Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-5220-72271

Bc. Martin Olejár

DETEKCIA SÉMANTICKÝCH KONFLIKTOV V SOFTVÉROVÝCH MODELOCH

Diplomová práca

Vedúci práce: Ing. Karol Rástočný, PhD.
máj 2018

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-5220-72271

Bc. Martin Olejár

DETEKCIA SÉMANTICKÝCH KONFLIKTOV V SOFTVÉROVÝCH MODELOCH

Diplomová práca

Študijný program:	Softvérové inžinierstvo
Študijný odbor:	9.2.5 Softvérové inžinierstvo
Miesto vypracovania:	Ústav informatiky, informačných systémov a softvérového inžinierstva, FIIT STU v Bratislave
Vedúci práce:	Ing. Karol Rástočný, PhD.

máj 2018

Zadanie diplomovej práce

Meno študenta: **Bc. Martin Olejár**

Študijný program: Softvérové inžinierstvo

Študijný odbor: Softvérové inžinierstvo

Názov práce: **Detekcia a vizualizácia konfliktov v softvérových modeloch**

Samostatnou výskumnou a vývojovou činnosťou v rámci predmetov Diplomový projekt I, II, III vypracujte diplomovú prácu na tému, vyjadrenú vyššie uvedeným názvom tak, aby ste dosiahli tieto ciele:

Všeobecný cieľ:

Vypracovaním diplomovej práce preukážte, ako ste si osvojili metódy a postupy riešenia relatívne rozsiahlych projektov, schopnosť samostatne a tvorivo riešiť zložité úlohy aj výskumného charakteru v súlade so súčasnými metódami a postupmi študovaného odboru využívanými v príslušnej oblasti a schopnosť samostatne, tvorivo a kriticky pristupovať k analýze možných riešení a k tvorbe modelov.

Špecifický cieľ:

Vytvorte riešenie zodpovedajúce návrhu textu zadania, ktorý je prílohou tohto zadania. Návrh bližšie opisuje tému vyjadrenú názvom. Tento opis je záväzný, má však rámcový charakter, aby vznikol dostatočný priestor pre Vašu tvorivosť.

Riaďte sa pokynmi Vášho vedúceho.

Pokiaľ v priebehu riešenia, opierajúc sa o hlbšie poznanie súčasného stavu v príslušnej oblasti, alebo o priebežné výsledky Vášho riešenia, alebo o iné závažné skutočnosti, dospejete spoločne s Vaším vedúcim k presvedčeniu, že niečo v texte zadania a/alebo v názve by sa malo zmeniť, navrhnite zmenu. Zmena je spravidla možná len pri dosiahnutí kontrolného bodu.

Miesto vypracovania: Ústav informatiky, informačných systémov a softvérového inžinierstva, FIIT STU Bratislava

Vedúci práce: **Ing. Karol Rástočný, PhD.**

Termíny odovzdania:

Podľa harmonogramu štúdia platného pre semester, v ktorom máte príslušný predmet (Diplomový projekt I, II, III) absolvovať podľa Vášho študijného plánu

Predmety odovzdania:

V každom predmete dokument podľa pokynov na www.fiit.stuba.sk v časti: home > Informácie o > štúdiu > organizácia štúdia > diplomový projekt.

V Bratislave dňa 13. 2. 2017

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4
1

prof. Ing. Pavol Návrat, PhD.
riaditeľ Ústavu informatiky, informačných systémov a softvérového inžinierstva

STU FIIT
SLOVENSKÁ TECHNICKÁ
UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY
A INFORMAČNÝCH TECHNOLÓGIÍ

Návrh zadania diplomovej práce

Finálna verzia do diplomovej práce [1]

Študent:

Meno, priezvisko, tituly:	Martin Olejár, Bc.
Študijný program:	Softvérové inžinierstvo
Kontakt:	martin.olejar@centrum.sk

Výskumník:

Meno, priezvisko, tituly:	Karol Rástočný, Ing. PhD.

Projekt:

Názov:	Detekcia a vizualizácia konfliktov v softvérových modeloch
Názov v angličtine:	Conflict detection and visualization in software models
Miesto vypracovania:	Ústav informatiky, informačných systémov a softvérového inžinierstva, FIIT STU, Bratislava
Oblasť problematiky:	modelovanie softvéru, vizualizácia softvéru

Text návrhu zadania[2]

Vývoj softvérových systémov zahŕňa vytváranie rôznych verzií modelov vyvíjaných systémov. Medzi jednotlivými verziami modelov je potrebné identifikovať jednotlivé zmeny a detegovať konfliktné zmeny, ktoré často vznikajú hlavne pri paralelnej práci vývojárov. Správna detekcia konfliktov a ich vhodná vizualizácia vytvárajú dobrý predpoklad na vyriešenie týchto konfliktov a umožnenie následnej synchronizácie jednotlivých verzií modelov. Taktiež vďaka vhodnej vizualizácii konfliktných zmien môžu vývojári predchádzať chybám v modeloch a tak uvoľnenejšie pracovať na svojich úlohách.

Analyzujte problematiku detekcie, vizualizácie a riešenia konfliktov v softvérových modeloch a synchronizácie jednotlivých verzií softvérových modelov. Navrhnite novú, resp. použite existujúcu metódu pre identifikáciu zmien v UML modeli a následne navrhnite vhodnú metódu na detekciu, vizualizáciu a riešenie konfliktov v UML modeloch. Navrhnuté riešenie overte prostredníctvom implementovaného softvérového prototypu.

---

[1] Vytlačiť obojstranne na jeden list papiera

[2] 150-200 slov (1200-1700 znakov), ktoré opisujú výskumný problém v kontexte súčasného stavu vrátane motivácie a smerov riešenia

Literatúra[3]

- COSTA, Valéria Oliveira; MONTEIRO, Rodrigo; MURTA, Leonardo. Detecting Semantic Equivalence in UML Class Diagrams. In: SEKE. 2014. p. 318-323.
- BROSCH, Petra, et al. Conflict Visualization for Evolving UML Models. Journal of Object Technology, 2012, 11.3: 2:1-30.

Vyššie je uvedený návrh diplomového projektu, ktorý vypracoval(a) Bc. Martin Olejár, konzultoval(a) a osvojil(a) si ho Ing. Karol Rástočný, PhD. a súhlasí, že bude takýto projekt viesť v prípade, že bude pridelený tomuto študentovi.

V Bratislave dňa 24.1.2017

Podpis študenta

Podpis výskumníka

Vyjadrenie garanta predmetov Diplomový projekt I, II, III

Návrh zadania schválený: áno / nie[4]

Dňa: 13. 2. 2017

Podpis garanta predmetov

[3] 2 vedecké zdroje, každý v samostatnej rubrike a s údajmi zodpovedajúcimi bibliografickým odkazom podľa normy STN ISO 690, ktoré sa viažu k téme zadania a preukazujú výskumnú povahu problému a jeho aktuálnosť (uveďte všetky potrebné údaje na identifikáciu zdroja, pričom uprednostnite vedecké príspevky v časopisoch a medzinárodných konferenciách)

[4] Nehodiace sa prečiarknite

STU FIIT
SLOVENSKÁ TECHNICKÁ
UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY
A INFORMAČNÝCH TECHNOLÓGIÍ

Návrh zadania diplomovej práce

Revízia č.: 1 [1]

Študent:

Meno, priezvisko, tituly: Martin Olejár, Bc.
Študijný program: Softvérové inžinierstvo
Kontakt: martin.olejar@centrum.sk

Výskumník:

Meno, priezvisko, tituly: Karol Rástočný, Ing. PhD.

Projekt:

Názov:	Detekcia sémantických konfliktov v softvérových modeloch
Názov v angličtine:	Semantic conflict detection in software models
Miesto vypracovania:	Ústav informatiky, informačných systémov a softvérového inžinierstva, FIIT STU, Bratislava
Oblasť problematiky:	modelovanie softvéru, detekcia konfliktov

Text návrhu zadania[2]

Vývoj softvérových systémov zahŕňa vytváranie rôznych verzií modelov vyvíjaných systémov. Medzi jednotlivými verziami modelov je potrebné v procese ich synchronizácie identifikovať jednotlivé zmeny a detegovať konfliktné zmeny, ktoré často vznikajú hlavne pri paralelnej práci vývojárov. Správna detekcia konfliktov a ich vhodná vizualizácia vytvárajú dobrý predpoklad na vyriešenie týchto konfliktov a umožnenie následnej synchronizácie jednotlivých verzií modelov. Detekcia konfliktov je dôležitá aj z hľadiska predchádzania vzniku chýb v modeloch, čo priamo ovplyvňuje výslednú kvalitu modelov vo vývoji softvérových systémov a zdrojového kódu vytvoreného na základe modelov.

Analyzujte problematiku konfliktov v softvérových modeloch, ktoré vznikajú pri paralelnej práci vývojárov. Analyzujte metódy a nástroje, ktoré sa zameriavajú na ich detekciu. Navrhnite novú, resp. použite existujúcu metódu na identifikáciu zmien v softvérových modeloch a následne navrhnite vhodnú metódu na detekciu sémantických konfliktov v softvérových modeloch. Navrhnuté riešenie overte prostredníctvom implementovaného softvérového prototypu.

---

[1] Vytlačiť obojstranne na jeden list papiera

[2] 150-200 slov (1200-1700 znakov), ktoré opisujú výskumný problém v kontexte súčasného stavu vrátane motivácie a smerov riešenia

Literatúra[3]

- COSTA, Valéria Oliveira; MONTEIRO, Rodrigo; MURTA, Leonardo. Detecting Semantic Equivalence in UML Class Diagrams. In: SEKE. 2014. p. 318-323.
- BROSCH, Petra, et al. Conflict Visualization for Evolving UML Models. Journal of Object Technology, 2012, 11.3: 2:1-30.

Vyššie je uvedený návrh diplomového projektu, ktorý vypracoval(a) Bc. Martin Olejár, konzultoval(a) a osvojil(a) si ho Ing. Karol Rástočný, PhD. a súhlasí, že bude takýto projekt viesť.

V Bratislave dňa 13.2.2018

Podpis študenta

Podpis výskumníka

Vyjadrenie garanta predmetov Diplomový projekt I, II, III

Návrh zadania schválený: áno / nie[4]

Dňa: 26. 2. 2018

Podpis garanta predmetov

[3] 2 vedecké zdroje, každý v samostatnej rubrike a s údajmi zodpovedajúcimi bibliografickým odkazom podľa normy STN ISO 690, ktoré sa viažu k téme zadania a preukazujú výskumnú povahu problému a jeho aktuálnosť (uveďte všetky potrebné údaje na identifikáciu zdroja, pričom uprednostnite vedecké príspevky v časopisoch a medzinárodných konferenciách)

[4] Nehodiace sa prečiarknite

ČESTNÉ VYHLÁSENIE

Čestne vyhlasujem, že som túto prácu vypracoval samostatne, na základe konzultácií a s použitím uvedenej literatúry.

V Bratislave, 3. 5. 2018 .....

Bc. Martin Olejár

POĎAKOVANIE

Chcel by som sa poďakovať vedúcemu mojej diplomovej práce, Ing. Karolovi Rástočnému, PhD., za jeho odborné vedenie, všetky cenné rady a všetok vynaložený čas, ktorými mi výrazne pomohol pri písaní tejto práce.

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Softvérové inžinierstvo

Autor: Bc. Martin Olejár
Diplomová práca: Detekcia sémantických konfliktov v softvérových modeloch
Vedúci diplomovej práce: Ing. Karol Rástočný, PhD.
máj 2018

Súčasťou vývoja softvérového systému je vytváranie rôznych verzií modelov vyvíjaného systému, ktoré prechádzajú neustálymi zmenami. Pri paralelnej práci členov vývojového tímu je potrebné tieto verzie modelov v procese synchronizácie zlúčiť do 1 verzie modelu. Tento proces zahŕňa identifikáciu zmien, ktoré boli v jednotlivých verziách modelov vykonané, a následne detekciu konfliktných zmien, ktoré sa často vyskytujú pri paralelnom vývoji a vedú k vzniku konfliktov v zlúčenej verzii modelu. Problémom je hlavne detekcia sémantických konfliktov, pri ktorej je potrebné uvažovať sémantiku modelu. Práve sémantické konflikty sú často detegované až pri manuálnej inšpekcii modelov, sú pôvodcom skrytých chýb v modeloch a môžu spôsobiť neplatnosť zlúčenej verzie modelu. V tejto práci analyzujeme problematiku konfliktov v softvérových modeloch, ktoré vznikajú pri paralelnom vývoji. Ponúkame kategorizáciu konfliktov a prehľad existujúcich algoritmov a nástrojov, ktoré sa zameriavajú na detekciu konfliktov v modeloch. Hlavnou časťou práce je metóda zameraná na detekciu sémantických konfliktov v modeloch a jej vyhodnotenie prostredníctvom softvérového prototypu. Táto metóda transformuje v procese synchronizácie jednotlivé verzie modelov do ontológie, používa mapovanie ontológií na identifikáciu sémantických vzťahov medzi prvkami modelu a vykonaných zmien vo verziách modelu a deteguje konflikty na základe definovaných podmienok.

Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree course: Software Engineering

Author: Bc. Martin Olejár
Master's Thesis: Semantic conflict detection in software models
Supervisor: Dr. Karol Rástočný
2018, May

Part of software system development is creation of different model versions of developed system that undergo significant changes. During parallel work of developer team, it is necessary to merge these model versions into one model version in process of synchronization. This process includes identification of changes that were made in model versions, and detection of conflict changes which often occur during parallel development and lead to occurrence of conflicts in merged model version. Main problem is semantic conflict detection where it is necessary to consider semantics of model. Semantic conflicts are often detected during manual inspection of models, produce latent defects in models and could cause invalidity of merged model version. In this thesis, we analyse issue of conflicts in software models that occur during parallel development. We offer categorization of conflicts and overview of existing algorithms and tools that deal with conflict detection. Main part of thesis consists of method focused on detection of semantic conflicts in models and its evaluation by software prototype. In synchronization process, method transforms model versions into ontology, uses ontology mapping algorithm on identification of semantic relationships between model elements and changes made in model versions and detects conflicts based on defined conditions.

Obsah

1 Úvod

V súčasnosti sú modely dôležitými prvkami počas celého vývoja systému. Poskytujú nielen určitú úroveň abstrakcie systému, ale aj základ pre implementáciu kódu. Na vývoji systému sa podieľa veľký počet ľudí, ktorí často pracujú paralelne s cieľom efektívneho využitia času a prostriedkov, pričom vytvárajú rôzne verzie modelov.

V tejto práci sa zaoberáme prípadom, kde dvaja vývojári nezávisle od seba pracujú vo svojej verzii modelu, ktorá je odvodená od základnej verzie modelu uloženej v spoločnom repozitári. Po uskutočnení všetkých zmien v modeli vložia svoju verziu do spoločného repozitára s cieľom pridania vykonaných zmien do zlúčenej verzie. Tu najprv vzniká problém identifikácie zmien, ktoré boli v jednotlivých verziách modelu vykonané. V praxi sa využívajú hlavne prístupy založené na operáciách a prístupy založené na stavoch.

Pre prístupy, ktoré sú založené na operáciách, platí, že zmeny vo verzii modelu sú zaznamenávané a uložené hneď po ich vykonaní (Koegel et al., 2010). Na druhej strane, prístupy založené na stavoch uchovávajú len stav modelu, čo vyžaduje nájdenie rozdielov medzi základnou a odvodenou verziou modelu prostredníctvom porovnania ich stavov.

Ďalším otvoreným problémom je detekcia konfliktných zmien, resp. konfliktov, ktoré sa často môžu vyskytnúť hlavne pri paralelnej práci. Kvalitná metóda na detekciu konfliktov by mala detegovať čo najviac konfliktov. Konflikty väčšinou vznikajú v prípade, že vývojári vykonajú navzájom si odporujúce zmeny, napr. jeden vývojár odstráni prvok modelu vo svojej verzii, ktorý druhý vývojár paralelne modifikuje vo svojej verzii modelu.

Konflikt však môže vzniknúť aj v dôsledku vykonania rozdielnych zmien, ktoré sémanticky vyjadrujú rovnaký zámer vývojárov. Príkladom môže byť pridanie rôznych prvkov do jednotlivých verzií modelu, ktoré vyjadrujú tú istú vec na základe ich statických a behaviorálnych vlastností. V tom prípade je detekcia konfliktov komplikovanejšia, keďže je nutné sledovať aj sémantiku modelu a vykonaných zmien.

Po detekcii je potrebná vhodná vizualizácia konfliktov spolu s časťami modelu, ku ktorým sa vzťahujú. Táto vizualizácia by mala pomôcť vývojárom detegované konflikty správnym spôsobom vyriešiť. Samotné riešenie konfliktov

je možné nechať na vývojárov, na druhej strane existujú prístupy umožňujúce automatické riešenie konfliktov (Barrett et al., 2011).

Proces detekcie a riešenia konfliktov je súčasťou mnohých verziovacích systémov, ktoré sú z hľadiska koordinácie vykonávania zmien v modeli viacerými vývojármi založené na optimistickom prístupe (Koegel et al., 2010). Optimistický prístup nebráni vzniku navzájom sa prekrývajúcich konfliktných zmien, avšak poskytuje prostriedky pre ich detekciu a riešenie. Tento prístup lepšie podporuje paralelnú prácu viacerých vývojárov, keďže nie je nutné zamykanie časti modelu ani pripojenie do internetovej siete. V praxi je väčšina systémov založená na tomto prístupe.

Hlavným cieľom tejto diplomovej práce je návrh a overenie metódy zameranej na detekciu statických sémantických a ontologických konfliktov. Ich detekcia je dôležitá hlavne preto, aby sa nevyskytovali v zlúčenej verzii modelu, čím je možné predchádzať vzniku neplatnej zlúčenej verzie modelu, vzniku chýb v modeloch a v zdrojovom kóde vytvorenom na základe modelov.

Detekciu statických sémantických a ontologických konfliktov sme si vybrali na základe vykonanej analýzy dostupných metód a nástrojov, ktoré sa zaoberajú detekciou konfliktov. Vykonaná analýza nám ukázala, že stále existujúcim problémom je detekcia sémantických konfliktov, v ktorej je potrebné zahrnúť aj sémantiku modelu. Navyše v súčasnosti chýba metóda, ktorá sa zameriava na detekciu ontologických konfliktov.

Štruktúra práce je nasledujúca. V kapitole 2 sa venujeme kategorizácii konfliktov a možným spôsobom ich vzniku. Kapitola 3 predstavuje prehľad dostupných nástrojov a metód, ktoré sa zaoberajú okrem iného aj detekciou konfliktov. V kapitole 4 podrobne rozoberáme jednotlivé časti navrhovanej metódy na detekciu statických sémantických a ontologických konfliktov v UML modeli tried. Kapitola 5 obsahuje vyhodnotenie detekcie konfliktov pomocou navrhovanej metódy spolu s opisom implementácie prototypu.

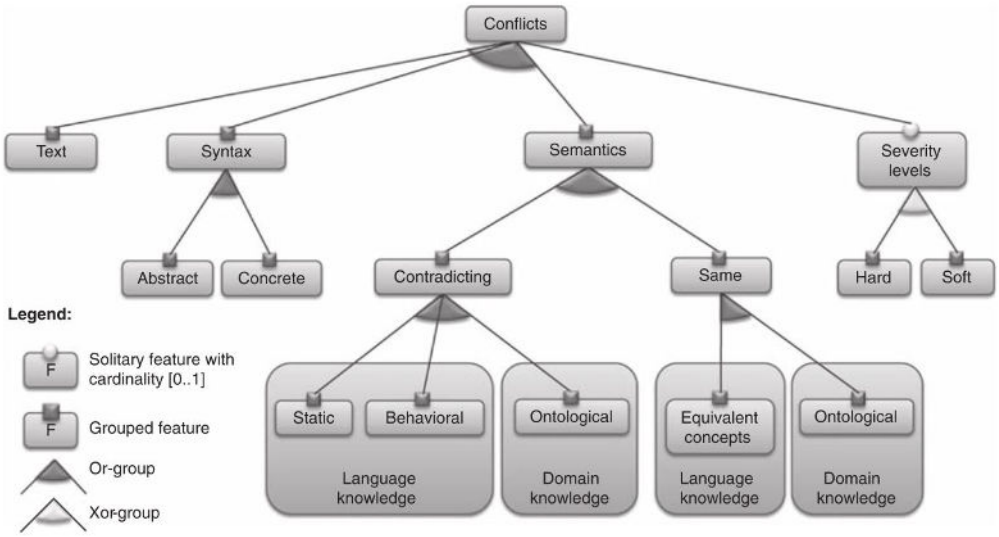
2 Konflikty v softvérových modeloch

Pri paralelnej práci viacerých vývojárov na modeli softvéru často vznikajú konflikty, ktoré komplikujú synchronizáciu jednotlivých verzií modelu. Konflikt v modeloch možno charakterizovať ako množinu odporujúcich sa zmien modelu, kde aspoň jedna zmena vykonaná vývojárom nie je v zhode s aspoň jednou zmenou vykonanou iným vývojárom (Altmanninger a Pierantonio, 2011). O konflikt ide aj v prípade rovnakého zámeru vývojárov, ktorí modifikujú rovnaký prvok modelu.

Táto kapitola obsahuje kategorizáciu konfliktov v modeloch a model konfliktov. Príklady uvedené pri jednotlivých typoch konfliktov predpokladajú existenciu základnej verzie modelu. Od tejto verzie sú odvodené 2 verzie modelu, v ktorých paralelne pracujú vývojári.

2.1 Kategorizácia konfliktov

Podľa štúdie (Altmanninger a Pierantonio, 2011) konflikty sú kategorizované z hľadiska úrovne abstrakcie a úrovne závažnosti. Diagram tejto kategorizácie je zobrazený na obr. 1. Ide o všeobecnú kategorizáciu aplikovateľnú aj na iné štruktúrované artefakty, napr. zdrojový kód programu.



Obrázok 1: *Diagram kategorizácie konfliktov (Altmanninger a Pierantonio, 2011).*

2.1.1 Textové konflikty

Textové konflikty sú podľa štúdie (Altmanninger a Pierantonio, 2011) detegované v prípade, že porovnávanie verzií modelov je zamerané na textovú reprezentáciu prvkov modelu, kde jednotkou porovnávania môže byť znak, slovo, veta, riadok textu alebo odsek. Textový konflikt môže nastať napríklad vtedy, ak je do triedy v 1 verzii modelu pridaný atribút s názvom *name* a do tej istej triedy v druhej verzii modelu je pridaný atribút s názvom *lastName*.

2.1.2 Syntaktické konflikty

Konflikty, ktoré môžu byť detegované štrukturálnym porovnaním verzií modelu, sa nazývajú syntaktické konflikty (Altmanninger, 2008). Ich reprezentáciou je strom alebo graf, ktorý priamo vyjadruje syntax použitého jazyka. Príkladom syntaktického konfliktu v diagrame tried je pridanie asociácie medzi 2 triedami v 1 verzii modelu a odstránenie 1 z týchto tried v druhej verzii modelu. O syntaktický konflikt ide aj v prípade modifikácie názvu rovnakého prvku v oboch verziách modelu.

Pre detekciu syntaktických konfliktov abstraktnej syntaxe sú potrebné informácie o syntaxi metamodelu (Altmanninger a Pierantonio, 2011). V prípade konkrétnej syntaxe sa vyžaduje modifikácia jednotlivých verzií modelov v rovnakom nástroji.

2.1.3 Sémantické konflikty

Sémantické konflikty vznikajú vtedy, ak vývojári modifikujú vo svojej verzii modelu prvky, ktoré sú od seba určitým spôsobom závislé alebo sémanticky navzájom súvisia (Altmanninger a Pierantonio, 2011). Na rozdiel od syntaktických konfliktov, pre ich detekciu nie je dostatočné štrukturálne porovnanie verzií modelu, keďže je nutné brať do úvahy aj sémantiku modelu. Sémantické konflikty sú rozdelené do jednotlivých kategórií na základe toho, či vývojári zmenili svoju verziu modelu sémanticky s opačným alebo rovnakým zámerom, a na základe toho, či je na ich detekciu potrebná znalosť syntaxe a sémantiky modelovacieho jazyka alebo znalosť domény.

Uvedená kategorizácia sémantických konfliktov vychádza z 3 hlavných sémantických aspektov, a to statická sémantika, behaviorálna sémantika a ekvivalentné koncepty (Altmanninger et al., 2007). Statické sémantické konflikty

vznikajú hlavne porušením statických vlastností modelu, čoho dôsledkom je neplatný zlúčený model alebo sémanticky navzájom odporujúce zmeny. Ich detekcia nie je možná tradičnými technikami na detekciu konfliktov a porušenia integrity alebo obmedzení modelu v zlúčenej verzii modelu. Príkladom týchto konfliktov je pridanie generalizácie medzi triedami v diagrame tried v oboch verziách modelu tak, že v zlúčenom modeli vznikne cyklus dedenia, alebo presúvanie atribútov pri refaktoringu (Altmanninger a Pierantonio, 2011).

Pre detekciu behaviorálnych sémantických konfliktov je nutné uvažovať paralelné zmeny v správaní prvkov modelu. Dátový tok alebo tok riadenia paralelne pridaných, modifikovaných alebo odstránených prvkov môže ovplyvniť správanie v zlúčenom modeli.

Ďalší druh konfliktov môže vzniknúť v súvislosti s pridaním sémanticky rovnakých konceptov alebo s aplikovaním refaktoringu. To je možné hlavne preto, že mnohé modelovacie jazyky umožňujú vyjadriť rovnaký význam rôznymi spôsobmi, napr. zobrazenie paralelného toku v UML stavovom diagrame.

Ontologické konflikty sú konflikty vznikajúce použitím lingvisticky rôznych termínov, ktoré sú sémanticky v určitom vzťahu alebo označujú tú istú vec (Altmanninger a Pierantonio, 2011). O ontologický konflikt ide vtedy, ak je v 1 verzii modelu tried pridaná trieda s názvom *Comment* a v druhej verzii trieda s názvom *Note*. Pre určitú ontológiu môžu byť tieto názvy synonymami.

2.1.4 Konflikty podľa úrovne závažnosti

Pre zníženie počtu konfliktov, ktoré je nutné riešiť manuálne počas zlúčenia modelov, je vhodná ich kategorizácia podľa úrovne závažnosti. V tomto prípade sa konflikty rozdeľujú na ťažké a ľahké konflikty (Koegel et al., 2010).

Ťažké konflikty sa vyznačujú tým, že zmeny, ktoré ich spôsobili, nie je možné aplikovať v zlúčenom modeli, čo znamená potrebu ich riešenia vývojárom (Altmanninger a Pierantonio, 2011). Napríklad modifikácia prvku modelu nie je možná po odstránení tohto prvku iným vývojárom. Ťažké konflikty sa týkajú hlavne textových a syntaktických konfliktov. Ich podmnožinou sú normálne konflikty, pre ktoré platí, že aplikácia konfliktných zmien je možná, ale 1 zmena prekryje druhú zmenu. Príkladom je premenovanie prvku modelu oboma vývojármi rôznym spôsobom.

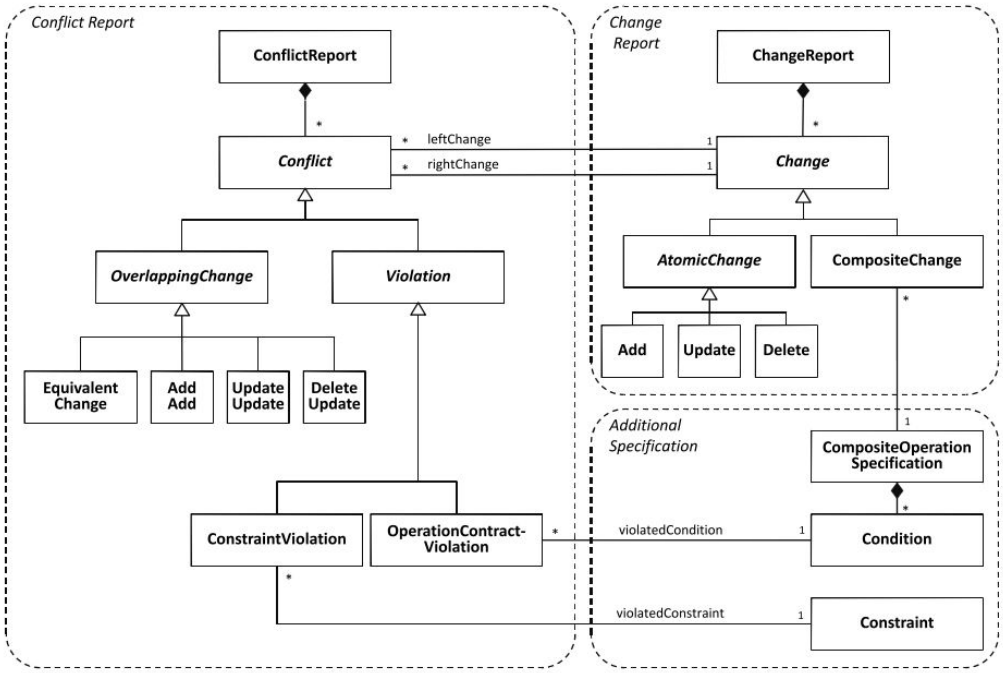
Ľahké konflikty sú tie konflikty, ktorých riešenie môže byť vykonané automaticky aplikovaním konfliktných zmien v zlúčenom modeli. Sú často zobra-

zené iba ako upozornenia, avšak môžu viesť k validačným chybám. Detekcia týchto konfliktov je často realizovaná štrukturálnym porovnaním alebo uvažovaním sémantiky modelu.

2.2 Model konfliktov

Na základe modelu konfliktov, ktorý je zobrazený na obr. 2, je možné rozdeliť konflikty do 2 skupín (Brosch et al., 2012):

1. konflikty spôsobené navzájom prekrývajúcimi zmenami, medzi ktoré patria navzájom si odporujúce a ekvivalentné zmeny modelu,
2. konflikty spôsobené porušením obmedzení metamodelu, modelu alebo kontraktu operácie.



Obrázok 2: *Model konfliktov (Brosch et al., 2012).*

2.2.1 Konflikty spôsobené navzájom prekrývajúcimi zmenami

Do skupiny navzájom prekrývajúcich zmien patria navzájom si odporujúce zmeny modelu a ekvivalentné zmeny modelu. Konflikty spôsobené navzájom si odporujúcimi zmenami patria medzi najčastejšie vyskytujúce sa konflikty. Môžeme hovoriť o nasledujúcich konfliktoch (Brosch et al., 2012):

(a) konflikt *Add/Add* - obaja vývojári pridajú prvok do svojej verzie modelu opačným spôsobom, napr. v oboch verziách modelu je pridaná generalizácia medzi triedami, ale smer generalizácie je opačný,
(b) konflikt *Update/Update* - obaja vývojári modifikujú rovnaký prvok modelu rôznym spôsobom, napr. konflikt je v názve prvku modelu,
(c) konflikt *Delete/Update* - prvý vývojár modifikuje prvok modelu, ktorý druhý vývojár vo svojej verzii modelu odstráni.

Na detekciu navzájom si odporujúcich zmien sa používajú hlavne spájacie matice (Barrett et al., 2011) a konfliktné množiny (Reiter et al., 2007). Detekcia týchto zmien modelu výrazne závisí od úrovne granularity metódy na detekciu konfliktov, ktorá môže považovať celý prvok modelu alebo len jeho vlastnosť za atomickú jednotku pri porovnávaní verzií modelov. Väčšina systémov určených na verziovanie modelov podporuje detekciu konfliktov založenú na jednotlivých vlastnostiach prvkov modelu.

Typickým príkladom konfliktov, ktoré vznikajú ekvivalentnými zmenami modelu, je pridanie rovnakého prvku s rovnakým názvom do oboch verzií modelu. Detekcia týchto konfliktov závisí od stratégie porovnávania verzií modelov – od jednotky porovnávania a heuristiky porovnávacieho algoritmu. Môžu nastať 3 situácie (Brosch et al., 2012):

(a) pridané prvky sú považované za rôzne hlavne v prípade, že porovnávanie verzií modelov je založené na univerzálne unikátnych identifikátoroch prvkov, výsledkom tejto situácie je duplikovaný výskyt pridaného prvku v zlúčenom modeli,
(b) pridané prvky sú považované za rovnaké a v zlúčenom modeli sa nachádza 1 prvok so všetkými vlastnosťami oboch pridaných prvkov,
(c) nastáva konflikt v pridaní rovnakých prvkov do modelu.

2.2.2 Konflikty spôsobené porušením obmedzení

Model, ktorý vznikne zlúčením verzií modelu, môže obsahovať porušenia obmedzení metamodelu, modelu a kontraktu operácie (Brosch et al., 2012). Príkladmi porušení obmedzení metamodelu sú cyklus dedenia medzi 2 triedami v diagrame tried a odpoveď na asynchrónnu správu v sekvenčnom diagrame v zlúčenom modeli.

Obmedzenia modelu je možné rozdeliť na vnútrodiagramové a medzidiagramové obmedzenia. O porušenie vnútrodiagramového obmedzenia ide napríklad vtedy, ak do oboch verzií modelu je pridaná asociácia, ktorá spája 2 triedy, s opačnou kardinalitou. Príkladom medzidiagramového obmedzenia je obmedzenie typu XOR, ktorý definuje, že objekt určitého typu môže byť spojený len s 1 z 2 objektov. Interakcia medzi týmito objektmi v zlúčenom modeli môže spôsobiť porušenie tohto obmedzenia napríklad v sekvenčnom diagrame.

