VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Semiformální modelování s využitím formálních omezení a transformace mezi notacemi procesů - diagram aktivit UML

Semiformal modeling using formal constraints and transformation between notation processes - UML activity diagram

Bc. Martin Gajdoš

Diplomová práce

Vedoucí práce: Ing. Svatopluk Štolfa, Ph.D.

Ostrava, 2022

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Martin Gajdoš**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Semiformální modelování s využitím formálních omezení a transformace mezi notacemi procesů - diagram aktivit UML
Semiformal modeling using formal constraints and transformation between process notations – Activity diagram notation

Jazyk vypracování: slovenština

Zásady pro vypracování:

Cílem práce je vytvořit grafický nástroj pro modelování procesů, který kombinuje formální a semiformální přístup k modelování procesů a je součástí metodiky vyvíjené na katedře informatiky. Konkrétním výstupem této práce je pak vyřešení transformace notace aktivitního diagramu UML jazyka do obecného metamodelu, což mimo jiné umožní trasformaci do jiných neformálních notací. Dále pak zpracování možnosti dodržování definovaných formálních omezení při modelování pomocí neformální notace.

1. Seznámení se se současným stavem vyvíjené metodiky pro modelování procesů.
2. Definice architektury nástroje, integrace s předchozími kroky metodiky a následným použitím výstupu nástroje.
3. Podrobný popis vybrané části k implementaci a návrh implementace nástroje.
4. Ukázkové použití a příklady.
5. Závěr a zhodnocení přínosů.

Seznam doporučené odborné literatury:

[1] John F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, ©2000
[2] Alec Sharp, Patrick McDermott: Workflow Modeling: Tools for Process Improvement and Application Development, Artech House; 2 edition (October 31, 2008)
[3] Pfleeger, Shari Lawrence, and Joanne M. Atlee. 2009. Software Engineering: Theory and Practice: Prentice Hall, ISBN 0136061699
[4] Pressman, Roger S. 2010. Software Engineering : A Practitioner's Approach. 7th ed. New York: McGraw-Hill Higher Education, ISBN 9780073375977
[5] Sommerville, Ian. 2010. Software Engineering. 9th ed, International Computer Science Series. Harlow: Addison-Wesley, ISBN 978-0137035151

Další literatura podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 01.09.2021

Datum odevzdání: 30.04.2022

doc. Ing. Petr Gajdoš, Ph.D.	prof. Ing. Jan Platoš, Ph.D.
vedoucí katedry	*děkan fakulty*

Abstrakt

Tato diplomová práce se zabývá semiformálním modelováním prostřednictvím procesní modelovací notace diagramu aktivit UML, následnou transformací tohoto diagramu do jiných procesních notací, a v neposlední řadě kombinováním semiformálního modelování ve spojení s definovanými formálními omezeními. V úvodní části je čtenáři představena i problematika modelování, kde jsou vysvětleny pojmy spojené s touto diplomovou prací. Následně je proveden průzkum aktuální situace řešené problematiky v rámci trhu. Dále následuje přiblížení řešení problematiky transformace a formálních omezení. V předposlední části je představen hlavní cíl diplomové práce, čímž bylo vytvoření nástroje, ve kterém bude možné modelovat v notaci diagramu aktivit UML, přičemž bude možné transformací převádět diagram aktivit do libovolných procesních notací a také bude umožněno omezit modelování v dané notaci prostřednictvím formálních omezení. Na závěr budou zhodnoceny dosažené výsledky a návrhy pro další postup řešení této problematiky.

Klíčová slova

modelování; diagram aktivit UML; semiformální modelování; formální omezení; metamodel; transformace; OWL

Abstract

This diploma thesis deals with semiformal modeling using process modeling notation of the UML activity diagram, subsequent transformation of this diagram into other process notations and, last but not least, combining semiformal modeling in connection with defined formal constraints. In the introductory part, the reader is introduced to the issue of modeling, which explains the concepts associated with this thesis. Subsequently, a survey of the current situation of the issue within the market is performed. Next comes the approach to the problem of transformation and formal constraints. The penultimate part presents the main goal of the thesis, which was to create a tool in which it will be possible to model in the notation of the UML activity diagram, while it will be possible to transform the activity diagram into arbitrary process notations and also allow to limit modeling in the given notation through formal constraints. In the end, the achieved results and suggestions for further progress of this issue will be evaluated.

Keywords

modeling; UML activity diagram; semiformal modeling; formal constraint; metamodel; transformation; OWL

Poďakovanie

V prvom rade by som sa chcel poďakovať mojej rodine a blízkym za podporu počas celého štúdia. Menovito chcem poďakovať vedúcemu tejto diplomovej práce Ing. Svatoplukovi Štolfovi, Ph.D. za konzultácie a vedenie pri tvorbe tejto diplomovej práce.

Obsah

Zoznam použitých skratiek a symbolov

UML	–	Unified Modeling Language
OOP	–	Objektovo orientované programovanie
OMG	–	Object Management Group
MOF	–	Meta Object Facility
OCL	–	Object Constraint Language
OWL	–	Web Ontology Language
BPMN	–	Business Process Model and Notation
IDEF	–	Integrated Definition
DFD	–	Data Flow Diagram
ERD	–	Entity Relationship Diagram
SoaML	–	Service Oriented Architecture Modeling Language
SysML	–	System Modeling Language
CMMN	–	Case Management Model and Notation
DDL	–	Data Definition Language
AWS	–	Amazon Web Services
UPMM	–	Unified Process Meta-Model
HTTP	–	Hypertext Transfer Protocol
DTO	–	Data Transfer Object
REST	–	Representational state transfer
API	–	Application programmig interface
JWT	–	JSON Web Token
DOM	–	Document Object Model
JPA	–	Java Persistence Api

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov zdrojového kódu

Kapitola 1

Úvod

V dnešnej dobe je značné množstvo úspechu firiem založené na efektívnom využívaní pracovných postupov. Pri neustálom boji s konkurenciou sú kladené obrovské nároky na kvalitu, ktorá závisí okrem iného aj od schopnosti podnikov organizovať, rozširovať, vylepšovať a hodnotiť svoje procesy. Vo všeobecnosti je možné tvrdiť, že procesy vypovedajú o charaktere podniku ako aj o jeho konečnom produkte. Narastajúcou zložitosťou týchto procesov dochádza k uvedomeniu, že ich písomná interpretácia môže pôsobiť príliš rozsiahle, chaoticky a mnohokrát nepochopiteľne, čo v ľuďoch môže vyvolať znechutenie či neochotu dané procesy dodržiavať. Tento trend následne môže viesť k zníženiu celkovej kvality finálneho produktu, nedôvere v podnik, a teda aj ku klesajúcemu záujmu o produkty podniku.

Efektívnymi nástrojmi, vďaka ktorým sme schopní čeliť tomuto problému sú procesné semiformálne modelovacie jazyky. Vizualizáciou je dosiahnuté oddelenie podstatného od nepodstatného pre popis procesu ako aj určitá miera abstrakcie. Vytvorené modely sú následne jednoducho interpretované a človek im dokáže jednoduchšie a rýchlejšie porozumieť. Pre popis procesov existuje mnoho semiformálnych modelovacích jazykov, pričom každý je populárny pre inú skupinu ľudí a môže poskytovať iný uhol pohľadu na modelovaný proces. Jedným zo spomínaných semiformálnych procesných modelovacích jazykov je modelovacia notácia diagramu aktivít UML (Unified Modeling Language).

Účelom tejto diplomovej práce je vytvoriť nástroj pre modelovanie procesov, ktorým bude možné modelovať práve v notácii diagramu aktivít jazyka UML. Tento nástroj bude okrem iného aj schopný vykonávať transformácie do iných semiformálnych procesných modelovacích jazykov, čím bude zabezpečená rozšíriteľnosť medzi širšie spektrum ľudí a taktiež bude na procesy poskytnutý iný uhol pohľadu, čo môže viesť k identifikácii nedostatkov a následnému vylepšeniu procesov. Nástroj bude tiež schopný využívať definované formálne obmedzenia prostredníctvom formálneho modelovacieho jazyka OWL pre zabezpečenie formalizácie vedúcej k dosiahnutiu exaktnej sémantickej interpretácie definovaných procesov prostredníctvom formálneho jazyka.

Práca obsahuje okrem úvodnej kapitoly ešte sedem kapitol. V kapitole 2 sú postupne rozobrané

všetky pojmy potrebné pre uvedenie čitateľa do riešenej problematiky. Sú vysvetlené pojmy ako *model* či *proces* z rôznych uhlov pohľadu pre zabezpečenie pochopenia riešenej problematiky. Následne je bližšie a z viacerých hľadísk predstavený pojem *modelovanie* a v závere kapitoly bude čitateľ oboznámený s už spomínaným modelovacím jazykom UML vrámci ktorého bude aj bližšie rozobraná modelovacia notácia diagramu aktivít.

V kapitole 3 bude predstavený rozbor aktuálne existujúcich modelovacích nástrojov vo webovom prostredí, ktoré sú svojim spôsobom podobné nástroju vyvíjanému v rámci tejto diplomovej práce. Zaujímavým poznatkom je, že žiaden z analyzovaných nástrojov neumožňuje prevádzať transformácie alebo modelovať pomocou formálnych obmedzení ale na druhej strane poskytujú veľké množstvo modelovacích notácii a tiež možnosť kolaborácie.

Kapitola 4 pojednáva o problematike transformácie, ktorej hlavnou úlohou je mapovanie medzi metamodelom modelovacieho jazyka a všeobecným metamodelom v oboch smeroch, čím bude zabezpečený prevod do iných modelovacích notácii a tiež z iných notácii do samotného diagramu aktivít. Pre tieto účely bolo nutné vytvoriť metamodel notácie diagramu aktivít UML a tiež všeobecný metamodel, ktorý predstavuje vyšší level abstrakcie, a teda izoláciu vizuálnej reprezentácie elementov konkrétneho modelovacieho jazyka. Sémantická reprezentácia elementov však zostáva nezmenená.

Nasledujúca kapitola 5 približuje čitateľovi problematiku modelovania s definovanými formálnymi obmedzeniami, ktoré zabezpečia exaktné modelovanie definovaných procesov. Prvá časť sa zaoberá pojmom *ontológia* a následne je predstavený jazyk pre modelovanie ontológii - *OWL* - ktorý je využívaný pre import formálneho obmedzenia v rámci vyvíjaného nástroja. V závere kapitoly je zhrnuté riešenie vrámci vyvíjaného nástroja.

V kapitole 6 bude predstavený samotný vyvíjaný systém. Práca na systéme prebiehala v spolupráci s iným študentom a bola riadne rozdelená, preto v rámci tejto kapitoly bude čitateľ oboznámený s detailami riešenými vrámci tejto diplomovej práce. V úvode budú špecifikované jednotlivé požiadavky na systém. Následne bude predstavená architektúra a návrh systému a na záver budú predstavené vybrané implementačné detaily.

V predposlednej kapitole 7 bude popísané vyhodnotenie samotného nástroja. V tejto časti bude prezentovaná ukážka aplikácie s príkladmi jej použitia.

Záverečná kapitola 8 obsahuje zhodnotenie dosiahnutých výsledkov a tiež návrhy pre vylepšenie a ďalší postup pri vývoji nástroja.

Kapitola 2

Pochopenie problematiky modelovania

Pri vykonávaní vedeckej činnosti sa častokrát stretávame s množstvom triviálnych i netriviálnych problémov spojených s vnímaním okolitého sveta. Narastajúcou komplexitou týchto problémov vzniká u ľudí prirodzená potreba abstrakcie. Pre potreby zovšeobecnenia je využívaný efektívny nástroj, ktorým je vedecké modelovanie.

Cieľom tejto kapitoly je uvedenie čitateľa do problematiky modelovania. Postupnými krokmi budú vysvetlené jednotlivé pojmy súvisiace s modelovaním (model, typy modelov, proces, typy procesov atď.) z rôznych uhlov pohľadu pre zabezpečenie dostatočného množstva informácii na zodpovedanie otázok: Čo je to modelovanie? Prečo modelujeme? Kedy modelujeme? Ako môžeme modelovať? Následne bude predstavený modelovací jazyk UML, ktorý je súčasťou vyvíjaného modelovacieho nástroja v rámci tejto diplomovej práce.

2.1 Model

Pred samotným vysvetlením problému modelovania je vhodné pre lepšie pochopenie uviesť, čo je to model. Tento pojem je chápaný z rôznych uhlov pohľadu.

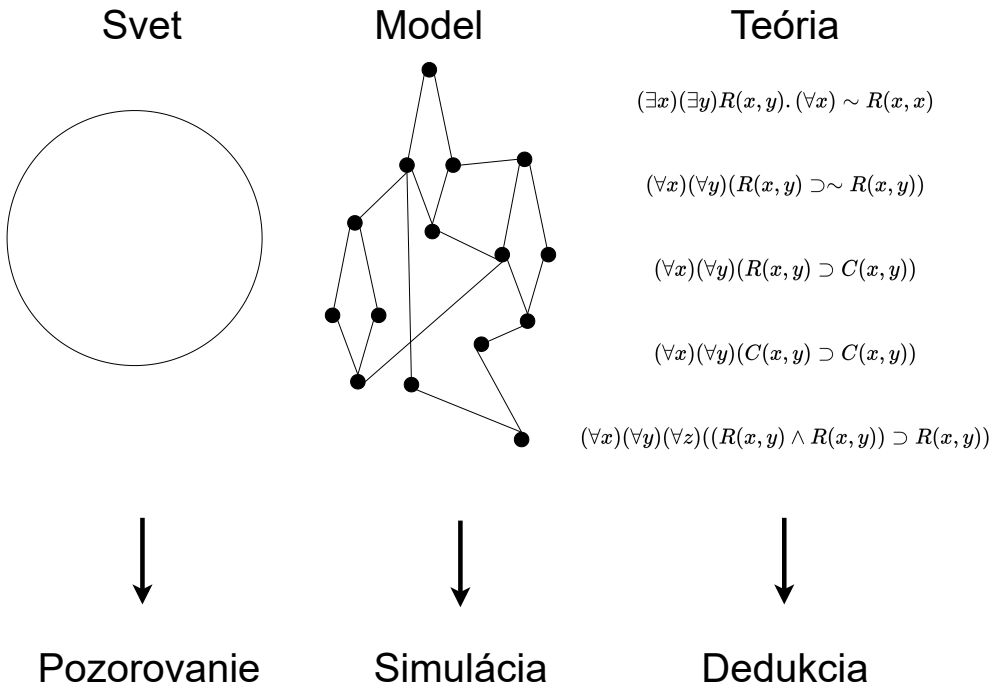
Vo všeobecnosti je model chápaný ako ľudská konštrukcia zložená z rôznych tvarov, veľkostí a štýlov, ktorá slúži pre lepšie pochopenie systémov reálneho sveta. Každý model je charakterizovaný informačným vstupom, informačným procesorom a výstupom očakávaných výsledkov. Nejde však o úplne exaktnú simuláciu reálneho sveta o čom hovorí aj slávny citát: „Všetky modely sú nesprávne ale niektoré sú použiteľné.“ - George E. P. Box.[1]

Iným uhlom pohľadu pre definíciu modelu je jeho chápanie v oblasti geovedy. Podľa článku amerického teoretického fyzika Davida O. Hestenesa môžeme model chápať ako reprezentáciu štruktúry fyzického systému a jeho vlastností. Opisuje štyri typy štruktúr, ktoré sú zložené z interných a externých komponentov:

- **Systémová štruktúra** - špecifikuje kompozíciu (interné častí systému), prostredie (externí agenti prepojení so systémom) a prepojenia (vonkajšie a vnútorné prepojenia).

- **Geometrická štruktúra** - špecifikuje polohu vzhľadom na referenčný rámec (externá geometria) a konfiguráciu (geometrické vzťahy medzi jednotlivými časťami).
- **Časová štruktúra** - špecifikuje zmenu stavových premenných.
- **Interakčná štruktúra** - špecifikuje pravidlá vyjadrujúce interakcie medzi kauzálnymi prepojeniami.[2]

Dôležitým uhlom pohľadu v rámci tejto diplomovej práce je vnímanie modelu v oblasti informatiky. Americký počitačový vedec John F. Sowa vo svojej publikácii z roku 2000 hovorí, že model predstavuje istý druh simulácie vytvorenej pri pozorovaní častí a javov reálneho sveta. Zo vzorcov a vzťahov v modeli následne vznikajú ďalšie levely abstrakcie tzv. axiómy teórie. Tento koncept Sowa nazýva ako „*Tri zdroje poznania sveta*“[3].



Obr. 2.1: Tri zdroje poznania sveta[3]

Vhodnú interpretáciu pre pojem *model* tiež poskytuje aj pohľad A. Sharpa a P. McDermotta v publikácii z roku 2008, kde popisujú model ako reprezentáciu témy, ktorá musí spĺňať nasledujúce pravidlá:

1. Predstavuje abstrakciou nejakého objektu alebo javu reálneho sveta.
2. Zvýrazňuje aspekty, ktoré sú potrebné pre zachytenie podstatných objektov a javov, a zároveň zakrýva nepodstatné detaily.

3. Využíva definované konvencie pre dosiahnutie čo najväčšej presnosti a zastrešenie detailov témy.
4. Má jasný východiskový bod a cieľ.
5. Používa optimálny počet grafických prvkov pre zaistenie lepšej pochopiteľnosti.

Podľa Sharpa a McDermotta je manipulácia s modelom pohodlnejšia, lacnejšia a bezpečnejšia, než práca s objektami reálneho sveta. Autori tiež rozdeľujú modely do dvoch skupín. Prvou skupinou sú modely, ktoré predstavujú vizuálnu, zmenšenú reprezentáciu reálneho objektu. Tieto modely nazývajú ako **ikonické modely**. Príkladom ikonického modelu môže byť zmenšený model lietadla. Druhou skupinou sú modely predstavujúce koncepty, ktoré sú v skutočnosti len veľmi ťažko pozorovateľné. Ide o tzv. **symbolické modely**. Príkladom symbolického modelu je *matematický model ekonomiky.* [4]

2.1.1 Modely systému

Okrem spomínaného delenia modelov podľa Sharpa a McDemotta je možné modely kategorizovať aj z inej perspektívy. Jednou z nich je kategória modelov systému, kde každý typ modelu predstavuje iný uhol pohľadu na vyvíjaný systém:

- **Kontextové modely** - modely určené na definíciu hraníc systému. Využívajú sa vo fáze špecifikácie systému na základe systémových požiadavkov a zdrojov vyhradených pre systém. Typickým predstaviteľom modelu definujúceho systémový kontext je aktivitný diagram.
- **Interakčné modely** - modely popisujúce interakcie medzi systémom a užívateľom pre špecifikovanie požiadavkov, interakciu systému s inými systémami pre zachytenie možných komunikačných problémov a nakoniec interakciu medzi jednotlivými systémovými komponentami pre overenie požadovaného výkonu a spoľahlivosti systému. Pre dané účely je využívaný diagram prípadov použitia a sekvenčný diagram.
- **Štrukturálne modely** - modely zobrazujúce organizáciu systémových komponent a ich vzťahov. Statické štrukturálne modely popisujú štruktúru návrhu systému. Naopak dynamické štrukturálne modely sú zamerané na popis systému za behu. Využívajú sa vo fáze návrhu systémovej architektúry za použitia triedneho diagramu.
- **Modely chovania** - modely dynamického chovania behu systému, ktoré simulujú čo sa stane (alebo čo by sa malo stať) vplyvom prostredia na daný systém. Vplyv prostredia predstavujú dáta, s ktorými model pracuje a akcie, ktoré sú v modeli simulované. Predstaviteľmi modelov chovania sú tzv. diagramy toku dát (z ang. data-flow diagrams).

Modely systému sú používané pri inžinierstve požiadavkov pre vývoj nových požiadavkov, prehľadný popis systému softvérovým inžinierom pri implementácii, a tiež pre dokumentáciu systémových štruktúr a operácii. [5]

2.1.2 Procesné modely

Poskytujú efektívny a systematický postup vývoja softvéru prostredníctvom overených metód a zabraňujú vzniku chaosu. Sú vytvárané softvérovými inžiniermi na základe potrieb pri vývoji softvérového produktu, a následne opakovane využívané pre zachovanie kontroly, stability a organizácie aktivít. Existuje niekoľko typov procesných modelov: [6]

- **Vodopádový model** - ide o historicky prvý používaný prístup v softvérovom inžinierstve. Jedná sa o sekvenčný model poskytujúci postup vývoja softvéru. Skladá sa zo špecifikácie požiadavkov, návrhu, implementácie, testovania a údržby.

- **V model** - ide o variant vodopádového modelu, ktorého cieľom je zobrazenie vzťahov medzi testovacími aktivitami s návrhom. Každá fáza návrhu je spojená s testovacou aktivitou.

- **Operačný model** - požiadavky sú prezentované prostredníctvom chovania systému, čo umožňuje, že ich dôsledky je možno vyhodnotiť ešte pred samotným návrhom. Oproti vodopádovému modelu sú aktivity návrhu a implementácie zlúčené.

- **Transformačný model** - eliminuje príležitosť vzniku chyby vynechaním niekoľkých hlavných krokov vývoja, čo je zabezpečené aplikovaním série transformácii na zmenu špecifikácie do podoby kompletného systému. Pre jeho použitie je však potrebná formálna špecifikácia.

- **Inkrementálny model** - systémové požiadavky sú rozdelené do menších celkov - spustiteľných subsystémov. Vydaním novej verzie pribúda požadovaná funkcionalita systému.

- **Iteratívny model** - systém je dodaný ako celok na začiatku, pričom každé nové vydanie sú modifikované existujúce subsystémy.

- **Špirálový model** - definuje ciele, alternatívy a vzťahy prostredníctvom požiadavkov a úvodného plánu. Následne kombinuje vývojové aktivity a rizikový manažment s cieľom eliminácie rizikových faktorov. [7]

2.2 Proces

Pri objasnení pojmu model bol čitateľ oboznámený s viacerými druhmi modelov. Jedným z nich boli aj procesné modely. Pre ďalší postup v chápaní riešenej problematiky v rámci tejto diplomovej práce je vhodné charakterizovať pojem proces.

