VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INTEGRACE NÁSTROJE TELOSYS PRO GENEROVÁNÍ ZDROJOVÉHO KÓDU DO VISUAL PARADIGM

INTEGRATION OF THE TELOSYS CODE GENERATOR INTO VISUAL PARADIGM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDŘEJ VRÁNA**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

153180

Ústav: Ústav informačních systémů (UIFS)
Student: **Vrána Ondřej**
Program: Informační technologie
Název: **Integrace nástroje Telosys pro generování zdrojového kódu do Visual Paradigm**
Kategorie: Softwarové inženýrství
Akademický rok: 2023/24

Zadání:

1. Seznamte se s modelovacím nástrojem Visual Paradigm a s jeho systémem rozšíření pomocí zásuvných modulů. Seznamte se také s nástrojem Telosys pro generování zdrojového kódu a formátem jeho vstupních doménově specifických modelů a šablon pro generování.
2. Navrhněte zásuvný modul pro Visual Paradigm, který umožní tvorbu doménově specifických modelů a šablon pro generování a jejich použití v nástroji Telosys. Inspirujte se již existujícím modulem pro integraci Telosys do nástroje Eclipse.
3. Po konzultaci s vedoucím zásuvný modul pro Visual Paradigm implementujte vč. integrace s nástrojem Telosys uvnitř modulu. Řešení otestujte na různých diagramech doménově specifických modelů a šablonách pro generování.
4. Vyhodnoťte a diskutujte výsledky a výsledný software publikujte jako open-source.

Literatura:

- How to Develop Visual Paradigm Plug-in? *Visual Paradigm* [online]. 2011. Dostupné z: https://www.visual-paradigm.com/tutorials/plugin.jsp [cit. 2023-10-01]
- Telosys documentation [online]. 2022. Dostupné z: https://doc.telosys.org/ [cit. 2023-10-01]
- GUERIN, Laurent. Telosys - a lightweight and pragmatic code-generator [online]. MOdeling LAnguages, 2018. Dostupné z: https://modeling-languages.com/telosys-tools-the-concept-of-lightweight-model-for-code-generation/ [cit. 2023-10-01]
- ONDRÁK, Lukáš. Převod UML diagramů mezi Visual Paradigm a textovými formáty. Brno, 2021. Diplomová práce. VUT, FIT. Dostupné z: https://www.fit.vut.cz/study/thesis/23243/
- PASTOR, Oscar; MOLINA, Juan Carlos. Model-driven architecture in practice: a software production environment based on conceptual modeling. New York: Springer, 2007. ISBN 978-3-540-71867-3

Při obhajobě semestrální části projektu je požadováno:
Body 1, 2 a rozpracovaný bod 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rychlý Marek, RNDr., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 30.10.2023

Abstrakt

Smyslem této práce je vytvoření Visual Paradigm pluginu, který by umožnil využívat generátor kódu Telosys uvnitř tohoto programu. Plugin musí tedy zahrnovat návrh diagramů pro vizualizaci Telosys DSL modelů a jejich převod na Telosys projekt. Práce tedy zahrnuje seznámení se s jednotlivými nástroji Visual Paradigm a Telosys spolu s jejich programovými rozhraními. Dále je navržen grafická reprezentace Telosys nazvaná Telosys diagram a algoritmus jejího převodu. Práce se také věnuje problematice implementace tohoto pluginu spolu s popisem jeho jednotlivých uživatelských funkcí. Na konci práce je implementovaný plugin otestován za pomocí demonstračního příkladu.

Abstract

This thesis aims at creating a Visual Paradigm plugin, which would make it possible to use Telosys code generator inside this program. Plugin must implement the designing of diagrams used for visualisation of Telosys DSL models and their translation to Telosys project. This thesis therefore includes an overview of both Visual Paradigm and Telosys together with their application programming interfaces. It then continues with designing the graphical representation of Telosys named Telosys diagram and describes an algorithm for the diagram's translation. Next chapter describes the implementation of this plugin together with description of all its user functions. Last chapter tests the implemented plugin with example demonstration.

Klíčová slova

Visual Paradigm, Telosys, Model Driven Software Engineering, Doménově specifický jazyk, UML, diagram, generátor kódu, CASE, zásuvný modul

Keywords

Visual Paradigm, Telosys, Model Driven Software Engineering, Domain specific language, UML, diagram, code generator, CASE, plugin

Citace

VRÁNA, Ondřej. *Integrace nástroje Telosys pro generování zdrojového kódu do Visual Paradigm.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

Integrace nástroje Telosys pro generování zdrojového kódu do Visual Paradigm

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Marka Rychlého Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Ondřej Vrána
5. května 2024

Poděkování

Chtěl bych poděkovat vedoucímu práce RNDr. Marku Rychlému Ph.D. za jeho vlídnost a rady v průběhu práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Diagramy a jiné vizuální prvky přispívají k přehlednosti návrhu systému. Mohou být využity k efektivnímu přenosu konceptů a informací mezi jednotlivými členy týmu nebo mezi týmem a zákazníky. Telosys je generátor kódu využívající jazyk Telosys DSL (Domain Specific Language) pro popis svých modelů [10]. Telosys DSL je textový jazyk. Vývojáři využívající tento nástroj tedy nemají k dispozici přehlednost, kterou by jim diagramy zaručili. Tato bakalářská práce má za cíl vytvořit Visual Paradigm plugin pro nástroj Telosys. Plugin využije diagramů a jiných funkcionalit Visual Paradigm pro návrh modelů a entit jazyka Telosys DSL. 2 D grafická reprezentace za pomocí známých prvků a diagramů může přispět k přehlednosti Telosys projektů, a to nejen při jejich tvorbě, ale i v případě následné modifikace. Navrhování datových struktur často začíná za pomocí diagramů, které se následně přepisují na kód. Plugin by v těchto případech mohl jednoduše převést diagramy na modely, ze kterých nástroj Telosys následně může generovat kód i jiné soubory. Uživatel tak nebude muset diagramy náročně přepisovat a tím se ušetří jeho čas a zamezí se potenciálním chybám, které tak mohou při přepisu nastat.

Součástí této práce bude tedy seznámení se s nástrojem Telosys a jím využívanými jazyky Telosys DSL a Velocity Template Language. Následovat bude navrhnutí a tvorba Telosys diagramu ve Visual Paradigm a návrh a implementace pluginu. Součástí pluginu bude několik akcí, z nichž nejdůležitější je samotný převod diagramů na modely a entity. Ostatní jsou doprovodné akce k hlavní. Plugin by tak měl být schopen inicializovat Telosys projekt a připravit projekt Visual Paradigm pro tvorbu Telosys diagramů. Vzhledem k tomu, že Visual Paradigm umožňuje tvorbu mnoha konstrukcí, které u Telosys diagramů nebudou využity nebo povoleny, by měl být plugin také schopný ověřit správnost diagramů.

Kapitola 2

Model Driven Software Engineering

Před programováním aplikace či systému je dobré mít systém alespoň částečně naplánovaný. Začít programovat systém bez předchozích úvah o tom, která funkcionalita bude naprogramována, kde nebo která data je nutné uskladňovat povede k opomenutí některých požadavků zákazníka. Metody a moduly implementující tuto opomenutou funkcionalitu mohou znamenat velké zásahy do systému. Tyto zásahy mohou vytvořit další problémy a než se všechny vyřeší program se natolik liší od původního, že by se dal považovat za kompletně jiný program. S rostoucí velikostí implementovaného systému narůstá nutnost pro plánování a popis systému před jeho naprogramováním.

Výsledkem plánování je model systému popsaného v nějakém modelovacím jazyku. Modelovací jazyky nejsou pouze pro softwarové inženýrství, ale i pro jiné obory. Vytvořit univerzální jazyk umožňující modelování všeho například od elektronických součástek po motory letadel by bylo složité a výsledný jazyk příliš komplikovaný na to, aby zjednodušil vývoj. Jazyky se tak specializují na určitou doménu. Každý prvek jazyka má význam v rámci této domény. Model popsaný v doménově specifickém jazyku je tak jednodušší, protože může použít kontext dané domény. [25]

2.1 Model Driven Architecture

Model Driven Architecture neboli MDA se zabývá popisem systému, a to na tak abstraktní úrovni, že je nezávislí na platformě. MDA označuje technologii vyvinutou společností OMG (Object Management Group) spadající pod MDE (Model Driven Engineering). MDA popisuje chování, funkcionalitu systému, ale i data zpracovávaná a ukládaná systémem. Výsledkem tohoto popisu je abstraktní model obsahující požadavky na systém. Model systému popsaném platformě nezávislým jazykem se označuje jako platformě nezávislí model (anglicky platform independent model – PIM). Platforma jsou technologie použité pro implementaci tohoto systému. Jsou důležité pro finální softwarový produkt ovšem nemají vliv na požadavky. Platforma může být zaměněna za novou bez nutnosti změnit model. PIM by se poté mohl automaticky pomocí nástrojů převést na platformě závislý model (anglicky platform specific model – PSM) pomocí mapování známých konstrukcí. Pod PSM si můžeme představit program zapsaný v nějakém programovacím jazyku nebo jen další model zohledňující konstrukce a technologie cílové platformy. Pokud je PIM popsán do dostatečné míry je možné vygenerovat celý kód programu a vyhnout se tak programování. [17]

Hlavní důvod upřednostnění tohoto přístupu je nezávislost na cílové platformě, tedy operačním systému nebo programovacím jazyku [17]. Generovaný kód je často vysoké kvality, díky času stráveném na vývoji a ladění generátoru[25]. Následná modifikace nebo rozšíření při dalším vývoji staví na již vytvořených modelech. V abstraktním modelu se často orientuje lépe než v programu popsaném ne všem vývojářům známými jazykovými konstrukcemi. Vývojáři neztratí celkový přehled o systému nebo jeho modulech a nedostatky v návrhu či problematika přidání dalších prvků je jasnější. Způsoby popisu, které se použijí při modelování PIM mohou přinést další výhody. Mezi nejznámější takovýto jazyk patří xUML (Executable Unified Modeling Language). Pro popis systému využívá část jazyka UML. Jednotlivým diagramům přiřazuje jasnou sémantiku. Diagramy v UML mají i několik využití. xUML přiřazuje každému diagramu jedno jasné využití, aby bylo vždy jasné, co daný diagram popisuje. [17]

2.2 Telosys a doménově specifické jazyky

Doménově specifické jazyky jsou navrženy pro popis problému v ohraničené části určitého oboru[25]. Práce ve specifické doméně umožňuje, aby jazyky byly malé a neobsahovali redundantní informace a konstrukce. Telosys se oproti jazykům jako xUML nesnaží popsat celou funkcionalitu systému, ale poskytnout základ pro další vývoj [23]. Zaměřuje se pouze na data objektů a nestará se o chování nebo funkcionalitu systému[11]. Spíše, než generování kompletní zdrojového kódu aplikace je výsledkem generování kostra aplikace. Programátor pak doplní chybějící chování. Toto umožní rychlý start do vývoje [23].

Jak Telosys, tak xUML vytváří platformě nezávislé modely. Rozdíl je v čase stráveném vývojem těchto modelů. Modelování Telosys se odehrává hlavně na začátku vývojového cyklu a po vygenerování kódu se již nemusí používat [23]. Modifikace již vygenerovaných struktur by totiž nutně znamenala i přepis člověkem naprogramovaného kódu. Smysl tedy dává na začátku nebo pro následné rozšíření o další struktury. Vývoj model v xUML trvá podstatně delší dobu a jeho výsledkem může být i kompletní aplikace bez nutnosti dodatečných zásahů. Modifikací modelu a znovu vygenerováním kódu se tak neztratí člověkem naprogramovaný kód. Telosys DSL a podobné neúplné jazyky má tedy smysl využít pro návrh systémů, u kterých nepředpokládáme velké změny nebo systémy kde ruční změna kódu je rychlejší než přepis modelu.

Kapitola 3

Case nástroje

Softwarové inženýrství se zabývá všemi etapami životního cyklu vývoje softwaru a všemi přidruženými aspekty jako například managmentem projektu a vývojem nástrojů a metodik pro vývoj softwaru. Doporučuje metodiky, které přispívají k úspěšnému doručení softwarových produktů, jak pro malé, tak i pro velké produkty. Různé projekty vyžadují různé přístupy, a to nejen podle velikosti vývojářského týmu nebo velikosti software, ale také podle typu softwaru, který je vyvíjen. Například počítačové hry by měli být vyvíjeny sérií prototypů, naproti tomu bezpečnostně kritické systémy potřebují kompletní a analyzovatelnou specifikaci. [18]

CASE (computer-aided software engineering) nástroje se používají v jednotlivých etapách vývoje [18]. Stejně jako softwarové inženýrství přispívají k rychlejšímu vývoji a vyšší kvalitě výsledného produktu. Podporují dodržování konvencí a standardů pomocí automatizace kontrol a generováním částí projektu jako například část kódu nebo dokumentace. Mohou obsahovat nástroje pro tvorbu diagramů, generování kódu z návrhu, nástroje umožňující sdílení projektů s ostatními návrháři nebo nástroje pro generování dokumentace. CASE nástroje mají ale i svoje nevýhody. Některé nástroje jsou velmi drahé a nemusí se tak vyplatit pro vývoj. Nástroje umožňující "round-trip engineerin" jsou velmi komplexní a vyžadují od vývojáře, aby nástroj důkladně znal [23]. Vzhledem k tématu této práce jsou nejdůležitější nástroje pro tvorbu diagramů a nástroje pro generování kódu. [16]

3.1 Visual Paradigm

Visual Paradigm je CASE nástroj pro rozsáhlou škálu aspektů softwarového vývoje [24]. Byl založený společností Visual Paradigm International se sídlem v Hong Kongu [24] [1]. Umožňuje návrh systému za pomocí diagramů UML, ERD, SysML a dalších. Podporuje práci v týmu pomocí ukládání do cloudu, komentářů, tagů atd. Obsahuje nástroje např: pro UX design a prototypování, generování kódu z modelů a textu nebo generování dokumentace. [24] [2]

Program je rozšiřitelný pomocí pluginů napsaných v jazyce Java [24] [3]. Uživatel je tak programově schopen přistupovat k prvkům modelu a diagramům, číst je a modifikovat. Může také vytvářet nové modely a diagramy nebo využít funkcionalitu generování kódu. Pro

[1] Tato infromace je na stránce https://www.visual-paradigm.com/aboutus/

[2] Tato infromace je na stránce https://www.visual-paradigm.com/editions/

[3] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/124/254/7039_introduction.html

komunikaci s Visual Paradigm pluginy využívají API definovanou v archivu „openapi.jar“. API je dostupné ze složky lib v adresáři instalace programu [24] [4].

Existuje několik způsobů zavedení pluginu do programu. Všechny využívají stejné soubory, které jsou pouze jinak zabaleny. Nejjednodušší je instalace plugin pomocí okna v záložce „Help“ -> „Install Plugin“. Bez ohledu na to, který ze způsobů si zvolíte je výsledkem instalace nová složka v adresáři plugins obsahující všechny nutné soubory. Jediný rozdíl ve způsobech instalace je způsob jakým se nutné soubory dostanou do této složky (překopírovat ručně nebo automaticky podle způsobu zabalení souborů). [24] [5]

3.1.1 Generováni kódu ve Visual Paradigm

V rámci UML a jiných diagramů generování kódu znamená generování části zdrojového kódu softwaru za použití informací extrahovaných z diagramů. Visual Paradigm umožňuje generování kódu, reverzní inženýrství a Round-Trip Engineering. Reverzní inženýrství v tomto kontextu znamená automatickou tvorbu diagramů ze zdrojového kódu. Round-Trip Engineering kombinuje oba předchozí případy tak, že při modifikaci jedné strany je změna automaticky přenesena na druhou. Při změně zdrojového kódu se tak aktualizují diagramy a při změně diagramů se aktualizuje zdrojový kód. [24] [6].

Round-Trip lze používat mezi diagramem tříd a kódem, ovšem pouze pro Java a ANSI C++. Jednostranné generování jak z modelů tříd do kódu, tak naopak je dostupné ve více jazycích, a to v Java, C#, C++, Python, PHP, Hibernate, Ruby, VB.NET, .NET, ODL, ActionScript, IDL, Delphi, Perl, XML, XML Schema, Objective-C 2.0 nebo Ada95. Program dále podporuje generování sekvenčních diagramů z Java tříd, a také generování kódu konečného automatu z diagramu tříd a stavového diagramu v mnoha jazycích. [24] [7].

Navíc mimo tyto základní možnosti obsahuje také Graphical API designer. Rozšířením diagramu tříd je uživatel schopen generovat REST API kód nebo Swagger a OpenAPI definice. [24] [8].

