Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Vykreslovač UML diagramů na základě textového popisu

Diplomová práce

2025 Eva Škopová

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Eva Škopová**
Osobní číslo:	I23284
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Vykreslovač UML diagramů na základě textového popisu**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem práce je vytvoření aplikace pro vytváření vybraných obrázků UML diagramů na základě zjednodušeného textového popisu modelu. Aplikace by měla vytvářet grafiku ve formátu SVG, podporovat postupně se rozvíjející diagramy a automaticky zvýraznit změny mezi jednotlivými stavy diagramu.

V textové části bude představen jazyk UML a vybrané diagramy (minimálně diagram tříd, sekvenční diagram a diagram aktivit). Dále bude provedena rešerše zabývající se automatizovaným rozvržením diagramu/grafu pro jeho vizualizaci. Pro potřeby popisu diagramů bude v textové části navržen a formalizován jazyk pro specifikaci textového popisu diagramu tříd (případně dalších typů diagramů) s podporou postupně rozvíjejících se diagramů v rámci několika slidů. Aplikace by měla podporovat vykreslit základní elementy diagramu – třída, rozhraní, genericita, různé typy relací, poznámky. Jazyk a rozvrhovací algoritmus by měl rovněž podporovat hinty pro částečné či přesné určení pozic jednotlivých elementů ve výsledném diagramu.

Praktická část bude realizovaná ve vybraném vyšším programovacím jazyku s podporou platforem Windows a Linux. Vstupem je textový popis modelu a výstupem je vykreslená grafika v souborech SVG.

Rozsah pracovní zprávy: **50 – 60 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

FOWLER, Martin. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2018. ISBN 9780134865126. MILES, Russ a HAMILTON, Kim. *Learning UML 2.0*. Sebastopol, CA: O'Reilly, c2006. ISBN 9780596009823.

Vedoucí diplomové práce: **Ing. Roman Diviš, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **31. října 2024**
Termín odevzdání diplomové práce: **23. května 2025**

L.S.

prof. Ing. Petr Doležel, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 29. listopadu 2024

Prohlašuji:

Práci s názvem Vykreslovač UML diagramů na základě textového popisu jsem vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.

Byla jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 21. 08. 2025

Eva Škopová v.r.

PODĚKOVÁNÍ

Ráda bych poděkovala Ing. Romanu Divišovi, Ph.D. za odborné vedení, cenné připomínky a doporučení. Poděkování patří také rodině a partnerovi za trpělivost a podporu během studia. Nakonec bych ráda vzdala dík i panu Tolkienovi, který poskytl inspiraci pro třídy v některých ukázkových diagramech.

ANOTACE

Práce se zabývá návrhem a implementací aplikace pro generování UML diagramů tříd z textového popisu. Cílem je umožnit vizualizaci postupného vývoje diagramů s automatickým zvýrazněním změn a podporou preferovaných pozic prvků. V teoretické části je představen jazyk UML, vybrané diagramy a automatické rozvržení grafu. Základem pro realizaci řešení jsou požadavky, srovnání existujících nástrojů a návrh vlastního jazyk pro popis diagramů tříd. Aplikace je postavena na využití PlantUML pro generování diagramů, ale zachovává vlastní nezávislou datovou reprezentaci. Pro podporu nové rozšiřující funkcionality využívá existující mechanismy PlantUML. Součástí práce jsou i návrhy budoucích rozšíření zvyšujících flexibilitu nástroje.

KLÍČOVÁ SLOVA

UML, diagram tříd, automatické rozvržení grafů, PlantUML

TITLE

UML diagram renderer based on text description

ANNOTATION

This thesis focuses on the design and implementation of an application for text-based rendering of UML class diagrams. The aim is to enable the visualization of the gradual diagram development with automatic change highlighting and support preferred element positions. The theoretical part introduces UML, selected types of diagrams, and the principles of automatic graph layout. The solution is based on defined requirements, a comparison of existing tools, and the design of a dedicated language for describing class diagrams. The application employs PlantUML for diagram rendering while maintaining its own independent internal data representation. Existing PlantUML mechanisms are utilized to support extended functionality. The thesis also presents proposals for future extensions increasing the flexibility of the tool.

KEYWORDS

UML, class diagram, automatic graph layout, PlantUML

OBSAH

SEZNAM OBRÁZKŮ 9

SEZNAM TABULEK 11

SEZNAM KÓDŮ 12

SEZNAM ZKRATEK A ZNAČEK 13

ÚVOD 14

1 Představení UML 15

1.1 Praktické využití 15

1.2 Vznik 15

1.3 Struktura 16

1.3.1 Stavební bloky UML 17

1.3.2 Pravidla UML 20

1.3.3 Společné mechanismy 20

1.3.4 Architektura 23

2 Vybrané diagramy 25

2.1 Diagram tříd 25

2.2 Sekvenční diagram 28

2.3 Diagram aktivit 31

3 Automatické rozvžení grafů 33

3.1 Kritéria a metriky pro rozvržení grafu 33

3.2 Metody pro automatické rozvržení 37

4 Požadavky pro praktickou část 45

4.1 Požadavky cílové aplikace 45

4.2 Porovnání existujích nástrojů 45

4.2.1 nomnoml 46

4.2.2 Mermaid 46

4.2.3 PlantUML 47

4.2.4 Srovnání existujících jazyků 49

5 Návrh jazyka 53

5.1 Hlavní modifikace PlantUML 53

5.2 Formální definice jazyka 55

5.3 Ukázka vstupu a výstupu 57

6 Návrh a implementace 60

6.1 Návrh aplikace 60

6.2 Implementace aplikace 64

6.3 Spuštění aplikace 71

6.4 Možnosti rozšíření aplikace 72

ZÁVĚR 75

POUŽITÁ LITERATURA 76

SEZNAM OBRÁZKŮ

Obrázek 1: Ukázka obecné závislosti a závislosti se stereotypem «use», vytvořeno pomocí VisualParadigm, zdroj: vlastní....................18
Obrázek 2: Ukázka asociací, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................19
Obrázek 3: Ukázka generalizace, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................19
Obrázek 4: Ukázka realizace, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................19
Obrázek 5: Přehled UML diagramů, vytvořeno pomocí VisualParadigm, zdroj: (Arlow, 2007) 20
Obrázek 6: Ukázka ornamentů, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................21
Obrázek 7: Ukázka třídy a jejích instancí, vytvořeno pomocí VisualParadigm, zdroj: vlastní 22
Obrázek 8: Ukázka stereotyp a označená hodnota, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................22
Obrázek 9: Ukázka omezení, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................23
Obrázek 10: Ukázka Modelu architektury 4+1 pohledů, vytvořeno pomocí VisualParadigm, zdroj: (Kruchten, 1995)(Arlow, 2007)....................24
Obrázek 11: Ukázka vlastností ve formě atributů, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................25
Obrázek 12: Ukázka vlastností ve formě asociací, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................26
Obrázek 13: Ukázka generické třídy, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................28
Obrázek 14: Ukázka asociační třídy, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................28
Obrázek 15: Ukázka Vytvoření objektu, vytvořeno pomocí VisualParadigm, zdroj: vlastní ..29
Obrázek 16: Ukázka zrušení objektu, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................29
Obrázek 17: Ukázka kombinovaných fragmentů, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................30
Obrázek 18: Ukázka diagramu aktivit, vytvořeno pomocí VisualParadigm, zdroj: vlastní.....32
Obrázek 19: Stejný graf vykreslení s křížením a bez křížení hran, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................34
Obrázek 20: Stejný graf s ohýbáním a bez ohýbání hran, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................35
Obrázek 21: Stejný graf s různou úrovní symetrie, vytvořeno pomocí VisualParadigm, zdroj: vlastní....................35

Obrázek 22: Stejný graf s různými vzdálenostmi vrcholů, vytvořeno pomocí VisualParadigm, zdroj: vlastní 36
Obrázek 23: Stejný graf bez zachování směru a se zachováním směru, vytvořeno pomocí VisualParadigm, zdroj: vlastní 36
Obrázek 24: Ukázka aplikace hierarchickéo rozvržení, převzato z (Automatic Diagram Layout, c2024) 39
Obrázek 25: Ukázka aplikace organického rozvržení, převzato z (Automatic Diagram Layout, c2024) 40
Obrázek 26: Ukázka aplikace ortogonálního rozvržení, převzato z (Automatic Diagram Layout, c2024) 42
Obrázek 27: Ukázka aplikace kruhového rozvržení, převzato z (Automatic Diagram Layout, c2024) 43
Obrázek 28: Ukázka aplikace stromového rozvržení, převzato z (Automatic Diagram Layout, c2024) 44
Obrázek 29: Diagram tříd vytvořený pomocí nomnoml, zdroj: vlastní 46
Obrázek 30: Diagram tříd vytvořený pomocí Mermaid, zdroj: vlastní 47
Obrázek 31: Diagram tříd vytvořený pomocí PlantUML, zdroj: vlastní 48
Obrázek 32: Očekávaný výstup pro zakládání a rušení prvků, zdroj: vlastní 58
Obrázek 33: Porovnání výstupu bez hintů a očekávaného výstupu s hinty, zdroj: vlastní 59
Obrázek 34: Ukázka konfliktních preferencí pro centrální prvek, zdroj: vlastní 64
Obrázek 35: Ukázka využití doplňkových funkcionalit, zdroj: vlastní 69
Obrázek 36: Ukázka použití aplikace, zdroj: vlastní 71

SEZNAM TABULEK

Tabulka 1: Vysvětlení jednotlivých částí definice členů třídy, údaje z (Fowler, 2009) 26
Tabulka 2: Prvky založené na třídě, údaje z (Unified Modeling Language, c2025)(Arlow, 2007) 27
Tabulka 3: Základy syntaktické konstrukce UML prvků ve vybraných jazycích, zdroj: dokumentace porovnávaných nástrojů...49
Tabulka 4: Pravidla pro formulaci vztahu podle preferovaných pozic, zdroj: vlastní.............63
Tabulka 5: Souhrn typu vykreslení pro konkrétní stav parametrů aktuálního snímku, zdroj: vlastní 67

SEZNAM KÓDŮ

Kód 1: Popis diagramu pro nomnoml 50
Kód 2: Popis diagramu pro Mermaid 51
Kód 3: Popis diagramu pro PlantUML 52
Kód 4: Definice gramatiky 55
Kód 5: Ukázka vstupu pro demonstraci zakládání a rušení prvků 58
Kód 6: Ukázka vstupu pro demonstraci preferencí pro rozvržení prvků 59
Kód 7: Popis diagramu s využitím doplňkových funkcionalit 68
Kód 8: Ukázka použití aplikace 70

SEZNAM ZKRATEK A ZNAČEK

CASE	Computer Aided Software Engineering
DAG	Directed Acyclic Graph
ELK	Eclipse Layout Kernel
JDK	Java Development Kit
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MSAGL	MicroSoft Automatic Graph Layout
OCL	Object Constraint Language
OMG	Object Management Group
OMT	Object-Modeling Technique
OOP	Objektově Orientované Programování
TSM	Topology-Shape-Metrics
UML	Unified Modeling Language

ÚVOD

Modelovací jazyk UML představuje široce používaný standard pro popis softwarových systémů. S postupným rozvojem diagramů, ke kterému dochází během návrhu a vývoje aplikací, se vyskytuje potřeba sledování postupných změn. Vizualizace jednoduchých změn, například na menších diagramech, které jsou součástí rozsáhlejšího modelu, umožňuje lépe vnímat dynamiku vývoje a zvýšit pochopení provedených změn systému. Vizualizace změn má využití i při studiu UML, kdy je vhodná pro praktické ukázky postupného vytváření diagramů.

Hlavním cílem práce je návrh a implementace aplikace pro generování UML diagramů tříd na základě zjednodušeného textového popisu modelu. Aplikace bude zaměřena na podporu postupného rozvoje diagramu a nabídne automatické zvýraznění změn mezi jednotlivými verzemi. Důležitou součástí bude také možnost využití nápovědy pro určení pozic jednotlivých prvků ve výsledném diagramu. Realizace aplikace vyžaduje formální definici jazyka, který určuje vstupní formát.

Na začátku teoretické části je detailně představen jazyk UML, jeho struktura a vybrané typy diagramů. Následující kapitola se soustředí na problematiku automatického rozvržení grafů, představení základních kritérií pro řešenou oblast, přehled dostupných metod a jejich implementací. Další část práce se zabývá specifikací požadavků na cílovou aplikaci a srovnáním existujících řešení, která mohou sloužit jako východisko pro návrh implementované aplikace či navrhovaného jazyka. Následně je popsán návrh vlastního popisného jazyka včetně formální definice jeho gramatiky. Závěrečná kapitola se věnuje návrhu a implementaci aplikace a obsahuje rovněž doporučení pro možná budoucí rozšíření programu.

1 Představení UML

UML (Unified Modeling Language) je jazyk pro vizuální modelování systémů, které bývají budované s objektově orientovaným přístupem. Je průmyslovým standardem s univerzálním využitím, které je přiblíženo v následující podkapitole. Dále je rozebrána stručná historie UML se zaměřením na okolnosti a motivaci jeho vzniku. Nakonec je věnována pozornost meta-modelu, stavbě jazyka a konceptuálnímu modelu UML.

1.1 Praktické využití

Jazyk UML má širokou škálu využití v životním cyklu softwarových systémů. Již v raných fázích vývoje umožňuje zachytit požadované uživatelské funkce systému a vytvořit konceptuální model. Pro návrh poskytuje prostředky pro modelování přesnější struktury a chování systému. Používá se také k tvorbě dokumentace, a v některých případech i při implementaci – zejména v kontextu MDD a MDA, kde se z modelů generuje zdrojový kód.

UML hraje důležitou roli ve vzdělávání v oblasti informačních technologií. Buď jako součást oboru softwarového inženýrství, které se soustředí na návrh a projektování systémů, tedy i jejich modelování. Nebo pro demonstraci principů OOP a návrhových zdrojů.

Modely zachycují jednotlivé požadavky, logiku a architekturu bez potřeby znát implementační detaily. UML umožňuje jednotně, srozumitelně a přehledně zachytit strukturu a chování systému, čímž usnadňuje komunikaci mezi zadavatelem, analytikem a programátorem. Sám o sobě nepředstavuje vývojovou metodiku, avšak je navržen tak, aby bylo možné jej kombinovat s různými metodikami (Arlow, 2007).

1.2 Vznik

Rozvoj OOP vedl v 80. a 90. letech dvacátého století ke vzniku řady metodik vývoje softwaru a objektově orientovaných grafických jazyků. Mezi nejznámější se řadí metodika Booch od stejnojmenného tvůrce, OMT od Jamese Rumbaugha a Objectory od Ivara Jacobsona (Kanisová, 2006). Existence velkého množství způsobů modelování byla příčinou zmatku a problémů v komunikaci různých týmů (Fowler, 2009).

Po neúspěšném pokusu o sjednocení v podobě metodiky Fusion dala společnost Rational dohromady autory nejpoužívanějších metodik, kteří společně dali vzniknout jednotné notaci UML (Arlow, 2007). V roce 1997 vyšla verze 1.0 UML a následující roky vycházely revize doplňující původní dokument. V roce 2005 byl jazyk předělán do verze 2.0, která přináší převážně změny metamodelu pro zpřesnění specifikace UML a zajištění její konzistence

(Arlow, 2007). Nejnovější verze 2.5.1 z roku 2017 sjednocuje specifikaci do jednoho dokumentu a zjednodušuje strukturu bez zásadních změn ve funkcionalitě (Unified Modeling Language, c2025).

Cílem UML byla standardizace a podpora vzájemné slučitelnosti CASE nástrojů, kterou prosazovalo konsorcium OMG (Fowler, 2009). OMG se také zapojilo do práce na UML a v dnešní době je UML pod jejich záštitou.

1.3 Struktura

Pro správné použití jazyka a jeho podrobnou analýzu, která je součástí této práce, je stěžejní porozumět jeho základním stavebním principům. Avšak strukturu jazyka UML lze vnímat z několik různých perspektiv, které odrážejí odlišné cíle a úrovně detailu.

