projektu je vyvinout metodiku pro zdokumentování REST API za pomoci diagramů v EA. Dále pak poskytnout nástroje pro generování kusů kódu a pomocných artefaktů, které následně pomohou vývojářům při implementaci dané REST API. Hlavním takto vygenerovaným artefaktem bude OpenAPI soubor.

2.3.1 OpenAPI

REST API můžeme detailně popsat pomocí OpenAPI specifikace. Tato specifikace je velice podrobná a slouží vývojářům pro lepší porozumění API bez přímého přístupu do kódu dané webové služby. Dále ji dokážou využívat i různé nástroje pro zobrazení API či generování kódu pro různé programovací jazyky. [7] [8]

V tomto projektu je jedním z cílů rozšířit modul, který z REST API diagramů generuje OpenAPI soubory ve formátu YAML. Tento modul byl převzat od Ing. Martina Mašaty ve výborném stavu. Vzhledem k tomu není zapotřebí provádět žádné významné změny, pouze menší rozšíření, jak je podrobněji popsáno v kapitole 2.2.

2.4 XSD

XSD neboli XML Schema Definition je jazyk pro popis struktury a obsahu XML dokumentů. Samotný XSD soubor je taktéž ve formátu XML. Slouží k zajištění konzistence a validity dat v XML, což je nezbytné při výměně dat mezi různými systémy. [9] [10]

V rámci tohoto projektu budeme XSD soubory generovat ze stejných diagramů jako OpenAPI, tedy z REST API diagramů. Máme totiž XML soubory, které se posílají přes REST API, ale je potřeba je později validovat i v dalších systémech pomocí XSD. Využijeme ovšem pouze segmenty těchto diagramů, které specifikují datovou strukturu, nikoliv ty části, které se věnují koncovým bodům a metodám. Tyto aspekty se týkají výlučně jen OpenAPI.

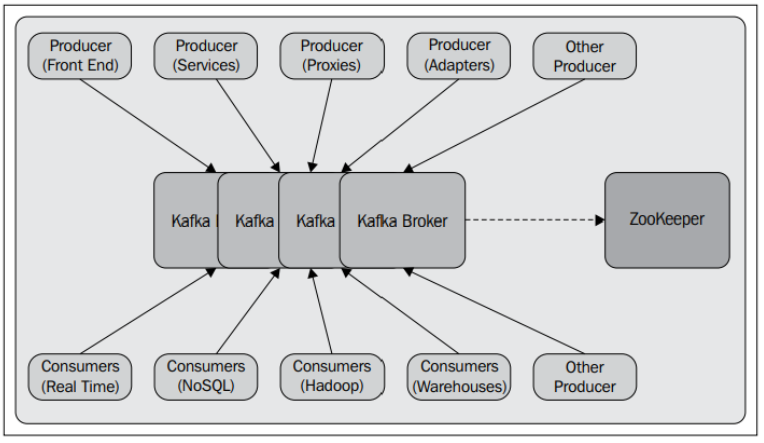
Jedná se o nový modul, pro jehož implementaci bude nutné vytvořit kompletní řešení od začátku.

2.5 Apache Kafka

Apache Kafka je open-source distribuovaný systém pro zpracování a publikování zpráv. Má následující charakteristiky:

- Persistence zpráv - Žádná data se nesmí ztratit, a to ani při chybách v systému.
- Velké množství zpráv - Kafka je schopna zpracovávat větší objem dat a umožnit potřebnou škálovatelnost.
- Distribuovaný - Data jsou rozdělena a následně distribuovaná do různých serverů.
- Podpora klientů - Kafka podporuje integraci mnoha různých klientů psaných například pomocí Javy, .NET nebo PHP.
- Zpracování v reálném čase – Zprávy by se měly zpracovávat okamžitě, když je to možné. [2] [11]

Zprávy produkuje Producer a následně přijímá Consumer. Pro přeposílání zpráv se používá Kafka Broker, který spravuje ZooKeeper. Na obrázku níže je znázorněna struktura Kafky. [2] [11]



Obrázek 2.4: Struktura Kafky, zdroj: [2]

2.5.1 Avro schéma

Avro schéma slouží pro popis struktury dat, která se posílají v Kafka zprávách. Avro soubory jsou ve formátu JSON s příponou AVSC. Tyto soubory slouží jak pro validaci dat, tak i pro různé nástroje generující kódy pro příslušné producery a consumery. [12]

Obdobně jako v případě modulu pro OpenAPI byl modul pro generování Avro schémat z velké části převzat od Ing. Martina Mašaty. Nicméně i zde jsou potřeba určitá rozšíření. Detailněji jsou tato rozšíření rozebrána v kapitole 2.2.

2.6 JSON schéma

JSON schéma slouží k popisu očekávané struktury a obsahu JSON dokumentů. JSON schéma je taktéž textový dokument v JSON formátu. [13] [14]

V tomto projektu plánujeme vytvářet JSON schémata ze stejných diagramů, které využíváme k vytváření Avro schémat, tedy z Kafka diagramů. Máme JSON soubory, které chceme validovat, a místo použití Avro schémat jsme se rozhodli využít JSON schémata, která nabízejí širší možnosti při definování struktury a omezení dat.

Stejně jako u XSD i zde se jedná o úplně nový modul, který bude třeba celý implementovat.

2.7 UML

UML neboli Unified Modeling Language je jazyk pro popis struktury a chování softwaru. UML definuje následující hlavní typy diagramů pro reprezentaci

systému:

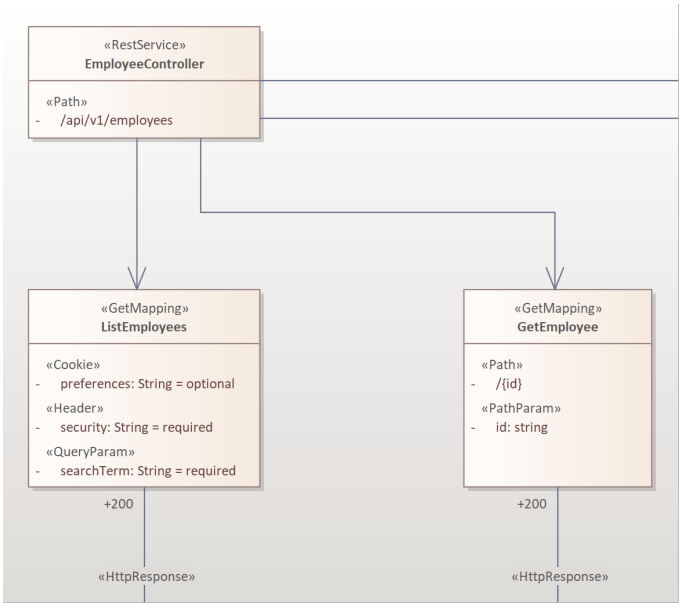
- Diagram tříd (Class Diagram)
- Diagram komponent (Component Diagram)
- Diagram nasazení (Deployment Diagram)
- Diagram aktivit (Activity Diagram)
- Diagram stavů (State Diagram)
- Sekvenční diagram (Sequence Diagram)
- Diagram případů užití (Use Case Diagram)
- ...

[15] [16]

2.7.1 Diagram tříd

Diagram tříd je jedním z nejdůležitějších diagramů v UML, neboť slouží k zachycení statické struktury softwaru. Skládá se ze tříd, atributů, operací, vazeb mezi třídami a kardinalitami. [15] [16]

Je to jediný typ diagramu, který budeme v tomto projektu potřebovat. Budeme totiž generovat soubory popisující strukturu dat, jež se posílají přes REST API nebo Kafku. Pro popis struktury dat je právě ideální diagram tříd.



Obrázek 2.5: Diagram tříd

2.7.2 EA

Enterprise Architect (EA) je nástroj pro modelování softwarové architektury. Umožňuje kolaboraci při tvorbě diagramů, podporuje různé jazyky jako například UML, SysML nebo ArchiMate a dále podporuje frameworky jako je TOGAF, Zachman, UPDM nebo UAF. [3]

V tomto projektu budeme EA používat pro vytváření diagramů tříd popisujících REST API nebo zprávy Kafce.

Obrázek 2.6: Enterprise Architect, zdroj: [3]

2.7.3 XMI

XML Metadata Interchange (XMI) je standard pro výměnu metadat ve formátu XML. Má více užití a v kontextu našeho projektu je nejdůležitější to, že je možné ho využít i pro popis modelů v jazyce UML, tedy pro popis toho, jak vypadají diagramy tříd, aktivit nebo případů užití. Umožňuje přenést UML modely vytvořené v jednom nástroji do jiného nástroje, a to za předpokladu, že oba nástroje podporují tento univerzální standard XMI. [17] [18]

V rámci tohoto projektu bude XMI sloužit k popisu diagramů tříd. Využijeme ho v případě, kdy už máme k dispozici existující rozhraní REST API s odpovídající specifikací OpenAPI, a máme zájem o vizualizaci tohoto rozhraní pomocí diagramů v nástroji EA. Aby tedy člověk nemusel manuálně tvořit diagramy v EA, generátor podle dodané specifikace v OpenAPI vygeneruje XMI soubor. Tento soubor pak bude možné importovat do EA, což usnadní proces dokumentace.

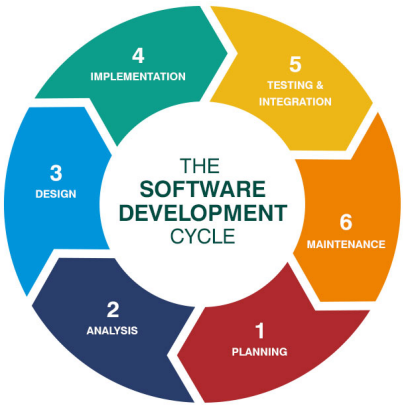
Pro použití souborů XMI jsme se rozhodli z několika důvodů. Jedním z nich je, že generátor nemusí vždy disponovat oprávněním k přímému zápisu do repozitáře EA. Takto se však využije oprávnění uživatele, který XMI soubor bude do EA importovat manuálně. Dalším důvodem je, že repozitář EA muže být realizován ve více podobách, jako například databáze nebo lokální soubor. Provedení přímých změn v modelu by vyžadovalo specifické implementace pro jednotlivé typy repozitářů, což by bylo náročné a méně univerzální. V neposlední řadě je tato metoda kompatibilní i s verzováním modelů v EA.

Modul pro generování XMI budeme implementovat od nuly. Jedná se o nejnáročnější část projektu. Největšími překážkami budou velký rozsah a nedostatečná dokumentace XMI, kterou používá EA. Je zde ale největší potenciál pro budoucí rozvoj a případné využití v rámci jiných projektů.

2.8 Vývoj softwaru

Vývoj softwaru je komplikovaný proces. Existuje mnoho různých modelů a metodik jako například vodopád nebo agile. Většina těchto metodik má však několik podobných fází:

- Plánování - Stanovení cílů a plánování rozpočtu, času a rozsahu.
- Analýza - Sběr požadavků od stakeholderů. Identifikace klíčových vlastností systému a okolí.
- Design - Podrobný technický návrh architektury.
- Implementace - Programování a tvorba systému.
- Testování - Testování, zda vše funguje správně a zda jsou splněny všechny požadavky.
- Údržba - Technická podpora a opravy chyb. [19]



Obrázek 2.7: Fáze vývoje softwaru, zdroj: [4]

Generátor, který bude výstupem tohoto projektu, se bude využívat v období mezi fázemi návrhu a implementace. Jeho funkčnost bude spočívat v analýze diagramů vzniklých během návrhu a následném automatickém vytvoření nezbytných souborů pro implementaci. Tímto způsobem bude doba potřebná pro počáteční etapu implementace zkrácena. Tento nástroj umožní vývojářům vyhnout se opakujícím se činnostem a soustředit se na náročnější unikátní aspekty systému.

Kapitola 3
Návrh

V této kapitole nejprve představíme zvolené technologie, které budou použity v našem projektu. Následně definujeme konkrétní metodiku pro tvorbu diagramů v EA. Dále rozebereme hlavní procesy a architekturu aplikace, což nám umožní získat komplexní přehled o tom, jak budou jednotlivé části aplikace spolu interagovat a jakým způsobem budou strukturovány. Na závěr této kapitoly se budeme zabývat dalšími možnostmi práce se soubory, které budou výstupem naší aplikace.

3.1 Technologie

Při volbě technologií jsme vycházeli z již existujícího projektu Ing. Martina Mašaty. Zvolené technologie se ukázaly být vhodné pro implementaci požadavků a nebylo tedy třeba je měnit za jiné. Několik z nich jsme však povýšili na novější verzi. Jako novou technologii zde vidíme *Swagger Parser*, který používáme při procesu generování XMI z existujícího OpenAPI souboru. Nakonec jsme zde upřesnili i verze *Enterprise Architecta* a příslušné databáze.