Používání generátoru kódu ve Visual Paradigm je jednoduché. Program umožňuje nastavit řadu vlastností přes GUI přímo v aplikaci [24] [9] ovšem pro specifičtější zásahy je nutné editovat šablony pro generování. Visual Paradigm používá pro generování Apache Velocity a jeho jazyk Velocity Template Language. Předpřipravené šablony lze modifikovat nebo použít vlastní. [24] [10]

Velocity je open-source projekt zaměřený na vytvoření šablonového jazyka. Využívá se hlavně pro tvorbu webových stránek s MVC architekturou. Slouží jako alternativa k Java Server Pages a PHP. Používá se také pro generování SQL, PostSriptu, XML, automatických emailů a dalších. Velocity bylo vytvořeno společností Apache Software Foundation pod licencí Apache Software License. [22]

Šablony používají reference odkazující na objekty a řadu klíčových slov a jazykové konstrukce jako iterace a selekce a předdefinované funkce. Tyto konstrukce dávají uživateli větší

[4] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/124/254/7040_implementing.html

[5] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/124/254/7041_installingpl.html

[6] Tato infromace je na stránce https://www.visual-paradigm.com/features/code-engineering-tools/

[7] Tato infromace je na stránce https://www.visual-paradigm.com/features/code-engineering-tools/

[8] Tato infromace je na stránce https://www.visual-paradigm.com/features/visual-api-designer/

[9] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/276/330/7334_instantgener.html

[10] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/276/330/7416_customizingc.html

kontrolu nad generovanými soubory, dokonce lze předčasně ukončit generování například není-li splněna nějaká podmínka [22]. Šablona nemusí být nutně definována v jednom souboru, ale je možné ji rozdělit na více souborů a odkazovat se na ně v jiných šablonách. Velocity umožňuje odkazovat se na Java třídy a přistupovat k jejich prvkům [22] [11] . Lze tedy volat metody jednotlivých instancí reprezentujících modely tříd ve Visual Paradigm [24] [12] . Objekty jejich rozhraní se shodují s rozhraním v API archivu „openapi.jar“.

Díky tomu, že VTL umožňuje přístup k metodám Java instancí má uživatel při psaní šablon přístup ke stejnému množství informací, jako má při psaní pluginů. Kromě výběru tříd, na které se použije šablona a modifikace šablon uživatel ale nemá další kontrolu nad generováním. Například když potřebuji použít několik různých šablon na několik skupin modelů. Generování se stává komplikovaným, není zde možnost automatizace.

3.2 Další CASE nástroje

Visual Paradigm není jediný CASE nástroje na trhu. Existuje celá řada jednodušších nástrojů pro tvorbu diagramů jako například Microsoft Visio [13] nebo LucidChart [12]. Mezi další nástroje s generováním kódu patří například UML lab [26] nebo Altova UModel [1].

UML lab se specializuje na práci s diagramy UML. V tomto ohledu nabízí podobnou funkcionalitu jako Visual Paradigm, a navíc obsahuje automatické dokončení a jiné vylepšení efektivity práce. Dále od předchozích alternativ obsahuje Round-Trip Engineering, generování kódu pomocí Xpand a Xtend šablon i reverzní inženýrství. UML lab nenabízí neplacenou verzi jako „community edition“ verze v případě Visual Paradigm. [26]

Altova UModel je další CASE nástroj specializující se na UML s podporou pro SysML a Business Process Model Notation. Opět nabízí všechny 3 způsoby transformace mezi kódem a modelem. Výběr jazyků je ale omezen na C#, Visual Basic, Java a C++. Stejně jako UML lab nenabízí neplacenou verzi. [1]

Enterprise Architect se zabývá spoustou aspektů vývoje systémů[19]. Diagramy jako UML, SysML, BPMN nebo UPDM. Testování, ladění a simulace chování navržených systémů. Z výše zmíněných programů podporuje Altova UModel [1] [13] a Enterprise Architect [19] [14] uživatelské pluginy.

[11] Tato infromace je na stránce https://velocity.apache.org/engine/1.7/user-guide.html

[12] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/276/330/7416_customizingc.html

[13] Tato infromace je na stránce https://www.altova.com/manual/UModel/umodelprofessional/test_umodel_plugin.html

[14] Tato infromace je na stránce https://sparxsystems.com/enterprise_architect_user_guide/16.1/the_model_repository/pcs_integration_plugins_custom_write.html a https://sparxsystems.com/enterprise_architect_user_guide/16.1/add-ins___scripting/creatingaddins.html

Kapitola 4

Telosys

Telosys Code Generator je open-source generátor kódu vyvinutý společností Telosys [10]. Cílem projektu Telosys bylo vytvořit jednoduchý, rozšiřitelný nástroj, schopný generovat kód v jakémkoli jazyce. Obsahuje sice několik předdefinovaných programovacích jazyků [10] [1], ovšem Telosys je možné použít i na ostatní programovací jazyky, popřípadě i na generování jiných souborů jako například XML nebo HTML. Uživatel komunikuje s programem přes Telosys-CLI; konzolové rozhraní spuštěné při zapnutí programu [10] (spouštěné přes příkazový řádek). Program je napsaný v Javě a poskytuje jednoduchou API knihovnu pro vývojáře rozšíření [7]. Telosys Code Generator je integrován do Eclipse IDE [10] [2] jako plugin. Ecplise plugin není jediný, který je zmíněný na stránkách Telosys. Lze najít i jiné pluginy jako například plugin pro Visual Studio Code [10] [3] . Poslední verze Visual Studio Code pluginu je ovšem z 5.1.2018, neobsahuje tedy nejnovější prvky Telosys DSL.

Telosys generuje textové soubory podle šablon a informací z modelů. Šablony jsou popsány v Apache Velocity Template Language [10] (dále jen VTL), tedy ve stejném jazyce jako šablony Visual Paradigm pro „Instant Generation“. Šablony se seskupují do balíčku označovaných bundle. Každý bundle obsahuje VTL šablony s příponou .vm, soubor README.md pro popis šablony, konfigurační soubor templates.cfg a další soubory. [11]

Pro modely Telosys používá vlastní Domain Specific Language, který je v dokumentaci označován jako Telosys DSL. Každý model je adresář, kde jméno složky je jméno modelu. Složka obsahuje model.yaml soubor (obsahující dodatečná data a nastavení modelu) a několik souborů s příponou .entity (obsahující definice jednotlivých entit v Telosys DSL). Model lze vytvořit ručně nebo zpětně vygenerovat z relační databáze. [11]

Před samotnou tvorbou entit a modelů je nutné projekt vytvořit. Telosys-CLI nabízí příkaz pro inicializaci projektu v uživatelem vybrané složce. Inicializace vytvoří složku „TelosysTools“ obsahující všechny soubory související s projektem. Uvnitř složky „TelosysTools“ je projekt rozdělen na několik podsložek a konfiguračních souborů. Mezi tyto konfigurační soubory mimo jiné patří například „databases.yaml“, který obsahuje informace o databázích, ze kterých si uživatel může přát vytvořit modely. Dalším z nich je „telosys-tools.cfg“, což je konfigurační soubor obsahující ostatní informace o projektu, které se netýkají databáze. Informace, jako cílová místa, kam mají být generované soubory ukládány (SRC, RES, TEST_SRC a další), konfigurace síťové proxy nebo jiné specifické projektové proměnné (PROJECT_NAME, PROJECT_VERSION, REST_API_ROOT a další). „telosys.env“ obsahuje poslední nastavené hodnoty při používání Telosys-CLI. Při následujícím

[1] Tato infromace je na stránce https://doc.telosys.org/target-languages
[2] Tato infromace je na stránce https://www.telosys.org/eclipsePlugin.html
[3] Tato infromace je na stránce https://doc.telosys.org/telosys-vscode

spuštění toho konkrétního projektu si pak toto nastavení terminál znovu načte a uživatel tak může pokračovat v práci bez nutnosti úprav prostředí. Ze složek jsou významné hlavně složky „./models“ a „./templates“. Složka „./models“ obsahuje složky jednotlivých modelů, kdy název složky je název modelu. Například složka „./models/eshop“ obsahuje entity na konfigurace modelu „eshop“. Ve složce „./templates“ jsou umístěny všechny balíčky šablon. Název složky opět tvoří název balíčku, tedy „./templates/RESTCSharp“ je balíček šablon s názvem „RESTCSharp“. Telosys-CLI obsahuje i řadu dalších příkazů pro manipulaci a prohledávání projektu. Uživatel může využít tyto funkce a jednoduše vytvořit modelové a entitní soubory. Následně vytvořené entity a modely prohledávat, kontrolovat a editovat. Editace spustí uživatelem nastavené textový editor jako například Visual Studio Code, Atom, Notepad++ a další. Typické práce s Telosys projektem tedy zahrnuje spuštění Telosys-CLI a inicializaci projektu. Pro úspěšné inicializaci uživatel použije ostatní funkce CLI k vytvoření a editaci modelů a entit. Následně nainstaluje potřebné šablony a upraví je pro své potřeby. Nakonec použije funkce pro generování z Telosys-CLI k vygenerování chtěných souborů. Visual Paradigm plugin by změnil první část vytváření tohoto projektu. Místo inicializace projektu pomocí Telosys-CLI by uživatel vytvořil Visual Paradigm projekt. V něm poté vytvořil UML modely a Telosys diagramy. Tento projektu pomocí plugin převedl na Telosys projekt a pokračovat pomocí Telosys-CLI instalací šablon atd. [11]

Jak Visual Paradigm [24] [4], tak Telosys používají Velocity Template Language [22]. Z tohoto pohledu jsou stejné. Zásadní rozdíl z pohledu šablon je v kontrole generování. Visual Paradigm nemá žádný konfigurační soubor, který by uživateli dal možnost mít větší kontrolu nad generováním. Definice modelů ve Visual Paradigm má výhodu ve větší přehlednosti, díky návrhu pomocí diagramů. Vytvoření grafické reprezentace by dokázalo vyřešit hlavní problém Telosys. Díky ní by pak byl stejně přehledný jako diagramy tříd ve Visual Paradigm a zároveň si zachoval všechny svá pozitiva.

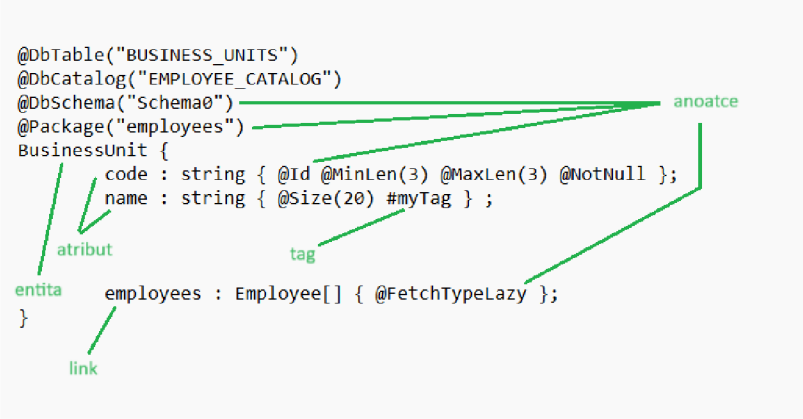
4.1 Použitá terminologie

V průběhu této práce bude často používáno několik pojmů, které budou používány v kontextu s ostatními. I když se jejich význam může zdát jasný vzhledem k názvu práce a dosud probraným pojmům, nemusí tomu tak být pro všechny čtenáře. Prvním takovým pojmem je Telosys diagram. Jedná se o diagram (konkrétně UML diagram tříd [14] využívající balíčky), který využívá zde navrženou grafickou reprezentaci k modelování Telosys DSL entit. Pouze Telosys diagramy mohou být pluginem převedeny na modely a entity Telosys DSL. Visual Paradigm projekt je projekt v programu Visual Paradigm [24] uložený v souboru s příponou „.vpp“. Tento pojem bude často používán v kontextu projektu vytvořeného za účelem tvorby nebo obsahující Telosys diagramy.

4.2 Telosys DSL

Telosys DSL (zkratka pro Domain Specific Language, česky doménově specifický jazyk) je jazyk, který Telosys Code Generator používá pro popis modelů. Každý projekt obsahuje modely, které sestávají z entit popsaných tímto jazykem. Existuje 5 základních prvků Telosys DSL: entita, atribut, link, anotace a tag.[11]

[4] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/276/330/7416_customizingc.html



Obrázek 4.1: Entita v Telosys DSL

Jak je vidět na obrázku 4.1 Definice entit se podobá popisu tříd z objektově orientovaných jazyků. Entita obsahuje jednotlivé atributy a linky. Atribut má v Telosys DSL stejný význam jako atribut třídy v objektově orientovaných paradigmatech. Musí mít název a jeden z neutrálních datových typů Telosys DSL. Neutrální typy se při generování konvertují na typy cílového jazyka. Linky ukazují na jiné entity v modelu. Jejich datový typ je tedy název jiné entity.[11]

Anotace a tagy jsou dekorace pro entity, atributy a linky. Používají se pro přidání dodatečných informací o anotovaném prvku. Mohou například omezit množinu hodnot, kterých atribut může nabývat, upravit neutrální datový typ atributů nebo přiřadit entitu do balíčku. Seznam všech anotací lze najít v dokumentaci pro Telosys DSL. Význam některých anotací je popsán neurčitě. Hlavní důvod je nejspíše aby uživatel mohl lehce přizpůsobit význam podle svých potřeb, popřípadě podle významu v cílové platformě. Příklad může být anotace @Context, jejíž celý popis v dokumentaci je „Defines the context to which the entity belongs“. Existují i anotace, které jsou v dokumentaci označené jako experimentální (experimental) nebo zastaralé (Deprecated). V této bakalářské práci se pokusím implementovat všechny anotace, i ty které žádný popis nemají. Anotace mohou vyžadovat dodatečné hodnoty. Například anotace @Label vyžaduje textový řetězec, který bude používán jako popisek. Tagy jsou jednoduše řečeno uživatelem definované anotace. Uživatelé je využijí hlavně v případě, kdy neexistuje anotace nesoucí požadovanou informaci. Mohou mít maximálně jednu hodnotu. Tagy není třeba nikde dopředu definovat.[11]

V šablonách jsou informace o základních prvcích modelu dostupné pomocí Telosys objektů. Globální objekty $model a $entity zpřístupňují informace o aktuálním modelu a aktuální entitě. Pomocí nich se lze dostat k ostatním objektům, jako $attribute a $link. [11]

4.3 Vztahy mezi entitami

Vztah mezi dvěma entitami lze definovat dvěma způsoby. První z nich je popis pomocí linku ukazujícího na jinou entitu. Druhým způsobem je definování cizího klíče. Cizí klíč lze definovat pomocí anotace @FK, který ukazuje na jinou entitu nebo její specifický atribut. Tyto dva způsoby lze kombinovat. V případě VTL Telosys objektů jsou vztahy reprezentovány 6 objekty: $fk, $fkAttribute, $fkPart, $link, $linkAttribute a $reference. Některé z těchto objektů, tedy $fk, $fkAttribute a $fkPart jsou závislé na cizím klíči a nejsou ovlivněny linky. Další objekty $link, $linkAttribute a $reference jsou zase závislé na linku a nejsou ovliv-

něny cizími klíči. Cizí klíč je v šablonách reprezentován objektem $fk, který obsahuje až n $fkAttribute. Objekt $fkAttribtue reprezentuje jeden atribut tvořící klíč. $fkPart je jiný pohled na atributy cizího klíče. Objekty $fkPart dostaneme jako odpověď na dotaz, kterých cizích klíčů je daný atribut součástí. Tyto jiné pohledy je další výhoda Telosys oproti Visual Paradigm. Link je v šablonách reprezentován objektem $link. $linkAttribute jsou atributy identifikující instanci entity dostupnou přes tento link. $reference je jiný pohled na linky, a to z pohledu entity. Objekty $reference jsou vráceny při dotazu, se kterými entitami je tato entita propojena a kolik a jaké spojení existují. [10] [5]

Vzhledem k nejasné dokumentaci a malému počtu oficiálních příkladů [6] bylo složité pochopit co je nebo co není možné při definici vztahu. Po důkladném prostudování dokumentace k Telosys VTL objektům a experimentování s nimi je nyní jasné, že jednotlivé anotace (až na výjimky) jednoduše pouze nastavují hodnotu nějakého VTL atributu na zadanou hodnotu. Následující odstavce budou popisovat, jak je možné definovat základní typy vztahů s co možná nejvíce informacemi, které lze ke vztahům v Telosys přidat (seznam testů a získané znalosti jsou v příloze).

```
// obsah souboru Auto.entity
Auto {
  id : int {@Id};
  idMotor : string {@FK(FK_AUTO_MOTOR, Motor)};

  motor : Motor {@OneToOne @LinkByFK(FK_AUTO_MOTOR)};
}

// obsah souboru Motor.entity
Motor {
  id : string {@Id};

  auto : Auto {@OneToOne @MappedBy(motor)};
}
```