Človek sa s procesmi častokrát stretáva aj bez toho, aby si to uvedomoval. Už samotná príprava raňajok, upratanie izby, či tankovanie paliva do automobilu sa dá považovať za proces. Môžeme teda tvrdiť, že proces predstavuje postupnosť krokov a rozhodnutí vedúcich k dosiahnutiu určitého cieľa. [8]

Podľa S. L. Pfleegerovej a J. M. Atleeovej môžeme proces definovať ako množinu usporiadaných úloh. V publikácii z roku 2009 tiež hovoria o procese ako o postupnosti krokov, ktoré zahŕňajú aktivity, obmedzenia a zdroje pre dosiahnutie očakávaného výstupu. [7]

Chronologicky - metódou od všeobecného ku konkrétnemu - bude čitateľ oboznámený so špecifickými druhmi procesov pre dosiahnutie väčšieho prehĺbenia sa do riešenej problematiky. Prvým uhlom pohľadu bude uvedenie procesu v oblasti informatiky - **softvérový proces**. Okrem toho bude tiež objasnený pojem **podnikový proces**, ktorý úzko súvisí s vyvíjaným nástrojom pre modelovanie.

2.2.1 Softvérový proces

Súbor navzájom súvisiacich aktivít vedúcich k vzniku softvérového produktu. Roger S. Pressman a Bruce R. Maxim popisujú softvérový proces z technického hľadiska ako rámec pre aktivity, akcie a úlohy, ktoré sú potrebné pre vytvorenie vysokokvalitného softvéru. V spoločnej publikácii z roku 2014 tiež tvrdia, že softvérový proces definuje prístup, ktorý je využívaný pri navrhovaní softvéru. [6][5]

2.2.2 Podnikový proces

Vyvíjaný nástroj v rámci tejto diplomovej práce slúži primárne pre modelovanie podnikových procesov. Podnikový proces predstavuje súbor logicky súvisiacich úloh vykonávaných pre dosiahnutie organizačného cieľa .[5] [9]

Po uvedení a popísaní pojmov model a proces z rôznych hľadísk bude čitateľ oboznámený s problematikou modelovania.

2.3 Modelovanie

Pojem modelovanie je vnímaný z rôznych uhlov pohľadu. Vo všeobecnosti je chápané ako vedecká činnosť, ktorej cieľom je uľahčiť vizualizáciu, pochopenie, kvantifikáciu, definovanie a simuláciu častí alebo funkcií reálneho sveta. Proces vedeckého modelovania vedie ku vygenerovaniu modelu ako konceptuálnej reprezentácie nejakého javu. [10]

Zaujímavý pohľad poskytuje aj vyššie spomínaný americký počítačový vedec John F. Sowa, ktorý popisuje modelovanie ako nepriamy prístup, ktorý je bežne využívaný inžiniermi pre konštrukciu modelu existujúceho alebo navrhovaného systému. Obvykle ide o systémy, ktorých opätovné vytvorenie v reálnom svete je drahé, neetické alebo nebezpečné. [3].

2.3.1 Typy modelovania

Delenie typov modelovania je možné opäť vnímať z rôznych hľadísk. V nasledujúcich častiach tejto diplomovej práce budú porovnané typy modelovania na rôznych úrovniach.

2.3.1.1 Statické/dynamické modelovanie

- **Statické modelovanie** - slúži pre vytvorenie modelov, ktoré zobrazujú štruktúru navrhovaného systému. Popisuje tiež na aké výstupy modelu sú vstupy transformované, pričom odhaduje využitie zdrojov v čase kompilácie.
- **Dynamické modelovanie** - oproti statickému modelovaniu je jeho cieľom zobrazenie štruktúry systému za behu, pričom popisuje akým spôsobom sú vstupy transformované na výstupy v čase. Okrem toho je jeho cieľom aj predpoveď výkonu riešenej úlohy.[11][5][7]

2.3.1.2 Neformálne/semiformálne/formálne modelovanie

- **Neformálne modelovanie** - základom modelovania je prirodzený jazyk, jednoduché obrázky, storyboardy atd. Nie je jednoznačne definovaná syntax a sémantika modelovania.
- **Formálne modelovanie** - jeho základom je presne a jednoznačne definovaná syntax a sémantika modelovacieho jazyka. Typickými predstaviteľmi sú jazyky: OCL (Object Constraint Language), OWL (Web Ontology Language) a iné.
- **Semiformálne modelovanie** - prienik medzi formálnym a neformálnym modelovaním. Ide o modelovanie, ktoré má jednoznačne definovanú syntax, ale neformálnu alebo čiastočne formálnu sémantiku. Medzi predstaviteľov semiformálneho modelovania patria jazyky: UML, BPMN (Business Process Model and Notation), IDEF (Integrated Definition) a iné.[12]

2.4 UML

Jazyk UML predstavuje štandardnú koncepciu modelovania. Ide o súbor grafických notácií, ktoré sú využívané pri vývoji softvéru vo fáze analýzy a návrhu pre popis vyvíjaného systému z rôznych uhlov pohľadu. Okrem iného je tiež definovaný ako štandardný nástroj pre špecifikáciu, konštrukciu, dokumentáciu či vizualizáciu systémových artefaktov. [13] [14]

Podľa aktuálne najnovšej oficiálnej špecifikácie z roku 2017 je cieľom UML poskytnúť systémovým architektom, softvérovým inžinierom a vývojárom nástroje pre analýzu, návrh a implementáciu softvérových systémov, ako aj nástroj pre modelovanie procesov. [15]

2.4.1 História

Príchodom OOP (Objektovo orientované programovanie) bol hľadaný spôsob, ako vnímať vyvíjaný systém z rôznych uhlov pohľadu. Na tento podnet zareagovala firma Rational Software, ktorá roku 1994 začala s vývojom štandardu UML. Príchodom verzie UML 1.0 v roku 1997 je štandard rozvíjaný organizáciou OMG (Object Management Group), zloženej z firiem ako IBM, Rational Software, Oracle, Microsoft a iné. Výrazná zmena štandardu bola zaznamenaná príchodom verzie 2.0 v roku

2001. Aktuálne je využívaná verzia štandardu 2.5.1 z roku 2017, ktorá je súčasťou vyvíjaného nástroja v rámci tejto diplomovej práce. [13][14][15][16]

2.4.2 Metamodel

V predchádzajúcich častiach tejto diplomovej práce, bol čitateľ oboznámený s pojmom model. Kým model predstavuje abstrakciu častí a javov reálneho sveta, metamodel predstavuje abstrakciu samotného modelu v danom modelovacom jazyku. Inými slovami, model je možné vnímať ako grafickú reprezentáciu metamodelu daného modelovacieho jazyka.

Podľa oficiálnej špecifikácie jazyka UML metamodel definuje abstraktnú syntax UML. Ide o množinu konceptov modelovania, ich atribútov, vzťahov a pravidiel, ktoré kombinujú tieto koncepty pri vytváraní čiastočných alebo kompletných UML modelov. Okrem iného spoločnosť OMG špecifikuje aj štandard MOF (Meta Object Facility) pre generovanie metamodelu, ktorý je inštanciou tzv. meta-metamodelu. Model, metamodel a meta-metamodel predstavujú tri úrovne - tzv. **levely abstrakcie**. [15] [17] [16]

2.4.3 Typy UML diagramov

V rámci UML sú rozoznávané dve skupiny diagramov: **štrukturálne diagramy** a **diagramy chovania**. Štrukturálne diagramy popisujú statickú štruktúru systému a vzťahy medzi jednotlivými časťami systému. Diagramy chovania popisujú dynamické chovanie objektov systému v čase. Nasledujúca tabuľka predstavuje niektorých zástupcov oboch skupín:

Tabuľka 2.1: Typy UML diagramov

Štrukturálne diagramy	**Diagramy chovania**
- Diagram tried - Diagram komponentov - Diagram nasadenia - Objektový diagram - Diagram balíčkov	- Diagram aktivít - Diagram prípadov použitia - Stavový diagram - Sekvenčný diagram

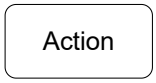
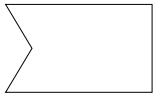
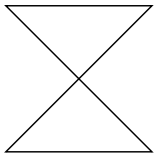
Pre ďalšie objasnenie problematiky modelovania v rámci diplomovej práce nie je nutný popis všetkých druhov diagramov. Keďže sa však práca zaoberá problematikou transformácie a formálnych obmedzení diagramu aktivít, bude tento druh diagramu popísaný v nasledujúcej časti.

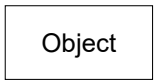
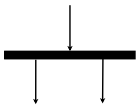
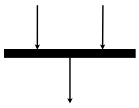
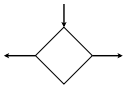
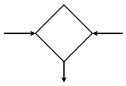
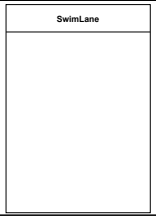
2.4.4 Diagram aktivít

Diagram aktivít zobrazuje sekvenciu akcií alebo riadiaceho toku systému od počiatočného uzla po koncový uzol. Je využívaný pri modelovaní podnikových procesov pre detailný popis procesu. [18][19]

Nasledujúca tabuľka zobrazuje názov, notačný zápis a popis vybraných elementov, využitých pri vývoji nástroja na modelovanie v rámci diplomovej práce:

Tabuľka 2.2: Popis vybraných elementov diagramu aktivít

Názov elementu	**Notačný zápis**	**Popis elementu**
Počiatočný uzol		- Začiatok aktivity. - Použitie šipky je možné iba v smere von z uzla.
Koncový uzol		- Konečný krok aktivity. - Použitie šipky je možné iba v smere do uzla.
Akcia		- Krok aktivity, ktorý je prevádzaný používateľom alebo softvérom.
Spúšťacia akcia		- Naznačuje akciu odoslania. - Zvyčajne býva v spojení s príjmovou akciou.
Príjmová akcia		- Naznačuje akciu príjmu. - Zvyčajne býva v spojení so spúšťacou akciou.
Časová udalosť		- Predstavuje udalosť, ktorá zastaví tok aktivity na určitý čas.

Objekt		-Indukuje, že inštancia klasifikátoru, ktorá objekt reprezentuje, je dostupná v danom bode aktivity.
Vetvenie		- Rozdeľuje tok aktivity na viac vzájomne konkurenčných vetví,pričom pri priebehu aktivity sú paralelne prevádzané všetky rozdelené vetvy.
Spojenie		- Zlučuje tok rozdelených vzájomne konkurenčných vetví aktivity do jednej vetvy.
Rozhodnutie		- Rozdeľuje tok aktivity na viac navzájom vylučujúcich sa vetví,pričom pri priebehu aktivity je vybraná práve jedna vetva.
Zlúčenie		- Zlučuje tok rozdelených navzájom vylučujúcich sa vetví aktivity do jednej vetvy.
Sprievodca	[Guide]	- Slúži na popis elementov.
Komentár		- Poznámka k elementu.
Riadiaca šipka		- Spojovník dvoch uzlov. - Orientovaná hrana, ktorá určuje smer toku aktivity. - Využíva sa v spojení so všetkými uzlami mimo objektov.
Objektová šipka		- Spojovník objektu a uzla.
Dráha		- Slúži na logickú organizáciu elementov.

Kapitola 3

Súčasný stav na trhu

V rámci tejto kapitoly bude čitateľ oboznámený so súčasnými nástrojmi na trhu, ktoré umožňujú modelovanie v notácií UML a nachádzajú sa vo webovom prostredí.

3.1 Visual Paradigm

Jeden zo súčasne najpopulárnejších poskytovateľov nástrojov pre návrh softvéru a riadenie projektov. Umožňuje používateľovi vytvárať diagramy v notáciach UML, BPMN, ArchiMate, DFD (Data Flow Diagram), ERD (Entity Relationship Diagram), SoaML (Service Oriented Architecture Modeling Language), SysML (System Modeling Language) a CMMN (Case Management Model and Notation) pričom poskytuje mnoho formátovacích možností, znovuvyužiteľnosť komponentov, prispôsobenie vlastností a iné. Okrem toho pomáha vylepšovať efektivitu a produktivitu podniku prostredníctvom nástrojov podporujúcich strategické plánovanie, procesného návrhu či procesnej analýzy. Pre dosiahnutie lepších výsledkov poskytuje podporu projektového riadenia prostredníctvom nástrojov na vytváranie implementačných plánov, roadmap a pod. Vďaka webovej verzii a cloud riešení nieje nutná inštalácia. Podporuje tiež generovanie kódu z diagramu a opačne vo viac ako desiatich programovacích jazykoch. Z pohľadu databáz tiež zabezpečuje prevod databázového modelu na DDL (Data Definition Language) a opačne. Vďaka podpory kolaborácie sú používatelia tiež schopní pracovať spoločne na jednom projekte.[20][21]

3.2 Lucidchart

Inteligentná aplikácia na vytváranie diagramov, kde má používateľ na výber z viac ako tisíc šablón v rôznych semiformálnych modelovacích jazykoch. Jedným z hlavných benefitov nástroja je funkcia automatického generovania diagramov na základe dát. Pre organizácie spoločnosť ponúka podporu v oblasti školení, poradenstvo pri používateľských problémoch, vysokú škálovateľnosť a bezpečnosť. Okrem toho aplikácia tiež ponúka administrátorské rozhranie pre zabezpečenie kontroly nad pou-

žívateľskými účtami, správu dokumentov a nastavenie obmedzení domény, čím je zaistené súkromie organizácie. Významnú úlohu zohráva aj podpora integrácie aplikácii ako sú Google, Slack, Atlassian, Salesforce, Asana a iné. V kombinácii s aplikáciou Lucidpark tiež aplikácia pokrýva možnosti tímovej kolaborácie.[22][23]

3.3 Drawio

Najlepší nástroj na trhu spoločnosti Atlassian, ktorý ponúka jednoduché a intuitívne používateľské rozhranie na tvorbu diagramov ako sú vývojové diagramy, diagramy notácie BPMN, diagramy notácie UML, inžinierske diagramy, sieťové diagramy, sekvenčné diagramy, pojmové mapy a iné. Kolaborácia je zabezpečená využitím cloudového riešenia Confluence, vďaka čomu sú tímy neustále synchronizované. Používatelia sú tiež schopní vytvárať šablóny pre efektívne využívanie nástroja, importovať a exportovať diagramy v rôznych formátoch či prepojiť aplikáciu so službami OneDrive a GoogleDrive. Spoločnosť z oblasti podpory tiež ponúka množstvo inštruktážnych videí, interaktívnych príručiek či chronologických manuálov.[24][25][26]

3.4 Creately

Predstavuje prostredie pokrývajúce plánovanie a vedomostný manažment. Je charakterizované tým, že prepája vizuálne objekty s dátami, a taktiež podporuje kolaboráciu tímov, čím vytvára vhodné podmienky pre realizáciu projektov. Okrem toho tiež ponúka nekonečné plátno pre modelovanie, vlastnú databázu, podporu pre dokumentovanie a poznámkovanie modelov a mnoho ďalšieho.[27] Nasledujúca tabuľka demonštruje, prečo je vhodnejší výber nástroja Creately oproti Lucidchart:

Tabuľka 3.1: Porovnanie nástroja Creately s Lucidchart [28]

	Creately	**Lucidchart**
Kolaborácia	- Výkonnejšia tímová spolupráca v reálnom čase. - Integrovaný nástroj pre videohovory.	- Chýbajú komunikačné funkcie ako sú audio a video chat.
Dáta	- Umožňuje migrovať údaje z akéhokoľvek zdroja, vizualizovať ich prostredníctvom tvarov a udržiavať ich synchronizované. - K dispozícii sú aj dodatočné poznámky pre upresnenie kontextu.	- Umožňuje prepojiť dátové súbory z externých zdrojov s diagramami a udržiavať ich synchronizované.

Jednoduchosť použitia	- Inteligentné skratky. - Pokročilé možnosti formátovania. - Kontextové panely nástrojov. - Prednastavené farebné témy a štýly.	- Zložitejšie na naučenie sa. - Obmedzené funkcie, čo spôsobuje, že vizualizácia zložitejších systémov je zdĺhavý proces.
Výstup	- Profesionálne predvoľby farieb a štýlov. - Inteligentné formátovanie. - Vstavané vyhľadávanie Google obrázkov.	- Nedostatočné farebné palety a možnosti formátovania.

3.5 Edrawsoft

Zástupca platených aplikácii, ktorý však pre svojich potencionálnych zákazníkov ponúka aj časovo limitovanú skúšobnú verziu. Aplikácia ponúka diagramové riešenia určené pre tímy z oblastí marketingu, inžinierstva, dizajnu a iných. Užívatelia sú schopní tiež spolupracovať na spoločných projektoch, pričom je možné využívať rôzne zariadenia s rôznymi operačnými systémami. Hlavnou výhodou aplikácie je tiež bezpečnosť, ktorá je zabezpečená prostredníctvom cloudového riešenia AWS (Amazon Web Services), dodržaním GDPR (General Data Protection Regulation) štandardov a využitím najvyššieho levelu 256-bit SSL (Secure Sockets Layer) šifrovania.[29]

Všetky spomínané nástroje však neponúkajú možnosť transformácie do iných notácií a ani možnosť modelovania s využitím formálnych obmedzení.

Kapitola 4

Transformácia

Jedným z hlavných cieľov tejto diplomovej práce je dokázanie konceptu transformácie procesnej notácie diagramu aktivít UML do iných notácii, čím bude zabezpečená lepšia čitateľnosť, pochopiteľnosť a flexibilita abstrakcie procesov pre širšie spoločenstvo ľudí.

V tejto kapitole bude čitateľ oboznámený s konceptom transformácie. Prvá časť kapitoly je venovaná vytvoreniu metamodelu modelovacieho jazyka diagramu aktivít UML. Následne bude predstavené vytvorenie všeobecného metamodelu, do ktorého bude metamodel diagramu aktivít UML mapovaný. Samotné mapovanie metamodelu bude bližšie popísané v poslednej časti tejto kapitoly.

4.1 Vytvorenie metamodelu diagramu aktivít UML

Metamodel (viď podkapitola 2.4.2), predstavuje abstrakciu modelu v danom modelovacom jazyku a samotný model je abstrakciou nejakého javu. Preto je možné tvrdiť, že metamodel je model modelovacieho jazyka.

Prvým krokom pri vývoji nástroja na modelovanie v notácii diagramu aktivít UML ako dôkazu koncepcie transformácie do iných notácii, bola konštrukcia zjednodušenej verzie metamodelu na základe aktuálnej špecifikácie jazyka UML (verzia 2.5.1 [15]).

Dôvodom zjednodušenia metamodelu bola jeho pomerne veľká rozsiahlosť, a tiež nadbytočnosť niektorých elementov pre zabezpečenie plnohodnotného modelovania v notácii diagramu aktivít jazyka UML. Vyradené boli najmä elementy, ktoré sú využívané v iných notáciach jazyka UML a niekedy zvyknú byť (nie však často) používané. Pre príklad takéhoto elementu uvediem asociatívnu šipku. Vizuálny prehľad jednotlivých vybraných elementov je možné vidieť v tabuľke 2.2. V nasledujúcej časti budú bližšie špecifikované jednotlivé prvky skonštruovaného metamodelu a ich náväznosti na špecifikáciu uml (viď. [15]).

- **Element** - Abstraktný predok všetkých elementov notácie UML. Slúži na zhromaždenie spoločných vlastností a chovania všetkých elementov, pričom je vo vzťahu s *Komentárom*, pretože

jeden *Element* môže mať niekoľko *Komentárov*. Okrem toho je aj vo vzťahu s *Dráhou*, pretože *Element* môže byť súčasťou *Dráhy*. Vrámci špecifikácie jazyka UML ide o *Element* viditeľný na obrázku *7.1 Root* a bližšie popísaný v časti 7.2.3.1 (viď. [15]).

- **Komentár** - Je *Elementom*, a zároveň je vo vzťahu s *Elementom*, pretože prináleží práve jednému elementu. Slúži ako poznámka k elementu. Náväznosť *Komentára* na špecifikáciu možno vidieť v časti 7.2.3.2 (viď. [15]).

- **Dráha** - Je *Elementom*, a zároveň je vo vzťahu s *Elementom*, pretože môže mať v sebe niekoľko elementov. Je využívaná pre logickú organizáciu elementov. V špecifikácii jazyka UML *Dráha* predstavuje *Aktivitné partície využívajúce dráhy*, ktoré sú popísané v časti 15.6.3.1 sémanticky a notačne v časti 15.6.4.1 (viď. [15]).

- **Sprievodca** - Je *Elementom*, a zároveň je vo vzťahu s *Časovou udalosťou*, *Aktivitnou šipkou*, *Počiatočným uzlom*, *Koncovým uzlom*, *Vetviacím/Spojovacím uzlom* a *Rozhodovacím/zlúčovacím uzlom*, pretože patrí práve jednému z týchto elementov. Slúži ako sprievodný popis pre spomínané elementy. Vrámci špecifikácie jazyka UML ide o *Špecifikáciu hodnoty* popísanú v kapitole *8* (viď. [15]).

- **Aktivitný uzol** - Abstraktný predok všetkých elementov, ktoré je možné spojiť šipkou. Sám o sebe je *Elementom*, pričom je vo vzťahu s *Aktivitnou šipkou*, pretože môže vystupovať ako zdrojový element alebo ako cieľový element šipky. Vrámci UML špecifikácie je bližšie popísaný v časti 15.2.3.2 (viď. [15]).

- **Aktivitná šipka** - Abstraktný predok všetkých šipiek diagramu aktivít. Je *Elementom*, a zároveň je vo vzťahu s *Aktivitným uzlom*, pretože šipka môže spájať dva elementy, pričom jeden je zdrojový a druhý cieľový uzol. Bližšia špecifikáciu je možné nájsť v oficiálnej špecifikácii v časti 15.2.3.3 (viď. [15]). Vzťah *Aktivitného uzla* a *Aktivitnej šipky* je možné vidieť na obrázku 15.1 (viď. [15]).

- **Objektový uzol** - Element, ktorý je *Aktivitným uzlom* a indikuje, že inštancia klasifikátoru, ktorá ho reprezentuje je dostupná v danom okamihu aktivity. V špecifikácii UML je popísaný vrámci časti 15.4 (viď. [15]). Ako je možné vidieť, špecifikácie popisuje viaceré typy objektových uzlov. Pre potreby modelovacieho nástroja postačovala základná špecifikácia objektového uzla.