Oficiální specifikace *Unified Modeling Language* pro verzi 2.5.1 definuje modelovací jazyk formálně prostřednictvím metamodelu (Unified Modeling Language, c2025). Ten se skládá z popisu metatříd, jejich vztahů a vlastností, které určují, jaké konstrukce UML umožňuje používat, jaký je jejich význam a jak je možné je kombinovat.

Těchto metadříd je velké množství a mají své vlastnosti, vazby a hierarchii. Na nejvyšší úrovni abstrakce OMG specifikace rozlišuje tří hlavní katogorie modelovaných elementů, které jsou základem všech UML modelů:

- Klasifikátory (Classifiers) – definují množinu objektů, které mají svůj individuální stav a vztah k ostatním objektům. Klasifikátor má rysy, které specifikují jejich strukturální či behaviorální charakteristiku. Na nejnižší úrovni se jedná například o třídy, balíčky či asociace.

- Události (Events) – označují výskyty dějů, které se mají následky pro modelovaný systém. Konkrétní instance událostí, jako jsou CallEvent, SignalEvent či TimeEvent, se vizuálně neobjevují jako samostatné prvky diagramu, ale jsou implicitní součástí jiných grafických konstrukcí – například jako spouštěče přechodů ve stavových diagramech nebo jako zprávy v sekvenčních diagramech.

- Chování (Behaviors) – popisuje potenciální průběh vykonávání (executions), tedy dynamiku systému v čase (Unified Modeling Language, c2025). Jedná se o abstraktní koncept, který je modelován prostřednictvím konkrétního diagramu.

Publikace *The Unified Modeling Language User Guide* (Booch, 2005) představuje jazyk UML z praktického pohledu, klade důraz na vysvětlení terminologie modelování a nabízí čtenáři srozumitelné seznámení s konceptuálním modelem. Pro jeho pochopení je stěžejní seznámit se s třemi hlavními elementy:

- základní stavební bloky,
- pravidla pro sestavení bloků dohromady,
- společné mechanismy, které se vyskytují napříč UML (Booch, 2005).

Dalším důležitým aspektem UML je to, z jakých perspektiv je vhodné modelovat cílový systém. K tomu slouží architektura 4+1 pohledů, která je představena v oddíle 1.3.4.

1.3.1 Stavební bloky UML

Základní stavební bloky UML se kategoricky dělí na:

- předměty – jednotlivé abstrahované prvky modelovaného systému,
- vztahy – zavádí relaci mezi předměty, která má podle typu vztahu speciální sémantiku,
- diagramy – pohledy na kolekce předmětů, které společně definují, co a jak systém dělá (Arlow, 2007).

Předměty

Předměty (*things*) představují jednotlivé abstrahované prvky modelovaného systému (Arlow, 2007). Dělí se na:

- strukturální,
- behaviorální (chování),
- skupinové (seskupení),
- poznámky (Booch, 2005).

Strukturální předměty představují podstatná jména UML modelu, představují statické části systému. Je možné je označovat jako klasifikátory (classifiers), ale s jiným významem, než mají klasifikátory z formální specifikace jazyka, přestože nesou některé společné znaky. Konkrétně se jedná o třídy, rozhraní, spolupráce, případy užití, aktivní třídy, komponenty, artefakty a uzly.

Behaviorální předměty jsou slovesa UML modelu a zachycují chování – dynamickou část systému v prostoru a čase. Dále se rozdělují do tří kategorií:

- Interakce – výměna zpráv mezi objekty pro docílení určitého výsledku v rámci daného kontextu. Zahrnuje zprávy, akce a konektory (propojení objektů).

- Stavový automat – sekvence stavů objektu, nebo interakce v čase. Obsahuje stavy, přechody, události a aktivity.
- Aktivita (*activity*) – sekvence akcí (*action*), což jsou drobnější výpočetní kroky (Booch, 2005).

Skupinové předměty slouží k organizaci v rámci UML modelu. Jsou to schránky pro dekompozici modelu do logických celků. Hlavním zástupcem jsou balíčky, které se mohou vyskytovat i ve variantách frameworku, modelu či subsystému (Booch, 2005).

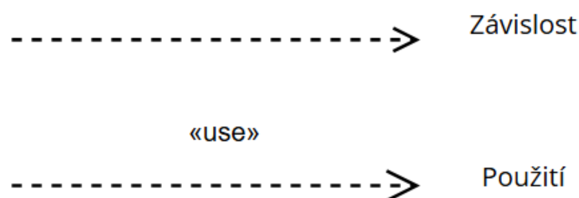
Poznámky jsou popisné a vysvětlující části UML pro okomentování jakéhokoliv prvku v modelu. Umožňují specifikovat dodatečné informace, které nejsou běžnou součástí formální sémantiky UML.

Vztahy

V UML jsou čtyři základní druhy vztahů:

- závislost,
- asociace,
- generalizace,
- realizace (Booch, 2005).

Závislost je sémantický vztah, kdy má změna jednoho prvku vliv na druhý prvek. Typ závislosti může být upřesněn pomocí stereotypů, což je vyobrazeno na Obrázek 1. Stereotypy jsou rozebrány podrobněji v oddíle 1.3.3.

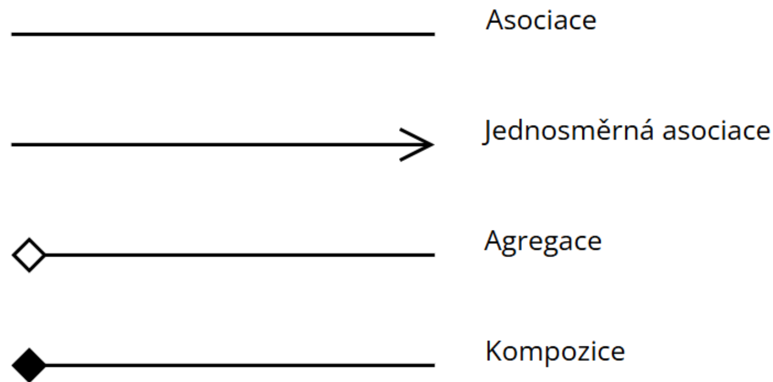


Obrázek 1: Ukázka obecné závislosti a závislosti se stereotypem «use», vytvořeno pomocí VisualParadigm, zdroj: vlastní

Obecné propojení strukturálních předmětů se nazývá asociace. Říká, že mezi instancemi těchto předmětů může existovat spojení. Asociace může obsahot různé ornamenty, které upřesňují její význam. Mezi základní ornamenty asociace patří:

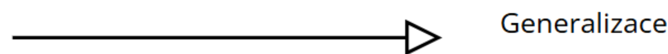
- Název – představuje popis vztahu, aby bylo jasné, co znamená. Může obsahovat i určení směru, ve kterém se má popisek číst.
- Role – označuje specifickou úlohu účastníků asociace.
- Násobnost – určuje kolik instancí předmětů ve vztahu může být spojeno. Lze zapsat jako konkrétní počet instancí, nebo rozsah hodnot.

- Agregace – umožňuje zahrnout do modelu informaci o vztahu celek-část, ve kterém předměty nejsou na stejné konceptuální úrovni, ale ve vztahu celku, ke kterému patří část.
- Kompozice – silnější forma agregace pro vyjádření toho, že daná část nemůže existovat samostatně bez celku (Booch, 2005).



Obrázek 2: Ukázka asociací, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Generalizací se zavádí hierarchie předmětů. Slouží k vyjádření dědičnosti, což je klíčový koncept objektově orientovaného přístupu, ze kterého UML vychází. Ve vztahu dědičnosti zúčasněné prvky vystupují v rolích rodiče a potomka, přičemž potomek přebírá strukturu a chování rodiče.



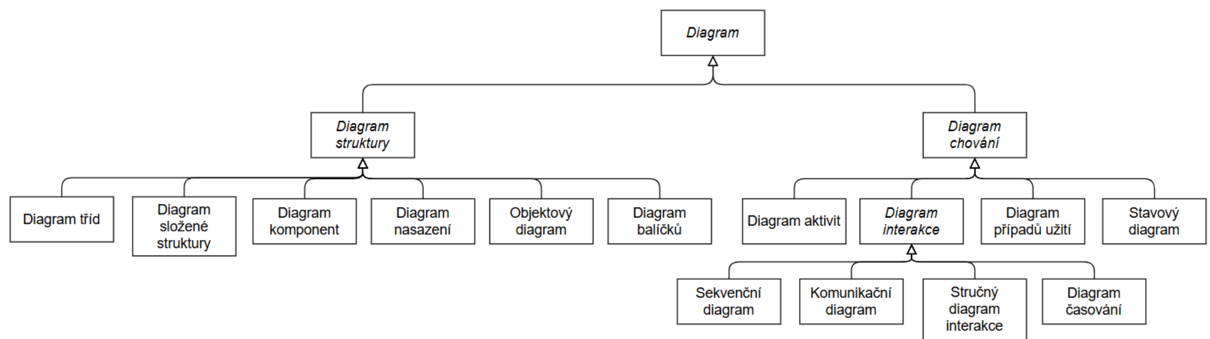
Obrázek 3: Ukázka generalizace, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Realizace je sémantický vztah mezi klasifikátory, kdy jeden klasifikátor specifikuje dohodu, která je naplněna druhým klasifikátorem (Arlow, 2007).

Obrázek 4: Ukázka realizace, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Diagramy

Diagramy je pohledem na model, model obsahuje prvky a diagram vizuálně znázorňuje některé z těchto prvků a jejich vztahů. Diagramy se modelují pro přehledné představení důležitých částí systému, které se týkají statické struktury nebo dynamického chování. V UML existuje 13 druhů diagramů, jejichž uspořádání znázorňuje Obrázek 5.



Obrázek 5: Přehled UML diagramů, vytvořeno pomocí VisualParadigm, zdroj: (Arlow, 2007)

Nejpoužívanějším diagramem je diagram tříd, na který se tato práce zaměřuje a který je podrobně představen v druhé kapitole. V rámci druhé kapitoly jsou probrány i sekvenční diagramy a diagramy aktivit. Pokryjeme tedy diagramy z obou hlavních kategorií – struktura a chování.

1.3.2 Pravidla UML

Pravidla určují, jak lze stavební bloky správně kombinovat a používat. Vizuální symboly totiž nestačí k modelování smysluplných a srozumitelných diagramů, a proto jsou v metamodelu popsaná pomocí OCL pravidla.

UML obsahuje syntaktická a sémantická pravidla pro:

- názvy – omezení pro pojmenovávání předmětů, vztahů a diagramů,
- rozsah – kontext, ve kterém daný název platí,
- viditelnost – určuje, kdo a odkud může přistupovat k určitému prvku,
- integritu – předepisuje, jak mají být předměty řádně a konzistentně vzájemně propojeny, aby dával model smysl,
- spouštění – význam provedení a simulace dynamického modelu (Booch, 2005).

1.3.3 Společné mechanismy

Rozšíření základních stavebních bloků UML se provádí pomocí společných mechanismů, které konzistentním způsobem umožňují lépe vyjádřit význam, kontext nebo chování modelovaných prvků. Hlavní typy těchto obecných mechanismů jsou:

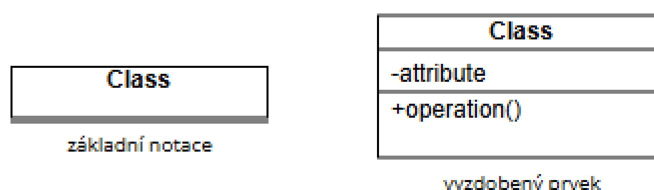
- specifikace,
- ornamenty,
- podskupiny,
- mechanismy rozšiřitelnosti (Arlow, 2007).

Specifikace

Každý grafický prvek UML má i svou textovou část, která se nazývá specifikace. Jde o textový popis poskytující sémantickou podporu grafické notace.

Ornamenty

Jsou grafické nebo textové dolňky modelovaných prvků, které rozšiřují základní notaci prvku o zobrazení detailnějších informací. Příkladem je vizuální přidání atributů a operací k základní notaci pro třídu, což je pouhý obdelník s názvem dané třídy (Arlow, 2007). Porovnání základní a ornamenty vyzdobené notace je zobrazeno na Obrázek 6.



Obrázek 6: Ukázka ornamentů, vytvořeno pomocí VisualParadigm, zdroj: vlastní

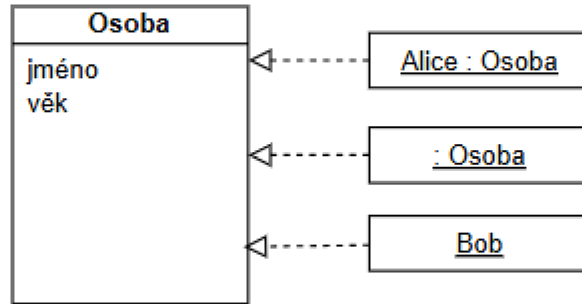
Podskupiny

Každá podskupina modelu odpovídá jinému způsobu, jak se dívat na svět konstruovaného systému. V objektově-orientovaném modelování se běžně uplatňuje princip vícepohledového přístupu, kdy se systém popisuje prostřednictvím několika navzájem se doplňujících aspektů.

Rozlišujeme tři skupiny:

- klasifikátory a jejich instance,
- rozhraní a jejich implementace,
- klasifikátory a jejich role (Booch, 2005).

Příkladem z první skupiny je rozdělení tříd a objektů, kdy třída je abstrakcí a objekt konkrétní instancí dané abstrakce. Graficky jsou instance v UML znázorněny stejnou základní notací jako jejich abstraktní klasifikátor s podtrženým názvem objektu navíc.



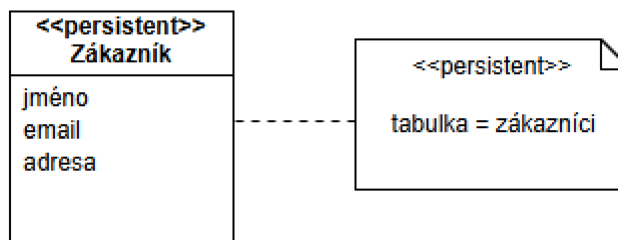
Obrázek 7: Ukázka třídy a jejích instancí, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Mechanismy rozšiřitelnosti

UML jako standardní modelovací jazyk usiluje o o širokou použitelnost, avšak jako obecný nástroj není schopen plně vystihnout veškeré doménově specifické požadavky, konvence a jiné nuance. Z tohoto důvodu je navržen jako otevřený jazyk, který je možné rozšířit prostřednictvím standardizovaných mechanismů tak, aby nebyla narušena základní syntaxe a metamodel jazyka. Tyto rozšiřující prostředky jsou umístěny v tzv. profilech a patří do nich stereotypy, označené hodnoty a omezení (Arlow, 2007).

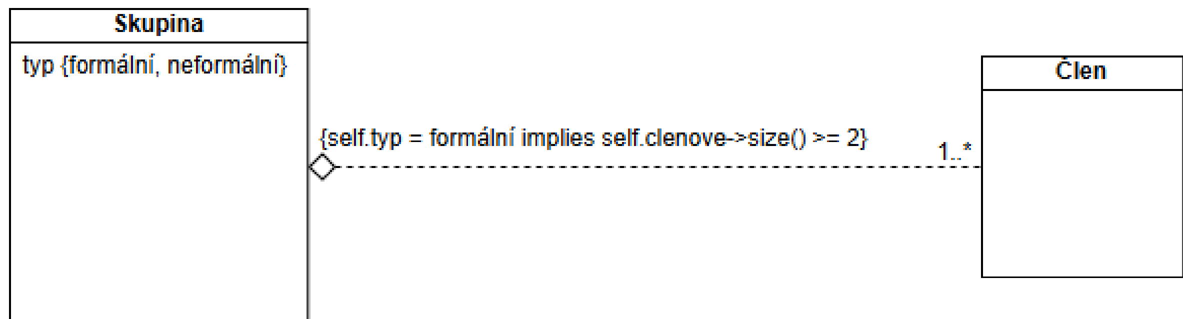
Stereotyp představuje specializaci existujícího modelového prvku. Zachovává jeho strukturu, ale mění nebo upřesňuje jeho význam. Lze tedy vytvořit nové druhy prvků, které vychází ze základních bloků. V nejběžnější grafické formě jsou vyznačeny ve francouzských uvozovkách a někdy je k nim přidána i vlastní ikona (Booch, 2005). Pro zjednodušení zápisu některé nástroje využívají místo francouzských uvozovek dvojité úhlové závorky, které jsou použity i v Obrázek 8.