Obrázek 4.2: Ukázka 1:1 vztahu v Telosys DSL

1:1 vztah lze nejjednodušeji definovat pomocí linku s anotací @OneToOne v obou entitách. Pokud by se anotace @OneToOne vynechala v jednom směru byl by daný link veden jako N:1 nebo 1:N podle zdali je nebo není kolekce. Tento vztah lze nyní rozšiřovat pomocí dalších anotací. Uživatel tak může například spojit link s cizím klíčem nebo definovat inverzní stranu (obrázek 4.2). Pokud je cizí klíč v entitě již definován pomocí anotace @FK, potom je propojení s linkem nejjednodušeji dosaženo pomocí anotace @LinkByFK. V případě, že cizí klíč definovaný není, ale přesto chceme propojit atributy cizího klíče s linkem, použijeme anotaci @LinkByAttr. Tato anotace vytvoří na pozadí cizí klíč s defaultním jménem. Cizí klíče definované pomocí @FK nemusí mít jméno, to jim Telosys přidělí automaticky. Jméno vznikne spojením jmen vlastnící a odkazované entity. Jméno ovšem musí být unikátní, což je problém, pokud uživatel chce definovat více cizích klíčů v stejné entitě ukazujících na stejnou entitu (viz. příloha B). Obě strany 1:1 vztahu jsou defaultně vlastnící strany (viz. příloha B). Inverzní strana je definována pomocí anotace @MappedBy. [11]

[5] Tato infromace je na stránce https://www.telosys.org/doc/latest/objects/

```
// obsah souboru Kolo.entity
Kolo {
  id : short {@Id};
  idAuto : int {@FK(FK_KOLO_AUTO, Auto)} ;

  auto : Auto {@LinkByFK(FK_KOLO_AUTO)};
}

// obsah souboru Auto.entity
Auto {
  id : int {@Id};

  kola : Kolo[] {@MappedBy(auto)};
}
```

Obrázek 4.3: Ukázka 1:N vztahu v Telosys DSL

N:1 a 1:N jsou výchozí pro vztahy. N:1 je dosažena definováním jednoduchého linku, jak je zmíněno v předchozím odstavci. Pro 1:N musíme definovat link jako kolekci. Standardní link, který byl zmiňován doposud je v dokumentaci popsán jako reference na entitu s kardinalitou „0..1“. Kolekce je oproti tomu reference na entitu s kardinalitou „0..*“. Ukázka vztahu 1:N je na obrázku 4.3. Pro spojení s cizím klíčem platí stejná pravidla jako pro 1:1 vztah. Oproti 1:1 se tento vztah, ale liší ve výchozím chování k inverzní straně, kdy link-kolekce je defaultně brán jako inverzní strana (viz. příloha B). Anotace @MappedBy je i tak užitečná, protože jasně spojí inverzní link s linkem na druhé straně. [11]

Poslední neprobraný vztah je M:N (viz. obrázek 4.4). Celkově se M:N velmi podobá 1:1. Pro úspěšné definování M:N musí mít linky anotaci @ManyToMany. Obě strany jsou defaultně vlastnící strany (viz. příloha B) a anotace @MappedBy je nutná pro definování inverzní strany. Při spojování anotací s cizími klíči se využívá tzv. JoinEntity reprezentující spojovací tabulku v databázi. Cizí klíče se definují v entitě anotované @JoinEntity a linky se s touto entitou propojí pomocí @LinkByJoinEntity. Vztah lze definovat i bez spojovací tabulky. Při definovaní vztahu M:N bez spojovací tabulky ovšem nemůžeme použít anotace @LinkByJoinEntity a @JoinEntity. Velmi obtížné bude i nalezení místa pro cizí klíče. Vztah bude tak definován pouze pomocí linku a ostatních anotací.[11]

Zatím byla definice vztahu probírána pro oboustranné vztahy. Jednostranný vztah se liší pouze tím, že na jedné straně není uveden link. Problém nastane pouze pokud je nutné vytvořit jednostranný vztah s linkem na inverzní straně. Díky výchozímu chování vztahů M:N a 1:1 je tato strana označována za vlastnící. Anotaci @MappedBy nelze použít, protože chybí link na protější straně, který anotace potřebuje. [11]

Kromě výše zmíněných anotací existují pro linky ještě jiné anotace. Většina z nich je jednoduchá, a to jak ze syntaktického, tak i sémantického hlediska. Anotace jako @Insertable a @Updatable akceptují na rozdíl od většiny anotací boolean hodnotu místo obvyklého textového řetězce. Neobvyklá je také anotace @Cascade, která akceptuje jeden a více hodnot. Všechny její hodnoty musí být ze seznamu „Valid options“. [11]

```
// obsah souboru Auto.entity
Auto {
  id : int {@Id};

  vlastnici : Osoba {@ManyToMany @LinkByJoinEntity(AutoOsoba)};
}

// obsah souboru AutoOsoba.entity
@JoinEntity
AutoOsoba {
  idAuto : int {@Id @FK(FK_OSOBA_AUTO, Auto)};
  idOsoba : int {@Id @FK(FK_AUTO_OSOBA, Osoba)};

  vlasnilOd : date ;
  vlasnilDo : date ;
}

// obsah souboru Osoba.entity
Osoba {
  id : int {@Id};
  vlastnenaAuta : Auto[] {@ManyToMany @MappedBy(vlastnici)
    @LinkByJoinEntity(AutoOsoba)};
}
```

Obrázek 4.4: Ukázka N:M vztahu v Telosys DSL

4.4 Jiné anotace

Aktuální verze Telosys DSL, tedy 4.1.0 obsahuje 60 anotací. Anotace související se vztahy a linky byly probrány v předchozí kapitole. V této se zaměřím na ostatní anotace. Vzhledem k počtu anotací ovšem nebudu probírat každou zvlášť, pouze skupiny významově nebo účelově podobných anotací nebo anotace, které se syntaxí významně liší a mohou tak být problematické z pohledu návrhu pluginu. Toto rozdělení není nijak oficiální a slouží jen jako rychlý přehled. [11] První v těchto kategorií je skupina anotací ovlivňující typ atributů v cílovém jazyce. Patří sem například @ObjectType, @PrimitiveType, @UnsignedType nebo @NotNull. Podle vybraného cílového jazyka při překladu neutrálního typu atributu a těchto anotací se atributu přiřadí typ v cílovém jazyce. [11]

Další skupina, kterou je možno nalézt je skupina omezující hodnotu atributů. Sem můžeme zařadit @Future, @NotEmpty, @Min, @Unique, ale i anotaci @Pattern specifikující vzor pro řetězce. Jaký jazyk se pro popis vzoru použije je na uživateli, jako příklad je v dokumentaci uveden RegEx. [11]

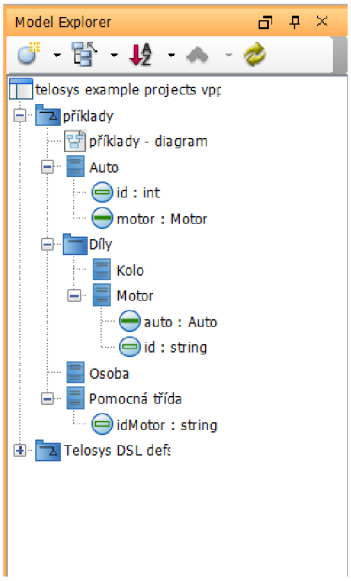
Jména databázových anotací začínají na Db, tedy @DbCatalog, @DbName, @DbComment, @DbType atd. @ReadOnly, @Extends a @Abstract jsou pro koncepty z Objektově orientovaného programování. Poslední jasně rozlišitelné skupina nastavuje hodnotu atributu. Zde lze najít anotace jako @AutoIncremented, @GeneratedValue nebo @InitialValue. Z pohledu syntaxe je zajímá @GeneratedValue. První zadaná hodnota totiž mění počet parametrů. Z ostatních anotací, které nejsou součástí žádné takovéto skupiny je zajímavá

anotace @Package. @Package přiřazuje entitu do balíčku. Je to jedna z mála anotací, která má v UML jasný grafický ekvivalent[14].[11]

Kapitola 5

Reprezentace Telosys DSL ve Visual Paradigm

První část úkolu této práce je navrhnout grafickou reprezentaci Telosys DSL pro Visual Paradigm. Jazyk UML [14] byl vybrán díky velmi detailní specifikaci a podobnosti základních prvků Telosys DSL (model, entita, atribut) s prvky v jazyce UML (model, třída, atribut třídy).



Obrázek 5.1: Ukázka hierarchie modelů ve Visual Paradigm projektu

Hierarchii projektu s Telosys diagramy ukazuje obrázek 5.1. Jak již bylo naznačeno výše, Telosys model bude reprezentován pomocí UML modelu. Všechny entity, které mají být součástí tohoto modelu musejí být definovány uvnitř tohoto UML modelu. Díky podobnostem mezi prvky lze Telosys entity a atributy jednoduše přenést do diagramu tříd. Entita je reprezentována třídou a atribut entity atributem třídy. Telosys obsahuje více datových typů než UML, naštěstí Visual Paradigm umožňuje vytvoření vlastního jazyka a v něm

definici datových typů [24] [1]. Projekty by tak měli jako svůj jazyk používat „Telosys DSL“ nebo podobně pojmenovaný jazyk s všemi neutrálními typy.

5.1 Využití ERD

Entity-relationship diagram neboli ERD je diagram modelující data a vztahy mezi nimi Používá se hlavně při návrhu databází a datově náročných systémů. Skládá se z entit a entitních množin reprezentující objekty a skupiny podobných objektů. Mezi entitami se definují vztahy reprezentující vazbu nebo závislost. Entity obsahují atributy. Některé atributy v každé entitě tvoří primární klíč, který jednoznačně identifikuje entitu v entitní množině. Dodatečné informace mohou být poskytnuty ve formě omezení, kardinality a cizích klíčů. Cizí klíč je definován nad jedním nebo více atributy, jejichž typy odpovídají atributům primárního klíče jiné entity. [4]

Entity, atributy a cizí klíče z Telosys DSL jsou v ERD přítomny. Jediný problém by byla reprezentace linku. Anotace a tagy by se mohli reprezentovat jako omezení. Jak bude v následujících kapitolách detailněji popsáno bylo nakonec rozhodnuto Telosys diagram modelovat na základě diagramu tříd. Důvodem je právě reprezentace linku, anotací a tagů. UML obsahuje rozšiřující prvky a jejich modely obsahují spoustu vlastností, které mohou být při modelování anotací využity. V diagramu tříd lze navíc definovat atribut, který má jako datový typ uživatelem definovanou třídu.

5.2 Vztahy mezi entitami

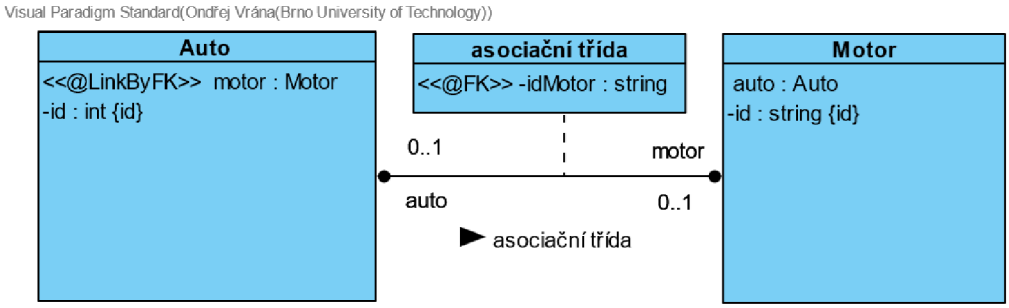
O něco složitější je reprezentace vztahů mezi entitami. Asi nejjednodušší způsob reprezentace linku je pomocí atributu s datovým typem třídy, na kterou link ukazuje. Dále by pouze stačilo vyřešit anotace. Toto řešení je sice jednoduché na pochopení ale není o moc přehlednější než vztahy definované textovým zápisem. K tomu, aby uživatel zjistil, zda jsou dvě entity propojeny musí projít seznam atributů obou modelujících tříd. V diagramu s více vzájemně propojenými třídami uživatel rychle ztratí přehled. Jediné pozitivum by v takovém případě bylo, že jsme schopni dostat více entit na jednu obrazovku vedle sebe.

Nakonec bylo k reprezentaci vztahů použito jiné, ale podobné řešení. Základem je asociace mezi dvěma třídami. Asociace graficky spojí obě entit. Uživatel pak na první pohled vidí souvislost mezi entitami a nemusí je díky tomu prohledávat. Asociace sama o sobě ovšem nenese žádnou informaci, která by se dala popsat pomocí Telosys DSL. Ty jsou přidány jejím rozšířením a modifikací jejích vlastností.

Asociace se skládá z několika (v případě Telosys vždy jen dvou) konců asociace. Konec asociace je atribut reprezentující jednu ze stran vztahu. Tyto konce asociace mají spoustu vlastností, které můžeme využít. Jednou z těchto vlastností je vlastnictví. V případě binární asociace může být vlastnictví konce asociace nastaveno na třídu místo asociace. Takovíto konec je vždy navigovatelný a je ve třídě reprezentován atributem s datovým typem třídy na opačném konci [14]. Díky tomu máme ve třídě atribut reprezentující link, ale také asociaci, která existenci vztahu vizuálně potvrzuje. Propojení asociace a tohoto atributu umožňuje jednoduše určit i další vlastnosti, jako například multiplicitu nebo vlastnící a inverzní stranu. Visual Paradigm navíc při přidělení vlastnictví konce asociace vytvoří atribut

[1] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/2270/2276/59851_datatype.html

ve třídě automaticky. Uživatel může definovat jednosměrný vztah ponecháním vlastnictví jednoho konce asociaci. Směr čtení asociace je využit k definování vlastnící a inverzní strany.

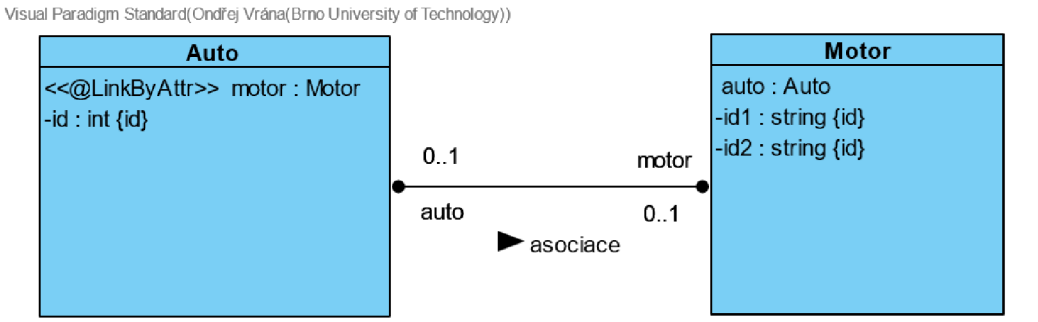


Obrázek 5.2: Telosys diagram se vztahem 1:1

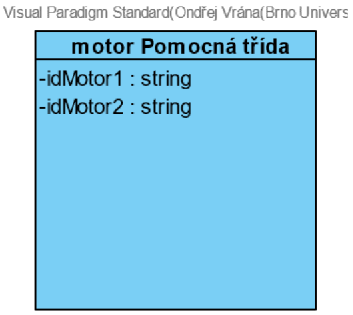
Vztah anotovaný @OneToOne je vytvořen nastavením kardinalit obou konců asociace na „0..1“ (viz. obrázek 5.2). Pro propojení s identifikujícími atributy nebo propojení s cizím klíčem jsou použity stereotypy s označenými hodnotami. Stereotyp @FK je nutný u každého atributu, ze kterého se cizí klíč skládá. Samotné propojení nastává pomocí stereotypu @LinkByFK přiřazenému k linku. Stereotyp ovšem nelze jednomu prvku modelu přiřadit vícekrát. Nelze tedy definovat více cizích klíčů na jednom atributu. Vzhledem k tomu, že cizí klíč se vždy vztahuje pouze k jednomu vztahu byla přidána možnost definování klíče v asociační třídě daného vztahu. Atributy tvořící cizí klíč se definují v asociační třídě dané asociace. Cizí klíč je na nich vytvořen opět pomocí stereotypu @FK. Při překladu se atributy asociační třídy přidají do třídy, která je vlastnící strana asociace. Pokud třída již obsahuje atribut se stejným jménem, nový atribut se ve třídě nevytváří a pouze se do již existujícího atributu přidají anotace a tagy definované na atributu v asociační třídě.

Propojení identifikujících atributů s linkem je dosaženo stereotypem @LinkByAttr. Jeho použití je ukázáno na obrázku 5.3. Většina anotací reprezentovaná stereotypy má označené hodnoty, které odpovídají parametrům anotace. Proměnný počet parametrů @LinkByAttr takto vytvořit nelze, protože neexistuje způsob, jak vytvořit stereotyp s proměnným počtem označených hodnot. Z toho důvodu je vytvořena pomocná třída (podobná jako na obrázku 5.4), jejíž atributy modelují jednotlivé parametry anotace. Stereotyp pak obsahuje jednu označenou hodnotu, která ukazuje na tuto pomocnou třídu. Při překladu do Telosys DSL je tato pomocná třída sjednocena s třídou, ve které je použita podobně jako asociační třídy zmíněné výše. Pokud neexistuje atribut se stejným jménem je do třídy přidán, a pokud existuje jsou přidány pouze anotace a tagy. Pomocná třída je vytvořena jako vnitřní třída třídy, ve které je použita. Je dobré takové třídy zakreslit do vlastních diagramů. Diagramy tříd s entitami tak zůstane přehlednější. Navíc bude díky tomu při překladu jednoduché rozlišit co je entita a co jen pomocná třída.