- **Akcia** - Ide o element, ktorý je *Aktivitným uzlom* a predstavuje krok aktivity. v špecifikácii UML je možné vidieť popis *Akcie* v časti 16.2.3.1 sémanticky a notačne v časti 16.2.4.1 (viď. [15]).

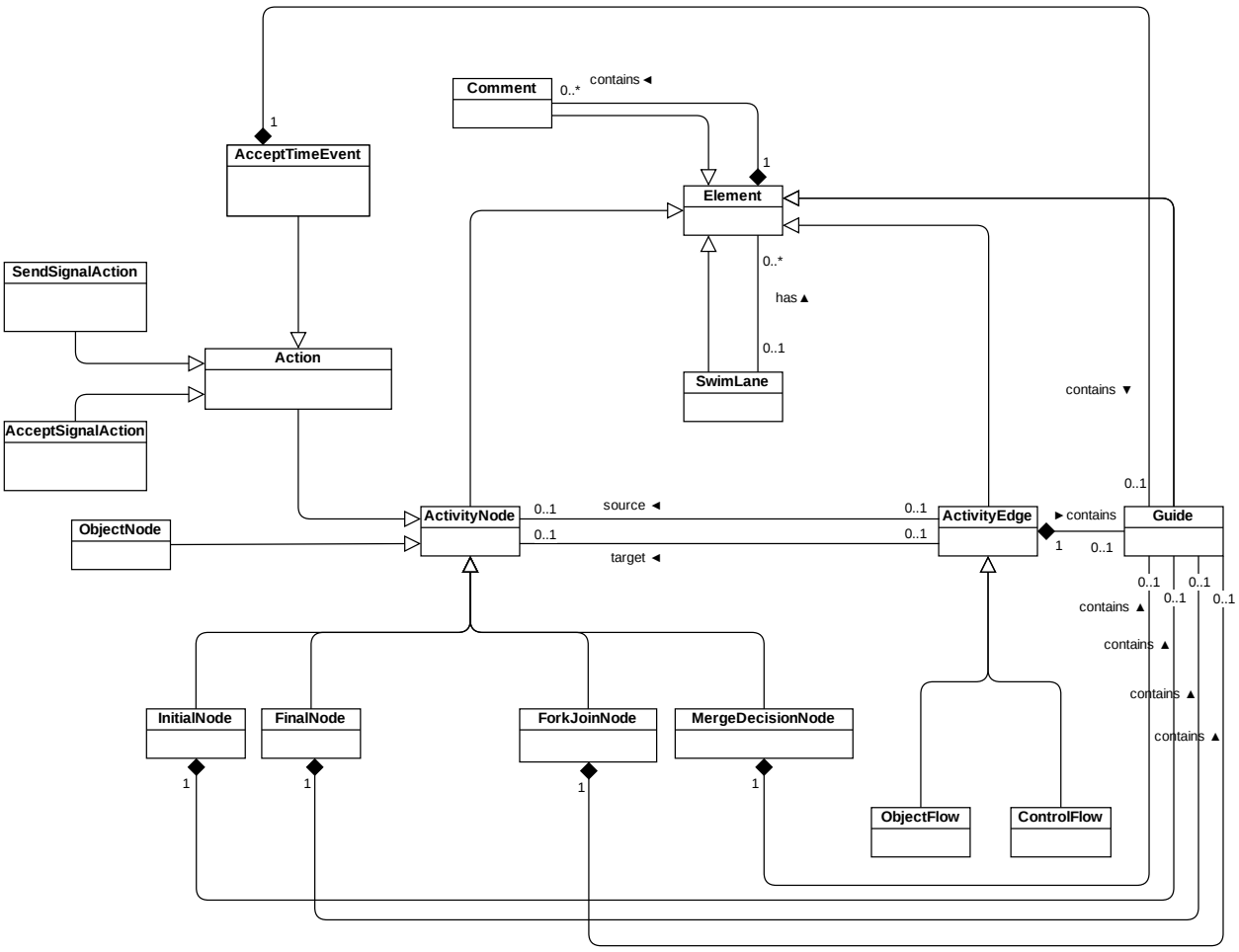
- **Spúšťacia akcia** - Je špecifickým druhom *Akcie*, ktorý slúži pre naznačenie odosielacieho kroku aktivity. V špecifikácii jazyka UML je popísaná vrámci v časti 16.3.5.2 (viď. [15]).

- **Príjmová akcia** - Špecifický druh *Akcie*, ktorý naznačuje akciu príjmu. V špecifikácii jazyka UML je popísaná vrámci v časti 16.10.3.1 sémanticky a notačne v časti 16.10.5.1 (viď. [15]).
- **Časová udalosť** - Ide o konkrétnejší druh *Akcie* reprezentujúcu udalosť, ktorá pozastavuje tok aktivity na určitý čas. Podľa špecifikácie jazyka UML môže tiež nahradzovať *Príjmovú akciu* s vyjadrením časového trvania (viď. časť 16.10.4 v [15]).
- **Počiatočný uzol** - Je *Aktivitným uzlom* ktorý predstavuje začiatok aktivity. Okrem toho je vo vzťahu so *Sprievodcom*, pretože môže obsahovať sprievodcu. Bližší popis je možný vidieť v oficiálnej špecifikácii v časti 15.3.3.1 (viď. [15]).
- **Koncový uzol** - Je *Aktivitným uzlom* ktorý predstavuje koniec aktivity. Tiež je vo vzťahu so *Sprievodcom*, pretože ho môže obsahovať. Bližší popis je možný vidieť v oficiálnej špecifikácii v časti 15.3.3.2 (viď. [15]).
- **Vetviaci/spojovací uzol** - Je *Aktivitným uzlom* rozdeľujúcim alebo zlučujúcim viacero paralelne plynúcich vetví. Je vo vzťahu so *Sprievodcom*, pretože ho môže obsahovať. Bližší popis je možný vidieť v oficiálnej špecifikácii v časti 15.3.3.3 a 15.3.3.4 (viď. [15]).
- **Rozhodovací/zlučovací uzol** - Je *Aktivitným uzlom* ktorý rozdeľuje tok aktivity do viacero navzájom vylučujúcich sa vetví alebo zlučuje rozdelený tok aktivity. Je vo vzťahu so *Sprievodcom*, pretože ho môže obsahovať. Bližší popis je možný vidieť v oficiálnej špecifikácii v časti 15.3.3.5 a 15.3.3.6 (viď. [15]).
- **Objektová šipka** - Druh *Aktivitnej šipky* spájajúci objekty s iným elementom.
- **Riadiaca šipka** - Druh *Aktivitnej šipky* spájajúci neobjektové uzly. *Objektová* a *Riadiaca šipka* sú v špecifikácii jazyka UML popísané v časti 15.2.2.3 a tiež na obrázku 15.1 (viď. [15]).

Element, *Aktivitný uzol* a *Aktivitná šipka* sú abstraktní predkovia, ktorí nemajú konkrétnu vizuálnu reprezentáciu ale spájajú elementy, s podobnými vlastnosťami. Všetky ostatné elementy sú charakterizované notačným zápisom (viď tabuľka 2.2).

4.2 Vytvorenie všeobecného metamodelu

Samotný koncept transformácie z ktorého bolo vychádzané, a ktorý bol osvedčený v rámci tejto diplomovej práce popisuje vo svojej dizertačnej práci z roku 2015 M. A. Košinár. Účelom tohto konceptu je mapovanie metamodelu konkrétnej modelovacej notácie do všeobecnejšej podoby (vyššieho levelu abstrakcie) s cieľom zachovania syntaxe a sémantiky mapovaného modelu. Následne je možné zo všeobecnej reprezentácie modelu daný model namapovať do ľubovolnej procesnej notácie. Podmienkou je, aby metamodel modelovacej notácie, do ktorej ideme vykonať transformáciu, bolo tiež

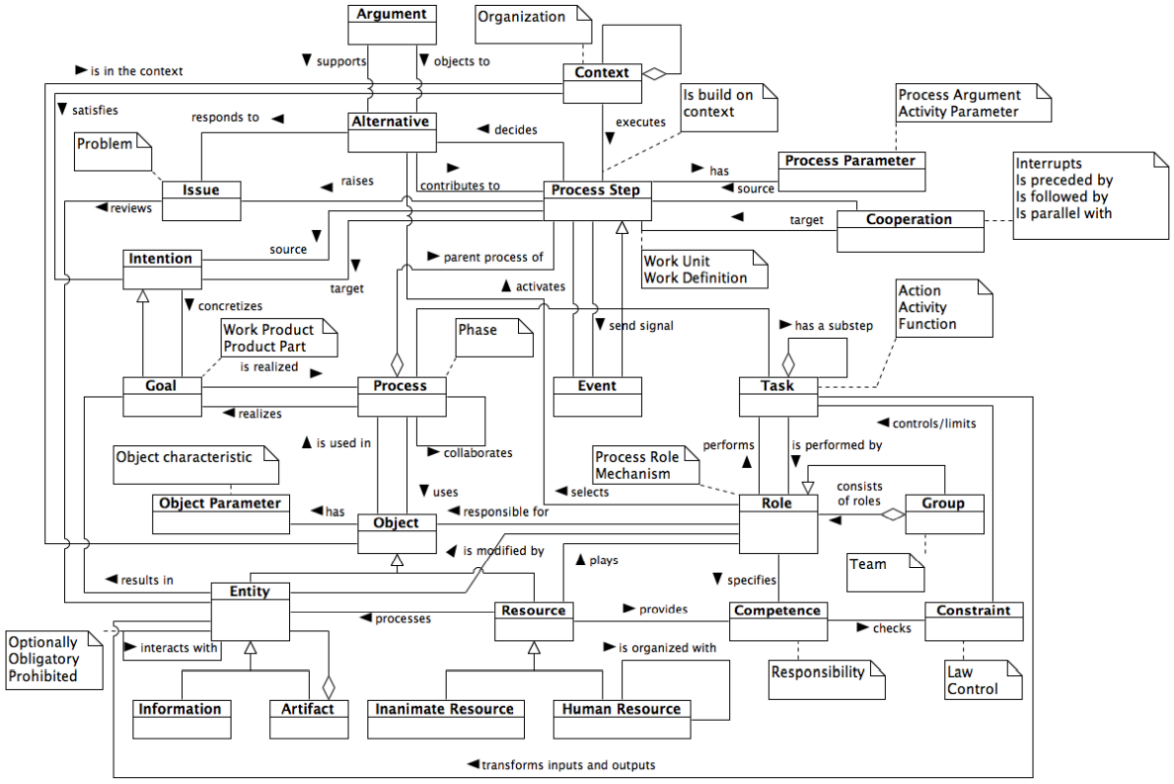


Obr. 4.1: Metamodel diagramu aktivít UML

možné mapovať do všeobecného metamodelu. Pre tieto účely slúži ako reprezentácia všeobecného metamodelu UPMM (Unified Process Meta-Model).[12]

UPMM ako všeobecný procesný metamodel predstavuje veľmi podrobnú procesnú modelovaciu definíciu. Mnohé elementy v rámci UPMM boli pre potreby dôkazu konceptu transformácie v rámci tejto diplomovej práce zanedbateľné a naopak, niektoré elementy bolo potrebné upresniť pre konsenzuálne zachovanie levelu abstrakcie všeobecného metamodelu, a rovnako syntaxe a sémantiky transformovaného modelu.

Pre tieto účely bol vytvorený nový všeobecný metamodel, ktorý vychádza z UPMM. Vývoj nového všeobecného metamodelu prebiehal v spolupráci so študentom pracujúcom na transformácii procesnej notácie BPMN pre zabezpečenie pokrytia mapovania elementov oboch notácii a taktiež dosiahnutie čo najväčšieho zachovania syntaktickej a sémantickej úrovne transformovaného modelu. V nasledujúcej časti diplomovej práce budú popísané jednotlivé elementy vytvoreného všeobecného metamodelu.



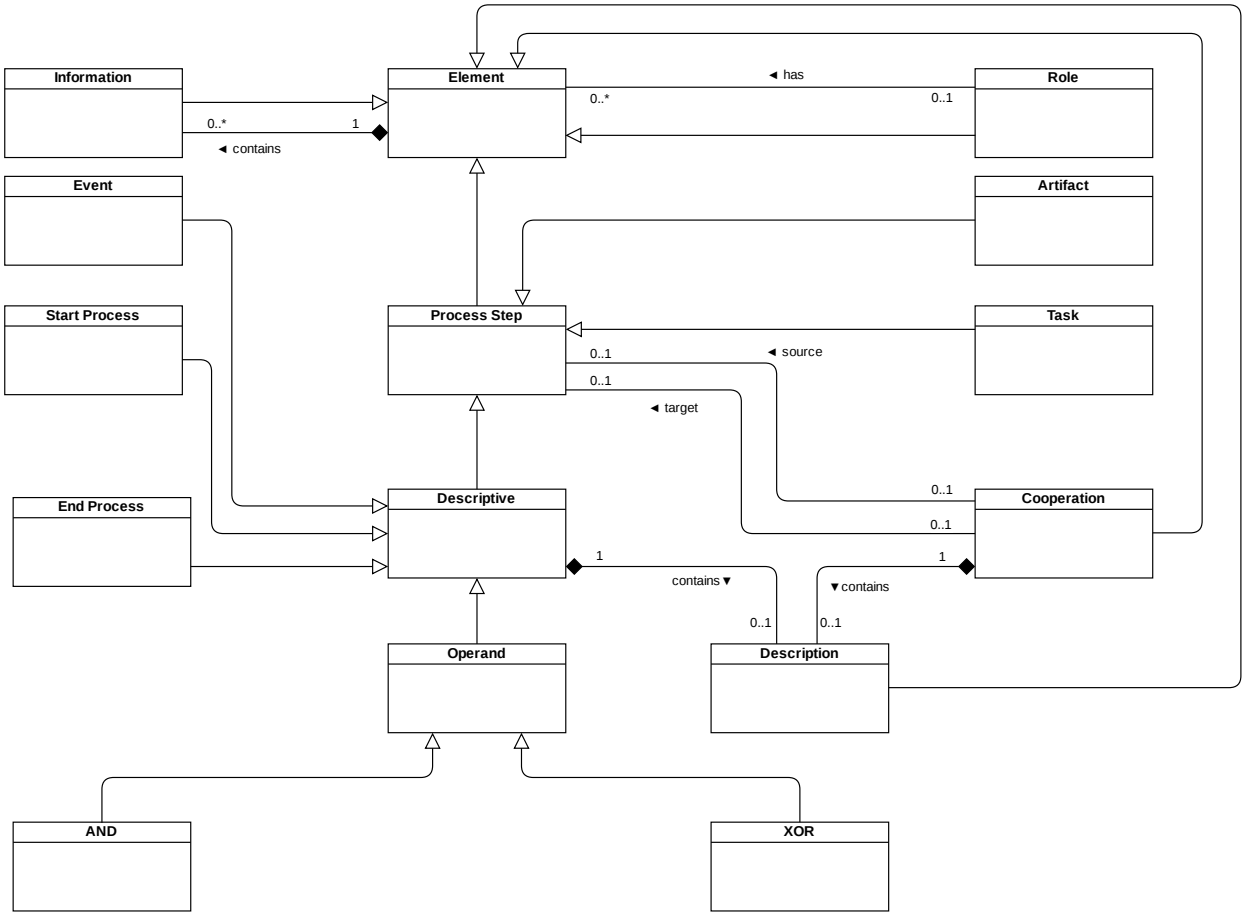
Obr. 4.2: Unified Process Meta-Model[12]

- **Element** - Abstraktný predok, pomocou ktorého sú charakterizované vlastnosti a metódy všetkých špecifických typov procesných elementov. Je vo vzťahu s *Informáciou* a *Rolou*, pretože každý procesný element môže byť bližšie popísaný niekoľkými informáciami, a zároveň každý procesný element môže vystupovať v kontexte nejakej roly, ktorou je proces obstarávaný.

- **Informácia** - Je špecifický druh *Elementu*, ktorý nesie istú popisnú informáciu o procesnom elemente. Je vo vzťahu s *Elementom*, pretože ak existuje, tak prináleží práce jednému procesnému elementu.

- **Rola** - Ide o *Element*, ktorý zhromažďuje procesné elementy do logickej štruktúry, čím naznačuje, že dané procesné elementy patria do daného kontextu, alebo sú vykonávané daným človekom, oddelením, strojom atď. Je vo vzťahu s *Elementom*, pretože môže obsahovať viacero elementov.

- **Procesný krok** - Ide o špecifický typ *Elementu*, ktorý je abstraktným predkom pre elementy s charakteristickou schopnosťou kooperácie. Preto je vo vzťahu práve s *Kooperáciou*, v ktorej plní rolu zdrojového alebo cieľového procesného kroku.



Obr. 4.3: Vytvorený všeobecný metamodel

- **Artefakt** - Špecifický druh *Procesného kroku*, ktorý sa viaže k vzniku konkrétnej inštancie objektu v súvislosti s výkonom určitej procesnej úlohy.
- **Úloha** - Jeden zo základných prvkov, ktorý je *Procesným krokom* a vyjadruje jednoduchú aktivitu v rámci procesu, pričom plní hlavnú úlohu pre zachovanie granularity procesu.
- **Popisný element** - Špecifický *Procesný krok*, ktorý vo všeobecnom metamodeli plní úlohu abstraktného predka pre určité druhy procesných krokov, ktoré môžu obsahovať *Popis*.
- **Popis** - Je *Elementom*, ktorý je vo vzťahu s *Popisným elementom* a *Kooperáciou*, pretože tieto druhy elementov môžu popis obsahovať.
- **Kooperácia** - Druh *Elementu*, ktorý slúži pre zabezpečenie prechodu medzi dvoma *Procesnými krokmi*, pričom jeden procesný krok predstavuje zdroj a druhý cieľ.
- **Udalosť** - Potomkom *Popisného elementu*, ktorý vyjadruje udalosť vzniknutú v dôsledku prevedenia určitého procesného kroku.

- **Štartovací proces** - Je *Popisný element*, ktorý vyjadruje začiatok procesu.
- **Koncový proces** - Je *Popisný element*, ktorý vyjadruje koniec procesu.
- **Operand** - Špecifický druh *Popisného elementu*, ktorý je abstraktným predkom pre elementy rozdeľujúce tok procesu.
- **Logický operand „AND“** - Ide o druh *Operandu* rozdeľujúci tok aktivity na viacero rovnocenných vetví.
- **Logický operand „XOR“** - Druh *Operandu* rozdeľujúci tok aktivity na viacero navzájom vylučujúcich sa vetví, pričom je vybraná práve jedna.

4.3 Mapovanie

Mapovanie pri transformácii je z pohľadu danej notácie realizované v dvoch smeroch. Za predpokladu, že ide o transformáciu do danej notácie, je nutné mapovať elementy všeobecného metamodelu do metamodelu danej notácie. V prípade že chceme vykonať transformáciu do inej notácie, je nutné mapovať elementy metamodelu danej notácie na všeobecný metamodel.

Nasledujúce tabuľky zobrazujú obojstranné mapovanie metamodelu diagramu aktivít UML a všeobecného metamodelu:

Tabuľka 4.1: Mapovanie metamodelu diagramu aktivít UML na všeobecný metamodel

Diagram aktivít UML	**Všeobecný metamodel**
Komentár	Informácia
Dráha	Rola
Počiatočný uzol	Štartovací proces
Koncový uzol	Koncový proces
Vetviaci/spojovací uzol	Logický operand „AND“
Rozhodovací/zlučovací uzol	Logický operand „XOR“
Riadiaca šipka	Kooperácia
Objektová šipka	Kooperácia
Sprievodca	Popis
Objektový uzol	Artefakt
Akcia	Úloha
Spúšťacia akcia	Udalosť
Príjmová akcia	Udalosť
Časová udalosť	Udalosť

Pri mapovaní bolo narazené na niekoľko problémov. Pri smere mapovania na všeobecný metamodel bolo postupované na základe sémantickej významnosti elementov. Niektoré elementy sú mapované na rovnaký element vo všeobecnom metamodeli, čo je spôsobené zvýšením levelu abstrakcie.

Tabuľka 4.2: Mapovanie všeobecného metamodelu na metamodel diagramu aktivít UML

Všeobecný metamodel	**Diagram aktivít UML**
Informácia	Komentár
Rola	Dráha
Kooperácia	Riadiaca šipka, Objektová šipka
Štartovací proces	Počiatočný uzol
Koncový proces	Koncový uzol
Logický operand „AND“	Vetviaci/spojovací uzol
Logický operand „XOR“	Rozhodovací/zlučovací uzol
Popis	Sprievodca
Artefakt	Objektový uzol
Úloha	Akcia
Udalosť	Akcia

V opačnom smere mapovania je možné pozorovať, že *Kooperácia* je mapovaná na *Riadiacu šipku* alebo *Objektovú šipku*. Výber konkrétneho druhu šipky závisí na pravidlách modelovacej notácie diagramu aktivít UML. Ak je zdrojový alebo cieľový element *Kooperácie Artefaktom*, potom *Kooperáciu* mapujeme na *Objektovú šipku*, pretože ide o spojenie s *Objektovým uzlom*. V opačnom prípade použijeme mapovanie na *Riadiacu šipku*.

Kapitola 5

Formálne obmedzenia

Okrem problému transformácie bolo v rámci diplomovej práce predmetom riešenia aj využitie formálnych obmedzení pri semiformálnom modelovaní. V tejto kapitole bude bližšie predstavená táto problematika. V prvej časti bude čitateľ oboznámený s pojmom ontológia. V druhej časti sa pozrieme na jazyk pre modelovanie ontológii OWL a v poslednej časti bude čitateľ zoznámený s riešením tejto problematiky v rámci diplomovej práce.