Konflikt spôsobený porušením kontraktu operácie vždy zahŕňa aspoň 1 kompozitnú zmenu, ktorá nemôže byť aplikovaná, pretože iná zmena spôsobuje porušenie vstupnej podmienky, výstupnej podmienky alebo invariantu špecifikovanom kontraktom operácie (Brosch et al., 2012). Kompozitná zmena je množinou atomických zmien, ktoré sú aplikované pre určitý cieľ, napríklad refaktoring. Tento typ konfliktov je možné detegovať, ak metóda na detekciu konfliktov sleduje kompozitné zmeny. Kompozitné zmeny sú často špecifikované ako transformácie grafu alebo prostredníctvom vzorov zmien.

3 Metódy a nástroje na detekciu konfliktov

Táto kapitola približuje metódy a nástroje, ktoré sa zameriavajú na detekciu konfliktov. Opis každej metódy obsahuje typ konfliktov, ktoré metóda deteguje, a spôsob ich detekcie, čo je zhrnuté v zhodnotení kapitoly.

3.1 Rainbow

Metóda nazývaná Rainbow bola vyvinutá pre detekciu sémantických konfliktov pre UML diagramy prípadov použitia (Costa et al., 2013) a diagramy tried (Costa et al., 2014). Základom metódy je porovnanie diagramov z hľadiska sémantiky s cieľom zníženia počtu *false positive* a *false negative* konfliktov.

Metóda rieši prípady, kde 2 vývojári nezávisle od seba vykonávajú zmeny vo svojich verziách diagramu tried alebo diagramu prípadov použitia. Spracúva a porovnáva 3 verzie diagramu:

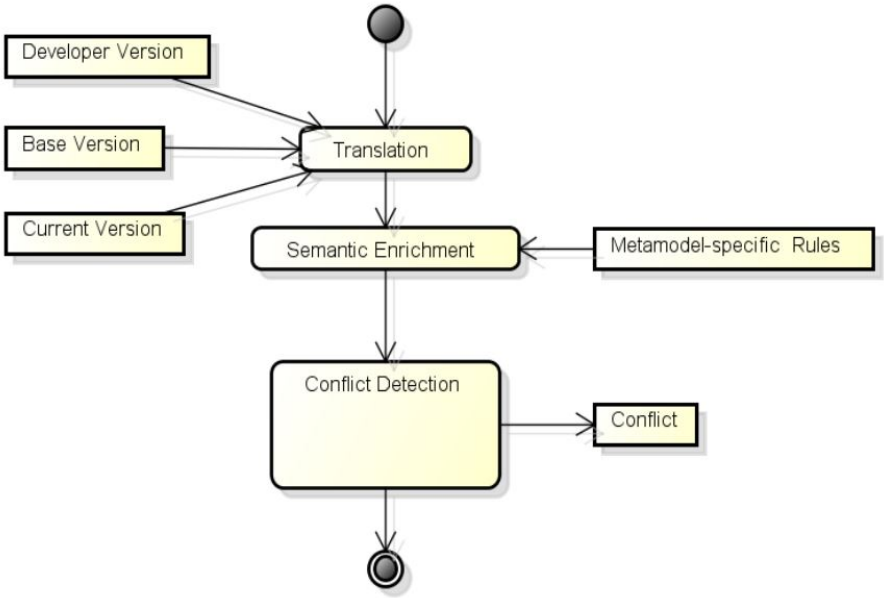
(a) verzia *developer* predstavuje verziu, v ktorej pracoval prvý vývojár a chce ju vložiť do repozitára,
(b) verzia *current* predstavuje verziu, v ktorej pracoval druhý vývojár, táto verzia je aktuálne uložená v repozitári,
(c) verzia *base* predstavuje verziu, od ktorej sú verzie *developer* a *current* odvodené.

Metóda Rainbow pozostáva z 3 fáz, ktoré sú znázornené na obr. 3:

(a) fáza prekladu,
(b) fáza sémantického obohatenia,
(c) fáza detekcie konfliktov.

3.1.1 Fáza prekladu

Cieľom fázy prekladu je transformácia daných 3 verzií diagramu do množiny faktov, ktoré sú zapísané v jazyku Prolog (Costa et al., 2013). Každý fakt slúži na reprezentáciu prvku alebo vzťahu v príslušnej verzii diagramu. V prípade vzťahu fakt obsahuje prvky, ktoré sú spojené týmto vzťahom. Príkladom Pro-



Obrázok 3: *Fázy metódy Rainbow (Costa et al., 2013).*

log faktov je *class(className)* alebo *inheritance(employee, person)*. Výsledkom tejto fázy sú 3 množiny Prolog faktov, ktoré predstavujú iba syntaktické informácie o všetkých verziách diagramu.

3.1.2 Fáza sémantického obohatenia

Fáza sémantického obohatenia je zameraná na odvodenie nepriamych vzťahov v jednotlivých verziách diagramu na základe analýzy Prolog faktov vytvorených vo fáze prekladu (Costa et al., 2014). Táto fáza obohacuje získané syntaktické informácie o sémantické informácie o všetkých verziách diagramu. Pre realizáciu tejto fázy sa používa množina definovaných sémantických pravidiel, ktoré opisujú sémantiku diagramu tried, resp. diagramu prípadov použitia na základe UML metamodelu. Príkladom sémantického pravidla je vzťah 3.1 (Costa et al., 2013). Odvodené nepriame vzťahy sú taktiež reprezentované prostredníctvom Prolog faktov a sú pridané do príslušnej množiny Prolog faktov získanej vo fáze prekladu.

$$\forall a, b \in Actor, \forall c \in UseCase,$$
$$inheritance(a, b) \wedge association(b, c) \implies association(a, c) \quad (3.1)$$

3.1.3 Fáza detekcie konfliktov

Vo fáze detekcie konfliktov sú najprv identifikované rozdiely medzi množinou Prolog faktov verzie *current* a verzie *base*, čoho výsledkom sú 2 množiny (Costa et al., 2013). Prvá množina predstavuje množinu pridaných faktov vo verzii *current*, druhá množina obsahuje všetky odstránené fakty vo verzii *current*. Opísaná aktivita je potom realizovaná aj medzi množinou Prolog faktov verzie *developer* a verzie *base*.

Na základe určených množín pridaní a odstránení pre verzie *current* a *developer* je možné detegovať konflikty v diagramoch. Konflikt je detegovaný, ak sa prvok diagramu nachádza v množine pridaní jednej verzie a súčasne v množine odstránení druhej verzie alebo naopak, čo je vyjadrené vzťahom 3.2. Pri detekcii konfliktov sa uvažujú nielen vzťahy, ale aj prvky, ktoré sú danými vzťahmi spojené. Napríklad konflikt je detegovaný, ak v 1 verzii diagramu prípadov použitia je aktér odstránený, ale v druhej verzii je pridaný vzťah medzi týmto aktérom a prípadom použitia.

$$Conflict_{v1,v2} = (Add_{v1} \cap Del_{v2}) \cup (Add_{v2} \cap Del_{v1}) \tag{3.2}$$

3.2 Detekcia konfliktov založená na modifikácii grafov

Táto metóda na detekciu konfliktov reprezentuje štruktúru modelu pomocou grafov a grafových morfizmov (Ehrig et al., 2006). Metóda vychádza z prístupu DPO (angl. Double pushout), ktorý využíva definície PO (angl. pushout) a PB (angl. pullback) v teórii grafových morfizmov, je aplikovaný na transformácie grafu a používa sa na formalizáciu detekcie konfliktov.

Základom tejto metódy sú modifikácie grafu, ktoré zovšeobecňujú koncept transformácií grafu na zmeny grafu (Täntzer et al., 2010). Modifikácia grafu G na graf H predstavuje sekvenciu priamych modifikácií grafu G, ktorých výsledkom je graf H. Priama modifikácia grafu $G \xleftarrow{g} D \xrightarrow{h} H$ pozostáva z grafov G, D, H a grafových morfizmov g a h, kde graf D je prechodný graf, v ktorom sú vykonané iba akcie odstránenia v grafe G.

Detekcia konfliktov je založená na identifikácii operácií, ktoré viedli k modifikácii grafu, a následne na procedúre zlúčenia 2 rôznych stavov, teda 2 rôznych modifikácií grafu.

3.2.1 Detekcia konfliktov založená na operáciách

Pre detekciu konfliktov je dôležité nájsť operáciu, ktorej aplikovaním vznikla konkrétna modifikácia grafu (Täntzer et al., 2010). Na tento účel sa používa extrakcia minimálneho pravidla, teda minimálnej operácie, ktorá obsahuje všetky atomické akcie vykonané danou modifikáciou grafu. Táto extrakcia vedie k minimálnej transformácii grafu nutnej na danú modifikáciu grafu. Konštrukcia minimálneho pravidla vychádza zo štúdie (Bisztray et al., 2008) a je unikátna, teda neexistuje menšie pravidlo.

Ak sa operácia, ktorá viedla k modifikácii grafu, nenájde touto extrakciou, metóda aplikuje extrakciu minimálneho pravidla ešte raz a nájde prislúchajúcu operáciu jej porovnaním s minimálnym pravidlom. Hľadaná operácia patrí danej modifikácii grafu vtedy, ak extrahované minimálne pravidlo je podpravidlom operácie a operácia je aplikovateľná v pôvodnom grafe G.

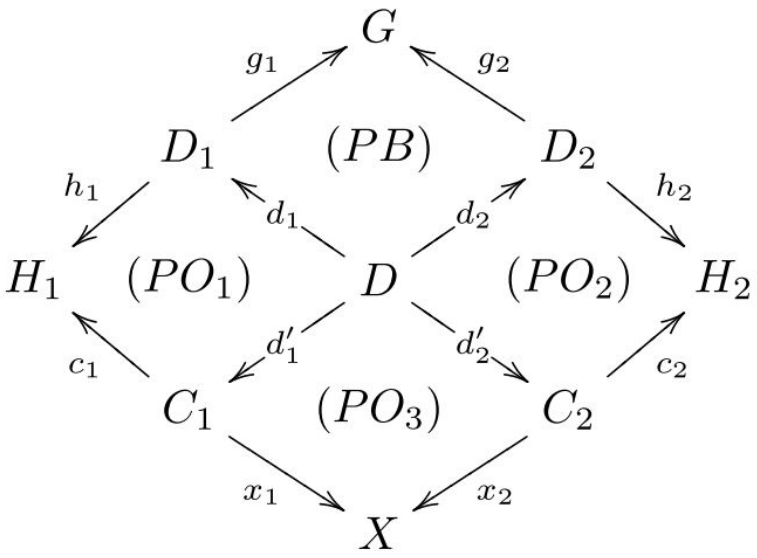
Pre detekciu konfliktov sa využívajú kritické páry (Ehrig et al., 2006). Kritický pár je možné nájsť pre každú dvojicu paralelne závislých transformácií grafu a predstavuje hľadaný konflikt. Táto metóda detekcie konfliktov umožňuje detegovať konflikty typu *Update/Update* a *Delete/Update*.

3.2.2 Detekcia konfliktov založená na stavoch

Po detekcii konfliktov založenej na operáciách nasleduje detekcia konfliktov založená na stavoch, ktorá využíva procedúru na zlúčenie 2 rôznych stavov, teda 2 rôznych modifikácií grafu (Täntzer et al., 2010). Na obr. 4 je príklad aplikácie tejto procedúry. Predpokladom je existencia modifikácie grafu G na graf H_1 a modifikácie grafu G na graf H_2, kde D_1 a D_2 sú prechodné grafy.

Táto procedúra vypočíta „najmenší spoločný menovateľ“ D oboch modifikácií grafu, ktorý je rozšírený o všetky zmeny oboch modifikácií grafu, teda o grafové morfizmy d'_1 a d'_2. Tak vzniknú grafy C_1 a C_2, ktoré sú potom pomocou grafových morfizmov x_1 a x_2 zlúčené do výsledného grafu X. Tento graf je výsledkom procedúry na zlúčenie 2 rôznych modifikácií grafu.

Výsledný zlúčený graf X je určený pre detekciu konfliktov. Metóda špecifikuje grafové obmedzenia na základe podmienok grafu definovaných voči celému modelu alebo metamodelu podľa (Habel a Pennemann, 2009). O konflikt ide v takom prípade, ak oba grafy H_1 a H_2 vyhovujú danému grafovému obmedzeniu, ale graf X mu nevyhovuje.



Obrázok 4: *Zlúčenie 2 modifikácií grafu H_1 a H_2 do grafu X (Täntzer et al., 2010).*

3.3 SMoVer

Semantically enhanced Model Version Control System (SMoVer) je verziovací systém, ktorý umožňuje detegovať statické a behaviorálne sémantické konflikty (Altmanninger et al., 2007). Detekcia konfliktov je založená na definovaní stratégií aktualizácie a vytvorení definícií sémantických pohľadov záujmu. SMoVer nie je závislý od vykonaných operácií vývojárov v paralelne modifikovaných verziách modelu (V' a V") a môže byť aplikovaný na ľubovoľný modelovací jazyk založený na EMF (Eclipse Modeling Framework).

Pred samotnou detekciou konfliktov sú jednotlivé verzie modelu porovnané zo štrukturálneho hľadiska. Sledovaním hlavne atribútov a referencií prvku modelu je možné určiť, či bol prvok modelu ako celok aktualizovaný. Pre tento účel existujú 4 rôzne stratégie aktualizácie, ktoré sa používajú na detekciu štrukturálnych zmien v grafe:

- aktualizácia atribútov (ATT) – hodnota atribútu bola zmenená,
- aktualizácia referencovaného prvku (REF),
- aktualizácia referencie (REFS) – prvky modelu boli vytvorené alebo odstránené vývojármi, môžu nastať nasledujúce 4 kombinácie:
  Create-Create, *Create-Delete*, *Delete-Create*, *Delete-Delete*,

- aktualizácia roly (ROL) – prvok modelu je referencovaný alebo derefencovaný na iný prvok modelu, taktiež môžu nastať 4 kombinácie ako v predchádzajúcej stratégii.

Systém SMoVer definuje konfliktné množiny na základe OCL výrazov, ktoré sú zapísané v ukážke 1 (Reiter et al., 2007). Množina *Con* obsahuje všetky konfliktné prvky modelu a je zjednotením 3 množín *CrCon*, *UpdCon* a *DelCon*, ktoré reprezentujú konflikty typu *Add/Add*, *Update/Update* a *Delete/Update*. Funkcia *isUpdated()* vracia hodnotu, či prvok modelu bol aktualizovaný, a funkcia *areNotEqual()* kontroluje zhodu 2 prvkov modelu.

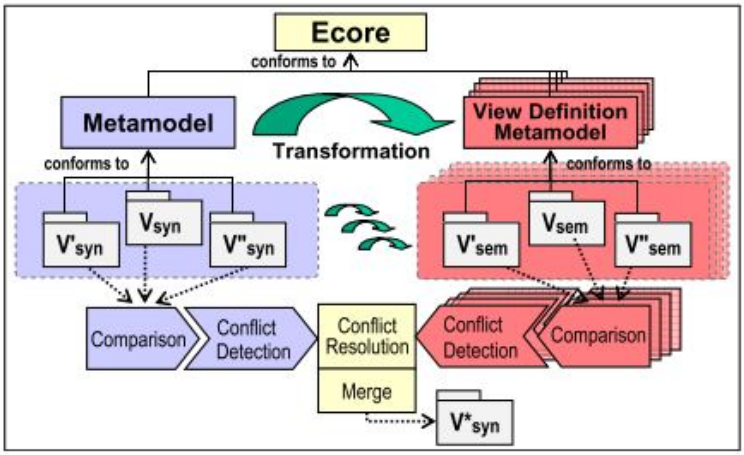
```
Con=UpdCon->union(CrCon->union(DelCon))

UpdCon=Updates'->intersection(Updates'')->select(e|e.areNotEqual(V', V''))
CrCon=Creates'->intersection(Creates'')->select(e|e.areNotEqual(V', V''))
DelCon=Deletes'->intersection(Deletes'')->select(e|e.areNotEqual(V', V''))

Updates'=V->select(e|e.isUpdated(V, V'))
Updates''=V->select(e|e.isUpdated(V, V''))

Creates'=(V' - V)
Creates''=(V'' - V)

Deletes'=(V - V')
Deletes''=(V - V'')
```

Ukážka 1: *OCL výrazy pre konfliktné množiny (Reiter et al., 2007).*

Definícia sémantických pohľadov, ktorá je znázornená na obr. 5, je zložená z (Altmanninger et al., 2007):

- metamodelu definícií pohľadov, ktorý definuje abstraktnú syntax a je reprezentovaný pomocou podmnožiny zdrojového metamodelu, metamodelu definícií doménovo špecifických pohľadov alebo metamodelu iného modelovacieho jazyka,
- transformácie modelu, ktorá definuje sémantické mapovanie medzi zdrojovým a cieľovým metamodelom (metamodelom definícií sémantických pohľadov). Výstupom tejto transformácie je model, ktorý odpovedá metamodelu reprezentujúcemu definíciu sémantických pohľadov záujmu.

Vďaka definícii sémantických pohľadov SMoVer realizuje proces detekcie konfliktov vo verziách modelu v syntaktických aj sémantických pohľadoch použitím štrukturálneho porovnávania (Altmanninger et al., 2007). O syntaktické konflikty ide vtedy, ak sú konflikty detegované len porovnávaním verzií modelu,



Obrázok 5: *Definícia sémantických pohľadov (Altmanninger et al., 2007).*

na druhej strane konflikty detegované medzi verziami modelu, ktoré boli transformované v sémantickom pohľade, sú sémantické. Pomocou definícií sémantických pohľadov záujmu systém SMoVer deteguje konflikty oveľa presnejšie, pretože algoritmus deteguje predtým nedetegované sémantické konflikty a znižuje počet nesprávne identifikovaných syntaktických konfliktov.

3.4 Mirador

Mirador je nástroj určený na spájanie softvérových modelov, ktorý umožňuje kontrolu a manipuláciu v procese spájania modelov (Barrett et al., 2011). Nástroj je založený na hybridnom prístupe k problému spájania modelov, keďže používa techniky založené na stavoch aj techniky založené na operáciách. Ďalej poskytuje nasledujúce možnosti:

- rozlíšenie operácií, ktoré sú v priamom alebo nepriamom konflikte,
- použitie techník pre vizualizáciu závislostí zmien a konfliktov s cieľom zachovať poradie vykonávania operácií,
- detekcia a automatické riešenie konfliktov použitím rozhodovacích tabuliek spolu s riešením sémantiky.

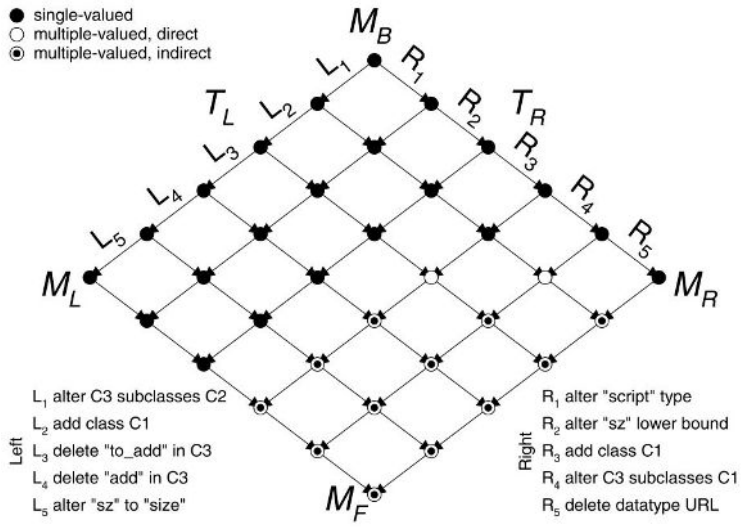
V procese spájania modelov sú najprv vstupné verzie modelu skonvertované do normálnej formy založenej na stavoch nazývanej Ecore. Následne sa vypočítajú rozdiely medzi základným modelom (verziou *base*) a jednotlivými verziami modelu, čím sa identifikujú zmenené prvky a spôsob, akým boli zmenené.

Ďalej nasleduje porovnanie verzií modelu založené na stavoch, ktorého cieľom je nájsť zhodu medzi prvkami vo verziách modelu. Namiesto použitia identifikátorov prvkov Mirador poskytuje 7 stratégií pre identifikáciu zhodných prvkov modelu, pričom 2 z nich môže definovať používateľ.

3.4.1 Rozlíšenie operácií v priamom alebo nepriamom konflikte

Na rozlíšenie operácií, ktoré sú v priamom alebo nepriamom konflikte, sa používa rovina zmien na obr. 6 (Barrett et al., 2011). Uzly reprezentujú modely a hrany reprezentujú jednoduché operácie zmien, tzv. subtransformácie. Zmeny na osiach sú transformácie základného modelu M_B na jeho ľavú a pravú verziu. Každý uzol predstavuje množinu modelov, ktoré sú výsledkom aplikovania všetkých zmien v modeli M_B na všetkých cestách k danému uzlu.

Podstatou je zlúčenie viacerých subtransformácií roviny zmien do jedinej transformácie obsahujúcej čo najviac zmien z oboch strán. Výsledkom aplikovania tejto transformácie v modeli M_B má byť konzistentný model. To znamená, že transformácia obsahuje všetky transformácie na osiach roviny len vtedy, ak neexistujú žiadne konflikty.

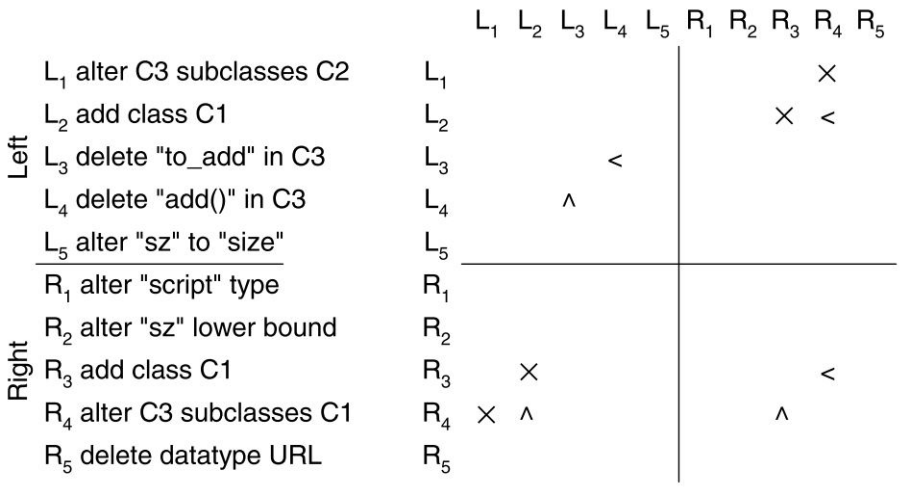


Obrázok 6: *Rovina zmien obsahujúca subtransformácie ľavej a pravej verzie modelu (Barrett et al., 2011). Uzly majú jednoduchú hodnotu a sú bez konfliktu, ak predstavujú 1 model, ktorý je výsledkom všetkých ciest smerujúcich k nemu. Viachodnotové uzly sú v priamom alebo nepriamom konflikte.*

3.4.2 Vizualizácia závislostí zmien a konfliktov

Vzťahy medzi zmenami a poradie ich vykonania v zlúčenom modeli sú vyjadrené prostredníctvom matice konfliktov na obr. 7 po aplikovaní predikátu *before(a, b)* (Barrett et al., 2011). Tento predikát určuje, či operácia a sa má vykonať pred operáciou b. Konflikt nastáva vtedy, ak platí vzťah 3.3. Matica konfliktov je rozdelená do 4 kvadrantov, ktoré porovnávajú zmeny v rámci 1 modelu a zmeny medzi modelmi.

$$before(a, b) = before(b, a) = true \tag{3.3}$$



Obrázok 7: *Matica konfliktov po aplikovaní predikátu before pre všetky páry zmien (Barrett et al., 2011). Znak „<“ vyjadruje poradie vykonania zmien. Znak „x“ označuje zmeny v priamom konflikte. Príkladom je konflikt medzi zmenami L_2 a R_3, ktorých aplikovaním v zlúčenom modeli vzniknú duplikátne triedy.*

3.4.3 Detekcia konfliktov

Detekcia a riešenie konfliktov sú založené na rozhodovacích tabuľkách, ktoré môže špecifikovať používateľ (Barrett et al., 2011). Hlavnými komponentami rozhodovacej tabuľky sú:

- názov tabuľky spolu s definovanými pravidlami,
- podmienky – funkcie, ktoré majú byť vyhodnotené a
- akcie – procedúry, ktoré majú byť vykonané.

Príklad rozhodovacej tabuľky je na obr. 8. Pre každé pravidlo sú vybraté podmienky na testovanie nastavením hodnoty „Y“ alebo „N“ pri názve podmienky. Znak „-“ vyjadruje ignorovanie podmienky pre dané pravidlo. Pravidlo je splnené vtedy, ak všetky testované podmienky vrátia očakávanú hodnotu. Pre každé splnené pravidlo sú vykonané procedúry v sekcii akcií v číselnom poradí. Akcie označené znakom „0“ sa nevykonávajú.

```
[Table]
before_same   // Evaluates before(op1, op2) for same side change operations.
              //        +-------------- Element is added to a container that itself is added.
              //        |  +----------- Added class element is target of an added reference.
              //        |  |  +-------- Element is deleted from a container that is deleted.
              //        |  |  |  +----- Altered class subtypes added class.
              //        |  |  |  |  +-- Catch-all rule, invokes linked decision table.
              //        |  |  |  |  |
[Conditions]  // Rule#  1  2  3  4  5
on_same_side           -  -  -  -  N
op1_is_add             Y  Y  -  Y  -
op1_is_delete          -  -  Y  -  -
op1_is_alter           -  -  -  -  -
op1_is_class           -  Y  -  Y  -
op2_is_add             Y  Y  -  -  -
op2_is_delete          -  -  Y  -  -
op2_is_alter           -  -  -  Y  -
op2_is_reference       -  Y  -  -  -
op2_is_class           -  -  -  Y  -
op1_contains_op2       Y  -  -  -  -
op2_contains_op1       -  -  Y  -  -
op2_is_ref_to_op1      -  Y  -  -  -
op2_subclasses_op1     -  -  -  Y  -
aux_match              -  -  -  -  Y  // Invokes auxiliary table, before cross.

[Actions]
do_true                1  1  1  1  1  // Put op1 before op2.
```

Obrázok 8: *Rozhodovacia tabuľka pre vyhodnotenie predikátu before v 1 verzii modelu (Barrett et al., 2011).*

Pre detekciu konfliktov je dôležitý predikát *before* určujúci poradie vykonania 2 operácií, čo umožňuje predísť nekonzistencii vo verzii modelu a identifikovať konflikty medzi verziami modelu (Barrett et al., 2011). Napríklad pravidlo č. 2 na obr. 8 predpokladá pridanie triedy operáciou č. 1 a pridanie referencie na túto triedu operáciou č. 2. Akcia, ktorá sa má vykonať po splnení tohto pravidla, je akcia vykonania operácie č. 1 a potom operácie č. 2.

Najprv je určený predikát *before* pre zmeny v rovnakej verzii modelu len kvôli ich zoradeniu, potom pre zmeny v inej verzii modelu s cieľom ich zoradenia a identifikácie konfliktov. Takto vzniknú 2 rozhodovacie tabuľky a páry zmien, ktoré viedli k vzniku konfliktu. Syntaktické konflikty, ktoré nevznikli navzájom si odporujúcimi zmenami, sú odfiltrované použitím pravidiel rozhodovacej tabuľky.

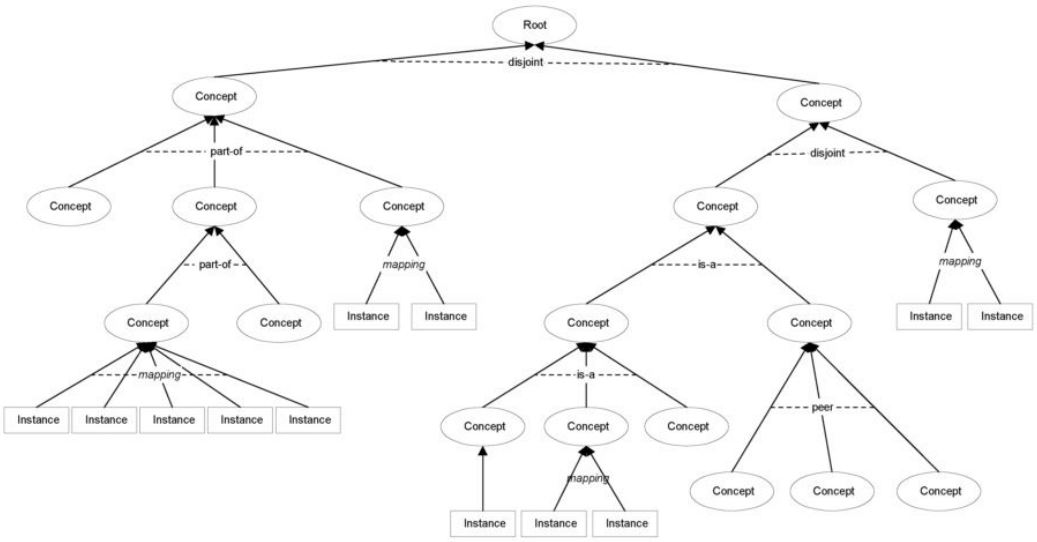
3.5 SCROL

Semantic Conflict Resolution Ontology (SCROL) je metóda zameraná na automatickú detekciu a riešenie rôznych sémantických konfliktov v heterogénnych databázach (Ram a Park, 2004). Zahŕňa spôsob automatického porovnania a manipulácie so znalosťami informačných zdrojov rôznych domén.

Štruktúra SCROL-u je strom, ktorý je rozšírený o horizontálne vzťahy medzi jeho uzlami (obr. 9). Ide o päticu nasledujúcich komponentov:

- množina konceptov - koncept je zovšeobecnený abstraktný termín, ktorý môže mať viacero konkrétnych inštancií. Príkladom konceptu je *Teplota* a jeho inštancie sú *Fahrenheit* alebo *Celsius*. Potomkom konceptu môže byť 0 alebo viacero konceptov alebo inštancií a rodičom konceptu môže byť práve 1 koncept. Každý koncept obsahuje vlastnosti názov, definíciu, množinu podkonceptov, rodičovský koncept, množinu inštancií a komponenty mapované na koncept.
- množina inštancií - každá inštancia má práve 1 rodičovský koncept a neobsahuje žiadnych potomkov. Každá inštancia obsahuje vlastnosti názov, definíciu, rodičovský koncept a komponenty mapované na inštanciu.
- množina príbuzenských vzťahov medzi 2 konceptami - používa sa vzťah *disjoint* pre spojenie sémanticky neekvivalentných konceptov, vzťah *peer* pre spojenie sémanticky ekvivalentných konceptov, vzťah *part-of* podobný agregácii a vzťah *is-a* podobný generalizácii.
- množina mapovacích vzťahov medzi 2 inštanciami - mapovacie vzťahy vyjadrujú, že všetky inštancie patriace konceptu sú považované za synonymá ich rodičovského konceptu, ak majú rozdielne názvy pre rovnaký koncept. Inštancie sú homonymá vtedy, ak majú rovnaké názvy a patria k rôznym konceptom. Mapovacie vzťahy určujú, či každá hodnota v 1 inštancii má 1 alebo viacero odpovedajúcich hodnôt v iných inštanciách.
- koreň stromu, ktorý nemá rodiča.

Pre detekciu konfliktov sa definujú rôzne koncepty, inštancie a ich vzťahy pomocou opisu a klasifikácie rôznych typov sémantických konfliktov v SCROL-e (Ram a Park, 2004). Ide o dátové konflikty, ktoré sú spôsobené rôznou reprezentáciou a interpretáciou podobných dát, a schematické konflikty, ktoré vznikajú rozdielmi v logickej štruktúre a nekonzistenciou schém v rovnakej doméne. Takto vznikne ontológia, ktorá môže byť použitá v rôznych doménach.



Obrázok 9: *Štruktúra SCROL-u (Ram a Park, 2004). Koncepty sú reprezentované elipsou, inštancie obdĺžnikom, príbuzenské a mapovacie vzťahy prerušovanou vertikálnou čiarou. Plná orientovaná čiara označuje vzťah medzi rodičom a potomkom.*

Následne sa v SCROL-e nájdu 3 dôležité informácie (Ram a Park, 2004):

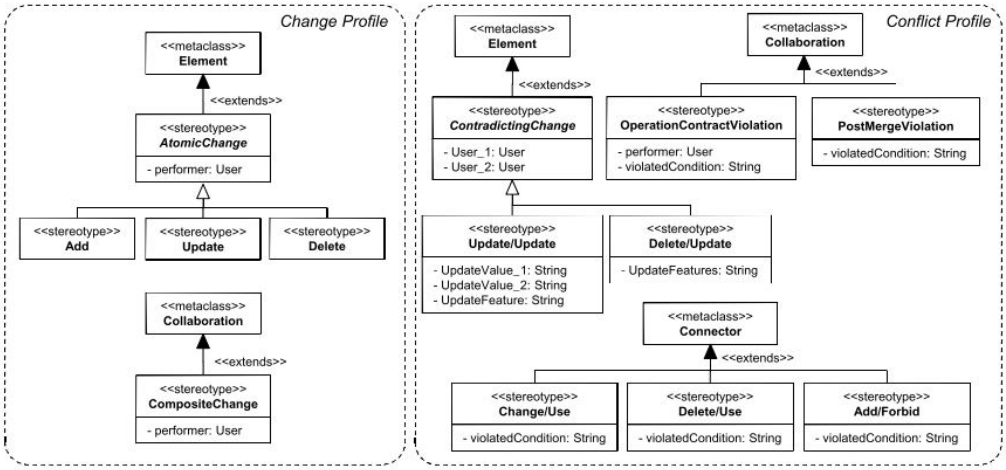
- správca konfliktov - ide o koncept, ktorého potomkovia (koncepty alebo inštancie) reprezentujú viacnásobné interpretácie sémanticky podobných objektov. Každý správca konfliktov má vlastný sémantický analyzátor.
- originálny kontext - koncept alebo inštancia, ktorá reprezentuje špecifický kontext komponentu v schéme.
- cieľový kontext - reprezentuje výsledný kontext, do ktorého sú transformované významy dát z lokálnych databáz podľa požiadaviek používateľa.