Označené hodnoty umožňují přidávat uživatelsky definované atributy k prvkům modelu. Nejsou součástí instance systému, ale slouží k indikaci informací pro správu projektu, generování kódu nebo doménově-specifických informací (Rumbaugh, c2005). Na Obrázek 8 je ilustrativní ukázka stereotypu s označenou hodnotou.



Obrázek 8: Ukázka stereotyp a označená hodnota, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Omezení určují dodatečné podmínky, které musí být v modelu splněny. V UML bývají vyjádřeny pomocí OCL, což je formální jazyk pro specifikaci pravidel (Rumbaugh, c2005). Notace pro omezení je text uzavřený ve složených závorkách.



Obrázek 9: Ukázka omezení, vytvořeno pomocí VisualParadigm, zdroj: vlastní

1.3.4 Architektura

Model architektury 4+1 pohledů, někdy označován jako Kruchtenův podle autora Philippea Kruchtena, poskytuje praktický rámec pro smysluplné použití UML diagramů s cílem vyvinout nový informační systém. Hlavním cílem této architektury je poskytnout srozumitelný návrh systému každé zainteresované straně, z nichž má odlišné potřeby a očekávání. Mezi tyto strany patří koncoví uživatelé, zákazníci, analytici, vývojáři, projektoví manažeři nebo správci systémů.

Model se skládá ze čtyř hlavních pohledů, které pokrývají funkční, vývojové a provozní charakteristiky systému:

- logický pohled,
- pohled procesů,
- implementační pohled,
- pohled nasazení (Kruchten, 1995).

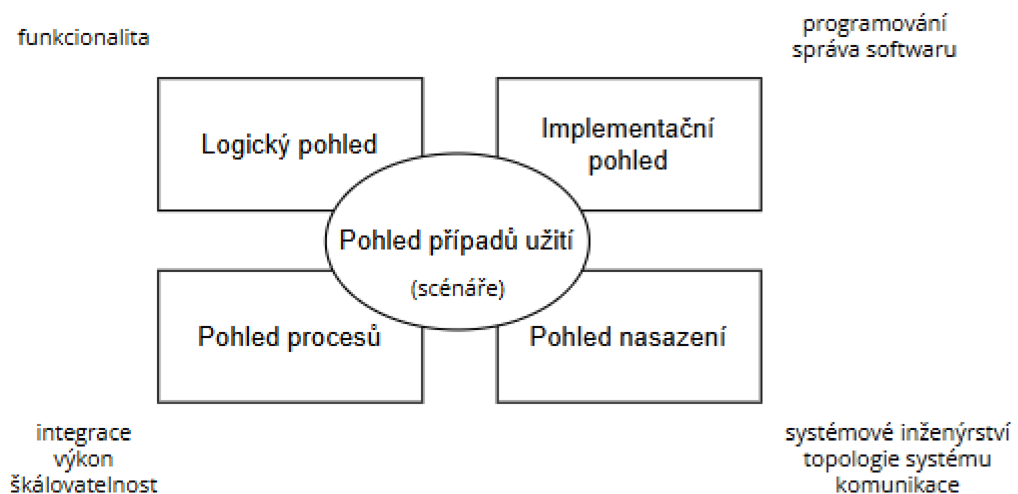
Logický pohled zachycuje funkční požadavky systému z hlediska struktury tříd, objektů a jejich vztahů. Typicky se vyjadřuje pomocí diagramů tříd či objektů.

Procesní pohled modeluje dynamické aspekty systému, zejména konkurenci, paralelismus a synchronizaci. K vyjádření se používají stavové, sekvenční a komunikační diagramy.

Implementační pohled popisuje statickou organizaci softwarových komponent ve vývojovém prostředí, tedy strukturu modulů, balíků a knihoven. Zobrazuje se např. pomocí diagramů komponent a balíků.

Pohled nasazení se zaměřuje na mapování softwaru na hardware, včetně síťové infrastruktury a rozmístění procesů. Pro jeho popis slouží především diagramy nasazení.

Pátý pohled se soustředí na vybrané klíčové scénáře, které dávají dohromady všechny ostatní pohledy. Scénáře jsou abstrakcí nejdůležitějších požadavků a modelují se pomocí případů užití.



Obrázek 10: Ukázka Modelu architektury 4+1 pohledů, vytvořeno pomocí VisualParadigm, zdroj: (Kruchten, 1995)(Arlow, 2007)

2 Vybrané diagramy

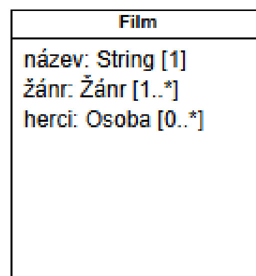
V této podkapitole jsou rozebrány vybrané diagramy, z čeho se skládají a jakým způsobem bývají použity.

2.1 Diagram tříd

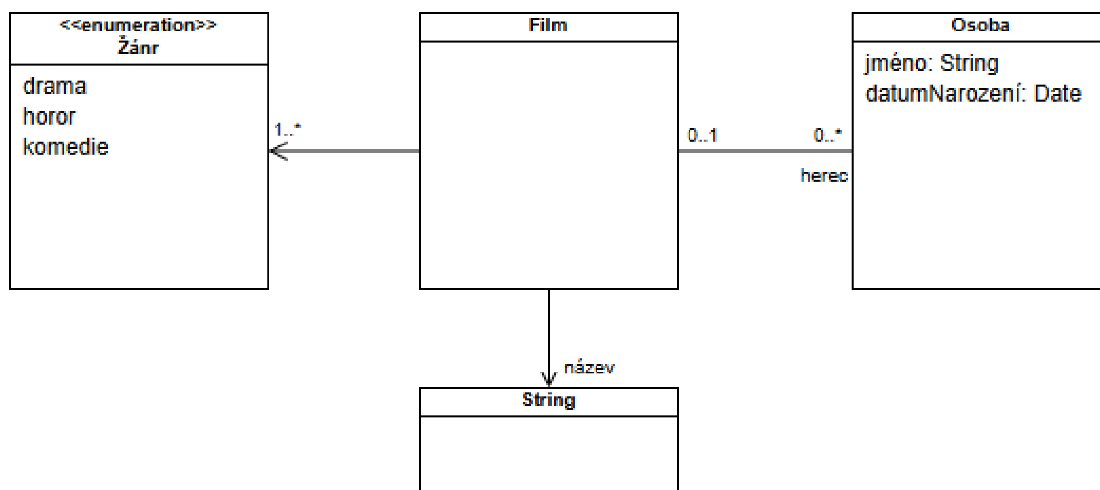
Diagram tříd je jedním z nejpoužívanějších a nejvýznamnějších typů UML diagramů, zejména v kontextu objektově orientovaného návrhu. Tento diagram je řazen do kategorie diagramů struktury, neboť zachycuje statickou strukturu systému – typy objektů a vztahy mezi nimi. Diagram tříd je rovněž základem pro jiné druhy diagramů – diagram komponent a diagram nasazení.

Hlavními prvky třídy, rozhraní, abstraktní třídy, výčtové typy, stereotypy, generické třídy a asociační třídy. Dalšími důležitými prvky diagramu tříd jsou vztahy: asociace, generalizace a závislost. Také se mohou v diagramu tříd vyskytovat poznámky a balíčky.

Třída má vlastnosti a operace, což představuje její stav a chování. Většinou obsahuje tři oddíly: pro název třídy, oddíl atributů a oddíl operací. Je však možné přidat i speciální pojmenované oddíly navíc, nebo vynechat oddíly pro atributy a operace.



Obrázek 11: Ukázka vlastností ve formě atributů, vytvořeno pomocí VisualParadigm, zdroj: vlastní



Obrázek 12: Ukázka vlastností ve formě asociací, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Vlastnost třídy může být vyjádřena dvojím způsobem – prostřednictvím atributu, nebo asociace. Atributy se používají pro vyjádření jednodušších vlastností, zatímco asociace je vhodé použít pro významnější třídy. Porovnání vlastnosti jako atributu a jako asociace je na Obrázek 11 a Obrázek 12. Syntaxe pro atribut je následující:

```
[viditelnost] název ['‘:’ typ] ['‘[’násobnost‘]’] ['‘=’ implicitní_hodnota]
[{příznaky}]
```

Operace jsou funkce, které může instance třídy provést. Syntaxe pro operaci je:

```
[viditelnost] název ['‘(’seznam_parametrů‘)’] ['‘:’ návratový_typ]
[{příznaky}]
```

Parametry z nepovinného seznamu v kulatých závorkách operace mají formát:

```
[směr] název : typ [= implicitní_hodnota]
```

Pro lepší orientaci uvádí Tabulka 1 přehled významu jednotlivých částí syntaxe atributů a operací v jazyce UML.

Tabulka 1: Vysvětlení jednotlivých částí definice členů třídy, údaje z (Fowler, 2009)

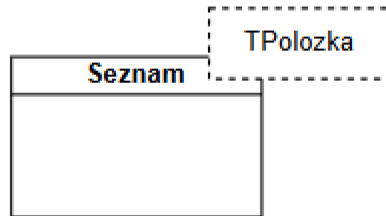
Název syntaktické části	**Výskyt**	**Význam**	**Příklady**
`viditelnost`	Atribut, operace	Určení, které části modelu mají přístup k danému členu	+ (public) - (private) # (protected) ~ (package)

název	Atribut, operace, parametr	Pojmenování	názevAtributu činnostMetody
typ	Atribut, parametr	Typ hodnot	: String : Integer : Boolean
návratový_typ	Operace	Typ hodnoty, kterou operace vrací	
násobnost	Atribut	Kolik hodnot může atribut obsahovat	[1] [0..1] [1..*] [*]
implicitní_hodnota	Atribut, parametr	Přednastavená hodnota	= "default text" = 0 = true
příznaky	Atribut, operace	Určení specifických vlastností, používají formát omezení	{readOnly} {static} {derived}
směr	Parametr	Určení vstupních a výstupních parametrů	in out inout

Specifické klasifikátory, jejichž význam je upřesněn rozšířením třídy, se běžně využívají při tvorbě diagramu tříd v jazyce UML. Rozšíření může být vyjádřeno buď pomocí grafických ornamentů, nebo formou stereotypů. Tabulka 2 uvádí přehled těchto prvků spolu s typem použitých rozšíření, jejich stručným popisem a názvem klasifikátoru z oficiální specifikace.

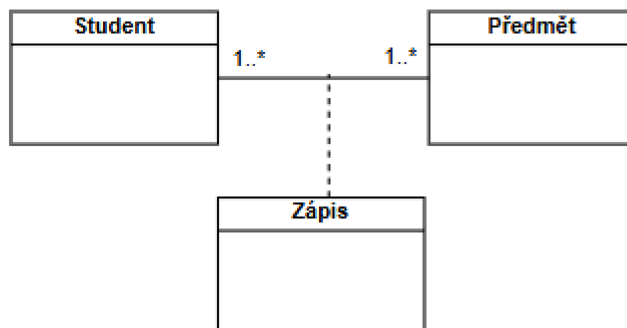
Tabulka 2: Prvky založené na třídě, údaje z (Unified Modeling Language, c2025)(Arlow, 2007)

Prvek	**Typ rozšíření**	**Význam**	**UML klasifikátor**
Rozhraní	Stereotyp «interface»	Definice chování bez implementace	Interface
Abstraktní třída	Většinou název třídy v kurzívě, ale může se vyskytovat ve formě stereotypu	Třída, která nemůže být instanciována, slouží jako společný základ pro odvozené třídy	Abstract Class
Výčtový typ	Stereotyp «enumeration»	Podtyp datového typu, který obsahuje výčet hodnot	Enumeration
Stereotyp	Stereotyp «stereotype»	Mechanika pro vytváření vlastních stereotypů	Stereotype
Generický klasifikátor	Ornament obdelníku s přerušovaným obrysem, který obsahuje parametry	Parametrizovaný klasifikátor	Template Class



Obrázek 13: Ukázka generické třídy, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Zvláštním případem třídy je asociační třída, která se vizuálně odlišuje tím, že je napojená pomocí přerušované čáry na asociaci. Používá se pro přiřazení vlastností k asociaci s mnohočetnou násobností.



Obrázek 14: Ukázka asociační třídy, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Podle úrovně detailů se diagram tříd často dělí na analytický a návrhový. Analytický je jednoduchým a přehledným modelem dané domény. Návrhový diagram tříd zahrnuje i implementační detaily a je modelován v pozdějších fázích vývoje.

Diagram tříd se používá pro různé účely v rámci vývoje softwaru. Při modelování doménového slovníku zachycuje základní pojmy a jejich vztahy v dané oblasti. Pro návrh spoluprací mezi objekty se využívá ke znázornění zapojení tříd do jednoduchých architektur a návrhových vzorů. Při návrhu databáze může diagram tříd reprezentovat logické databázové schéma, kde třídy odpovídají tabulkám a atributy sloupcům. Dopředné inženýrství umožňuje generovat kód z modelu, zatímco zpětné inženýrství slouží k odvození modelu z existujícího kódu pro jeho analýzu či dokumentaci.

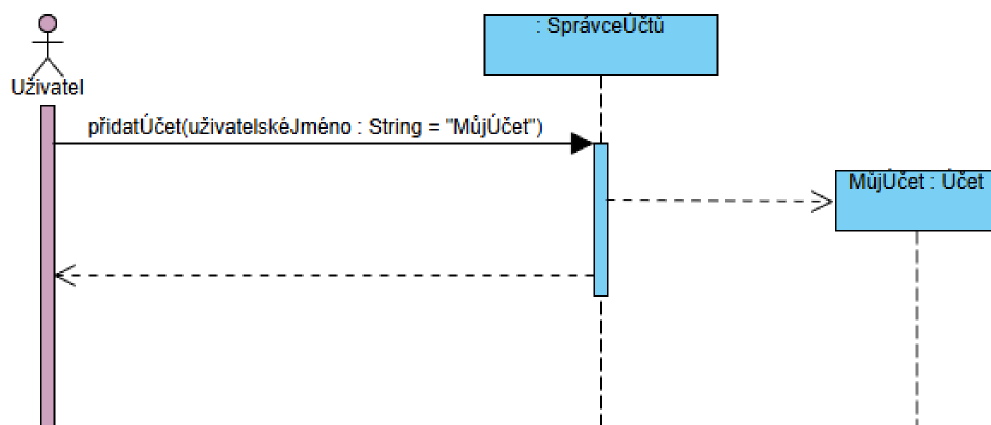
2.2 Sekvenční diagram

Sekvenční diagram patří mezi interakční diagramy z obecnější kategorie diagramů chování. Používá se pro modelování výměny zpráv mezi objekty v čase a zachycuje interakci v konkrétním případu užití. Je vhodný zejména pro upřesnění pořadí zpráv a popis spolupráce

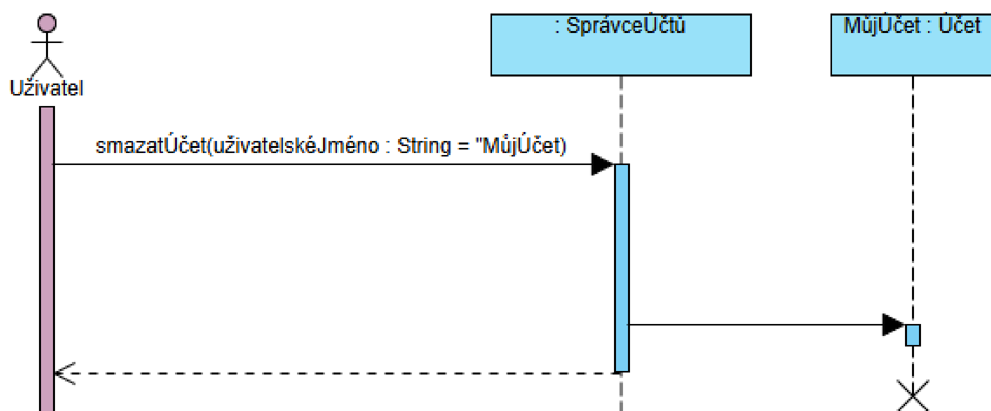
zúčastněných komponent systému. Základními prvky diagramu jsou aktéři, třídy a objekty, čáry života, aktivace, zprávy a fragmenty.