5.1 Ontológia

Slovo ontológia je zložené z gréckych slov *ontos* (byť, bytosť) a *logá* (štúdium, veda, teória), čo naznačuje samotnú definíciu tohto pojmu. Ide o oblasť filozofie, zaoberajúcou sa otázkou existencie a súcna, pričom zahŕňa aj abstraktnejšie otázky existencie či neexistencie niektorých entít a sú definované základné vzťahy „bytia“.[30]

Americký počítačový vedec a odborník v oblasti umelej inteligencie John F. Sowa vo svojej publikácii z roku 2000 hovorí o ontológii ako o doplnku k logike pre popis entít. Kým v logike je definovaný existenčný kvantifikátor ∃, ktorý hovorí že nejaká entita existuje, samotný popis entít logika neposkytuje. Pre tieto účely podľa Sowu existuje ontológia ako veda o existencii všetkých druhov entít - konkrétnych i abstraktných - ktoré tvoria svet.[3]

Tom Gruber, spoluzakladateľ tímu, ktorý vytvoril prvého inteligentného asistenta Siri využitím umelej inteligencie, hovorí o ontológii vo svojej publikácii z roku 2008 ako o súbore reprezentačných primitív, ktoré slúžia pre modelovanie domény vedomostí alebo diskurzu. Medzi reprezentačné primitíva patria triedy, atribúty a vzťahy. [31]

Ontológie sú typicky špecifikované v jazykoch, ktoré abstrahujú od dátových štruktúr či implementačných stratégii. Oproti napríklad databázovej schéme alebo doménovému modelu, ktoré sú bližšie k logickej či fyzickej úrovni, je ich využitie hodnotnejšie ku špecifikovaniu sémantickej úrovne. V nasledujúcej časti bude predstavený jazyk pre modelovanie ontológie OWL, ktorý bol využitý pri definovaní formálnych obmedzení vo vyvíjanom nástroji.

5.2 OWL

Sémantický web predstavuje budúcnosť v podobe tzv. Webu 3.0. Kým Web 1.0 bol charakteristický prepojením dát pomocou hyperlinkov, Web 2.0 zas zavedením aplikačnej integrácie do webu, Web 3.0 prichádza s myšlienkou prepojenia dát. Toto prepojenie je možné definovať práve pomocou OWL. Tento sémantický webový jazyk, založený na výpočtovej logike, je navrhnutý tak, aby bolo pomocou neho možné formálne reprezentovať komplexné znalosti o veciach (v jazyku OWL nazývaných ako *Thing*) a vzťahoch medzi nimi.[32][33]

V nasledujúcej časti budú popísané základné elementy jazyka OWL[34][35]:

- **Trieda (*Class*)** - Definuje skupinu jednotlivcov, ktoré majú rovnaké vlastnosti. Okrem toho je tu možná tiež hierarchizácia prostredníctvom subtried (*subClassOf*). Používateľsky definované triedy sú subtriedami špeciálnej všeobecnej triedy *Thing*. Táto trieda je tiež triedou všetkých jedincov. Okrem toho existuje v OWL aj špeciálna všeobecná trieda *Nothing*, ktorá nieje triedou žiadneho jedinca a zároveň k nej neexistujú žiadne subtriedy.

```
<owl:Class rdf:about="#Zviera"/>
<owl:Class rdf:about="#Mačka">
    <rdfs:subClassOf rdf:resource="#Zviera" />
</owl:Class>
<owl:Class rdf:about="#Pes">
    <rdfs:subClassOf rdf:resource="#Zviera" />
</owl:Class>
```

Výpis 5.1: Príklad zápisu triedy a subtried v jazyku OWL

- **Vlastnosť (*Property*)** - Slúži na vyjadrenie vzťahov medzi jednotlivcami alebo medzi jednotlivcom a dátovou hodnotou. Pre tieto účely rozlišujeme objektové vlastnosti (*ObjectProperty*) a dátové vlastnosti (*DatatypeProperty*). Je tu tiež možná hierarchizácia prostredníctvom subvlastností (*subPropertyOf*). Vlastnosti môže byť definovaná doména (*domain*) a rozsah (*range*). Doména vlastnosti obmedzuje jednotlivcov, na ktorých môže byť vlastnosť použitá a rozsah vlastnosti obmedzuje jedincov, ktorí môžu byť danej vlastnosti hodnotou.

```
<owl:ObjectProperty rdf:about="#naháňa">
    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
        topObjectProperty"/>
    <rdfs:domain rdf:resource="#Zviera"/>
    <rdfs:range rdf:resource="#Zviera"/>
</owl:ObjectProperty>
```

Výpis 5.2: Príklad zápisu vlastnosti a subvlastnosti v jazyku OWL

- **Jedinec (*Individual*)** - Inštancia triedy, pri ktorej môžu byť prostredníctvom vlastností definované vzťahy medzi jednotlivými jedincami. Ak nie je definovaný typ jedinca tak je daný jedinec inštanciou triedy *Thing*.

```
<owl:NamedIndividual rdf:about="#Elza">
    <rdf:type rdf:resource="#Mačka"/>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="#Buffy">
    <rdf:type rdf:resource="#Pes"/>
    <p:naháňa rdf:resource="#Elza"/>
</owl:NamedIndividual>
```

Výpis 5.3: Príklad zápisu jedinca v jazyku OWL

5.3 Riešenie vo vyvíjanom nástroji

Cieľom použitia formálnych obmedzení v rámci vyvíjaného nástroja na modelovanie v notácii diagramu aktivít UML bolo sformalizovanie sémantiky tohto semiformálneho jazyka. Vďaka formalizácii sémantiky budú jednoznačne definované modelované procesy, čo povedie k zamedzeniu chybovosti modelovania a teda aj eliminácii nesprávnej interpretácie procesov.

Pre tieto účely bol vybraný jazyk pre modelovanie ontológii OWL, pomocou ktorého je možné definovať formálne obmedzenia pre modelovanie konkrétnych procesov. V rámci modelovania procesov je dôležité striktné zachovanie poradia definovaných procesných akcii. V jednom OWL súbore je modelovaná definícia jedného procesu. Proces je charakterizovaný jedincami typu *Akcia (Action)*, čím je naznačené, že dané formálne obmedzenia je možné nastaviť pre elementy jazyka diagramu aktivít UML typu *Akcia (Action)*. Každý jedinec predstavuje jeden procesný krok. Pre definíciu následnosti jednotlivých akcii sú v rámci daného OWL definované objektové vlastnosti *predchodca (previous)* a *nasledovník (next)*. Dané objektové vlastnosti majú nastavenú *doménu (domain)* a *rozsah (range)* na jedincov triedy *Akcia (Action)*. Tieto vlastnosti sú následne priradené pre jedincov typu *Akcia (Action)*, pričom platí, že jedinec môže mať viac predchodcov a nasledovníkov.

```
<owl:Class rdf:about="#Action"/>
<!-- #next -->
<owl:ObjectProperty rdf:about="#next">
    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
        topObjectProperty"/>
    <rdfs:domain rdf:resource="#Action"/>
    <rdfs:range rdf:resource="#Action"/>
</owl:ObjectProperty>
```

```
<!-- #previous -->
<owl:ObjectProperty rdf:about="#previous">
    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
        topObjectProperty"/>
    <rdfs:domain rdf:resource="#Action"/>
    <rdfs:range rdf:resource="#Action"/>
</owl:ObjectProperty>
```

Výpis 5.4: Definícia následnosti v jazyku OWL

Prostredníctvom vyvinutého nástroja pre modelovanie je používateľ schopný importovať formálne obmedzenie v jazyku OWL. Nástroj spracuje importované OWL a uloží ako formálne obmedzenie, ktoré je možné priradiť jednotlivým akciám. Po ukončení modelovania a uložení modelu s využitým formálnym obmedzením prebieha prostredníctvom algoritmu prehľadávania grafu do hĺbky validácia modelovania daného procesu.

Kapitola 6

Vyvíjaný systém

Výstupným prvkom riešenia tejto diplomovej práce bolo vytvoriť nástroj, ktorý umožňuje semiformálne modelovanie v notácii diagramu aktivít UML s využitím formálnych obmedzení a umožňuje transformáciu medzi procesnými modelovacími notáciami. Samotný nástroj je súčasťou systému, ktorého vývoj začal v rámci diplomovej práce od úplných základov - v informatickom žargóne „na zelenej lúke“ - a prebiehal v spolupráci s iným študentom, pričom bola práca riadne rozdelená.

V nasledujúcich častiach textu bude čitateľovi predstavený vyvíjaný systém, pričom bude zamerané na časti ktoré sú výsledkom tejto diplomovej práce. V prvej časti bude predstavená špecifikácia požiadavkov na vyvíjaný systém a prípady použitia, následne bude predstavená architektúra a návrh systému a v poslednej časti budú demonštrované dôležité implementačné detaily.

6.1 Špecifikácia požiadavkov

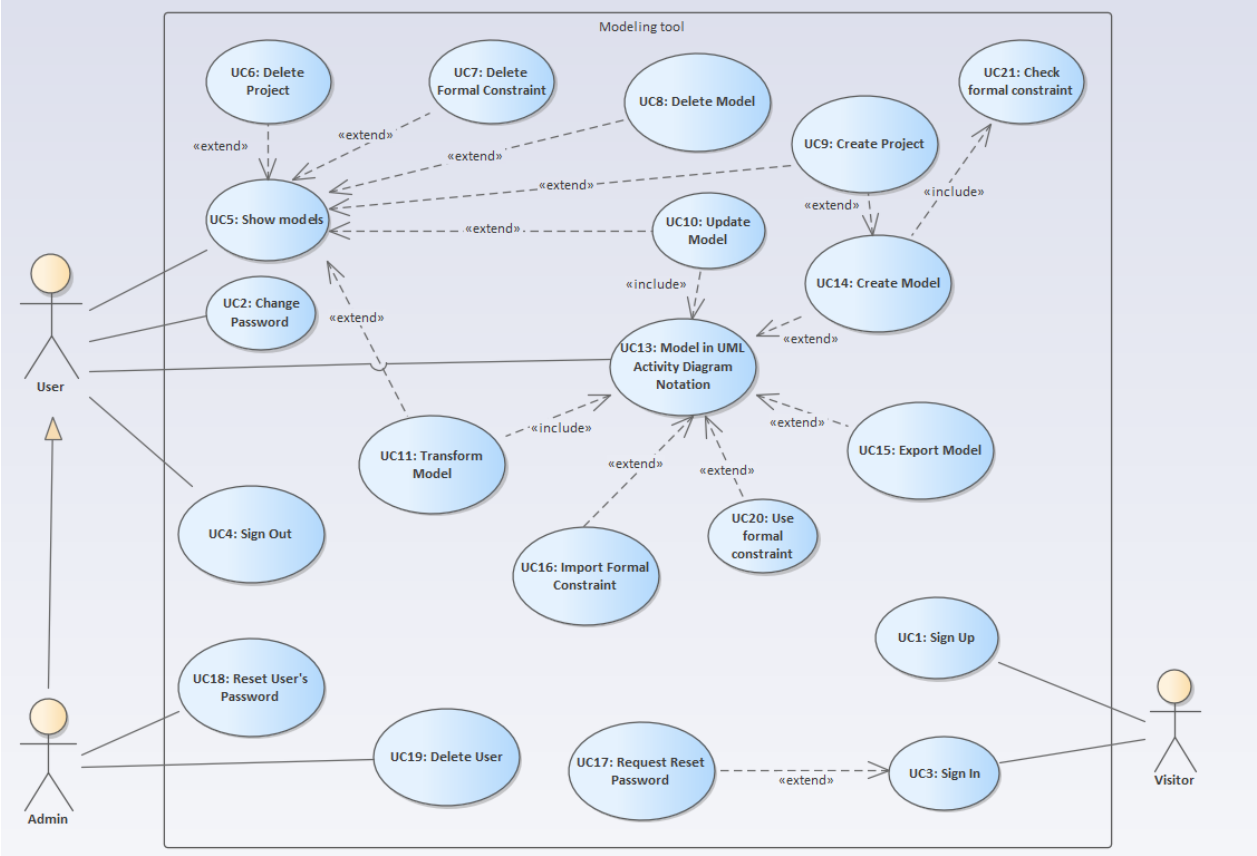
Vrámci analýzy pre vývoj spomínaného nástroja bolo nutné identifikovať konkrétne požiadavky na vyvíjaný systém. Pre špecifikáciu, evidenciu, správu a aktualizáciu požiadavkov bol využívaný nástroj pre riadenie projektu - *JIRA* - ktorý patrí medzi populárne nástroje v spojení s konfiguračným manažmentom a špeciálne v oblasti riadenia zmien. Požiadavky boli rozdelené do niekoľkých kolekcií (v nástroji JIRA nazývané z ang. *Epic*):

- **Analýza** - Kolekcia určená pre zhromaždenie nefunkčných požiadavkov súvisiacich s analýzou riešenej problematiky vrámci diplomovej práce. Patria tu požiadavky pre návrh dizajnu aplikácie, požiadavky pre vytvorenie metamodelu diagramu aktivít, mapovanie metamodelu na všeobecný metamodel a vytvorenie prípadov použitia.
- **Architektúra** - Súbor nefunkčných požiadavkov spojených s architektúrou a návrhom samotnej aplikácie.
- **Implementácia** - Rodina funkčných i nefunkčných požiadavkov spojených s implementáciou vyvíjaného nástroja. Z pohľadu nefunkčných požiadavkov tu patrí napríklad požiadavok

pre responzívny dizajn. Funkčné požiadavky budú bližšie popísané prostredníctvom diagramu prípadov použitia v nasledujúcich častiach tejto diplomovej práce.

- **Nasadenie** - Požiadavky spojené s riešenými technikami nasadenia.
- **Písomná časť** - Kolekcia požiadavkov, ktoré súvisia s textovou časťou tejto diplomovej práce.

Popis jednotlivých požiadavkov je možný z viacerých uhlov pohľadu. Súčasťou tejto diplomovej práce bol popis požiadavkov z pohľadu prípadov použitia:



Obr. 6.1: Diagram prípadov použitia modelovacieho systému

Z diagramu prípadov použitia je možné pozorovať interakcie medzi systémom a aktérmi *Používateľ* a *Administrátor*. *Administrátor* bude môcť vystupovať v roli *Používateľa* pričom bude môcť navyše spravovať používateľské účty. V nasledujúcej časti budú predstavené scenáre hlavných prípadov použitia:

UC1: Registrácia

Prípad použitia *Registrácia* umožní zaregistrovanie nového používateľa do systému, čím bude zabezpečená plnohodnotná manipulácia so systémom pre nového používateľa.

Aktéri: Návštevník
Spúšťač: Aktér klikne na tlačidlo pre registráciu
Podmienky: Aktér nieje prihlásený
Hlavný scenár:

1. Systém zobrazí formulár pre registráciu používateľa.
2. Aktér vyplní formulár.
3. Systém prevedie validáciu formulára.
4. Systém informuje používateľa o úspešnej registrácii.

Alternatívny scenár:

3. 3.1. Systém informuje o chybe pri validácii.
   3.2. Aktér pokračuje krokom 2.

UC2: Zmena hesla

Umožní používateľovi a administrátorovi zmenu hesla.

Aktéri: Používateľ, Administrátor
Spúšťač: Aktér klikne na používateľské tlačidlo (ikona používateľa) a vyberie možnosť zmena hesla
Podmienky: Prihlásený používateľ
Hlavný scenár:

1. Systém zobrazí formulár pre zmenu hesla.
2. Aktér zadá staré heslo a opakovane nové heslo.
3. Systém prevedie validáciu formulára.
4. Systém informuje o úspešnej zmene hesla.

Alternatívny scenár:

3. 3.1. Systém informuje o zadaní nesprávneho starého alebo nového hesla.
   3.2. Aktér pokračuje krokom 2.

UC3: Prihlásenie

Umožní prihlásenie používateľa do systému pre jeho plnohodnotné využitie.

Aktéri: Návštevník
Spúšťač: Aktér klikne na tlačidlo pre prihlásenie
Podmienky: Aktér nieje prihlásený a je registrovaný
Hlavný scenár:

1. Systém zobrazí formulár pre prihlásenie používateľa.
2. Aktér zadá používateľské meno a heslo.
3. Systém prevedie kontrolu používateľského mena a hesla.
4. Systém informuje o úspešnom prihlásení a zobrazí úvodnú stránku nástroja.

Alternatívny scenár:

2. 2.1. Aktér vyberie možnosť *Zabudnuté heslo*.
   2.2. UC17: Vyžiadanie obnovy hesla.
3. 3.1. Systém informuje o zadaní nesprávneho mena alebo hesla.
   3.2. Aktér pokračuje krokom 2.

UC5: Ukážka modelov

Umožní používateľovi zobrazenie a sekundárne aj správu modelov.

Aktéri: Používateľ, Administrátor
Spúšťač: Aktér klikne na tlačidlo *Moje modely*
Podmienky: Aktér je prihlásený
Hlavný scenár:

1. Systém zobrazí obrazovku *Moje modely*.
2. Aktér vyberie model pre zobrazenie.
3. Systém zobrazí na plátne vybraný model.

Alternatívny scenár:

2. 2.1. (a) Aktér vyberie možnosť *Vymazanie projektu*.
   2.2. (a) UC6: Vymazanie projektu.

2.1. (b) Aktér vyberie možnosť *Vymazanie formálneho obmedzenia*.

2.2. (b) UC7: Vymazanie formálneho obmedzenia.

2.1. (c) Aktér vyberie možnosť *Vymazanie modelu*.

2.2. (c) UC8: Vymazanie modelu.

2.1. (d) Aktér zadá meno nového projektu a vyberie možnosť *Vytvorenie projektu*.

2.2. (d) UC9: Vytvorenie projektu.

2.1. (e) Aktér vyberie možnosť *Zmena modelu*.

2.2. (e) UC10: Zmena modelu.

2.1. (f) Aktér vyberie možnosť *Transformuj model*.

2.2. (f) UC11: Transformuj model.

UC13: Modelovanie v notácii diagramu aktivít UML

Poskytne používateľovi možnosť modelovania pomocou notácie diagramu aktivít jazyka UML

Aktéri: Používateľ, Administrátor
Spúšťač: Aktér klikne na záložku *Diagram aktivít*
Podmienky: Aktér je prihlásený
Hlavný scenár:

1. Systém zobrazí obrazovku *Diagram aktivít* zloženú z modelovacieho plátna a ľavého menu, ktoré poskytuje elementy modelovacej notácie, a tiež záložku pre obsluhu formálnych obmedzení.
2. Aktér si vyberie element a presunie ho na plátno.
3. Systém zobrazí na plátne presunutý element a zobrazí pravé menu pre konfiguráciu elementu.
4. Aktér konfiguruje element pomocou pravého menu podľa potreby.
5. Systém zobrazuje konfiguráciu aktéra.
6. Aktér spája konfigurované elementy cez body závislostí pomocou šipiek.
7. Systém vyhodnotí použitie šipky a použije korektný typ šipky.

Alternatívny scenár:

2. 2.1. (a) Aktér vyberie možnosť *Vytvorenie modelu*.

   2.2. (a) UC14: Vytvorenie modelu.

2.1. (b) Aktér vyberie možnosť *Export modelu*.

2.2. (b) UC15: Export modelu.

2.1. (c) Aktér vyberie možnosť *Import formálneho obmedzenia*.

2.2. (c) UC16: Import formálneho obmedzenia.

2.1. (d) Aktér vyberie pre element formálne obmedzenie.

2.2. (d) UC20: Použitie formálneho obmedzenia.

UC18: Obnova hesla používateľa

Prípad použitia *Obnova hesla používateľa* umožní obnovu zabudnutého používateľského hesla poslaním emailu s nový heslom.

Aktéri: Administrátor
Spúšťač: UC17: Žiadosť používateľa o obnovu hesla
Podmienky: Aktér je administrátor, je prihlásený a používateľ požiadal o obnovu hesla
Hlavný scenár:

1. Systém zobrazí kolekciu žiadostí o obnovu hesla.
2. Aktér vyberie používateľa, ktorému chce obnoviť heslo a vyberie možnosť *Potvrdiť*.
3. Systém zobrazí hlásenie, že prebehne obnova hesla na daný email.
4. Aktér skontroluje email a potvrdí obnovu hesla.
5. Systém zašle email s novým, vygenerovaným heslom používateľovi žiadajúcemu o nové heslo.

Alternatívny scenár:

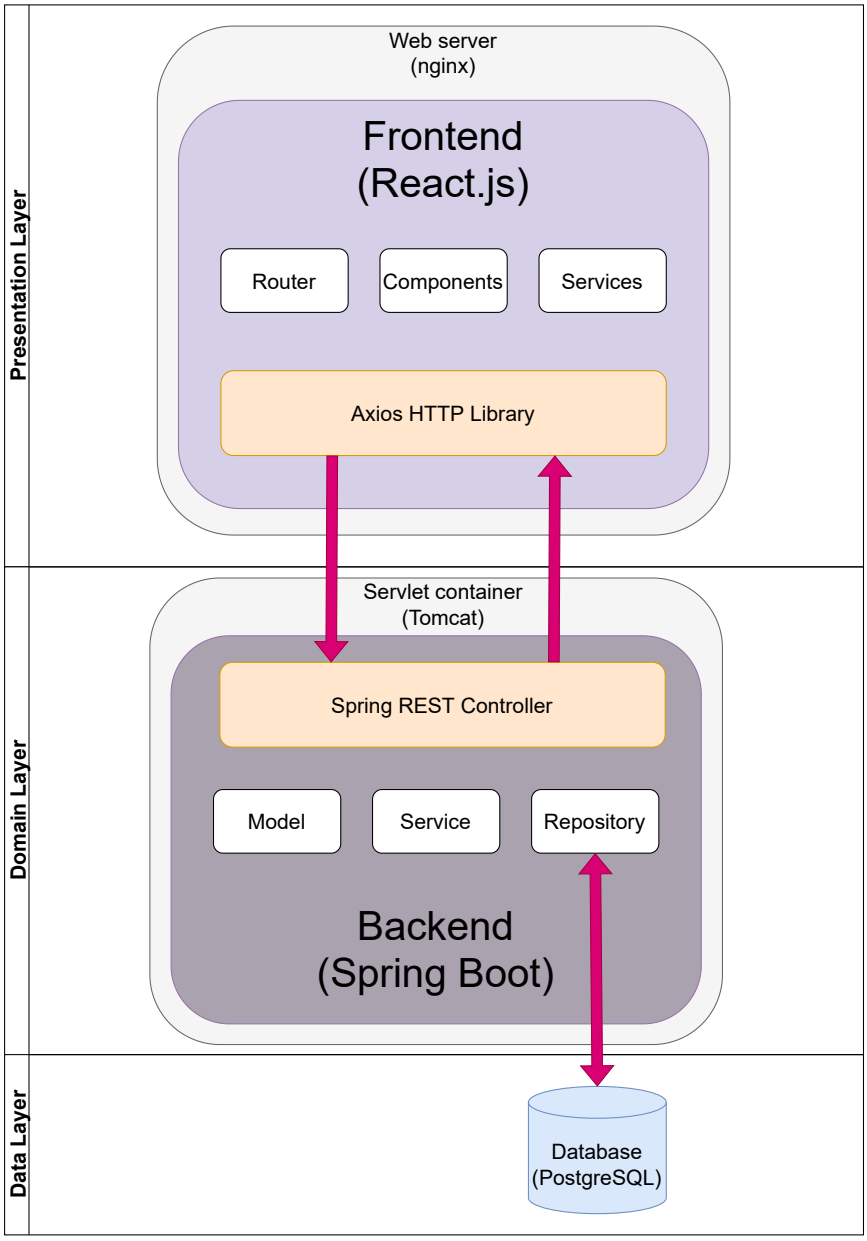
4. 4.1. Aktér nepotvrdí obnovu hesla.
   4.2. Systém pokračuje krokom 1.