Ďalším krokom je explicitný opis znalostí ontologických vzťahov medzi komponentami SCROL-u a znalostí mapovania ontológií medzi SCROL-om a databázovými schémami. Znalosti mapovania ontológií sú opísané na úrovni atribútov, entít alebo vzťahov a definujú koncepty alebo inštancie v SCROL-e, ktoré sú mapované na komponenty schémy.

Úspešnosť detekcie dátových a schematických konfliktov pomocou metódy SCROL bola testovaná pre 3 rôzne aplikačné domény (územné, ekologické a publikačné heterogénne databázy). SCROL detegoval viacero rôznych sémantických konfliktov, ktoré boli spoločné pre všetky 3 aplikačné domény.

3.6 Profil zmien a konfliktov

Veľkým predpokladom pre úspešné vyriešenie konfliktov je reprezentácia a vizualizácia konfliktov a vykonaných zmien v jednotlivých verziách UML modelu. Na tento účel sú definované profil zmien a profil konfliktov, ktoré sú znázornené na obr. 10 (Brosch et al., 2011). Tieto profily využívajú stereotypy pre vyjadrenie typu zmeny a konfliktu v modeli.



Obrázok 10: *Profil zmien a profil konfliktov (Brosch et al., 2011).*

Pre každý typ zmeny modelu existuje vďaka profilu zmien stereotyp a uchováva sa jej autor. Stereotypy *«Add»*, *«Update»* a *«Delete»* použité pre atomické zmeny môžu byť využité pre všetky UML prvky, keďže dedia od stereotypu *«AtomicChange»*, ktorý rozširuje metatriedu *Element* UML metamodelu.

Pre reprezentáciu kompozitných zmien, ktoré zahŕňajú niekoľko prvkov modelu, je určený stereotyp *«CompositeChange»* (Brosch et al., 2011). Tento stereotyp rozširuje metatriedu *Collaboration* UML metamodelu na to, aby sa kolaborácia vzťahovala na prvky modelu zahrnuté v kompozitnej zmene použitím UML *Connectors*, pričom na tieto prvky modelu sú aplikované stereotypy atomických zmien.

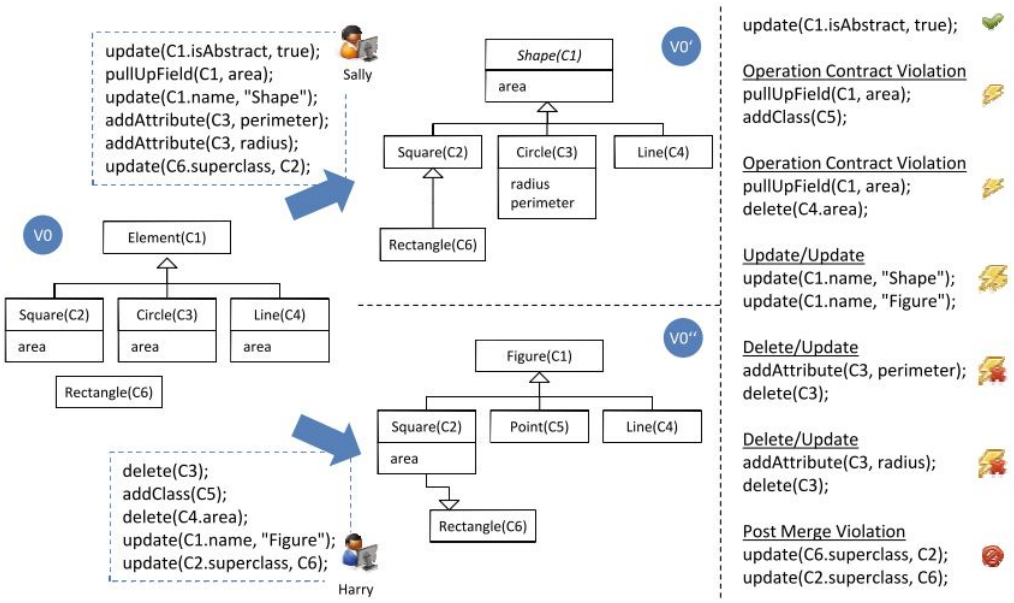
Profil konfliktov na obr. 10 poskytuje nasledujúce stereotypy:

- *«Update/Update»* a *«Delete/Update»* pre konflikty spôsobené navzájom si odporujúcimi zmenami, ktoré je možné aplikovať na všetky prvky,
- *«OperationContractViolation»* pre konflikty spôsobené porušením kontraktu operácie,

- *«PostMergeViolation»* pre konflikty spôsobené porušením obmedzenia, ktoré sú detegované v zlúčenom modeli po zlúčení verzií modelu,
- *«Change/Use»*, *«Delete/Use»* a *«Add/Forbid»* pre konflikty spôsobené porušením predpokladov a podmienok.

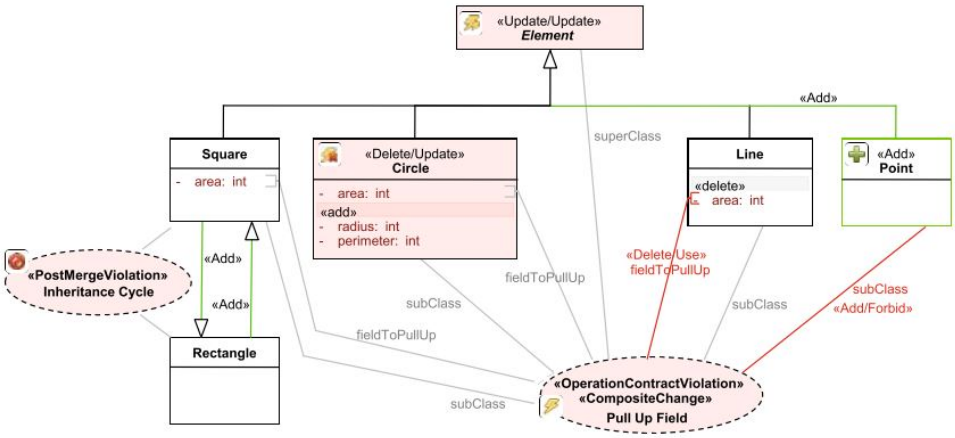
Pre reprezentáciu obmedzení, ktoré môžu zahŕňať niekoľko prvkov modelu, je podobne ako v profile zmien definovaný stereotyp a kolaborácia sa vzťahuje na prvky modelu zahrnuté v obmedzení použitím UML *Connectors*.

Pre ukážku použitia opísaných profilov pre vizualizáciu zmien a konfliktov v modeloch predstavujeme príklad, ktorý je znázornený na obr. 11.



Obrázok 11: *Príklad paralelnej práce vývojárov na modeli (Brosch et al., 2011). Vývojári pracovali vo vlastných verziách modelu V0‘ a V0“ odvodených od základnej verzie V0. Zmeny, ktoré vykonali v týchto verziách modelu, sú znázornené pri danej verzii modelu. Vpravo sú prehľadne opísané detegované konflikty.*

Ide o typickú paralelnú prácu vývojárov, kde každý vývojár si vyberie základnú verziu modelu z centrálneho repozitára a pracuje sám vo svojej verzii modelu, v ktorej vykonáva nejaké zmeny. Po vykonaní všetkých zmien vložia svoju verziu modelu do centrálneho repozitára a nasleduje proces ich synchronizácie. Po identifikácii zmien a detekcii konfliktov v modeloch sú zmeny a konflikty vizualizované použitím profilu zmien a profilu konfliktov, čím sa vytvorí konfliktný diagram znázornený na obr. 12.



Obrázok 12: *Konfliktný diagram, ktorý vizualizuje zmeny a detegované konflikty (Brosch et al., 2011). Pridané prvky modelu sú znázornené zelenou farbou a sú označené stereotypom «Add». Odstránené prvky modelu sú znázornené červenou farbou a sú označené stereotypom «Delete». Typy konfliktov sú vyjadrené stereotypmi a pre každý konflikt je jasné, ktorých prvkov modelu sa týka.*

Výhody vizualizácie zmien a konfliktov prostredníctvom konfliktného diagramu sú nasledujúce (Brosch et al., 2011):

- všetky potrebné informácie pre riešenie konfliktov sú uvedené v 1 diagrame v spoločnom pohľade,
- použitie filtrov umožňuje zobraziť len určité typy stereotypov alebo konkrétne konflikty,
- pre každý stereotyp môžu byť definované metódy na vyriešenie konfliktu a zvolené riešenie konfliktu môže byť uložené v histórii riešenia konfliktov,
- konflikty môžu byť dočasne tolerované, uložené v zlúčenej verzii modelu a neskôr vyriešené iným vývojárom.

Ako nevýhodu tohto prístupu vizualizácie vidíme možnú stratu prehľadnosti konfliktného diagramu. Hlavne v prípade rozsiahlych modelov, veľkého počtu vykonaných zmien vo verziách modelu a komplikovaných konfliktov môže konfliktný diagram obsahovať množstvo prvkov modelu, stereotypov pre zmeny a konflikty a čiar, čo zníži jeho prehľadnosť.

Ďalším problémom je farebné vyznačenie tried *Element* a *Circle*, ktorých sa týkajú konflikty, a jednotlivých typov konfliktov. V niektorých nástrojoch môžu byť aj nekonfliktné prvky modelu znázornené tou istou farbou, čo by výrazne znížilo prehľadnosť vizualizácie konfliktov.

3.7 Enterprise Architect

Enterprise Architect[1] je známy modelovací nástroj, ktorý podporuje tvorbu modelov založených na biznise, analýzu softvérových požiadaviek a všetky prvky, vzťahy a diagramy špecifikované štandardom UML 2.5. Jednou z jeho funkcií je detekcia a vizualizácia konfliktov pri zlúčení viacerých verzií modelu. V tomto procese používa nasledujúce pravidlá[2]:

- po synchronizácii verzií modelu sa všetky zmeny aplikujú na verziu *base* a verziu modelu, ktorá bola synchronizovaná ako posledná, to znamená, že tieto verzie sú rovnaké,
- pridávania sú kumulatívne, napríklad ak pridáme do oboch verzií modelu 2 nové triedy, zlúčený model obsahuje 4 triedy, pričom Enterprise Architect nesleduje názov a štruktúru nových prvkov, ale riadi sa iba identifikátormi prvkov,
- odstránenie má prednosť pred modifikáciou, teda ak je prvok modelu v 1 verzii modifikovaný a v 2. verzii je odstránený, zlúčený model neobsahuje tento prvok.

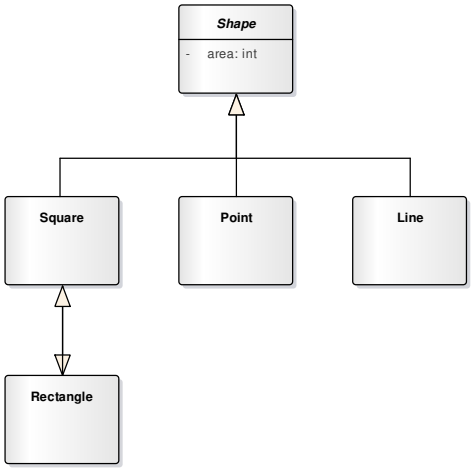
Enterprise Architect odporúča, aby vývojári pracovali paralelne na rôznych častiach modelu (v rôznych balíkoch), pretože ak viacerí vývojári modifikujú rovnaký prvok modelu, modifikácia vložená do zlúčenej verzie nie je zachovaná a je prepísaná ďalšou. Pri paralelnej práci vývojárov na rovnakej časti modelu odporúča ich prítomnosť pri synchronizácii verzií modelu.

Funkcionalitu detekcie a vizualizácie konfliktov nástroja Enterprise Architect sme otestovali vytvorením modelov, ktoré sú prezentované na obr. 11. Po zlúčení verzií modelu V0‘ a V0“ vznikol zlúčený model zobrazený na obr. 13, v ktorom neboli hlásené žiadne detegované konflikty.

Verzia modelu V0“, ktorá bola synchronizovaná ako posledná, bola rovnaká ako zlúčená verzia modelu a v tabuľke konfliktov obsahovala 1 detegovaný konflikt, čo je znázornené na obr. 14. Tento konflikt sa týka zmeny názvu a typu triedy *Element* v pôvodnom modeli. Môžeme vidieť, že tabuľka konfliktov obsahuje globálny univerzálny identifikátor tejto triedy, ale nikde nešpecifikuje, že ide o triedu. Pri synchronizácii väčších modelov by bol určite problém nájsť

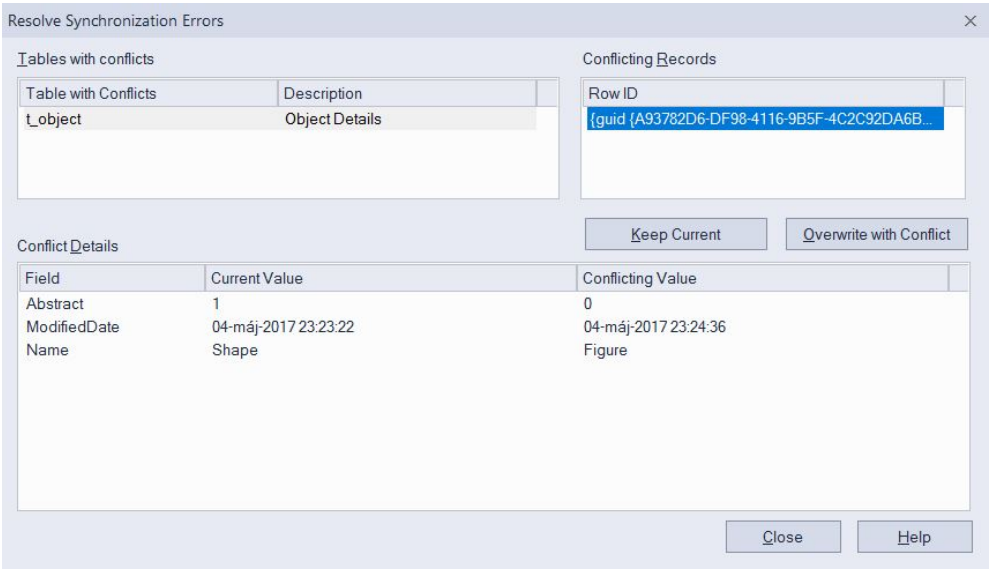
[1] http://www.sparxsystems.com.au/products/ea/index.html

[2] http://www.sparxsystems.com/enterprise_architect_user_guide/13.0/model_repository/replication.html



Obrázok 13: *Zlúčený diagram tried po synchronizácii verzií modelu V0‘ a V0“.*

prvok, ktorého sa konflikt týka. V detailoch konfliktu vidíme, že konflikt nastal v type a názve triedy *Element* v pôvodnom modeli.



Obrázok 14: *Tabuľka konfliktov v 2. verzii modelu V0“.*

3.8 Zhodnotenie metód a nástrojov

Vykonaná analýza metód a nástrojov zameraných na detekciu konfliktov ukázala, že súčasťou väčšiny metód a nástrojov je detekcia syntaktických konfliktov, ktorá si vyžaduje iba štrukturálne porovnanie modelov. Existuje však aj viacero prác zameraných na detekciu sémantických konfliktov, kde môžeme hovoriť o statickej, behaviorálnej a ontologickej stránke.

Tabuľka 1 ponúka prehľad analyzovaných metód a nástrojov, kde uvádzame typy konfliktov, ktoré detegujú, a oblasť ich použitia. Statické sémantické konflikty sú detegované hlavne metódami Rainbow a SMoVer, SMoVer sa navyše zaoberá aj behaviorálnou stránkou sémantických konfliktov. Chýbajú však prístupy zamerané na detekciu ontologických konfliktov.

Tabuľka 1: *Prehľad metód zameraných na detekciu konfliktov.*

Metóda	Typy konfliktov	Použitie
Rainbow	syntaktické a statické sémantické konflikty	UML diagram tried, UML diagram prípadov použitia
Modifikácie grafov	syntaktické konflikty a konflikty porušujúce obmedzenia modelu	všeobecné
SMoVer	statické a behaviorálne sémantické konflikty	modelovacie jazyky založené na EMF
Mirador	syntaktické konflikty	Ecore a Fujaba modely
SCROL	dátové a schematické sémantické konflikty	heterogénne databázy

4 Metóda na detekciu konfliktov

Obsahom tejto kapitoly je návrh metódy, ktorej hlavným cieľom je detegovať konflikty v UML modeli. Na základe vykonanej analýzy metód a nástrojov na detekciu konfliktov sa metóda zameriava na detekciu statických sémantických a ontologických konfliktov s využitím ontológií a mapovania ontológií.

Ako sme uviedli v kapitole 2, statické sémantické konflikty sú spôsobené porušením statických vlastností modelu, čím vzniká neplatný zlúčený model (Altmanninger a Pierantonio, 2011). Ontologické konflikty vznikajú použitím lingvisticky rôznych termínov, ktoré označujú tú istú vec alebo sú v určitom sémantickom vzťahu.

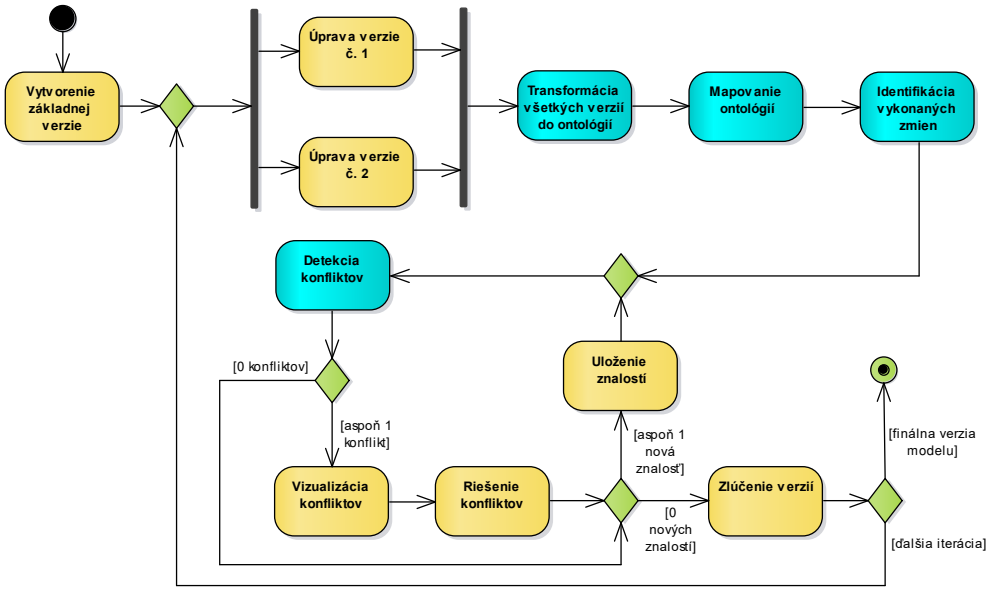
Zo skupiny štrukturálnych diagramov jazyka UML navrhujeme metódu pre detekciu konfliktov v diagrame tried, ktorý je najpoužívanejší UML diagram v softvérových projektoch, knihách, nástrojoch, kurzoch a tutoriáloch (Reggio et al., 2013).

Hypotéza: Našou hypotézou je tvrdenie, že použitím zvolenej metódy na mapovanie ontológií dokážeme detegovať niektoré syntaktické, statické sémantické a ontologické konflikty v UML modeli tried, ktoré môžu vzniknúť paralelnou prácou vývojárov.

Základná schéma navrhovanej metódy je vyjadrená diagramom aktivít na obr. 15. Metóda má na vstupe základnú verziu modelu a od nej odvodené verzie modelu, ktoré sú upravené vývojármi alebo skupinou vývojárov nezávisle od seba. Medzi hlavné kroky metódy, ktoré sú podrobnejšie opísané v nasledujúcich podkapitolách, patria:

1. transformácia základnej verzie modelu a odvodených verzií modelu do ontológií. Navrhli sme schému ontológie pre model tried, na základe ktorej je každý prvok modelu tried transformovaný a týmto spôsobom je vytvorená ontológia pre verziu modelu. Výstupom tohto procesu sú 3 ontológie reprezentujúce jednotlivé verzie modelu.
2. mapovanie ontológií medzi základnou verziou a odvodenými verziami a medzi odvodenými verziami navzájom. Zvolili sme metódu na mapovanie ontológií, ktorá definuje sémantické vzťahy medzi jednotlivými konceptami a vlastnosťami ontológií a tak určuje sémanticky podobné prvky v jednotlivých verziách modelu.

3. identifikácia vykonaných zmien v odvodených verziách modelu vykonaná na základe mapovania ontológií.
4. detekcia statických sémantických a ontologických konfliktov v modeli, ktoré mohli vzniknúť paralelnou prácou vývojárov vo verziách modelu. Na základe mapovania ontológií je uskutočnená analýza vykonaných zmien a sémantických vzťahov s cieľom nájsť konfliktné zmeny.

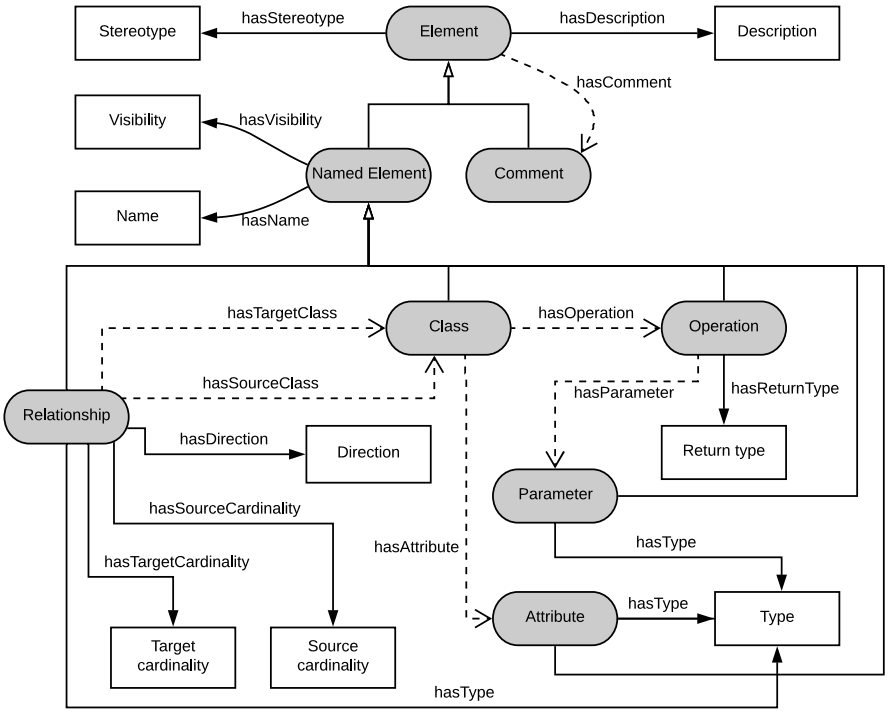


Obrázok 15: *Schéma aktivít navrhovanej metódy na detekciu konfliktov (zvýraznené modrou farbou) s jej využitím v procese modelovania softvéru.*

4.1 Transformácia modelu do ontológie

Prvým krokom navrhovanej metódy je transformácia základnej verzie modelu a upravených odvodených verzií modelu do ontológie, ktorej schéma je zobrazená na obr. 16. Výstupmi tohto procesu sú jednotlivé ontológie, ktoré reprezentujú jednotlivé verzie modelu.

Schéma ontológie definuje koncepty a vlastnosti pre model tried. Pri vytváraní tejto schémy sme nepostupovali presne podľa UML metamodelu, ktorého štruktúra je značne komplikovaná. Naším cieľom bolo definovanie schémy ontológie, ktorá je dostatočne jednoduchá pre manipuláciu a obsahuje najviac používané prvky modelu tried.



Obrázok 16: *Schéma ontológie pre model tried. Hlavnými konceptami ontológie sú trieda, atribút, operácia, parameter, poznámka a vzťah, ktoré dedia od elementu. Rozhranie uvažujeme ako triedu so stereotypom «interface». Poznámka môže byť priradená ku všetkým konceptom ontológie. Orientované hrany reprezentované plnou čiarou vyjadrujú priradenie vlastností ku konceptom ontológie. Orientované hrany reprezentované prerušovanou čiarou vyjadrujú vzťahy medzi konceptami, napr. trieda môže obsahovať 1 alebo viacero atribútov, vzťah spája 2 triedy.*

Z UML metamodelu sme použili prvok *Element* so stereotypom a opisom, od ktorého dedia všetky elementy jazyka UML, a prvok *NamedElement* (pomenovaný element) s názvom a viditeľnosťou.

Koncept *Relationship* (vzťah) môže byť typu asociácia, generalizácia, realizácia, agregácia, kompozícia alebo závislosť a slúži na spojenie 2 tried s určeným smerom a kardinalitou na oboch stranách vzťahu. Každý pomenovaný element má vlastnosť *Visibility* (viditeľnosť) v modeli, ktorá môže byť *public*, *private*, *protected* alebo *package*.

Samotný proces transformácie modelu do ontológie je zapísaný pseudokódom v ukážke 2. Cieľom je transformácia každého prvku modelu do jeho zodpovedajúceho konceptu ontológie spolu s jeho vlastnosťami na základe definovanej schémy ontológie.

```
algorithm model-transformation:
    input: model versions
    output: ontologies

    Initialize ontology and ontologies as empty lists

    for each version in model versions do
        for each class in version do
            Transform class with its properties and add to ontology

            for each attribute in class do
                Transform attribute with its properties and add to ontology
            endfor

            for each operation in class do
                Transform operation with its properties and add to ontology
                for each parameter in operation do
                    Transform parameter with its properties and add to ontology
                endfor
            endfor
        endfor

        for each comment in version do
            Transform comment with its properties and add to ontology
        endfor

        for each relationship in version do
            Transform relationship with its properties and add to ontology
        endfor

        Add ontology to ontologies
    endfor
end
```

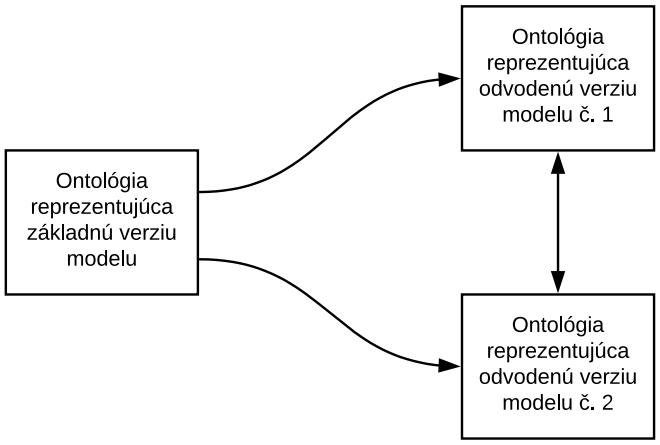
Ukážka 2: *Pseudokód algoritmu transformácie modelu do ontológie. Proces pozostáva postupne z transformácie všetkých tried modelu, ich atribútov, operácií a parametrov operácií do príslušných konceptov ontológie spolu s ich vlastnosťami. Potom nasleduje transformácia poznámok a vzťahov medzi triedami do príslušných konceptov ontológie spolu s ich vlastnosťami.*

4.2 Mapovanie ontológií

Získané ontológie reprezentujúce jednotlivé verzie modelu sú vstupom do metódy na mapovanie ontológií. Mapovanie ontológií je definované nasledujúcim spôsobom: ak existujú ontológie A a B, mapovanie ontológie A na ontológiu B vyžaduje pre každý koncept ontológie A nájsť súvisiaci koncept v ontológii B, ktorý má rovnaký alebo podobný význam (Ehrig a Sure, 2004).

Metóda na mapovanie ontológií slúži na identifikáciu sémantických vzťahov medzi konceptami ontológií, z ktorej je možné odvodiť zmeny vykonané v jednotlivých odvodených verziách modelu, a je dôležitým predpokladom pre následnú detekciu konfliktov. Výstupom mapovania sú dvojice konceptov jednotlivých ontológií, ktoré sú v určitom sémantickom vzťahu. Metóda pozostáva z vykonania dvoch mapovaní, ako je znázornené na obr. 17. Ide o:

1. mapovanie ontológie reprezentujúcej základnú verziu modelu na ontológie reprezentujúce všetky odvodené verzie modelu,
2. mapovanie ontológií reprezentujúcich odvodené verzie modelu navzájom.



Obrázok 17: *Schéma mapovania ontológií, ktoré reprezentujú jednotlivé verzie modelu. Hrany vyjadrujú smer mapovania a spájajú ontológie použité v mapovaní.*

Pri návrhu metódy na mapovanie ontológií sme vzájomne skombinovali už existujúce základné metódy (Hooi et al., 2014), ktoré sú založené na spracovaní prirodzeného jazyka, podobnosti slov a identifikátorov prvkov modelu, a rozšírené metódy ARTEMIS (Castano et al., 2001) a COMA (Do a Rahm, 2002).

Spomenuté rozšírené metódy poskytujú vhodné vlastnosti pre identifikáciu podobných prvkov vo verziách modelu. Metóda ARTEMIS určuje podobnosť prvkov na základe kombinácie porovnania ich názvov a porovnania ich štruktúry, vďaka čomu uvažuje terminologické a štrukturálne aspekty prvkov. Metóda COMA definuje hodnotu podobnosti prvkov v intervale $<0, 1>$ a umožňuje výber parametrov, ktoré sa majú použiť na určenie podobnosti.

Navrhovaná metóda na mapovanie ontológií identifikuje 2 typy sémantických vzťahov medzi konceptami ontológií, ktorých proces identifikácie je opísaný v ďalších podkapitolách. Medzi tieto sémantické vzťahy patria:

- vzťah *same_as*, ktorý vyjadruje mieru sémantickej podobnosti medzi konceptami ontológií,
- vzťah *is_a* (vzťah hyperonymie), ktorý vyjadruje významovú nadradenosť medzi konceptami ontológií.

4.2.1 Identifikácia same_as vzťahov

Pre identifikáciu *same_as* vzťahov medzi konceptami ontológií je potrebné sémanticky porovnať tieto koncepty. Tento proces pozostáva z porovnania názvov konceptov, ostatných vlastností konceptov a vzťahov s inými konceptami. Pre všetky jednotlivé porovnania je priradená váha v intervale <0, 1>, ktorá vyjadruje jeho dôležitosť pri vypočítaní hodnoty celkovej sémantickej podobnosti medzi konceptami. Súčet priradených váh je 1.

Celková sémantická podobnosť medzi konceptami je určená hodnotou v intervale <0, 1>, kde 0 je žiadna podobnosť medzi konceptami a 1 je silná podobnosť medzi konceptami. Táto hodnota vznikne súčtom hodnôt sémantickej podobnosti názvov, ostatných vlastností konceptov a vzťahov s inými konceptami, ktoré sú vynásobené priradenými váhami.

Porovnanie názvov konceptov je riešené pomocou slovníka pojmov a algoritmu určeného na porovnávanie slov. Zo slovníka pojmov využívame vzťah synonymie (vzťah významovej podobnosti), pre ktorý je definovaná váha v intervale (0, 1). Názvy konceptov, ktoré nie sú v synonymickom vzťahu, sú porovnané pomocou algoritmu, ktorý určí lexikálne rozdiely medzi nimi. V takom prípade je hodnota sémantickej podobnosti medzi názvami konceptov v intervale <0, 1>, kde hodnota rovná 1 je určená v prípade rovnosti názvov.

Ostatné vlastnosti konceptov, napr. stereotyp, viditeľnosť, opis alebo typ, sú vzájomne porovnané a hodnota ich podobnosti vynásobená priradenou váhou je započítaná do hodnoty celkovej podobnosti medzi konceptami. Medzi najdôležitejšie vlastnosti patrí opis konceptu, na základe ktorého je možné dedukovať jeho význam.

V poslednej časti procesu sú vzájomne porovnané vzťahy konceptov s ostatnými konceptami. V tejto časti sa zohľadňujú nielen vzťahy, ktoré spájajú

koncepty, napr. asociácia, dedenie alebo závislosť, ale aj iné koncepty, ktoré sú k porovnávaným konceptom priradené, napr. atribúty, operácie, parametre operácií alebo poznámky. Hodnota ich podobnosti vynásobená priradenou váhou je tiež započítaná do hodnoty celkovej podobnosti medzi konceptami.