Technologie	**Verze**
Java	17.0.2
Maven	3.9.3
Spring Boot	3.1.1
Lombok	1.18.28
Freemarker	2.3.32
Swagger Parser	2.1.16
Enterprise Architect	15.2
MySQL	8.0.28

Tabulka 3.1: Verze zvolených technologií

3.2 Metodika zakreslování diagramů

V této kapitole je popsána metodika, kterou jsme zvolili pro zakreslování diagramů v EA. Vycházeli jsme z dříve stanovených pravidel a požadavků od Ministerstva práce a sociálních věcí. Dále jsme se snažili zajistit, aby naše zvolená metodologie byla co nejvíce konzistentní s tou, kterou Ing. Martin Mašata využíval ve svém projektu.

3.2.1 Metodika REST API diagramů

K popisu REST API používáme REST API diagramy, podle nichž pak generátor vytváří OpenAPI soubory ve formátu YAML. Při modelování těchto diagramů je nezbytné dodržovat pravidla, která jsou podrobně popsána v této kapitole.

Pojmy

- třída - Neboli anglicky *class* je entita v EA obsahující atributy.
- číselník - Neboli anglicky *enumeration* (zkráceně *enum*) je entita v EA, ve které můžeme definovat výčet nějakých předem stanovených hodnot.
- tag - Značka, která podrobněji popisuje nějaký element.
- objekt - Datový objekt ve výsledném OpenAPI souboru, který obsahuje vlastnosti.
- vlastnosti - Neboli anglicky *properties* jsou data nějakého objektu.
- koncový bod - Neboli anglicky *endpoint* je místo se specifickou URL adresou, kam chodí HTTP požadavky.
- kontrolér - Neboli anglicky *controller* je část aplikace, která zpracovává požadavky na koncových bodech.
- swagger - Oficiálně se jedná o nástroj pro práci s OpenAPI specifikací a dříve se tak i říkalo starší verzi specifikace, proto se často tyto pojmy zaměňují. V tomto kontextu se tím většinou myslí vygenerovaný soubor ve formátu YAML.

Config

Nastavení Swagger generátoru má svojí část v souboru *config.json*. Níže je uvedena ukázka:

```
"swagger": {
    "title": "Vlastni nazev API",
    "subfolder": "rest_api/openapi",
    "separateFIles": true
}
```

- *title* - Název swaggeru.
- *subfolder* - Podsložka, do které se budou generovat výsledné swaggery.
- *separateFiles* - Určuje, zda se má pro každý kontrolér vygenerovat samostatný Swagger nebo jeden velký Swagger, kde budou spojené všechny kontroléry.

Entity v EA

V této kapitole vidíme seznam entit a jejich vlastností, které používáme v REST API diagramech.

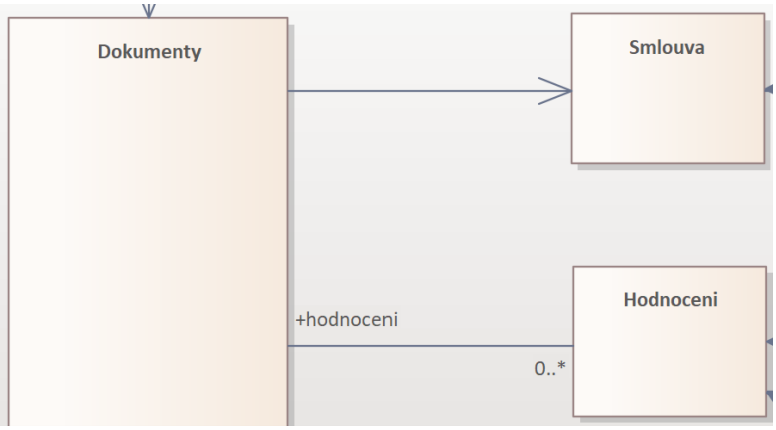
- Třída
  - Účel tříd se mění podle vybraného stereotypu. Viz kapitola *Stereotypy v EA*.
  - Pokud třída nemá žádný stereotyp, pak se jedná jen o prostý objekt.
  - Pokud je v diagramu více objektů se stejným názvem, pak se k těmto názvům přidávají ve výsledném swaggeru na konci indexy 2, 3, 4 ...
- Číselník
  - Na rozdíl od Kafka diagramu se zde číselník propojuje s objektem jen pomocí přímé vazby. Už není možné použít název číselníku v typu atributu.

Dále platí, že jména entit a hodnoty číselníků musí být bez diakritiky a mezer. Entitám i jejich atributům lze přidat poznámky, jež se propíšou do *description* ve vygenerovaném swaggeru.

Vazby v EA

Stanovili jsme metodiku pro zakreslování vazeb mezi třídami a číselníky v EA. Používáme zde následující vazby a jejich vlastnosti:

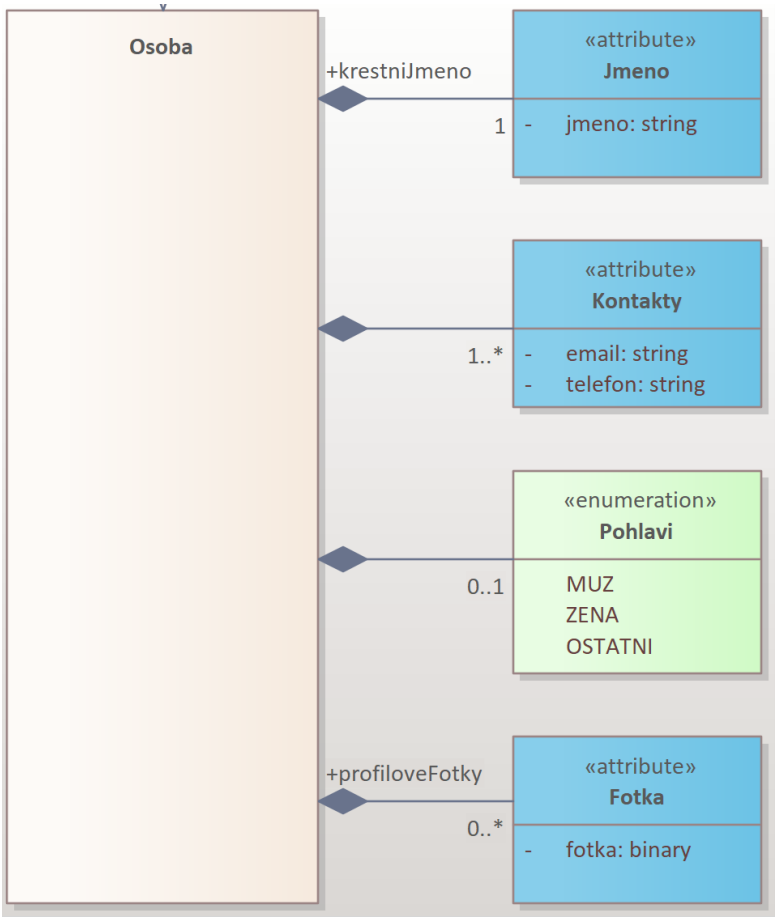
- Asociace
  - Vazba mezi třídami.
  - Vždy se nějak musí určit směr, což chrání proti zacyklení. Jsou 2 způsoby: šipkou, pojmenováním.



Obrázek 3.1: Asociace v REST API diagramu

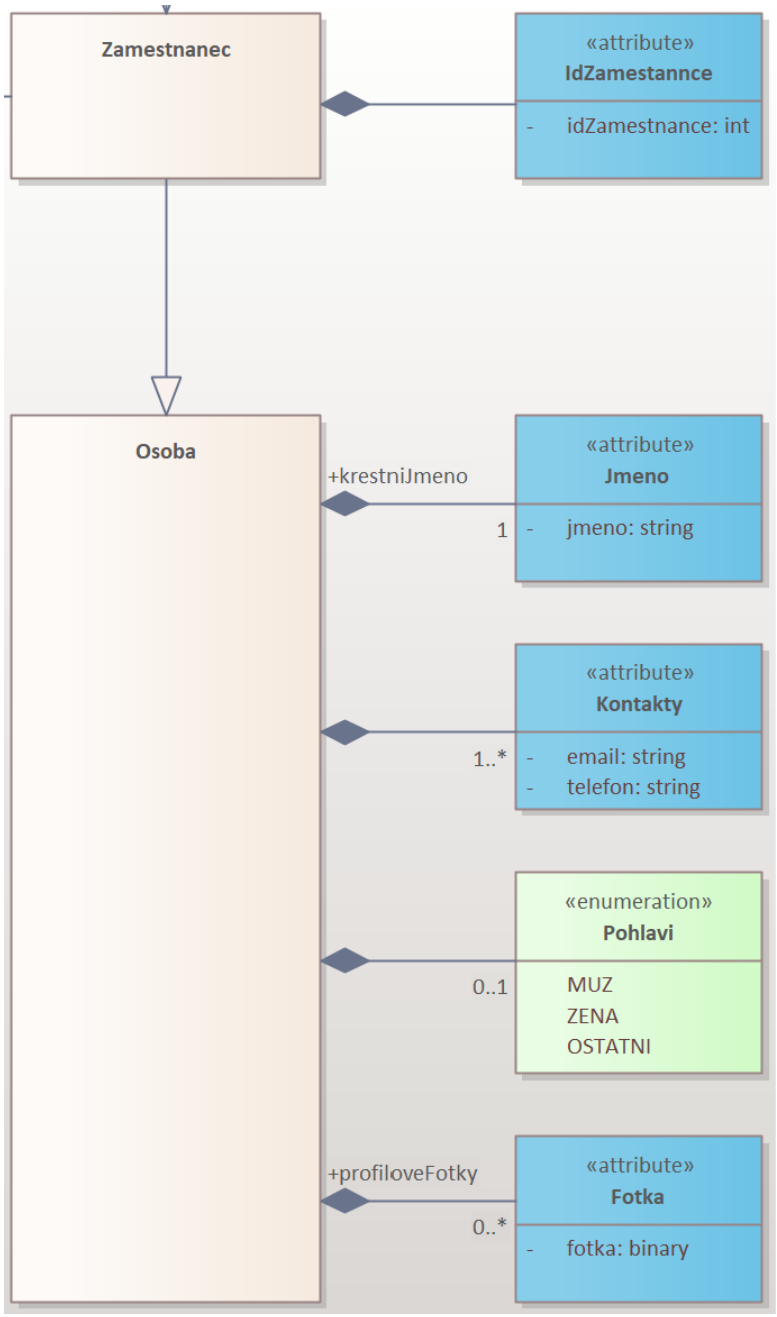
  - Pokud směr nejde určit nebo je šipka v opačném směru, pak se třída při mapování ignoruje.

- Agregace / Kompozice
  - Vazba mezi objektem a atributem, nebo skupinou atributů nebo číselníkem.



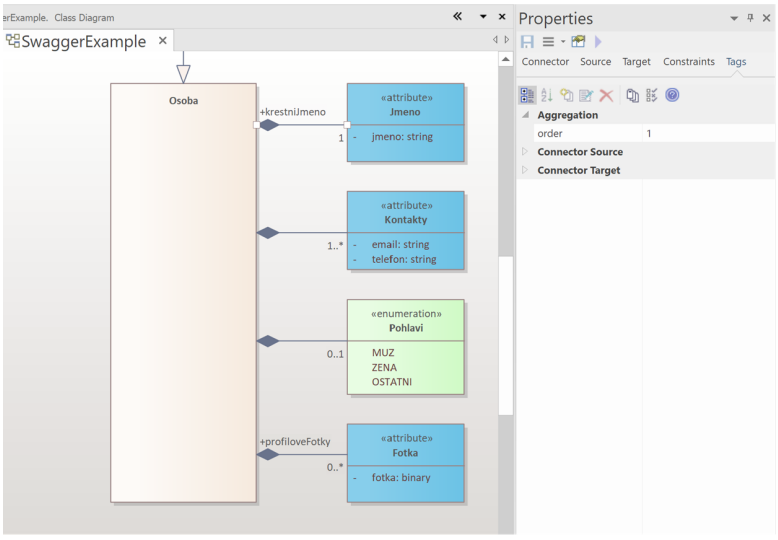
Obrázek 3.2: Kompozice v REST API diagramu

- Dědičnost
  - Vazba mezi třídami.
  - Potomek přebírá vlastnosti rodiče.