6.2 Architektúra a návrh

Po špecifikovaní požiadavkov prostredníctvom prípadov použitia bolo nutné špecifikovať aj architektúru samotného systému. Vyvíjaný nástroj bol implementovaný využitím trojvrstvovej architektúry - prezentačná vrstva, aplikačná vrstva a dátová vrstva.

Prezentačná vrstva je vyvinutá prostredníctvom reaktívnej JavaScript knižnice React.js. Pre komunikáciu s aplikačnou vrstvou je využívaná knižnica pre konštrukciu požiadavkov Axios, ktorá komunikuje cez protokol HTTP (Hypertext Transfer Protocol) s aplikačnou vrstvou fungujúcou ako REST (Representational state transfer) API (Application programmig interface) implementovanou

prostredníctvom frameworku nad Javou EE - Spring Boot. API je zabezpečené pomocou technológie JWT (JSON Web Token). Aplikačná vrstva komunikuje s dátovou vrstvou prostredníctvom DTO (Data Transfer Object - v Spring Boot tzv. Repozitár). Repozitáre pre objektovo-relačné mapovanie využívajú technológiu Hibernate. Vrámci dátovej vrstvy bol využitý objektovo-relačný databázový systém PostgreSQL. Nasledujúci diagram zobrazuje popísanú architektúru systému:



Obr. 6.2: Diagram architektúry systému

V nasledujúcej časti budú bližšie popísané jednotlivé vrstvy a v nich využité technológie.

6.2.1 Prezentačná vrstva

Pre bližšiu špecifikáciu prezentačnej vrstvy budú čitateľovi v nasledujúcich častiach textu predstavené využité technológie pri vývoji modelovacieho nástroja. Okrem technológii bude poukázané na vytvorené obrazovky vrámci tejto diplomovej práce a na záver budú predstavené aj balíčky klientskej aplikácie.

6.2.1.1 Využité technológie vrámci prezentačnej vrstvy

React
Technológia zvolená pre vývoj klientskej časti systému. Táto JavaScript knižnica umožňuje efektívne vytváranie hierarchických webových stránok prostredníctvom komponentov, ktoré uchovávajú svoj stav. Pri zmene stavu komponentu nieje nutné obnovenie celej stránky, ale len zmenenej časti, čo je zabezpečené prostredníctvom virtuálneho DOMu (Document Object Model). Okrem toho je možné v knižnici tiež využívať aj metódy životného cyklu komponentov. [36]

MxGraph
JavaScript knižnica určená pre vývoj modelovacích nástrojov alebo nástrojov zobrazujúcich výstupy v podobe grafov. Poskytuje široké spektrum funkcií, ktoré je možné využiť pri modelovaní vrátane modelovacieho plátna, automatické usporiadanie elementov, funkcie krok späť/krok vpred, podporu pre konfiguráciu elementov a mnoho ďalšieho. Knižnica je tiež využívaná u popredných modelovacích nástrojov ako je *Draw.io*.[37]

Bootstrap
Knižnica kaskádových štýlov poskytujúca množstvo šablón a vizualizácii prvkov webu ako sú tlačidlá, formuláre, navigačné panely a iné. Výhodou využitia technológie bootstrap je novodobý a responzívny dizajn, vďaka ktorému je zabezpečená kompatibilita klientskej aplikácie pre rôzne zariadenia. Okrem kaskádových štýlov poskytuje tiež vytvorené komponenty a JavaScript pluginy.[38]

Axios
Technológia pre vytváranie jednoducho konfigurovateľných HTTP požiadavkov, ktoré v klientskeh aplikácii slúžia pre komunikáciu s REST API. Základom technológie Axios je objekt typu *Promise*, ktorý zabezpečí informovanie o odpovedi zo serveru, či prípadnej chybe.[39]

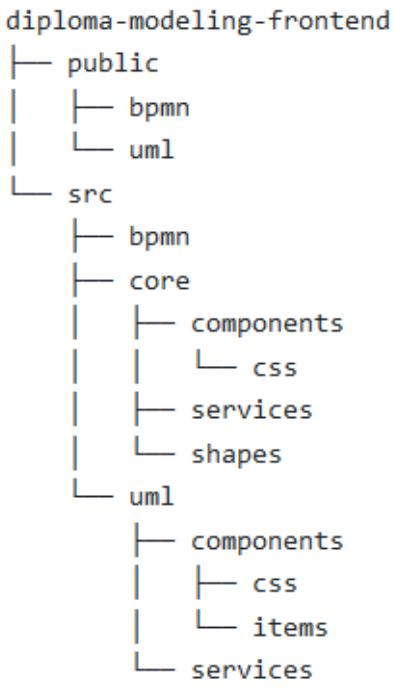
6.2.1.2 Vytvorené obrazovky prezentačnej vrstvy

Na základe návrhov používateľského rozhrania iným študentom pracujúcim na vyvíjanom systéme, boli vrámci tejto diplomovej práce vytvorené obrazovky: *Modelovanie prostredníctvom diagramu aktivít UML*, *Úvodná stránka s popisom a používateľským manuálom*, *Prihlásenie*, *Registrácia*, *Modálne*

okno pre ukladanie modelu, *Záložka pre prácu s formálnymi obmedzeniami*. Jednotlivé obrazovky budú podrobnejšie popísané v kapitole 7.

6.2.1.3 Štruktúra balíčkov klientskej aplikácie

V nasledujúcej časti bude čitateľ oboznámený so štruktúrou klientskej aplikácie. Klientska aplikácia sa skladá z viacerých balíčkov, avšak, keďže bola vyvíjaná v spolupráci s iným študentom, v tejto časti budú detailne prezentované balíčky riešené vrámci tejto diplomovej práce.



Obr. 6.3: Štruktúra klientskej aplikácie

Klientska aplikácia je na prvej úrovni zložená z balíčkov *public* a *src*. Balíček *public* obsahuje hlavnú stránku, konfiguračný súbor *manifest.json*, a tiež obrázky použité vo webovej aplikácii. V zložke *uml* sa nachádzajú obrázky jednotlivých elementov modelovacej notácie diagramu aktivít UML. Vrámci zložky *src* sú komponenty zlučujúce všetky ostatné komponenty aplikácie. Je tu však podstatná zložka *core* obsahujúca hlavné komponenty, služby a definované konštrukcie tvarov elementov pre knižnicu *MxGraph*. Balíček *uml* tiež obsahuje komponenty a služby využívané vrámci modelovacieho nástroja v notácii diagramu aktivít UML. Komponenty tiež obsahujú zložku *items*, kde sú uložené stavové komponenty jednotlivých modelovacích elementov.

6.2.2 Aplikačná vrstva

Aplikačnú vrstvu systému predstavuje serverová aplikácia, fungujúca ako REST API, pričom komunikuje s klientskou aplikáciou prostredníctvom požiadavkov. Klient pošle HTTP požiadavok na server a ten následne po vyhodnotení a prevedení požadovaných akcii vráti odpoveď klientovi. Okrem toho serverová aplikácia zabezpečuje aj komunikáciu s dátovou vrstvou. V nasledujúcej časti bude čitateľ oboznámený s využitými technológiami vrámci aplikačnej vrstvy, štruktúrou balíčkov a doménovým modelom hlavnej časti systému.

6.2.2.1 Využité technológie vrámci aplikačnej vrstvy

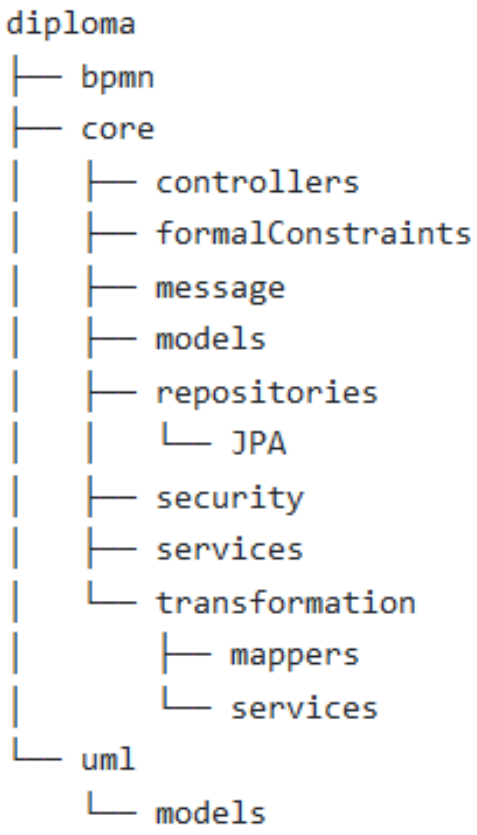
Spring Boot
Framework využívaný pri implementácii serverovej aplikácie, ktorý je určený pre rýchly a efektívny vývoj REST API. Výhodou frameworku je aj oddelenie nutnosti konfigurácie, pretože poskytuje automatické nastavenie potrebných závislostí, čo vedie k rýchlejšiemu a jednoduchšiemu vývoju aplikácii. Okrem toho ponúka možnosť využitia vstavaných alebo externých serverov ako *Tomcat* alebo *Jetty*. Endpointy sú vyvíjané prostredníctvom *ovládačov (z ang. controller)*, ktoré pre výpočtové operácie využívajú *služby (z ang. service)*. *Služby* pri rôznych logických úkonoch ukladajú potrebné výsledky do databáze využitím *repozitárov (z ang. repository)*, ktoré zachovávajú dátovú prezistenciu využitím technológie *JPA (Java Persistemce Api)*. [40]

JPA (+Hibernate)
Predstavuje štandard alebo špecifikáciu pre objektovo relačné mapovanie a je súčasťou Java EE. Niektoré servery implementujú JPA ako napríklad Glassfish. K serverom, ktoré JPA neimplementujú (ako napríklad Tomcat) je potrebné využitie technológie **Hibernate**, ktorá predstavuje framework mapujúci java objekty na tabuľky relačných databáz. Inými slovami Hibernate je implementáciou JPA. [41]

JWT
Štandard, ktorý zabezpečuje bezpečný transport informácii prostredníctvom JSON objektov. Vrámci serverovej aplikácie bol využitý pri autorizácii, kedy server po overení používateľa vráti späť klientovi identifikačný token, ktorým sa používateľ pri nasledujúcich požiadavkoch na server identifikuje v hlavičke požiadavku. [42]

6.2.2.2 Štruktúra balíčkov serverovej aplikácie

Obdobne ako u prezentačnej vrstvy bola definovaná štruktúra klientskej aplikácie, vrámci aplikačnej vrstvy bude predstavená štruktúra serverovej aplikácie, pričom budú podrobnejšie rozobrané balíčky vytvorené vrámci tejto diplomovej práce.



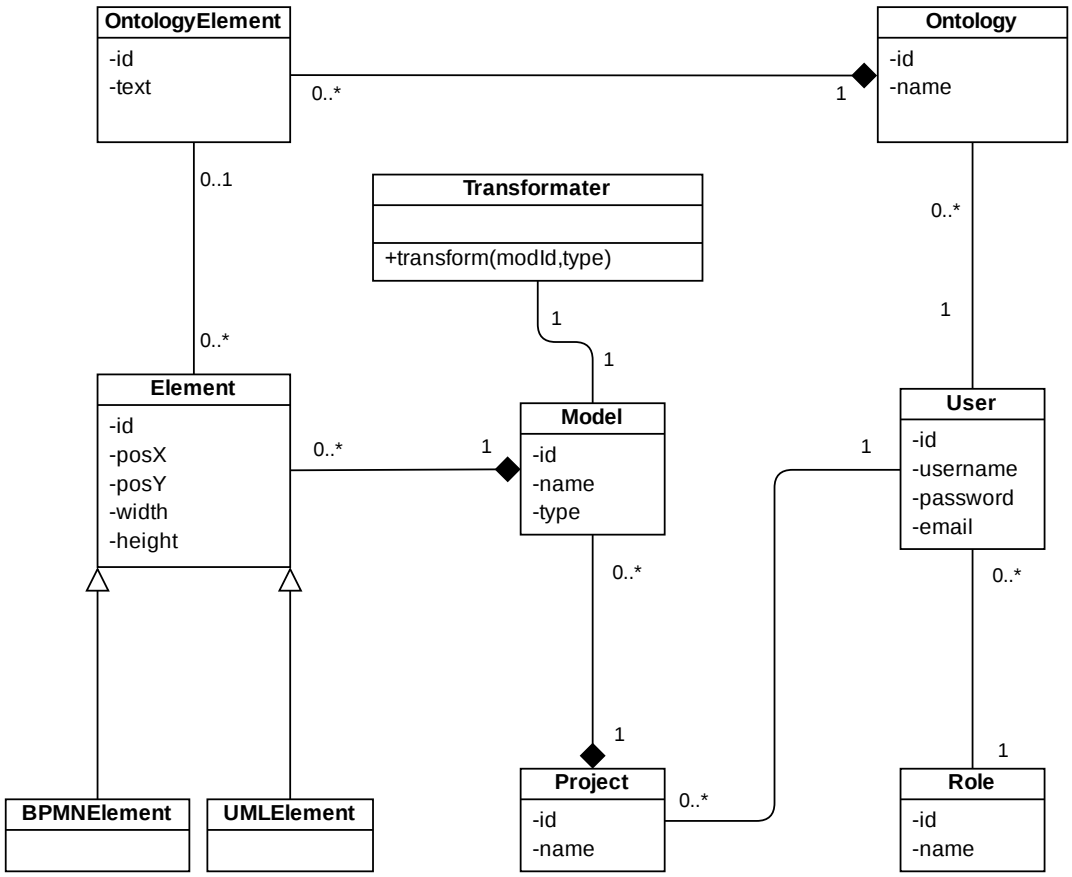
Obr. 6.4: Štruktúra serverovej aplikácie

Balíčky serverovej aplikácie boli rozdelené do viacerých kategórii podľa významnosti jednotlivých tried. Vrámci tejto diplomovej práce sú dôležité balíčky *core* a *uml*. Balíček *uml* obsahuje balíček *models*, ktorý pozostáva z tried reprezentujúcich entity metamodelu diagramu aktivít UML. Tieto entity sú následne využívané v hlavnom balíčku *core*, zloženého z tried, ktoré plnia konfiguračnú funkciu serverovej aplikácie. Okrem základných balíčkov obsahujúcich typické triedy pre framework Spring (*controllers*, *models*, *services*, *repositories*) je tu možné nájsť aj balíček *message* obsahujúci triedy, ktoré sa starajú o návratové správy zo serveru. Balíček security obsahuje triedy potrebné pre zabezpečenie aplikácie. Pre triedy potrebné k riešeniu problému transformácie bol vytvorený balíček *transformation*.

6.2.2.3 Doménový model hlavnej časti systému

Pre popis hlavnej časti systému (balíček *core*) bol vytvorený doménový model. *Používateľ* v systéme vystupuje pod jednou z rolí *ADMIN* alebo *USER*, pričom si môže do systému importovať *Ontológie* definujúce formálne obmedzenia a vytvárať *Projekty* obsahujúce *Modely*. *Model* obsahuje *Elementy* a využíva *Transformátor* pre transformovanie modelu do inej modelovacej notácie metódou *trans-*

form(). Pri transformácii je využívaný všeobecný metamodel (viď 4.3). Element môže využiť ako formálne obmedzenie *Element ontológie*, z ktorých samotná *Ontológia* pozostáva. Z *Elementu* následne dedia elementy rôznych modelovací notácii, ktoré sú reprezentované vlastným metamodelom. Metamodel diagramu aktivít (viď 4.1) bude podrobnejšie popísaný prostredníctvom databázového relačného modelu vrámci dátovej vrstvy.



Obr. 6.5: Doménový model hlavnej časti serverovej aplikácie

6.2.3 Dátová vrstva

Pre upresnenie návrhu a architektúry na úrovni dátovej vrstvy bude čitateľ oboznámený s využitým databázovým systémom a relačným modelom notácie diagramu aktivít UML.

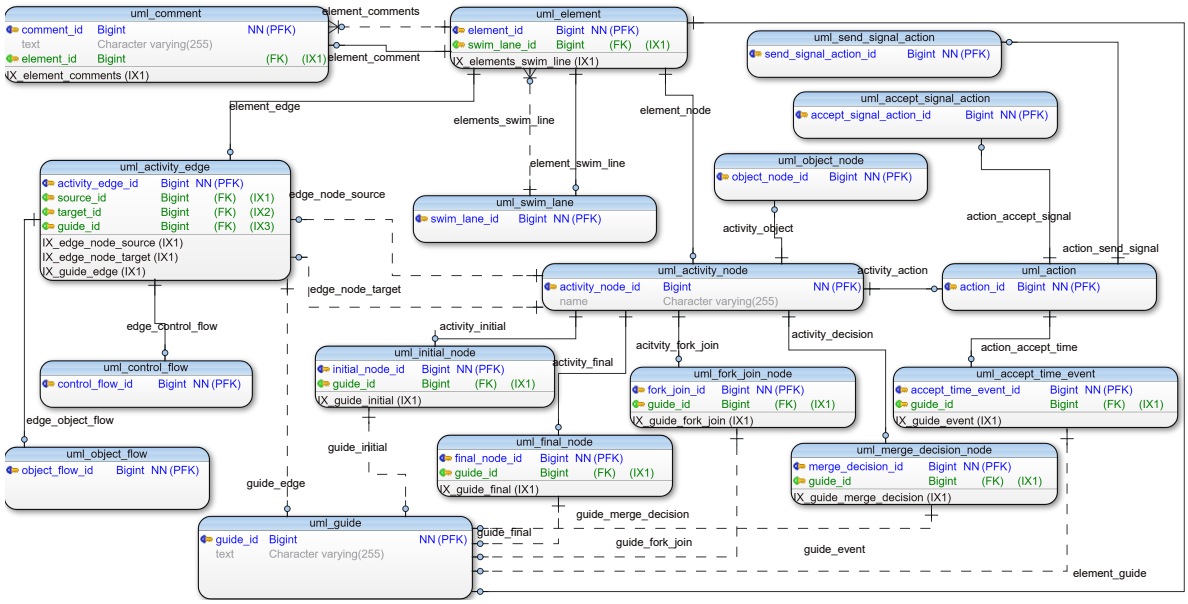
6.2.3.1 Využitý databázový systém

Vrámci dátovej vrstvy bol vybraný jeden zo súčasne najpopulárnejších databázových systémov - **PostgreSQL**. Ide o objektovo-relačný databázový systém rozširujúci štandardný databázový jazyk SQL (Structured Query Language), ktorý je obohatený o množstvo využiteľných funkcii.

Okrem primitívnych dátových typov tiež obsahuje aj štrukturovanejšie dátové typy, geometrické či dátové typy pre ukladanie rôznych typov dokumentov (JSON, XML). Medzi ďalšie výhody databázového systému patrí aj bezpečnosť, dátová integrita či výkon. [43]

6.2.3.2 Relačný dátový model

Pre potreby dlhodobého uchovania modelov modelovacej notácie diagramu aktivít UML, a tiež popis jednotlivých dátových reprezentácii ukladaných informácii, bol zostrojený relačný dátový model metamodelu diagramu aktivít:



Obr. 6.6: Relačný dátový model metamodelu diagramu aktivít

V konceptuálnom modeli je možné pozorovať dva typy väzieb. Takzvané *identifikujúce* väzby (plná čiara) interpretujú, že daná entita, ktorá reprezentuje element, je bližšie špecifikovaná pomocou inej entity. V OOP je tento jav možné interpretovať dedičnosťou. Druhým typom väzby sú *neidentifikujúce* väzby (prerušovaná čiara), ktoré znázorňujú relácie medzi jednotlivými entitami. Pre príklad uvediem vzťahy medzi entitami *uml_element (element)* a *uml_comment (komentár)*. Komentár je elementom a zároveň k jednému elementu môže existovať niekoľko komentárov.

Z relačného modelu bol následne vygenerovaný skript pre zostrojenie databázového modelu. Pre tieto účely bol využívaný nástroj Toad Data Modeler. Po aplikácii skriptu, ktorý zostrojil databázový model, bol využitý JPA Buddy (nástroj vývojového prostredia IntelliJ IDEA), pomocou ktorého

boli metódou reverzného inžinierstva vygenerované java triedy, ktoré predstavujú jednotlivé entity a väzby medzi nimi.

6.3 Implementácia

Ako už bolo spomenuté, vývoj modelovacieho systému začal od úplného začiatku - v informatickom žargóne „na zelenej lúke“ - a prebiehal v spolupráci s iným študentom, pričom jednotlivé implementačné časti systému boli riadne rozdelené. Správa verzii zdrojového kódu bolo zabezpečená využitím softvéru GIT.

Implementačné požiadavky riešené vrámci tejto diplomovej práce možno rozdeliť do niekoľkých kategórii - *všeobecné*, *modelovanie*, *transformácia* a napokon *formálne obmedzenia*. V nasledujúcej časti budú čitateľovi predstavené najdôležitejšie detaily implementácie jednotlivých kategórii.