Pseudokód algoritmu pre identifikáciu *same_as* vzťahov je rozdelený do 2 častí a je zapísaný v ukážkach 3 a 4. Vykonáva sa v uvedenom poradí. Najprv sú v 1. časti identifikované sémanticky najviac podobné triedy podľa opísaného algoritmu, ktoré sú v 2. časti použité na nájdenie ostatných sémanticky najviac podobných konceptov. V oboch častiach je dôležitá stanovená hodnota prahu sémantickej podobnosti, ktorý reprezentuje minimálnu hodnotu celkovej sémantickej podobnosti na to, aby koncepty mohli byť vo vzťahu *same_as*.

```
algorithm ontology-mapping-part-1:
    input: 2 ontologies ontologyA and ontologyB, threshold, set of weights
    output: set of same_as relationships with semantic similarity value
        in the interval [0, 1]

    Initialize same_as_relationships as an empty list
    Set max_similarity to 0

    for each conceptA in ontologyA do
        for each conceptB in ontologyB do
            if (conceptA is class and conceptB is class) then
                Compare conceptA and conceptB
                Calculate semantic_similarity
                if (semantic_similarity >= max_similarity) then
                    Set same_as_node to conceptB
                    Set max_similarity to semantic_similarity
                endif
            endif
        endfor
        if (max_similarity >= threshold) then
            Set same_as_relationship between conceptA and same_as_node
            Add same_as_relationship to same_as_relationships
        endif
        Set max_similarity to 0
    endfor
end
```

Ukážka 3: *Pseudokód algoritmu mapovania ontológií - časť 1. Algoritmus prehľadáva triedy oboch ontológií a pre každú triedu v ontológii A identifikuje sémanticky najviac podobnú triedu v ontológii B. Porovnanie tried je vykonané na základe opísaného algoritmu, kde je možné ľubovoľne modifikovať hodnoty váh a hodnotu prahu sémantickej podobnosti. Vo výslednej množine same_as vzťahov sa nachádzajú same_as vzťahy medzi sémanticky najviac podobnými triedami spolu s hodnotou ich sémantickej podobnosti, ktorá je väčšia alebo rovná ako stanovená hodnota prahu.*

```
algorithm ontology-mapping-part-2:
    input: 2 ontologies ontologyA and ontologyB, threshold, set of weights,
        same_as classes
    output: set of same_as relationships with semantic similarity value
        in the interval [0, 1]

    Initialize same_as_relationships as an empty list
    Set max_similarity to 0

    for each conceptA in ontologyA do
        for each conceptB in ontologyB do
            if (conceptA is attribute and conceptB is attribute) then
                 if (conceptA and conceptB are in same_as classes) then
                    Compare conceptA and conceptB
                    Calculate semantic_similarity
                    if (semantic_similarity >= max_similarity) then
                        Set same_as_node to conceptB
                        Set max_similarity to semantic_similarity
                    endif
                endif
            endif
            if (conceptA is relationship and conceptB is relationship) then
                Compare conceptA and conceptB
                Calculate semantic_similarity
                if (semantic_similarity >= max_similarity) then
                    Set same_as_node to conceptB
                    Set max_similarity to semantic_similarity
                endif
            endif
        endfor
        if (max_similarity >= threshold) then
            Set same_as_relationship between conceptA and same_as_node
            Add same_as_relationship to same_as_relationships
        endif
        Set max_similarity to 0
    endfor
end
```

Ukážka 4: *Pseudokód algoritmu mapovania ontológií - časť 2. Algoritmus prehľadáva atribúty a vzťahy oboch ontológií. Pre každý atribút v ontológii A identifikuje sémanticky najviac podobný atribút v ontológii B, pričom tieto atribúty musia byť definované v sémantických podobných triedach identifikovaných v 1. časti algoritmu. Pre sémanticky podobné vzťahy musí platiť, že sú rovnakého typu a spájajú sémanticky podobné triedy. Vo výslednej množine same_as vzťahov sa nachádzajú same_as vzťahy medzi sémanticky najviac podobnými atribútmi a vzťahmi spolu s hodnotou ich sémantickej podobnosti, ktorá je väčšia alebo rovná ako stanovená hodnota prahu. Algoritmus založený na tomto princípe sa aplikuje aj pre koncepty operácia, parameter operácie a poznámka.*

4.2.2 Identifikácia is_a vzťahov

Identifikácia *is_a* vzťahov je vykonávaná iba medzi triedami modelu. Pre identifikáciu týchto vzťahov je potrebné porovnať iba názvy jednotlivých tried. Podobne ako pri identifikácii *same_as* vzťahov je porovnanie názvov riešené pomocou slovníka pojmov, z ktorého využívame vzťah hyperonymie (vzťah významovej nadradenosti). V tomto prípade nie je uvažovaná žiadna hodnota sémantickej podobnosti, žiadna hodnota prahu ani žiadne váhy. Pseudokód algoritmu pre identifikáciu *is_a* vzťahov je zapísaný v ukážke 5.

```
algorithm ontology-mapping-part-3:
    input: 2 ontologies ontologyA and ontologyB
    output: set of is_a relationships

    Initialize is_a_relationships as an empty list

    for each conceptA in ontologyA do
        for each conceptB in ontologyB do
            if (conceptA is class and conceptB is class) then
                if (names of concepts are in hyperonymy relation) then
                    Set is_a_relationship between conceptA and conceptB
                    Add is_a_relationship to is_a_relationships
                endif
            endif
        endfor
    endfor
end
```

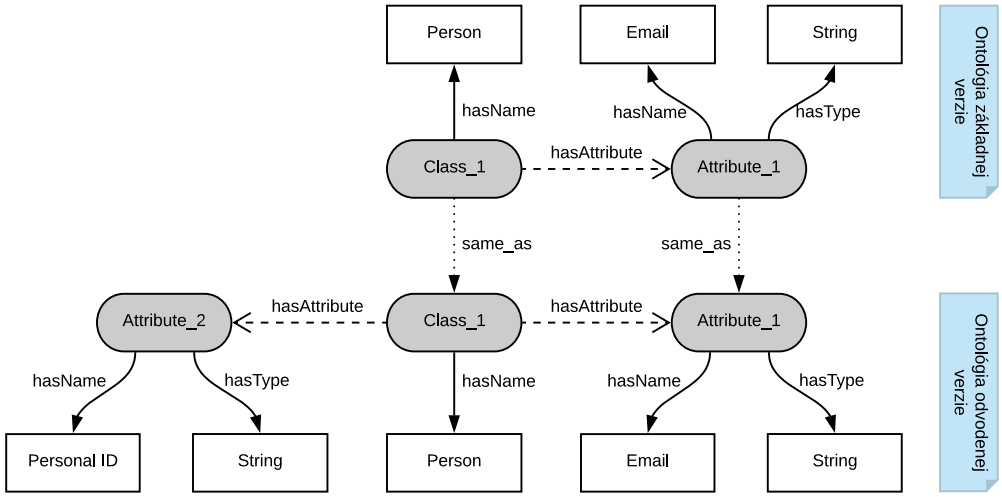
Ukážka 5: *Pseudokód algoritmu mapovania ontológií - časť 3. Algoritmus prehľadáva triedy oboch ontológií a pre každú triedu v ontológii A identifikuje triedy v ontológii B, ktoré sú vo vzťahu hyperonymie. V prípade, že sú názvy tried vo vzťahu hyperonymie, vznikne nový is_a vzťah medzi týmito triedami, ktorý je uložený do množiny všetkých is_a vzťahov.*

4.3 Identifikácia vykonaných zmien v modeli

Z množiny vytvorených *same_ as* vzťahov vieme odvodiť zmeny, ktoré vykonali vývojári v odvodených verziách modelu. Ide teda o stavový prístup k identifikácii zmien vo verziách modelu, keďže počas úpravy verzií modelu nie sú uchovávané žiadne zmeny. Medzi zmeny, ktoré môžu byť vykonané v modeli, patria pridanie, úprava a odstránenie prvku modelu, resp. konceptu ontológie.

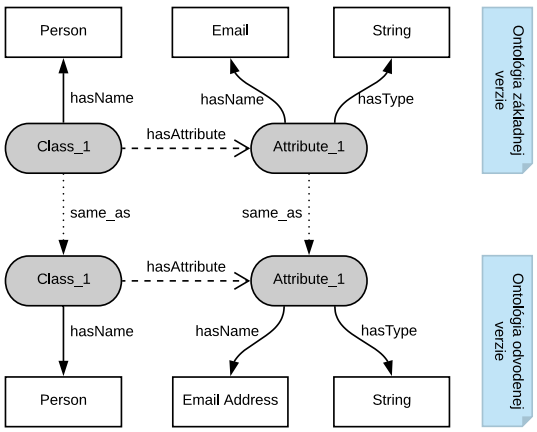
Pre lepšie vysvetlenie identifikácie vykonaných zmien v odvodených verziách modelu použijeme príklad modelu tried. Základná verzia tohto modelu pozostáva z triedy s názvom *Person*, ktorá obsahuje atribút s názvom *Email* typu *String*. Základná verzia a odvodené verzie modelu boli transformované do ontológie. Zmeny identifikujeme nasledujúcim spôsobom:

- pridanie prvku - prvok je pridaný do verzie modelu vtedy, ak sa nenájde sémanticky podobný prvok v 1. mapovaní ontológií. Príklad pridania prvku do uvedeného modelu tried je znázornený na obr. 18.



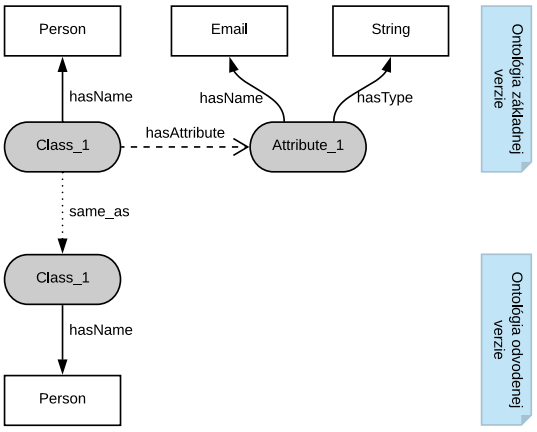
Obrázok 18: *Príklad pridania prvku. Vývojár pridal do svojej odvodenej verzie modelu atribút s názvom Personal ID do triedy Person. Keďže sa v 1. mapovaní ontológií nenájde same_ as vzťah pre pridaný atribút v danej triede, možno povedať, že atribút bol pridaný do triedy v danej odvodenej verzii modelu.*

- úprava prvku - vlastnosť prvku je zmenená vtedy, ak sa pri 1. mapovaní ontológií zistí, že vlastnosť prvku v odvodenej verzii modelu sa nezhoduje s vlastnosťou prvku v základnej verzii modelu. Príklad úpravy prvku v uvedenom modeli tried je znázornený na obr. 19.



Obrázok 19: *Príklad úpravy prvku. Vývojár vo svojej odvodenej verzii modelu zmenil názov atribútu Email na Email address. Vďaka metóde na mapovanie ontológií vieme, že atribúty Email a Email address v jednotlivých ontológiách sú sémanticky podobné a porovnaním ich názvov je odhalená úprava prvku.*

- odstránenie prvku - prvok je odstránený z verzie modelu vtedy, ak sa nenájde sémanticky podobný prvok v 1. mapovaní ontológií. Príklad odstránenia prvku v uvedenom modeli tried je znázornený na obr. 20.



Obrázok 20: *Príklad odstránenia prvku. Vývojár vo svojej odvodenej verzii modelu odstránil atribút Email. V mapovaní ontológií vidíme, že pre atribút Email v základnej verzii modelu neexistuje sémanticky podobný atribút v odvodenej verzii modelu.*

4.4 Detekcia konfliktov

Ďalšou časťou metódy je samotná detekcia konfliktov v modeli. V prvom rade identifikujeme konfliktné zmeny z množín zmien, ktoré boli vykonané v jednotlivých odvodených verziách modelu. Jednotlivé množiny zmien porovnáme a sledujeme navzájom si odporujúce zmeny a ekvivalentné zmeny modelu.

Na základe konfliktných zmien vieme jednoznačne určiť typ konfliktov. V tabuľke 2 sú uvedené možné konflikty, ktoré môžu nastať.

Tabuľka 2: *Konflikty, ktoré môžu nastať pri paralelnej práci vývojárov vo verziách modelu. Symbol „x“ znamená, že sa prvok nachádza v danej verzii modelu.*

Číslo	Základná verzia	Verzia č. 1	Verzia č. 2	Možné konflikty
1	x	x	x	Rôzna úprava rovnakej vlastnosti
2	x	x		Úprava a odstránenie rovnakého prvku
3	x			Žiadne konflikty
4		x		Žiadne konflikty
5		x	x	Pridanie sém. podobných prvkov, pridanie tried vo vzťahu hyperonymie

Medzi ďalšie konflikty, ktoré môžu vzniknúť pri paralelnej práci vývojárov a sú detegované navrhovanou metódou, patria nasledujúce konflikty:

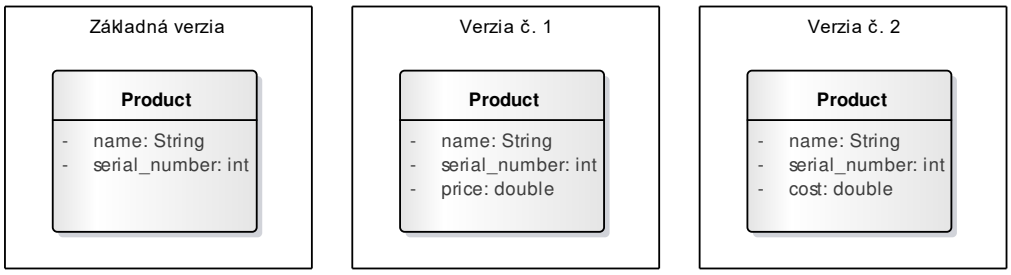
- pridanie vzťahu medzi triedami v 1. verzii modelu a odstránenie 1 z týchto tried v 2. verzii modelu,
- pridanie alebo zmena vlastností potomka prvku v 1. verzii modelu a odstránenie tohto prvku v 2. verzii modelu - napr. potomkom môže byť atribút umiestnený v triede, ktorá je odstránená v inej verzii modelu.

Tieto konflikty a konflikty číslo 1 a 2 v tabuľke 2 patria medzi syntaktické konflikty. Detekcii týchto konfliktov sa venujeme v podkapitole 4.4.4.

V ďalších podkapitolách rozoberáme podrobnejšie niektoré statické sémantické a ontologické konflikty, ktoré deteguje navrhovaná metóda. Súčasťou podkapitol je príklad takýchto konfliktov a podmienky, ktoré musia platiť pre ich detekciu.

4.4.1 Pridanie sémanticky podobných prvkov

Konflikt pridania sémanticky podobných prvkov patrí medzi ontologické konflikty. Príklad tohto typu konfliktu je znázornený na obr. 21. Tento konflikt nastal po tom, ako vývojári nezávisle od seba pridali do rovnakej triedy *Product* vo svojej verzii modelu atribút označujúci cenu produktu. Problém spočíva v rozličnom pomenovaní tohto atribútu.



Obrázok 21: *Príklad konfliktu pridania sémanticky podobných prvkov do jednotlivých verzií modelu. Vývojári pridali atribúty price a cost do triedy Product, ktoré vyjadrujú cenu produktu.*

Konfliktnými zmenami sú pridania 2 atribútov do triedy *Product*. Na základe mapovania ontológií medzi odvodenými verziami modelu, v ktorom využívame slovník pojmov, vieme, že pridané atribúty sú v sémantickom vzťahu *same_as*, pretože ich názvy *price* a *cost* sú v synonymickom vzťahu. Ak by tento konflikt nebol detegovaný, trieda *Product* v zlúčenej verzii modelu by obsahovala oba atribúty *price* a *cost*, ktoré vyjadrujú tú istú vlastnosť produktu.

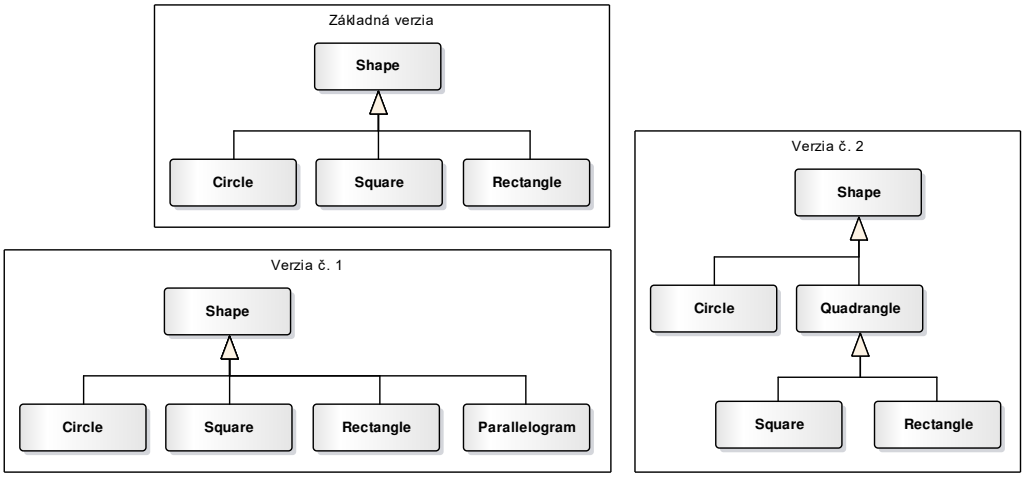
Pre detekciu tohto typu konfliktu musia platiť nasledujúce podmienky:

- prvky musia byť pridané do odvodených verzií modelu,
- prvky musia byť sémanticky najviac podobné, v tom prípade je medzi nimi identifikovaný vzťah *same_as* na základe mapovania ontológií medzi odvodenými verziami modelu.

Ak tieto podmienky platia, všetky vlastnosti pridaných prvkov sú vzájomne porovnané. V popise konfliktu sa následne nachádzajú informácie o rozdieloch medzi prvkami.

4.4.2 Pridanie tried vo vzťahu hyperonymie

Ďalší ontologickým konfliktom je konflikt pridania tried, ktoré sú vo vzťahu hyperonymie, na rovnakú úroveň v hierarchii dedenia. Príklad tohto typu konfliktu je znázornený na obr. 22, ktorý nastal po tom, ako vývojári nezávisle od seba pridali triedu do hierarchie dedenia vo svojej verzii modelu.



Obrázok 22: *Príklad konfliktu pridania tried vo vzťahu hyperonymie do hierarchie dedenia (Altmanninger a Pierantonio, 2011). Vývojár pridal do verzie č. 1 triedu Parallelogram (rovnobežník). Vo verzii č. 2 bola hierarchia dedenia upravená, vývojár pridal triedu Quadrangle (štvoruholník), od ktorej dedia triedy Square a Rectangle.*

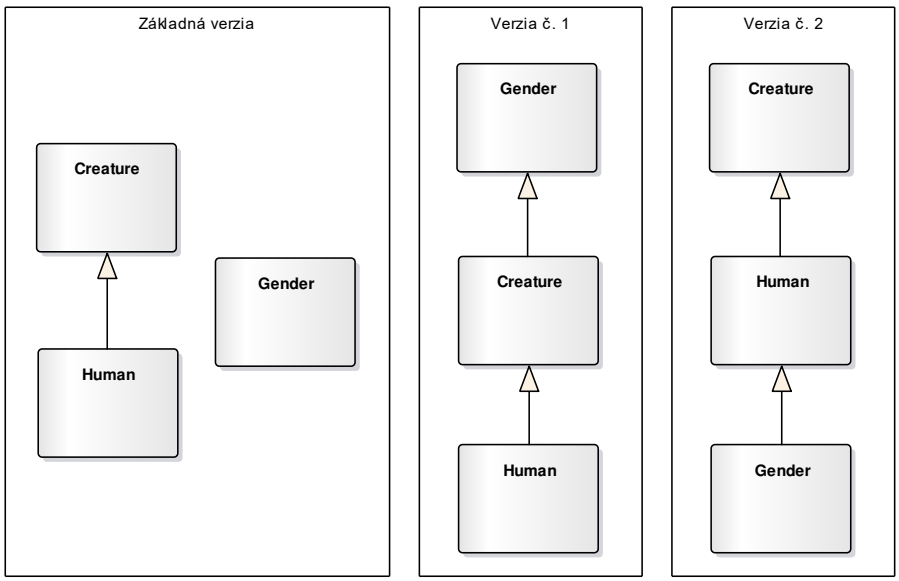
Konfliktnými zmenami sú pridania tried *Parallelogram* a *Quadrangle*, ktorých názvy sú vo vzťahu hyperonymie. Ak by tento konflikt nebol detegovaný, pridané triedy by boli v zlúčenej verzii modelu na rovnakej úrovni v hierarchii dedenia. Navrhovaná metóda upozorňuje na tento ontologický konflikt, keďže identifikuje vzťah *is_a* medzi pridanými triedami na základe mapovania ontológií medzi odvodenými verziami modelu.

Pre detekciu tohto typu konfliktu musia platiť nasledujúce podmienky:

- triedy musia byť pridané do odvodených verzií modelu,
- medzi triedami musí byť identifikovaný vzťah *is_a* na základe mapovania ontológií medzi odvodenými verziami modelu,
- triedy musia dediť od tried, ktoré sú sémanticky najviac podobné, tzn. existuje vzťah *same_as* medzi týmito triedami na základe mapovania ontológií medzi odvodenými verziami modelu.

4.4.3 Cyklus dedenia v zlúčenej verzii modelu

Medzi statické sémantické konflikty patrí konflikt cyklu dedenia v zlúčenej verzii modelu. Na obr. 23 je znázornený príklad tohto typu konfliktu, ktorý vznikol v situácii, keď vývojári pridali do svojej verzie modelu vzťahy generalizácie medzi triedami.



Obrázok 23: *Príklad konfliktu cyklu dedenia v zlúčenej verzii modelu (Altmanninger a Pierantonio, 2011). Vo verzii č. 1 bol pridaný vzťah generalizácie medzi triedami Creature a Gender a vo verzii č. 2 bol pridaný vzťah generalizácie medzi triedami Gender a Human.*

V tomto príklade sú konfliktnými zmenami pridania vzťahov generalizácie medzi triedami. V prípade, že by tento konflikt nebol detegovaný, vykonané zmeny by boli aplikované v zlúčenej verzii modelu a spôsobili by cyklus dedenia pre uvedené triedy.

Tento typ konfliktu môže nastať po vykonaní rôzneho počtu rôznych zmien v odvodených verziách modelu. Z tohto dôvodu by bolo komplikované definovať podmienky pre jeho detekciu, a preto navrhujeme nasledujúce kroky:

1. aplikovanie všetkých vykonaných zmien do 1 zlúčenej verzie modelu,
2. detekcia cyklu dedenia začínajúca v každej triede v zlúčenej verzii modelu pomocou rekurzívneho algoritmu, ktorý sa postupne presúva do nadtried v hierarchii dedenia a deteguje konflikt vtedy, ak sa dostane do počiatočnej triedy.

4.4.4 Syntaktické konflikty

Medzi základné syntaktické konflikty patria konflikty typu *Update/Update* a *Update/Delete*. Na detekciu konfliktu typu *Update/Update* navrhujeme použitie množín úprav prvkov, ktoré boli vykonané v odvodených verziách modelu, a množiny *same_as* vzťahov medzi odvodenými verziami modelu. Detekcia konfliktu pozostáva z nasledujúcich krokov:

1. prehľadávame vzájomne obe množiny úprav prvkov a hľadáme také úpravy, ktoré sa týkajú rovnakej vlastnosti prvku a nové hodnoty týchto vlastností nie sú zhodné,
2. pre nájdené úpravy sledujeme prvky, ktorých sa týkajú, a konflikt nastáva vtedy, ak sú tieto prvky vo vzťahu *same_as* v množine *same_as* vzťahov medzi odvodenými verziami modelu.

Detekcia konfliktu typu *Update/Delete* by mala zahŕňať použitie množiny úprav vykonaných v 1. verzii modelu, množiny odstránení vykonaných v 2. verzii modelu a množiny *same_as* vzťahov medzi základnou verziou a 1. verziou modelu. V tomto prípade prehľadávame vzájomne množinu úprav a množinu odstránení a konflikt nastáva vtedy, ak upravený prvok a odstránený prvok modelu sú vo vzťahu *same_as* v množine *same_as* vzťahov medzi základnou verziou a 1. verziou modelu.

Ďalším typom syntaktického konfliktu je konflikt, ktorý vzniká pridaním vzťahu medzi triedami v 1. verzii modelu a odstránením 1 z týchto tried v 2. verzii modelu. Pre detekciu tohto konfliktu je potrebné použiť množinu pridaní v 1. verzii modelu, množinu odstránení v 2. verzii modelu a množinu *same_as* vzťahov medzi základnou verziou a 1. verziou modelu. Detekcia konfliktu pozostáva z nasledujúcich krokov:

1. nájdeme všetky triedy v 1. verzii modelu, medzi ktorými bol pridaný ľubovoľný vzťah,
2. prehľadávame množinu odstránení v 2. verzii modelu a konflikt nastáva vtedy, ak odstránená trieda je vo vzťahu *same_as* s ľubovoľnou triedou nájdenou v kroku 1 v množine *same_as* vzťahov medzi základnou verziou a 1. verziou modelu.

5 Vyhodnotenie metódy

Navrhovanú metódu na detekciu sémantických konfliktov sme vyhodnotili prostredníctvom implementovaného softvérového prototypu a vytvoreného datasetu UML modelov tried, ktorý obsahuje rôzne typy konfliktov.

5.1 Prototyp

Prototyp sme implementovali v rámci architektúry klient-server, ktorej komponenty sú podrobnejšie opísané v prílohe B. Ide o ASP.NET aplikáciu, kde klientom je webový prehliadač a server je implementovaný v pracovnom rámci ASP.NET Core.

Prototyp má na vstupe základnú verziu modelu a 2 odvodené verzie modelu vo forme troch súborov s koncovkou .xmi. Tieto súbory musia byť exportované z nástroja Enterprise Architect vo formáte XMI 2.4.2, ktorý je podporovaný knižnicou slúžiacou na deserializáciu XMI. Prototyp načíta jednotlivé súbory a transformuje ich do hierarchií objektov, ktoré reprezentujú všetky balíky, modely a ich prvky uložené v súboroch.

5.1.1 Transformácia modelu do ontológie

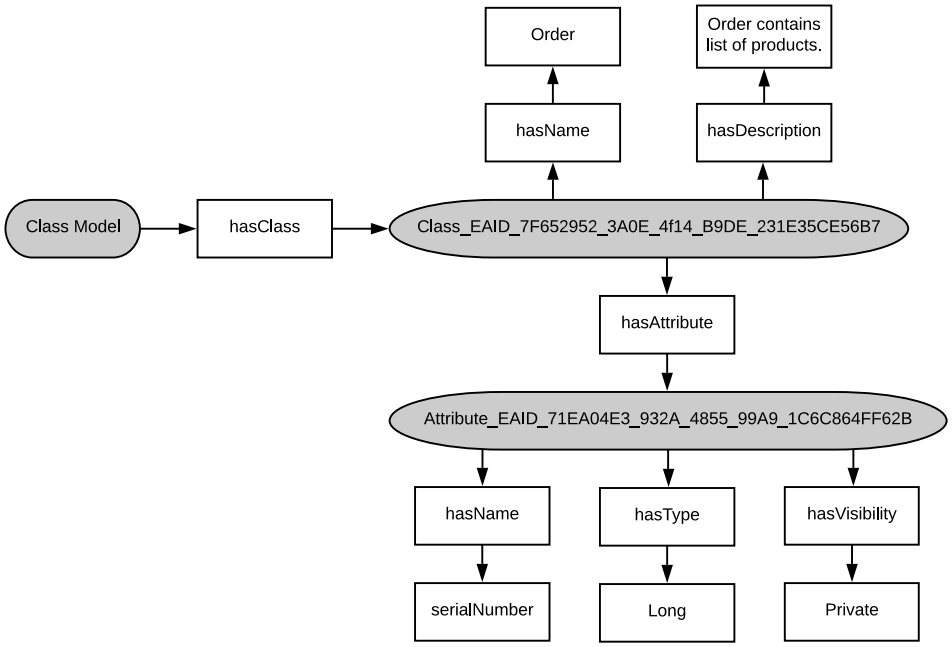
K jednotlivým prvkom modelu a ich vlastnostiam pristupujeme prehľadávaním hierarchie objektov v časti Extension, ktorá sa nachádza v načítanom XMI súbore. Najprv prototyp nájde zoznam identifikátorov všetkých tried, vzťahov a poznámok, ktoré sú súčasťou modelu typu *Logical* (model tried). Následne hľadá ich vlastnosti a potomkov (napr. atribúty) na základe ich identifikátora a typu, ktorý je zapísaný v špecifickom formáte, napr. *uml:Class* alebo *uml:Note*.

Týmto spôsobom sú získané všetky prvky vo verziách modelu a ich vlastnosti, ktoré sú uvedené ako koncepty a ich vlastnosti v schéme ontológie na obr. 16. Výnimkou sú operácie, parametre operácií a ich vlastnosti, ktoré nie sú transformované do hierarchie objektov pomocou použitej knižnice, a preto nie je možné k nim pristupovať.

V ďalšom kroku metódy sú všetky prvky danej verzie modelu a ich vlastnosti transformované do ontológie podľa definovanej schémy. V prototype používame ontológie založené na RDF (Resource Description Framework) pro-

stredníctvom funkcií voľne dostupnej .NET knižnice dotNetRDF[1] zameranej na spracovanie, dopytovanie a vytváranie RDF.

Všetky verzie modelu sú reprezentované pomocou trojíc vo formáte subjekt-predikát-objekt v RDF. Príklad vytvorenej ontológie pre verziu modelu tried je uvedený na obr. 24.



Obrázok 24: *Príklad ontológie v RDF, ktorá reprezentuje verziu modelu tried. Pozostáva z množiny trojíc subjekt-predikát-objekt. Príkladom trojice je subjekt Class_EAID_7F652952_3A0E_4f14_B9DE_231E35CE56B7, predikát hasName a objekt Order. Trojica vyjadruje, že trieda s daným identifikátorom má názov Order.*

5.1.2 Mapovanie ontológií

V procese mapovania ontológií sú vytvorené ontológie reprezentujúce jednotlivé verzie modelu navzájom porovnané s cieľom identifikovať *same_as* vzťahy a *is_a* vzťahy podľa uvedených algoritmov v ukážkach 3, 4 a 5. Na porovnanie názvov konceptov v ontológiách používame slovník pojmov WordNet[2] dostupný prostredníctvom voľne dostupnej .NET knižnice Syn.WordNet[3].

[1] https://www.dotnetrdf.org/
[2] https://wordnet.princeton.edu/
[3] https://developer.syn.co.in/tutorial/wordnet/index.html

WordNet je lexikálna databáza pre anglický jazyk, ktorá zoskupuje podstatné mená, prídavné mená, slovesá a príslovky do množín kognitívnych synoným. WordNet uchováva aj vzťahy medzi anglickými slovami, konkrétne ide o vzťah synonymie, vzťah hyperonymie (vzťah významovej nadradenosti) a vzťah meronymie (vzťah časť-celok).

Pre identifikáciu *same_as* vzťahov porovnávame sémanticky názvy a opisy konceptov v jednotlivých ontológiách. Názvy a opisy majú priradené váhy, ktorých súčet je 1. Tieto váhy je možné ľubovoľne meniť v závislosti od toho, ktorú z vlastností názov a opis prvku modelu považujeme za dôležitejšiu pre určenie hodnoty sémantickej podobnosti medzi prvkami modelu. Sémantická podobnosť medzi konceptami ontológií je tak vypočítaná pomocou vzorca:

$$SemPodobnost = w * PodobnostNazvov + (1 - w) * PodobnostOpisov \quad (5.1)$$

kde w je váha pre názov prvku z intervalu <0, 1>.

Pri porovnaní názvov konceptov sledujeme, či sú v synonymickom vzťahu. Zo slovníka WordNet získame všetky synonymá pre daný názov konceptu a určíme, či sa názov konceptu v inej ontológii nachádza v získanej množine synoným. Ak názvy konceptov nie sú synonymá, vypočítame Levenshteinovu vzdialenosť medzi nimi s cieľom určenia ich lexikálnej podobnosti.

Na určenie podobnosti opisov konceptov používame Jaro-Winklerov algoritmus, ktorý podľa (W. Cohen et al., 2003) dosahuje dobré výsledky v porovnaní sekvencie slov. Použitie tohto algoritmu a Levenshteinovej vzdialenosti nám umožňuje voľne dostupná .NET knižnica SimMetricsCore[4].

Pre identifikáciu *is_a* vzťahov porovnávame názvy konceptov ontológií pomocou slovníka WordNet, z ktorého využívame vzťah hyperonymie. Získame všetky hyperonymá a hyponymá pre daný názov konceptu a rozhodneme, či sa názov konceptu v inej ontológii nachádza v získanej množine hyperoným a hyponým. Ak áno, medzi konceptami ontológií je identifikovaný *is_a* vzťah.

5.2 Metóda vyhodnotenia

Úspešnosť detekcie konfliktov v modeloch použitím navrhovanej metódy sme vyhodnotili pomocou datasetu UML modelov tried. Keďže sme nenašli voľne

[4]https://github.com/StefH/SimMetrics.Net

dostupný dataset UML modelov, vytvorili sme manuálne vlastný dataset, ktorý pozostáva zo základnej verzie a 2 odvodených verzií UML modelov tried.

V odvodených verziách modelov tried sme vykonali také konfliktné zmeny, ktoré vedú v procese zlúčenia verzií k vzniku rôznych syntaktických, statických sémantických, ontologických a iných sémantických konfliktov. Snažili sme sa aplikovať čo najväčší počet a kombináciu rôznych zmien tak, aby sme našu metódu na detekciu konfliktov čo najlepšie vyhodnotili a umožnili jej porovnanie s existujúcimi metódami.

Vytvorený dataset UML modelov tried je podrobne zdokumentovaný v prílohe E. Obsahuje nasledujúce typy konfliktov, pri ktorých uvádzame konfliktné zmeny v jednotlivých odvodených verziách modelu:

- Update-Update – rôzna úprava rovnakej vlastnosti rovnakých prvkov,
- Update-Delete – ľubovoľná úprava a odstránenie rovnakých prvkov,
- Add-Delete – pridanie vzťahu medzi triedami a odstránenie 1 z tried,
- UpdateSuccessor-DeleteAncestor – pridanie alebo ľubovoľná úprava potomka prvku a odstránenie prvku,
- Sémanticky podobné prvky – pridanie prvkov, ktoré majú rovnaký alebo podobný význam,
- Triedy vo vzťahu hyperonymie na rovnakej úrovni v hierarchii dedenia – pridanie tried, ktorých názvy sú vo vzťahu hyperonymie a ktoré dedia od tried s podobným významom,
- Cyklus dedenia v zlúčenej verzii modelu – vykonanie takých zmien, ktoré vedú k vzniku cyklu dedenia v zlúčenej verzii modelu,
- Rovnaké atribúty v hierarchii dedenia – pridanie, úprava alebo presunutie atribútov v rámci hierarchie dedenia tak, že v zlúčenej verzii modelu existujú rovnaké atribúty v rôznych triedach v hierarchii dedenia,
- Viacnásobné dedenie – pridanie generalizácií medzi triedami tak, že v zlúčenej verzii modelu vznikne viacnásobné dedenie pre danú triedu,
- Presúvanie atribútov pri refaktoringu – presunutie atribútov v rámci hierarchie dedenia a pridanie tried do hierarchie dedenia.