Z každého účastníka interakce vede čára života, která reprezentuje jeho existenci během komunikace. Čáry života jsou vertikální přerušované úsečky. Zakládání objektů je zobrazeno na Obrázek 15 a rušení objektů je na Obrázek 16. Podle čáry života se dá poznat založení, zrušení či předání řízení objektu.

Zprávy v sekvenčním diagramu reprezentují volání operací, přenosy dat nebo signály mezi klasifikátory. Jsou znázorněny orientovanými šipkami mezi čárami života. Dělí se na synchronní zprávy, které vyžadují odpověď, a asynchronní zprávy, které odpověď nečekají. UML umožňuje také modelovat návratové zprávy a zprávy zpožděné, ztracené či nalezené. Každá zpráva může obsahovat název operace, seznam parametrů a případně podmínku nebo iteraci, čímž podporuje přesnou specifikace chování systému v čase.



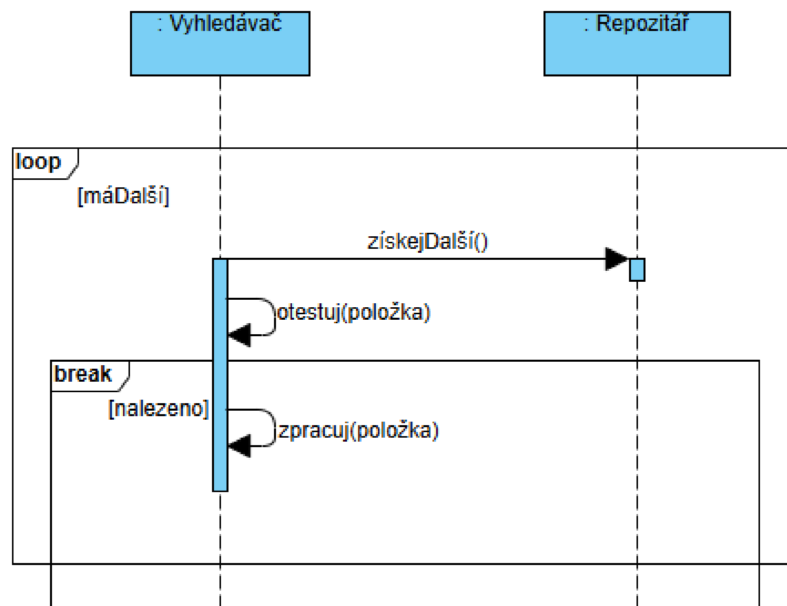
Obrázek 15: Ukázka Vytvoření objektu, vytvořeno pomocí VisualParadigm, zdroj: vlastní



Obrázek 16: Ukázka zrušení objektu, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Pro modelování složitějších scénářů v sekvenčním diagramu, například podmínek, opakování nebo paralelních interakcí, se využívají tzv. kombinované fragmenty. Jedná se o obdelníkové oblasti, které mají právě jeden operátor, jeden či více operandů a podmínky. Operátor určuje typ fragmentu a nejčastěji se používají následující:

- *opt* – má jeden operand, který se provede při splnění podmínky,
- *alt* – má alespoň dva operandy, spustí se ten, jehož kontrolní podmínka je splněna,
- *loop* – opakované vykonávání, dokud platí podmínka,
- *break* – má jeden operand, který v případě splnění podmínky ukončí cyklus,
- *par* – všechny operandy jsou provedeny souběžně.



Obrázek 17: Ukázka kombinovaných fragmentů, vytvořeno pomocí VisualParadigm, zdroj: vlastní

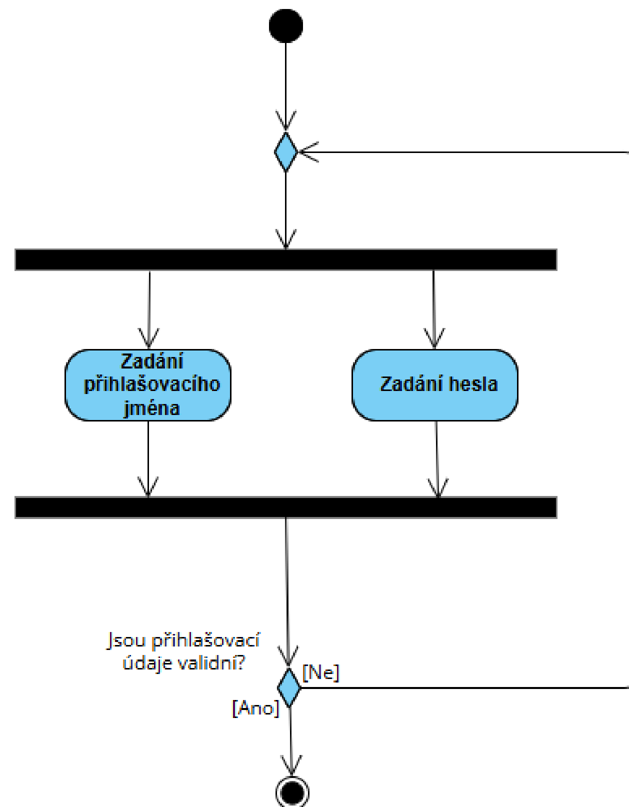
2.3 Diagram aktivit

Diagram aktivit je kolekcí různých druhů uzlů, které jsou propojeny toky. Modeluje aktivity složené z atomicky proveditelných akcí. Vizualizovaný tok událostí typicky reprezentuje případy užití, operace, metody, algoritmy a případně i podnikové procesy.

Prvky diagramu aktivit jsou:

- token – imaginární prvek, který „cestuje“ po uzlech diagramu, reprezentuje tok řízení, informace a data,
- uzly:
  - akční – popisují jednoduché i složené akce, které se spustí při „obdržení“ tokenu,
  - řídící – speciální uzly určující logiku postupu,
  - objektové – zastupují data,
- toky:
  - řídící – předává řízení mezi akcemi,
  - objektové – přenáší data ve formě objektů,
- signály – posílání a přijímání asynchronních oznámení o události,
- konektory – notace pro nesouvislé vykreslení toku,
- piny – rozšíření pro zaznamenání parametrů akce, což jsou vstupy a výstupy,
- transformace – specifikuje výpočet dále požadovaného formátu dat z dostupných informací,
- oddíly – dříve zvané plavecké dráhy rozdělují diagram aktivit podle odpovědností jednotlivých částí systému, které dané akce vykonávají,
- rozšiřující oblast – oblast pro zpracování kolekce objektů.

Řídicí uzly slouží k řízení toku provádění aktivity a určují, jak se tokeny pohybují mezi jednotlivými uzly. Aktivita začíná v počátečním uzlu, který generuje první token. Rozhodovací uzel větví tok na základě podmínky a uzel splynutí tyto alternativní větve sloučí dohromady do jednoho toku. Řídící uzly pro rozdělení vytváří paralelní větve, zatímco spojení čeká na příchod všech tokenů, než pustí tok řízení k dalšímu uzlu. Tok paralelní větve může být předčasně ukončen prostřednictvím koncového uzlu toku, ale ostatní toky zůstanou neovlivněny. Tyto uzly umožňují modelovat jak sekvenční, tak paralelní a podmíněné chování, což je zásadní pro přesné zachycení dynamiky systému. Aktivita končí, když se token dostane do koncového uzlu, bez ohledu na počet běžících větví.



Obrázek 18: Ukázka diagramu aktivit, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Diagram aktivit vznikl v rámci verze UML 1 jako rozšíření stavového diagramu, s nímž sdílí základní prvky a podobný způsob znázorňování řízeného toku. Od verze UML 2 však doznal významného přepracování podle Petriho sítí, což mu umožnilo přesněji vyjadřovat paralelní a asynchronní chování. Svou strukturou a zaměřením na sled činností zároveň připomíná klasické vývojové diagramy (*flowchart*). Díky bohaté sadě vyjadřovacích mechanismů je všestranným nástrojem pro modelování procesů, algoritmů i podnikových činností.

3 Automatické rozvžení grafů

V předchozích kapitolách byly UML diagramy představeny jako prostředek k modelování systémů. Obecněji je však možné UML diagram chápat jako graf, v němž vrcholy reprezentují prvky modelu a hrany vyjadřují vztahy mezi nimi. Tento abstraktnější pohled umožňuje využívat metody teorie grafů a aplikovat grafové algoritmy při práci s diagramy. V této kapitole se budeme zabývat problematikou automatického rozvržení grafů, které představuje poddisciplínu širší oblasti vykreslování grafů (*graph drawing*).

Formálně je graf definován jako uspořádaná dvojice $G = (V, E)$, kde V označuje množinu vrcholů a $E \subseteq V \times V$ je množina hran mezi těmito vrcholy. Podle charakteru hran rozlišujeme orientované a neorientované grafy. Orientovaná hrana je uspořádaná dvojice vrcholů (u, v), zatímco v případě neorientované hrany se jedná o neuspořádanou dvojici $\{u, v\}$. V kontextu teoretického rámce pro řešenou problematiku je vhodné představit i pojem dosažitelnosti. Vrchol v je dosažitelný z vrcholu u, pokud existuje posloupnost hran vedoucí z u do v. Tato formální specifikace poskytuje teoretický základ pro diskusi o metodách vizualizace a automatického rozvržení grafů.

Disciplína vykreslování grafů se zabývá vizuální reprezentací grafů tak, aby byly srozumitelné a vhodně indikovaly podstatné informace o daném grafu. Mezi hlavní oblasti zájmu vykreslování grafů patří zejména analýza vlastností grafů relevantních k jejich grafické reprezentaci, metriky pro hodnocení výsledného zobrazení a návrh algoritmů pro vizualizaci grafů.

Automatické rozvržení grafů, které je poddisciplínou oblasti vykreslování grafů, se zabývá algoritmickým určením polohy vrcholů a vedením hran grafu v rovině. Součástí tohoto procesu jsou určitá požadovaná kritéria spolu s odpovídajícími metrikami, na něž se zaměřuje následující podkapitola. Poté je věnována pozornost jednotlivým druhům, odpovídajícím metodám s příklady softwarových implementací, které se pro automatické rozvržení používají.

3.1 Kritéria a metriky pro rozvržení grafu

Pro účely vykreslování grafů lze stanovit různá kritéria v závislosti na tom, k jakému účelu vizualizace slouží a jaké informace má zdůraznit. Studie ukazují, že algoritmy automatického rozvržení by měly prioritně optimalizovat ta kritéria, která mají empiricky prokázaný vliv na uživatelskou srozumitelnost, a teprve následně zohledňovat méně významné estetické či prostorové požadavky (Purchase, 1997). Splnění požadovaných vlastností je ověřeno pomocí

příslušných metrik, které poskytují dobře měřitelné kvantitativní vyhodnocení vizualizace. V následujících oddílech jsou rozebrány nejčastěji používaná kritéria a metriky.

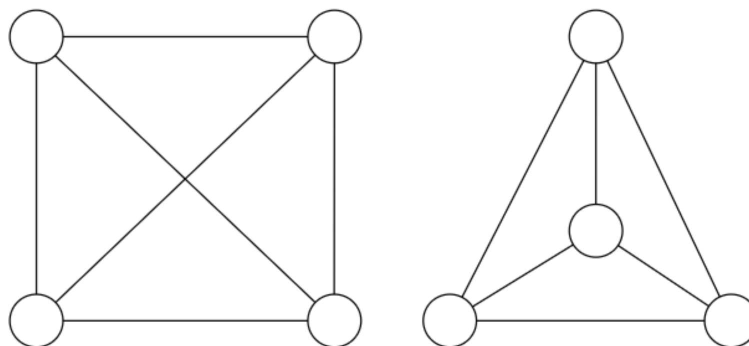
Křížení hran

Jedním ze základních požadavků je minimalizace křížení hran. Překřížené hrany v grafu výrazně ztěžují čitelnost a můžou vést k chybnému pochopení struktury grafu (Purchase, 1997).

Speciální skupina grafů, které je možné zobrazit v rovině bez křížení hran, se nazývají rovinné grafy a tato jejich vlastnost planarita. Ostatní nerovinné grafy při libovolném rozvržení obsahují alespoň jedno křížení.

Minimalizace křížení hran je výpočetně náročná, protože jde o NP-těžký problém, a v praxi se proto uplatňují především heuristické přístupy poskytující suboptimální řešení.

Hodnocení tohoto kritéria probíhá na základě absolutního počtu křížení nebo jejich poměru k celkovému počtu hran.



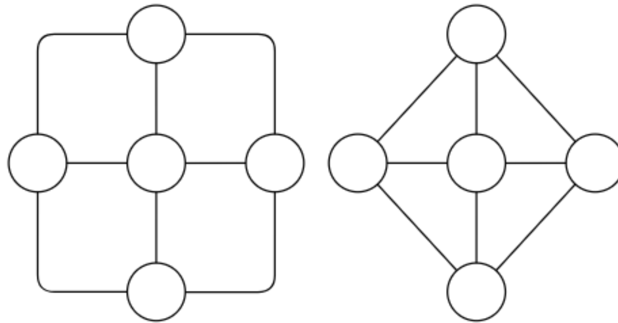
Obrázek 19: Stejný graf vykreslení s křížením a bez křížení hran, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Ortogonalita a ohýbání hran

Ortogonalita v kontextu vykreslování grafů označuje způsob vedení hran výhradně vodorovným a svislým směrem, tedy v pravých úhlech. Jedním z dílčích parametrů ortogonálního rozvržení je počet ohybů hran, tedy míst, kde se směr hrany mění. Nadměrný počet ohybů může negativně ovlivnit přehlednost a ztížit sledování spojení mezi vrcholy, jak dokládají uživatelské studie (Purchase, 1997).

Minimalizace ohybů hran v planárním ortogonálním vykreslení se řadí mezi NP-těžké úlohy, avšak pro pevně dané ukotvení hran je problém řešitelný v polynomiálním čase (Tamassia, *n.d.*).

Metrika vyhodnocuje buď celkový počet ohybů v celém grafu, nebo průměrný počet ohybů na jednu hranu.

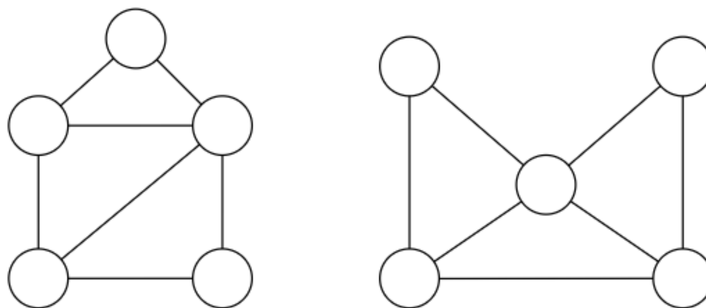


Obrázek 20: Stejný graf s ohýbáním a bez ohýbání hran, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Symetrie

Symetrie v rozvržení grafu označuje prostorové uspořádání vrcholů a hran tak, aby se geometricky odrážela topologická pravidelnost dané struktury. U grafů, které obsahují symetrické podstruktury, může jejich zachování usnadnit uživateli rozpoznávání opakujících se vzorů a celkově urychlit orientaci v diagramu. Přesto uživatelské studie (Purchase, 1997) ukazují, že zachování symetrie má na subjektivně vnímanou čitelnost menší vliv než minimalizace křížení či ohybů hran.