Obrázek 3.3: Dědičnost v REST API diagramu

- Kardinality
  - Default / Nezadán - 1 - povinný
  - 1 - povinný
  - 1..* - povinné pole
  - 0..1 - nepovinný
  - 0..* - nepovinné pole
- Pořadí vlastností
  - Na vazbě je možné definovat vlastní pořadí vlastností pomocí tagu *order*.
  - *Order* se definuje na vazbě, ne na atributu, protože atribut je možné opakovaně použít v jiných diagramech, kde se pořadí může lišit.
  - Nižší čísla jsou první v pořadí. Nedefinovaná čísla jsou na konci.
  - Pokud mají dvě vazby stejný order, pak se řadí sekundárně podle abecedy jména vlastnosti.
  - Příklad pořadí:
    1. order: 1
    2. order: 2, name: A
    3. order: 2, name: B
    4. order: null, name: A
    5. order: null, name: B

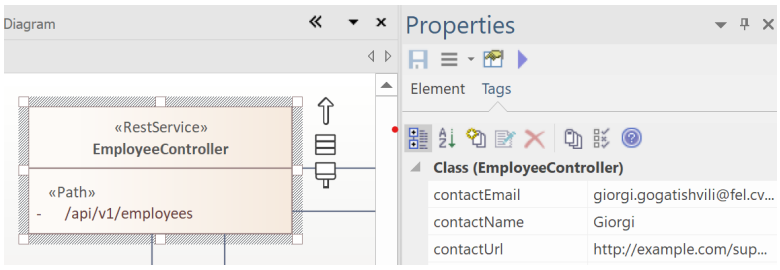


Obrázek 3.4: Tag order na vazbě

Stereotypy v EA

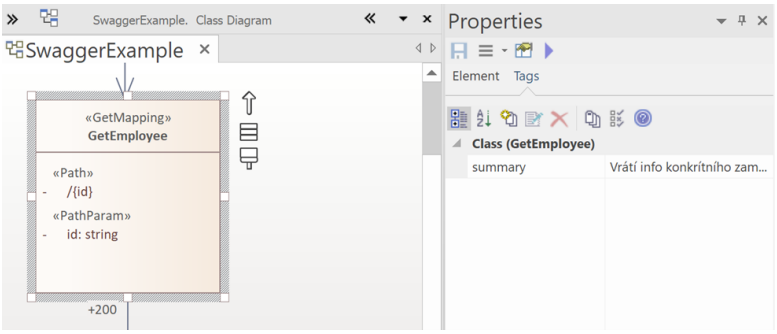
V REST API diagramech jsme si stanovili stereotypy pro následující elementy v EA:

- Třídám v EA přiřazujeme následující stereotypy:
  - *RestService*
    - Třída s tímto stereotypem je začátkem každého REST API diagramu.
    - Jsou na ni navázané jednotlivé koncové body.
    - Název této třídy se přidá jako *tag* do každého koncového bodu.
    - Může zde být definovaný atribut se stereotypem *Path*.
    - Mohou zde být definované tagy pro *contact*. Mělo by se jednat o člověka, který vytvářel tento diagram. Všechny tyto tagy jsou nepovinné.
      - *contactEmail*
      - *contactName*
      - *contactUrl*



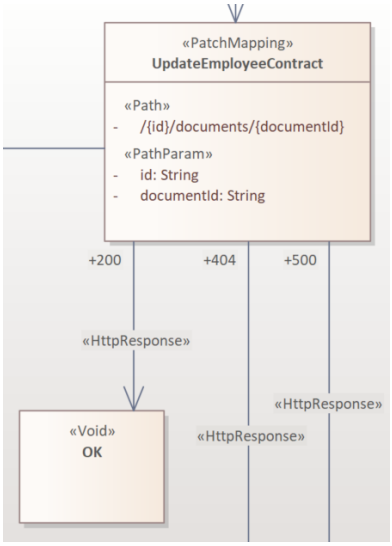
Obrázek 3.5: Třída se stereotypem RestService

  - *GetMapping, PostMapping, PutMapping, PatchMapping, DeleteMapping*
    - Třídy s těmito stereotypy označují jednotlivé koncové body a jsou přímo navázané na *RestService*.
    - Mohou zde být definovány atributy se stereotypy *Path, PathParam, QueryParam, Header, Cookie*.
    - Může zde být definovaný tag *summary*. Tento tag je nepovinný.



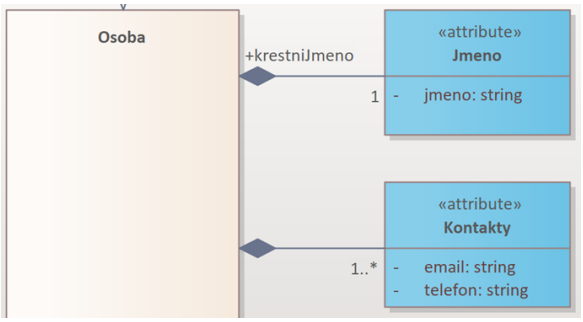
Obrázek 3.6: Třída se stereotypem GetMapping

- *Void*
  - Třída s tímto stereotypem slouží k definování odpovědi, která nemá žádné tělo.



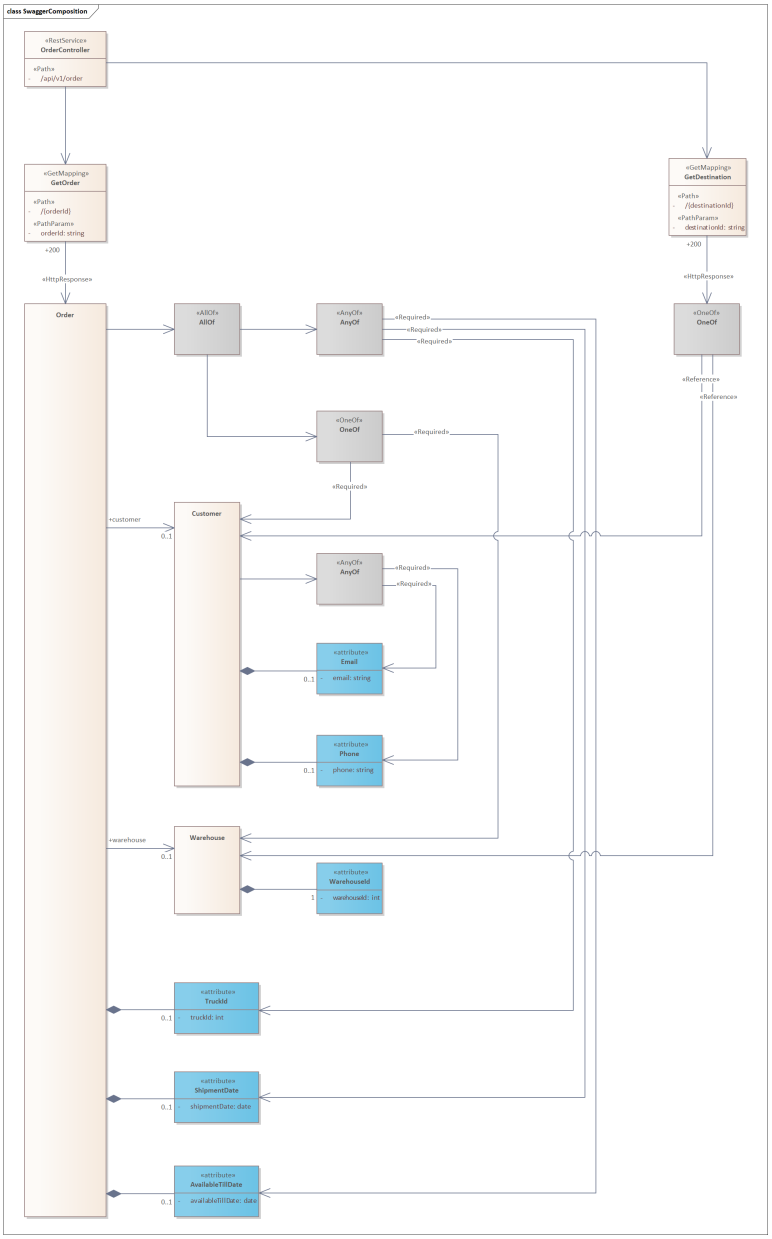
Obrázek 3.7: Třída se stereotypem Void

- *attribute*
  - Třída s tímto stereotypem obsahuje atributy nějakých datových typů.
  - V případě, že je zde definován jen jeden atribut, pak se název vlastnosti bere primárně z vazby. Pokud vazba není pojmenována, použije se název atributu.
  - V případě, že je zde více atributů, pak se názvy vlastností berou vždy z názvů atributů. Na název vazby se nepřihlíží. (Důvodem je, že by pak více atributů mělo stejný název a to nelze.)



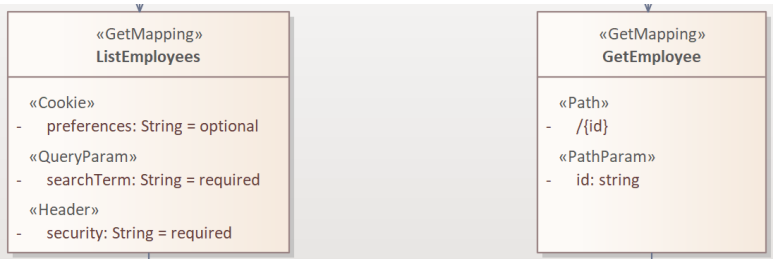
Obrázek 3.8: Třída se stereotypem attribute

- *AllOf, AnyOf, OneOf, Not*
  - Třídy s těmito stereotypy se používají pro definici kompozice.
  - Může být jak přímo na koncovém boudu, tak i pod nějakým jiným objektem.
  - Je možné je mezi sebou kombinovat.
  - V kompozici je možné specifikovat schéma 2 způsoby:
    - Jednoduché schéma, které definuje, které vlastnosti jsou povinné. V tomto případě je konektor na danou vlastnost se stereotypem *Required*. Na těchto konektorech lze definovat i tag *group*, jak se mají uskupit povinné vlastnosti. (Viz sekce Konektor níže.)
    - Reference na složitější objekt. Zde je konektor na daný objekt se stereotypem *Reference*.



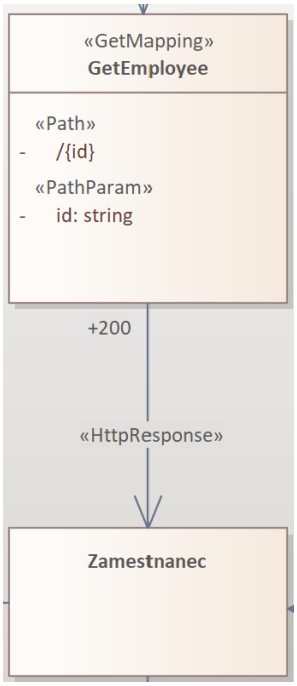
Obrázek 3.9: Ukázka kompozice v REST API diagramu

- Atributům tříd v EA přiřazujeme následující stereotypy:
  - *Path*
    - Může být definovaný na *RestService* i na koncových bodech.
    - Musí být definovaný alespoň na jednom místě.
    - Pokud je definovaný na více místech, pak se spojují a skládají za sebe.
  - *PathParam*
    - Musí odpovídat názvu parametru v cestě.
    - Je vždy povinný.
    - Lze použít více najednou.
  - *QueryParam*
    - V *Inital Value* atributu lze nastavit, zda je povinný. (*required* / *optional*). Default je *optional*.
    - Lze použít více najednou.
  - *Header*
    - V *Inital Value* atributu lze nastavit, zda je povinný. (*required* / *optional*). Default je *optional*.
    - Lze použít více najednou.
  - *Cookie*
    - V *Inital Value* atributu lze nastavit, zda je povinný. (*required* / *optional*). Default je *optional*.
    - Lze použít více najednou.



Obrázek 3.10: Atributy s různými stereotypy

- Konektorům v EA přiřazujeme následující stereotypy:
  - *HttpRequest*
    - Konektor s tímto stereotypem spojuje koncový bod s objektem reprezentujícím tělo HTTP požadavku.
    - Neměl by být více než jeden objekt požadavku na jeden koncový bod.
    - Konektor není třeba pojmenovávat. Kódy patří jen k objektům odpovědí.
    - Metoda *GET* nemá tělo dotazu, proto tento konektor ani jeho objekt požadavku nejsou pro *GetMapping* potřeba.
  - *HttpResponse*
    - Konektor s tímto stereotypem spojuje koncový bod s objektem reprezentujícím tělo HTTP odpovědi.
    - Každý koncový bod by měl mít alespoň jeden objekt odpovědí.
    - Název tohoto konektoru určuje kód odpovědi.



Obrázek 3.11: Konektor se stereotypem HttpResponse

Datové typy

Tabulka níže představuje seznam datových typů, kterých mohou nabývat atributy na třídě se stereotypem *attribute*. (Nerozlišují se velká a malá písmena.)