Vyhodnotenie úspešnosti navrhovanej metódy sme realizovali prostredníctvom štatistickej metódy F_1, ktorá berie do úvahy presnosť a pokrytie. Proces vyhodnotenia sme automatizovali pomocou funkcionality, ktorá automaticky porovnáva množinu konfliktov vo verziách modelu detegovaných navrhovanou

metódou s referenčnou množinou vytvorených konfliktov a zobrazí úspešnosť detekcie konfliktov pomocou hodnôt presnosti, pokrytia a F_1.

V procese vyhodnotenia metódy sme podobne ako v implementácii postupovali iteratívne. Navrhovanú metódu sme vyhodnotili v 3 iteráciách, ktoré sa od seba líšili iba v procese identifikácie *same_as* vzťahov. V prvých dvoch iteráciách sme sledovali iba názvy prvkov pre určenie ich sémantickej podobnosti. V tretej iterácii sme vzájomne porovnali názvy aj opisy prvkov, pričom súčet podobnosti názvov a podobnosti opisov vynásobených rôznymi váhami tvoril hodnotu celkovej sémantickej podobnosti prvkov, ako je uvedené vo vzorci 5.1. Vo všetkých iteráciách sme identifikovali *is_a* vzťahy podľa algoritmu v ukážke 5.

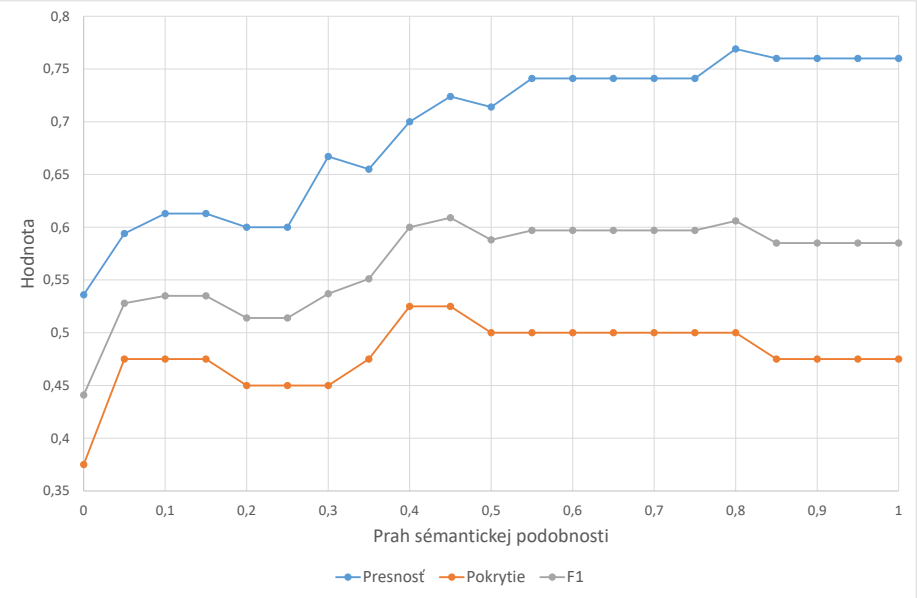
5.3 Vyhodnotenie detekcie konfliktov na základe sémantickej podobnosti názvov

V prvej iterácii náš prototyp porovnával iba názvy prvkov pri identifikácii *same_as* vzťahov, konkrétne určil, či názvy prvkov sú v synonymickom vzťahu, alebo sú identické. Neexperimentovali sme s hodnotou prahu sémantickej podobnosti. V tejto iterácii bola hodnota presnosti 0,72, hodnota pokrytia bola 0,35 a hodnota F_1 bola 0,47. Keďže hodnota pokrytia bola dosť nízka, snažili sme sa ju v ďalších iteráciách zvýšiť.

5.4 Vyhodnotenie detekcie konfliktov so zohľadnením lexikálnej podobnosti názvov

V druhej iterácii sme zahrnuli aj Levenshteinovu vzdialenosť pri porovnávaní názvov prvkov v procese identifikácie *same_as* vzťahov. Súčasťou vyhodnotenia bolo experimentovanie s hodnotou prahu sémantickej podobnosti a vypočítanie hodnôt presnosti, pokrytia a F_1 pre každú hodnotu prahu sémantickej podobnosti. Vyhodnotenie tejto iterácie je uvedené na obrázku 25.

Vo vyhodnotení tejto iterácie môžeme vidieť, že hodnota presnosti sa zvyšuje so zvyšujúcou sa hodnotou prahu sémantickej podobnosti. Maximálna hodnota presnosti je 0,769 pri hodnote prahu sémantickej podobnosti 0,8. Maximálna hodnota pokrytia je 0,525 pri hodnote prahu 0,4 a 0,45. Maximálna hodnota F_1 je 0,609 pri hodnote prahu 0,45.



Obrázok 25: *Vyhodnotenie druhej iterácie – hodnoty presnosti, pokrytia a F_1 pre každú hodnotu prahu sémantickej podobnosti pri porovnávaní názvov prvkov v procese identifikácie same_as vzťahov.*

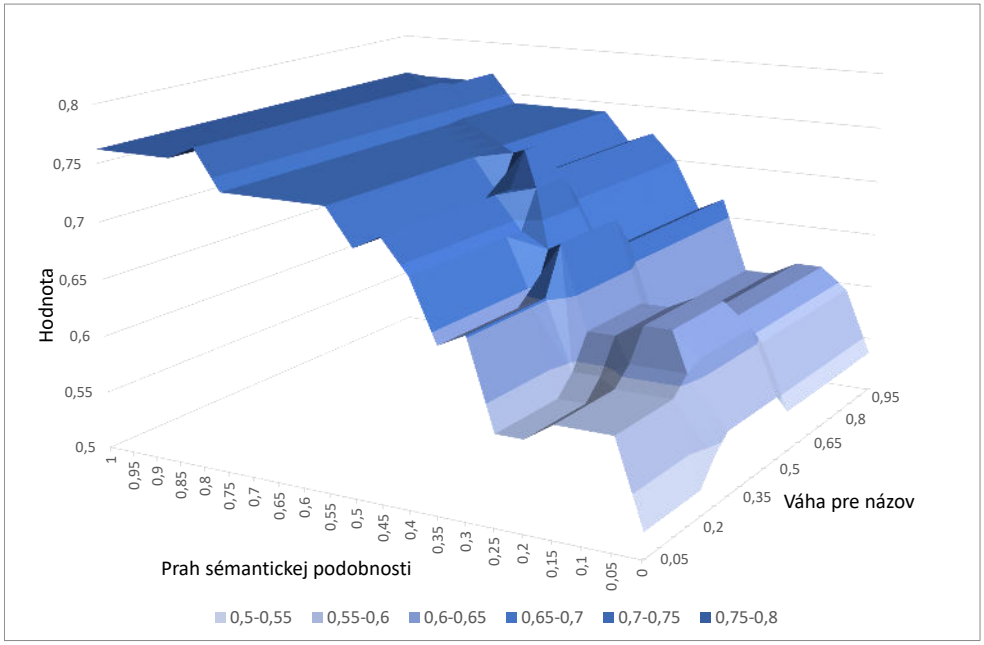
5.5 Vyhodnotenie detekcie konfliktov na základe podobnosti názvov a podobnosti opisov

Tretia finálna iterácia obsahovala porovnanie názvov aj opisov prvkov pri identifikácii *same_as* vzťahov. Sémantická podobnosť prvkov bola vypočítaná pomocou vzorca 5.1. Vyhodnotenie tejto iterácie zahŕňalo experimentovanie s hodnotou váhy pre názov a s hodnotou prahu sémantickej podobnosti. Hodnoty presnosti, pokrytia a F_1 sú uvedené na obrázkoch 26, 27 a 28.

Maximálna hodnota presnosti metódy na obr. 26 je 0,773 pri hodnote prahu sémantickej podobnosti 0,8 a hodnote váhy pre názov prvkov v intervale <0, 0,75>. Vidíme, že hodnota presnosti sa zvyšuje so zvyšujúcou sa hodnotou prahu sémantickej podobnosti.

Maximálna hodnota pokrytia metódy na obr. 27 je 0,525, ak hodnota váhy pre názov prvkov patrí do intervalu <0,45, 1> a hodnotu prahu sémantickej podobnosti nastavíme na 0,4 alebo 0,45.

Maximálna hodnota F_1 metódy na obr. 28 je 0,618 pri hodnote váhy pre názov 0,45 a 0,5 a hodnote prahu sémantickej podobnosti 0,45. Z toho vyplýva, že pre určovanie sémantickej podobnosti prvkov v rôznych verziách



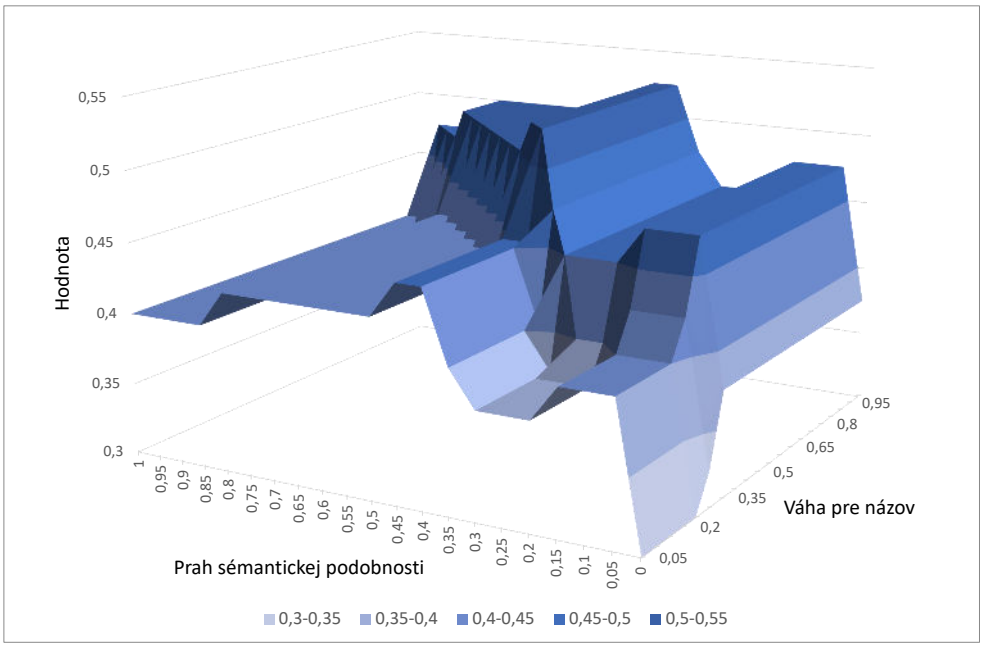
Obrázok 26: *Vyhodnotenie tretej iterácie – hodnoty presnosti metódy pri experimentovaní s hodnotou váhy pre názov prvkov a hodnotou prahu sémantickej podobnosti.*

modelu je najlepšie zahrnúť ich názov a opis s rovnakou váhou. To znamená, že názov a opis prvkov sú rovnako dôležité a súčet podobnosti názvov a opisov prvkov vo verziách modelu vynásobených rovnakými váhami dáva najlepšie výsledky detekcie konfliktov.

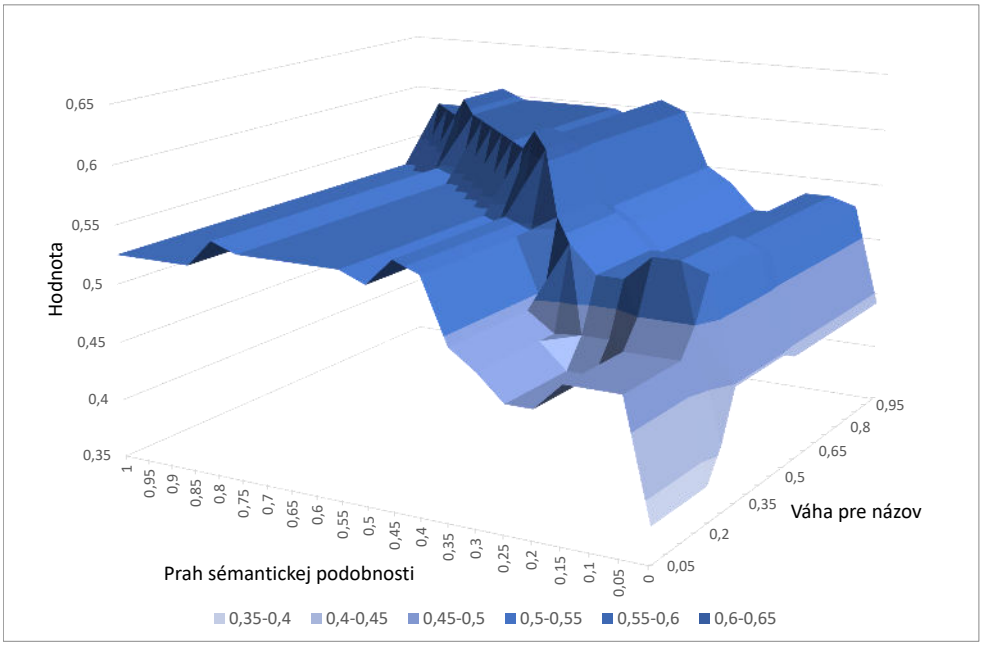
5.6 Overenie hypotézy

Našu hypotézu, ktorú sme uviedli v kapitole 4, sme overili pomocou vytvoreného datasetu. V odvodených verziách modelov tried sme vykonali konfliktné zmeny vedúce k vzniku tých syntaktických, statických sémantických a ontologických konfliktov, ktoré navrhovaná metóda deteguje. Identifikácia *same_as* vzťahov a vyhodnotenie boli vykonané rovnako ako v tretej iterácii. Hodnoty presnosti, pokrytia a F_1 sú uvedené na obrázkoch 29, 30 a 31.

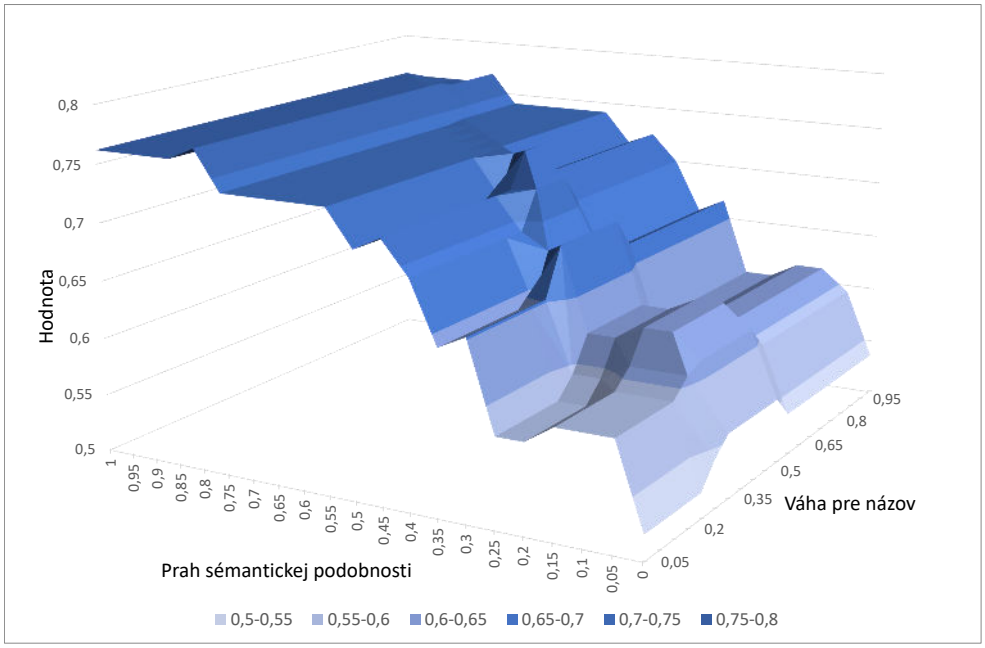
Dosiahli sme rovnaké výsledné hodnoty presnosti metódy ako v tretej iterácii. Maximálna hodnota pokrytia 0,808 a maximálna hodnota F_1 0,778 boli určené pri rovnakých hodnotách váhy pre názov a prahu sémantickej podobnosti ako v tretej iterácii.



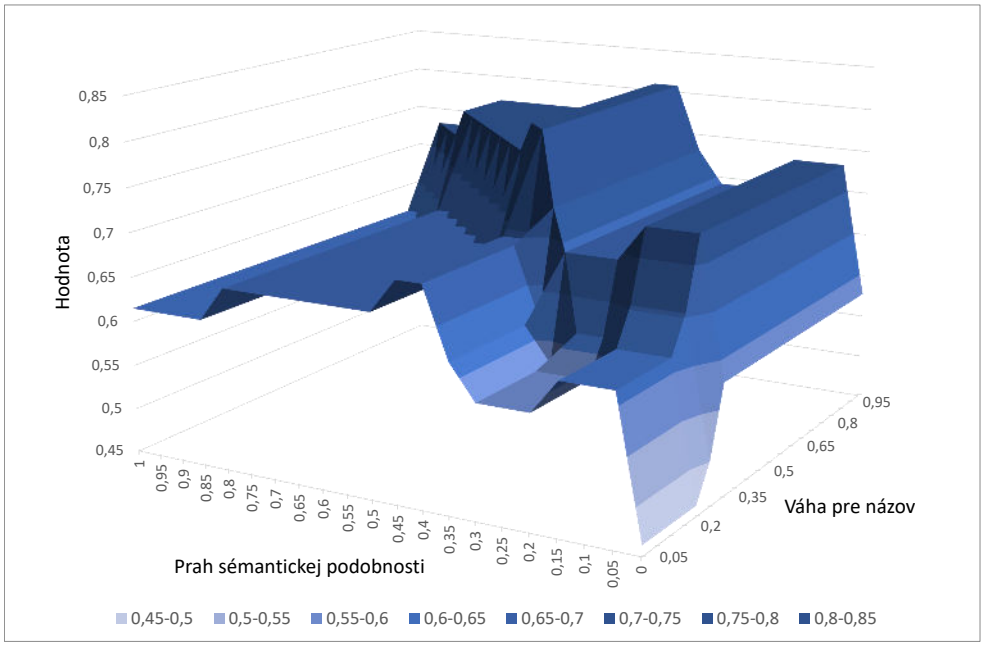
Obrázok 27: *Vyhodnotenie tretej iterácie – hodnoty pokrytia metódy pri experimentovaní s hodnotou váhy pre názov prvkov a hodnotou prahu sémantickej podobnosti.*



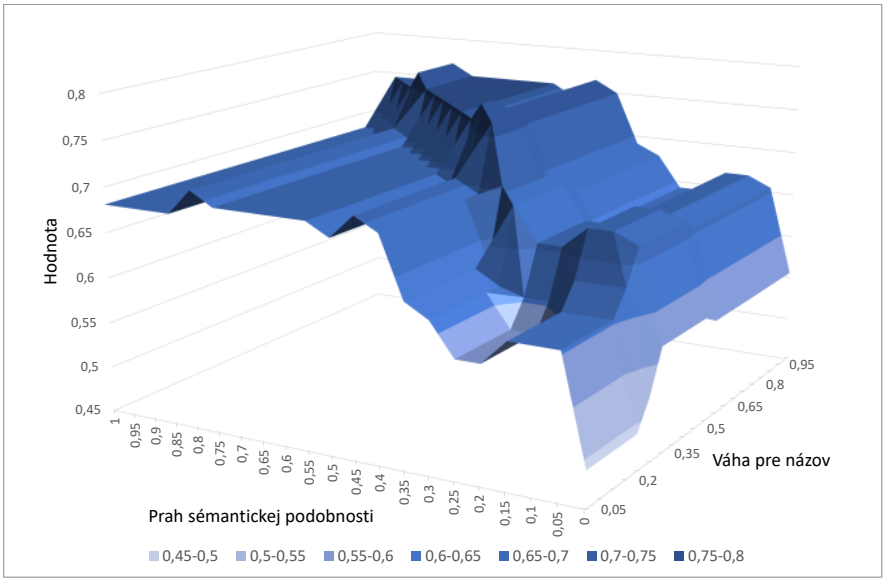
Obrázok 28: *Vyhodnotenie tretej iterácie – hodnoty F_1 metódy pri experimentovaní s hodnotou váhy pre názov prvkov a hodnotou prahu sémantickej podobnosti.*



Obrázok 29: *Overenie hypotézy - hodnoty presnosti metódy pri experimentovaní s hodnotou váhy pre názov prvkov a hodnotou prahu sémantickej podobnosti.*



Obrázok 30: *Overenie hypotézy - hodnoty pokrytia metódy pri experimentovaní s hodnotou váhy pre názov prvkov a hodnotou prahu sémantickej podobnosti.*



Obrázok 31: *Overenie hypotézy - hodnoty F_1 metódy pri experimentovaní s hodnotou váhy pre názov prvkov a hodnotou prahu sémantickej podobnosti.*

5.7 Porovnanie s existujúcimi metódami

V tabuľke 3 porovnávame implementovanú metódu s existujúcimi metódami, ktoré sme rozobrali v kapitole 3. Z tabuľky vyplýva fakt, že naša metóda ako jediná deteguje ontologické konflikty. Detekcia syntaktických konfliktov je súčasťou takmer všetkých metód. Naša metóda deteguje tie statické sémantické konflikty, ktoré spôsobujú neplatný zlúčený model.

Tabuľka 3: *Porovnanie detekcie konfliktov pomocou našej metódy a metód analyzovaných v kapitole 3. V ľavom stĺpci sa nachádzajú typy konfliktov podľa ich kategorizácie na obr. 1. Symbol „x“ znamená, že metóda deteguje daný typ konfliktov.*

	Rainbow	Grafy	SMoVer	Mirador	Naša metóda
syntaktické	x	x		x	x
statické sémantické	x		x		x
behavior. sémantické			x		
ontologické					x

6 Zhodnotenie

V tejto diplomovej práci sme sa zamerali na detekciu sémantických konfliktov v softvérových modeloch, ktoré môžu vzniknúť pri paralelnej práci členov vývojového tímu. Zaoberali sme sa prípadom, kde dvaja vývojári nezávisle od seba modifikujú verzie modelu odvodené od základnej verzie a v procese synchronizácie je nutné tieto odvodené verzie modelu zlúčiť. Práve v takom prípade je nevyhnutné identifikovať vykonané zmeny a detegovať konflikty, ktoré často vznikajú pri paralelnom vývoji.

Na začiatku práce sme na základe odbornej literatúry uviedli základnú kategorizáciu konfliktov. V ďalšej časti sme vykonali analýzu metód a nástrojov určených primárne na detekciu konfliktov. Zistili sme, že v tejto oblasti sa viacero prác venuje hlavne detekcii sémantických konfliktov, ktorá je náročnejšia v porovnaní s detekciou syntaktických konfliktov, pretože je potrebné uvažovať nielen syntax, ale aj sémantiku modelu. Nedostatky sme videli v detekcii statických sémantických konfliktov. Vykonaná analýza nám navyše ukázala, že žiadna metóda sa nevenuje detekcii ontologických konfliktov.

Hlavnou časťou práce je metóda na detekciu statických sémantických a ontologických konfliktov v UML modeli tried s využitím ontológií a mapovania ontológií. V metóde je najprv transformovaná základná verzia a odvodené verzie modelu do ontológie, ktorej schému sme definovali. Táto schéma využíva niektoré prvky UML metamodelu a pozostáva z konceptov a vlastností, ktoré reprezentujú najviac používané prvky UML modelu tried.

Kľúčovou časťou metódy je mapovanie získaných ontológií s cieľom určenia sémantických vzťahov medzi konceptami ontológií, kde na určenie hodnoty sémantickej podobnosti prvkov rôznych verzií modelu sémanticky porovnávame ich názvy a opisy. Nasleduje identifikácia vykonaných zmien v odvodených verziách modelu a detekcia konfliktov na základe stanovených podmienok.

Úspešnosť detekcie konfliktov navrhovanou metódou sme vyhodnotili pomocou vlastného datasetu UML modelov tried, ktorý pozostáva zo základnej verzie a z 2 odvodených verzií modelov. V odvodených verziách sme vykonali také konfliktné zmeny, ktoré vedú v procese zlúčenia verzií modelov k vzniku rôznych syntaktických a sémantických konfliktov. Vyhodnotenie úspešnosti navrhovanej metódy sme realizovali prostredníctvom štatistickej metódy F_1.

Vo vyhodnotení detekcie konfliktov sme zistili, že navrhovaná metóda dokáže detegovať 52,5 % všetkých konfliktov v datasete a 80,8 % konfliktov v datasete, na ktoré sa metóda zameriava. Maximálna hodnota presnosti detekcie konfliktov bola 77,3 %.

Z vykonaného vyhodnotenia detekcie konfliktov vyplýva tiež fakt, že na určenie hodnoty sémantickej podobnosti prvkov rôznych verzií modelu je optimálne zahrnúť názov a opis prvkov s rovnakou váhou. Pri takomto nastavení váh metóda dávala najlepšie výsledky detekcie konfliktov. Vyhodnotenie tiež ukázalo, že hodnota presnosti detekcie sa zvyšuje so zvyšujúcou sa hodnotou prahu sémantickej podobnosti.

Uvedenú úspešnosť detekcie konfliktov pomocou navrhovanej metódy je potrebné v budúcej práci zvýšiť. To môžeme dosiahnuť aplikovaním nasledujúcich vylepšení. Na určenie hodnoty sémantickej podobnosti prvkov rôznych verzií modelu môžeme použiť sémantické porovnanie nielen názvov a opisov prvkov, ale aj ich ďalších vlastností (napr. atribútov a operácií v prípade tried) a vzťahov s inými prvkami modelu.

Možným vylepšením metódy je tiež detekcia ďalších sémantických konfliktov (napr. konflikt presúvania atribútov pri refaktoringu a pridania tried do hierarchie dedenia, konflikt pridania ekvivalentných konceptov), ktoré môžu vzniknúť pri paralelnej práci. To zahŕňa definovanie podmienok, ktoré musia platiť pre ich detekciu, alebo iného procesu ich detekcie.

Medzi problémy, ktoré súvisia s detekciou konfliktov a ktorými je potrebné sa zaoberať v ďalšej práci, patrí vizualizácia a riešenie konfliktov. Správna vizualizácia konfliktov by mala byť prehľadná a mala by poskytovať čo najviac užitočných informácií potrebných na ich riešenie. Pre riešenie konfliktov je vhodné vylepšovať prístupy, ktoré automatizujú proces riešenia konfliktov.

Literatúra

[Altmanninger, 2008] Altmanninger, K.: *Models in Conflict – Towards a Semantically Enhanced Version Control System for Models*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 293–304.

[Altmanninger et al., 2007] Altmanninger, K.; Bergmayr, A.; Schwinger, W.: Semantically Enhanced Conflict Detection between Model Versions in SMoVer by Example. *Models in Software Engineering, Workshops and Symposia (MoDELS'07)*, 2007: s. 293–304.

[Altmanninger a Pierantonio, 2011] Altmanninger, K.; Pierantonio, A.: A categorization for conflicts in model versioning. *Elektrotechnik und Informationstechnik*, ročník 128, č. 11-12, 2011: s. 421–426.

[Altmanninger et al., 2009] Altmanninger, K.; Seidl, M.; Wimmer, M.: A survey on model versioning approaches. *International Journal of Web Information Systems*, ročník 5, č. 3, 2009: s. 271–304.

[Barrett et al., 2011] Barrett, S.; Chalin, P.; Butler, G.: Table-Driven Detection and Resolution of Operation-Based Merge Conflicts with Mirador. In *Modelling Foundations and Applications - 7th European Conference, ECMFA 2011, Birmingham, UK, June 6 - 9, 2011 Proceedings*, 2011, s. 329–344.

[Bisztray et al., 2008] Bisztray, D.; Heckel, R.; Ehrig, H.: Verification of Architectural Refactorings: Rule Extraction and Tool Support. *ECEASST*, ročník 16, 2008.

[Brosch et al., 2011] Brosch, P.; Kargl, H.; Langer, P.; et al.: Conflicts as First-Class Entities: A UML Profile for Model Versioning. *Models in Software Engineering - Workshops and Symposia at MODELS 2010, Reports and Revised Selected Papers*, ročník 6627, 2011: s. 184–193.

[Brosch et al., 2012] Brosch, P.; Seidl, M.; Wimmer, M.; et al.: Conflict Visualization for Evolving UML Models. *Journal of Object Technology*, ročník 11, č. 3, 2012: s. 2:1–30.

[Castano et al., 2001] Castano, S.; De Antonellis, V.; De Capitani di Vimercati, S.: Global Viewing of Heterogeneous Data Sources. *IEEE Transactions on Knowledge and Data Engineering*, ročník 13, č. 2, Marec 2001: s. 277–297, ISSN 1041-4347.

[Chong et al., 2016] Chong, H.; Zhang, R.; Qin, Z.: Composite-based Conflict Resolution in Merging Versions of UML Models. In *Proceedings of the 17th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD, Shanghai, China, May 30–June 1, 2016.*, 2016, s. 127–132.

[Costa et al., 2013] Costa, V. O.; Junior, J. M. B. O.; Murta, L. G. P.: Semantic Conflicts Detection in Model-driven Engineering. In *The 25th International Conference on Software Engineering and Knowledge Engineering, Boston, MA, USA, June 27-29, 2013.*, 2013, s. 656–661.

[Costa et al., 2014] Costa, V. O.; Monteiro, R.; Murta, L.: Detecting Semantic Equivalence in UML Class Diagrams. In *The 26th International Conference on Software Engineering and Knowledge Engineering, Hyatt Regency, Vancouver, BC, Canada, July 1-3, 2013.*, 2014, s. 318–323.

[Do a Rahm, 2002] Do, H.-H.; Rahm, E.: COMA: A System for Flexible Combination of Schema Matching Approaches. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, VLDB Endowment, 2002, s. 610–621.

[Ehrig et al., 2006] Ehrig, H.; Ehrig, K.; Prange, U.; et al.: *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, ISBN 3540311874.

[Ehrig a Sure, 2004] Ehrig, M.; Sure, Y.: Ontology Mapping - An Integrated Approach. In *Proceedings of the First European Semantic Web Symposium*, *Lecture Notes in Computer Science*, ročník 3053, editácia C. Bussler; J. Davis; D. Fensel; R. Studer, Heraklion, Greece: Springer Verlag, Máj 2004, s. 76–91.

[Habel a Pennemann, 2009] Habel, A.; Pennemann, K.-H.: Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, ročník 19, č. 2, 2009: s. 245–296.

[Hooi et al., 2014] Hooi, Y. K.; Hassan, M. F.; Shariff, A. M.: *A Survey on Ontology Mapping Techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, ISBN 978-3-642-41674-3, s. 829–836.

[Koegel et al., 2010] Koegel, M.; Herrmannsdoerfer, M.; von Wesendonk, O.; et al.: Operation-based Conflict Detection. In *Proceedings of the 1st International Workshop on Model Comparison in Practice*, IWMCP '10, 2010,

s. 21–30.

[Ram a Park, 2004] Ram, S.; Park, J.: Semantic Conflict Resolution Ontology (SCROL): An Ontology for Detecting and Resolving Data and Schema-Level Semantic Conflicts. *IEEE Transactions on Knowledge and Data Engineering*, ročník 16, č. 2, 2004: s. 189–202.

[Reggio et al., 2013] Reggio, G.; Leotta, M.; Ricca, F.; et al.: What are the used UML diagrams? A Preliminary Survey. In *Proceedings of the 3rd International Workshop on Experiences and Empirical Studies in Software Modeling co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013), Miami, USA, October 1, 2013.*, 2013, s. 3–12.

[Reiter et al., 2007] Reiter, T.; Altmanninger, K.; Bergmayr, A.; et al.: Models in Conflict – Detection of Semantic Conflicts in Model-based Development. In *Proceedings of the 3rd International Workshop on Model-Driven Enterprise Information Systems - MDEIS 2007; In Conjunction with ICEIS 2007*, 2007, s. 29–40.

[Täntzer et al., 2010] Täntzer, G.; Ermel, C.; Langer, P.; et al.: Conflict Detection for Model Versioning Based on Graph Modifications. In *Proceedings of the Fifth International Conference on Graph Transformations*, ročník 6372, editácia H. Ehrig; A. Rensink; G. Rozenberg; A. Schür, Springer International Publishing, 2010, s. 171–186.

[W. Cohen et al., 2003] W. Cohen, W.; Ravikumar, P.; E. Fienberg, S.: A Comparison of String Metrics for Matching Names and Records. *KDD Workshop on Data Cleaning and Object Consolidation*, ročník 3, 2003: s. 73–78.

Príloha A: Vyhodnotenie plánu práce

Príloha A obsahuje vyhodnotenie plánu práce na projekte počas celej doby jeho riešenia. V tabuľke A-1 sa nachádza plán práce na projekte v zimnom semestri počas DP II. Zameranie práce sa zmenilo z detekcie a vizualizácie konfliktov primárne na detekciu sémantických konfliktov v softvérových modeloch.

Tabuľka A-1: *Plán práce na projekte v zimnom semestri 2017/2018 (DP II).*

Činnosť	Týždeň semestra
podrobný návrh metódy na detekciu konfliktov	1.-4.
implementácia spracovania verzií modelu	5.-7.
implementácia metódy na detekciu konfliktov	8.-12.

V zimnom semestri sme sa vo významnej miere sústredili na podrobný návrh metódy pre detekciu statických sémantických a ontologických konfliktov. Pri návrhu metódy sa vyskytlo viacero problémov a otázok, ktoré sme museli vyriešiť a zabrali veľa času. Pre tieto okolnosti sa nám nepodarilo začať s implementáciou metódy. Preto sme v medzisemestrálnom období pracovali najmä na implementácii navrhovanej metódy na detekciu konfliktov.