Měření lze provádět pomocí několika formálních metrik. Jednou z nejpoužívanejších je detekce geometrických symetrií, která se soustředí na hledání os zrcadlové symetrie nebo středů rotační symetrie v kresbě grafu (Kobourov, 2017).



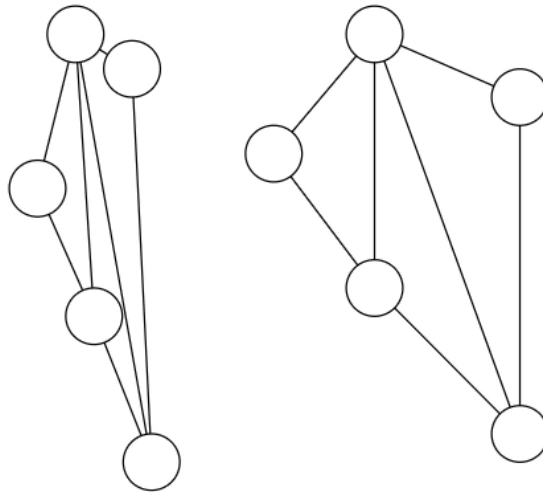
Obrázek 21: Stejný graf s různou úrovní symetrie, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Rozmístění vrcholů a úhel hran

Samozřejmým požadavkem na rozvržení grafu je, aby se vrcholy ve vizualizaci nepřekrývaly. Kromě této minimální vzdálenosti mezi vrcholy se však na rozmístění vrcholů klade i požadavek na určitou rovnoměrnost, aby nevznikaly oblasti s nadměrnou hustotou nebo naopak výrazně prázdné. Případně lze také definovat požadavek na bližší vzdálenost vrcholů,

které jsou vzájemně propojeny. Dalším důležitým kritériem pro dostatečnou čitelnost grafu je velikost minimálních úhlů, neboť příliš malé úhly mezi hranami, které jsou incidentní ke stejnému vrcholu, ztěžují jejich vizuální rozlišení a mohou vést k nesprávné interpretaci vazeb.

Hodnocení těchto aspektů se provádí pomocí měření nejmenší vzdálenosti mezi dvojicemi vrcholů, rozptylu vzdáleností mezi vrcholy pro posouzení rovnoměrnosti, a měření minimálního či průměrného úhlu mezi hranami vycházejícími z jednoho vrcholu.

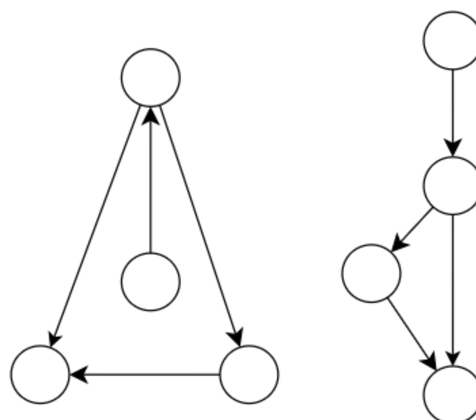


Obrázek 22: Stejný graf s různými vzdálenostmi vrcholů, vytvořeno pomocí VisualParadigm, zdroj: vlastní

Zachování směru či hierarchie

V orientovaných grafech bývá žádoucí konzistentní směřování hran, například shora dolů nebo zleva doprava. Neméně významným a souvisejícím požadavkem je zachování hierarchie, tedy uspořádání vrcholů do odpovídajících úrovní podle jejich postavení ve struktuře grafu.

Metriky pro hodnocení tohoto kritéria zahrnují počet hran vedoucích proti definovanému směru, počet porušených hierarchických úrovní či míru konzistence směru v celém grafu.



Obrázek 23: Stejný graf bez zachování směru a se zachováním směru, vytvořeno pomocí VisualParadigm, zdroj: vlastní

3.2 Metody pro automatické rozvržení

Jednotlivé metody automatického rozvržení jsou navrženy tak, aby optimalizovaly specifická kritéria zobrazení a často přímo určují výsledný typ rozvržení. Ke každé metodě existují obecné algoritmické postupy a principy (např. Sugiyama framework pro hierarchické rozvržení), které lze také chápat jako metodu v užším slova smyslu. Na nejnižší úrovni jsou konkrétní implementace, které mohou produkavat odlišné výstupy, přestože implementují stejné metody. Každý přístup se vyznačuje odlišnými oblastmi vhodného použití, silnými stránkami i omezeními.

Mezi nejčastěji používané metody patří:

- hierarchické metody,
- force-directed metody,
- přístup topologie–tvar–metriky (topology-shape-metrics),
- kruhové metody,
- stromové metody (Automatic Diagram Layout, c2024).

V následujících oddílech jsou jednotlivé přístupy popsány podrobněji, včetně principu činnosti, typu rozvržení, optimalizovaných omezení, typických oblastí použití a ukázek vykreslení.

Hierarchické metody

Hierarchické metody se snaží uspořádat prvky grafu do vrstev či úrovní podle jejich postavení v logické struktuře. Princip algoritmu spočívá v rozdělení vrcholů do vrstev podle jejich úrovně v hierarchii a v umístění hran tak, aby směřovaly převážně jedním směrem (např. shora dolů nebo zleva doprava). Pořadí vrcholů v jednotlivých vrstvách je voleno tak, aby se minimalizoval počet křížení hran.

Typické grafy s hierarchickým rozvržením jsou procesní diagramy, organizační schémata, diagramy závislostí a modely pracovních toků.

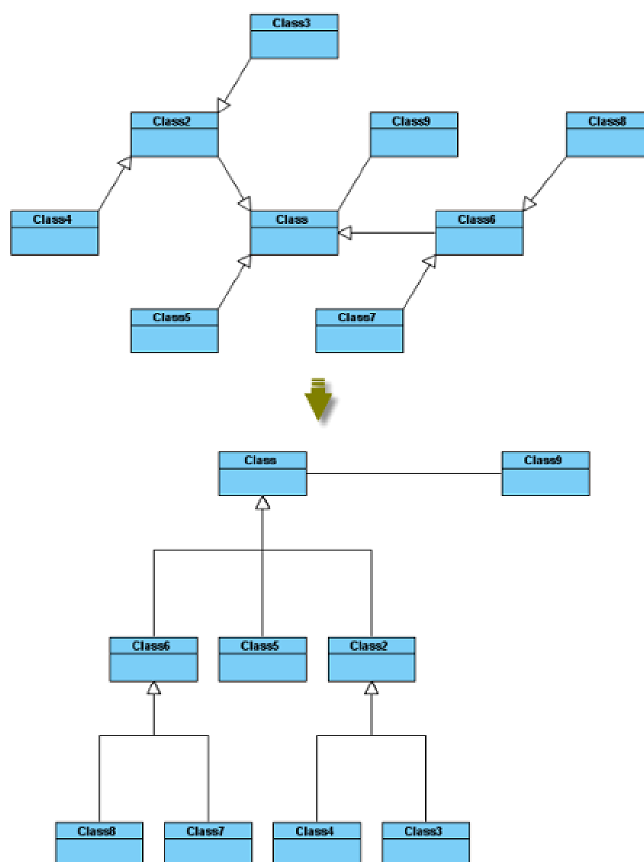
Většina moderních implementací hierarchického rozvržení vychází z tzv. Sugiyama frameworku, původního algoritmického schématu pro vrstvené rozvržení publikovaného Kozo Sugiyamou a kol. v roce 1981 (Sugiyama, 1981). Sugiyama framework rozděluje výpočet rozvržení do čtyř fází:

- Odstranění cyklů (*Cycle removal*) pro získání acyklického grafu (DAG).
- Přiřazení vrstev (*Layer assignment*) jednotlivým vrcholům, vytvoření pomocných vrcholů a hran.
- Uspořádání vrcholů ve vrstvách (*Vertex ordering*) tak, aby se předešlo křížení.
- Přiřazení souřadnic (*Coordinate assignment*) (Mazetti, 2012).

Konkrétní implementace lze nalézt například v následujích knihovnách:

- Graphviz – dot engine[1],
- Eclipse Layout Kernel – layered[2],
- Dagre[3],
- MSAGL – layered layout[4],
- yFiles – hierarchical layout[5].

[1] https://graphviz.org/docs/layouts/dot/
[2] https://eclipse.dev/elk/reference/algorithms/org-eclipse-elk-layered.html
[3] https://github.com/dagrejs/dagre/wiki
[4] https://microsoft.github.io/msagljs/docs/api/
[5] https://docs.yworks.com/yfiles-html/dguide/layout/hierarchical_layout.html



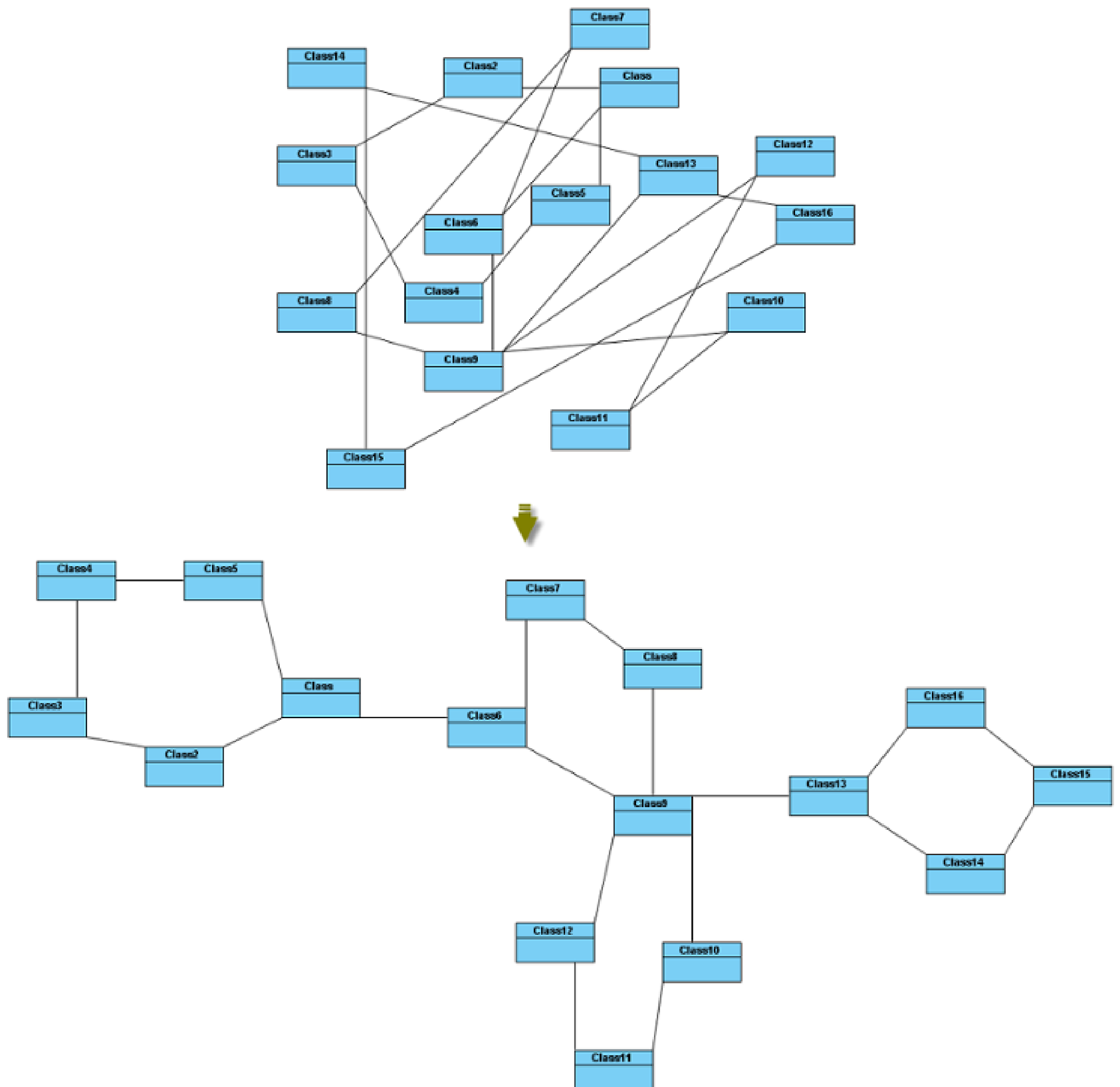
Obrázek 24: Ukázka aplikace hierarchickéo rozvržení, převzato z (Automatic Diagram Layout, c2024)

Force-directed metody

Force-directed metody představují skupinu algoritmů pro automatické rozvržení grafů založenou na fyzikální simulaci (Tamassia, *n.d.*). Podle zvoleného fyzikálního modelu či typu sil lze tyto metody rozdělit do různých podskupin. Jednou z nejpoužívanějších podskupin je rozvržení podle pružinového modelu (*spring model layout*), který modeluje hrany jako pružiny přitahující spojené vrcholy k sobě a zároveň aplikuje odpudivé síly mezi všemi vrcholy. Algoritmus simuluje působení těchto sil s cílem najít rovnovážný stav systému, v němž jsou síly vyrovnané.

Výsledkem je rozvržení, které se často označuje jako organické (*organic layout*), protože přirozeně uspořádává prvky do vyvážených tvarů a sousedící prvky jsou umístěny pospolu. Organické rozvržení obvykle odhaluje přirozené symetrie a shluky v grafu (YFiles for HTML Documentation, c2025). Jeho kvalita však může výrazně záviset na počátečním rozmístění, které bývá často generováno náhodně. Tyto metody navíc zpravidla neberou v úvahu směrovost hran, a proto nemusí zřetelně vystihovat hierarchické vztahy v grafu.

Organické diagramy se běžně používají pro vizualizaci komplexních dat, například v bioinformatice, podnikových sítích, vizualizaci sociálních sítí, vizualizaci sítí nebo správě systémů (YFiles for HTML Documentation, c2025).



Obrázek 25: Ukázka aplikace organického rozvržení, převzato z (Automatic Diagram Layout, c2024)

Konkrétní implementace lze nalézt například v následujích knihovnách:

- Graphviz – neato[6] nebo fdp[7] engine,
- Eclipse Layout Kernel – Force[8] nebo Stress[9],
- MSAGL – IPSepCola,
- yFiles – organic layout[10].

Přístup topologie-tvar-metriky

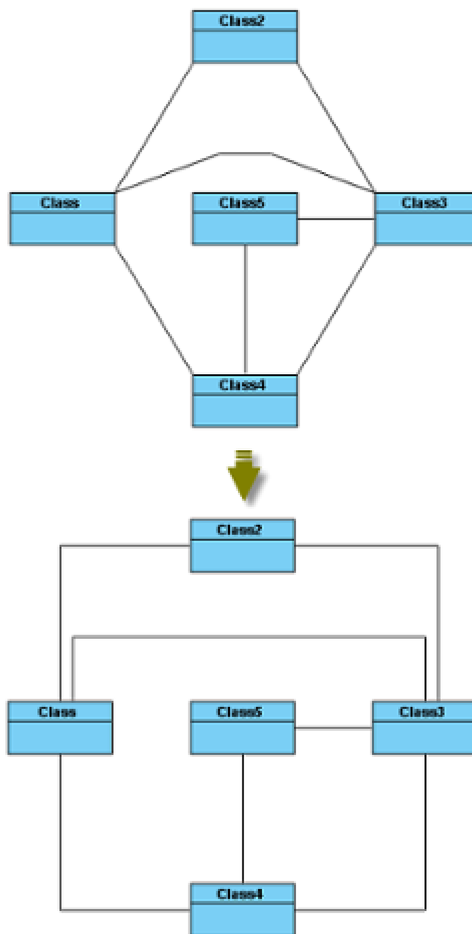
Topology-shape-metrics (TSM) přístup je třífázový algoritmický rámec, jehož výsledkem je ortogonální rozvržení. První fází je planarizace, která určuje topologii grafu, tedy uspořádání a sousednosti vrcholů. Pokud vstupní graf není planární, jsou během planarizace vytvořeny pomocné vrcholy. Druhá fáze je ortogonalizace a jejím výsledkem je geometrický tvar hran podle zvolených omezení, což je většinou požadevek na ortogonální vedení hran. Poslední je kompakce, ve které je vytvořena finální podoba grafu označovaná jako metrika. Kompakce zahrnuje odstranění pomocných vrcholů a výpočet souřadnic (Didimo, c2025).