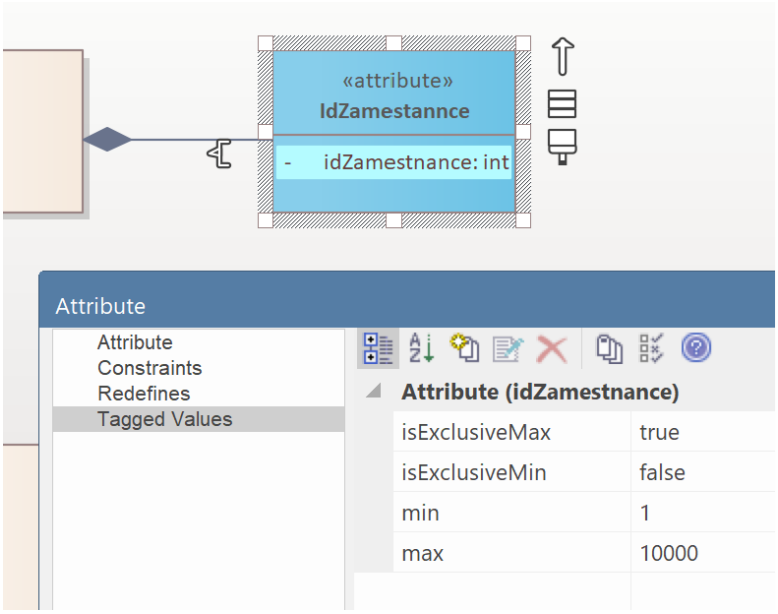
Typy v EA	Typ v OpenAPI	Formát v OpenAPI
string	string	
int, integer	integer	int32
long	integer	int64
float	number	float
double	number	double
date	string	date
datetime	string	date-time
boolean, bool	boolean	
bigdecimal	string	decimal
uuid, id	string	uuid
email, mail	string	email
password, pass, passwd	string	password
uri, url	string	uri
byte	string	byte
binary, bin	string	binary

Tabulka 3.2: Datové typy atributů v REST API diagramech a jejich příslušné datové typy v OpenAPI specifikaci

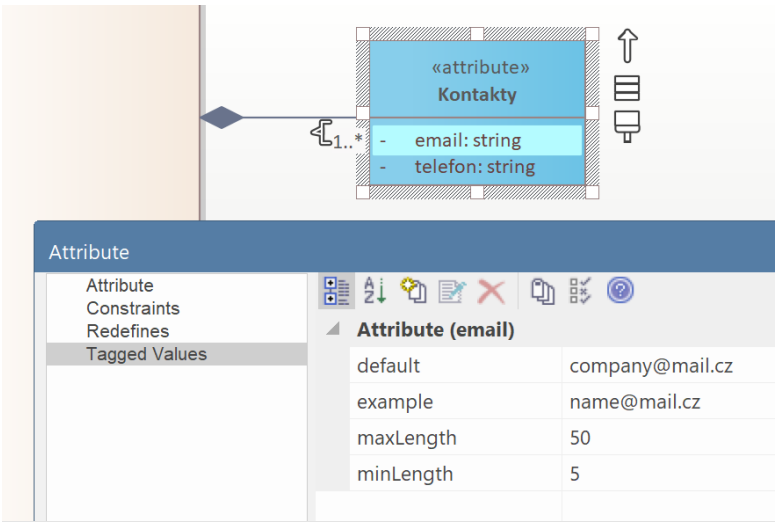
Omezení

U atributů lze definovat následující omezení pomocí tagů v EA:

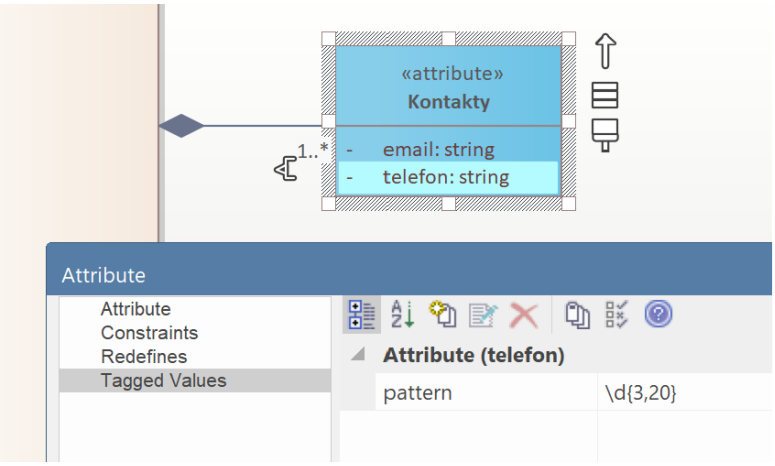
- *min* - minimum čísla
- *isExclusiveMin* - zda se má vyloučit *min* čísla z intervalu [*true*/*false*]
- *max* - maximum čísla
- *isExclusiveMax* - zda se má vyloučit *max* čísla z intervalu [*true*/*false*]
- *minLength* - minimální délka textu
- *maxLength* - maximální délka textu
- *example* - ukázková hodnota
- *default* - defaultní hodnota
- *pattern* - vzor textu
  - Pro účely generování XSD souborů je lepší nepoužívat oddělovač (/), začátek (^) a konec ($).



Obrázek 3.12: Ukázka REST API omezení 1



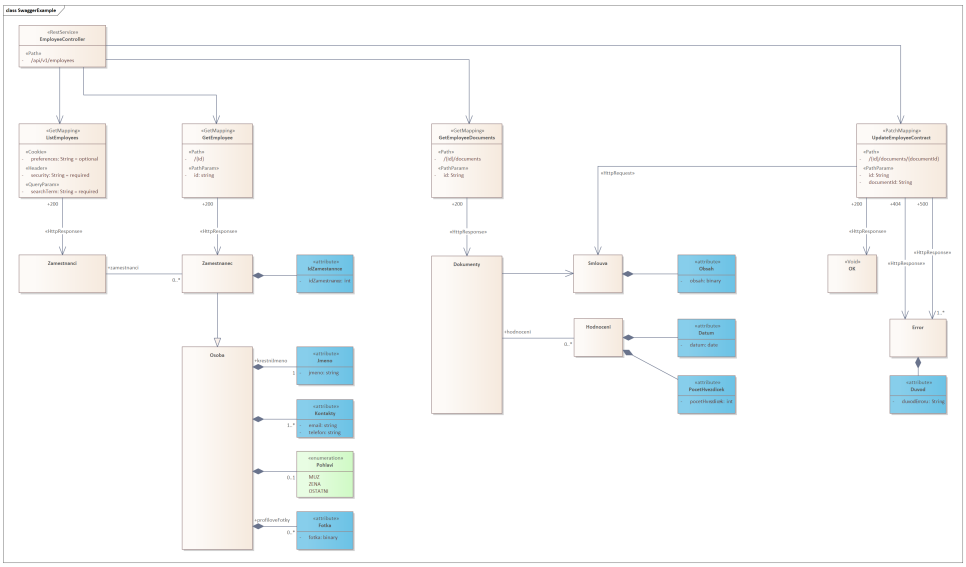
Obrázek 3.13: Ukázka REST API omezení 2



Obrázek 3.14: Ukázka REST API omezení 3

Ukázka celého diagramu

Zde je ukázka celého REST API diagramu v EA.



Obrázek 3.15: REST API diagram

3.2.2 XSD modul

V této kapitole je popsána funkcionalita generování XSD souborů pro REST API diagramy.

Config

Nastavení XSD modulu má svojí část v souboru *config.json*. Níže je uvedena ukázka:

```
"xsd": {
    "subfolder": "rest_api/xsd"
}
```

- *subfolder* - Podsložka, do níž se budou generovat výsledné XSD soubory.

Popis funkcionality

Při generování REST API se pro každý *HttpRequest* nebo *HttpResponse* vygeneruje i příslušný XSD soubor. Z REST API diagramů se tedy pro potřeby XSD ignorují entity *RestService, GetMapping, PostMapping, PutMapping, PatchMapping, DeleteMapping.*

Pro XSD není potřeba žádná rozdílná metodika modelování diagramů v EA.

3.2.3 Metodika Kafka diagramů

Zprávy, které se posílají pomocí nástroje Apache Kafka, je možné popsat Avro specifikací. Právě tyto Avro soubory vytváří náš generátor podle námi definovaných diagramům v EA. Těmto diagramům říkáme Kafka diagramy a při jejich modelování je třeba dodržovat pravidla popsaná v této kapitole.

Pojmy

- třída - Neboli anglicky *class* je entita v EA obsahující atributy.
- číselník - Neboli anglicky *enumeration* (zkráceně *enum*) je entita v EA, ve které můžeme definovat výčet nějakých předem stanovených hodnot.
- tag - Značka, která podrobněji popisuje nějaký element v EA.
- záznam - Neboli anglicky *record* je datový objekt ve výsledném Avro schématu, který obsahuje pole.
- pole - Neboli anglicky *fields* jsou data nějakého záznamu. Mohou to být jiné vnořené záznamy, číselníky nebo datové typy jako například *string*, *int* ...

Config

Nastavení Avro generátoru má svojí část v souboru *config.json*. Níže je uvedena ukázka:

```
"avro": {
    "baseNamespace": "cz.cvut.fel",
    "projectSubNamespace": "gg",
    "enumSubNamespace": enums,
    "subfolder": "kafka/avro"
}
```

- *baseNamespace* - Základní jmenný prostor pro všechny záznamy a číselníky, který se dále rozšiřuje o konkrétní jmenný prostor záznamu či číselníku. Př.: Výsledný kompletní jmenný prostor podle výše uvedené konfigurace bude pro záznam *cz.cvut.fel.gg* a pro číselník *cz.cvut.fel.enums*.
- *projectSubNamespace* - Upřesňuje jmenný prostor pro záznamy. Je možné ho v diagramu přepsat pro konkrétní záznam pomocí atributu se stereotypem *Namespace*.
- *enumSubNamespace* - Upřesňuje jmenný prostor číselníků.
- *subfolder* - Podsložka, do které se budou generovat výsledná Avro schémata.

Poznámky

Pro každý diagram se vygeneruje samostatné Avro schéma s názvem počáteční třídy, tedy třídy se stereotypem *KafkaMessage*.

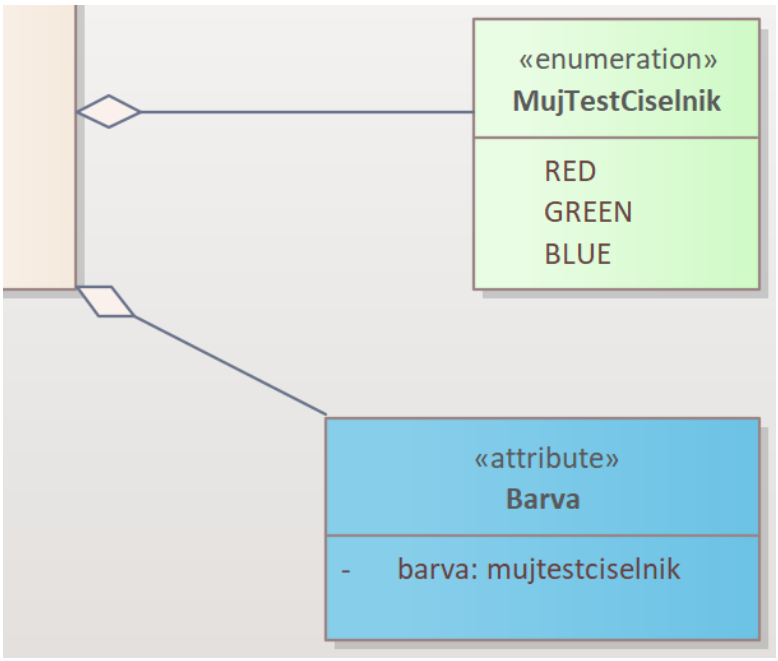
Obecný přístup ke špatnému diagramu je, že se špatné části ignorují nebo nahradí defaultními hodnotami tak, aby se vždy vygenerovalo validní Avro schéma. Na špatné části je uživatel upozorněn v logu pomocí warningů.

Entity v EA

V této kapitole vidíme seznam entit a jejich vlastností, které používáme v Kafka diagramech.

- Třída
  - Počátkem každého Kafka diagramu je třída se stereotypem *KafkaMessage*.
  - Třída se stereotypem *attribute* reprezentuje základní datové typy, logické datové typy a číselníky.
  - Třída, která nemá stereotyp *attribute*, reprezentuje záznam. Záznam, který nemá žádné atributy, se ignoruje a nemapuje.
  - V jednom diagramu nesmí být více tříd se stejným názvem.

- Číselník
  - Obsahuje výčet nějakých předem stanovených hodnot.
  - Jsou dva způsoby, jak navázat číselník na záznam. Přímo konektorem nebo typem atributu. Primárně se doporučuje konektor. Typ atributu je stále podporován kvůli zpětné kompatibilitě, aby fungovaly i starší diagramy.



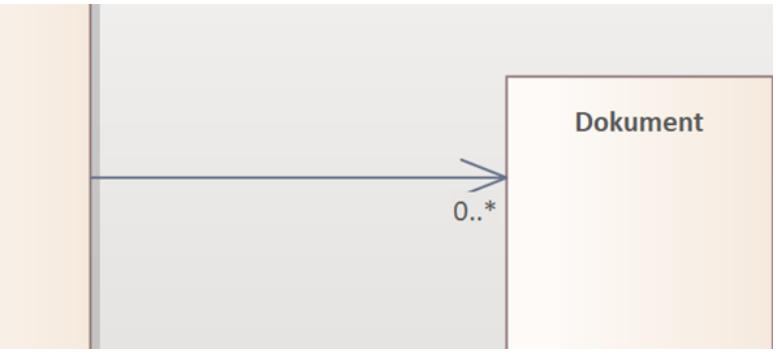
Obrázek 3.16: Způsobů provázání záznamů a číselníků

Dále platí, že jména entit a hodnoty číselníků musí být bez diakritiky a mezer. Entitám i jejich atributům lze přidat poznámky, které se propíšou do vlastnosti *doc* ve vygenerovaném Avro schématu.