V tabuľke A-2 sa nachádza plán práce na projekte v letnom semestri počas DP III. Keďže sme v medzisemestrálnom období úspešne implementovali veľkú časť metódy na detekciu konfliktov, v tejto etape riešenia projektu sa nám podarilo splniť stanovený plán v daných časových bodoch.

Tabuľka A-2: *Plán práce na projekte v letnom semestri 2017/2018 (DP III).*

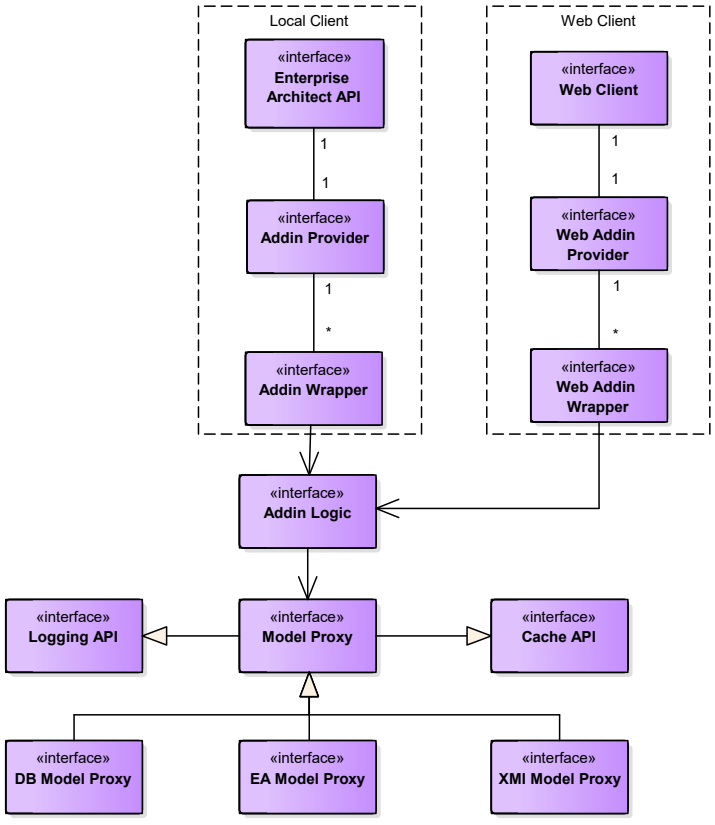
Činnosť	Týždeň semestra
dokončenie implementácie metódy na detekciu konfliktov	1.-3.
vytvorenie testovacej vzorky UML modelov	4.-5.
vyhodnotenie úspešnosti metódy	6.-9.
finalizácia práce	10.-12.

Príloha B: Technická dokumentácia

Táto príloha obsahuje opis architektúry implementovaného prototypu, jeho dátový model a ukážky implementácie niektorých dôležitých funkcií.

B1: Architektúra

Na obr. B-1 je zobrazená architektúra implementovaného prototypu, ktorú navrhol a implementoval Bc. Jakub Ondik v rámci semestrálneho projektu na predmete Výskumná projektová práca v zimnom semestri 2017/2018. Náš prototyp využíva v rámci tejto architektúry komponenty webového klienta, *Addin Logic* a *XMI Model Proxy*.



Obrázok B-1: *Architektúra prototypu implementovaná Bc. Jakubom Ondikom v rámci semestrálneho projektu na predmete Výskumná projektová práca v zimnom semestri 2017/2018.*

Komponent *Web client* zabezpečuje interakciu s používateľom a možnosť manipulácie s modelmi prostredníctvom webového prehliadača. Komunikuje s komponentom *Web Addin Provider*, ktorý priamo manipuluje s modelom bez potreby ďalšieho API. Komponent *Web Addin Wrapper* slúži ako obal pre dynamické rozšírenia a reprezentuje proxy rozšírenia.

Hlavným komponentom je komponent *Addin Logic*, ktorý musia všetky dynamické rozšírenia implementovať. V rámci tohto komponentu sme implementovali hlavnú časť nášho prototypu. *Addin Logic* používa komponent *XMI Model Proxy* na sprístupnenie všetkých prvkov jednotlivých verzií modelu, ktoré sú načítané vo forme súborov s koncovkou .xmi.

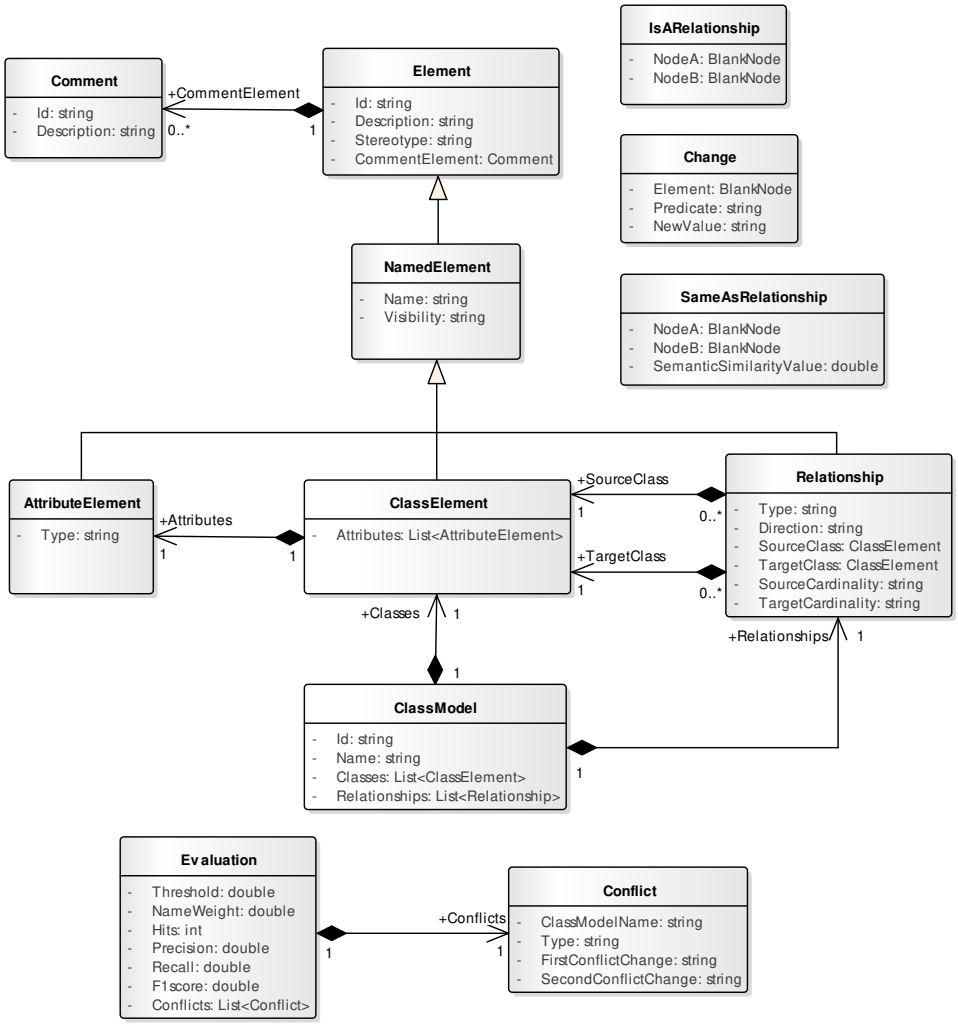
B2: Dátový model

Dátový model prototypu je zobrazený na obr. B-2. Trieda *ClassModel* slúži na reprezentáciu každej verzie modelu pred jej transformáciou do ontológie. Obsahuje identifikátor, názov, všetky triedy a vzťahy danej verzie modelu.

Trieda *ClassElement* reprezentuje triedu modelu, ktorá obsahuje zoznam atribútov definovaných triedou *AttributeElement*. Trieda *Relationship* slúži na uchovanie všetkých vlastností vzťahu v modeli. Ďalšou triedou je trieda *Comment*, ktorá predstavuje komentár priradený ku každému prvku modelu. V dátovom modeli prototypu sa nachádzajú tiež triedy *Element* a *NamedElement*, ktoré sme použili z UML metamodelu a obsahujú atribúty spoločné pre triedy *ClassElement*, *AttributeElement* a *Relationship*.

Triedy *SameAsRelationship* a *IsARelationship* slúžia na uloženie konceptov ontológií, medzi ktorými je *same_as* vzťah, resp. *is_a* vzťah. Jednotlivé koncepty sú reprezentované uzlom typu BlankNode. Trieda *Change* reprezentuje zmenu vykonanú v odvodenej verzii modelu a obsahuje atribút *Element* označujúci koncept v ontológii, ktorého sa zmena týka. V prípade úpravy prvku modelu sa používajú aj ďalšie atribúty *Predicate* a *NewValue*, ktoré určujú zmenenú vlastnosť a jej novú hodnotu.

Vyhodnotenie detekcie metódy je uchovávané v triede *Evaluation*, ktorá obsahuje hodnoty prahu, váhy pre názov, presnosti, pokrytia, F_1, počet správne detegovaných konfliktov a zoznam všetkých detegovaných konfliktov. Konflikt je reprezentovaný triedou *Conflict*, v ktorej ukladáme názov modelu, typ konfliktu a konfliktné zmeny vykonané v odvodených verziách modelu.



Obrázok B-2: *Dátový model prototypu.*

B3: Inštalačná príručka

Implementovaný softvérový prototyp je potrebné iba nasadiť na server. Postup pre nasadenie[1] na rôzne platformy a operačné systémy je uvedený v dokumentácii ASP.NET.

[1] https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/?view=aspnetcore-2.1tabs=aspnetcore2x

B4: Ukážka kódu

V tejto prílohe uvádzame niektoré implementované funkcie. Zdrojový kód prototypu je umiestnený na priloženom elektronickom médiu.

```
public List<string> GetSynonyms(List<Triple> triples)
{
    IGraph graph = new Graph();
    ILiteralNode hasName = graph.CreateLiteralNode("hasName");
    List<string> synonyms = new List<string>();

    foreach (Triple t in triples.WithPredicate(hasName))
    {
        string name = t.Object.ToString().ToLower();
        List<SynSet> synsets = wordNetEngine.GetSynSets(name);

        foreach (SynSet synset in synsets)
        {
            if (synset.PartOfSpeech.GetHashCode() == 1)
            {
                foreach (string word in synset.Words)
                {
                    if (!synonyms.Contains(word) && !name.Equals(word))
                    {
                        synonyms.Add(word);
                    }
                }
            }
        }
    }
    return synonyms;
}
```

Ukážka 6: *Funkcia, ktorá nájde synonymá pre názov prvku modelu vo WordNete.*

```
public double CompareDescriptions(List<Triple> nodeA, List<Triple> nodeB)
{
    IGraph graph = new Graph();
    ILiteralNode hasDescription = graph.CreateLiteralNode("hasDescription");
    foreach (Triple tA in nodeA.WithPredicate(hasDescription))
    {
        foreach (Triple tB in nodeB.WithPredicate(hasDescription))
        {
            JaroWinkler jw = new JaroWinkler();
            return jw.GetSimilarity(tA.Object.ToString(),tB.Object.ToString());
        }
    }
    return 0.0;
}
```

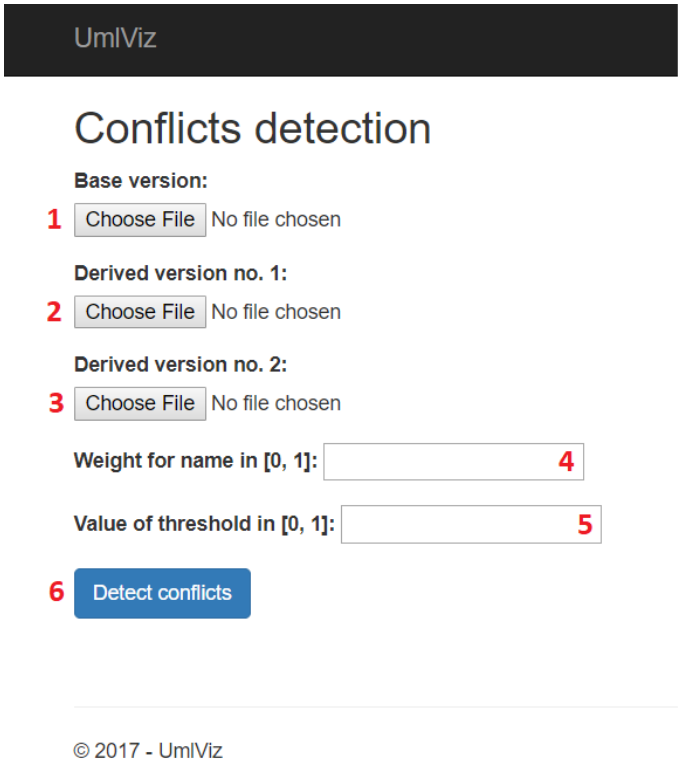
Ukážka 7: *Funkcia, ktorá vypočíta podobnosť opisov prvkov modelu pomocou Jaro-Winklerovho algoritmu.*

Príloha C: Používateľská príručka

Táto príloha obsahuje príručku pre používateľov nášho prototypu zameraného na detekciu konfliktov v softvérových modeloch. K prototypu je možné pristupovať na webovej adrese https://umlviz.fiit.stuba.sk/ConflictAddin.

Vstupmi prototypu sú základná verzia a 2 odvodené verzie modelu tried vo forme troch súborov s koncovkou .xmi. Tieto súbory exportujte z nástroja Enterprise Architect vo formáte XMI 2.4.2. Do prototypu vložte základnú verziu modelu pomocou tlačidla č. 1 a odvodené verzie modelu pomocou tlačidiel č. 2 a č. 3 na obr. C-1.

Následne vyplňte v prototype hodnotu váhy pre názov prvkov modelu v intervale <0, 1> do textového poľa označeného č. 4 a hodnotu prahu sémantickej podobnosti v intervale <0, 1> do textového poľa označeného č. 5. Po vložení súborov a vyplnení uvedených hodnôt kliknite na tlačidlo *Detect conflicts*, ktoré je označené č. 6 na obr. C-1.



Obrázok C-1: *Úvodná obrazovka prototypu.*

Po pár sekundách sa nižšie zobrazia výsledky detekcie konfliktov v tabuľke, ktorú je možné vidieť na obr. C-2 a detailnejšie na obr. C-3. Nad tabuľkou sú uvedené zadané hodnoty váhy pre názov prvkov modelu a prahu sémantickej podobnosti. Tabuľka obsahuje 4 stĺpce, v ktorých sú postupne uvedené názov modelu, v ktorom vznikol konflikt, typ konfliktu a konfliktné zmeny vykonané v odvodených verziách modelu, ktoré spôsobili konflikt.

Conflicts

Weight for name: 0,3

Value of threshold: 0,5

Class Model	Type of conflict	Conflict change in the derived version no. 1	Conflict change in the derived version no. 2
Class Model 01	Similar elements	addition of attribute foundDefect in class ComplaintProtocol with description "Defect of product that was found by customer."	addition of attribute foundIssue in class ComplaintProtocol with description "Found issue of product in complaint."
Class Model 01	Similar elements	addition of attribute price in class Component with description "Sale price of component."	addition of attribute cost in class Component with description "Selling cost of component."
Class Model 01	Similar elements	addition of attribute quantity in class Component with description "Quantity of component."	addition of attribute quality in class Component with description "Quality of component."
Class Model 01	Similar elements	addition of attribute description in class TestResults	addition of attribute decsription in class TestResults
Class Model 01	Similar elements	addition of association fills between classes Order and Customer	addition of association places between classes Order and Customer
Class Model 01	Update-Update	class Order: change of description to "Order contains list of products that customer wants to buy."	class Order: change of description to "Order represents products list and other information about order."
Class Model 01	Update-Update	attribute date in class Order: change of name to "creationDate"	attribute date in class Order: change of name to "filingDate"

Obrázok C-2: *Tabuľka obsahujúca detegované konflikty pomocou nášho prototypu.*

Conflict change in the derived version no. 1	Conflict change in the derived version no. 2
addition of attribute foundDefect in class ComplaintProtocol with description "Defect of product that was found by customer."	addition of attribute foundIssue in class ComplaintProtocol with description "Found issue of product in complaint."
addition of attribute price in class Component with description "Sale price of component."	addition of attribute cost in class Component with description "Selling cost of component."

Obrázok C-3: *Detail tabuľky, ktorá obsahuje detegované konflikty pomocou nášho prototypu. V stĺpcoch tabuľky sú v špecifickom formáte uvedené konfliktné zmeny v odvodených verziách modelu, ktoré spôsobili konflikt.*

Príloha D: Príspevok na IIT.SRC 2018

Obsahom tejto prílohy je príspevok publikovaný na konferencii IIT.SRC 2018.

Semantic Conflict Detection in Software Models

Martin OLEJÁR*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
martin.olejar.7@gmail.com

Abstract. Software system development includes creation of different model versions that continuously undergo significant changes. In the merge process, it is necessary to identify changes between the model versions and to detect conflicts that often occur during the parallel work of team members. Nowadays, one of the main problems is semantic conflict detection which should take into consideration the semantics of model too. Their correct detection is a good prerequisite for their resolution and prevention of defects in models. In the paper, we present a semantic conflict detection method in UML class models with the utilization of ontologies and ontology mappings.

1 Introduction

Modeling is very important activity during whole software system development. Software models provide not only some abstraction levels of system, but also the basis for its implementation. Teams involve many people participating in software system development what often encounter parallel work of team members with the aim of effective use of time and resources.

During the parallel work of team members, different model versions are created – the base version that is stored in the shared repository and model versions that are derived from the base version and that are modified by team members in parallel. In a merge process, these derived model versions need to be merged to apply all changes into common model version.

Such the merge process includes solution of many problems. The first one is an identification of changes that have been made in the derived model versions. In practice, mainly operation-based and state-based approaches are used [10]. Operation-based approaches store model changes immediately after their application. On the other side, state-based approaches compare the state of the base version with states of final derived versions to identify differences between them.

The second problem is detection of conflicts that are often caused by the parallel work of team members. A conflict can be generally defined as a set of contradicting changes where at least one change does not commute with at least one change applied by another developer [3]. There are syntactic conflicts that are related to the model structure and semantic conflicts that are related to the model meaning.

Syntactic conflicts can be detected by structural comparison of model versions [1]. Typical example of syntactic conflict is a change of the same element property in the model versions in a contradicting way. The representation of these conflicts is a tree or a graph which reflects the syntax of used language.

Semantic conflict detection is quite complicated because understanding the model semantics is needed. Semantic conflicts occur when elements that are changed in the model versions depend on each other or are interrelated semantically [3]. They can occur due to contradicting or same intentions of developers.

The paper is structured as follows. We offer a brief overview of existing semantic conflict detection methods in Section 2. Section 3 introduces our proposed method for semantic conflict detection. Actual evaluation can be found in Section 4.

2 Related work

Many researchers have dealt with the development of methods for syntactic conflict detection [4, 13]. On the other side, there are some methods focused on semantic conflict detection, as described below.

Method called Rainbow is a semantic conflict detection method for UML use case and class diagrams [6]. This method is composed of 3 phases: translation, semantic enrichment and conflict detection. In the translation phase, the method transforms the base version and the derived versions into a set of Prolog facts that represent elements and relationships

* Master study programme in field: Software Engineering
Supervisor: Dr. Karol Rástočný, Institute of Informatics, Information Systems and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

IIT.SRC 2018, Bratislava, April 18, 2018, pp. 197–200.

in the diagrams and comprise syntactic information. In the semantic enrichment phase, the method infers indirect relationships from the generated Prolog facts based on a set of rules that describe the semantics of the diagrams and adds new Prolog facts that represent semantic information. In the conflict detection phase, the additions and deletions made in the derived versions are identified and checked for conflicts. A conflict occurs if a model element appears in the set of additions of one derived version and in the set of deletions of another derived version.

SMoVer (Semantically enhanced Model Version Control System) detects static and behavioral semantic conflicts based on update strategies and semantic view definitions [2]. There are 4 update strategies - attribute update, referenced element update, reference update and role update - that are used for detection of structural changes. The semantic view definition consists of 2 parts. The first one is a view definition metamodel that defines the abstract syntax. The second one is a model transformation that defines a semantic mapping between the source and target metamodel and creates another model that corresponds to the metamodel representing the semantic view definition. Thanks to the semantic view definition, conflict detection process can be done both in the syntax and semantic views using the structural comparison. A semantic conflict is detected between model versions that have been transformed in the semantic view.

SCROL (Semantic Conflict Resolution Ontology) provides a method for automatically detecting and resolving data level and schema level conflicts in heterogeneous databases [11]. Its structure is a tree that consists of 5 components - a set of concepts, a set of instances, a set of sibling relationships between 2 concepts, a set of domain value mapping relationships between 2 instances and the root. Conflict detection process consists of 3 activities. Firstly, 3 information from SCROL are identified - a conflict controller that represents multiple interpretations of semantically related objects, an original and target context of components. Secondly, ontology relationship knowledge between SCROL components is constructed to determine whether 2 terms are semantically related. Finally, the method defines mapping knowledge between SCROL and database schemes. It has been used for land-use, ecology and publication databases.

3 Method for semantic conflict detection

Based on the analysis of existing conflict detection methods, our proposed method is focused on the detection of static semantic and ontological conflicts (defined bellow) with the utilization of ontologies and ontology mappings. We propose this method for UML class model that is the most used UML model in software projects, books, tools, courses and tutorials [12].

Static semantic conflicts result to invalid merged model version. For instance, addition of generalizations into the derived model versions can create inheritance hierarchy cycle in the merged model version [3]. Ontological conflicts are conflicts that occur by using lexically different terms that semantically interfere with each other or denote the same object. Typical example of ontological conflict is addition of new attributes into the same class in the derived model versions that have names in a synonymous relation.

The inputs of our method for semantic conflict detection are the base model version and derived model versions modified in parallel. The method consists of the following steps that are described in the next subsections in more detail:

1. transformation of model versions into ontology,
2. ontology mapping between model versions,
3. identification of changes in the derived versions,
4. conflict detection process.

3.1 Model transformation into ontology

In the first step of the method, the base model version and derived model versions are transformed into ontologies. A scheme of our proposed ontology that defines the most used concepts of UML class model and their properties is shown in Figure 1.

In the transformation process, each element of UML class model is transformed to its corresponding concept in the ontology with its properties. The outputs of this step are three ontologies that represent the base model version and derived model versions and are used for the ontology mapping.

3.2 Ontology mapping

Mapping one ontology with another means that for each concept in one ontology we try to find a corresponding concept in another ontology that has the same or semantically similar meaning [8]. The outputs of the mapping are pairs of concepts in the ontologies that are most similar in a semantic way. Moreover, it enables us to identify changes made in the derived model versions. The proposed method consists of 2 ontology mappings:

- mapping the ontology representing the base model version with the ontologies representing the derived model versions,
- mapping the ontologies representing the derived model versions with each other.

Our algorithm for ontology mapping has been proposed by a combination of existing basic methods [9] that are based on string comparison and advanced methods ARTEMIS [5] and COMA [7].

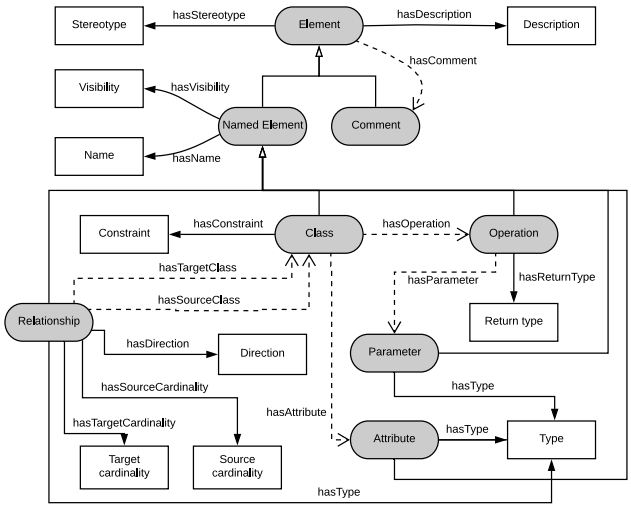


Figure 1. Scheme of the ontology for UML class model. The oval gray shapes represent concepts and the white rectangles represent their properties. The dashed lines between concepts represent relationships between them and the solid lines represent assignment of properties to concepts. We have used elements Element and NamedElement defined in UML metamodel.

In our algorithm, concept names, relationships between concepts and other concept properties are semantically compared. Results of this semantic comparison are combined and semantic similarity values are calculated that takes into account both terminological and structural aspects of the model elements. Semantic similarity values between concepts belong to the interval of [0, 1] where value 0 determines no similarity between concepts and value 1 determines equivalence between concepts.

For the comparison of concept names, we use a large lexical database of English called WordNet[1] that groups nouns, adjectives, verbs and adverbs into sets of cognitive synonyms called synsets. Moreover, WordNet enables us to check many relations among English words. The most important relations are synonymy, hyperonymy (super-subordinate relation) and meronymy (part-whole relation) that can represent the most used relationships in UML class model.

We have defined weights for the mentioned relations in the interval of (0, 1). The relation of synonymy has the biggest weight from the mentioned relations because concepts are semantically most similar in that case. The semantic similarity value is equal to 1 if concept names are identical. If there is none of the mentioned relations among concept names and concept names are not identical, then this value is equal to calculated Levenshtein distance between them.

The comparison of relationships between concepts comes after the comparison of concept names. Our algorithm compares relationships between pairs of semantically similar classes and checks if attributes or operations belong to the semantically similar classes in the model versions. For instance, the semantic similarity value of attributes that have identical names but belong to semantically different classes is equal to 0. Other concept properties are compared for identification of changes made in the derived model versions.

3.3 Change identification

Based on pairs of the most similar concepts in the ontologies, we can identify changes that have been made in the derived model versions. It is a case of state-based approach for change identification in the model versions. There are 3 types of changes that can be made: addition, property change and removal of a model element or a concept in the ontology.

We have determined following conditions that helps us to identify these types of changes. Firstly, a model element is added to the derived model version or removed from the derived model version if there is no semantically similar element found in the ontology mapping process. Secondly, property change of a model element in the derived model version can be identified by property comparison.

[1] https://wordnet.princeton.edu/

3.4 Conflict detection

Final step of our method compares the sets of changes made in the derived model versions and identifies contradicting conflict changes.

Our method detects static semantic conflicts by application of all changes in the derived model versions into one merged version and checking conflict occurrences in the merged version. For example, inheritance hierarchy cycle is detected using a recursion starting from each class in the merged version.

Ontological conflicts are detected based on predefined rules. Such an example rule is defined as follows. If there has been added new attributes to the semantically similar classes in the derived model versions that have names in a synonymous relation, conflict occurs because these attributes may denote the same object.

4 Evaluation

For the evaluation of our conflict detection method, we have created a dataset of UML class models that contains sets of 3 model versions - the base version and 2 derived versions. We have made such conflict changes in the derived model versions that lead to occurrence of static semantic and ontological conflicts in the merge process. The proposed method has been evaluated using the statistical method F_1 measure.

In Figure 2, we see evaluation of the method using our dataset. We experimented with value of threshold in the interval of [0, 1]. This threshold represents minimal semantic similarity value that must be between concept names in the ontology mapping phase in order that these concepts can be semantically similar.

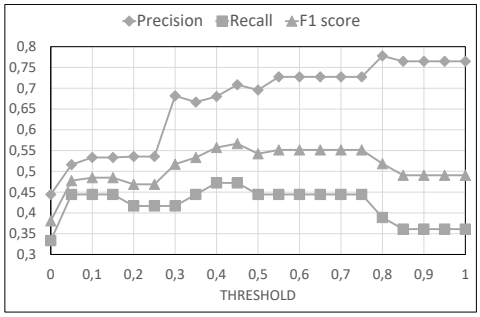


Figure 2. Precision, recall and F_1 score of conflict detection method.

5 Conclusions

Main goal of this paper was to introduce a method focused on detection of static semantic and ontological conflicts. The method uses ontologies and ontology mappings, large lexical database of English words called WordNet, Levenshtein distance and defines rules for conflicts detection.

References

[1] Altmanninger, K. In: *Models in Conflict – Towards a Semantically Enhanced Version Control System for Models*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 293–304.

[2] Altmanninger, K., Bergmayr, A., Schwinger, W.: Semantically Enhanced Conflict Detection between Model Versions in SMoVer by Example. *Models in SW Eng., Workshops and Symposia (MoDELS'07)*, 2007, pp. 293–304.

[3] Altmanninger, K., Pierantonio, A.: A categorization for conflicts in model versioning. *Elektrotechnik und Informationstechnik*, 2011, vol. 128, no. 11-12, pp. 421–426.

[4] Barrett, S., Chalin, P., Butler, G.: Table-Driven Detection and Resolution of Operation-Based Merge Conflicts with Mirador. In: *7th European Conf. on Modelling Foundations and Applications, (ECMFA)*, 2011, pp. 329–344.

[5] Castano, S., De Antonellis, V., De Capitani di Vimercati, S.: Global Viewing of Heterogeneous Data Sources. *IEEE Trans. on Knowl. and Data Eng.*, 2001, vol. 13, no. 2, pp. 277–297.

[6] Costa, V.O., Monteiro, R., Murta, L.: Detecting Semantic Equivalence in UML Class Diagrams. In: *The 26th Int. Conf. on SW Eng. and Knowledge Eng.*, 2014, pp. 318–323.

[7] Do, H.H., Rahm, E.: COMA: A System for Flexible Combination of Schema Matching Approaches. In: *Proc. of the 28th Int. Conf. on Very Large Data Bases*. VLDB '02, VLDB Endowment, 2002, pp. 610–621.

[8] Ehrig, M., Sure, Y.: Ontology Mapping - An Integrated Approach. In Bussler, C., Davis, J., Fensel, D., Studer, R., eds.: *Proc. of the 1st European Semantic Web Symposium*. Volume 3053 of *Lecture Notes in Computer Science*., Springer Verlag, 2004, pp. 76–91.

[9] Hooi, Y.K., Hassan, M.F., Shariff, A.M. In: *A Survey on Ontology Mapping Techniques*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 829–836.

[10] Koegel, M., Herrmannsdoerfer, M., von Wesendonk, O., Helming, J.: Operation-based Conflict Detection. In: *Proc. of the 1st Int. Workshop on Model Comparison in Practice*. IWMCP '10, 2010, pp. 21–30.

[11] Ram, S., Park, J.: Semantic Conflict Resolution Ontology (SCROL): An Ontology for Detecting and Resolving Data and Schema-Level Semantic Conflicts. *IEEE Transactions on Knowledge and Data Eng.*, 2004, vol. 16, no. 2, pp. 189–202.

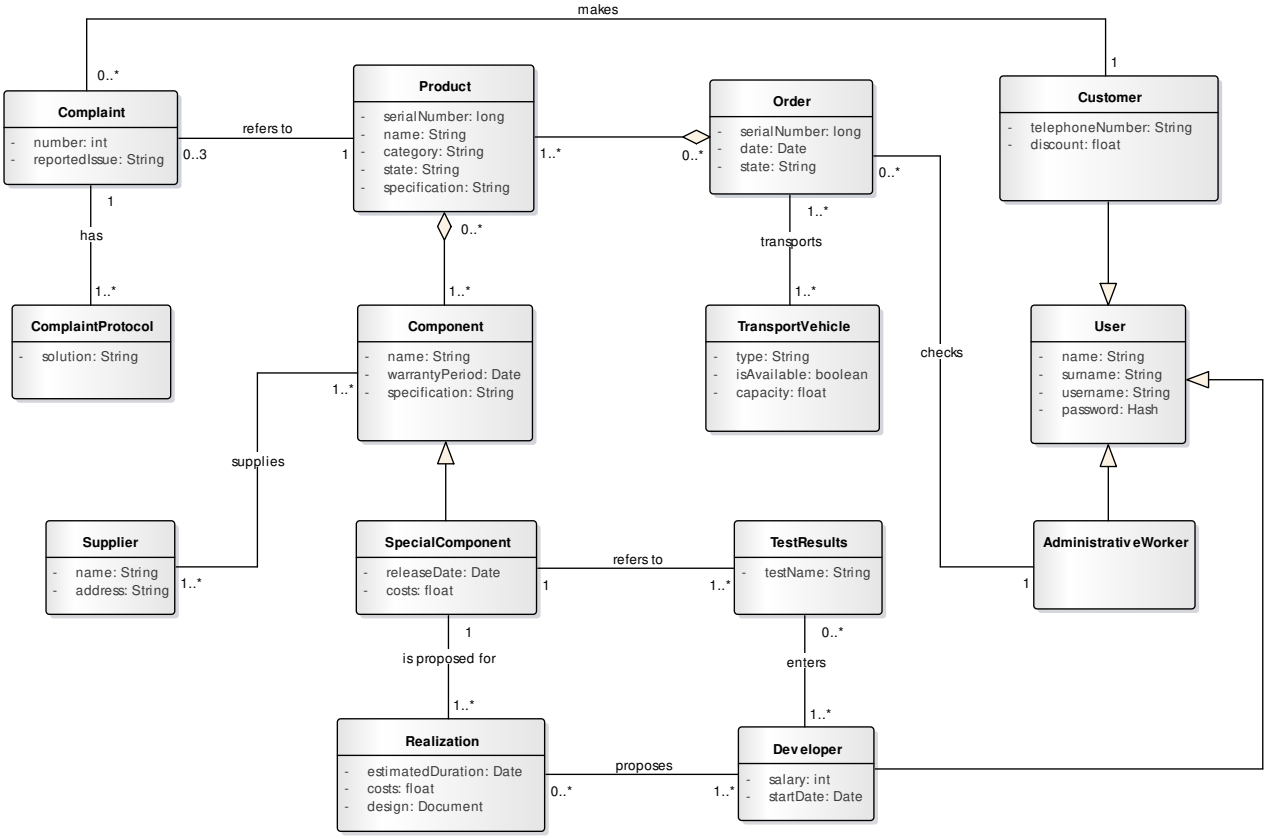
[12] Reggio, G., Leotta, M., Ricca, F., Clerissi, D.: What are the used UML diagrams? A Preliminary Survey. In: *Proc. of the 3rd Int. Workshop on Experiences and Empirical Studies in SW Modeling co-located with 16th Int. Conf. on Model Driven Eng. Languages and Systems (MoDELS 2013)*, 2013, pp. 3–12.