Ortogonální rozvržení je často využíváno v softwarovém inženýrství, reprezentaci databázových schémat, či reprezentaci znalostí.

Plnohodnotnou implementaci přístupu TSM nabízí knihovna yFiles[11]. V ostatních nástrojích bývá ortogonální rozvržení dostupné pouze jako volitelný parametr v rámci jiných metod.

---

[6] https://graphviz.org/docs/layouts/neato/
[7] https://graphviz.org/docs/layouts/fdp/
[8] https://eclipse.dev/elk/reference/algorithms/org-eclipse-elk-force.html
[9] https://eclipse.dev/elk/reference/algorithms/org-eclipse-elk-stress.html
[10] https://docs.yworks.com/yfiles-html/dguide/layout/organic_layout.html#organic_layout
[11] https://docs.yworks.com/yfiles-html/api/OrthogonalLayout.html



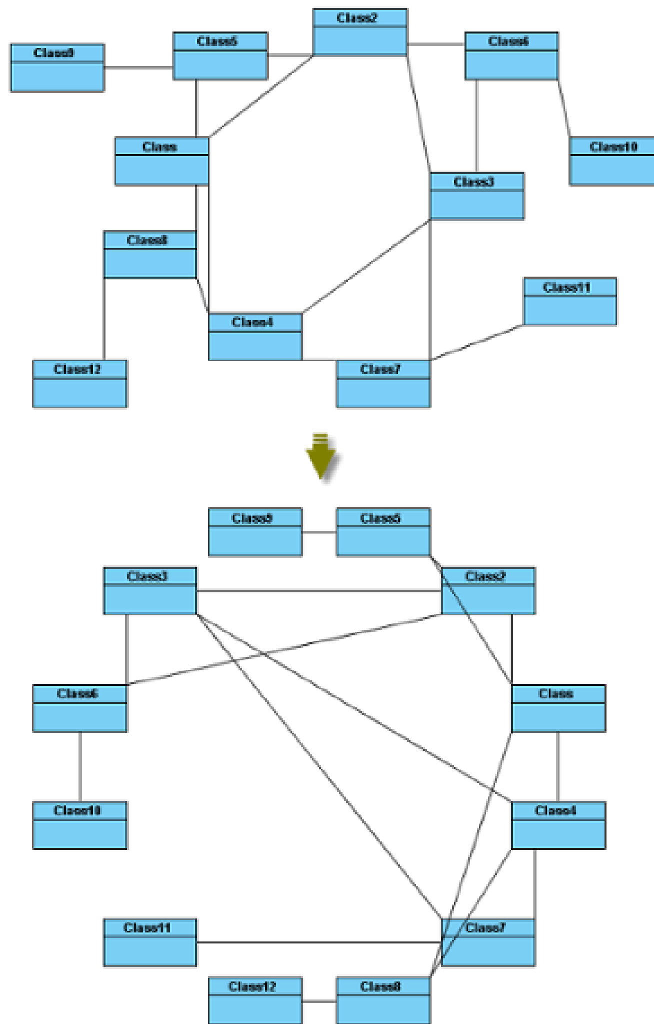
Obrázek 26: Ukázka aplikace ortogonálního rozvržení, převzato z (Automatic Diagram Layout, c2024)

Kruhové metody

Kruhové metody uspořádávají vrcholy grafu na obvod jedné či více kružnic. Kruhové metody jsou vhodné pro vizualizaci cyklů, sociálních sítí, telekomunikačních sítí či molekulárních struktur.

Implementace kruhového rozvržení nabízejí například yFiles[12] nebo Graphviz[13].

[12] https://docs.yworks.com/yfiles-html/dguide/layout/circular_layout.html#circular_layout
[13] https://graphviz.org/docs/layouts/circo/



Obrázek 27: Ukázka aplikace kruhového rozvržení, převzato z (Automatic Diagram Layout, c2024)

Stromové metody

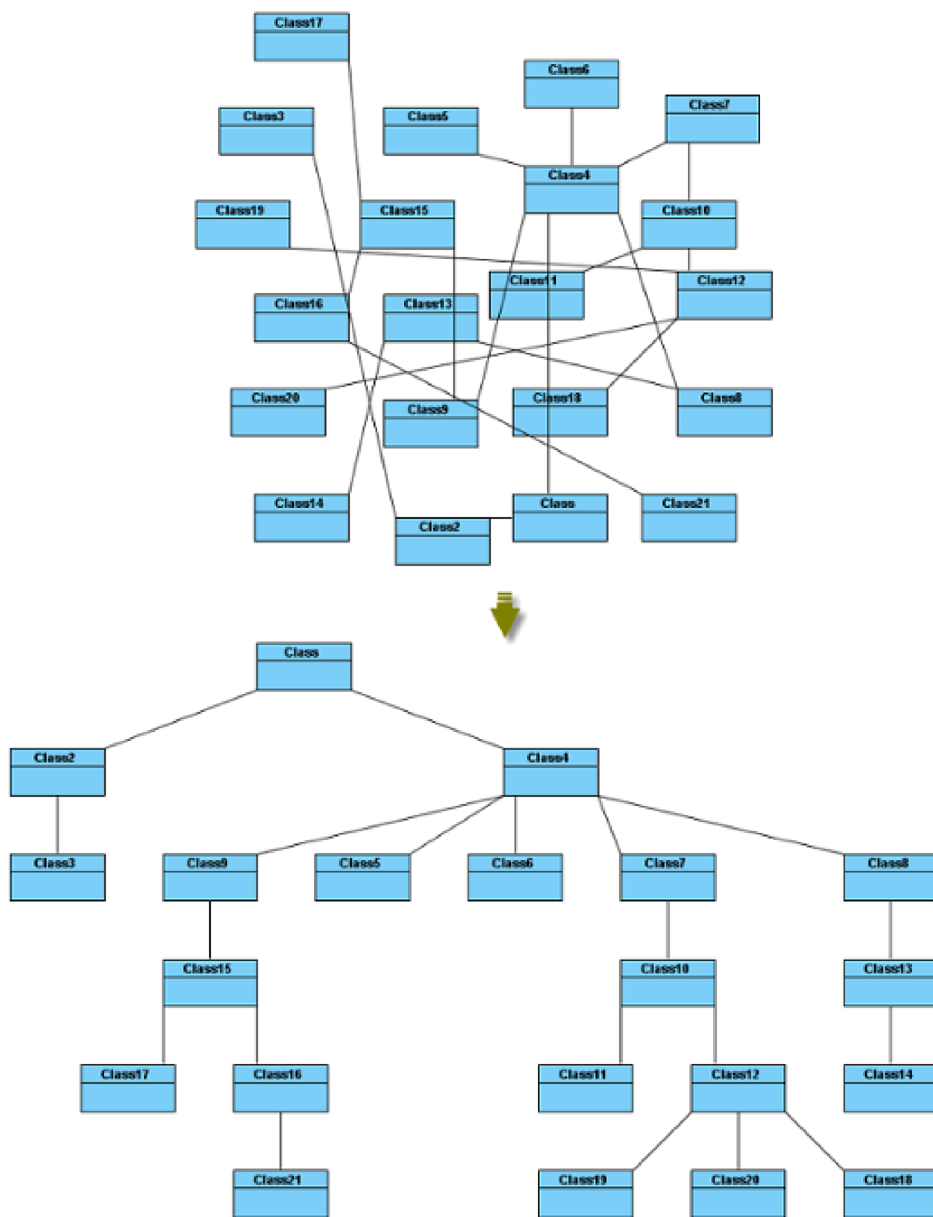
Stromové metody představují skupinu algoritmů určených pro vizualizaci stromových struktur, tedy souvislých neorientovaných acyklických grafů, v nichž mezi libovolnými dvěma vrcholy existuje právě jedna cesta.

Kritéria pro stromové metody zahrnují minimalizaci celkového prostoru diagramu, zobrazení symetrie, vyvážení podstromů a zachování konzistentní vzdálenosti mezi úrovněmi.

Implementace stromového rozvržení jsou dostupné v knihovnách yFiles[14] nebo ELK[15].

---

[14] https://docs.yworks.com/yfiles-html/dguide/layout/tree_layouts.html#tree_layouts
[15] https://eclipse.dev/elk/reference/algorithms/org-eclipse-elk-mrtree.html



Obrázek 28: Ukázka aplikace stromového rozvržení, převzato z (Automatic Diagram Layout, c2024)

4 Požadavky pro praktickou část

Cílem praktické části je formálně definovat jazyk pro textový popis diagramu tříd a implementovat generátor diagramů, který tento jazyk přijímá na vstupu. K naplnění těchto cílů je nezbytné prvně formulovat funkční požadavky, které určují potřeby pro jazykové konstrukce a očekávané chování aplikace. Dále je třeba stanovit i nefunkční požadavky, neboť vymezují vhodné softwarové nástroje pro realizaci aplikace.

Tato kapitola se rovněž zabývá analýzou obdobných existujících řešení, jejichž zhodnocení vytváří praktický rámec pro návrh vlastního jazyka a poskytuje podklady pro rozhodnutí, zda mohou sloužit jako základ navrhovaného jazyka či implementace. Využití již vyvinutých a udržovaných knihoven představuje preferovaný přístup, neboť eliminuje potřebu znovu vytvářet existující funkčnost a umožňuje zaměřit se na rozšiřující funkcionality programu.

4.1 Požadavky cílové aplikace

Funkční požadavky aplikace jsou:

- generování všech základních prvků UML diagramu tříd,
- podporu postupného rozvoje diagramu prostřednictvím vícesnímkové animace,
- automatické zvýraznění provedených změn,
- možnost specifikace preferovaného umístění prvků v diagramu,
- zachování konzistentní pozice prvků na všech verzích diagramu.

Mezi nefunkční kritéria pro implementaci aplikace patří:

- ovládání z příkazové řádky,
- textový vstup,
- výstup ve formátu SVG,
- spuštění v prostředí operačních systémů Linux i Windows.

4.2 Porovnání existujích nástrojů

Základem vyvíjené aplikace je generování diagramů tříd z textového popisu, které je již součástí existujících nástrojů. Pro účely této práce byly k porovnání zvoleny nástroje nomnoml,

Mermaid a PlantUML, neboť jsou volně dostupné, bezplatné, publikované jako otevřený software a disponují přehledně strukturovanou dokumentací spolu se zpřístupněným zdrojovým kódem. Všechny vybrané generátory nabízí export obrázků ve formátech PNG a SVG. Většina nástrojů pro generování UML z textu nepodporuje standard UML ani formát XMI a neprovádí validaci vůči metamodelu, což omezuje jejich interoperabilitu.

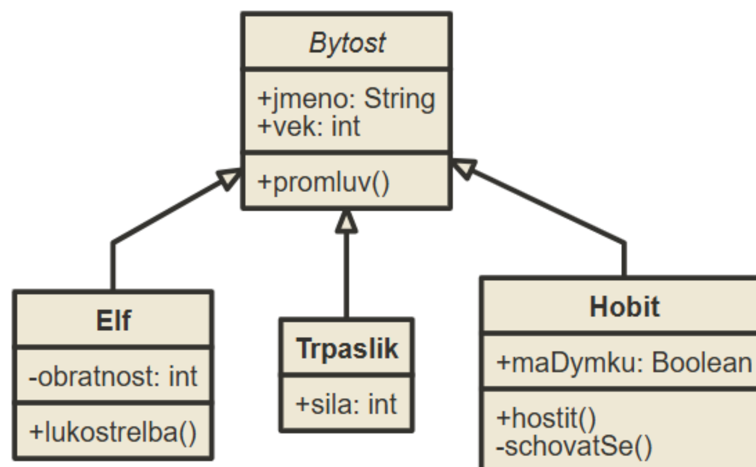
Nejdříve jsou stručně představeny jednotlivé nástroje, se zaměřením na jejich silné stránky, podporované diagramy, možnosti rozvržení a dokumentaci. Následná analýza se zaměřuje na podporu a syntaxi klíčových konstrukcí UML diagramů tříd a porovnání efektivity zápisu. Výsledkem je určení, které jazykové konstrukce se nejvíc hodí pro základ vlastního jazyka.

4.2.1 nomnoml

JavaScriptová knihovna nomnoml (Kallin, [2024]) podporuje pouze generování diagramů tříd. Syntaxe jazyka je minimalistická a přímo odráží vizuální podobu výsledného diagramu, což napomáhá snažšímu pochopení syntaxe a rychlejší práci.

Pro rozvržení prvků používá modifikaci knihovny Dagre, a proto má výsledná grafická podoba hierarchické rozvržení.

Referenční dokumentace jazyka je dostupná přímo ve webové aplikaci s editorem a nabízí stručné seznámení s možnostmi tohoto nástroje.



Obrázek 29: Diagram tříd vytvořený pomocí nomnoml, zdroj: vlastní

4.2.2 Mermaid

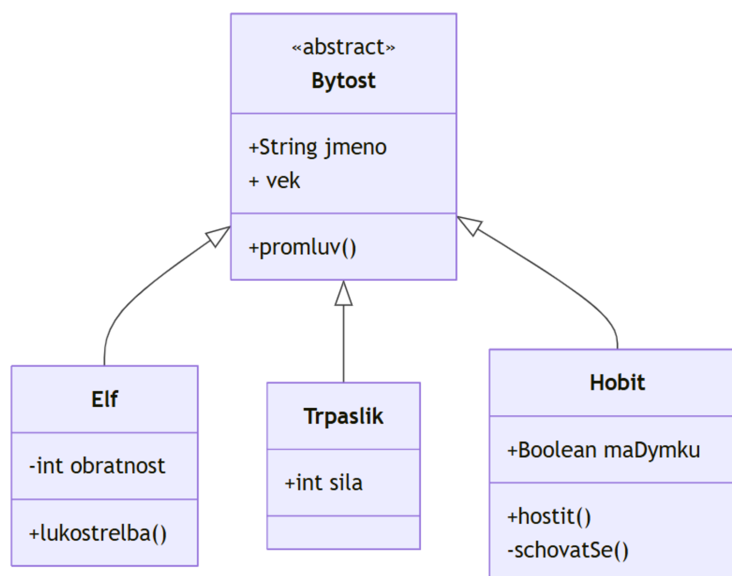
Mermaid (Mermaid Documentation, [2025]) je JavaScriptový nástroj určený pro jednoduchou a rychlou dokumentaci zdrojového kódu, který podporuje širokou škálu typů diagramů, včetně diagramů tříd, sekvenčních nebo entity-relationship diagramů. Přesto však chybí možnost

generovat některé UML diagramy, jako například diagram aktivit, místo kterého musí vystačit vývojový, či stavový diagram.

Pro rozvržení prvků v diagramu tříd je defaultní možností hierarchické rozvržení z knihovny Dagre, přičemž v placené verzi (MermaidChart) je možné použít i adaptivní rozvržení z knihovny ELK.

Nejen dokumentace je dostupná na oficiálních webových stránkách, ale i seznam výukových materiálů. V online editoru jsou navíc dostupné ukázky kódu.

Na Obrázek 30 je uveden ukázkový výstup vytvořený pomocí nástroje Mermaid. Zápis atributů v tomto diagramu neodpovídá formální notaci definované ve specifikaci UML, což je důsledkem omezení syntaxe daného nástroje. Pro srovnání jsou na Obrázek 29 a Obrázek 31 zobrazeny diagramy vytvořené v alternativních nástrojích, které umožňují zápis atributů v souladu s nároky UML standardu. Formální zápis atributů a operací je probrán v kapitole 2.1.