Vztah 1:N a N:1 jsou vztahy, kdy jedna strana je kolekce a zároveň není použita ani jedna z anotací @OneToOne a @ManyToMany[11]. Podobně jako vztah 1:1 z předchozích odstavců je tento vztah definován nastavením multiplicity. Zde musí být kardinalita jednoho konce asociace nastavena na „0..1“ a druhý na „0..*“ (viz. obrázek 5.5). Propojení atributů s linkem je opět možné pomocí cizího klíče (stereotypy @FK a @LinkByFK) nebo přímo stereotypem @LinkByAttr a pomocnou třídou. Při definování inverzní a vlastnící strany by strana s linkem-kolekcí měla vždy být inverzní a neměla by obsahovat atributy cizího klíče. Při překladu musí existovat kontrola, která vypíše chybovou hlášku.



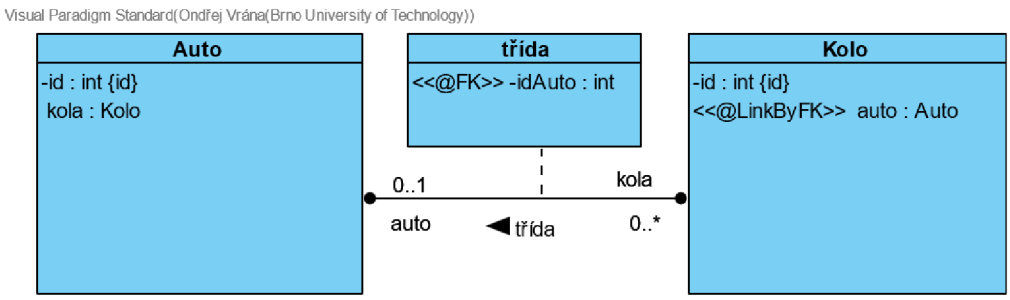
Obrázek 5.3: Telosys diagram využívající @LinkByAttr a pomocnou třídu



Obrázek 5.4: Pomocná třída pro @LinkByAttr ve vlastním diagramu.

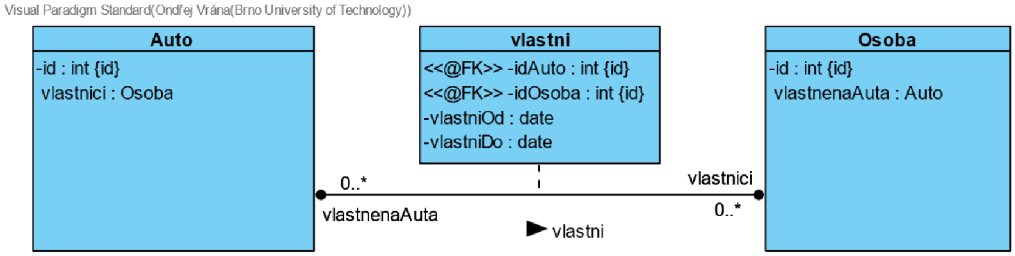
M:N je vztah, kdy oba linky jsou kolekce a zároveň je použita anotace @ManyToMany [11]. V Telosys diagramu se tento vztah reprezentuje jako binární asociaci, kde oba konce asociace mají kardinalitu „0..*“ (viz. obrázek 5.6). Spojení atributů s linkem funguje jinak než u předchozích typů vztahů. Speciální pro M:N je možnost vytvořit entitu pro ukládání spojení [11] [2] . Tato entita je anotována @JoinEntity. Ve Visual Paradigm je “Join Entity” vytvořena z asociační třídy asociace s multiplicitou obou konců „0..*“. V takovémto případě se asociační třída se nebude sjednocovat s žádnou entitou a bude při překladu vlastní entita s anotací @JoinEntity. Případné linky budou automaticky anotovány @LinkByJoinEnity bez nutnosti označení v atributech konců asociace. Anotace @LinkByJoinEntity nebude

[2]Tato infromace je na stránce https://doc.telosys.org/dsl-model/annotations



Obrázek 5.5: Telosys diagram se vztahem 1:N

mít vlastní reprezentující stereotyp, protože bude vždy generována automaticky. Směr asociace nesjednocuje asociační třídu s ostatními třídami. V tomto případě pouze rozhoduje o vlastnící a inverzní straně vztahu, tedy přiřazení anotace @MappedBy.



Obrázek 5.6: Telosys diagram se vztahem N:M a Spojovací tabulkou

5.3 Anotace

Již byla probrána reprezentace vztahů a základních prvků Telosys DSL ve Visual Paradim. Nyní budou vysvětleny, jakým způsobem jsou reprezentovány anotace a zmíním řešení složitějších anotací. Anotace probírané v předchozí kapitole, tedy @FK, @LinkByAttr, @LinkByFK, @LinkByJoinEntity, @OneToOne, @ManyToMany a @MappedBy, jsou již probrané, a proto budou v této kapitole zmíněny maximálně okrajově.

UML modely obsahují spoustu vlastností, které mají stejný význam jako některé anotace Telosys DSL. Vlastnosti jako Abstract u třídy nebo Id u atributu. Ve Visual Paradigm jsou dostupné v okně Specification. Zbytek je řešen pomocí stereotypů se označenými hodnotami. Některé anotace mají význam omezení například @Future značící, že datum zapsané do atributu musí být v budoucnosti nebo @Max stanovující maximální číselnou hodnotu, které atribut může nabývat [11] [3] . Původně se všechny podobné anotace měli modelovat za pomocí omezení v UML (anglicky constraints). Omezení ovšem nemají označené hodnoty, pomocí kterých by se dali zapsat hodnoty parametrů, a proto se nakonec pomocí omezení modelovat pouze anotace bez parametrů. Jedná se o anotace @Future, @InMemoryRepository, @NotBlank, @NotEmpty, @NotNull, @Past a @ReadOnly. Telosys také umožňuje vytváření nových uživatelský značek označovaných jako tagy. Pokud chce uživatel použít tag v Telosys diagramu musí definovat nový stereotyp začínající na znak #. V definici stereotypu může přidat maximálně jednu označenou hodnotu. Její typ musí být Text nebo Model Element. Ostatní typy nemají smysl, protože v Telosys DSL je hodnota tagu pouze řetězec, který se nijak nekontroluje. Typ Text je pro takovou hodnotu ideální. Může ale nastat situace, kdy uživatel chce využít hodnotu tag k tomu, aby ukázal na entitu nebo atribut. V takovém případě by dodatečná kontrola existence odkazovaného modelu mohla být užitečná.

Všechny anotace popisované stereotypy začínají znakem @. Ve většině případů je jméno anotace shodné se jménem stereotypu. Stereotypy, které modelují anotace s parametry mají označené hodnoty. Většina označených hodnot je typu Text a hodnota zapsaná do nich není nijak kontrolována. Existují anotace, které jsou více specifické a jejich parametry mají specifický typ. Všechny typy označených hodnot ve Visual Paradigm, které se používají jsou:

[3] Tato infromace je na stránce https://doc.telosys.org/dsl-model/annotations

Text, Model Element, Floating Point Number, Integer a Boolean. Model Element je ukazatel na jiný prvek modelu. Využívají ho stereotypy: @LinkByAttr, @Cascade a @FK. Typ Floating Point Number je využit pro stereotypy @Min a @Max. Integer se vyskytuje u @MaxLen, @MinLen, @Size, @SizeMax a @SizeMin a Boolean u @Insertable a @Updatable. Jména označených hodnot jsou převzata z dokumentace [11] nebo vymyšlena, pokud jméno v dokumentaci není. Pro anotace s jedním parametrem je jméno označené hodnoty jméno stereotypu bez @ a počáteční písmeno je malé. Pokud je označená hodnota vedena v dokumentaci jako volitelná, pak ji uživatel nemusí vyplnit.

Některé anotace nelze jednoduše přenést do stereotypu s označenými hodnotami. Může za to proměnný počet parametrů anotace nebo zvláštní datový typ parametru. @LinkByAttr byl zmíněn již dříve, ale existují i další.

@Cascade může mít jeden až 4 parametry [11]. Každý z parametrů nabývá jedné z hodnot tzv. „Valid Options“. Toto vytváří podobný problém jako u @LinkByAttr s proměnným počtem parametrů, a navíc jsou hodnoty parametrů vybírány z předdefinovaného seznamu hodnot. Stereotyp @Cascade tedy bude ukazovat na pomocnou třídu. Tato třída bude obsahovat atributy, každý z nich modeluje jeden parametr. Model pomocné třídy bude podobně jako u @LinkByAttr uvnitř třídy, pro kterou je určen. Všechny akceptované hodnoty z „Valid Options“ budou definovány jako hodnoty výčtového typu. Všechny atributy pomocné třídy mají za typ tento výčtový typ a jejich počáteční hodnota je jedna z hodnot výčtového typu.

Problém s proměnným počtem označených hodnot stereotypů @LinkByAttr a @Cascade má ještě alternativní řešení. Místo aby každá hodnota anotace měla svojí vlastní označenou hodnotu, by anotace obsahovala jen jednu označenou hodnotu pro všechny. Všechny hodnoty by se zapsali do této označené hodnoty jedna za druhou s oddělovačem (například čárkou nebo novým řádkem) mezi nimi. Stávající řešení ovšem poskytuje mnohem větší kontrolu nad uživatelovým vstupem.

Anotace @Size má sice podle dokumentace pouze jeden parametr [11], který je tvořen dvěma čísly oddělenými čárkou. Prakticky ale nelze atribut obalit do uvozovek. Ve výsledku jsou to syntakticky dva samostatné parametry, přičemž druhý z nich je volitelný. Části jsou modelovány jako dvě označené hodnoty typu Integer. Jediný rozdíl mezi touto anotací a jinými syntakticky nevýznamnými anotacemi je volitelný druhý parametr.

Poslední speciální anotace je @GeneratedValue. První parametr @GeneratedValue je typ generované strategie [11]. Strategie je možné vybrat ze 4 možností: AUTO, IDENTITY, SEQUENCE nebo TABLE. Výběr strategie následně ukáže, kolik dalších parametrů anotace může maximálně nabývat. Opět je zde problém proměnlivého počtu parametrů, ovšem nyní se 4 možnostmi, přičemž maximum je vždy pevně stanoveno. Jednodušší než vytvářet novou pomocnou třídu a výčtový typ bude rozdělit tuto anotaci ve Visual Paradigm na 4 stereotypy podle strategie: @GeneratedValueAUTO, @GeneratedValueIDENTITY, @GeneratedValueSEQUENCE a @GeneratedValueTABLE. Každý ze stereotypů bude obsahovat všechny možné označeného hodnoty. Uživatel vyplní pouze ty, které potřebuje a zbytek může nechat nevyplněný. Všechny označené hodnoty jsou typu Text.

Kromě anotací modelovaných stereotypy a omezeními, existují i anotace, které jsou modelovány jinými způsoby. Kromě anotací souvisejících se vztahy sem patří i @Id, @InitialValue, @Package a @Extends. @Id identifikuje atributy primárního klíče. Ve Visual Paradigm je pro něj využita UML „property-modifier“ id. @InitialValue udává počáteční hodnotu atributu a je mapována na počáteční hodnotu atributu třídy. Anotace @Package doplňuje entitu o název balíčku, do které patří. Tato anotace je modelována pomocí UML balíčku [14]. Balíčky ve Telosys diagramu mohou být zanořeny do sebe přičemž výsledná

anotace dostane jako hodnotu konkatenaci jmen jednotlivých balíčku. @Extends je modelováno pomocí generalizace.

5.4 Metamodel

Metamodel je model modelu [9]. Definuje prvky, které jsou poté používány při vytváření modelů [17]. Příklad metamodelu je UML metamodel popsaný v UML specifikaci [14]. Ve specifikaci jsou definovány jednotlivé prvky jako třída, atribut nebo asociace spolu s jejich vlastnostmi a značeními.

Některé prvky používané při vytváření Telosys modelů ve Visual Paradigm nejsou v programu dostupné. Každý projekt tak bude obsahovat metamodel doplňující tyto prvky. Patří sem hlavně omezení modelující anotace a výčtový typ s „Cascade options“. Definované stereotypy a datové typy také patří do metamodelu, ale ty jsou ukládány globálně a mohou být používány ve více projektech bez redefinice.

Kapitola 6

Algoritmus převodu

Algoritmus pracuje s objekty Visual Paradgim a komunikuje s nimi přes jejich rozhraní. Dobrá znalost rozhraní a vazeb mezi třídami může pomoci při návrhu algoritmu. Archiv „openapi.jar“ poskytuje API přístup k celé řadě služeb [24] [1] . Přístup k jednotlivým službám je zajištěn třídou ApplicationManager[24]. Informace, které plugin potřebuje jsou uloženy v aktuálním projektu. Odkaz na něj vrací ProjectManager dostupný právě z ApplicationManager [24]. Nyní s přístupem k objektu aktuálního projektu lze přes rozhraní IProject prohledávat hierarchii projektu a dostat se tak k potřebným informacím.

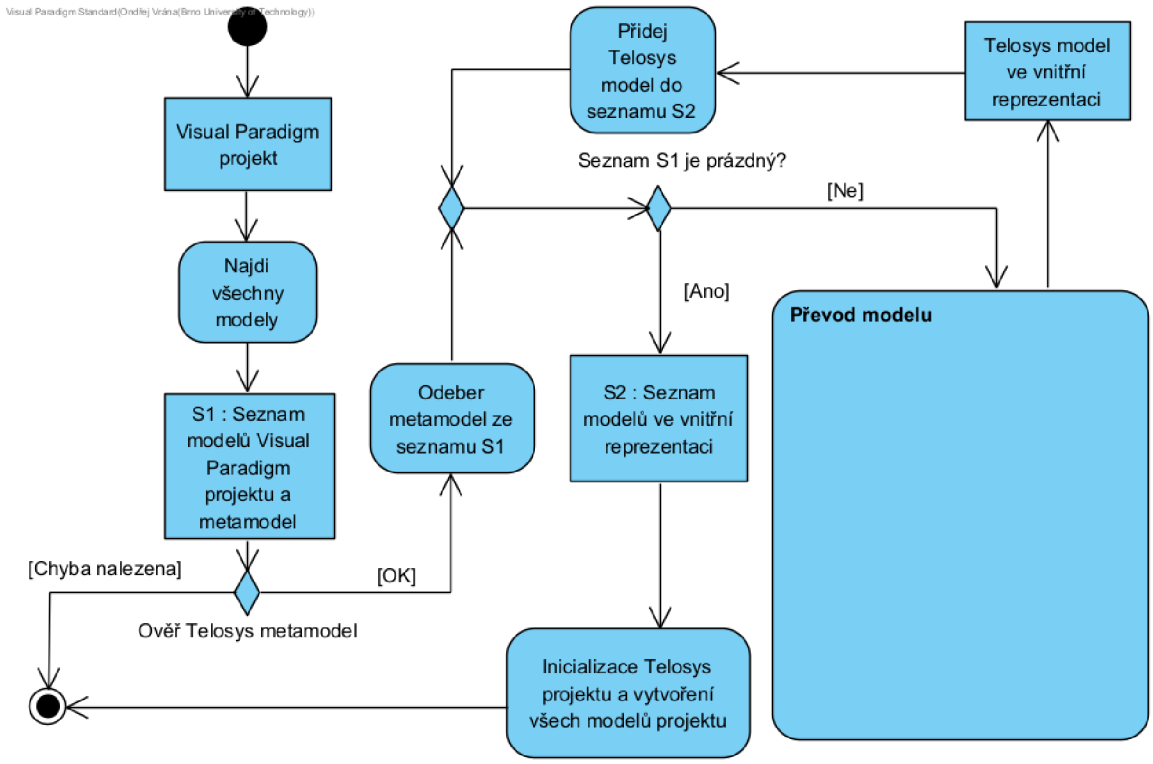
Všechny objekty v projektové hierarchii modelů implementují společné rozhraní IModelElement a speciální rozhraní pouze pro tento typ objektu. Správně vytvořený Visual Paradigm projekt určený pouze pro Telosys diagramy bude na nejvyšší úrovni modelového stromu obsahovat pouze objekty typu IModelElementFactory.MODEL_TYPE_MODEL implementujících rozhraní IModel a několik další objektů, které obsahuje každý prázdný Visual Paradigm projekt. IModelElementFactory.MODEL_TYPE_MODEL objekty již dále obsahují pouze objekty IModelElementFactory.MODEL_TYPE_PACKAGE s rozraním IPackage a IModelElementFactory.MODEL_TYPE_CLASS s rozhraním IClass. Balíček může mít třídy a jiné balíčky. Třída umístěná v balíčku může obsahovat pomocné třídy. Každý IModelElement obsahuje rozhraní pro procházení jejich stereotypů, označených hodnot a omezení. Atributy tříd s rozhraním IAttribute jsou v hierarchii také, tedy vyskytují se mezi potomky objekt třídy. Mezi metody rozhraní IModelElement patří i metody pro práci s potomky. Přetížené verze těchto metod mají jako parametr filtr umožňující výběr pouze určitých typů objektů. [24]

6.1 Algoritmus

Vstupem algoritmu znázorněného na obrázku 6.1 je projekt obsahující překládané modely s Telosys diagramy a metamodel. Algoritmus najde všechny modely v nejvyšší úrovni hierarchie projektu. Následně najde v seznamu modelů projektu metamodel Telosys. Pokud je metamodel nalezen ověří existenci všech jeho prvků. Bez správně definovaného metamodelu nelze pokračovat dále. Metamodel je vyjmut ze seznamu modelů a převod zbytku seznamu může začít.