Vazby v EA

Stanovili jsme metodiku pro zakreslování vazeb mezi třídami a číselníky v EA. Používáme zde následující vazby a jejich vlastnosti:

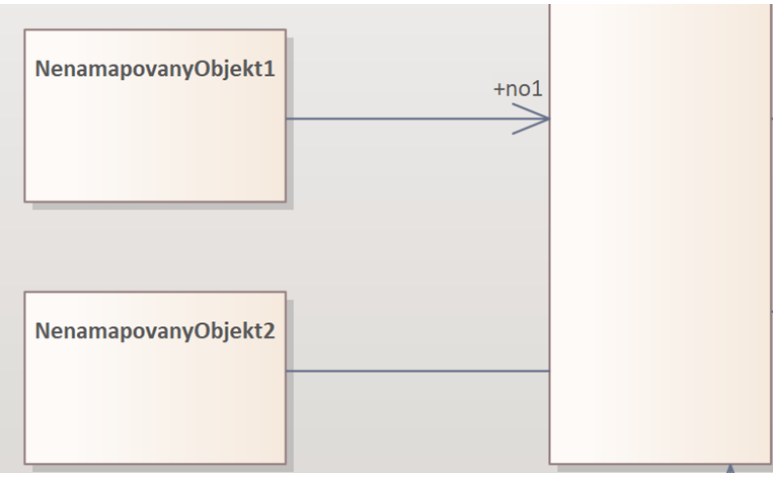
- Asociace
  - Vazba mezi třídami.
  - Vždy se nějak musí určit směr. To chrání proti zacyklení. Jsou 2 způsoby: šipkou, pojmenováním.



Obrázek 3.17: Směr asociace určen šipkou

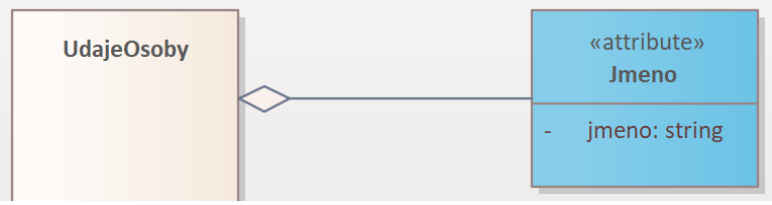
Obrázek 3.18: Směr asociace určen pojmenováním

- Pokud směr nejde určit nebo je šipka v opačném směru, pak se záznam při mapování ignoruje.



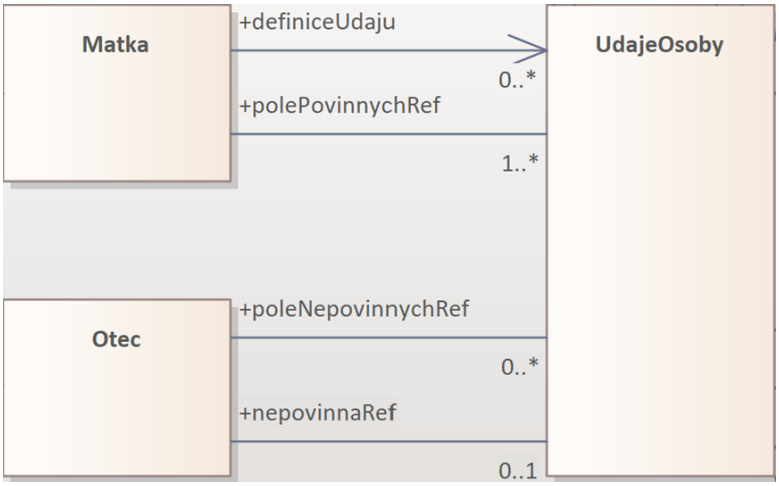
Obrázek 3.19: Směr asociace nelze určit

- Agregace / Kompozice
  - Vazba mezi záznamem a atributem nebo číselníkem.



Obrázek 3.20: Agregace v Kafka diagramu

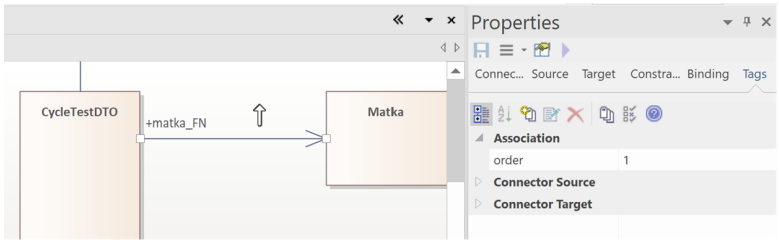
- Kardinality
  - Default / Nezadán - 1 - povinný
  - 1 - povinný
  - 1..* - povinné pole
  - 0..1 - nepovinný
  - 0..* - nepovinné pole



Obrázek 3.21: Druhy kardinalit mezi záznamem a jeho poli

- Pojmenování polí
  - Název pole se uvádí na vazbě na straně rodičovského záznamu.
  - Název pole se vezme primárně z pojmenování vazby.
  - Pokud není vazba pojmenovaná, pak se defaultně použije název atributu.
  - V jednom záznamu nemůže být více polí se stejným názvem. Duplikáty se ignorují a nemapují. Může k tomu dojít například, když jeden záznam bude mít více vazeb na stejný atribut a vazby nebudou pojmenované. Použije se totiž defaultně název atributu a ten bude samozřejmě stejný.

- Pořadí polí
  - Na vazbě je možné definovat vlastní pořadí polí pomocí tagu *order*.
  - *Order* se definuje na vazbě, ne na atributu, protože atribut je možné opakovaně použít v jiných diagramech, kde se pořadí může lišit.
  - Nižší čísla jsou první v pořadí. Nedefinovaná čísla jsou na konci.
  - Pokud mají dvě vazby stejný *order*, pak se řadí sekundárně podle abecedy jména fieldů.
  - Příklad pořadí:
    1. order: 1
    2. order: 2, name: A
    3. order: 2, name: B
    4. order: null, name: A
    5. order: null, name: B



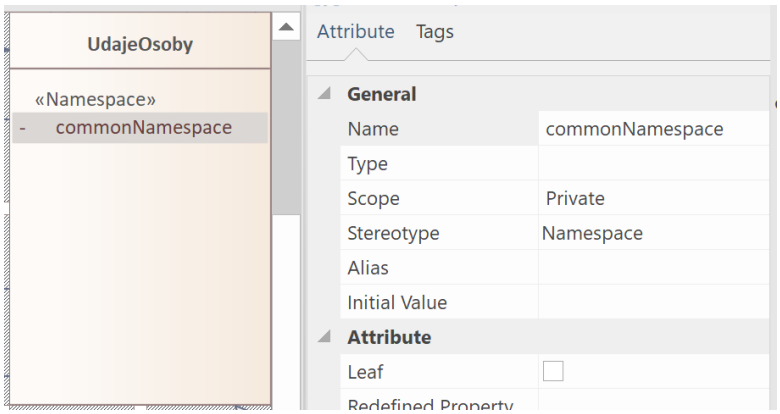
Obrázek 3.22: Tag order na vazbě

Stereotypy v EA

V EA jsme si stanovili následující stereotypy pro Kafka diagramy:

- *KafkaMessage*
  - Třída s tímto stereotypem je počátkem každého Kafka diagramu.
- *attribute*
  - Třída s tímto stereotypem v sobě obsahuje základní, logické a číselníkové datové typy.
- *Namespace*
  - Na záznamu je možné definovat atribut s tímto stereotypem.
  - Určuje se tím jmenný prostor, který přepíše *projectSubNamespace* z konfigurace.
  - Aplikuje se na daný záznam i všechny jeho vnořené záznamy.

- Př.: Pokud je *baseNamespace* z konfigurace nastaven na *cz.cvut.fel* a na záznamu je definován atribut se stereotypem *Namespace* s hodnotou *commonNamespace*, pak výsledný jmenný prostor pro tento záznam i jeho vnořené záznamy bude *cz.cvut.fel.commonNamespace*.



Obrázek 3.23: Definice Namespace na recordu

Datové typy

Tabulka níže představuje seznam datových typů, kterých mohou nabývat atributy na třídě se stereotypem *attribute*. (Nerozlišují se velká a malá písmena.)

Typy v EA	Základní typ v Avro	Logický typ v Avro
string, str	string	
stringuuid	string	uuid
int, integer	int	
long, datumacas	long	
float	float	
double	double	
date, datum	int	date
datetime	long	timestamp-millis
boolean, bool	boolean	
byte, bytes	bytes	
decimal	bytes	decimal

Tabulka 3.3: Datové typy atributů v Kafka diagramech a jejich příslušné datové typy v Avro specifikaci

3.2.4 JSON modul

V této kapitole je popsána funkcionalita generování JSON souborů pro Kafka diagramy.

Config

Nastavení JSON modulu má svojí část v souboru *config.json*. Níže je uvedena ukázka:

```
"jsonSchema": {
    "subfolder": "kafka/json_schema"
}
```

- *subfolder* - Podsložka, do které se budou generovat výsledné JSON soubory.

Popis funkcionality

Při generování souborů pro Kafka diagramy se kromě Avro souborů vygenerují i příslušná JSON schémata.

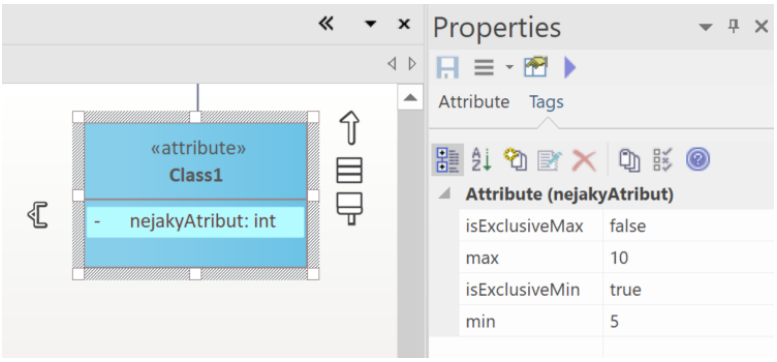
Rozšíření metodiky pro zakreslení Kafka diagramů

Kafka diagram, který byl doposud určen pro generování Avro schémat, není třeba nijak měnit. Generování Avro schémat bude fungovat stejně jako dříve. Je zde však možné navíc definovat omezení pro účely JSON schémat.

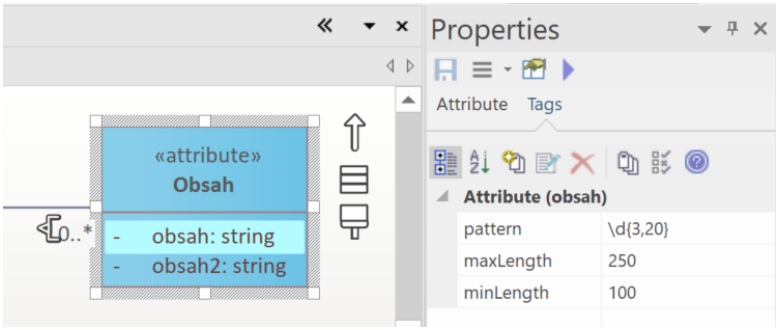
Omezení

U atributů lze definovat následující omezení pomocí tagů v EA:

- *min* - minimum čísla
- *isExclusiveMin* - zda se má vyloučit *min* čísla z intervalu [*true*/*false*]
- max - maximum čísla
- *isExclusiveMax* - zda se má vyloučit *max* čísla z intervalu [*true*/*false*]
- *minLength* - minimální délka textu
- *maxLength* - maximální délka textu
- *pattern* - vzor textu



Obrázek 3.24: Ukázka JSON schéma omezení 1



Obrázek 3.25: Ukázka JSON schéma omezení 2

Datové typy

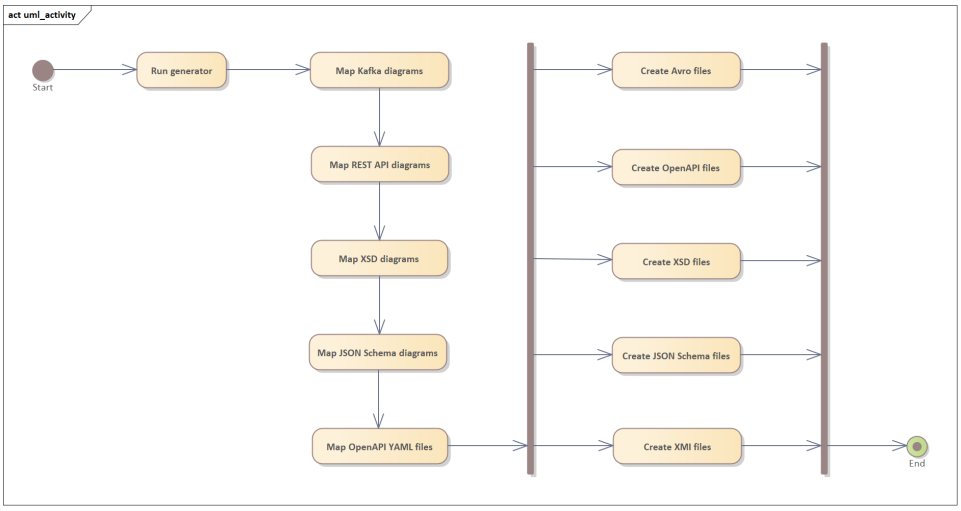
Tabulka níže představuje seznam datových typů, kterých mohou nabývat atributy na třídě se stereotypem *attribte*. (Nerozlišují se velká a malá písmena.)