6.3.1 Všeobecné

Všeobecné implementačné detaily predstavujú vybrané implementačné detaily pokrývajúce funkcionality a štandardy, ktoré by každý vyvíjaný systém mal obsahovať. Neoddeliteľnou súčasťou každého systému pre zachovanie istej miery bezpečnosti je vedomosť o používateľoch využívajúcich daný systém.

Používateľ by mal byť teda schopný sa do systému prihlásiť, prípadne zaregistrovať. Registráciu používateľa z klientskej aplikácie zabezpečuje *autentifikačná služba* ktorá prostredníctvom technológie *axios* vytvorí požiadavku na server typu *POST* a pošle v tele požiadavku zadané údaje z formulára.

```
register(username, email, password) {
    return axios.post(API_URL + "register", {
        username,
        email,
        password
    });
}
```

Výpis 6.1: Metóda autentifikačnej služby klientskej aplikácie pre registráciu

Server spracuje požiadavok prostredníctvom ovládača (z ang. controller) *AuthController* metódou určenou pre obstaranie požadovaného koncového bodu */register*. Ovládač využíva metódu používateľskej služby *registerUser*, ktorej ako parameter posiela objekt typu *User*, v ktorom sú mapované údaje z tela požiadavku.

```
@Transactional
public ResponseEntity<?> registerUser(User user) {
```

```
    if (userRepository.existsByUsername(user.getUsername())) {
        return ResponseEntity
                .badRequest()
                .body(new MessageResponse("Error: Username is already taken!"));
    }
    if (userRepository.existsByEmail(user.getEmail())) {
        return ResponseEntity
                .badRequest()
                .body(new MessageResponse("Error: Email is already in use!"));
    }
    // Create new user's account
    user.setPassword(passwordEncoder.encode(user.getPassword()));
    Role userRole = roleRepository.findByName(ERole.ROLE_USER);
    user.setRole(userRole);
    userRepository.insert(user);
    return ResponseEntity.ok(new MessageResponse("User registered successfully!"))
        ;
}
```

Výpis 6.2: Spracovanie požiadavky pre registráciu používateľa na serveri

Metóda najskôr skontroluje prípadnú exitujúcu registráciu daného používateľa. Ak je používateľ zaregistrovaný alebo je zadaný email využívaný, metóda vráti informáciu o existencii a registrácia sa neprevedie. V opačnom prípade dôjde k zašifrovaniu používateľského hesla, nastaveniu role *USER* a uloženiu nového používateľského konta. O tejto skutočnosti informuje metóda klientskú aplikáciu pomocou objektu typu *MessageResponse*.

Obdobným spôsobom prebieha prostredníctvom autentifikačnej služby v klientskej aplikácie aj prihlásenie. Rozdiel je v tom, že po odoslaní požiadavku na server pre prihlásenie prostredníctvom metódy *post* je následne na tento objekt volaná metóda *then*, ktorá čaká na odpoveď zo serveru pre spracovanie. Server prevedie požadované akcie prostredníctvom metódy používateľskej služby *authentificateUser*. V metóde dôjde k autentifikácii používateľa, vygenerovaniu bezpečnostného tokenu technológiou *JWT* a vráteniu spolu s rolami používateľa späť klientskej aplikácii.

```
public ResponseEntity<?> authentificateUser(User user) {
    try {
        Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(user.getUsername(), user.
                    getPassword()));
        SecurityContextHolder.getContext().setAuthentication(authentication);
        String jwt = jwtUtils.generateJwtToken(authentication);
```

```
        UserDetailsImpl userDetails = (UserDetailsImpl) authentication.
            getPrincipal();
        List<String> roles = userDetails.getAuthorities().stream()
                .map(item -> item.getAuthority())
                .collect(Collectors.toList());
        return ResponseEntity.ok(new JwtResponse(jwt,
                userDetails.getId(),
                userDetails.getUsername(),
                userDetails.getEmail(),
                roles));
    }catch (Exception e){
        return ResponseEntity.badRequest().body(e.getMessage());
    }
}
```

Výpis 6.3: Spracovanie požiadavky pre prihlásenie používateľa na serveri

Po odpovedi serveru je v prípade úspechu uložený vygenerovaný token do lokálneho úložiska prehliadača. Klientská aplikácia bude po tomto momente využívať vygenerovaný token pre identifikáciu serverom prostredníctvom technológie JWT pri každom nasledujúcom požiadavku na server. Životnosť vygenerovaného tokenu je časovo ohraničená prostredníctvom nastavenia *jwtExpirationMs* v konfiguračnom súbore *application.yml*.

6.3.2 Modelovanie

Medzi najdôležitejšie implementačné detaily modelovacej časti patrí spôsob uchovávania dát v klientskej aplikácii. Každý prvok metamodelu je reprezentovaný samostatnou triedou predstavujúcou v knižnici *React.js* jeden komponent klientskej aplikácie. Komponenty sú charakteristické možnosťou uchovania svojho stavu.

```
class UmlMergeDecisionNode extends UmlActivityNode {
  constructor(props) {
    super(props);
    // overrides
    this.state.width = 50;
    this.state.height = 50;
    this.state.type = "UmlMergeDecisionNode";
    // state
    this.state = {
      ...this.state,
```

```
      guide: null,
    }
  }
  // getters
  getGuide() {
    return this.state.guide;
  }
  // setters
  setGuide(guide) {
    this.setState({
      guide: guide
    });
  }
}
export default UmlMergeDecisionNode;
```

Výpis 6.4: Komponent klientskej aplikácie reprezentujúci element metamodelu

Pridávaním elementov na modelovacie plátno bolo nutné synchronizovane udržiavať tri kolekcie reprezentujúce elementy na modelovacom plátne. Kolekcia *elements* predstavuje kolekciu slúžiacu na vytvorenie komponentov príslušného typu mapovaním tejto kolekcie v zobrazovacej časti. Po vzniku komponentu je následne pre plnohodnotnú prácu s elementom uložená jeho referencia do kolekcie *children*. Poslednou kolekciou je kolekcia udržiavaná knižnicou *mxGraph* predstavujúcu jednotlivé elementy na modelovacom plátne.

Cieľom využitia stavov komponentov bolo zachovanie štruktúry serverovej reprezentácie elementu, a tiež podpora pri konštrukcii objektu typu *JSON*, ktorý bude následne poslaný na server a mapovaný na príslušné objekty. Vytvorenie *JSONu* posielaného na server bolo následne realizované aj za pomoci funkcie *mapElements*, ktorá z elementov štruktúry *children* prostredníctvom rekurzívneho algoritmu vytvorí *JSON* odpovedajúci serverovej reprezentácii elementov.

```
mapElements(elements) {
    var ret;
    ret = elements.map((element) => {
        var retStateOfElement = element.state;
        for (const itemKey in element.state) {
            if (Object.hasOwnProperty.call(element.state, itemKey)) {
                const itemValue = element.state[itemKey];
                if (itemValue !== null) {
                    if (itemValue.state) {
                        retStateOfElement[itemKey] = itemValue.state;
```

```
                } else if (Array.isArray(itemValue)) {
                    retStateOfElement[itemKey] = this.mapElements(itemValue);
                } else {
                    retStateOfElement[itemKey] = itemValue;
                }
            }
        }
        return retStateOfElement;
    })
    return ret;
}
```

Výpis 6.5: Funkcia pre mapovanie elementov do JSONu

6.3.3 Transformácia

Ako už bolo spomínané, transformácia prebiehala prostredníctvom mapovania elementov metamodelu diagramu aktivít UML do všeobecného metamodelu a opačne. Pre tieto účely bola využitá knižnica *Orika*, ktorá sa stará o mapovanie objektov na základe definície mapovacieho manažéra. Pre príklad uvediem mapovanie elementu *Akcia*.

```
this.mapperFactory.classMap(UmlAction.class, TTask.class).
        mapNulls(false).
        mapNullsInReverse(false).
        byDefault().
        register();
```

Výpis 6.6: Nastavenie mapovania elementu *Akcia*

Príklad predstavuje jednoduché mapovanie primitívnych dátových typov objektov z oboch strán. V prípade, že atribúty elementu sú reprezentované komplexnejšími triedami, je možné pre dané triedy nastaviť mapovanie pomocou nastavenia *field()*. Okrem toho je možné mapovanie plnohodnotne režírovať preťažením metódy *mapAtoB* v nastavení *customize()*.

Samotné mapovanie na všeobecný metamodel prevádza funkcia *transformIntoGeneralMetaModel*, ktorá z kolekcie *Elementov* vytvorí mapovaním nové *TElementy*, čo je vlastne predok všetkých ostatných špecifík elementov všeobecného metamodelu. Mapovací manažér si následne vo svojich nastaveniach zvolí potrebné mapovanie.

```
public List<TElement> transformIntoGeneralMetaModel(Model model) {
    UmlMapperManager mapperManager = new UmlMapperManager();
```

```
    List<Element> list = model.getElements();
    List<TElement> result = new ArrayList<TElement>();
    for (Element element : list) {
        element.setId(iteratorService.getId());
    }
    for (Element element : list) {
        if (element instanceof UmlAction &&
                !(element instanceof UmlAcceptSignalAction ||
                        element instanceof UmlSendSignalAction ||
                        element instanceof UmlAcceptTimeEvent)){
            TTask task = mapperManager.map(element, TTask.class);
            result.add(task);
        } else {
            TElement tel = mapperManager.map(element, TElement.class);
            result.add(tel);
        }
    }
    return result;
}
```

Výpis 6.7: Mapovacia funkcia v smere na všeobecný metamodel

Obdobne funguje aj mapovacia funkcia v smere zo všeobecného metamodelu. Jednotlivé mapovania elementov je možno vidieť v tabuľke 4.1 a v tabuľke 4.2.

6.3.4 Formálne obmedzenia

Vrámci formálnych obmedzení bolo v implementačnej časti tejto diplomovej práce riešené načítanie súboru *OWL*, spracovávanie prostredníctvom OWL API a následné uloženie ontológie. Posledným krokom bola implementácia použitia formálneho obmedzenia v klientskej aplikácii.

Načítanie súboru OWL bolo implementované využitím komponentu *react-dropzone*. Tento komponent umožňuje načítať súbor presunutím do priestoru komponentu alebo vyhľadaním cez súborový systém. Následne je súbor poslaný ako objekt typu *Multipartfile* na server.

Na serveri je súbor OWL validovaný a spracovávaný využitím OWL API prostredníctvom transakčnej metódy služby ontológii *saveOntology*, ktorá ako prvé vytvorí ontológiu zodpovedajúcu názvu importovaného OWL.

```
@Transactional
public Ontology saveOntology(User user, MultipartFile file) {
    //create ontology
```

```
Ontology ontology = new Ontology();
ontology.setName(file.getOriginalFilename());
ontology.setUser(user);
this.ontologyRepository.insert(ontology);
```

Výpis 6.8: Vytvorenie ontológie zo súboru

Následne je využitím OWL API vytvorená kolekcia elementov ontológie.

```
OWLOntologyManager m = OWLManager.createOWLOntologyManager();
OWLOntology o = m.loadOntologyFromOntologyDocument(file.getInputStream());

List<OntologyElement> ontologyElements = o.individualsInSignature()
        .map(element -> ontologyElementRepository.insert(new OntologyElement(
           null,element.getIRI().getRemainder().get(), ontology)))
        .collect(Collectors.toList());
```

Výpis 6.9: Vytvorenie kolekcie elementov ontológie

Ďalším krokom vrámci funkcie pre uloženie ontológie bolo filtrovanie axiómov jednotlivých elementov a vytvorenie vzťahov medzi elementami ontológie. Nasledujúci príklad zobrazuje vytvorenie kolekcie predchodcov pre aktuálne iterovaný element ontológie, a tiež vytvorenie a uloženie spomínaného vzťahu do databáze.

```
OntologyElement current = ontologyElements.get(i);
OWLNamedIndividual individual = o.individualsInSignature()
        .collect(Collectors.toList())
        .get(i);
List<OntologyElementRelation> relations = o.axioms(individual)
        .filter(element -> {
            if (element instanceof OWLObjectPropertyAssertionAxiom){
                OWLObjectPropertyAssertionAxiom axiom = ((
                    OWLObjectPropertyAssertionAxiom) element);//get axiom
                OWLObjectProperty owlObjectProperty = axiom.getProperty().
                    asOWLObjectProperty();//cast to object property
                return owlObjectProperty.getIRI().getRemainder().get().equals("
                    previous");//return true if property iri is previous;
            }
            return false;
        })
        .map(element -> {
```

```
        OntologyElement ontologyElement = ontologyElements
                .stream()
                .filter(oElement -> {
                    OWLObjectPropertyAssertionAxiom axiom = ((
                        OWLObjectPropertyAssertionAxiom) element);//get axiom
                    OWLNamedIndividual owlNamedIndividual =axiom.getObject().
                        asOWLNamedIndividual();//get axiom object as named
                        individual
                    return oElement.getText().equals(owlNamedIndividual.getIRI()
                        .getRemainder().get());//return ontology element with
                        same text
                })
                .findAny()
                .get();
        return ontologyElementRelationRepository.insert(new
            OntologyElementRelation(current,ontologyElement,null));
    })
    .collect(Collectors.toList());
```

Výpis 6.10: Príklad vytvorenia vzťahov z axiómov elementu ontológie prostredníctvom OWL API

Obdobným spôsobom boli vytvorené aj vzťahy pre prvky nasledovníkov elementu ontológie, čím končí metóda pre uloženie ontológie.

Kapitola 7

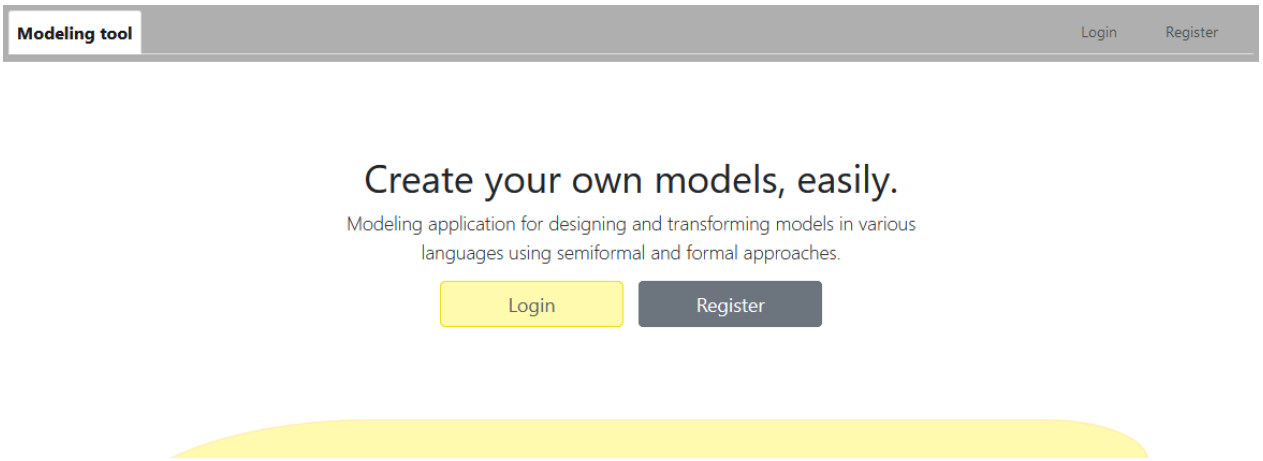
Vyhodnotenie

V tejto kapitole bude čitateľ oboznámený s dosiahnutými výsledkami vrámci diplomovej práce. V prvej časti bude prezentovaná ukážka aplikácie prostredníctvom vyvinutých obrazoviek. Následne budú prezentované príklady použitia transformácie a formálnych obmedzení. Na záver bude čitateľ oboznámený s vybranými riešenými problémami.

7.1 Ukážka aplikácie

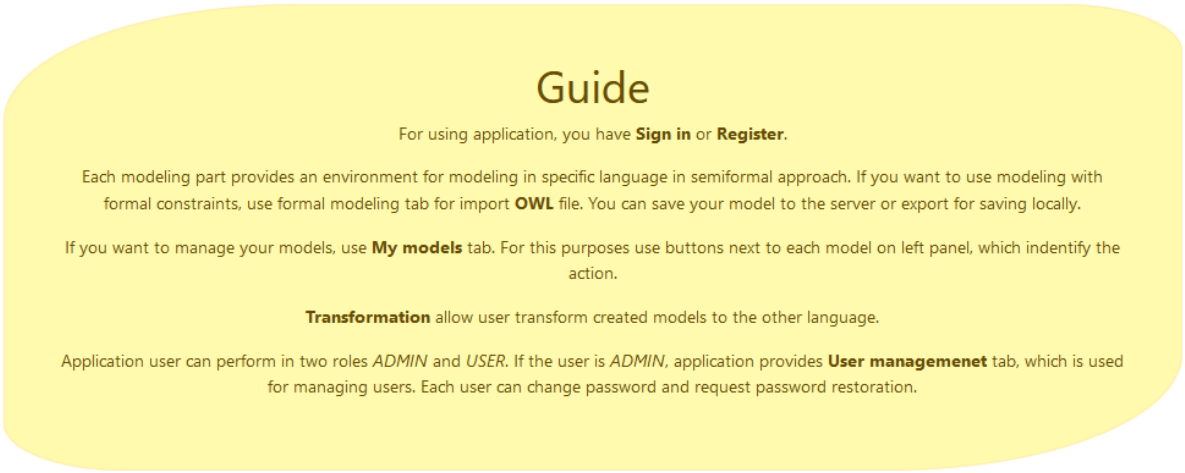
V kapitole 6 boli spomenuté vytvorené obrazovky vrámci tejto diplomovej práce. Nasledujúca časť je zameraná na detailnejší popis jednotlivých obrazoviek v chronologickom poradí. Jednotlivé ukážky aplikácie boli vytvorené na zariadení s operačným systémom Windows 10 a použitím prehliadača Google Chrome. V prípade potreby je možná inštalácia aplikácie podľa prílohy B).

7.1.1 Úvodná stránka



Obr. 7.1: Prvá časť úvodnej stránky

Po spustení webovej aplikácie sa používateľovi zobrazí úvodná stránka, ktorá pozostáva z dvoch častí. Prvá časť úvodnej stránky zobrazuj motivačný slogan pre používanie nástroja, krátky popis aplikácie a štartovacie tlačidla. V prípade, že používateľ nieje prihlásený, tlačidlá odkazujú na prihlásenie a registráciu. Druhá časť úvodnej stránky zobrazuje príručku práce s modelovacím systémom.

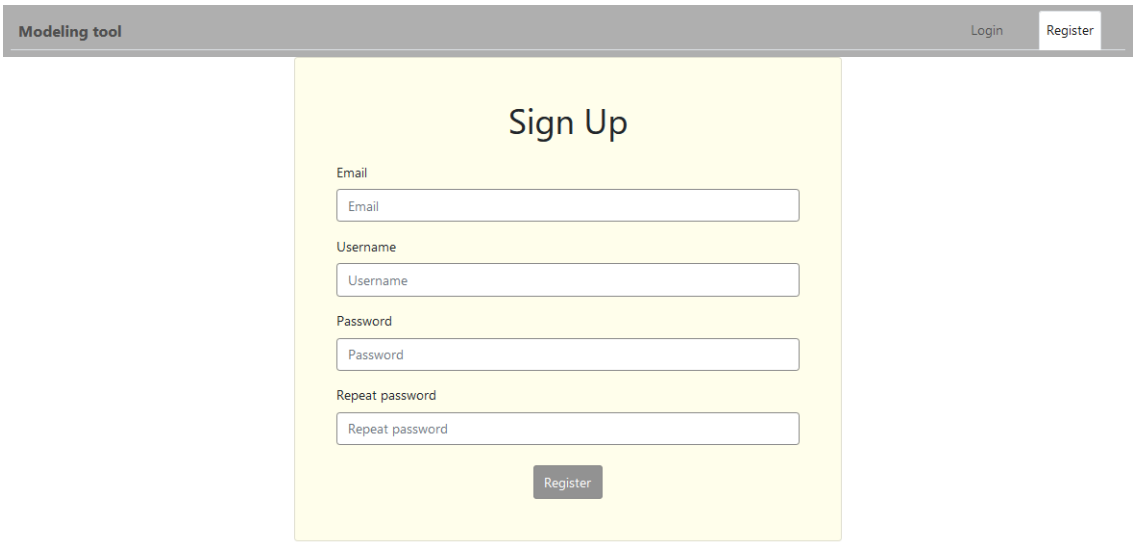


Obr. 7.2: Druhá časť úvodnej stránky

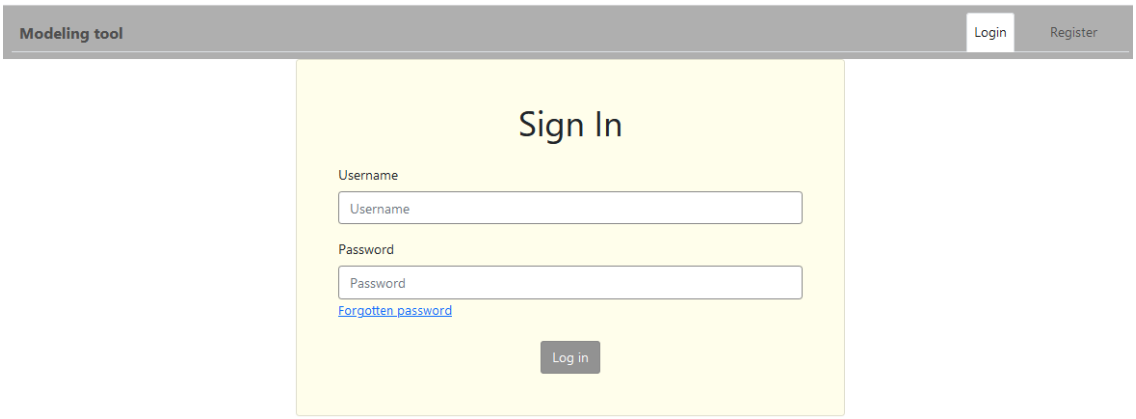
7.1.2 Registrácia a prihlásenie

Stlačením štartovacieho tlačidla alebo použitím položky navigačného menu s názvom *Registrácia (z ang. Register)*, je používateľovi zobrazený registračný formulár. Vyplnením formulára prebieha validácia jednotlivých položiek. V prípade chybnej validácie je používateľ vyzvaný na opravu, v opačnom prípade registrácia prebehne úspešne.