Každý UML model ve Visual Paradigm ze seznamu je převeden následujícím algoritmem na Telosys model. Převádí se nejprve do vnitřní reprezentace. Hlavními důvody vyu-

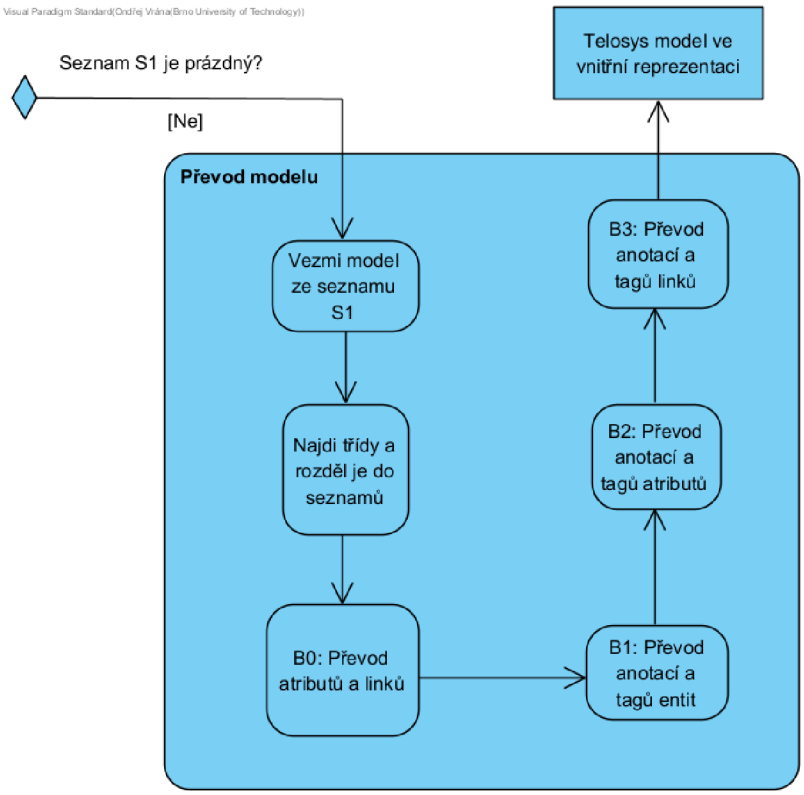
[1] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/pluginjavadoc/com/vp/plugin/package-summary.html



Obrázek 6.1: Algoritmus převodu, přehled

žití vnitřní reprezentace je větší srozumitelnost převodu, jednodušší algoritmus a možnost většího počtu sémantických kontrol. Implementace bude navíc snáze modifikovatelná a rozšiřitelná pro budoucí verze Telosys DSL. Po kompletním převodu všech modelů na vnitřní reprezentaci a všech kontrolách jsou vytvořeny složky a soubory těchto modelů uvnitř inicializovaného Telosys projektu. Pokud Telosys projekt již obsahuje model se stejným jménem budou soubory uvnitř nahrazeny novými. Soubory, u kterých nedochází ke konfliktu budou v modelu ponechány. Na závěr každého převodu je vypsána statistika varování a chyb, na které se narazilo při převodu.

Následující odstavce budou popisovat přímo algoritmus převodu. Nejprve bude popsán z vyšší úrovně jako sekvence černých skříněk. Poté se detailně vysvětlí jednotlivé černé skříňky. Převod každého modelu začíná nalezením všech tříd v hierarchii modelu. Nalezené třídy jsou roztříděny do tří kategorií: entitní třídy, asociační třídy a pomocné třídy. Rozdíl mezi entitními a asociačními třídami je, že asociační třídy obsahují vazbu na asociaci. Pomocné třídy lze najít jednoduše, protože na rozdíl od ostatních jsou definovány uvnitř jiných tříd. Následuje sekvence černých skříněk, které se provedou pro každou třídu v entitních a asociačních třídách. Pouze až je aktuální černá skříňka dokončena pro všechny třídy se začíná s následující skříňkou. První taková skříňka projde atributy třídy a vytvoří vnitřní reprezentace atributů a linků. Na obrázku 6.2 je označen jako B0. Po provedení B0 mají svojí vnitřní reprezentaci všechny entity, atributy a linky. Zbývá tedy pouze převod stereotypů, omezení a jiných vlastností na anotace a tagy. Následující skříňka B1 se zabývá anotacemi a tagy entit. B2 zpracovává anotace a tagy pro atributy a B3 pro linky. Důvod rozdělení do tří skříněk je lepší přehlednost a modifikovatelnost algoritmu. Na závěr pře-



Obrázek 6.2: Diagram aktivit, detail aktivity Převod modelu

vodu modelu se provedou kontroly, které nelze udělat bez dostupnosti všech informací nebo takové kontroly které by byly příliš neefektivní.

Existuje důvod, proč se atributy nezpracovávají při nalezení třídy. Entita, na kterou link ukazuje by jinak ještě nemusela mít vnitřní reprezentace. Vytvoření vnitřní reprezentace linku by se tak musel rozdělit na dvě části. V první části by se pouze vytvořil objekt link. Až v druhé části by se objekt linku propojil s objektem entity, čímž by se zároveň dokázalo, že entita, na kterou ukazuje skutečně v modelu existuje. Atributy by se mohli zpracovávat při nalezení třídy. Jediný rozdíl mezi Telosys atributem a Telosys linkem ve Visual Pardigm je, že link je atribut představující konec asociace. Seznam atributů třídy by se musel procházel dvakrát.

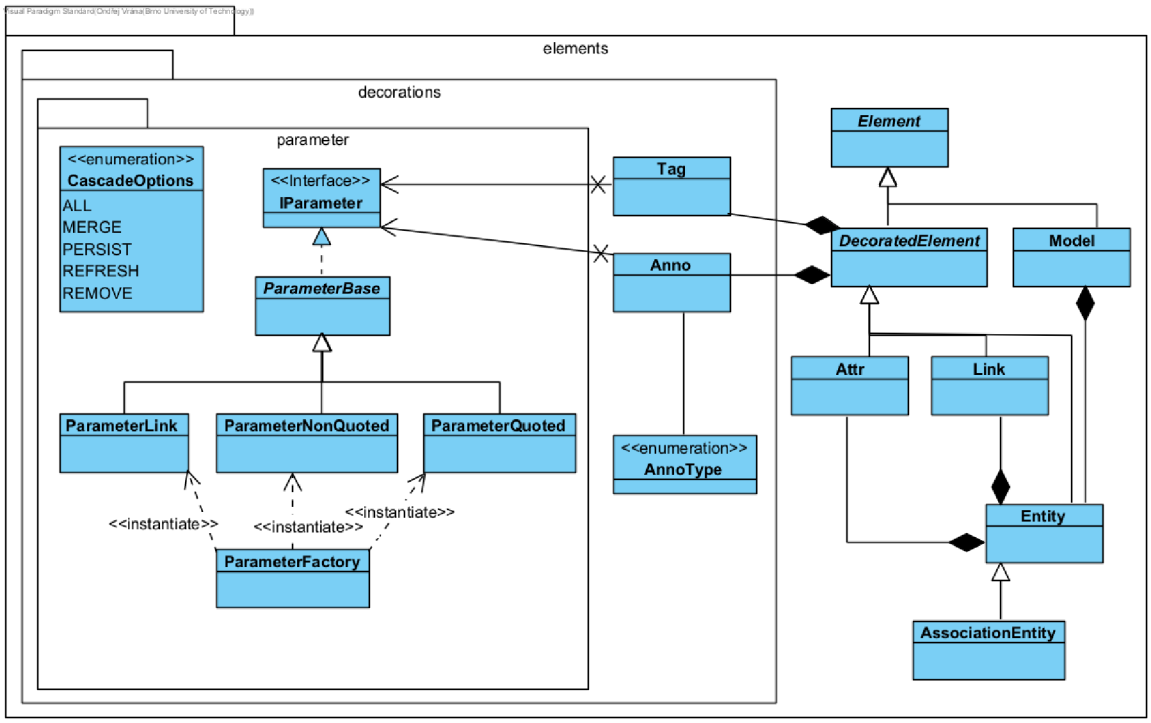
Skříňky B1, B2 a B3 jsou si velmi podobné. Anotace budou hledány mezi stereotypy, omezeními a vlastnostmi UML modelů. Tagy jsou všechny stereotypy jejichž jména začínají na #. Vlastnosti, které nejsou využívány v Telosys diagramu jsou ignorovány. Jejich změna nevypíše žádné varovné hlášení. Neznámé omezení vypíší varovnou hlášku. Totéž platí i pro neznámé stereotypy nebo známé stereotypy, které obsahují jinou chybu. Mezi tyto jiné chyby patří například špatný počet nebo jména označených hodnot, neuvedení hodnoty ve označené hodnotě, která je povinná atd.. Výskyt chyb nezastaví celý převod, ale pouze vynechá převod prvků, které chybu obsahují. Na pořadí převodu prvků v rámci iterace jedné skříňky nezáleží.

Skříňka B1 se zabývá anotacemi a tagy entit. To tedy znamená všechny anotace vedené v Telosys DSL dokumentaci [11] jako „Scope: entity“ s výjimkou @JoinEntity a @Packge. @Abstract, @Extends jsou hledány ve vlastnostech modelů, @InMemoryRepository a @Rea-

dOnly ve stereotypech i v omezeních. Ostatní jsou mezi stereotypy. B2 zpracovává dekorace atributů. Patří sem všechny anotace vedené v dokumentaci jako „Scope: attribute“[11]. @Id a @IntialValue jsou hledány ve vlastnostech modelů. @Future, @NotEmpty, @NotBlank, @NotNull mají definované omezení. @DbComment je jak pro atribut i entitu, proto je převáděn v obou skříňkách. B3 se zabývá anotacemi a tagy definovaných pro linky, tedy „Scope: link“. Výjimkou je @JoinEntity jako anotace přidělená asociační třídě, pokud je kardinalita obou stran asociace „0..*“. S kardinalitami, asociacemi a asociačními třídami se pracuje v této části. Dává tedy větší smysl tuto anotaci řešit zde. @ManyToMany a @OneToOne jsou hledány ve vlastnostech. B3 nezpracovává žádné omezení.

6.2 Datové struktury

Jak již bylo zmíněno dříve plugin nejprve převede všechny modely z reprezentace ve Visual Paradigm do vnitřní reprezentace. digram tříd vnitřní reprezentace je znázorněn na obrázku 6.3. Vnitřní reprezentace umožňuje jednodušší algoritmy převodu, přehlednější kód a tím i jednodušší modifikace. Z vnitřní reprezentace se následně vytváří soubory Telosys projektu.



Obrázek 6.3: Datové struktury vnitřní reprezentace Telosys DSL

Abstraktní třída Element obsahuje společný základ pro třídy Model, Entity, Attribute a Link. Důležité je Id Visual Paradigm prvku reprezentující daný prvek v diagramech. Z modelu se v průběhu převodu postupně čtou další informace a je tedy důležité, aby objekt věděl, k jakému prvku modelu patří. Link obsahuje identifikační řetězec dvou modelů. První je identifikační řetězec atributu konce asociace uložený ve vPId. Navíc potřebuje i identifikační řetězec asociace. Model dědí přímo od Element. Entity, Attribute a Link dědí od jeho potomka DecoratedElement. Abstraktní třída DecoratedElement implemen-

tuje rozhraní pro práci s anotacemi a tagy. Obsahuje seznam objektů Annotation a seznam objektů Tag, se kterými se manipuluje pomocí rozhraní. Pro samotné hodnoty parametrů anotací a tagů se používají třídy dědící z abstraktní třídy ParameterBase. Třídy se liší způsobem serializace. ParameterQuoted zabalí vypisovanou hodnotu do dvojitých uvozovek. ParameterNonQuoted vypíše pouze hodnotu a ParameterLink vypíše jméno odkazovaného DecoratedElement. Jednotlivé „Cascade Options“ jsou implementovány výčtovým typem CascadeOptions. AssociationEntity byla do návrhu přidána později, když se zjistilo, že propojení asociační třídy a asociace není ve Visual Paradigm z asociační třídy jasné. Instance této třídy proto obsahují všechny důležité informace z asociace.

Kapitola 7

Implementace

Po analýze problému a návrhu algoritmu a datových struktur je na řadě implementace pluginu. Používaná verze Javy je Java 8. Plugin přeložený v novější verzi Visual Paradigm odmítne. Vybraný systém pro správu verzí je git, díky dřívějším pozitivním zkušenostem. Kód je dostupný z github repozitáře[1]. Původně byl jako nástroj pro automatizaci sestavení používán maven [21]. Kvůli problémům s přidáním knihoven, které nejsou součástí maven repozitářů se, ale projekt později předělal na gradle[5]. Gradle zjednodušil i řadu jiných akcí jako zabalení do zip archivu. Výsledný archiv obsahuje všechny .class přeložené soubory, konfiguraci pluginu plugin.xml, adresář lib se všemi závislými knihovnami a ostatní soubory, které plugin potřebuje jako například ikony pro tlačítka.

```
<plugin
  id="com.vp.plugin.sample.scd"
  name="Generate Sample Communication Diagram"
  description="Generate Sample Communication Diagram"
  provider="Visual Paradigm"
  class="com.vp.plugin.sample.scd.GenerateSCDPlugin">

  <actionSets>
    <actionSet id="com.vp.plugin.sample.scd.actionset">
      <action
        id="com.vp.plugin.sample.scd.actionset.GenerateSCDAction"
        label="Generate Sample Communication Diagram"
        menuPath="Modeling/#"
        icon="icons/generateSCD.png">
        <actionController
          class="com.vp.plugin.sample.scd.actions.GenerateSCDActionCtrl"/>
      </action>
    </actionSet>
  </actionSets>
</plugin>
```

Obrázek 7.1: Upravený plugin.xml z ukázkového pluginu ”Create Communication Diagram” na stránkách Visual Paradigm

[1] https://github.com/myval2012/vpTelosysPlugin

Před začátkem vývoje pluginu musí vývojář definovat minimálně 2 soubory. Tyto soubory musí mít každý plugin a tvoří základ pluginu. První z nich je konfigurační soubor plugin.xml. plugin.xml obsahuje základní informace o pluginu jako například jméno, id, popis atd. a hlavně definici uživatelského rozhraní. Ukázka plugin.xml je na obrazku 7.1. plugin.xml byl převzat z ukázkového pluginu na stránkách Visual Paradigm [24] [2]. UI obsahuje 3 typy akcí. Prvním typem jsou bezkontextové akce spouštěné přes menu. Kontextové akce se nachází ve vyskakovacím menu, které se objeví při pravém kliknutí na prvek diagramu. Při volání kontextové akce se metodě navíc předá action.VPContext, tedy prvek (prvek modelu, diagram nebo prvek diagramu [24] [3]), nad kterým je akce volána. Posledním typem jsou kreslící akce „drawShape“ pro vykreslení vlastních prvků diagramu. Každá akce musí mít přiřazený kontroler, který implementuje funkcionalitu akce v java kódu. Pro bezkontextovou menu akci je kontroler třída implementující action.VPActionController, pro kontextovou akci action.VPContextActionController a pro kreslící akci diagram.VPShapeController. VPActionController obsahuje dvě metody: performAction() a update(). Metoda perfromAction() je zavolána při stisku tlačítka, aby vykonala požadovanou funkcionalitu. Metoda update je volána při otevření menu. Mohu tak povolit nebo zakázat akci plugin například když za současné situace akce nemá smysl nebo by vyústila v chybu. Druhým nutným souborem pluginu je .java soubor s třídou implementující VPPlugin. Třída musí být v pluginu i kdyby byla prázdná. [24] [4]

Visual Paradigm rozlišuje mezi prvky modelu a prvky diagramu. Prvek modelu obsahuje informace o modelu, které nezávisí na grafické reprezentaci. Jedná se například o jméno, popis, prvky modelu dostupné z tohoto modelu a další vlastnosti. Prvek diagramu je grafická reprezentace prvku modelu používaná v diagramech. Obsahuje informace ovlivňující vzhled a umístění prvku v diagramu. Informace uložené v prvku diagramu jsou například souřadnice v diagramu, rozměry a barva. [24] [5]

Visual Paradigm umožňuje ladit pluginy za použití IntelliJ IDEA [24] [6] . Stačí přidat několik .jar knihoven z Visual Paradigm adresáře do externích knihoven projektu a vytvořit konfiguraci.