Obrázek 30: Diagram tříd vytvořený pomocí Mermaid, zdroj: vlastní

4.2.3 PlantUML

PlantUML (PlantUML Language Reference Guide, 2025) je všestranný nástroj napsaný v jazyce Java, který podporuje generování nejen většiny standardních UML diagramů, včetně diagramu tříd, aktivit, stavů, případů užití a sekvenčních diagramů.

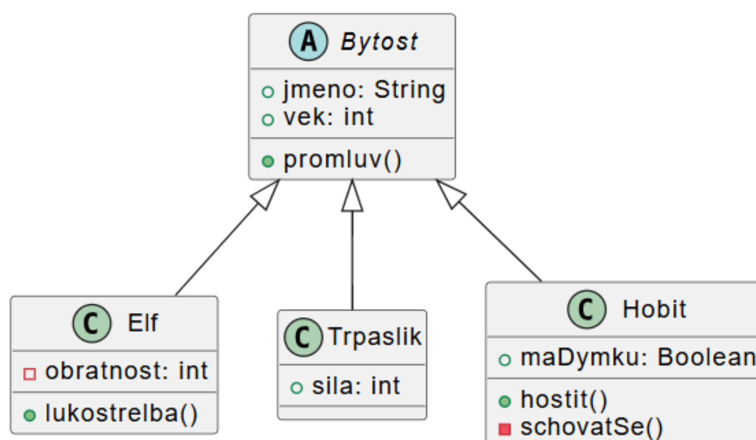
Primárním plně podporovaným enginem pro automatické rozvržení diagramů tříd v PlantUML je Graphviz dot, který implementuje hierarchické metody rozmístění prvků. Při využití výchozího rozvržení vzniká závislost na externím programu Graphviz, jehož implementace je

napsána v jazyce C. Alternativu představují enginy Smetana a ELK. Smetana[16], verze dot implementovaná v jazyce Java, nabízí zjednodušenou variantu původního algoritmu, což může vést k méně optimálním výsledkům. ELK[17] je knihovna v rané fázi vývoje, není stabilní a jeho zahrnutí do PlantUML je určené zejména pro testovací účely. Momentálně použití ELK není funkční[18].

Mimo přímou volbu algoritmu pro generování rozvržení poskytuje PlantUML při použití výchozího enginu možnost ovlivnit výsledné uspořádání diagramu prostřednictvím několika mechanismů. Patří mezi ně zejména specifikace směru vykreslování, úprava pořadí prvků či doplnění explicitní specifikace směru v definici vztahů a nebo v dokumentaci doporučované využití neviditelných vztahů.

Základní dokumentace k PlantUML je dostupná na oficiálních webových stránkách projektu, kde je zpracována přehledným a srozumitelným způsobem a doplněna názornými ukázkami. Podrobnější informace poskytuje referenční příručka jazyka, která slouží jako hlavní zdroj pro studium syntaxe a možností nástroje. Dalším užitečným zdrojem jsou stránky *The Hitchhiker's Guide to PlantUML!* (The Hitchhiker's Guide to PlantUML!, *n.d.*), které rovněž obsahují množství informací a příkladů rozšiřujících oficiální dokumentaci.

Mezi přednosti nástroje PlantUML patří mimo jiné možnost skrývání vybraných prvků diagramu, podpora přidávání hypertextových odkazů či tooltipů k jednotlivým elementům.



Obrázek 31: Diagram tříd vytvořený pomocí PlantUML, zdroj: vlastní

[16] https://plantuml.com/smetana02
[17] https://plantuml.com/elk
[18] https://forum.plantuml.net/18998/elk-dont-work

Za určité omezení nástroje PlantUML lze považovat zobrazování ikon vlevo od názvu prvků, jak je patrné na Obrázek 31. Tento způsob zobrazení neodpovídá běžně používané notaci UML, která je používaná v oficiálních CASE nástrojích. Další odchylkou je používání ikon pro viditelnost prvků, což je ale možné vypnout pomocí následujícího příkazu:

```
skinparam classAttributeIconSize 0
```

4.2.4 Srovnání existujících jazyků

Tabulka 3 porovnává základní syntaktické konstrukce pro definici UML prvků diagramu tříd v jednotlivých jazycích. Pro zjednodušení nejsou uvedeny některé nepovinné části a všechny kontexty. Pokud je daný prvek zadefinován v rámci nadřazeného prvku, je v příslušné buňce nadřazený prvek uveden jako `x`, případně `y`, pokud jsou dva. U definice některých prvků je nutné, aby před jejich definicí byly v textovém popisu diagramu explicitně deklarovány nadřazené prvky. Příkladem je definice asociační třídy v PlantUML, kdy musí být vztah, na který je tato třída navázána (pro kód z tabulky zapsaný například jako `x - y`), v kódu uveden před danou definicí.

Tabulka 3: Základy syntaktické konstrukce UML prvků ve vybraných jazycích, zdroj: dokumentace porovnávaných nástrojů

UML prvek	nomnoml	Mermaid	PlantUML
Třída	`[nazev]`	`class nazev`	`class nazev`
Rozhraní	*nepodporováno*	*viz řádek Stereotyp*	`interface nazev`
Abstraktní třída	`[<abstract>nazev]`	*viz řádek Stereotyp*	`abstract nazev`
Výčtový typ	*nepodporováno*	*viz řádek Stereotyp*	`enum nazev`
Genericita	`x<T>`	`x~T~`	`x<T>`
Poznámka	`x --[<note>obsah]`	`note for x "obsah"`	`note "obsah" as nazev` `nazev .. x`
Balíček (namespace)	`[<package>nazev]`	`namespace nazev {` `}`	`namespace nazev {` `}`
Stereotyp	*nepodporováno*	`class x {<<nazev>>}` `<<nazev>> x`	`class x <<nazev>>`

<table>
<tr><td>Atribut</td><td>[x|nazev]</td><td>x : nazev</td><td>x : nazev</td></tr>
<tr><td>Metoda</td><td>[x||nazev()]</td><td>x : nazev()</td><td>x : nazev()</td></tr>
<tr><td>Asociace</td><td>-</td><td>--</td><td>-</td></tr>
<tr><td>Orientovaná asociace</td><td>-></td><td>--></td><td>-></td></tr>
<tr><td>Agregace</td><td>o-</td><td>o--</td><td>O-</td></tr>
<tr><td>Kompozice</td><td>+-</td><td>*-</td><td>*-</td></tr>
<tr><td rowspan="2">Generalizace</td><td rowspan="2"><:-</td><td rowspan="2"><|--</td><td><|-</td></tr>
<tr><td>^-</td></tr>
<tr><td rowspan="2">Realizace</td><td rowspan="2"><:--</td><td rowspan="2"><|..</td><td><|.</td></tr>
<tr><td>^.</td></tr>
<tr><td>Multiplicita</td><td>[x]1-0..*[y]</td><td>x "1" -- "0..*" y</td><td>x "1" - "0..*" y</td></tr>
<tr><td>Popisek vztahu</td><td>nepodporováno</td><td>x -- y : popisek</td><td>x - y : popisek</td></tr>
<tr><td>Asociační třída</td><td>nepodporováno</td><td>nepodporováno</td><td>(x,y) . nazev</td></tr>
</table>

Pro porovnání efektivity vybraných jazyků byla zvolena metrika založená na velikosti vstupu, konkrétně na celkovém počtu řádků zdrojového kódu potřebného pro vygenerování ukázkového diagramu tříd, který je vyobrazen na Obrázek 29, Obrázek 30 a Obrázek 31. Ve všech případech byl vstupní kód zapsán v podobě s minimálním počtem řádků, která je pro daný nástroj platná a umožňuje vygenerovat požadovaný ukázkový diagram.

Kód 1: Popis diagramu pro nomnoml

```
1. [<abstract>Bytost|+jmeno: String;+vek: int|+promluv()] <:- [Elf|-
   obratnost: int|+lukostrelba()]
2. [Bytost] <:- [Trpaslik|+sila: int]
3. [Bytost] <:- [Hobit|+maDymku: Boolean|+hostit();-schovatSe()]
```

V Kód 2 pro nástroj Mermaid je nutno uvést na prvním řádku o jaký diagram se jedná, což pro komplexnější diagramy nijak zásadně velikost vstupu neovlivní. Co ale může velikost vstupu pro větší diagramy významně snížit je implicitní deklarace obyčejných tříd.

Kód 2: Popis diagramu pro Mermaid

```
1. classDiagram
2. Bytost: +String jmeno
3. Bytost: + vek
4. Bytost: +promluv()
5. <<abstract>> Bytost
6. Elf: -int obratnost
7. Elf: +lukostrelba()
8. Trpaslik: +int sila
9. Hobit: +Boolean maDymku
10. Hobit: +hostit()
11. Hobit: -schovatSe()
12. Bytost <|-- Elf
13. Bytost <|-- Trpaslik
14. Bytost <|-- Hobit
```

Z Kód 3 byly vynechány úvodní direktivy (`@startuml`, `@enduml`) a definice tříd použitých ve vztazích, což činí 5 řádků. Opomenutí vynechaných řádků nemá na výsledný diagram pro tento příklad vliv, a proto je možné psát kratší kódy pro diagramy se základním typem klasifikátoru.

Kód 3: Popis diagramu pro PlantUML

```
1. abstract Bytost
2. Bytost : +jmeno: String
3. Bytost : +vek: int
4. Bytost : +promluv()
5. Elf : -obratnost: int
6. Elf : +lukostrelba()
7. Trpaslik : +sila: int
8. Hobit : +maDymku: Boolean
9. Hobit : +hostit()
10. Hobit : -schovatSe()
11. Bytost <|-- Elf
12. Bytost <|-- Trpaslik
13. Bytost <|-- Hobit
```

Na základě uvedených ukázek je patrné, že nejkratší zápis poskytuje jazyk nomnoml. Avšak jeho kompaktnost je vykoupena nižší přehledností kódu. Z hlediska minimálního počtu řádků dosahují jazyky Mermaid a PlantUML podobných hodnot.

Celkové výsledky srovnání vybraných jazyků ukazují, že nejefektivnější nástroj nedisponuje dostatečně srozumitelným a výřečným jazykem. Z pohledu potřeby v jazyce zadefinovat základní prvky UML diagramu tříd vychází Mermaid a PlantUML rovnoměrně. PlantUML navíc nabízí i možnost generování asociačních tříd, což je notace pro tzv. syntaktický cukr, který lze vyjádřit i prostřednictvím jiných konstrukcí jazyka UML.

Jako nejvhodnější základ pro realizace požadované aplikace se jeví PlantUML, neboť zachovává čitelnost kódu, podporuje všechny základní prvky diagramu tříd a jeho doplňková funkcionalita poskytuje prostředky pro realizaci požadovaných rozšiřujících funkcionalit. Těmito mechanismy, které činí z PlantUML vysoce flexibilní nástroj, jsou skrývání objektů, možnosti manuálního ovlivnění finálního rozvržení grafu a přizpůsobení barev elementů a textů.

5 Návrh jazyka

Cílem této kapitoly je zadefinovat syntaktické konstrukce navrženého jazyka. Požadavky na jazyk vycházejí z funkčních požadavků, které byly specifikovány v podkapitole 4.1. Základní syntaxe navrženého jazyka vychází z PlantUML, avšak pro potřeby cílové aplikace je jazyk modifikován. Nejprve jsou rozebrány nejdůležitější modifikace jazyka, z kterého návrh vychází. Následuje formální definice jazyka pomocí EBNF, což je rozšířená varianta bezkontextové gramatiky BNF (Chu, C2001-2025). Na závěr jsou ukázány vstupy pro speciální případy s očekávanými výstupy.

5.1 Hlavní modifikace PlantUML

V této podkapitole jsou představeny:

- nové konstrukce rozšiřující jazyk,
- původní konstrukce, které je možné v jazyce ponechat,
- potřebné modifikace některých definic.

Rozšiřující jazykové konstrukce

Nové syntaktické prostředky slouží pro podporu rozšířené funkcionality cílového programu, která je zaměřena na postupný vývoj diagramu tříd s podporou animace přes více snímků a na nápovědu pro preferované umístění prvků.

Platnost konkrétních vlastností prvku lze vymezit prostřednictvím definice prvku o daných vlastnostech a přidáním anotace s klíčovým slovem `slide`. První, povinný argument určuje počáteční snímek, od něhož definované vlastnosti vstupují v účinnost. Druhý argument je volitelný a specifikuje okamžik ukončení této platnosti. Pokud anotace chybí, implicitně se uvažuje interval od nultého snímku do nekonečna. Použití výrazu `@slide(0)` má tedy ekvivalentní význam jako opomenutí této specifikace.

Zápis preferované pozice prvku je navržen anotací s klíčovým slovem `hint`. Argumentem je jeden z předem definovaných názvů pozice. Anotaci je možné použít i pro vztah, kdy má stejný význam jako původní specifikace směru vztahu z PlantUML. Pro směr vztahu jsou vhodné pouze základní směry, a proto jsou možnosti pro argument specifikace směru propojení omezenější.

Zachované možnosti

Současně se zavedením nových pravidel lze v modifikovaném jazyce zachovat určité možnosti poskytované nástrojem PlantUML, které nejsou součástí požadavků výsledné aplikace. Konkrétně se jedná o:

- odkazy – přidání hypertextového odkazu k UML prvku a informačního popisku (*tooltip*),
- `extends` a `implements` – zjednodušená syntaxe pro generalizaci a realizaci v definici klasifikátoru,
- kvalifikátory – rozšiřující mechanismus pro upřesnění významu prvku ve vztahu.
- kódové označení a zobrazovaný text – prvky, které mají v grafové reprezentaci význam vrcholů, jsou v textovém popisu identifikovány pomocí kódového názvu, ale jejich zobrazovaný název může být odlišný.

Ostatní nadstandardní funkce, jako je například ruční volba barvy prvků, nebudou v navrhovaném jazyce podporovány, neboť jsou v cílovém programu využívány pro automatické zvýrazňování a není žádoucí, aby do tohoto procesu uživatel zasahoval.

Modifikace vycházející z původního jazyka

Pro lepší kontrolu editace změn atributů je zavést přísnější syntaxi pro definici atributů a operací tříd. PlantUML totiž umožňuje vložit do jejich definice libovolný text, což není pro účely této práce vhodné. Přísnější formát vychází z notace specifikované ve standardu UML, čímž se podpoří konzistence a možnost automatické kontroly změn členů tříd. Použití nového zápisu nerozporuje pravidlům původního.