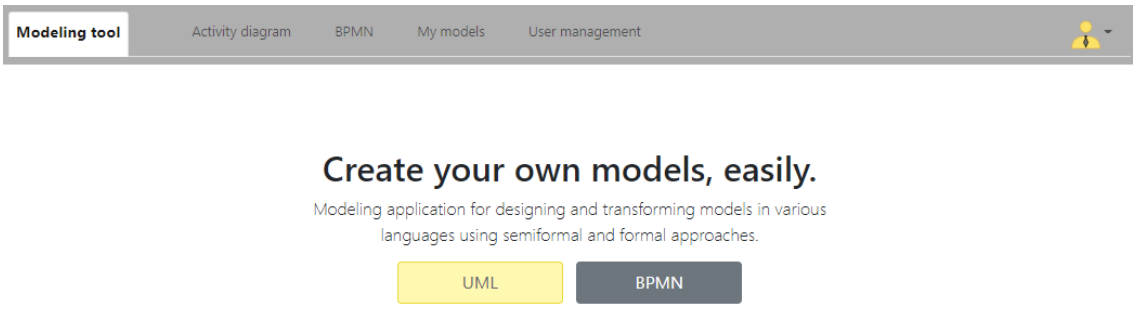
Pri použití štartovacieho tlačidla (alebo položky navigačného menu) pre prihlásenie, je používateľ presmerovaný na prihlasovací formulár, kde zadaním používateľského mena a hesla prebehne validácia, kontrola existencie daného používateľa, a nakoniec v prípade úspechu samotné prihlásenie. Prihlásením je používateľ presmerovaný na úvodnú stránku, kde nastane zmena štartovacích tlačidiel. Od doby keď sa používateľ prihlási, štartovacie tlačidla odkazujú na jednotlivé modelovacie nástroje. Okrem toho sú po prihlásení zobrazené položky navigačného menu, ktoré obsahujú jednotlivé komponenty systému. Používateľ v roli *ADMIN* má oproti používateľovi v roli *USER* navyše komponent pre správu používateľov.



Obr. 7.3: Registračný formulár



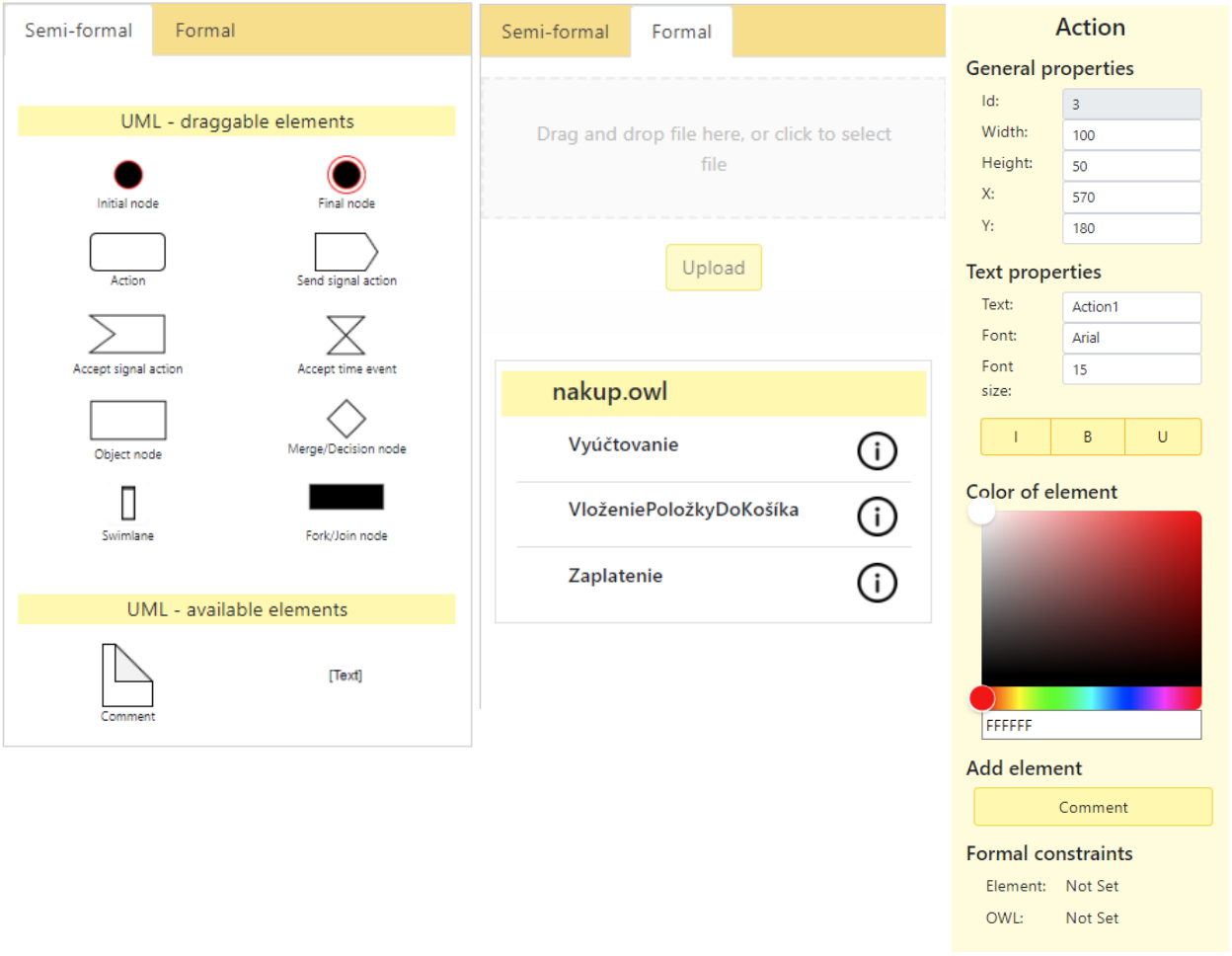
Obr. 7.4: Prihlasovací formulár



Obr. 7.5: Prvá časť úvodnej stránky po prihlásení

7.1.3 Modelovanie v notácii diagramu aktivít UML

Po výbere modelovacieho nástroja *Diagram aktivít UML* pomocou štartovacieho tlačidla alebo položkou navigačného menu, je používateľovi zobrazený daný modelovací nástroj. Pre prehľadnejší a viditeľnejší popis jednotlivých častí bude obrazovka rozdelená.

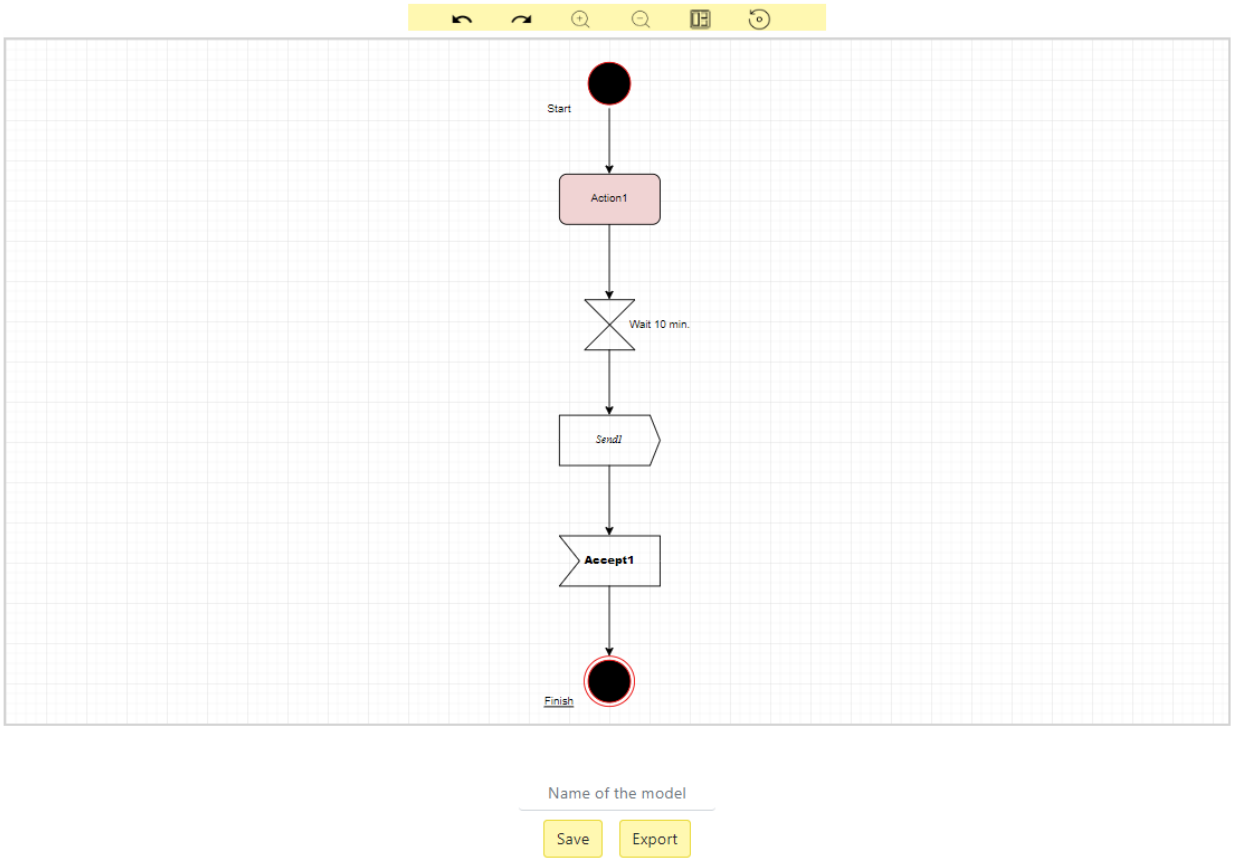


Obr. 7.6: Ľavé a pravé menu vrámi modelovacej časti

Ľavé menu pozostáva zo záložiek *Semiformálne (z ang. Semi-formal)* a *Formálne (z anf. Formal)*. Prvá záložka obsahuje jednotlivé elementy modelovacieho nástroja, ktoré je možné premiestniť na modelovacie plátno. Okrem toho obsahuje aj elementy, ktoré je možné využiť prostredníctvom výberu v pravom menu.

Druhá záložka je zložená z importovacieho vstupu, ktorý slúži pre vloženie formálneho obmedzenia. Jednotlivé formálne obmedzenia a ich elementy sú následne zobrazené pod importovacím vstupom. Používateľ môže výberom jednotlivých elementov priraďovať akciám dané formálne reprezentácie. Formálne elementy je tiež možné preskúmať stlačením tlačidla pre informácie o formálnom elementy, kde sú používateľovi zobrazení predchodcovia a nasledovníci daného elementu.

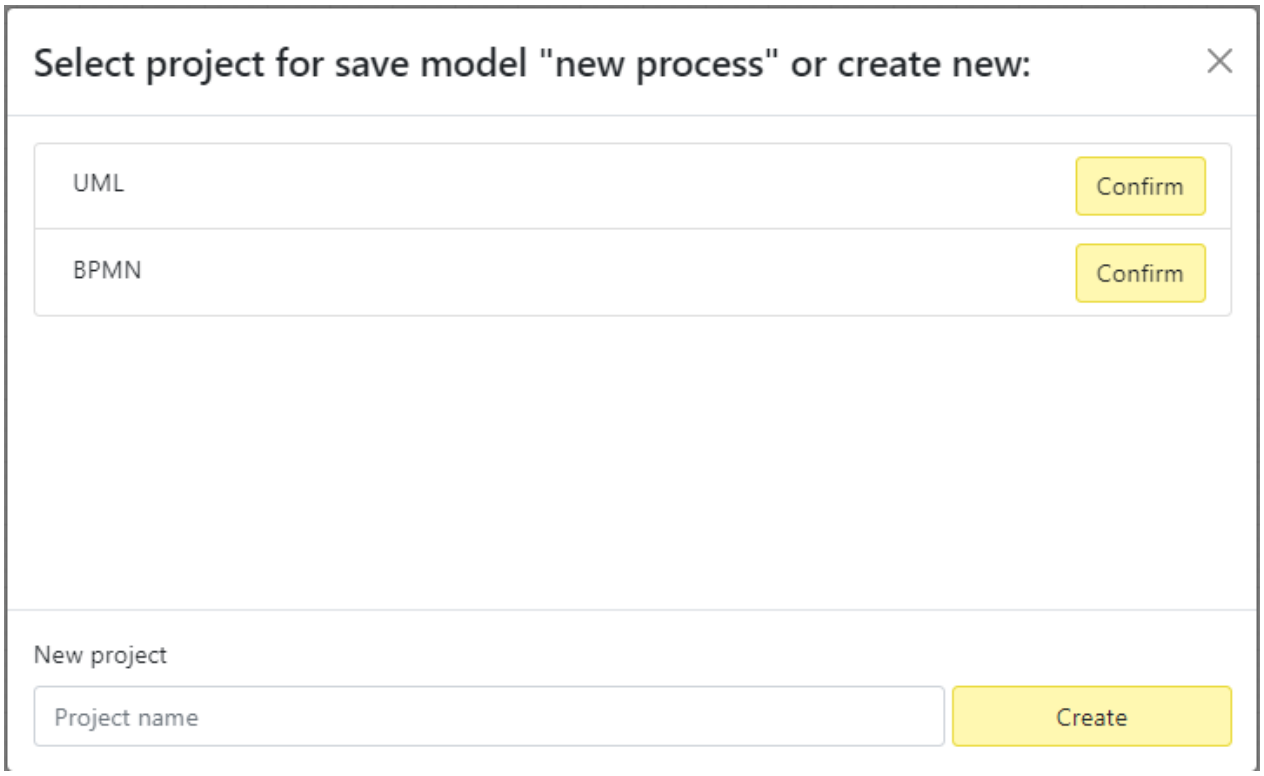
Posledné menu - pravé menu - slúži pre konfiguráciu elementov (nastavovanie šírky, výšky, pozície, formátovacích prvkov, priraďovanie elementov a pod.). Obsah pravého menu je premenlivý v závislosti na konfigurovanom elemente.



Obr. 7.7: Modelovacie plátno

Modelovacie plátno pozostáva z konfiguračnej lišty, kde je možné využiť akcie pre krok späť, krok vpred, priblíženie plátna, oddialenie plátna, automatické zarovnanie elementov či návrat na počiatočnú pozíciu na modelovacom plátne. Samotné plátno je nekonečne veľké, pričom pohyb po plátne je možný využitím pravého tlačidla myši. Pod modelovacím plátnom sa nachádza pole pre zadanie názvu modelu a tlačidlo pre uloženie a export modelu.

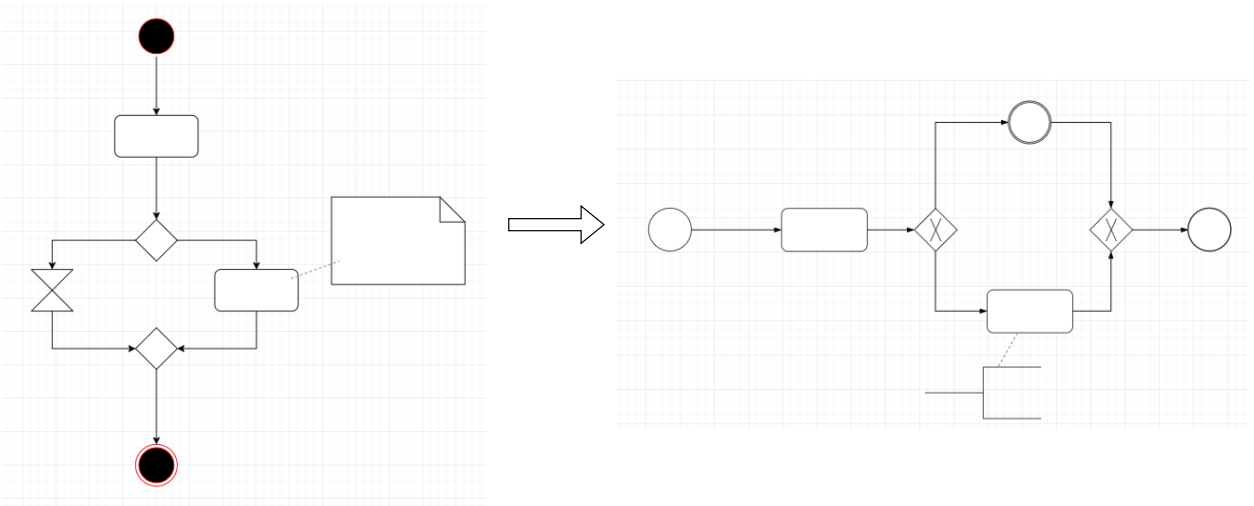
Po stlačení tlačidla pre uloženie modelu je zobrazené modálne okno, ktoré slúži pre prípadné vytvorenie a hlavne výber projektu do ktorého bude model uložený. Po uložení modelu je používateľ presmerovaný na obrazovku pre správu modelov, kde sú jednotlivé modely viditeľné. Táto obrazovka slúži tiež pre vyvolanie úpravy modelu alebo transformáciu modelu. Tieto akcie boli tiež vytvorené vrámci tejto diplomovej práce, avšak vizuálna podoba obrazovky pre správu modelov nebude prezentovaná, pretože bola vytvorená spolupracujúcim študentom.



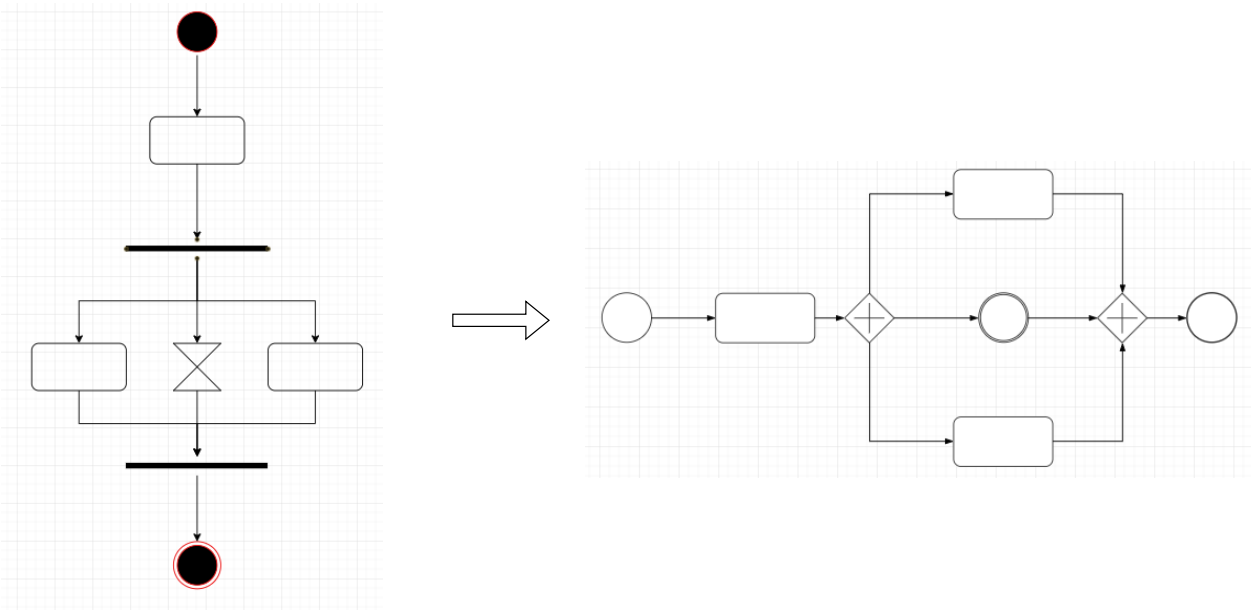
Obr. 7.8: Modálne okno pre uloženie modelu

7.2 Použitie transformácie

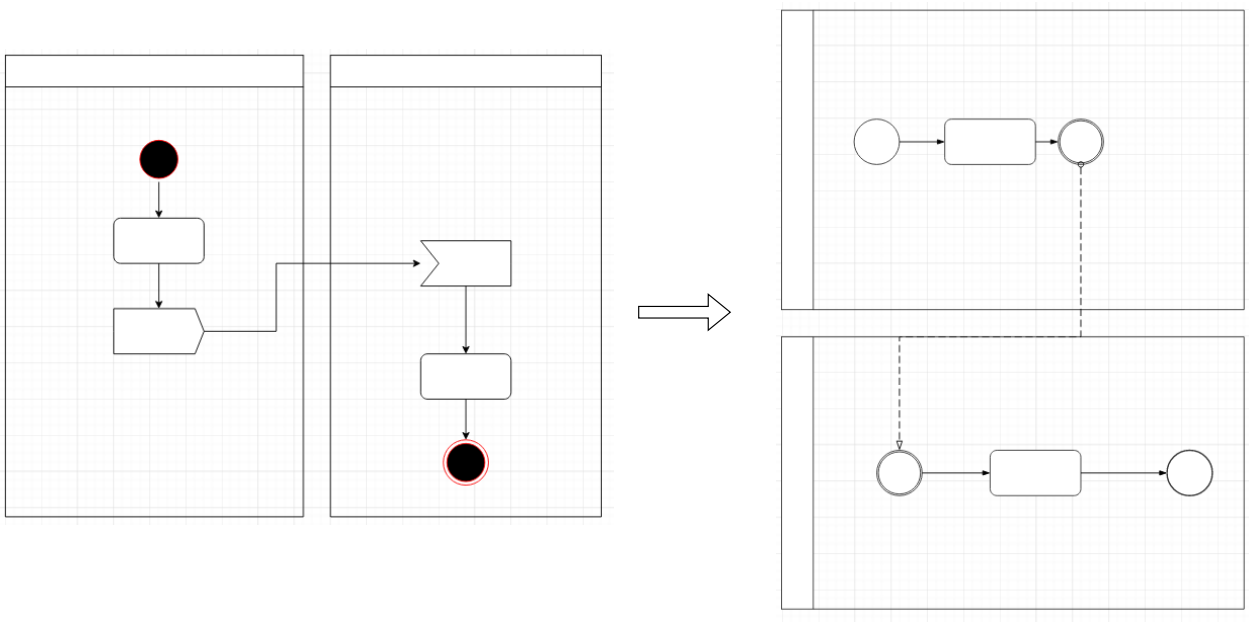
Pre demonštrovanie funkcionality transformácie boli prevedené testovacie prípady transformácie nešpecifikovaných procesných modelov medzi notáciami diagramu aktivít UML a BPMN.