Typy v EA	Základní typ v JSON	Formát v JSON
string, str	string	
stringuuid	string	uuid
int, integer	integer	
long, datumacas	integer	
float	number	
double	number	
date, datum	string	date
datetime	string	date-time
boolean, bool	boolean	
byte, bytes	string	
decimal	number	

Tabulka 3.4: Datové typy atributů v Kafka diagramech a jejich příslušné datové typy v json schématech

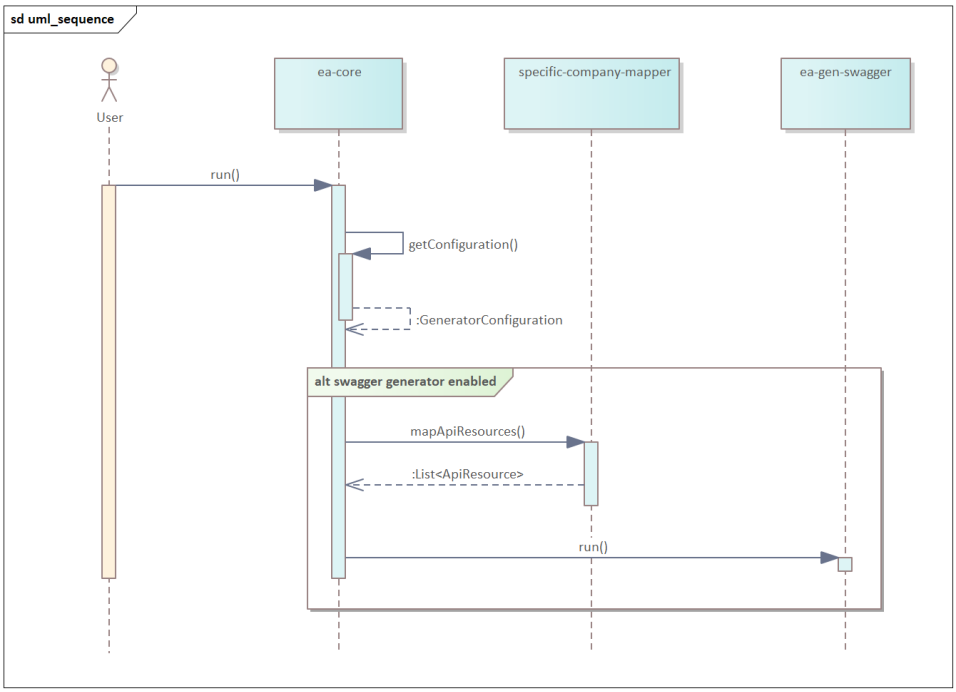
3.3 Proces generování

Hlavní proces generování funguje následovně: Po spuštění aplikace se nejprve provádí mapování všech diagramů z EA a všech vstupních OpenAPI souborů. Poté se pro každý namapovaný objekt generují výstupní soubory pomocí příslušných modulů. Průběh tohoto procesu je znázorněn níže na obrázku 3.26. Tento diagram je zjednodušen pro lepší přehlednost, ale ve skutečnosti se nemusí vykonat každá aktivita, pokud daný modul není zapnutý v konfiguraci.



Obrázek 3.26: Proces generování

Jako příklad se zaměříme na generování OpenAPI souborů. Na obrázku 3.27 je zobrazen sekvenční diagram tohoto procesu, který ukazuje jednotlivé kroky. Vidíme zde i komponenty, které provádí jednotlivé aktivity. Hlavní běh aplikace je řízen modulem *ea-core*, který nejdříve načte konfiguraci a zkontroluje, zda je zapnutý daný modul. Pokud je zapnutý, nejdříve namapuje REST API diagramy pomocí specifického mapperu dané společnosti. Nakonec pak spustí modul *ea-gen-swagger*, který vygeneruje OpenAPI soubory. (Tyto komponenty budou podrobněji rozebrány v následující kapitole 3.4.)



Obrázek 3.27: Proces generování OpenAPI

3.4 Architektura

Aplikaci jsme se rozhodli rozdělit do dvou knihoven: *ea-generator* a mapper specifické společnosti.

3.4.1 Ea-generator

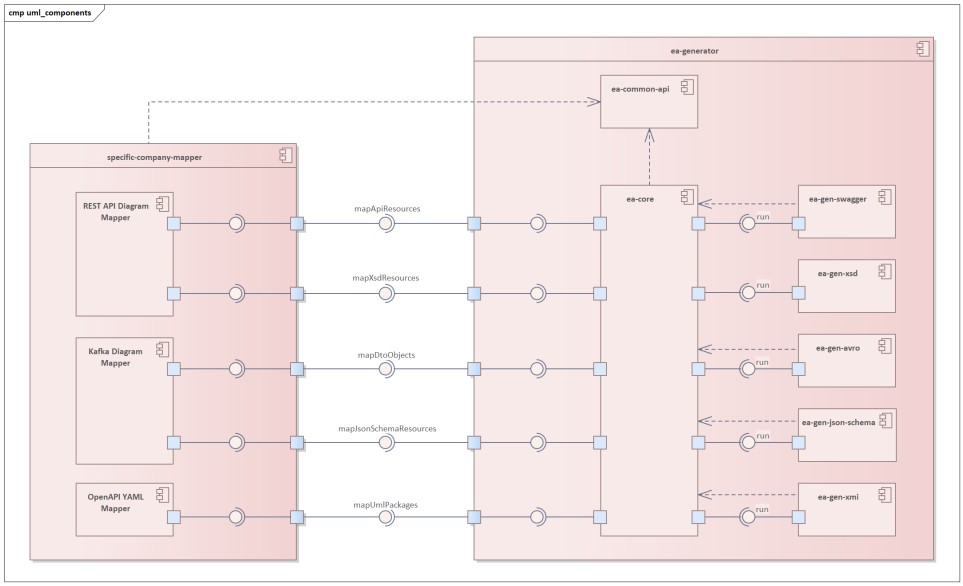
První knihovna *ea-generator* slouží jako jádro aplikace. Je společná pro všechny projekty, a tedy nezávislá na metodice dané společnosti. Obsahuje sedm hlavních modulů:

- *ea-core* - Řídí proces generování a obsahuje databázové entity pro JPA.
- *ea-common-api* - Obsahuje třídy, které slouží k popisu datových struktur a UML metamodelu. (Viz kapitoly 4.1 a 4.2.)
- *ea-gen-swagger* - Generuje OpenAPI soubory.
- *ea-gen-xsd* - Generuje XSD soubory.
- *ea-gen-avro* - Generuje Avro soubory.
- *ea-gen-json-schema* - Generuje JSON schema soubory.
- *ea-gen-xmi* - Generuje XMI soubory.

3.4.2 Mapper specifické společnosti

Druhá knihovna je vždy závislá na konkrétním projektu. Zde je implementována metodika zakreslování či procházení diagramů v EA v souladu s požadavky dané společnosti.

V našem případě se jedná o MPSV a metodologii popsanou v kapitole 3.2.



Obrázek 3.28: Komponenty systému

3.5 Možnosti pro následující generování kódu

Generátor nám podle nakreslených diagramů vygeneruje například OpenAPI nebo Avro schémata. Tohle jsou velice rozšířené a standardizované specifikace, proto pro další práci s nimi existuje již mnoho veřejně dostupných nástrojů. Mimo jiné i nástroje pro generování tříd v kódu.

Pro OpenAPI specifikaci existuje více generátorů. Jsou to například *Swagger Codegen* nebo z toho vzniklý *OpenAPI Generator*. Ty umožňují generovat kód v různých jazycích, včetně Java, JavaScript, C# a dalších. [20], [21]

Pro Avro schéma můžeme použít generátor kódu přímo od společnosti *Apache Avro*, který nám vygeneruje potřebné třídy a číselníky v Javě. [22]

Obdobně existují i knihovny pro XSD schémata. Jsou to například *JAXB* nebo *XMLBeans*. [23]

Nakonec zmíníme i nástroj pro generování Java tříd z JSON schémat. Jedná se o *jsonschema2pojo*. Můžeme ho použít například jako Maven plugin. [24]

Kapitola 4
Implementace

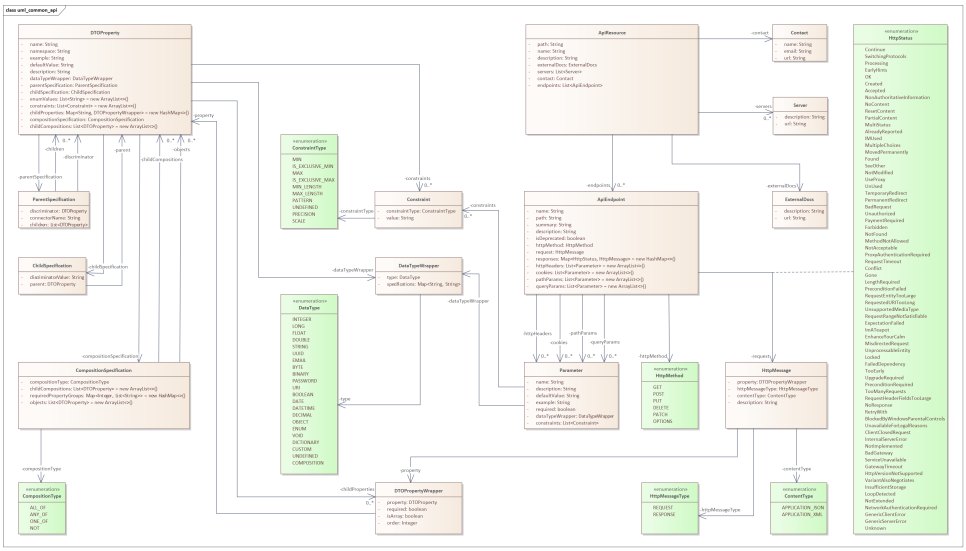
V následující kapitole podrobně popíšeme implementaci klíčových částí aplikace. Především se zaměříme na popis datových struktur a UML metamodelu. Dále rozebereme návrhové vzory použité v průběhu vývoje a na závěr zmíníme zásady a principy, jimiž jsme se řídili při samotné implementaci.

4.1 Common API

Common API je model sloužící pro popis datové struktury zpráv. Například umožňuje zaznamenat, že daný objekt má nějaké vlastnosti a ty zase můžou nabývat jistých hodnot. Podobnou funkčnost mají i jednotlivé specifikace pro OpenAPI, XSD, Avro nebo JSON schéma. Popisují datovou strukturu zpráv. Naším cílem tedy bylo implementovat jednotný model, jenž by byl použitelný napříč všemi moduly, nezávisle na specifických metodikách používaných danou společností. Tímto univerzálním modelem je *Common API*. [5]

V aplikaci je tento model využíván následujícím způsobem: Mapper specifické společnosti projde diagramy v EA a zaznamená data pomocí tohoto modelu *Common API*. Tyto záznamy jsou poté předány do *ea-core* a následně do modulů, kde se vygenerují příslušné výstupní soubory. [5]

Common API jsme převzali z projektu Ing. Martina Mašaty a rozšířili dle funkčních a nefunkčních požadavků uvedených v kapitole 2.2. Nový výsledný model je znázorněn na obrázku 4.1.