[13] Täntzer, G., Ermel, C., Langer, P., Wimmer, M.: Conflict Detection for Model Versioning Based on Graph Modifications. In Ehrig, H., Rensink, A., Rozenberg, G., Schür, A., eds.: *Proc. of the 5th Int. Conf. on Graph Transformations*. Volume 6372., Springer International Publishing, 2010, pp. 171–186.

Príloha E: Dokumentácia datasetu modelov

Táto príloha obsahuje dokumentáciu datasetu UML modelov tried vygenerovanú z nástroja Enterprise Architect. Tento dataset pozostáva z 3 XMI súborov, ktorých súčasťou sú základná verzia a 2 odvodené verzie piatich UML modelov tried. Pri každej verzii modelu tried je uvedený jeho diagram spolu s názvom a opisom všetkých tried. Pre každý model tried uvádzame zoznam zmien a konfliktov spolu s konfliktnými zmenami, ktoré boli v odvodených verziách modelu vykonané a spôsobujú jednotlivé konflikty. Niektoré konflikty v modeloch sme prevzali z vedeckých článkov (Altmanninger a Pierantonio, 2011), (Altmanninger et al., 2009), (Brosch et al., 2011) a (Chong et al., 2016).

Class Model 01 – základná verzia



AdministrativeWorker

Administrative worker deals with current business related to administration of order.

Complaint

Complaint is added to system after reporting of issue by customer.

ComplaintProtocol

Complaint protocol is document that is brought out after solution of complaint.

Component

Component represents part from which product is composed of.

Customer

Customer is person that creates order.

Developer

Developer is responsible for proposal of development realization and administration of test results.

Order

Order is list of products that customer wants to buy.

Product

Product represents whole that is composed of at least one component.

Realization

Realization is method that is used for development of new component or software.

SpecialComponent

Special component is part of product that is developed by request of customer.

Supplier

Supplier supplies components that company offers.

TestResults

Results of test are outputs of test that was done during development.

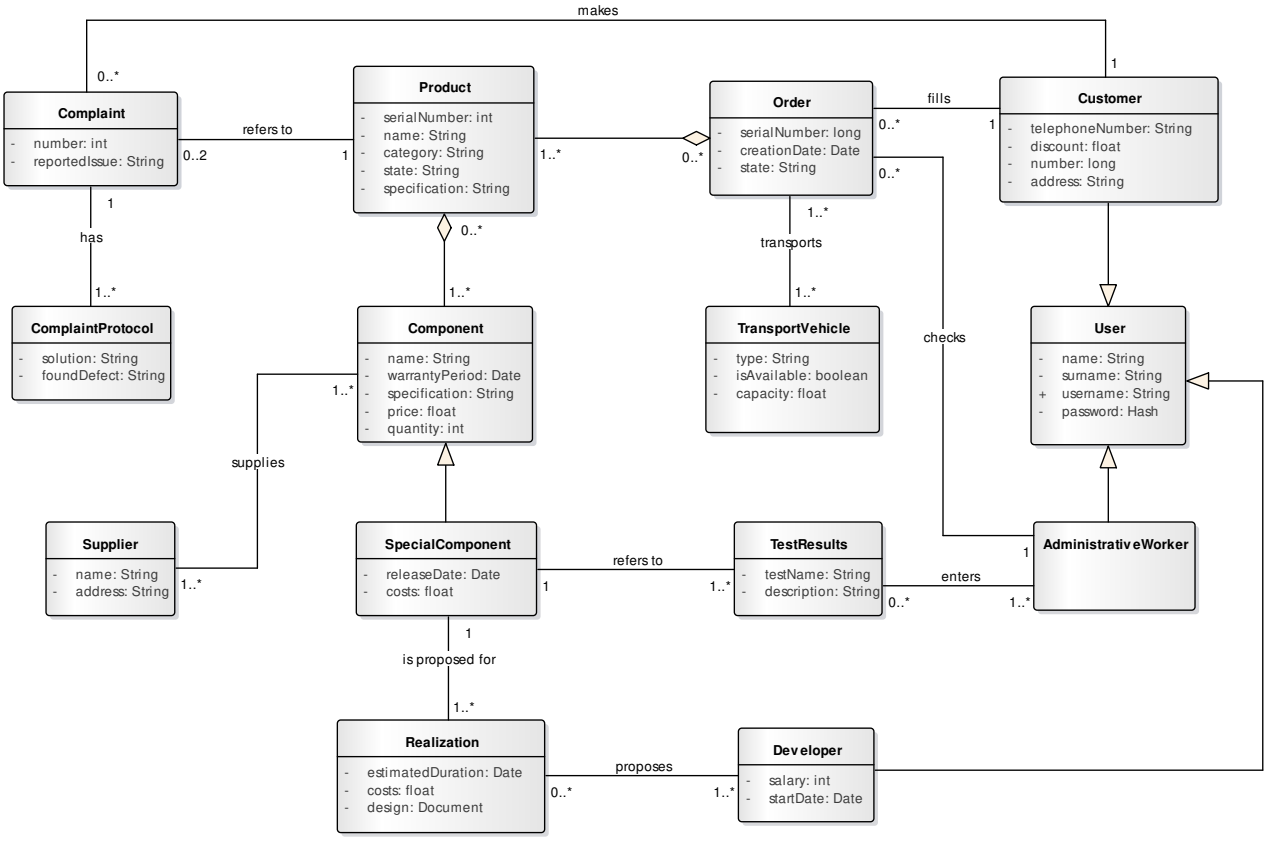
TransportVehicle

Transport vehicle is means of transport that is used for transport of order from store-house to customer.

User

User represents user of system.

Class Model 01 – odvodená verzia č. 1



AdministrativeWorker

Administrative worker deals with current business related to administration of order.

Complaint

Complaint is added to system after reporting of issue by customer.

ComplaintProtocol

Complaint protocol is document that is brought out after solution of complaint.

Component

Component represents part from which product is composed of.

Customer

Customer is person that creates order.

Developer

Developer is responsible for proposal of development realization and administration of test results.

Order

Order contains list of products that customer wants to buy.

Product

Product represents whole that is composed of at least one component.

Realization

Realization is method that is used for development of new component or software.

SpecialComponent

Special component is part of product that is developed by request of customer.

Supplier

Supplier supplies components that company offers.

TestResults

Results of test are outputs of test that was done during development.

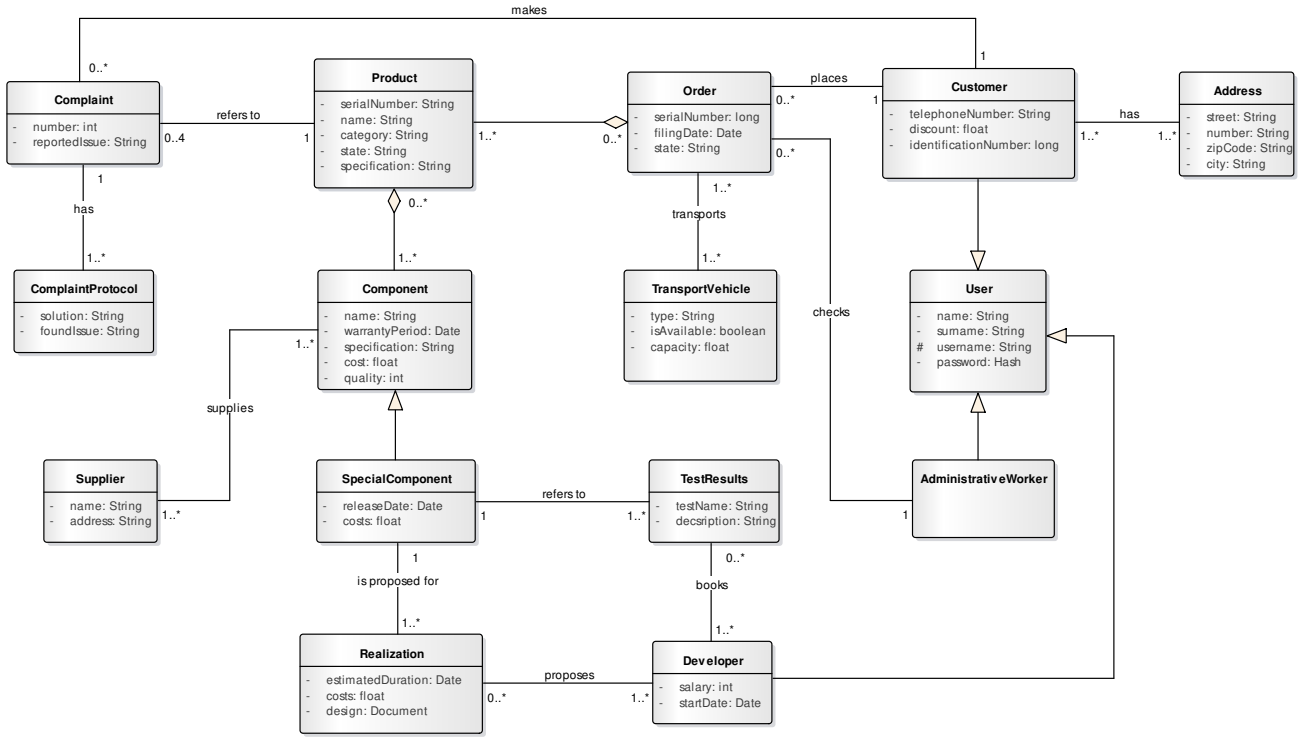
TransportVehicle

Transport vehicle is means of transport that is used for transport of order from store-house to customer.

User

User represents user of system.

Class Model 01 – odvodená verzia č. 2



Address

Address represents address of customer that is composed of street, number, zip code and city.

AdministrativeWorker

Administrative worker deals with current business related to administration of order.

Complaint

Complaint is added to system after reporting of issue by customer.

ComplaintProtocol

Complaint protocol is document that is brought out after solution of complaint.

Component

Component represents part from which product is composed of.

Customer

Customer is person that creates order.

Developer

Developer is responsible for proposal of development realization and administration of test results.

Order

Order represents products list and other information about order.

Product

Product represents whole that is composed of at least one component.

Realization

Realization is method that is used for development of new component or software.

SpecialComponent

Special component is part of product that is developed by request of customer.

Supplier

Supplier supplies components that company offers.

TestResults

Results of test are outputs of test that was done during development.

TransportVehicle

Transport vehicle is means of transport that is used for transport of order from store-house to customer.

User

User represents user of system.

Konflikty v Class Model 01

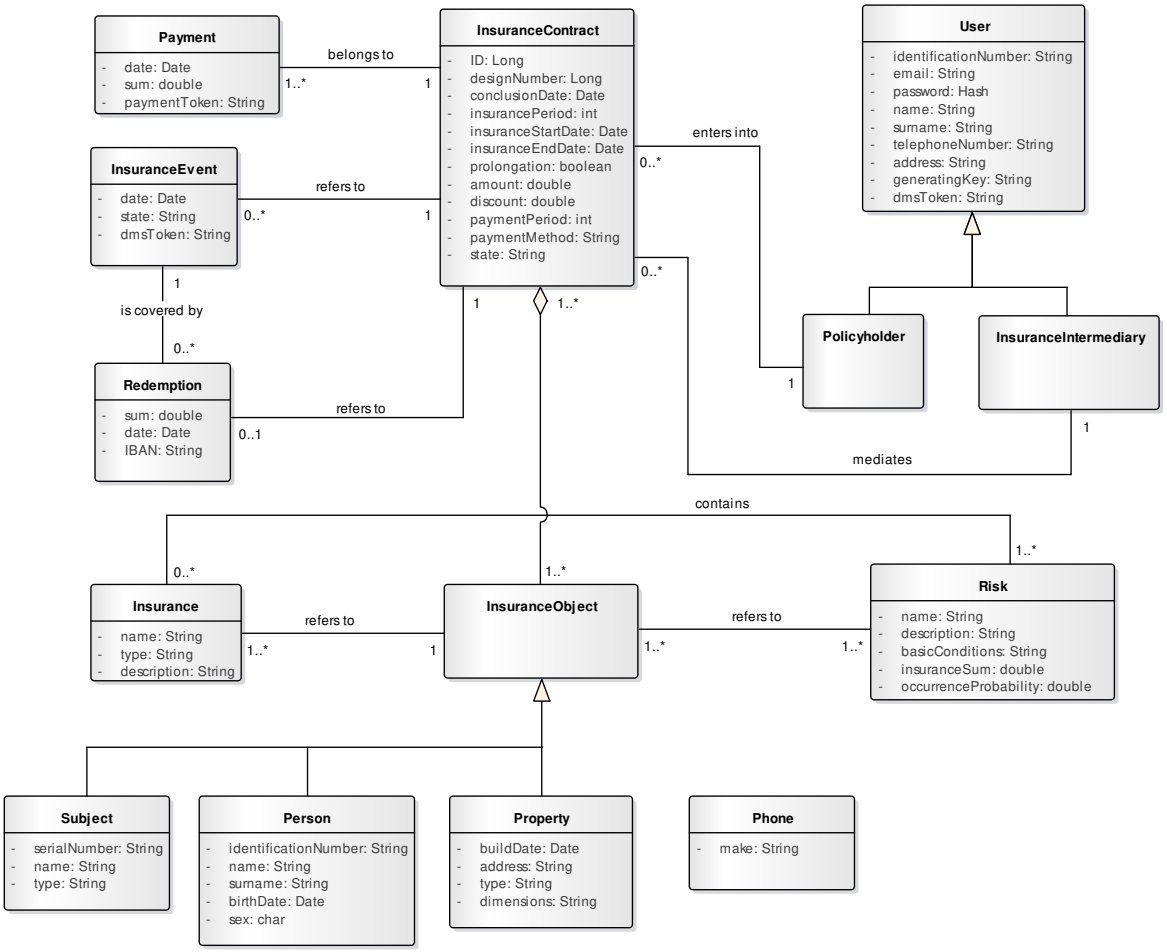
1. Update-Update: typ atribútu serialNumber v triede Product (long)
    - V1: Zmena typu atribútu na int
    - V2: Zmena typu atribútu na String

2. Update-Update: popis triedy Order (Order is list of products that customer wants to buy.)
    - V1: Zmena popisu triedy na „Order contains list of products that customer wants to buy.“
    - V2: Zmena popisu triedy na „Order represents products list and other information about order.“
3. Update-Update: viditeľnosť atribútu username v triede User (Private)
    - V1: Zmena viditeľnosti atribútu na Public
    - V2: Zmena viditeľnosti atribútu na Protected
4. Update-Update: kardinalita asociácie medzi triedami Complaint a Product (0..3)
    - V1: Zmena kardinality asociácie na 0..2
    - V2: Zmena kardinality asociácie na 0..4
5. Update-Update: názov atribútu date v triede Order (popis „Date when order was created.“)
    - V1: Zmena názvu atribútu na creationDate (popis „Date when order was created.“)
    - V2: Zmena názvu atribútu na filingDate (popis „Date when order was filled.“)
6. Sémanticky podobné prvky: pridané atribúty v triede ComplaintProtocol (typ String)
    - V1: Pridanie atribútu foundDefect (popis „Defect of product that was found by customer.“)
    - V2: Pridanie atribútu foundIssue (popis „Found issue of product in complaint.“)
7. Sémanticky podobné prvky: pridané atribúty v triede Customer (typ long)
    - V1: Pridanie atribútu number (popis „ID number of customer generated by system.“)
    - V2: Pridanie atribútu identificationNumber (popis „Identification number of customer.“)
8. Sémanticky podobné prvky: pridané atribúty v triede TestResults (typ String)
    - V1: Pridanie atribútu description (popis „Description of test results.“)
    - V2: Pridanie atribútu decsription (popis „Description of test results.“)
9. Sémanticky podobné prvky: pridané atribúty so synonymickými názvami v triede Component (typ float)
    - V1: Pridanie atribútu price (popis „Sale price of component.“)
    - V2: Pridanie atribútu cost (popis „Selling cost of component.“)
10. Sémanticky podobné prvky: pridané asociácie medzi triedami Order a Customer (kardinality 0..* a 1)
    - V1: Pridanie asociácie s názvom fills
    - V2: Pridanie asociácie s názvom places
11. Sémanticky podobné prvky: pridaný atribút address v triede Customer a pridaná trieda Address
    - V1: Pridanie atribútu address typu String do triedy Customer (popis „Address of customer.“)
    - V2: Pridanie triedy Address s atribútmi street, number, zipCode a city typu String (popis „Address represents address of customer that is composed of street, number, zip code and city.“) a asociácie medzi triedami Customer a Address

Ďalšie zmeny v Class Model 01

1. Asociácia enters medzi triedami Developer a TestResults
    - V1: Zmena zdrojovej triedy asociácie na AdministrativeWorker
    - V2: Zmena názvu asociácie na books
2. Pridané atribúty v triede Component
    - V1: Pridanie atribútu quantity typu int do triedy Component (popis „Quantity of component.“)
    - V2: Pridanie atribútu quality typu int do triedy Component (popis „Quality of component.“)

Class Model 02 – základná verzia



Insurance

Insurance represents various types of insurance that are provided by insurance company. Each insurance contains at least 1 insurance risk.

InsuranceContract

InsuranceContract represents contract of insurance that is concluded by policyholder.

InsuranceEvent

InsuranceEvent represents insurance event that occurred and must be covered by concluded insurance.

InsuranceIntermediary

InsuranceIntermediary is employee of insurance company that mediates creation of insurance and conclusion of insurance contract with policyholder.

InsuranceObject

Payment

Payment represents payment of policyholder for concluded insurance.

Person

Person represents human being that is object of insurance.

Phone

Phone represents electronic equipment used for calling.

Policyholder

Policyholder is person that can conclude various types of insurance on offered insurance objects or configure products from insurance risks.

Property

Property represents real estate that consists of houses and land.

Redemption

Redemption contains information about disbursed sum for concluded insurance.

Risk

Risk represents insurance risks.

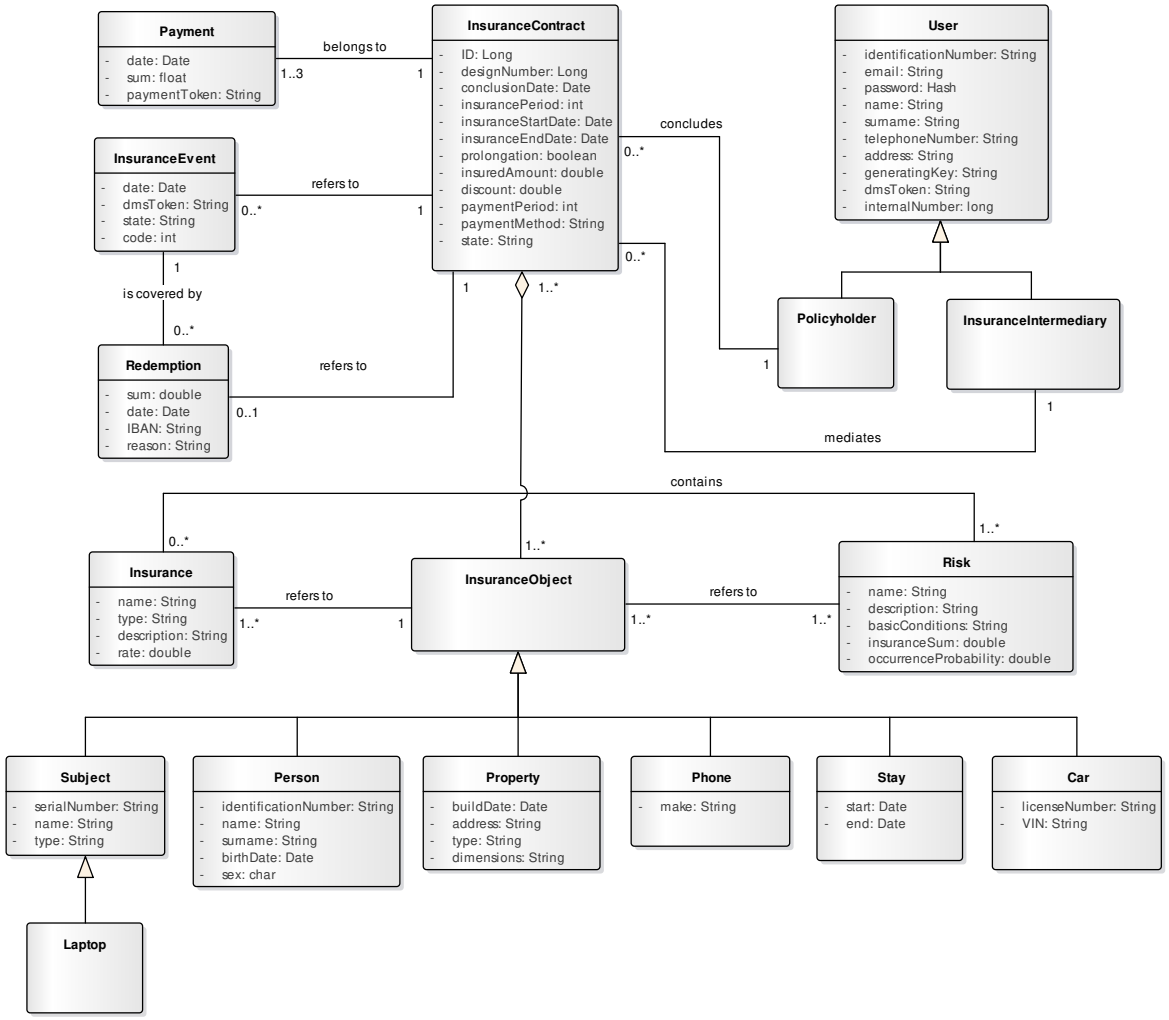
Subject

Subject represents thing that is object of insurance.

User

User represents user of system.

Class Model 02 – odvodená verzia č. 1



Car

Car represents motor vehicle used for transport of passengers.

Insurance

Insurance represents various types of insurance that are provided by insurance company. Each insurance contains at least 1 insurance risk.

InsuranceContract

InsuranceContract represents contract of insurance that is concluded by policyholder.

InsuranceEvent

InsuranceEvent represents insurance event that occurred and must be covered by concluded insurance.

InsuranceIntermediary

InsuranceIntermediary is employee of insurance company that mediates creation of insurance and conclusion of insurance contract with policyholder.

InsuranceObject

Laptop

Laptop represents computer for use outside office.

Payment

Payment represents payment of policyholder for concluded insurance.

Person

Person represents human being that is object of insurance.

Phone

Tool used for voice chat.

Policyholder

Policyholder is person that can conclude various types of insurance on offered insurance objects or configure products from insurance risks.

Property

Property represents real estate that consists of houses and land.

Redemption

Redemption contains information about disbursed sum for concluded insurance.

Risk

Risk represents insurance risks.

Stay

Stay represents period of time spent in a destination.

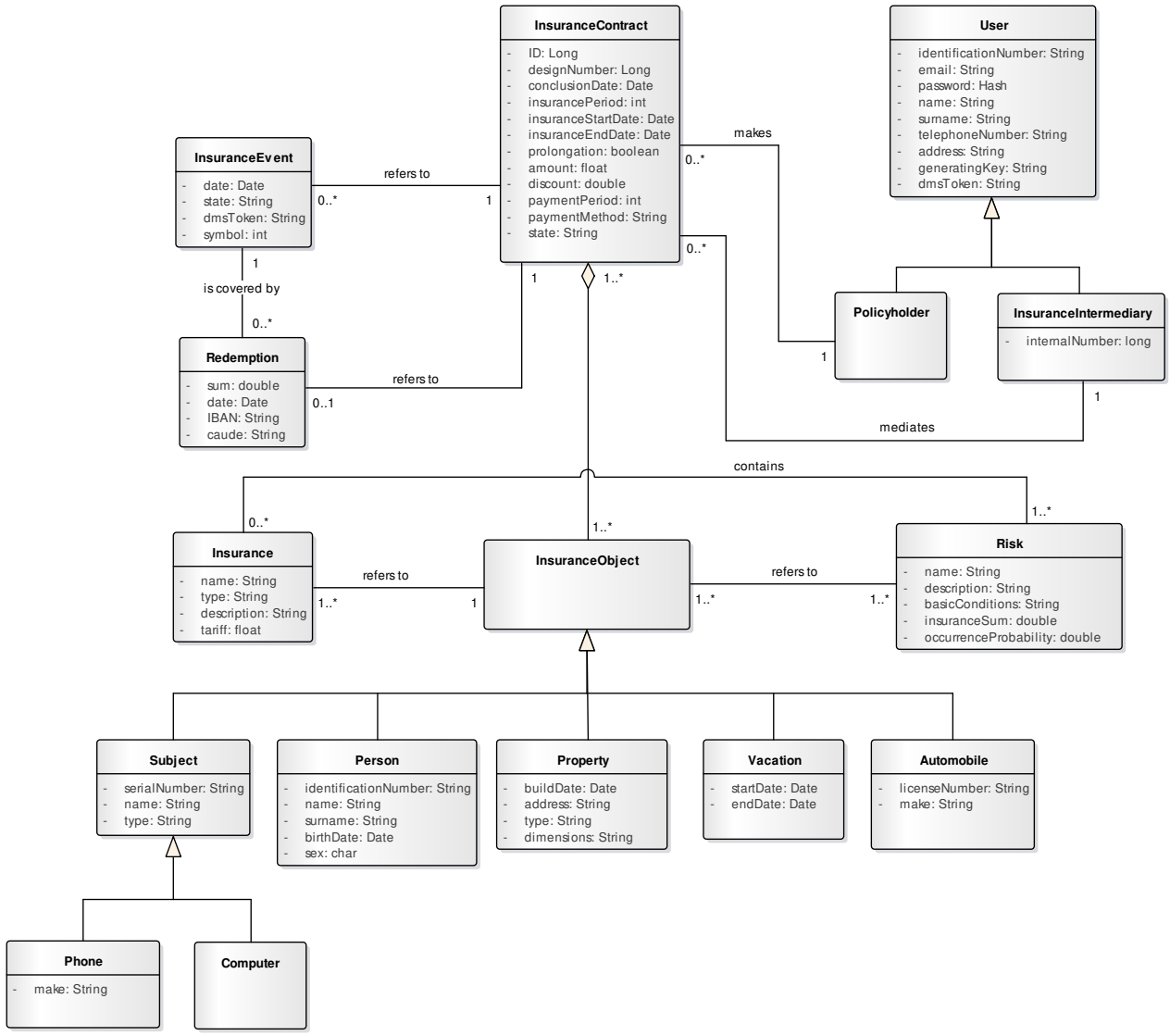
Subject

Subject represents thing that is object of insurance.

User

User represents user of system.

Class Model 02 – odvodená verzia č. 2



Automobile

Automobile represents passenger vehicle with four wheels.

Computer

Computer represents electronic equipment for doing calculations.

Insurance

Insurance represents various types of insurance that are provided by insurance company. Each insurance contains at least 1 insurance risk.

InsuranceContract

InsuranceContract represents contract of insurance that is concluded by policyholder.

InsuranceEvent

InsuranceEvent represents insurance event that occurred and must be covered by concluded insurance.

InsuranceIntermediary

InsuranceIntermediary is employee of insurance company that mediates creation of insurance and conclusion of insurance contract with policyholder.

InsuranceObject

InsuranceObject is object of insurance.

Person

Person represents human being that is object of insurance.

Phone

A telephone with access to a cellular radio system so it can be used over a wide area, without a physical connection to a network.

Policyholder

Policyholder is person that can conclude various types of insurance on offered insurance objects or configure products from insurance risks.

Property

Property represents real estate that consists of houses and land.

Redemption

Redemption contains information about disbursed sum for concluded insurance.

Risk

Risk represents insurance risks.

Subject

Subject represents thing that is object of insurance.

User

User represents user of system.

Vacation

Vacation represents leisure time away from work devoted to rest in a destination.

Konflikty v Class Model 02

1. Update-Update: názov asociácie enters into medzi triedami InsuranceContract a PolicyHolder
    - V1: Zmena názvu asociácie na concludes
    - V2: Zmena názvu asociácie na makes
2. Update-Update: popis triedy Phone (Phone represents electronic equipment used for calling.)
    - V1: Zmena popisu triedy na „Tool used for voice chat.“
    - V2: Zmena popisu triedy na „A telephone with access to a cellular radio system so it can be used over a wide area, without a physical connection to a network.“
3. Update-Delete: asociácia belongs to medzi triedami InsuranceContract a Payment
    - V1: Zmena zdrojovej kardinality asociácie na 1..3 (pôvodne 1..*)
    - V2: Odstránenie triedy Payment
4. UpdateSuccesor-DeleteAncestor: trieda Payment
    - V1: Zmena typu atribútu sum na float (pôvodne double)

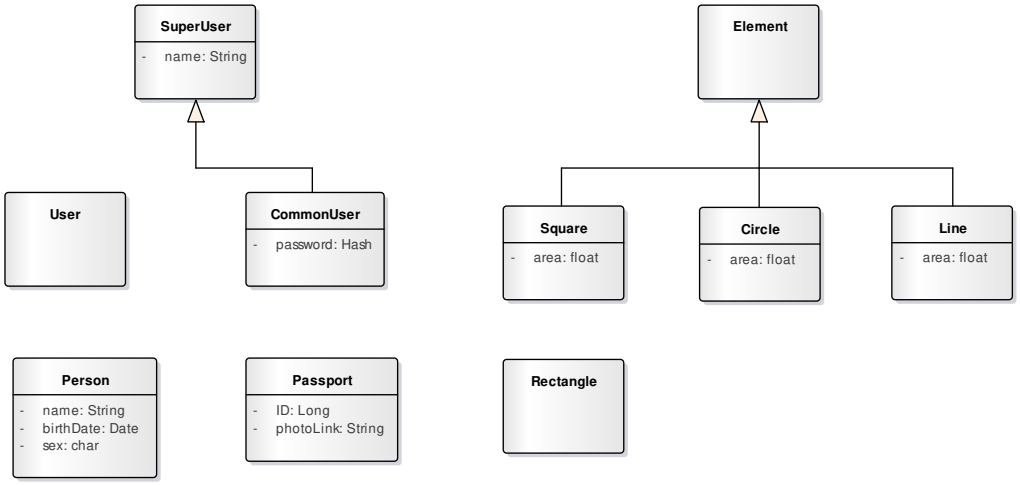
- V2: Odstránenie triedy Payment

5. Sémanticky podobné prvky: pridané atribúty v triede Redemption (typ String)
   - V1: Pridanie atribútu reason (popis „Reason of redemption.“)
   - V2: Pridanie atribútu caude (popis „Cause of redemption.“)
6. Sémanticky podobné prvky: pridané atribúty v triede Insurance
   - V1: Pridanie atribútu rate typu double (popis „Current rate that is related to insurance.“)
   - V2: Pridanie atribútu tariff typu float (popis „Tariff that refers to insurance.“)
7. Sémanticky podobné prvky: pridané atribúty v triede InsuranceEvent (typ int)
   - V1: Pridanie atribútu code (popis „Number that represents type of insurance event.“)
   - V2: Pridanie atribútu symbol (popis „Number of type of insurance event.“)
8. Sémanticky podobné prvky: pridané triedy s podobnými atribútmi typu Date a vzťahu generalizácie do triedy InsuranceObject
   - V1: Pridanie triedy Stay (popis „Stay represents period of time spent in a destination.“) s atribútmi start (popis „Date when stay starts.“) a end (popis „Date when stay ends.“)
   - V2: Pridanie triedy Vacation (popis „Vacation represents leisure time away from work devoted to rest in a destination.“) s atribútmi startDate (popis „Start date of vacation.“) a endDate (popis „End date of vacation.“)
9. Sémanticky podobné prvky: pridané triedy so synonymickými názvami a atribútmi typu String, pridanie vzťahu generalizácie do triedy InsuranceObject
   - V1: Pridanie triedy Car (popis „Car represents motor vehicle used for transport of passengers.“) s atribútmi licenseNumber (popis „License number of car.“) a VIN (popis „VIN of car.“)
   - V2: Pridanie triedy Automobile (popis „Automobile represents passenger vehicle with four wheels.“) s atribútmi licenseNumber (popis „License number of automobile.“) a make (popis „Make of automobile.“)
10. Sémanticky podobné prvky: pridané triedy s podobným významom a pridanie vzťahu generalizácie do triedy Subject
    - V1: Pridanie triedy Laptop (popis „Laptop represents computer for use outside office.“)
    - V2: Pridanie triedy Computer (popis „Computer represents electronic equipment for doing calculations.“)
11. Rovnaké atribúty v hierarchii dedenia (typ long)
    - V1: Pridanie atribútu internalNumber do triedy User (popis „Internal number of user generated by system.“)
    - V2: Pridanie atribútu internalNumber do triedy InsuranceIntermediary (popis „Internal number of insurance intermediary.“)
12. Viacnásobné dedenie
    - V1: Pridanie generalizácie smerujúcej z triedy Phone do triedy InsuranceObject
    - V2: Pridanie generalizácie smerujúcej z triedy Phone do triedy Subject

Ďalšie zmeny v Class Model 02

1. Atribút amount v triede InsuranceContract
   - V1: Zmena názvu atribútu na insuredAmount (popis „Insured amount of insurance contract.“)
   - V2: Zmena typu atribútu na float
2. Trieda InsuranceObject
   - V1: Žiadna zmena
   - V2: Zmena popisu na „InsuranceObject is object of insurance.“

Class Model 03 – základná verzia



Circle

Circle is plane curve generated by one point moving at a constant distance from a fixed point.