Při vývoji pluginu byly využity dvě knihovny: openapi od Visual Paradigm a telosys-tools-all [7] od Telosys. openapi se nachází ve složce lib v adresáři instalace programu Visual Paradigm [24] [7] . Nicméně tato cesta získání knihovny není ideální. Lepší by bylo stažení knihovny ze serveru s maven repozitáři například[8]. Takto by klonování github repozitáře projektu a přechod na novou verzi API bylo jednodušší. Telosys api z knihovny telosys-tools-all na serverech s maven repozitáři také není. Tuto knihovnu lze získat naklonováním a přeložením github repozitářů Telosys. Visual Paradigm za roky nasbíralo spoustu materiálů a dokumentace k tvorbě pluginů. Hlavní materiál, ze kterého se při tvorbě tohoto

[2] ukázkový projekt je dospný z https://www.visual-paradigm.com/support/documents/pluginsample.jsp

[3] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/pluginjavadoc/overview-summary.html

[4] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/pluginuserguide/2186/2188/57181_implementing.html

[5] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/pluginuserguide/2186/2187/57180_introduction.html

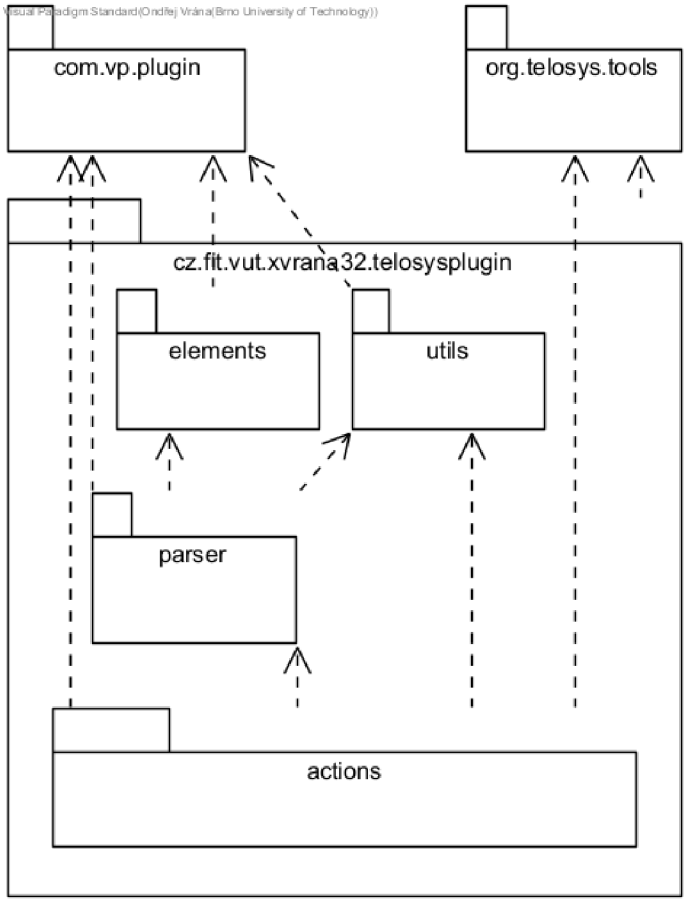
[6] Tato infromace je na stránce https://knowhow.visual-paradigm.com/openapi/debug-plugins-ij/

[7] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/124/254/7040_implementing.html

[8] https://mvnrepository.com/

pluginu čerpalo je javadoc dokumentace openapi [24] [9]. Dokumentace obsahuje všechny rozhraní a práce s nimi je intuitivní. Pouze malá část metod ovšem obsahuje komentáře, tedy pokud není účel metody jasný nezbývá než ji vyzkoušet. Plugin byl vytvořen podle návodu v uživatelské příručce „Extend functionalities with Open API“. Další užitečné zdroj je Visual Paradigm fórum [24] [10] a KNOW-HOW sekce na stránkách Visual Paradigm [24] [11] . U Telosys taková rozmanitost materiálu k API není. Komentáře v kódu se ukázali být dostačující.

7.1 Telosys plugin



Obrázek 7.2: Balíčky pluginu a jejich závislosti

Celý plugin spadá do balíčku cz.fit.vutbr.xvrana32.telosysplugin a je rozdělen na 5 modulů. závislosti mezi jednotlivými balíčky jsou znázorněny na obrázku 7.2. První modul se skládá ze třídy přímo v kořenovém balíčku. Jedná se o třídu VPPluginClass implementující com.vp.plugin.VPPlugin vyžadovanou pluginem. Modul actions obsahuje kontrolery k akcím. Další modul elements obsahující třídy implementující struktury popsané v kapitole 5.1

[9] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/pluginjavadoc/overview-summary.html

[10] Tato infromace je na stránce https://forums.visual-paradigm.com/

[11] Tato infromace je na stránce https://knowhow.visual-paradigm.com

o datových strukturách. Obsahuje balíček decorations pro vnitřní reprezentaci anotace a tagu. Balíček decorations.parameter obsahuje implementace všech parametrů. Modul parser obsahuje všechny třídy zodpovědné za převod z Visual Paradigm modelů do objektů z modulu elements. Modul obsahuje i všechny informace o stereotypech a jejich převodu na anotace. Tyto informace jsou uloženy v polích deklarací. Třídy využívané pro deklarace jsou v balíčku parser.declarations. Poslední modul je modul utils. Zde je třída Logger pro výpis chybových nebo ladících zpráv, třída Config s konfiguračními proměnnými projektu, a také třída Constants s konstantami.

Zajímavou částí kódu je část modulu parser zodpovědné za převod anotací. Převod anotací je rozdělen podle typu prvku, pro který je anotace určena na tři podobné třídy: AttrDecorationParser, EntityDecorationParser a LinkDecorationParser. Každá třída obsahuje statickou metodu parse() volanou nad atributem, entitou nebo linkem. Metoda postupně převede anotace reprezentované speciálními způsoby, následně stereotypy a nakonec omezení. Pro převod stereotypů i omezení využívají pole deklarací. Každá třída využívá svoje vlastní pole deklarací, tak aby věděly, které anotace jsou povoleny u aktuálně převáděného prvku. Deklarace stereotypů obsahují instanční metodu createAnno() zodpovědnou za převod daného stereotypu a jeho označených hodnot. Pomocné metody pro tento převod jsou v abstraktních třídách AnnoDeclaration a AnnoDeclarationMultiple. Většina stereotypů je takto převedena za pomocí třídy AnnoCommon. Její implementace metody createAnno() neobsahuje žádné zvláštní kontroly. Stereotypy vyžadující speciální úpravy nebo kontroly jako například @Cascade nebo @GeneratedValue mají vlastní implementace.

Každá z deklarací stereotypů dědí z jedné z abstraktních tříd AnnoDeclaration nebo AnnoDeclarationMultiple. AnnoDeclaration implementuje pomocnou metodu getValidTaggedValue(), která vyhledá, zkontroluje a vrátí hledanou označenou hodnotu popsanou v deklaraci. AnnoDeclarationMultiple pak obsahuje metodu findTaggedValues(). Metoda findTaggedValues() je určena pro stereotypy s více označenými hodnotami. Podobně jako getValidTaggedValue() vyhledává označené hodnoty, ale navíc je převede vytvoří pomocí jednoduché tovární metody objekt s rozhraním telosysplugin.elements.decorations.parameter.IParameter z této hodnoty.

Metoda LoopThroughModel() ukázaná na obrázku 7.3 je soukromá metoda třídy ModelParser. Metoda rekurzivně prochází hierarchii modelu s cílem najít všechny třídy, vytvořit jejich reprezentující entity a zařadit je do jednoho z 3 seznamů entit. Metoda také hlídá aktuální balíček pomocí parametru currentPackage. Aktuální balíček je tvořen spojenými jmény všech předchozích balíčků. Rekurzivní volání zajišťuje úložiště pro lokální proměnné, jinak by se metoda musela přepsat za použití fronty nebo zásobníku. Metoda je spouštěna z metody parse(), s počátečními hodnotami parametrů nastavenými na aktuální převáděný model typu IModelElementFactory.MODEL_TYPE_MODEL a prázdným řetězcem jako aktuálním balíčkem. Metoda si vyžádá seznam všech potomků aktuálního modelu. Pro každého potomka typu IModelElementFactory.MODEL_TYPE_PACKAGE přidá název balíčku do aktuálního balíčku a rekurzivně se zavolá nad tímto potomkem. Pro každého potomka typu IModelElementFactory.MODEL_TYPE_CLASS se nejprve rozhodne, zda je asociační třída nebo jen třída pomocí metody sortClass(), a poté prohledá její potomky kvůli pomocným třídám pomocí meotdy addSupportEntities(). sortClass() také vytváří anotaci @Package ze jména aktuální balíčku currentPackage.

Jednoduchá tovární metoda je zjednodušený návrhový vzor tovární metody. Mezi návrhové vzory nepočítá, ale jasně s nimi souvisí. Většinou se používá na místo konstruktoru v případě, že potřebuji větší kontrolu nad vráceným objektem. To může znamenat napří-

```
private void LoopThroughModel(
    IModelElement vPModel,
    String currentPackage) {

  IModelElement[] vPChildren = vPModel.toChildArray();
  for (IModelElement vPChild : vPChildren) {
    String modelType = vPChild.getModelType();

    if (modelType.equals(IModelElementFactory.MODEL_TYPE_PACKAGE)) {
      String nextPackage = currentPackage;

      if (vPChild.getName().isEmpty()) {
        Logger.logW(
          "Name of the package cannot be empty. " +
            "Package name skipped in concatenation.");
      } else {
        nextPackage = currentPackage.isEmpty() ? vPChild.getName() :
          currentPackage + Config.getSeparator() + vPChild.getName();
      }

      LoopThroughModel(vPChild, nextPackage);
    } else if (modelType.equals(IModelElementFactory.MODEL_TYPE_CLASS)) {
      sortClass(vPChild, currentPackage);
      addSupportEntities((IClass) vPChild);
    }
  }
}
```

Obrázek 7.3: Metoda LoopThroughModel()

klad vracení již vytvořeného objektu podobně jako návrhový vzor Jedináček. Na rozdíl od jedináčku ovšem může těchto před vytvořených objektů mít více pro různé situace. [15]

V případě této práce existuje více tříd implementujících rozhraní IParameter. Konkrétně se jedná o ParameterQuoted, ParameterNonQuoted a ParameterLink. Hlavní rozdíl mezi nimi je způsob serializace, tedy implementace metody toString(). Aby bylo rozhodování o tom, který z nich vytvořit jednodušší byly vytvořeny 2 jednoduché tovární metody createParameter(), které podle vstupních parametrů vrátí instanci jedné z těchto 3 tříd. První pracuje s pouze vnitřními informacemi a používá se v případě, kdy potřebuji vytvořit hodnotu anotace nebo tagu bez nutnosti mít objekt označené hodnoty z Telosys diagramu. Přetížení této metody využívá toho, že většinou je takový objekt po ruce a používá informace z něj.

Většina programu je složena z cyklů procházejících kolekce objektů. Při poskytování přístupu k seznamům mimo vlastnící objekt může dojít k porušení zapouzdření. Například abstraktní třída DecoratedElement musí umožňovat přístup k seznamům List<Anno> a List<Tag>. Pokud by přístup k nim byl zajištěn pouze jednoduchou metodou getAnnos() a getTags() vracející odkaz na seznamy programátor by měl přístup k seznamu a mohl ho libovolně modifikovat. Tím by obešel zbytek rozhraní definovaného v DecoratedElement

a mohl by tak seznam dostat do nepovoleného stavu. Nejjednodušší by bylo vrátit kopii seznamu. Programátor by pak měl přístup ke všem prvkům a mohl je modifikovat stejně jako doposud, bez možnosti přidávat nebo odebírat prvky v opravdovém seznamu.

Vzhledem k tomu, že se operace nad těmito a podobnými seznamy se omezují pouze na iteraci přes všechny prvky lze použít návrhový vzor Iterátor. Návrhový vzor iterátor poskytuje metody k průchodu kolekcí i jiné nelineární struktury (například strom) bez nutnosti znát vnitřní implementaci této struktury. Vzhledem k tomu, že ve většině případů je nutné projít všechny prvky seznamu a každý z nich modifikovat je toto ideální řešení. Navíc v případě změny z aktuálně používaného ArrayList na jinou datovou strukturu, která nepoužívá rozhraní List bude tato změna pouze interní. V jazyce Java stačí aby použitá datová struktura dědila rozhraní Iterable a vracela objekt implementující rozhraní Iterator. [15]

7.2 Nevyřešené problémy

Některé položky specifikace ovšem nelze implementovat zcela podle plánu. Řešení pro tyto problémy není ideální, ale díky podmínkám nejlepší možné. V následujících odstavcích budou popsány problémové body řešení, proč k problému došlo a jaké je náhradní řešení, pokud existuje.

Telosys DSL obsahuje seznam neutrálních datových typů, kterých mohou nabývat atributy entit. Modely ve Visual Paradigm sice neobsahují typové kontroly, nicméně by bylo dobré je v pluginu mít k dispozici, aby uživatel měl jednoduchou kontrolu, zdali Telosys obsahuje určitý neutrální datový typ bez nutnosti otevřít dokumentaci. Ve Visual Paradigm se datové typy přidávají pomocí jazyka projektu [24] [12] . Původní plán byl vytvořit jazyk při inicializaci projektu za použití API a změnit projektový jazyk na něj. V javadoc dokumentaci ani na fóru nejsou metody, které by to umožnili. Náhradní řešení, je tedy, že si uživatel datové typy do Visual Paradigm přidá sám. Jazyk se definuje globálně a nenáleží pouze jednomu projektu, takže to uživatel musí udělat pouze jednou při instalaci pluginu. Při převodu modelů se datové typy atributů porovnávají jako řetězce. Záleží tedy pouze na tom, aby datové typy měli stejné jméno.

V průběhu návrhu se vyskytlo několik konstant, které by uživatel mohl chtít změnit. Mezi tyto konstanty patří spojovací řetězec názvů balíčků a cesta ke složce, ve které se má vygenerovat Telosys projekt. Spojovací řetězec balíčků má výchozí hodnotu „.“. Výchozí cesta k projektu je absolutní cesta ke složce, kde je uložen „.vpp“ soubor aktuálního Visual Paradigm projektu. Původní plán byl uložit je mezi projektové proměnné nebo do podobného úložiště dvojic klíč hodnota. Jedna z otázek na fóru [24] [13] ukázala, že žádné takové místo v projektu není. Náhradní řešení bylo inspirováno řešením z odpovědi. V Telosys metamodelu byla vytvořena speciální třída config. Každá z označených hodnot této třídy představuje jednu projektovou proměnnou. Třída config je spolu se zbytek metamodelu vytvořena při inicializaci projektu.

[12] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/2270/2276/59851_datatype.html

[13] Tato infromace je na stránce https://forums.visual-paradigm.com/t/how-to-stor-config-info-for-plugin/11772

7.3 Další funkce pluginu

Kromě akce převodu obsahuje plugin ještě dvě další akce. Inicializace projektu je spuštěna tlačítkem „Init project“. Po stisknutí akce plugin prohledá projektu a pokusí se najít metamodel „Telosys defs“. Pokud takový model nenajde vytvoří ho a všechny jeho komponenty. Následně projede seznam stereotypů a vytvoří všechny stereotypy používané pluginem. Pokud projekt již metamodel „Telosys defs“ obsahuje doplní pouze jeho komponenty, které v něm nenajde.

Druhá akce je spuštěna tlačítkem „Verify models“. Akce prohledá model kvůli syntaktickým a sémantickým chybám. Chybová a varovná hlášená vypisuje postupně a na závěr vypíše statistiku chyb a varování. Implementačně je při kontrole spuštěn převod, ale po jeho dokončení se nic nezapisuje do souborů.

Všechny výše popisované akce pluginu jsou bezkontextové menu akce spouštěné ze záložky plugin. Visual Paradigm ale obsahuje i kontextové akce [24] [14] , které mohou být zavolány nad určitým prvkem modelu či diagramu. Takovéto akce mají velký potenciál pro zrychlení návrhu a tvorbu Telosys diagramů. Části diagramů, které jsou často stejné nebo mají omezený počet možností popsání by tak mohli být z části či úplně automatizovány. Při návrhu a testování se ukázalo, že jedna z takovýchto oblastí je definice vztahů mezi třídami Telosys diagramu. Základem takového vztahu je asociace. Akce tedy můžeme vytvořit specificky pro asociaci a nebudou dostupné pro ostatní typy prvků což značně zjednodušuje ošetřování chybových stavů při implementaci a uživatele nebude svádět k používaní těchto akcí na ostatních prvcích modelu.

Vzhledem k tomu, že jediné povolené multiplicity asociací v Telosys diagramu jsou „0..1“ a „0..*“ je možné vytvořit kontextové akce, které by realizovali všechny možné kombinace těchto multiplicit. Kontextová akce „OneToOne“ by tedy změnila multiplicitu na „0..1“ pro oba konce asociace. Akce „ManyToMany“ provedla podobnou změnu nastavením multiplicity na „0..*“ pro oba konce asociace. Pro zbývající vztahy 1:N a N:1 je nutné najít způsob, jak se rozhodnout, který konce asociace dostane kterou multiplicitu. Způsobem určení se nakonec stal směr čtení asociace, který je již dříve použit pro rozhodnutí mezi vlastnící a inverzní stranou. Oba vztahy 1:N a N:1, pak mohou být implementovány jako jedna akce pojmenovaná „OneToMany“.