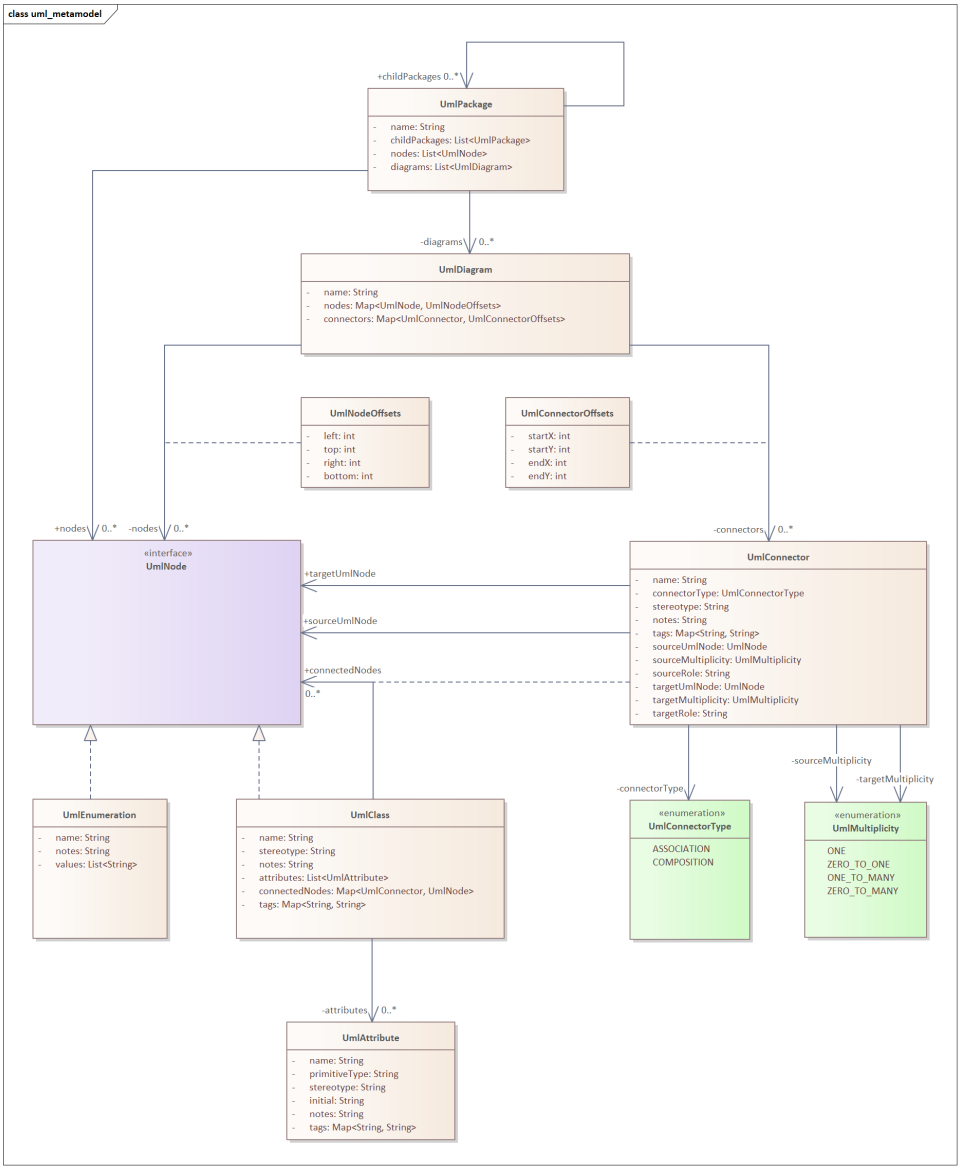
Obrázek 4.1: Common API

4.2 UML Metamodel

UML Metamodel je dalším klíčovým modelem naší aplikace. Jedná se o model popisující UML diagramy. Definuje elementy, atributy a jejich vzájemné vztahy. Obdobně jako *Common API*, i *UML Metamodel* slouží jako univerzální model napříč více moduly a odstiňuje specifickou implementaci metodiky od obecné funkcionality generování diagramů. Poskytuje tedy abstrakci, díky které implementace i konkrétní formát výstupu závisí čistě jen na daném modulu. V našem případě se zatím jedná jen o XMI modul, jehož výstupem jsou soubory XMI 2.1.

Příklad procesu, který demonstruje využití *UML Metamodelu*, je následující: Mapper specifické společnosti načte vstupní OpenAPI soubory a dle metodik dané společnosti namapuje jejich strukturu do *UML Metamodelu*. Tato data jsou poté předána do *ea-core* a následně do *XMI modulu*, kde se vygenerují odpovídající soubory. Výsledné XMI soubory lze nakonec importovat do EA, kde jsou vizualizovány jako UML diagramy.

Jedná se o úplně novou funkcionalitu. Diagram tohoto modelu, který jsme implementovali, je zobrazen na obrázku 4.2.



Obrázek 4.2: UML Metamodel

4.3 Použité návrhové vzory

Během vývoje jsme využívali řadu návrhových vzorů, které řeší standardní problémy spojené s vytvářením objektů, struktur a implementací chování aplikace. [25]

Například při průchodu diagramem potřebujeme aplikovat odlišné postupy v závislosti na typech konektorů. Proto zde využíváme návrhový vzor *Strategy*, pomocí kterého implementujeme rozdílné chování pro průchod přes asociace nebo agregace v rozdílných typech diagramů.

Další klíčovou funkcionalitu v naší aplikaci představuje mapování datových struktur, kde je zapotřebí vytvářet mnoho různých objektů a struktur. Využili jsme tedy vzory jako *Builder*, *Abstract Factory*, *Porototype*, nebo *Composite*.

Nakonec ještě zmíníme použití návrhových vzorů, jež vyplynulo z volby frameworku *Spring Boot*. Jedná se především o *IC/DI (Inversion of Control / Dependency Injection)*. Framework za nás spravuje životní cyklus objektů označených například anotacemi *@Service*, *@Component* nebo *@Repository*. Navíc jsme ponechali defaultní chování *Spring Bootu*, proto se vytváří jen jedna instance takovýchto objektů. Jinými slovy, na pozadí frameworku se používá i návrhový vzor *Singleton*. [26]

4.4 Použité principy

Během implementace projektu jsme využívali osvědčené principy softwarového inženýrství, aby náš kód byl jednoduchý, čitelný a aby se s ním dobře pracovalo. Mezi tyto principy patří například DRY nebo SOLID.

Princip DRY (*Don't Repeat Yourself*) nám říká, že se při implementaci nemáme opakovat. To znamená, že by se ten samý kus kódu neměl vyskytnout na více místech v programu. Pomáhá nám to s udržitelností kódu a předcházíme tím řadě problémů spojených s budoucími úpravami. [27]

SOLID představuje soubor principů objektově orientovaného programování. Jedná se o 5 zásad: *Single responsibility principle*, *Open/closed principle*, *Liskov substitution principle*, *Interface segregation principle*, *Dependency inversion principle*. Stručně řečeno, tyto principy pomáhají psát přehledný, přepoužitelný, rozšiřitelný a modulární kód, který se dobře udržuje. [28]

Kapitola 5
Testování

Nyní se zaměříme na metody, které jsme při testování aplikace uplatnili. Konkretně jsme zvolili tři přístupy: automatizované testy, testovací scénáře a uživatelské testy. Rozebereme každou metodu a řekneme si, proč jsme takto postupovali. Nakonec shrneme, co jsme vyvodili z provedených testů.

5.1 Automatizované testy

Rozhodli jsme se implementovat automatizované testy, konkrétně unit testy, pro jádro aplikace. Toto jádro, tedy knihovna *ea-generator*, představuje kritickou část naší aplikace. Tato knihovna se totiž využívá na více projektech a pracuje na ní více jednotlivců. Existuje zde tedy i vyšší pravděpodobnost výskytu potenciálních chyb, které by mohly mít dopad na chod ostatních projektů.

S ohledem na to, že naše aplikace je vyvíjena v Javě, jsme se rozhodli při tvorbě unit testů využít široce známé knihovny *JUnit* a *Mockito*. *JUnit* poskytuje sadu nástrojů pro vytváření testů a jejich řízení. *Mockito* naopak pomáhá při mockování objektů. [29] [30]

5.2 Testovací scénáře

Dále jsme definovali pět testovacích scénářů, které pokrývají většinu případů užití naší aplikace. Jedná se vždy o celý proces od přípravy diagramů až po vygenerování výstupních souborů. Tyto testovací scénáře pak ověřujeme před každým větším nasazením do produkce.

Zde je uveden přehled těchto pěti scénářů se stručným popisem kroků a očekávaných výsledků. Specifické diagramy, na které se budeme odkazovat, máme definované v našem prostředí, avšak zde je ukazovat nebudeme. Stejně tak nebudeme uvádět ani podrobnější podmínky jednotlivých scénářů, které většinou zahrnují jen nastavení příslušných oprávnění nebo splnění předchozích kroků.

- **RestApiCommon** - V EA připravíme diagram *RestApiCommon*. Spustíme generátor nad tímto diagramem. Na výstupu dostaneme OpenAPI a XSD soubory. Zkontrolujeme, že vygenerované soubory odpovídají očekávanému výsledku.
- **RestApiComposition** - V EA připravíme diagram *RestApiComposition*. Spustíme generátor nad tímto diagramem. Na výstupu dostaneme OpenAPI a XSD soubory. Zkontrolujeme, že vygenerované soubory odpovídají očekávanému výsledku.
- **KafkaCommon** - V EA připravíme diagram *KafkaCommon*. Spustíme generátor nad tímto diagramem. Na výstupu dostaneme Avro a JSON Schema soubory. Zkontrolujeme, že vygenerované soubory odpovídají očekávanému výsledku.
- **KafkaComposition** - V EA připravíme diagram *KafkaCommon*. Spustíme generátor nad tímto diagramem. Na výstupu dostaneme Avro a JSON Schema soubory. Zkontrolujeme, že vygenerované soubory odpovídají očekávanému výsledku.
- **ImportedController** - Pro import připravíme OpenAPI soubor *ImportedController.yaml*. Spustíme generátor. Na výstupu dostaneme XMI soubor *ImportedController.xml*. XMI soubor importujeme do EA. Zkontrolujeme, že výsledný diagram odpovídá očekávání.

5.3 Uživatelské testy

Nasazenou aplikaci jsme podrobili testování různými osobami z rozdílných projektů. Především jsme se zaměřili na funkcionality generování OpenAPI souborů, Avro a JSON schémat. Během testování jsme objevili a následně opravili množství chyb. Zároveň jsme identifikovali i několik chybějících funkcionalit, které jsme poté doplnili.

Funkcionalita pro generování diagramů z existujících OpenAPI souborů je poměrně nová, což znamená, že dosud ještě nebyla podrobena zcela důkladnému testování. Zatím jsme provedli jen krátké testování, avšak i zde jsme už identifikovali a následně odstranili jisté nedostatky. Dále zde vyvstaly i náměty na zlepšení topologie diagramů, na kterých plánujeme pracovat v budoucnu.

Kapitola 6

Návrhy na budoucí rozšíření

Vývoj této aplikace rozhodně není u konce. Plánujeme nadále přidávat další funkcionality. Několik návrhů na rozšíření vzešlo i od uživatelů, kteří aplikaci již využívají v reálném prostředí. Níže uvedeme několik konkrétních návrhů, které hodláme implementovat v blízké době a které mohou mít největší přínos do budoucna.

- **Generování diagramů z dalších formátů** - V současné době generujeme diagramy pouze z OpenAPI souborů. Tato funkcionalita se ukázala být úspěšná a zaznamenali jsme tedy zájem o její rozšíření o další formáty. Jedná se především o možnost generovat diagramy ze vstupních souborů typu XSD, Avro nebo JSON schémat.
- **Topologie diagramů** - Při generování diagramů je klíčové přehledně nastavit velikost elementů a jejich rozložení. Toho jsme dosáhli poměrně úspěšně. Nicméně umístění konektorů není vždy ideální. Občas se stává, že se překrývají s ostatními elementy nebo jinými konektory, což snižuje přehlednost diagramů. V budoucnu bychom se chtěli více zaměřit právě na tento aspekt.
- **Vyšší pokrytí unit testů** - Existuje možnost, že se aplikace začne používat na více projektech. K rozšíření jádra aplikace tedy bude přispívat více lidí, což zvýší pravděpodobnost výskytu chyb, které by měly dopad na ostatní projekty. Zvýšením počtu unit testů a pokrytím všech možných scénářů použití předejdeme mnoha problémům, které by mohly vyvstat.
- **Paralelizace mapování** - Tento návrh má asi nejnižší prioritu z výše zmíněných. Nicméně pokud aplikace bude dostávat velké množství dat na vstupu, pak stojí za zvážení, jestli nezlepšit její rychlost. Nabízí se tedy možnost paralelizovat procesy mapování vstupních souborů a diagramů.

Kapitola 7

Závěr

Cílem této práce bylo rozšířit existující moduly pro Swagger a Avro schémata a zároveň vyvinout nové moduly pro generování JSON schémat, XSD a XMI souborů. Tím pak usnadnit práci vývojářů při implementaci.

Práce navázala na diplomovou práci Ing. Martina Mašaty. Motivací pro Martinův a následně pak i můj projekt byly požadavky reálného projektu MPSV na automatizovaný generátor souborů pro vývoj softwaru.

Nejprve jsme si v analýze definovali funkční a nefunkční požadavky. Podrobně jsme prozkoumali REST API, XSD, Apache Kafka, JSON schéma, UML a Enterprise Architect, tedy klíčové technologie, se kterými generátor pracuje.

V návrhu jsme představili zvolené technologie a metodiku pro tvorbu diagramů v EA. Popsali jsme hlavní procesy včetně obecné architektury aplikace. Dále jsme se podívali na možnosti následného generování kódu.

V další kapitole jsme se věnovali tomu, jak probíhala samotná implementace. Zaměřili jsme se především na popis datových struktur a UML metamodelu. Popsali jsme použité návrhové vzory a principy, kterých jsme se při implementaci drželi.

Dále jsme podrobně diskutovali průběh testování a návrhy na budoucí rozšíření, jež z toho vyplynuly.