Obr. 7.9: Testovacia transformácia 1



Obr. 7.10: Testovacia transformácia 2

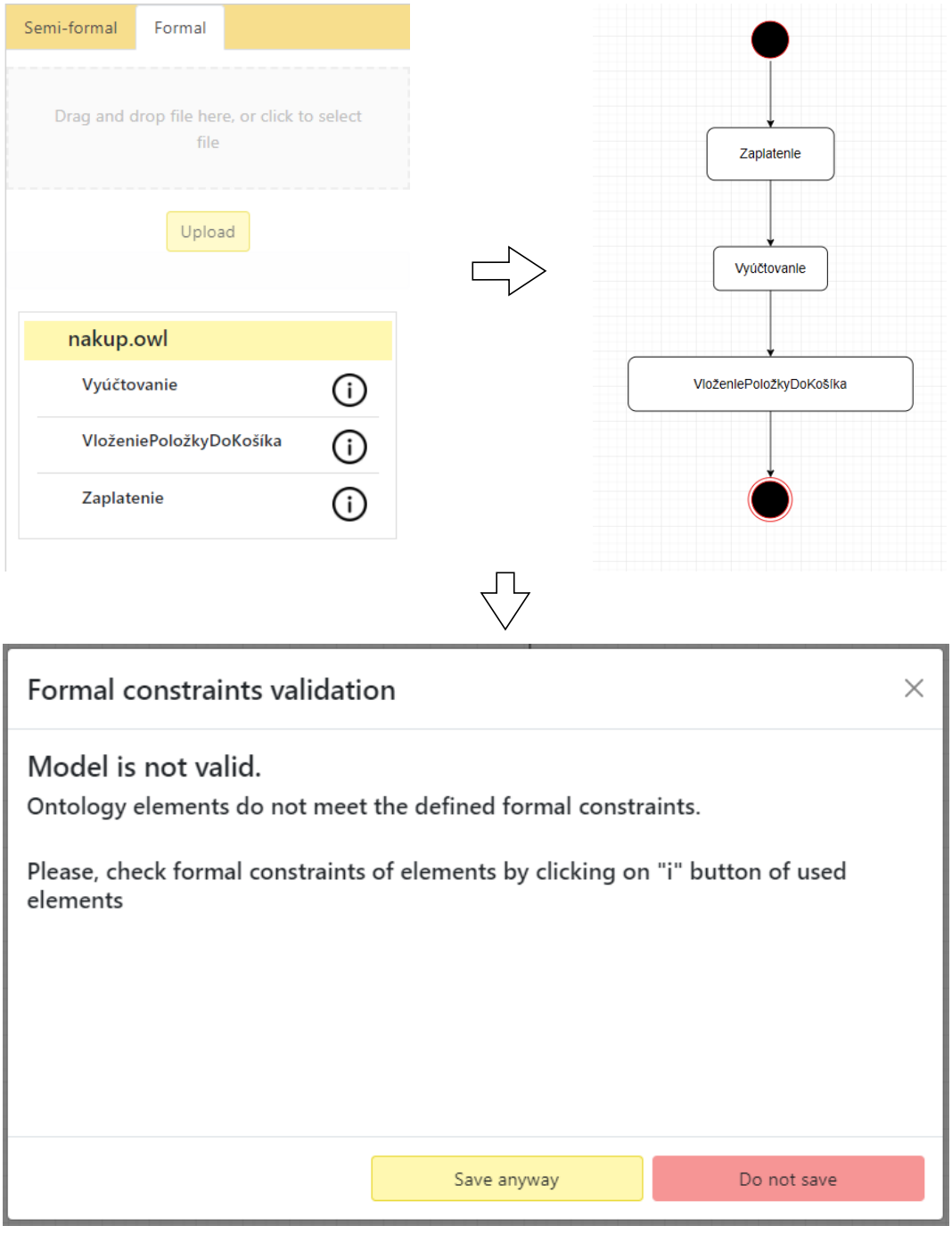


Obr. 7.11: Testovacia transformácia 3

7.3 Modelovanie s formálnym obmedzením

Validácia využitia formálneho obmedzenia prebieha pri ukladaní modelu. Pre príklad modelovania s využitím formálnych obmedzení bola importovaná definícia procesu nákupu (viď A). Nasledujúci

obrázok znázorňuje porušenie formálneho obmedzenia, keďže nákup je definovaný v postupnosti: *VloženiePoložkyDoKošíka*, *Vyúčtovanie*, *Zaplatenie*. Systém zobrazí hlásenie o chybnej validácii modely na úkor formálnych obmedzení. Používateľ sa môže rozhodnúť, či si chce daný model uložiť, alebo model upraví.



Obr. 7.12: Porušenie formálneho obmedzenia

7.4 Vybrané riešené problémy

Vyvinutím modelovacieho systému bol dokázaný spomínaný koncept transformácie (viď kapitola 4) a myšlienka využitia modelovania prostredníctvom formálnych obmedzení (viď kapitola 5). Napriek tomu pri vývoji aplikácie a priebežnom testovaní bolo narazené na niekoľko problémov.

Samotná vyvinutá aplikácia predstavuje použiteľný nástroj pre modelovanie, kde možno využívať formálne obmedzenia a tiež prevádzať transformácie do iných procesných modelovacích jazykov. Keďže sa však jedná o prototyp, a prioritnejším účelom diplomovej práce bolo dokázanie spomínaných konceptov modelovania, nie je naplno vyladená a systém niekedy dôjde do neočakávaného stavu. Dané skutočnosti však nastávajú veľmi ojedinele.

Ďalší problém nastáva pri transformácii, ktorá je umožnená vďaka mapovaniu metamodelu konkrétneho jazyka na všeobecný metamodel, pretože prináša z dôvodu prechodu na vyššiu úroveň abstrakcie určitú stratu špecifikácie elementu. Pre príklad uvediem prípad transformácie elementu diagramu aktivít UML *Spúšťacia akcia*, ktorý po prevode do modelovacej notácie BPMN a následnom spätnom prevode do diagramu aktivít UML, vystupuje ako ako element typu *Akcia*. Keďže neboli bližšie špecifikované typy udalostí vo všeobecnom metamodeli, bolo nutné pri mapovaní v smere zo všeobecného metamodelu vykonať mapovanie *Udalostí* na element typu *Akcia*. V praxi to však nespôsobuje problém s validáciou výsledného transformovaného modelu. Vylepšením tejto problematiky by mohlo byť rozšírenie všeobecného metamodelu o konkrétnejšie druhy udalostí.

Iným problémom transformácie bola horšia manipulácia s programovým zostrojením modelu prostredníctvom knižnice *MxGraph*, čo spôsobovalo, že niektoré elementy boli po transformácii rozhádzané. Pre zníženie dopadov tohto problému je používateľ pri transformácii ihneď v editačnom režime, pre dosiahnutie možnosti vizuálnej korektúry elementov modelu.

Kapitola 8

Záver

Hlavným cieľom tejto diplomovej práce bolo vytvorenie prototypu nástroja, ktorý kombinuje semiformálne modelovanie prostredníctvom notácie diagramu aktivít UML s využitím formálnych obmedzení a umožňuje transformovať vytvorené modely do iných procesných modelovacích jazykov. Vytvorený nástroj predstavuje dôkaz konceptu transformácie medzi modelmi rôznych modelovacích jazykov z dizertačnej práce M. A. Košinára z roku 2015 [12], a tiež dôkaz možnosti obmedzenia semiformálneho jazyka využitím formálnych obmedzení.

V kapitole 1 a 2 bol čitateľ uvedený do riešenej problematiky a bol oboznámený s pojmami dôležitými pre celkové pochopenie tejto diplomovej práce. Následne bola v kapitole 3 vypracovaná analýza porovnateľných modelovacích nástrojov vo webovom prostredí, ktoré vo svojej podstate vynikajú pokrytím veľkého množstva modelovacích jazykov či možnosťou kolaborácie. Analýzou však nebol nájdený nástroj, ktorý by umožňoval transformáciu do iných modelovacích notácii alebo formálne obmedzenie semiformálneho modelovacieho jazyka, čo je možné vnímať ako prínos tejto diplomovej práce.

Výsledkom riešenia problematiky transformácie, popísanej v kapitole 4, bolo zostrojenie metamodelu notácie diagramu aktivít jazyka UML a všeobecného metamodelu. Okrem toho bolo nutné popísať mapovanie medzi metamodelmi v oboch smeroch. Vďaka mapovaniu v oboch smeroch bolo umožnené prevedenie modelu diagramu aktivít na všeobecný model a opačne. Za predpokladu, že mapovanie prebieha medzi viacerými jazykmi a všeobecným metamodelom, je možné plnohodnotne transformovať medzi rôznymi modelovacími notáciami, čo bolo vytvorením nástroja aj dokázané.

Využitie formálnych obmedzení pri modelovaní prostredníctvom semiformálnej notácie diagramu aktivít jazyka UML bolo popísané vrámci kapitoly 5. Myšlienkou bolo sformalizovať definície procesov, pre zachovanie ich sémantiky, čo bolo umožnené využitím formálneho jazyka OWL. Vrámci definície procesu pomocou OWL je striktne definované poradie jednotlivých akcií pomocou predchodcov a následníkov jednotlivých akcii. Pri použití formálneho obmedzenia a uložení modelu tak dochádza prostredníctvom grafového algoritmu prehľadávania do hĺbky validácia modelovaného procesu.

Námetom na vylepšenie a ďalšie kroky pri riešení problematiky transformácie a semiformálneho modelovania s využitím formálnych obmedzení by mohlo byť umožnenie rozširovania vyvinutého modelovacieho systému o ďalšie modelovacie jazyky, čo by viedlo k umožneniu viacerých transformácii a tiež k dosiahnutiu iných uhlov pohľadu na modelovanú inštanciu. Pre tieto účely by mohol byť využitý aj formálny jazyk, pomocou ktorého by bol definovaný metamodel nového modelovacieho jazyka a mohol by byť následne importovaný do nástroja bez potreby ďalšieho programovania systému čo predstavuje silný potenciál do budúcna.

Literatura

1. *What is model?* [Online] [cit. 2022-01-13]. Dostupné z : https://serc.carleton.edu/introgeo/models/WhatIsAModel.html.
2. HESTENES, David. Modeling methodology for Physics teachers. 1997-03, roč. 399. Dostupné z DOI: 10.1063/1.53196.
3. SOWA, John F. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., Pacific Grove, CA, ©2000. Computer Science Series. ISBN 9780534949655.
4. MCDERMOTT, Patrick; SHARP, Alec. *Workflow Modeling: Tools for Process Improvement and Application Development, Artech House, Second Edition*. 2008.
5. SOMMERVILLE, Ian. *Software Engineering. 9th Edition*. 2010. International Computer Science Series. Harlow: Addison-Wesley. ISBN 9780137035151.
6. ROGER S. PRESSMAN, Bruce R. Maxim. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 2014. ISBN 9780078022128.
7. PFLEEGER, Shari L.; ATLEE, Joanne M. *Software Engineering: Theory and Practice*. Prentice Hall, 2009. An Alan R. Apt book. ISBN 9780136061694.
8. *What is process?* [Online] [cit. 2022-03-12]. Dostupné z : https://www.processmodel.com/blog/what-is-a-process/.
9. *What is business process?* [Online] [cit. 2022-03-16]. Dostupné z : https://www.techtarget.com/searchcio/definition/business-process.
10. *Scientific modeling* [online] [cit. 2022-02-23]. Dostupné z : https://en.wikipedia.org/wiki/Scientific_modelling#cite_note-4.
11. *Static and dynamic models* [online] [cit. 2022-03-17]. Dostupné z : https://www.ibm.com/docs/en/iis/9.1?topic=model-static-dynamic-models.
12. *Knowledge Support for Software Processes* [online] [cit. 2022-03-17]. Dostupné z : https://dspace.vsb.cz/handle/10084/110917.

13. *UML - Overview* [online] [cit. 2022-03-01]. Dostupné z : https://www.tutorialspoint.com/uml/uml_overview.htm.

14. *Úvod do UML* [online] [cit. 2022-03-01]. Dostupné z : https://www.itnetwork.cz/navrh/uml/uml-uvod-historie-vyznam-a-diagramy.

15. COOK, Steve; BOCK, Conrad; RIVETT, Pete; RUTT, Tom; SEIDEWITZ, Ed; SELIC, Bran; TOLBERT, Doug. *Unified Modeling Language (UML) Version 2.5.1*. 2017-12. Standard. Object Management Group (OMG). Dostupné tiež z: https://www.omg.org/spec/UML/2.5.1.

16. *Unified Modeling Language* [online] [cit. 2022-03-01]. Dostupné z : https://sk.wikipedia.org/wiki/Unified_Modeling_Language.

17. *UML, Meta Meta Models and Profiles* [online] [cit. 2022-03-06]. Dostupné z : https://www.uml-diagrams.org/uml-meta-models.html.

18. *Activity Diagram* [online] [cit. 2022-03-09]. Dostupné z : https://www.smartdraw.com/activity-diagram/?fbclid=IwAR16Gurb41WNjFM_Im5E_ZBXiQTWBtvpF7sH3_XCpL3qicg1RnO4u8x7Jvc.

19. *UML 2 Tutorial - Activity Diagram* [online] [cit. 2022-03-09]. Dostupné z : https://sparxsystems.com/resources/tutorials/uml2/activity-diagram.html?fbclid=IwAR38Jpd2CBrAWMtpOGb9GelAm9IfreYfzVSMR0TaesHA7IGkbbtYkD5jYY.

20. *About Us* [online] [cit. 2022-03-21]. Dostupné z : https://www.visual-paradigm.com/aboutus/.

21. *Discover the Benefits of Visual Paradigm* [online] [cit. 2022-03-21]. Dostupné z : https://www.visual-paradigm.com/features/.

22. *Intelligent diagramming Lucidchart* [online] [cit. 2022-03-24]. Dostupné z : https://www.lucidchart.com/pages/.

23. *Enterprise Lucidchart* [online] [cit. 2022-03-24]. Dostupné z : https://www.lucidchart.com/pages/enterprise.

24. *Diagrams for Confluence and Jira - draw.io* [online] [cit. 2022-03-24]. Dostupné z : https://drawio-app.com/.

25. *Enterprise - draw.io* [online] [cit. 2022-03-24]. Dostupné z : https://drawio-app.com/enterprise/.

26. *Examples - draw.io* [online] [cit. 2022-03-24]. Dostupné z : https://drawio-app.com/examples/.

27. *Ship Better Software Faster* [online] [cit. 2022-03-27]. Dostupné z : https://creately.com/solutions/creately-for-software-teams/.

28. *Ship Better Software Faster* [online] [cit. 2022-03-27]. Dostupné z : https://creately.com/lucidchart-alternative/.

29. *Edraw Software: Unlock Diagram Possibilities* [online] [cit. 2022-03-27]. Dostupné z : https://creately.com/lucidchart-alternative/.

30. *Význam ontológie* [online] [cit. 2022-04-02]. Dostupné z : https://slovak.encyclopedia-titanica.com/significado-de-ontolog.

31. *Ontology (definition)* [online] [cit. 2022-04-02]. Dostupné z : https://tomgruber.org/writing/definition-of-ontology.

32. *OWL* [online] [cit. 2022-04-02]. Dostupné z : https://www.w3.org/OWL/.

33. *An Introduction to the Semantic Web* [online] [cit. 2022-04-02]. Dostupné z : https://www.youtube.com/watch?v=V6BR9DrmUQA&ab_channel=CambridgeSemantics.

34. *OWL Web Ontology Language Overview* [online] [cit. 2022-04-04]. Dostupné z : http://www.ksl.stanford.edu/people/dlm/webont/OWLOverviewMay12003.htm.

35. *OWL Web Ontology Language Guide* [online] [cit. 2022-04-04]. Dostupné z : https://www.w3.org/TR/2004/REC-owl-guide-20040210/#BasicDefinitions.

36. *React - A JavaScript library for building user interfaces* [online] [cit. 2022-04-19]. Dostupné z : https://reactjs.org/.

37. *mxGraph 4.2.2* [online] [cit. 2022-04-19]. Dostupné z : https://jgraph.github.io/mxgraph/.

38. *Bootstrap* [online] [cit. 2022-04-19]. Dostupné z : https://getbootstrap.com/.

39. *Axios* [online] [cit. 2022-04-19]. Dostupné z : https://axios-http.com/.

40. *Spring vs Spring Boot: Know The Difference - InterviewBit* [online] [cit. 2022-04-22]. Dostupné z : https://www.interviewbit.com/blog/spring-vs-spring-boot/.

41. *Java persistence – JPA, Hibernate, ORM* [online] [cit. 2022-04-22]. Dostupné z : https://skillmea.sk/blog/java-persistence-jpa-hibernate-orm.

42. *Introduction to JSON Web Tokens* [online] [cit. 2022-04-22]. Dostupné z : https://jwt.io/introduction.

43. *PostgreSQL: About* [online] [cit. 2022-04-22]. Dostupné z : https://www.postgresql.org/about/.

Dodatok A

Definícia procesu nákupu pomocou OWL

```
<?xml version="1.0"?>
<rdf:RDF xmlns="urn:webprotege:ontology:182f1401-ded2-480f-913f-4e07ce56bfae#"
     xml:base="urn:webprotege:ontology:182f1401-ded2-480f-913f-4e07ce56bfae"
     xmlns:p="#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="urn:webprotege:ontology:182f1401-ded2-480f-913f-4
        e07ce56bfae"/>

    <!--
    ///////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////
     -->

    <!-- #next -->
    <owl:ObjectProperty rdf:about="#next">
        <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
            topObjectProperty"/>
        <rdfs:domain rdf:resource="#Action"/>
        <rdfs:range rdf:resource="#Action"/>
```

```
</owl:ObjectProperty>

<!-- #previous -->
<owl:ObjectProperty rdf:about="#previous">
    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
        topObjectProperty"/>
    <rdfs:domain rdf:resource="#Action"/>
    <rdfs:range rdf:resource="#Action"/>
</owl:ObjectProperty>

<!--
///////////////////////////////////////////////////////////
//
// Classes
//
///////////////////////////////////////////////////////////
 -->

<!-- #Action -->
<owl:Class rdf:about="#Action"/>

<!--
///////////////////////////////////////////////////////////
//
// Individuals
//
///////////////////////////////////////////////////////////
 -->

<!-- #Zaplatenie -->
<owl:NamedIndividual rdf:about="#Zaplatenie">
    <rdf:type rdf:resource="#Action"/>
    <p:previous rdf:resource="#VloženiePoložkyDoKošíka"/>
    <p:previous rdf:resource="#Vyúčtovanie"/>
</owl:NamedIndividual>

<!-- #VloženiePoložkyDoKošíka -->
<owl:NamedIndividual rdf:about="#VloženiePoložkyDoKošíka">
```

```
        <rdf:type rdf:resource="#Action"/>
        <p:next rdf:resource="#Zaplatenie"/>
        <p:next rdf:resource="#Vyúčtovanie"/>
    </owl:NamedIndividual>

    <!-- #Vyúčtovanie -->
    <owl:NamedIndividual rdf:about="#Vyúčtovanie">
        <rdf:type rdf:resource="#Action"/>
        <p:next rdf:resource="#Zaplatenie"/>
        <p:previous rdf:resource="#VloženiePoložkyDoKošíka"/>
    </owl:NamedIndividual>
</rdf:RDF>

<!-- Generated by the OWL API (version 4.5.13) https://github.com/owlcs/owlapi -->
```

Výpis A.1: Príklad OWL

Dodatok B

Inštalácia aplikácie

V ranných fázach tejto diplomovej práce bol pri vytvorení projektu modelovacej aplikácie kladený dôraz aj na jednoduchú prenositeľnosť aplikácie na iné zariadenia. Častokrát sa stáva, že systém pôsobí ako fungujúci na vývojovom zariadení, avšak pri presune na produkčný server alebo na iné zariadenie môže byť opak pravdou. Problémy môžu nastať z dôvodu odlišnej konfigurácie systému na inom zariadení, alebo na produkčnom serveri, oproti konfigurácii vývojového zariadenia.

Tento problém bol vrámci diplomovej práce vyriešený použitím nástroja *Docker*. Pre jednotlivé systémové jednotky boli definované tzv. *docker súbory (z ang. dockerfile)*, ktoré obsahujú príkazy pre zostrojenie *docker obrazov (z ang. docker images)*. Následne bol vytvorený konštrukčný súbor - *docker-compose.yml* - ktorý jednotlivé docker obrazy združuje. Po zadaní príkazu *docker-compose up -d* sú následne zostrojené obrazy jednotlivých systémových jednotiek ktoré sú spustené vo virtuálnom prostredí softvéru *Docker*.

B.1 Podmienky pre inštaláciu a postup inštalácie

Podmienkou inštalácie aplikácie je nainštalovaný softvér Docker, prístup na internet a internetový prehliadač (odporúčam Google Chrome). Následne je nutné stiahnuť priložený softvér k diplomovej práci a v koreňovom adresári projektu spustiť príkaz *docker-compose up -d*. Príkazom sa nainštaluje potrebná databáza, serverová aj klientska aplikácia. Aplikáciu je následne možné spustiť prostredníctvom prehliadača zadaním adresy *http://localhost:3000/*.

B.2 Možné chyby pri inštalácii

Na niektorých operačných systémoch môže nastať problém so zostavením serverovej aplikácie. Zostavenie zlyhá, pretože pri zostavovaní obrazu serverovej aplikácie, konkrétne pri použití *maven wrapperu mvnw*, docker nieje schopný prekopírovať súbor mvnw a následne ho využiť pri ďalších príkazoch. Systém hlási chybu *ERROR [build 6/9] RUN ./mvnw dependency:go-offline -B ./mvnw:*

not found. Je to spôsobené kvôli odriadkovaniu v súbore *mvnw*, ktorý používa odriadkovanie typu *CRLF*. Tento problém je možné odstrániť využitím nástroja *dos2unix* na súbor mvnw, ktorý upraví odriadkovanie z typu *CRLF* na *LF*.