CommonUser

CommonUser represents common user of system.

Element

This class represents geometric figure.

Line

Line is straight long geometric figure.

Passport

Passport represents passport that is related to person.

Person

Person represents person in our database.

Rectangle

Rectangle is parallelogram with four right angles.

Square

Square is four-sided regular polygon.

SuperUser

SuperUser represents user that has special abilities.

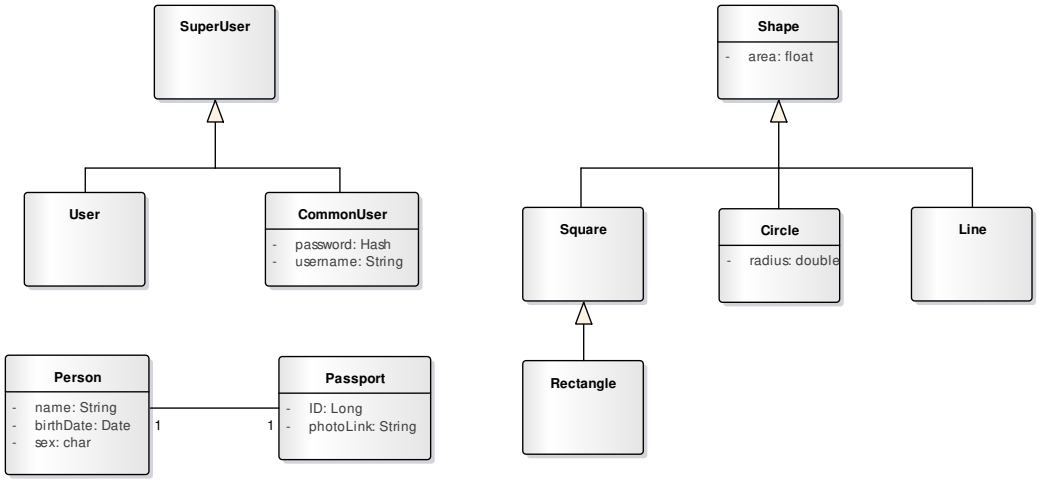
User

User represents user of system.

Class Model 03 – odvodená verzia č. 1

Circle

Circle is plane curve generated by one point moving at a constant distance from a fixed point.



CommonUser

CommonUser represents common user of system.

Line

Line is straight long geometric figure.

Passport

Passport represents passport that is related to person.

Person

Person represents person in our database.

Rectangle

Rectangle is parallelogram with four right angles.

Shape

This class represents geometric figure.

Square

Square is four-sided regular polygon.

SuperUser

SuperUser represents user that has special abilities.

User

User represents user of system.

Class Model 03 – odvodená verzia č. 2

CommonUser

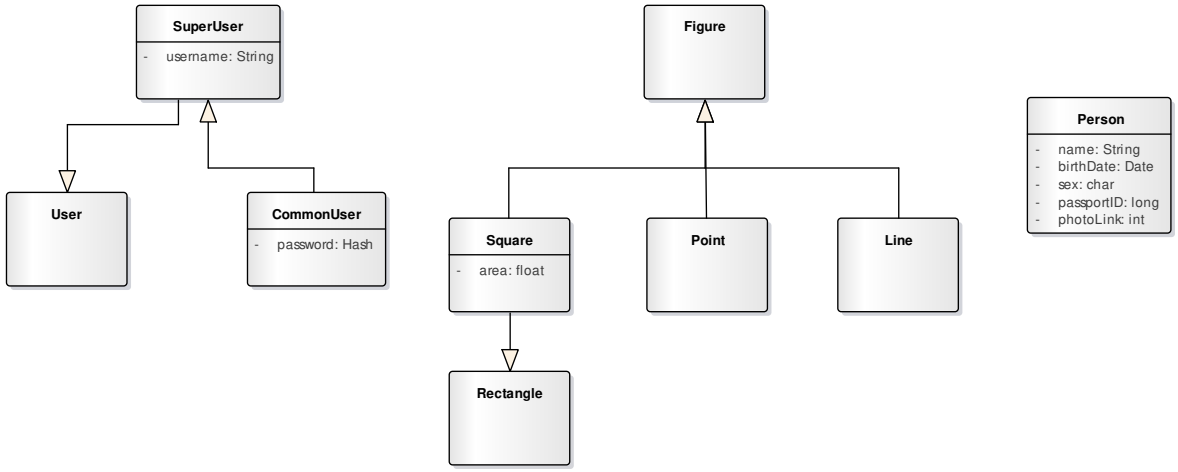
CommonUser represents common user of system.

Figure

This class represents geometric figure.

Line

Line is straight long geometric figure.



Person

Person represents person in our database.

Point

Point is very small circular shape.

Rectangle

Rectangle is parallelogram with four right angles.

Square

Square is four-sided regular polygon.

SuperUser

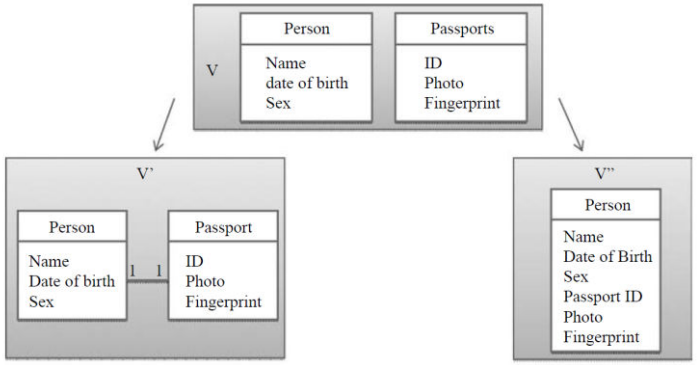
SuperUser represents user that has special abilities.

User

User represents user of system.

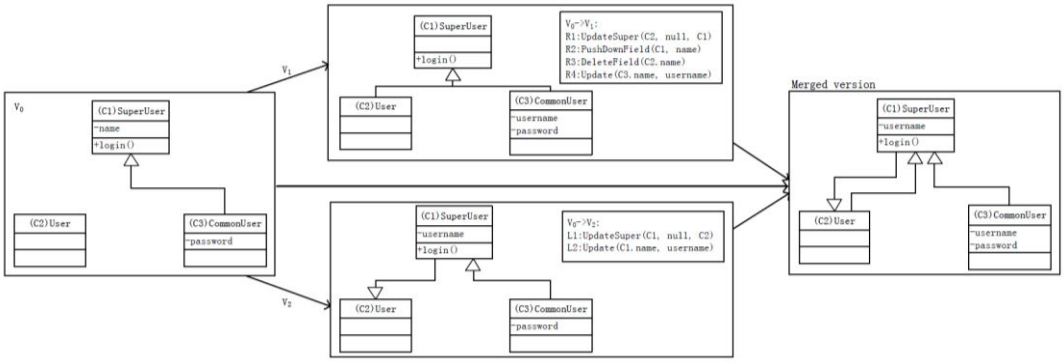
Konflikty v Class Model 03

1. Add-Delete: pridanie vzťahu a odstránenie triedy
    - V1: Pridanie asociácie medzi triedy Person a Passport
    - V2: Odstránenie triedy Passport a presunutie jej atribútov do triedy Person



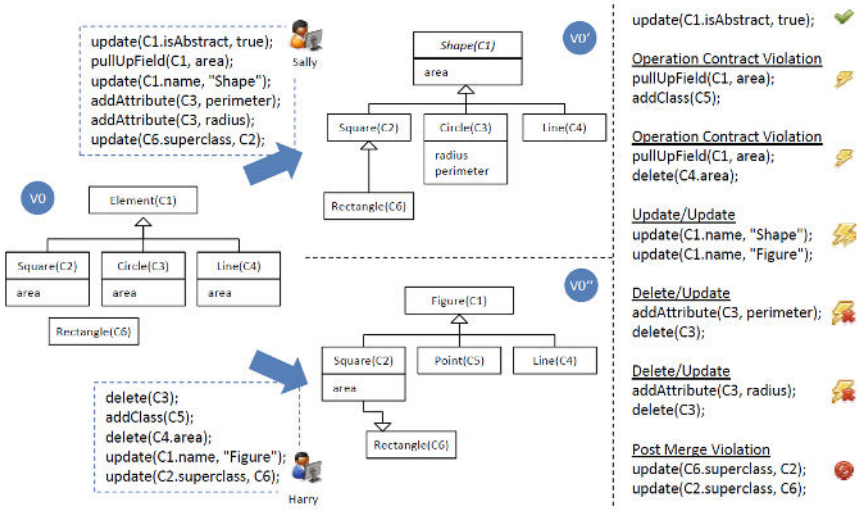
Obrázok E-1: Konflikt č. 1 v Class Model 03 (Altmanninger et al., 2009)

2. Cyklus dedenia
    - V1: Pridanie generalizácie smerujúcej z triedy User do triedy SuperUser
    - V2: Pridanie generalizácie smerujúcej z triedy SuperUser do triedy User
3. Rovnaké atribúty v hierarchii dedenia (typ String)
    - V1: Presunutie atribútu name v triede SuperUser do triedy CommonUser a zmena jeho názvu na username (popis „Username of common user.“)
    - V2: Zmena názvu atribútu name na username v triede SuperUser (popis „Username of super user.“)



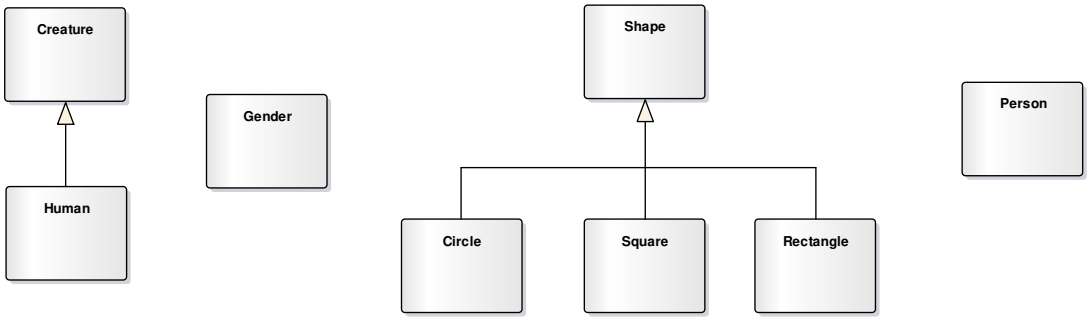
Obrázok E-2: Konflikty č. 2 a 3 v Class Model 03 (Chong et al., 2016)

4. Update-Update: názov triedy Element (popis „This class represents geometric figure.“)
    - V1: Zmena názvu triedy na Shape (popis „This class represents geometric figure.“)
    - V2: Zmena názvu triedy na Figure (popis „This class represents geometric figure.“)
5. AddSuccesor-DeleteAncestor: trieda Circle
    - V1: Pridanie atribútu radius do triedy Circle (popis „Radius of circle.“)
    - V2: Odstránenie triedy Circle
6. Pridanie triedy a presunutie atribútu v hierarchii dedenia
    - V1: Presunutie atribútu area do triedy Shape
    - V2: Pridanie triedy Point (popis „Point is very small circular shape.“)
7. Presunutie a odstránenie atribútu v hierarchii dedenia
    - V1: Presunutie atribútu area do triedy Shape
    - V2: Odstránenie atribútu area z triedy Line
8. Cyklus dedenia
    - V1: Pridanie generalizácie smerujúcej z triedy Rectangle do triedy Square
    - V2: Pridanie generalizácie smerujúcej z triedy Square do triedy Rectangle



Obrázok E-3: Konflikty č. 4 až č. 8 v Class Model 03 (Brosch et al., 2011).

Class Model 04 – základná verzia



Circle

Circle is plane curve generated by one point moving at a constant distance from a fixed point.

Creature

Creature is living organism characterized by voluntary movement.

Gender

Gender represents properties that distinguish organisms on the basis of their reproductive roles.

Human

Human represents any living or extinct member of the family Hominidae characterized by superior intelligence, articulate speech, and erect carriage.

Person

Person represents person in our database.

Rectangle

Rectangle is parallelogram with four right angles.

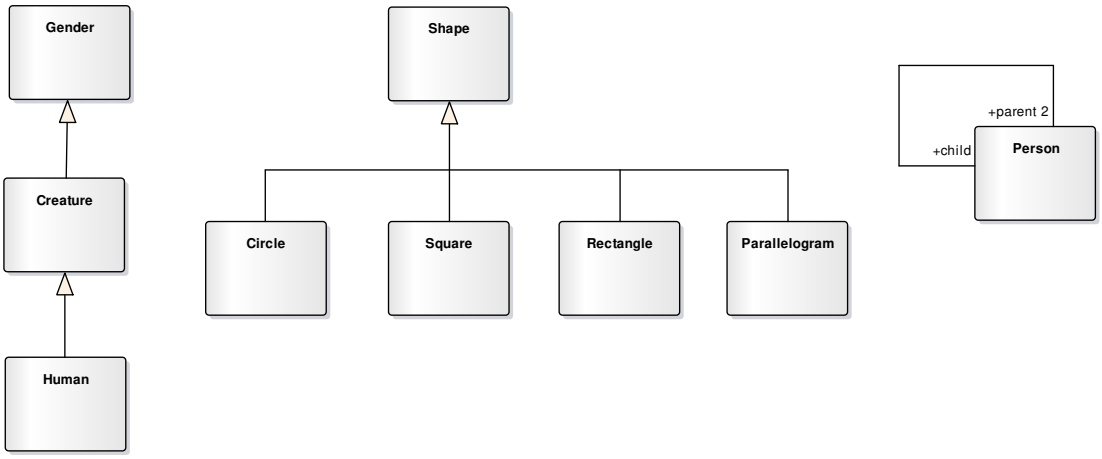
Shape

Shape represents geometric figure.

Square

Square is four-sided regular polygon.

Class Model 04 – odvodená verzia č. 1



Circle

Circle is plane curve generated by one point moving at a constant distance from a fixed point.

Creature

Creature is living organism characterized by voluntary movement.

Gender

Gender represents properties that distinguish organisms on the basis of their reproductive roles.

Human

Human represents any living or extinct member of the family Hominidae characterized by superior intelligence, articulate speech, and erect carriage.

Parallelogram

Parallelogram is quadrilateral whose opposite sides are both parallel and equal in length.

Person

Person represents person in our database.

Rectangle

Rectangle is parallelogram with four right angles.

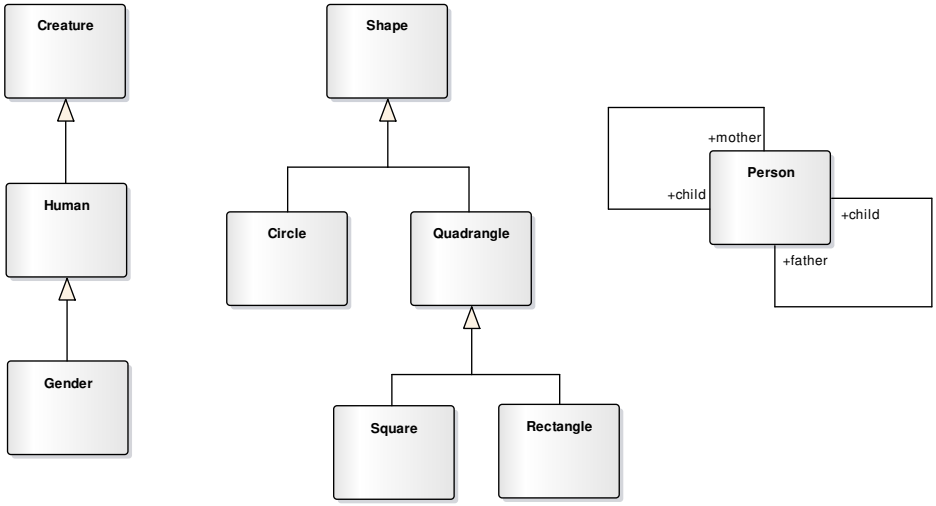
Shape

Shape represents geometric figure.

Square

Square is four-sided regular polygon.

Class Model 04 – odvodená verzia č. 2



Circle

Circle is plane curve generated by one point moving at a constant distance from a fixed point.

Creature

Creature is living organism characterized by voluntary movement.

Gender

Gender represents properties that distinguish organisms on the basis of their reproductive roles.

Human

Human represents any living or extinct member of the family Hominidae characterized by superior intelligence, articulate speech, and erect carriage.

Person

Person represents person in our database.

Quadrangle

Quadrangle is four-sided polygon.

Rectangle

Rectangle is parallelogram with four right angles.

Shape

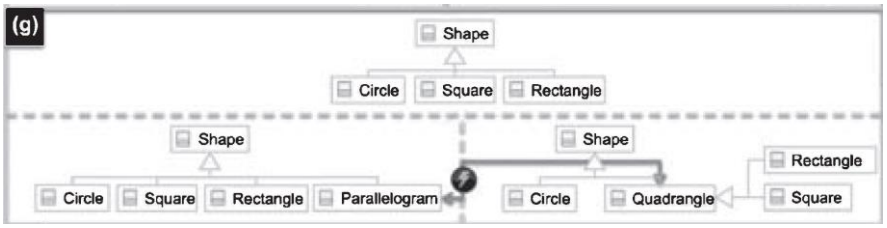
Shape represents geometric figure.

Square

Square is four-sided regular polygon.

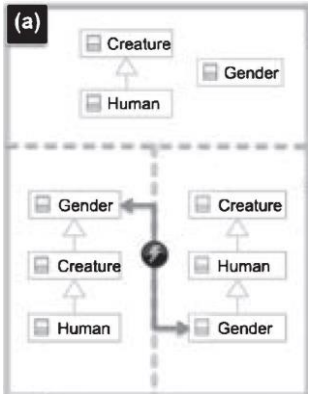
Konflikty v Class Model 04

1. Pridanie tried vo vzťahu hyperonymie na rovnakú úroveň v hierarchii dedenia
    a. V1: Pridanie triedy Parallelogram a generalizácie smerujúcej do triedy Shape (popis „Parallelogram is quadrilateral whose opposite sides are both parallel and equal in length.“)
    b. V2: Pridanie triedy Quadrangle a generalizácie smerujúcej do triedy Shape, triedy Square a Rectangle dedia od triedy Quadrangle (popis „Quadrangle is four-sided polygon.“)



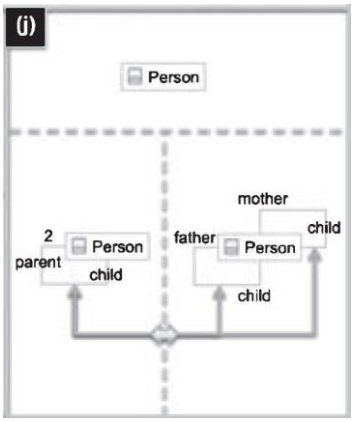
Obrázok E-4: Konflikt č. 1 v Class Model 04 (Altmanninger a Pierantonio, 2011).

2. Cyklus dedenia
    a. V1: Pridanie generalizácie smerujúcej od triedy Creature do triedy Gender
    b. V2: Pridanie generalizácie smerujúcej od triedy Gender do triedy Human



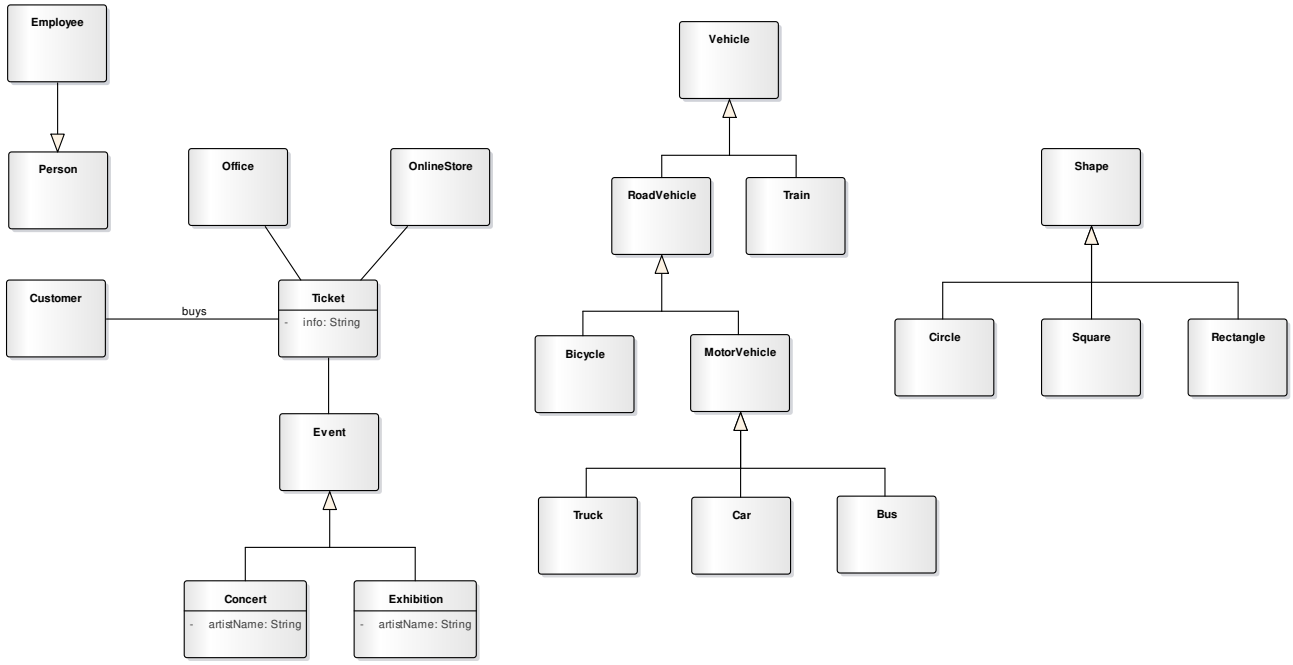
Obrázok E-5: Konflikt č. 2 v Class Model 04 (Altmanninger a Pierantonio, 2011).

3. Rovnaké modelovanie rodičov osoby
    a. V1: Pridanie asociácie s triedou Person s rolami parent a child
    b. V2: Pridanie asociácií s triedou Person s rolami mother a child, father a child



Obrázok E-6: Konflikt č. 3 v Class Model 04 (Altmanninger a Pierantonio, 2011).

Class Model 05 – základná verzia



Bicycle

Bicycle is wheeled vehicle that has two wheels and is moved by foot pedals.

Bus

Bus is vehicle carrying many passengers; used for public transport.

Car

Car is motor vehicle with four wheels; usually propelled by an internal combustion engine.

Circle

Circle is plane curve generated by one point moving at a constant distance from a fixed point.

Concert

Concert is performance of music by players or singers not involving theatrical staging.

Customer

Customer is person that can buy tickets for various events through our system.

Employee

Employee represents person that is employed in our company.

Event

Event is something that happens at a given place and time.

Exhibition

Exhibition is the act of exhibiting.

MotorVehicle

MotorVehicle is kind of road vehicle that has engine.

Office

Office is place of business where professional or clerical duties are performed.

OnlineStore

OnlineStore is place where customers can buy tickets online.

Person

Person represents user in our database.

Rectangle

Rectangle is parallelogram with four right angles.

RoadVehicle

RoadVehicle is kind of vehicle that can be used only on roads.

Shape

Shape represents geometric figure.

Square

Square is four-sided regular polygon.

Ticket

Ticket is commercial document showing that the holder is entitled to visit event.

Train

Train is public transport provided by a line of railway cars coupled together and drawn by a locomotive.

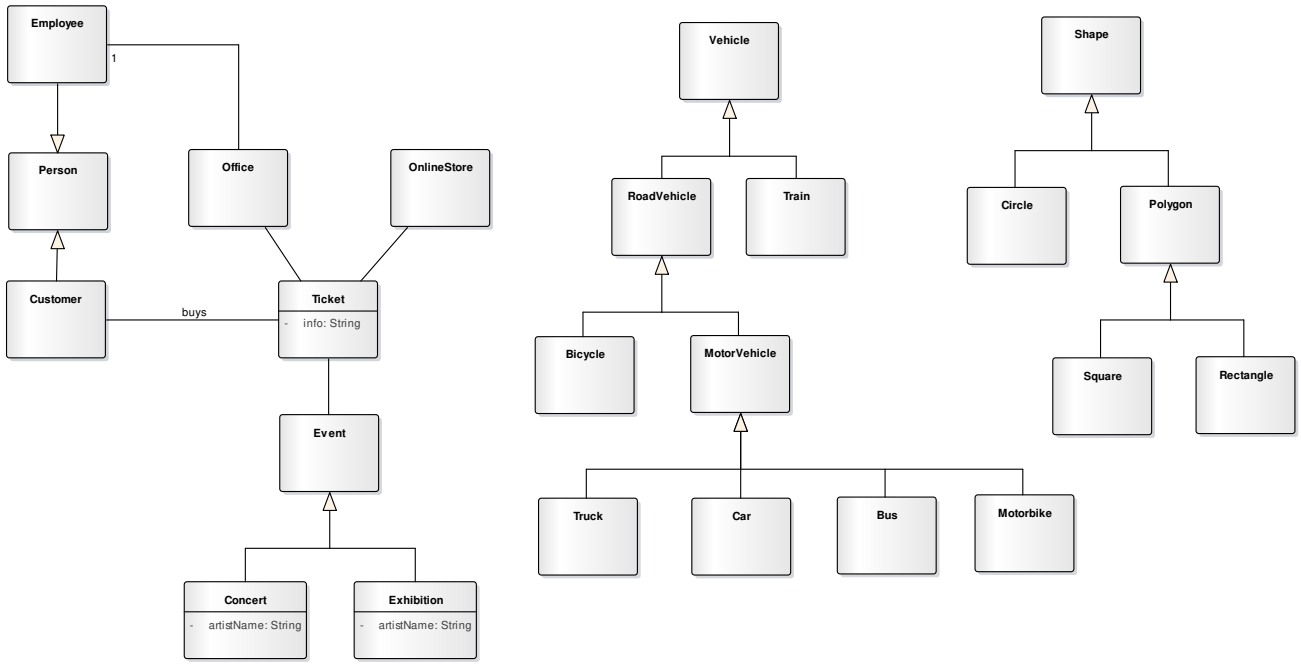
Truck

Truck is handcart that has a frame with two low wheels and a ledge at the bottom and handles at the top; used to move crates or other heavy objects.

Vehicle

Vehicle is conveyance that transports people or objects.

Class Model 05 – odvodená verzia č. 1



Bicycle

Bicycle is wheeled vehicle that has two wheels and is moved by foot pedals.

Bus

Bus is vehicle carrying many passengers; used for public transport.

Car

Car is motor vehicle with four wheels; usually propelled by an internal combustion engine.

Circle

Circle is plane curve generated by one point moving at a constant distance from a fixed point.

Concert

Concert is performance of music by players or singers not involving theatrical staging.

Customer

Customer is person that can buy tickets for various events through our system.

Employee

Employee represents person that is employed in our company.

Event

Event is something that happens at a given place and time.

Exhibition

Exhibition is the act of exhibiting.

MotorVehicle

MotorVehicle is kind of road vehicle that has engine.

Motorbike

Motorbike is small motorcycle with a low frame and small wheels and elevated handlebars.

Office

Office is place of business where professional or clerical duties are performed.

OnlineStore

OnlineStore is place where customers can buy tickets online.

Person

Person represents user in our database.

Polygon

Polygon is closed plane figure bounded by straight sides.

Rectangle

Rectangle is parallelogram with four right angles.

RoadVehicle

RoadVehicle is kind of vehicle that can be used only on roads.

Shape

Shape represents geometric figure.

Square

Square is four-sided regular polygon.

Ticket

Ticket is commercial document showing that the holder is entitled to visit event.

Train

Train is public transport provided by a line of railway cars coupled together and drawn by a locomotive.

Truck

Truck is handcart that has a frame with two low wheels and a ledge at the bottom and handles at the top; used to move crates or other heavy objects.

Vehicle

Vehicle is conveyance that transports people or objects.

Class Model 05 – odvodená verzia č. 2

Bicycle

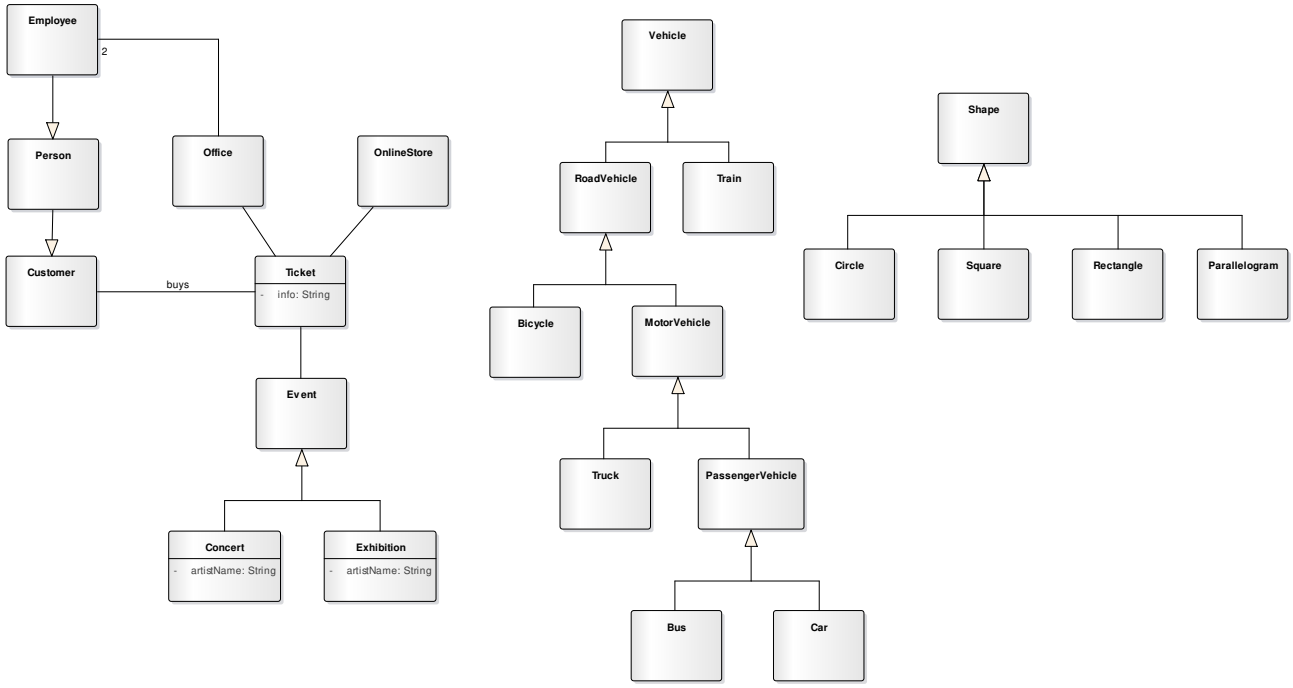
Bicycle is wheeled vehicle that has two wheels and is moved by foot pedals.

Bus

Bus is vehicle carrying many passengers; used for public transport.

Car

Car is motor vehicle with four wheels; usually propelled by an internal combustion engine.



Circle

Circle is plane curve generated by one point moving at a constant distance from a fixed point.

Concert

Concert is performance of music by players or singers not involving theatrical staging.

Customer

Customer is person that can buy tickets for various events through our system.

Employee

Employee represents person that is employed in our company.

Event

Event is something that happens at a given place and time.

Exhibition

Exhibition is the act of exhibiting.

MotorVehicle

MotorVehicle is kind of road vehicle that has engine.

Office

Office is place of business where professional or clerical duties are performed.

OnlineStore

OnlineStore is place where customers can buy tickets online.

Parallelogram

Parallelogram is quadrilateral whose opposite sides are both parallel and equal in length.

PassengerVehicle

PassengerVehicle is kind of motor vehicle that is used for transport of passengers.

Person

Person represents user in our database.

Rectangle

Rectangle is parallelogram with four right angles.

RoadVehicle

RoadVehicle is kind of vehicle that can be used only on roads.

Shape

Shape represents geometric figure.

Square

Square is four-sided regular polygon.

Ticket

Ticket is commercial document showing that the holder is entitled to visit event.

Train

Train is public transport provided by a line of railway cars coupled together and drawn by a locomotive.

Truck

Truck is handcart that has a frame with two low wheels and a ledge at the bottom and handles at the top; used to move crates or other heavy objects.

Vehicle

Vehicle is conveyance that transports people or objects.

Konflikty v Class Model 05

1. Cyklus dedenia
    a. V1: Pridanie generalizácie smerujúcej od triedy Customer do triedy Person
    b. V2: Pridanie generalizácie smerujúcej od triedy Person do triedy Customer
2. Rozdielne kardinality pridanej asociácie
    a. V1: Pridanie asociácie medzi triedami Office a Employee s cieľovou kardinalitou 1
    b. V2: Pridanie asociácie medzi triedami Office a Employee s cieľovou kardinalitou 2
3. Pridanie tried vo vzťahu hyperonymie na rovnakú úroveň v hierarchii dedenia
    a. V1: Pridanie triedy Polygon a generalizácie smerujúcej do triedy Shape, triedy Square a Rectangle dedia od triedy Polygon (popis „Polygon is closed plane figure bounded by straight sides.“)
    b. V2: Pridanie triedy Parallelogram a generalizácie smerujúcej do triedy Shape (popis „Parallelogram is quadrilateral whose opposite sides are both parallel and equal in length.“)
4. Pridanie tried vo vzťahu hyperonymie na rovnakú úroveň v hierarchii dedenia
    a. V1: Pridanie triedy Motorbike a generalizácie smerujúcej do triedy MotorVehicle (popis „Motorbike is small motorcycle with a low frame and small wheels and elevated handlebars.“)
    b. V2: Pridanie triedy PassengerVehicle a generalizácie smerujúcej do triedy MotorVehicle, triedy Car a Bus dedia od PassengerVehicle (popis „PassengerVehicle is kind of motor vehicle that is used for transport of passengers.“)

Príloha F: Obsah elektronického média