Nastavování multiplicit asociace je sice velmi repetitivní, ale nezabere příliš mnoho času. Mnohem delší je modelování linků, a hlavně cizích klíčů. V případě vztahu N:M musí uživatel obvykle vytvoří asociační třídu pro spojovací tabulku, kterou následně musí naplnit atributy obou primárních klíčů atd. Akce „Create FK(s)“ tuto zdlouhavou práci automatizuje. Výsledkem této akce je vždy asociační třída s jedním (v případě 1:1 nebo 1:N vztahu) nebo dvěma cizími klíči a jejich atributy. Asociační třída se při překladu promění na entit nebo sloučí s jednou z entit podle typu vztahu. Akce „Create Links“ vytvoří reprezentativní atributy pro oba konce asociace. Poslední implementovaná kontextová akce je akce „Create Links and FK(s)“, která provede obě předchozí akce najednou.

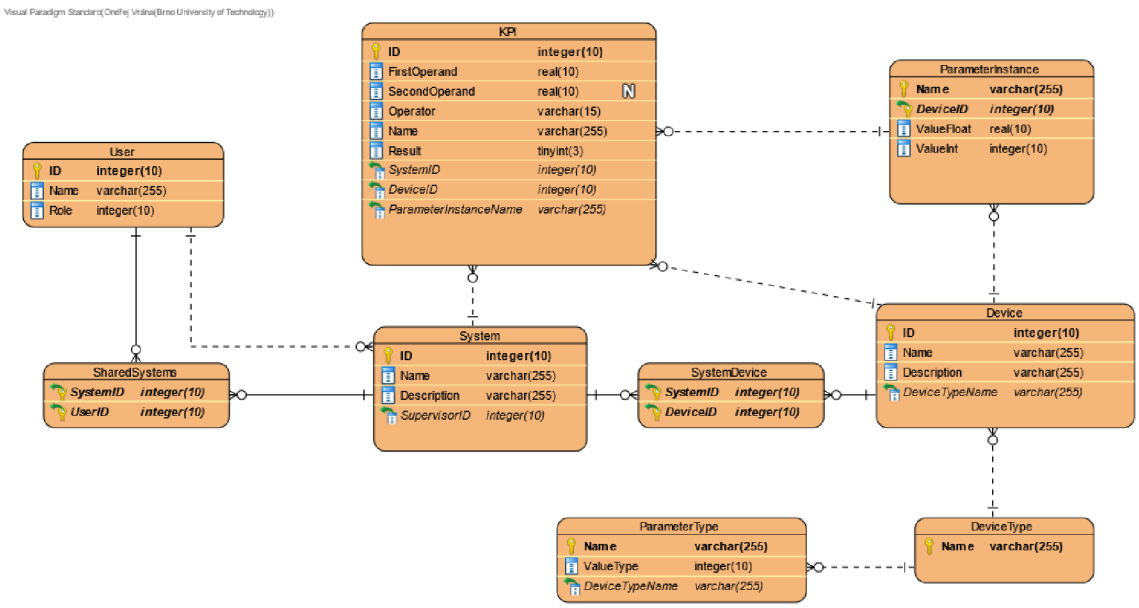
[14] Tato infromace je na stránce https://www.visual-paradigm.com/support/documents/vpuserguide/124/254/7040_implementing.html

Kapitola 8

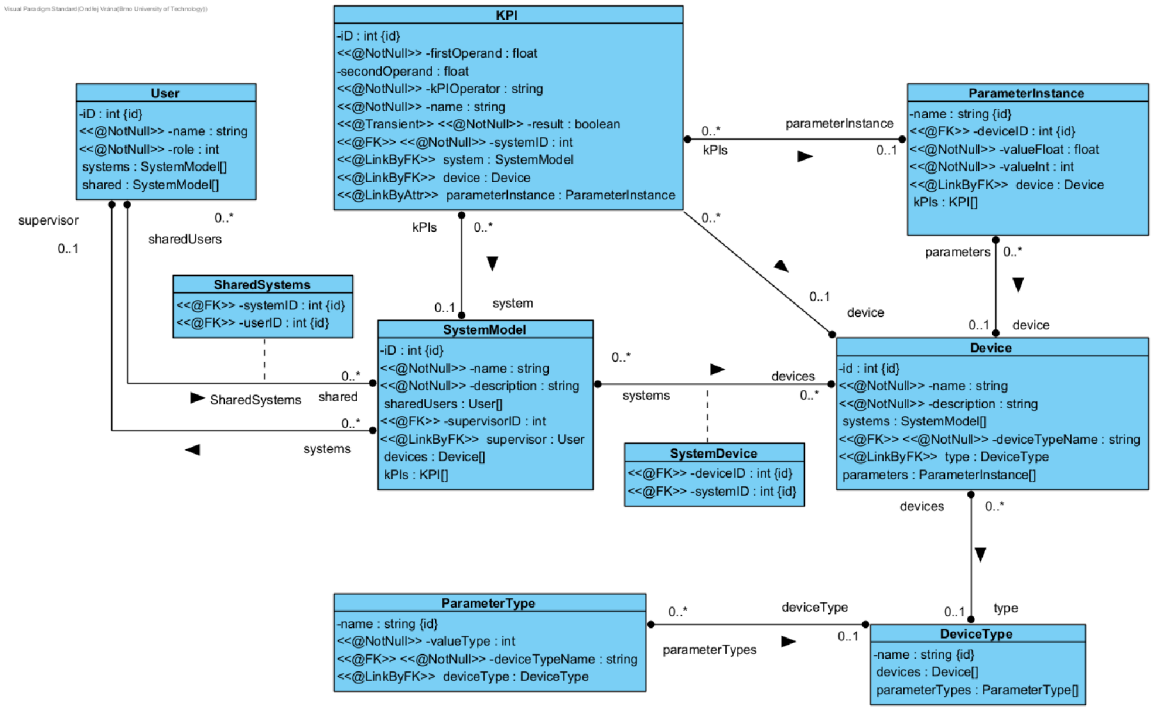
Testování

Nyní, když je plugin implementovaný je nutné ukázat jeho použití na demonstračním příkladu a zároveň otestovat jeho základní funkcionalitu. Tímto zajistíme, že systém odpovídá požadavkům stanoveným na začátku práce. Jedná se tedy o akceptační testování [2]. Demonstrační příklad je převzat z předmětu Informační systémy. Jedná se o projekt „IoT: správa zařízení“ z akademického roku 2023/2024. Nejprve bude vysvětleno zadání a vytvořen ER diagram návrhu databáze. Dále bude ukázána práce s pluginem, vytvoření Telosys diagramu, vygenerování souborů a aplikování šablony. Na závěr bude ukázka aplikace s cvičnými daty. vzheldem k závisoti modulů na Visual Paradigm projektu je testování částí projektu velmi obtížné. Z tohoto důvodu se testování na menších částech projektu nebude provádět.

Demonstrační příklad je jednoduchá webová aplikace pro správu IOT zařízení. Jednotlivá zařízení budou spojována do systému, které mohou uživatelé mezi sebou sdílet. Nad zařízením v systému jsou definovány KPI (Key performance indicator). KPI je podmínka nad jednou ze snímaných hodnot zařízení. Projekt dále rozlišuje několik rolí uživatelů, ale takové detaily by zkoušeli spíše složitost šablony, a nikoliv funkcionalitu pluginu.



Obrázek 8.1: ER diagram demontračního příkladu



Obrázek 8.2: Telosys diagram demonstračního příkladu

Diagramy na obrázcích 8.1 a 8.2 obsahují stejný počet entitních množin. Účel entitních množin User, KPI, System a Device je patrný ze zadání výše. Kromě nich obsahují diagramy ještě spojovací entitní množiny SharedSystems a SystemDevice pro M:N vztahové množiny. Každé zařízení má svůj typ, který je v diagramech reprezentován entitní množinou DeviceType. Lze tedy říci, že entity Device jsou instance nějaké entity DeviceType. Každý typ zařízení snímá jiné předdefinované hodnoty. Předdefinované snímané hodnoty každého typu DeviceType jsou definovány jako entity entitní množiny ParameterType. Opět lze říci, že instance ParameterType jsou v entitní množině ParameterInstance. Počet ParameterType přiřazených přes DeviceType k Device je roven počtu entit ParameterInstance přiřazených ke stejné entitě Device. ParameterInstance a ParameterType jsou slabé entitní množiny. Jejich silné entitní množiny, ze kterých berou diskriminátor jsou Device a DeviceType. Telosys neobsahuje nic pro označení slabé entitní množiny. Podobného efektu lze ale docílit označením komponent diskriminátoru primárním i cizím klíčem zároveň.

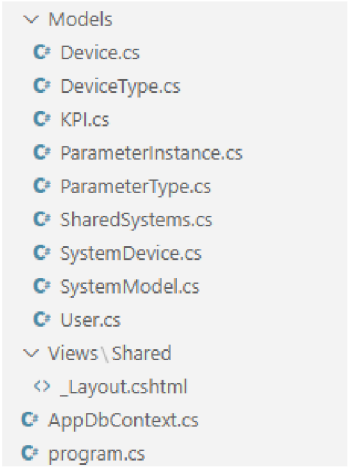
Telosys diagram je na první pohled nerozpoznatelný od diagramů tříd. Jediné, co by mohlo diagram prozradit je vysoký počet stereotypů. Telosys DSL neobsahuje metody, a tedy i Telosys diagramy tuto neumožňují, čímž se velmi podobají ERD.

Telosys diagram (obrázek 8.2) byl vytvořen přepisem z ER diagramu (obrázek 8.1). Diagram lze, ale vytvořit i bez předchozího návrhu. Uživatel spustí Visual Paradigm a vytvoří nový projekt. Jak jazyk projektu zvolí jazyk Telosys DSL nebo jinak pojmenovaný jazyk obsahující neutrální datové typy Telosys DSL. Inicializuje projekt pomocí akce „Init Project“ a následně může pracovat na modelech vytvářením modelů a diagramů tříd. Průběžně může kontrolovat svou práci za pomocí akce „Verify Models“. Jakmile je hotoví nastaví konfiguraci projektu ve třídě config v metamodelu „Telosys DSL defs“. Následně použije akci „Generate Models“ k vygenerování Telosys projektu a modelů.

Výsledkem překladu je inicializovaný projekt na místě specifikovaném v projektové proměnné „Telosys project directory“ spolu s modelem a všemi soubory „.entity“. Inicializace

projektu vytvoří složku „TelosysTools“ obsahující všechny soubory a složky Telosys projektu. Soubor s definicí entit je vytvořen pro každou třídu i pro obě asociační třídy, protože multiplicita obou vztahů, kde je použita asociační třída jsou „0..*“ na obou stranách. Pomocné třídy byly sjednoceny s jinými třídami a nemají vlastní soubory.

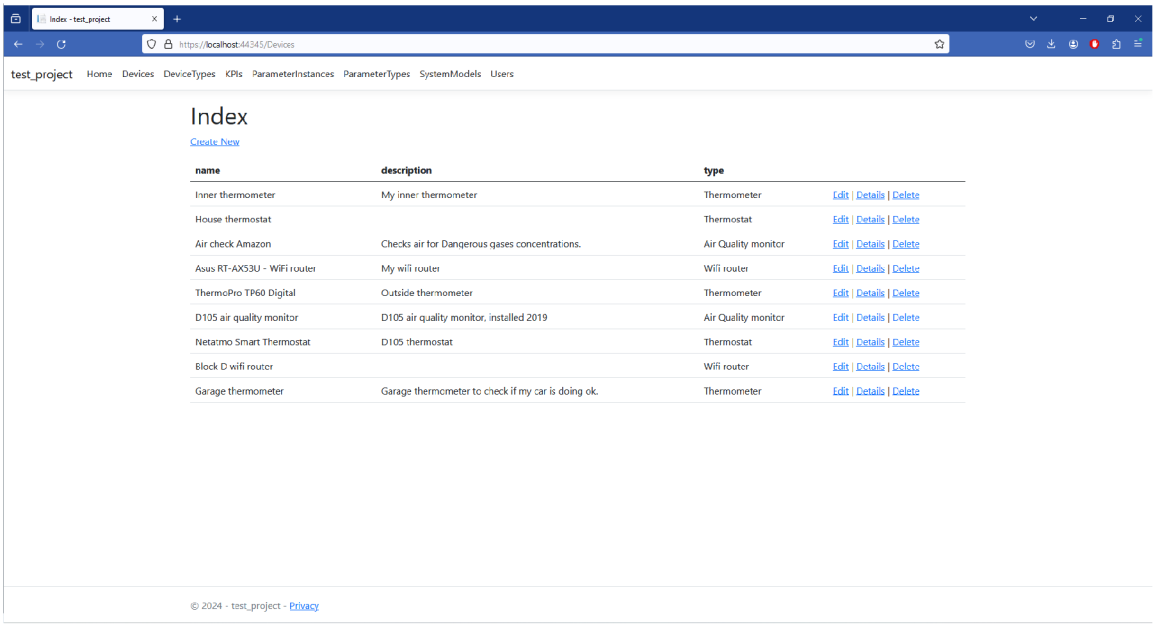
Na vygenerované entity a modely lze následně aplikovat balíčky šablon. V závislosti na balíčku pak lze vygenerovat například SQL skript pro vytvoření databáze, kostru tříd nebo CRUD operace nad nimi. Šablony lze stáhnout z [1] klonováním nebo pomocí Telosys-CLI. Generování je spouštěn zadáním příkazu „gen“ nebo „genb“ do Telosys-CLI. „gen“ aplikuje vybrané šablony z aktuálního balíčku na vybrané entity z aktuálního modelu. „genb“ aplikuje vybrané balíčky šablon na vybrané modely [11].



Obrázek 8.3: Vygenerované soubory v průzkumníku souborů Visual Studio Code

Na obrázku 8.3 jsou vidět vygenerované soubory z použité šablony. Pro testování validity vygenerovaných modelů může být použit příkaz cm Telosys-CLI [11]. Pro ověření více do hloubky je nutné na model aplikovat šablonu. Z tohoto důvodu byl vytvořen balíček šablon models_bundle [11][22][20][8], který vytovří soubory pro modely a několik doprovodných souborů pro C# .NET MVC s SQL Serverem. Vygenerování kontrolerů a oken MVC za pomocí za pomocí funkcionalit prostředí Visual Studio[3] a inicializace databáze se vytvoří spustitelná kostra aplikace (obrázek 8.4). Telosys je schopen vygenerovat i okna a kontrolery, nicméně pro ověření správnosti vygenerovaného modelu stačí šablona pouze na modely.

[1] https://github.com/telosys-templates



Obrázek 8.4: Ukázka výsledné spustitelné aplikace

Kapitola 9

Závěr

Cílem bakalářské práce bylo navrhnout a implementovat Telosys plugin pro Visual Paradigm, který bude sloužit k tvorbě a převodu Telosys diagramů na Telosys projekt. Navržený Telosys diagram byl založen na digramu tříd. Pokus o návrh založený na ER diagramu vypadal na první pohled slibněji, ale neumožňoval jednoduchou specifikaci všech prvků Telosys DSL. Rozšířením diagramů tříd o metamodel obsahující stereotypy, omezení, neutrální datové typy a výčtové typy umožňuje tvorbu stejného množství konstrukcí jako umožňuje Telosys DSL jazyk v aktuální verzi 4.1.0. Telosys diagram je podobně přehledný jako diagram tříd, a tedy mnohem přehlednější než skupina textových „.entity“ souborů, jejichž modifikací se Telosys projektu navrhoval doposud. Navrhnutí grafické reprezentace bylo úspěšné bez žádných nevyřešených problémů.

Implementace pluginu[1] se setkala s problémy hlavně na začátku a na konci. Způsob získání závislostí, které plugin potřeboval bylo složitější, než bývá obvyklé, ale problémy se podařilo vyřešit. Plugin je schopen převést všechny konstrukce navržené pro Telosys diagramy bez problémů, kontrolovat jejich správnost a na případné problémy upozornit uživatele. Kromě základní funkce převodu obsahuje i další funkce, a to například funkci pro inicializaci metamodelu a verifikaci vytvořených modelů. Některé z problémů, na které se narazilo při implementaci se ovšem nedokázalo dokonale vyřešit. Visual Paradigm nenabízí možnost programově manipulovat s jazyky a datovými typy projektů. Kvůli tomu plugin nedokáže při inicializaci metamodelu vytvořit neutrální datové typy Telosys DSL. Uživatel bude muset tento jazyk vytvořit ručně. Druhý problém je nemožnost uložit projektové proměnné mimo modely. Metamodel proto obsahuje i konfigurační soubor.

Implementovaná funkcionalita představuje základ plugin, který se dále může rozšiřovat. Nyní je možné převod mezi Telosys diagramy a Telosys projektem pouze jednostranně, a to z diagramů do modelů a entit. Jedno z možných rozšíření je převod i na druhou stranu, tedy z modelů a entit na diagramy. Převod na tuto stranu by mohl pomoci hlavně při modifikaci již existujících Telosys projektů.