Na závěr bychom chtěli konstatovat, že jsme dosáhli stanovených cílů této práce. Získaný nástroj poskytuje efektivní způsob generování souborů pro implementaci softwarových systémů, což odpovídá potřebám aktuálního projektu Ministerstva práce a sociálních věcí. Generátor bude v budoucnu určitě rozšiřován i o další funkcionality. Velký potenciál vidím v komponentě pro generování XMI, o kterou jeví zájem i další projekty mimo ministerstvo.

Literatura

[1] Mark Masse. REST API design rulebook: designing consistent RESTful web service interfaces, 2011. Dostupné z: https://pepa.holla.cz/wp-content/uploads/2016/01/REST-API-Design-Rulebook.pdf.

[2] Nishant Garg. Apache Kafka, 2013. Dostupné z: http://archive.keyllo.com/L-%E7%BC%96%E7%A8%8B/Kafka-Apache%20Kafka.pdf.

[3] Sparx Systems. ENTERPRISE ARCHITECT. Dostupné z: https://sparxsystems.com/resources/user-guides/16.1/basics/getting-started.pdf.

[4] DATAROB. Software Development Life Cycle. Dostupné z: https://datarob.com/essentials-software-development-life-cycle/.

[5] Martin Mašata. Generování kódu z modelů Enterprise Architect. Dostupné z: https://dspace.cvut.cz/handle/10467/109230.

[6] Matthias Biehl. API Architecture, 2015. Dostupné z: https://books.google.cz/books?id=6D64DwAAQBAJ.

[7] Swagger supported by SmartBear. OpenAPI Specification, 2021. Dostupné z: https://swagger.io/specification/.

[8] Joshua S Ponelat and Lukas L Rosenstock. Designing APIs with Swagger and OpenAPI, 2022. Dostupné z: https://dl.ebooksworld.ir/books/Designing.APIs.with.Swagger.and.OpenAPI.Ponelat.Rosenstock.Manning.9781617296284.EBooksWorld.ir.pdf.

[9] W3C. W3C XML Schema Definition Language (XSD). Dostupné z: https://www.w3.org/TR/xmlschema11-1/.

[10] Priscilla Walmsley. Definitive XML schema, 2001. Dostupné z: https://books.google.cz/books?id=8yd9AUZXI4IC.

[11] Kafka. DOCUMENTATION. Dostupné z: https://kafka.apache.org/documentation/.

[12] Apache Avro. Specification. Dostupné z: https://avro.apache.org/docs/1.11.1/specification/.

[13] JSON Schema. Specification. Dostupné z: https://json-schema.org/specification.

[14] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of JSON schema. Dostupné z: https://jreutter.sitios.ing.uc.cl/www16.pdf.

[15] Dan Pilone and Neil Pitman. UML 2.0 in a Nutshell, 2005. Dostupné z: https://books.google.cz/books?id=Xg8wLPmt5CMC&lpg=PR7&ots=EzL83Tei1B&dq=uml&lr&pg=PR7#v=onepage&q&f=false.

[16] Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado. UML 2 Toolkit, 2003. Dostupné z: https://books.google.cz/books?id=jEXnzbE_OooC.

[17] OMG. XMI. Dostupné z: https://www.omg.org/spec/XMI/2.1.1.

[18] Timothy J Grose, Gary C Doney, and Stephen A Brodsky. Mastering Xmi: Java Programming with Xmi, XML and UML, 2002. Dostupné z: https://books.google.cz/books?id=6B5-Wz6WbIIC.

[19] Nayan B. Ruparelia. Software Development Lifecycle Models. Dostupné z: https://www.researchgate.net/publication/220631422_Software_development_lifecycle_models.

[20] Swagger supported by SmartBear. Swagger Codegen Documentation. Dostupné z: https://swagger.io/docs/open-source-tools/swagger-codegen/.

[21] OpenAPI Generator. Generators List. Dostupné z: https://openapi-generator.tech/docs/generators.

[22] Apache Avro. Getting Started (Java). Dostupné z: https://avro.apache.org/docs/1.11.1/getting-started-java/.

[23] JetBrains IntelliJ IDEA. Generate Java Code from XML Schema. Dostupné z: https://www.jetbrains.com/help/idea/generating-java-code-from-xml-schema.html.

[24] Joe Littlejohn. jsonschema2pojo: Generate Plain Old Java Objects from JSON or JSON-Schema. Dostupné z: https://www.jsonschema2pojo.org/.

[25] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: Elements of reusable object-oriented software, 1994. Dostupné z: https://www.javier8a.com/itc/bd1/articulo.pdf.

[26] Craig Walls. Spring in action, 2022. Dostupné z: https://dl.ebooksworld.ir/books/Spring.in.Action.6th.Edition.Craig.Walls.Manning.9781617297571.EBooksWorld.ir.pdf.

[27] Girish Suryanarayana, Ganesh Samarthyam, and Tushar Sharma. Refactoring for software design smells: managing technical debt, 2014. Dostupné z: https://books.google.cz/books?id=1SaOAwAAQBAJ&lpg=PP1&ots=Oyb_FG4yEf.

[28] Robert C Martin. Clean architecture, 2017. Dostupné z: https://agorism.dev/book/software-architecture/%28Robert%20C.%20Martin%20Series%29%20Robert%20C.%20Martin%20-%20Clean%20Architecture_%20A%20Craftsman%E2%80%99s%20Guide%20to%20Software%20Structure%20and%20Design-Prentice%20Hall%20%282017%29.pdf.

[29] Stefan Bechtold, Sam Brannen, Johannes Link, Matthias Merdes, Marc Philipp, Juliette de Rancourt, and Christian Stein. JUnit 5 User Guide. Dostupné z: https://junit.org/junit5/docs/current/user-guide/.

[30] Java Docs. Mockito. Dostupné z: https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html.

Příloha A

Seznam zkratek

API	Application Programming Interface
AVSC	Avro Schema
DRY	Don't Repeat Yourself
EA	Enterprise Architect
FEL	Fakulta elektrotechnická
FR	Functional Requirement
IC/DI	Inversion of Control / Dependency Injection
JPA	Java Persistence API
JSON	JavaScript Object Notation
MPSV	Ministerstvo práce a sociálních věcí
NFR	Non-Functional Requirement
REST	Representational State Transfer
SOLID	Single responsibility principle, Open/closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XSD	XML schema definition
YAML	YAML Ain't Markup Language

Příloha B

Instalace a spuštění

Tato kapitola popisuje, jak připravit prostředí, nakonfigurovat aplikaci a jak ji následně spustit.

B.1 Prerekvizity

Je nezbytné mít nainstalovány následující technologie na počítači. Předpokládá se základní znalost práce s těmito nástroji.

Technologie	Verze
Java	17.0.2
Enterprise Architect	15.2
MySQL	8.0.28

Tabulka B.1: Prerekvizity

B.2 Příprava databáze

Jako repozitář pro EA budeme používat databázi MySQL. Zde jsou kroky, jak připravit takovýto nový repozitář.

1. V MySQL vytvoříme prázdné schéma. Pojmenujeme si ho například *ea_repo*. Stačí defaultní nastavení.

2. Ze stránek společnosti *Sparx Systems* stáhneme SQL skript pro přípravu MySQL schématu. (Kapitola nadepsaná *Enterprise Architect Schema Creation Scripts*.) Následně tento skript pustíme v našem prázdném schématu *ea_repo*. Tím jsme v naší databázi vytvořili prázdné tabulky.

   `https://sparxsystems.com/resources/repositories/`

3. Následně z té samé webové stránky stáhneme SQL skript pro vytvoření počátečních hodnot v MySQL databázi. (Kapitola nadepsaná *Enterprise Architect Initial data Scripts*.) Pustíme skript v našem schématu *ea_repo*. Tím jsme do databáze přidali základní hodnoty číselníku a systémových tabulek.

   `https://sparxsystems.com/resources/repositories/`

Přípravu databáze máme tedy hotovou.

B.3 Připojení EA do databáze

Nyní připojíme EA do naší MySQL databáze, kde se budou ukládat veškerá data.

1. Ujistíme se, že máme na počítači *32-bit MySQL ODBC 8.0 Unicode Driver*. Případně se dá stáhnout z této stránky:

   `https://dev.mysql.com/downloads/connector/odbc/`

2. Následně přidáme nový datový zdroj do EA, což bude naše MySQL databáze. Tento proces je popsán zde:

   `https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_repository/setupmysqlodbcdriver.html`

3. Nakonec se v EA připojíme k databázi pomocí nového datového zdroje. Detailní kroky můžeme vidět zde:

   `https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_repository/connecttomysql.html`

Připojili jsme se tedy k databázi a v EA máme otevřený nový prázdný projekt. Můžeme zde kreslit diagramy v souladu se zvolenou metodikou našeho generátoru.

B.4 Konfigurace a spuštění generátoru

B.4.1 Příprava JAR souboru

Buď dostaneme JAR soubor předem, nebo si ho budeme muset připravit sami následujícím postupem:

1. Vezmeme zdrojové kódy první knihovny *ea-generator*, která je jádrem aplikace, a pustíme *Maven* příkaz:

   maven clean install.

2. Dále provedeme to samé i s druhou knihovnou *ea-gen*, která implementuje mapování dle zvolené metodiky. Opět příkazem:

   maven clean install.

3. Výsledný JAR soubor nalezneme v adresáři *target* druhého projektu nebo v lokálním *Maven* repozitáři. V našem případě se tento soubor jmenuje *ea-gen-0.0.1-SNAPSHOT.jar*.

B.4.2 Konfigurace

Konfiguraci nastavíme v JSON souboru, který si pojmenujeme například *config.json*. Zde vidíme ukázku:

```
{
  "databaseConnection": {
    "url": "jdbc:mysql://127.0.0.1:3306/ea_repo",
    "user": "ea_repo_user",
    "password": "ea_repo_password"
  },
  "mappingConfiguration": {
    "type": "custom",
    "profile": "custom-mapper"
  },
  "eaStartPackage": "Model.ea_gen",
  "version": "1.0.0",
  "enabledGenerators": [
    "swagger",
    "xsd",
    "avro",
    "json-schema",
    "xmi"
  ],
  "swagger": {
    "title": "Custom TITLE of API",
    "subfolder": "rest_api/openapi",
    "separateFiles": true
  },
  "xsd": {
    "subfolder": "rest_api/xsd"
  },
  "avro" : {
    "baseNamespace" : "cz.cvut.fel",
    "projectSubNamespace" : gg,
    "enumSubNamespace" : "enums",
    "subfolder": "kafka/avro"
  },
  "jsonSchema": {
    "subfolder": "kafka/json_schema"
  },
  "xmi": {
    "subfolder": "xmi"
  }
}
```

Parametr	Popis
databaseConnection.url	URL databáze a schématu.
databaseConnection.user	Uživatel pro připojení k DB.
databaseConnection.password	Heslo pro připojení k DB.
mappingConfiguration.type	Typ mapování. V tuto chvíli vždy *custom*.
mappingConfiguration.profile	Název mapperu. Odpovídá názvu v anotaci *@Mapper*.
eaStartPackage	Umístění diagramů v EA.
version	Verze výstupních souborů.
enabledGenerators	Seznam zapnutých generátorů. Odpovídá názvu v anotaci *@Generator*.
swagger.title	Název vygenerovaného OpenAPI dokumentu.
swagger.subfolder	Zpřesňuje podsložku exportu. Nepovinný atribut.
swagger.separateFiles	Určuje zda se má vygenerovat jeden velký OpenAPI soubor nebo více menších. Default: false
xsd.subfolder	Zpřesňuje podsložku exportu pro XSD soubory. Nepovinný atribut.
avro.baseNamespace	Základní jmenný prostor pro všechny definice.
avro.projectSubNamespace	Jmenný prostor projektu pro všechny záznamy bez definováného *Namespace*. Přidává se za *baseNamespace*.
avro.enumSubNamespace	Jmenný prostor vygenerovaných číselníku. Přidává se za *baseNamespace*.
avro.subfolder	Zpřesňuje podsložku exportu pro Avro soubory. Nepovinný atribut.
jsonSchema.subfolder	Zpřesňuje podsložku exportu pro JSON Schema soubory. Nepovinný atribut.
xmi.subfolder	Zpřesňuje podsložku exportu pro XMI soubory. Nepovinný atribut.

Tabulka B.2: Parametry konfigurace, zdroj: Autor, [5]

B.4.3 Spuštění

Vše máme připravené a můžeme tedy aplikaci spustit pomocí *Javy* příkazem:

java -Dfile.encoding=UTF-8 -jar ea-gen-0.0.1-SNAPSHOT.jar config.json

Parametr	**Popis**
UTF-8	Zajistí správné zpracování českých znaků.
ea-gen-0.0.1-SNAPSHOT.jar	Cesta k našemu JAR souboru. V tomto ukázkovém případě se tedy nachází v aktuálním adresáři.
config.json	Cesta k našemu konfiguračnímu souboru. V tomto ukázkovém případě se tedy nachází v aktuálním adresáři.

Tabulka B.3: Parametry spuštění