[1] Plugin je dostupný z repozitáře https://github.com/myval2012/vpTelosysPlugin

Literatura

[1] Altova. UModel. *Altova* online. C2005-2024. Dostupné z: https://www.altova.com/umodel. [cit. 2024-04-25].

[2] Ammann, P. a Offutt, J. *Introduction to Software Testing*. 2. vyd. Cambridge: Cambridge University Press, 2017. ISBN 978-1-107-17201-2.

[3] Anderson, R. et al. ASP.NET Scaffolding in Visual Studio 2013. *Visual Studio* online. 4. března 2022. Dostupné z: https://learn.microsoft.com/en-us/aspnet/visual-studio/overview/2013/aspnet-scaffolding-overview. [cit. 2024-04-29].

[4] Bagui, S. S. a Earp, R. W. *Database Design Using Entity-Relationship Diagrams*. 3. vyd. Boca Raton: CRC Press, 2023. ISBN 978-1-003-31445-5.

[5] Gradle. *Gradle Build Tool* online. C2024. Dostupné z: https://gradle.org/. [cit. 2024-04-26].

[6] Guerin, L. *Telosys-models* online. Dostupné z: https://github.com/telosys-models. [cit. 2024-04-25].

[7] Guerin, L. *Telosys-tools-bricks* online. Dostupné z: https://github.com/telosys-tools-bricks. [cit. 2024-04-25].

[8] Guerin, L. *Templates.cfg* online. Prosinec 2022. Dostupné z: https://github.com/telosys-templates/java-jdbc/blob/master/templates.cfg. [cit. 2024-04-29].

[9] Jeusfeld, M. A. Metamodel. In: LIU, L. a ÖZSU, M. T., ed. *Encyclopedia of Database Systems*. Boston, MA: Springer US, 2009, s. 1727–1730. ISBN 978-0-387-39940-9. Dostupné z: https://doi.org/10.1007/978-0-387-39940-9_898.

[10] *Telosys* online. Dostupné z: https://www.telosys.org/. [cit. 2024-04-25].

[11] *Telosys documentation* online. Dostupné z: https://doc.telosys.org/. [cit. 2024-04-25].

[12] LucidSoftware. *Lucidchart* online. C2024. Dostupné z: https://www.lucidchart.com/pages/. [cit. 2024-04-25].

[13] Miscrosoft. Visio. *Miscrosoft* online. C2024. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-365/visio/flowchart-software. [cit. 2024-04-25].

[14] ObjectManagementGroup. *Unified Modeling Language* online. 2.5.1. Milford: [b.n.], prosinec 2017. Dostupné z: https://www.omg.org/spec/UML/2.5.1. [cit. 2024-04-25].

[15] Pecinovksý, R. *Návrhové vzory: 33 vzorových postupů pro objektové programování*. 1. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

[16] pp_pankaj. *Comprehensive Overview of CASE Tools: Streamlining Software Development* online. 19. ledna 2024. Dostupné z: https://www.geeksforgeeks.org/computer-aided-software-engineering-case/. [cit. 2024-04-25].

[17] Raistrick, C.; Francis, P.; Wright, J.; Carter, C. a Wilkie, I. *Model Driven Architecture with Executable UML*. 1. vyd. Cambridge: Cambridge University Press, 2004. ISBN 0-521-53771-1.

[18] Sommerville, I. *Software engineering*. 9. vyd. London: Pearson, březen 2010. ISBN 978-0-13-703515-1.

[19] SparxSystems. *UML modeling tools for Business, Software, Systems and Architecture* online. C2000-2024. Dostupné z: https://sparxsystems.com/. [cit. 2024-04-25].

[20] Svyryd, A. et al. *Create a model* online. 2023. Dostupné z: https://learn.microsoft.com/en-us/ef/core/modeling/. [cit. 2024-04-29].

[21] TheApacheSoftwareFoundation. *Apache Maven Project* online. C2002-2024. Dostupné z: https://maven.apache.org/. [cit. 2024-04-26].

[22] TheApacheSoftwareFoundation. *The Apache Velocity Project* online. C2020. Dostupné z: https://velocity.apache.org/. [cit. 2024-04-25].

[23] Tomassetti, F. Telosys: a Code Generation Tool by Laurent Guerin. *Strumenta* online. Dostupné z: https://tomassetti.me/telosys-code-generation-tool/. [cit. 2024-04-28].

[24] VisualParadigm. *Visual Paradigm* online. C2024. Dostupné z: https://www.visual-paradigm.com/. [cit. 2024-04-25].

[25] Wąsowski, A. a Berger, T. *Domain-Specific Languages: Effective Modeling, Automation, and Reuse*. 1. vyd. Cham: Springer, 2023. ISBN 978-3-031-23668-3.

[26] YattaSolutions. *UML Lab* online. C2024. Dostupné z: https://www.uml-lab.com/. [cit. 2024-04-25].

Příloha A

Výtah z příručky k pluginu

Tato příloha je část příručky týkající se instalace, překladu pluginu a základních struktur Telosys diagramu přeložená z angličtiny do češtiny. Kompletní příručka je v paměťovém médiu nebo na github stránce projektu.

Překlad projektu

1. V adresáři dependency je archiv telosys-tools-all-x.x.x.jar. Pokud chcete použít jinou verzi Telosys API nahraděte tento archiv a upravte build.gradle.kts.
2. Najděte openapi.jar knihovnu ve adresáři programu Visual Paradigm (ve složce „./Visual Paradigm xx.x/lib/“) a:
   - Zkopírujte ho do složky dependency nebo
   - Upravte build.gradle.kts tak aby o této knihovně věděl
3. Překlad je spouštěn příkazem:

   ```
   gradle build (počítač s nainstalovaným nástrojem gradle)
   gradlew.bat build (pro Windows)
   ./gradlew build (pro Linux)
   ```

4. Sestavení zip archivu pro instalaci pluginu se spustí příkazem

   ```
   gradle build zip (počítač má nainstalovanou aplikaci gradle)
   gradlew.bat build zip (pro Windows)
   ./gradlew build zip (pro Linux)
   ```

Instalace pluginu

Přeložený zip archiv lze nainstalovat přes rozhraní v programu Visual Paradigm jako jakýkoli jiný plugin. V záložce Help -> Install Plugin -> Install from a zip of plugin.

Po úspěšné instalaci je doporučen vytvořit Telosys jazyk a jeho datové typy podle https://www.visual-paradigm.com/support/documents/vpuserguide/2270/2276/

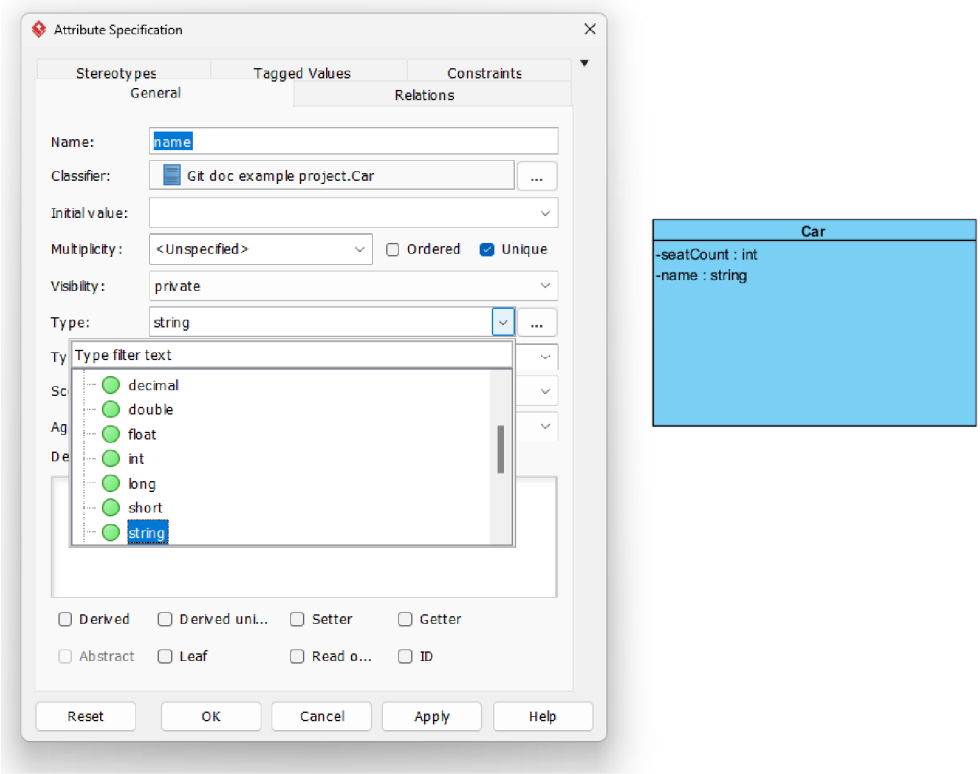
59851_datatype.html. Jazyk může být pojmenován jakkoliv, ale datové typy v něm se musí shodovat s datovými typy Telosys DSL.

Telosys diagram přehled

Tato reprezentace je založena na UML diagram tříd. Následující sekce popisují, jak je každý prvek Telosys DSL modelován.

Model a entita

Telosys model je reprezentován UML v kořenu Visual Paradigm projektu. UML model může obsahovat balíčky (anotace @Package) a třídy (reprezentují entity). Každý balíček může obsahovat další balíčky a třídy.



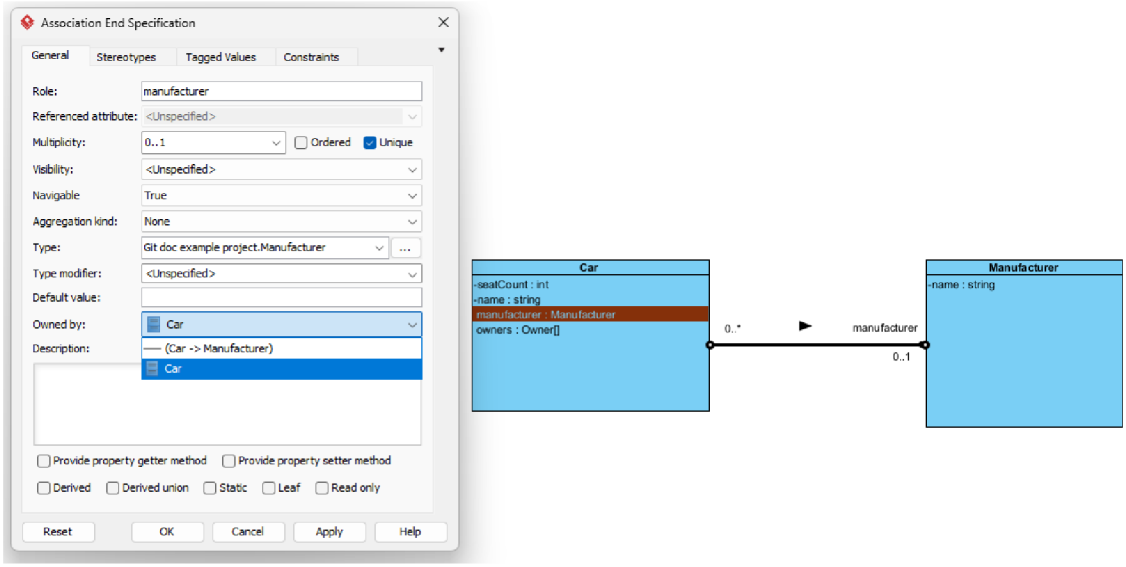
Obrázek A.1: Příklad atributu

Link

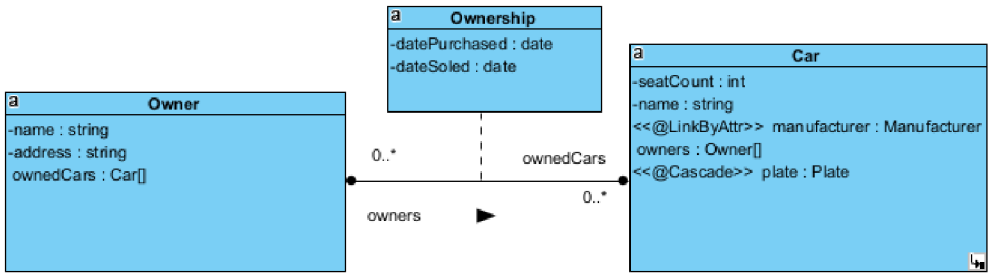
Linky jsou nejvíce komplikovanou částí Telosys diagramu. Účelem bylo učinit vztay mezi objekty přehlednější. Z tohoto důvodu je Telosys link reprezentován asociací, asociační třídou a reprezentativní atributem.

Asociace tvoří základ vztahu. Reprezentativní atribut je ve Visual Paradgim vytvořen přidělením vlastnictví konce asociace třídě namísto asociaci. Visual Paradigm automaticky vygeneruje reprezentativní atribut v protilehlé třídě.

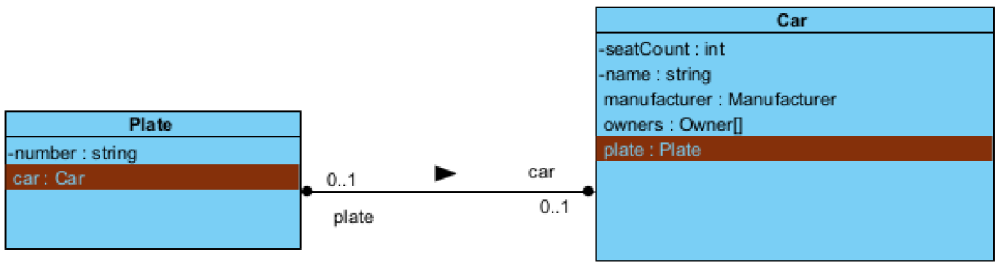
Multiplicita popisuje, zdali je link reference na jednu entitu nebo na kolekci entit. Asociační třída se používá pro specifikaci „Join entity“ (pokud je to asociace N:M).



Obrázek A.2: Jeden reprezentativní atribut. 1:n / N:1



Obrázek A.3: Oba reprezentativní atributy. N:M



Obrázek A.4: 1:1

Existuje více způsobů, jak propojit reprezentativní atribut s cizím klíčem v Telosys:

- Odvození (Inference) – Nejsou implementovány žádné kontroly, ale je to možné.
- @LinkByAttr – Pomocí stereotypu.

- @LinkByFK – Pomocí stereotypu.
- @LinkByJoinEntity – Automaticky při N:M vztahu a asociační třídě.

Navíc u 1:1 a 1:N vztahu mohou atributy cizího klíče přidány do asociační třídy. Při překladu se automaticky přidají do entity podle směr čtení asociace. Směr čtení se používá pro popis vlastnící a inverzní strany vztahu (vlastnící -> inverzní strana).

Anotace

Většina anotací je reprezentována stereotypem, některé jsou také navíc reprezentovány jako omezení a ostatní jejich UML protějšky.

Tag

Uživatel musí definovat vlastní stereotyp, který začíná na '#'. Parameter se přidá jako označená hodnota (typ označené hodnoty musí být "Text" nebo "Model Element").

Příloha B

Průzkumné testy k Telosys vztahům

Seznam provedených testů

„Co nejvíce popsaný“ popisuje vztah využívající všechny možné prvky Telosys DSL pro popis tohoto vztahu.

- Test 1: 1:1 vztah podle Telosys github příkladů. https://github.com/telosys-models/employees
- Test 2: Co nejvíce popsaný 1:1.
- Test 3: Co nejvíce popsaný 1:1 bez @MappedBy.
- Test 4: Link pouze na vlastnící straně pro 1:1.
- Test 5: Link pouze na inverzní straně pro 1:1.
- Test 6: Co nejvíce popsaný 1:N.
- Test 7: Co nejvíce popsaný 1:N bez @MappedBy.
- Test 8: Link pouze na vlastnící straně pro 1:N.
- Test 9: Link pouze na inverzní straně pro 1:N.
- Test 10: N:M vztah podle Telosys github příkladů. https://github.com/telosys-models/employees
- Test 11: Co nejvíce popsaný N:M.
- Test 12: Co nejvíce popsaný N:M bez @MappedBy.
- Test 13: N:M bez „Join Entity“
- Test 14: N:M za použití @LinkByAttr.
- Test 15: Více cizích klíčů v entitě.
- Test 16: Více cizích klíčů v entitě (cizí klíče nejsou pojmenovány).

- Test 17: Více cizích klíčů na jediném atributu.
- Test 18: N:M za použití @LinkByFK

Získané znalosti

- Testy 1,2,6,10,11 – 1:N / N:1 vztah je výchozí (pokud se nepoužije anotace @OneToOne nebo @ManyToMany)
- Testy 2,3,11,12 – pro 1:1 a N:M jsou obě strany vlastnící, dokud není použita anotace @MappedBy
- Testy 6,7 – pro 1:N / N:1 není potřeba použít anotace @MappedBy pro určení vlastnící strany, stále ale může být použita pro doplnění jména linku na opačné straně
- Testy 2,5 – pro 1:1 nelze definovat inverzní stranu bez linku na vlastnící straně
- Testy 10,11,13 – N:M lze definovat bez „Join Entity“
- Testy 14,18 – @LinkByAttr a @LinkByFK nelze použít pro N:M