5.2 Formální definice jazyka

Kód 4: Definice gramatiky

```
diagram             ::= [element {new-line element}]

element             ::= classifier | namespace | relationship | note | member

classifier          ::= classifier-base ["{" new-line classifier-body
new-line "}" ]
classifier-body     ::= [in-member {new-line in-member}]
classifier-base     ::= [visibility] classifier-type element-name [generic]
[stereotype] [url] [extends] [implements] [slide] [hint]

namespace           ::= namespace-base ["{" new-line namespace-body new-line
"}"]
namespace-body      ::= [diagram]
namespace-base      ::= namespace-type element-name [stereotype] [url] [slide]
          [hint]

relationship        ::= ( basic-relationship | association-class ) [url]
          [slide] [hint-link] [caption]
basic-relationship ::= code relationship-base code
association-class   ::= link-code relationship-base code | code
          relationship-base link-code
relationship-base   ::= [qualifier] ['"' multiplicity '"'] [header-left] body
          [header-right] ['"' multiplicity '"'] [qualifier]

note                ::= "note" code-display [slide] [hint]

member              ::= code ":" ( attribute | method )
attribute           ::= [visibility] member-name [":" member-type ["["
          multiplicity "]"]] ["=" value] ["{" property-string "}"] [slide]
method              ::= [visibility] member-name "(" [parameters] ")" [":"
          value] ["{" property-string "}"] [slide]

classifier-type     ::= "class" | "interface" | "abstract" | "enum" |
          "stereotype"
namespace-type      ::= "namespace" | "package"

codes               ::= code {"," code }
link-code           ::= "(" code "," code ")"
element-name        ::= code | code-display
code-display        ::= code "as" display | display "as" code

slide               ::= "@slide(" number ["," number] ")"
hint                ::= "@hint(" hint-extended ")"
hint-link           ::= "@hint(" hint-base ")"
url                 ::= ("[[" hyperlink "," [tooltip] "]]" | "[[" tooltip
          "]]")
extends             ::= "extends" codes
implements          ::= "implements" codes
```

```
parameters          ::= parameter {"," parameter}
multiplicity        ::= number | number ".." number | [number ".."] "*"
parameter           ::= member-name [":" member-type] ["=" value]
header-left         ::= "^"| "<|"| "<" | "*" | "o"
header-right        ::= "^" | "|>" | ">"| "*"| "o"
body                ::= "-" | "." |"--" | ".."
visibility          ::= "+" | "-" | "#" | "~"
hint-base           ::= "up" | "down" | "right" | "left"
hint-extended       ::= hint-base | "center" | "up-right" | "up-left" |
          "down-right" | "down-left"
hyperlink           ::= alpha-numeric { alpha-numeric | "/" | "." }

code                ::= alpha-numeric
member-name         ::= alpha-numeric
member-type         ::= alpha-numeric
value               ::= alpha-numeric
property-string     ::= alpha-numeric
qualifier           ::= alpha-numeric
generic             ::= "<" alpha-numeric ">"
stereotype          ::= "<<" alpha-numeric ">>"
display             ::= '"' alpha-numeric-ws '"'
tooltip             ::= "{" alpha-numeric-ws "}"
caption             ::= ":" alpha-numeric-ws

alpha-numeric-ws    ::= ( "A"…"Z" | "a"…"z" | "0"…"9" ) { "A"…"Z" | "a"…"z" |
          "0"…"9" | " " }
alpha-numeric       ::= ( "A"…"Z" | "a"…"z" | "0"…"9" ) { "A"…"Z" | "a"…"z" |
          "0"…"9" }
number              ::= "0"…"9" {"0"…"9"}
new-line            ::= "\n" | "\r\n"
```

Gramatika je rozdělena do několika logických skupin:

- prvky struktury diagramu:
    - `diagram` je startovní neterminál,
    - `element` představuje jednotlivé prvky diagramu,
- vnořený obsah prvků:
    - `classifier-body` a `namespace-body` popisují obsah tříd a jmenných prostorů,
- vlastnosti prvků:
    - základní povinné vlastnosti určují například název nebo typ prvku,

    - volitelné vlastnosti jako například `caption`, `multiplicity`, `qualifier`, `header-left`, `header-right`, `member-type`, `visibility` umožňují přesnější popis modelu,
    - doplňková pravidla `slide` a `hint` slouží k rozšíření funkcionality o přidání animací a informace o preferovaném umístění prvků,
- lexikální prvky:
    - `alpha-numeric`, `alpha-numeric-ws`, `number` a `new-line` definují základní stavební bloky jazyka,

Je třeba poznamenat, že jazyk generovaný touto gramatikou není plně ekvivalentní s jazykem, který je akceptován cílovým programem. Zde uvedená definice se zaměřuje pouze na nejdůležitější syntaktické konstrukce. V gramatice záměrně chybí specifikace umístění nepovinných bílých znaků, aby byla gramatika přehlednější a čitelnější. Neterminál zastupující nepovinné bílé znaky by mohl být na pravé straně pravidel:

- mezi jednotlivými neterminály,
- mezi neterminálem a terminálem s výjimkou pravidel pro `generic` a `stereotype`.

Dále jsou z formální gramatiky vynechány komplexnější varianty zápisu, protože by také zhoršily srozumitelnost gramatiky. Jedná se například o zápis ignorovaných komentářů pro interní potřeby uživatele, nebo použití vnořených generických typů.

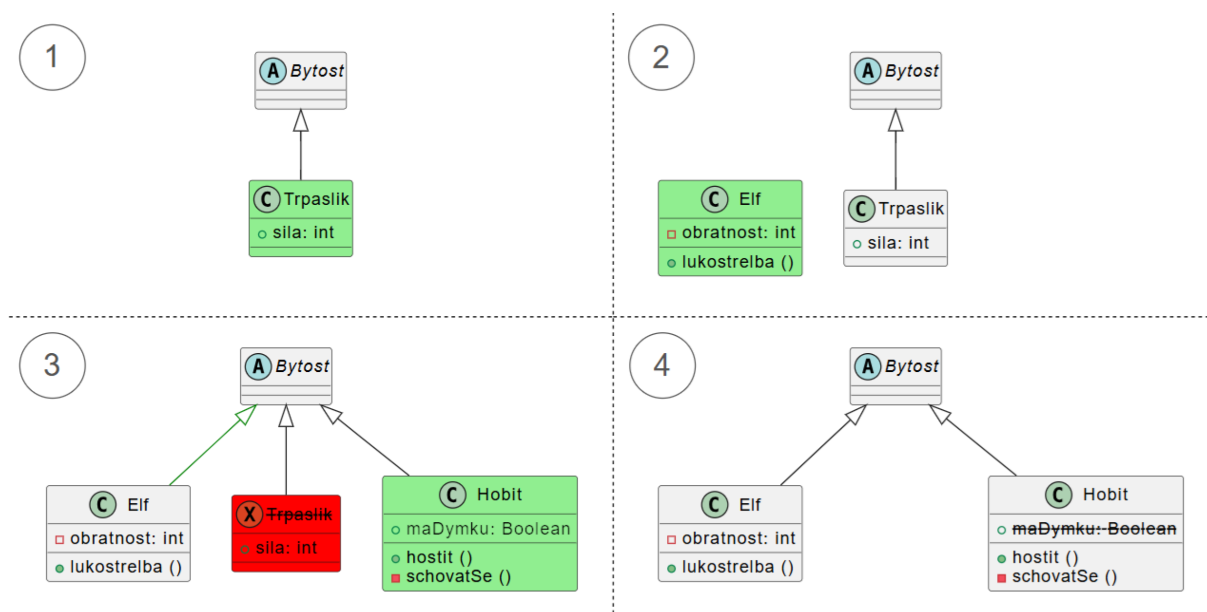
5.3 Ukázka vstupu a výstupu

Cílem této podkapitoly je názorně demonstrovat, jakým způsobem mají nově zavedené konstrukce navrženého jazyka ovlivňovat výsledek generování. Ukázky slouží jako ilustrace semantiky použitých syntaktických prvků.

V rámci první ukázky je představena funkcionalita podpory rozvoje diagramu, kdy výstupem bude několik diagramů s drobnými změnami. Vstupem je Kód 5 a očekávaný výstup reprezentuje Obrázek 32, ve kterém jsou ilustrovány i automatická zvýraznění změn.

Kód 5: Ukázka vstupu pro demonstraci zakládání a rušení prvků

```
abstract Bytost
Bytost : +jmeno: String
Bytost : +vek: int @slide(0,1)
Bytost : +promluv()
class Elf @slide(2)
Elf : -obratnost: int
Elf : +lukostrelba()
class Trpaslik @slide(1,2)
Trpaslik : +sila: int
class Hobit @slide(3)
Hobit : +maDymku: Boolean @slide(3,3)
Hobit : +hostit()
Hobit : -schovatSe()
Bytost <|-- Elf @slide(3)
Bytost <|-- Trpaslik
Bytost <|-- Hobit
```

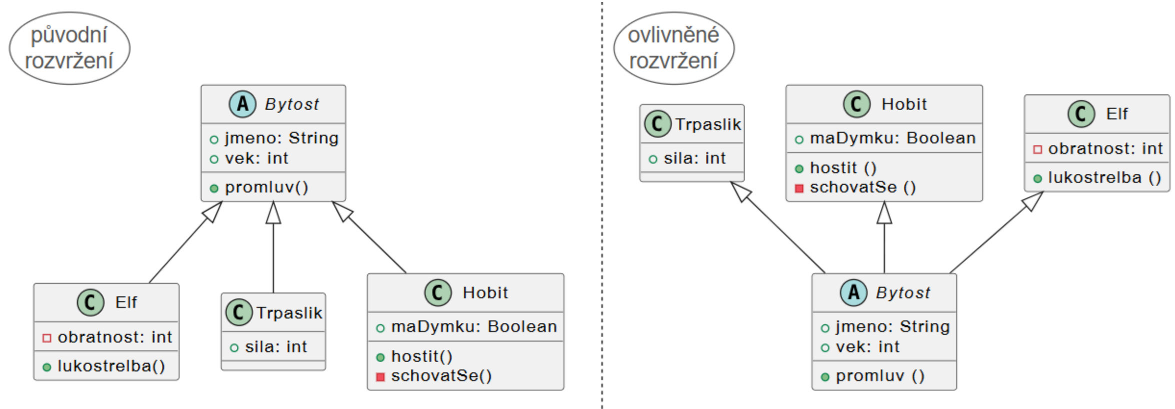


Obrázek 32: Očekávaný výstup pro zakládání a rušení prvků, zdroj: vlastní

Druhá demonstrace ilustruje vliv anotace pro preferované umístění prvků na výsledné rozvržení diagramu. Vstupní Kód 6 obsahuje specifikaci pozic vybraných prvků prostřednictvím této anotace. Na Obrázek 33 jsou verze generovaného diagramu bez uplatnění preferencí a s uplatněním, kde je patrná změna uspořádání podle preferencí.

Kód 6: Ukázka vstupu pro demonstraci preferencí pro rozvržení prvků

```
abstract Bytost @hint(down)
Bytost : +jmeno: String
Bytost : +vek: int
Bytost : +promluv()
class Elf @hint(right)
Elf : -obratnost: int
Elf : +lukostrelba()
class Trpaslik @hint(left)
Trpaslik : +sila: int
class Hobit
Hobit : +maDymku: Boolean
Hobit : +hostit()
Hobit : -schovatSe()
Bytost <|-- Elf
Bytost <|-- Trpaslik
Bytost <|-- Hobit
```



Obrázek 33: Porovnání výstupu bez hintů a očekávaného výstupu s hinty, zdroj: vlastní

6 Návrh a implementace

Cílem této kapitoly je představit postup realizace programu pro generování diagramů tříd, který přijímá vstup v podobě jazyka navrženého v kapitole 5. Tato kapitola shrnuje klíčové části od implementace až po spuštění aplikace. Nakonec jsou rozebrána další možná rozšíření aplikace.

6.1 Návrh aplikace

Návrh cílového programu vychází z požadavků, které byly specifikovány v kapitole 4. Součástí návrhu aplikace je popis základní architektury programu a formulace postupů, které umožňují implementaci rozšířených funkcionalit.

Architektura programu

Navržená architektura programu je rozdělena na čtyři hlavní části:

- syntaktický analyzátor,
- model,
- aplikátor nápovědy pro pozice,
- generátor.

Tyto komponenty umožňují převod textového vstupu do grafické podoby UML diagramů tříd s podporou rozšířené funkcionality.

Vstupní textový popis diagramu vytvořený v navrženém jazyce je zpracován syntaktickým analyzátorem, který provádí analýzu po jednotlivých řádcích. Pro rozpoznání konstrukcí jazyka využívá regulární výrazy, pomocí kterých je vystavěna interní reprezentace dat. Výstupem parseru je model, který představuje centrální datovou strukturu, z níž lze odvodit všechny požadované vizualizace.

Datový model obsahuje nejen podklady pro generování sekvence výstupních diagramů, ale i specifikaci priorit a směru propojení prvků, z které se vytváří cílové rozvržení. Tyto informace jsou zpracovány prostřednictvím aplikátoru nápovědy, jehož úlohou je modifikovat priority a směrování vztahů tak, aby byly preferované pozice zohledněny ve výsledném uspořádání. Aplikátor lze koncipovat v různých variantách, odlišujících přesností a složitostí. V rámci této práce jsou rozlišovány dvě úrovně vlivu na rozvržení: lokální, omezující se na bezprostředně propojené vrcholy, a globální, která navíc reflektuje i vzájemné vztahy mezi oddělenými podgrafy.

Generátor iterativně zpracovává informace z modelu. Na základě aktuální platnosti prvků v jednotlivých iteracích jsou vytvářeny odpovídající diagramy. Generování se skládá z tvorby vstupního popisu pro PlantUML a z volání této knihovny pro uložení vytvořeného diagramu. Každý takto vygenerovaný diagram je uložen ve formátu SVG na předem definovaném umístění. Současně je výslednému souboru přiřazen název s předem stanoveným prefixem a sufixem podle čísla snímku, který odráží pozici diagramu v celé sekvenci.

Princip podpory animace

Pro zajištění generování více výstupů, které zachycují postupný vývoj diagramu tříd, je nezbytné zavést následující mechanismy:

- udržování informací o prvcích v čase,
- zajištění konzistentního rozvržení prvků napříč všemi snímky.

V rámci udržování potřebných informací a tvorbě datového modelu musí mít každý prvek diagramu jednoznačnou identifikaci, která vychází z jeho neměnných vlastností. Vedle toho existují editovatelné vlastnosti, které se mohou v čase měnit. Do této kategorie spadají např. viditelnost atributů, zobrazený název třídy, popisek asociace či stereotypy. Je možné poznamenat, že tyto vlastnosti mohou být povinné i nepovinné, neboť třída musí mít nějaký zobrazovaný název, ale asociace nemusí mít nutně popisek. Editovatelné vlastnosti jsou uchovávány ve formě parametrů s rozsahem snímkové platnosti. Díky identifikaci prvků lze sledovat změny proměnlivých vlastností. Princip uchovávání informací tedy spočívá ve vytvoření modelu, v němž je každý prvek reprezentován jednoznačnou identifikace a má seznam parametrů v jednotlivých fázích vývoje diagramu.

Ustáleného rozvržení prvků napříč generovanými snímky lze dosáhnout tak, že algoritmus pro výpočet rozvržení pracuje se všemi prvky diagramu současně. Ty prvky, které pro daný snímek nemají platné parametry, jsou vygenerovány z výsledné vizualizace schovány, aby se v konečném výstupu nezobrazily.

Princip ovlivnění pozice prvků

Nejpřímějším řešením pro určení pozice prvků v automatickém rozvržení je použít constraint-based engine. Například knihovna ELK podporuje specifikaci prostorových omezení, ale jelikož se jedná o pouze předběžnou verzi a aktuálně není v PlantUML funkční, není to pro účely této práce vyhovující. Jako vhodnější řešení bylo vyhodnoceno využít defaultního enginu Graphviz dot a jeho známého chování, které umožňuje ovlivnit rozvržení pozicí prvků

v definicích. Graphviz dot implementuje algoritmus pro hierarchické uspořádání grafu, v němž jsou vrcholy rozmísťovány do vrstev podle jejich hierarchické důležitosti.

Existují návody, jak dosáhnout cíleně ovlivnit výsledné rozvržení. V této práci jsou dané postupy využity pro algoritmickou podporu preferovaných pozic. Zahrnují přeskupení řádků, prvků ve vztazích a zavedení skrytých propojení.

Pozice jednotlivých prvků je výsledkem kombinace několika mechanismu. Základem je pořadí, v jakém jsou prvky a jejich vztahy deklarovány ve vstupním kódu. Obecně platí, že prvky uvedené dříve mají tendenci být umístěny výše a více vlevo. Dále je kladen důraz na dodržení prostorového typu vztahů. Pro horizontální vztahy se první uvedený prvek zobrazuje více vlevo, zatímco u vztahů vertikálních bývá první prvek zobrazen výše.

Problémem může být existence nedosažitelných vrcholů v rámci neorientovaného grafu. V případě, že pro takové prvky existuje požadavek na jejich preferovanou pozici a je požadováno udržovat sestavení diagramu podle preferencí na globální úrovni, je nutné doplnit neviditelné vazby. Tyto hrany, které nejsou ve výsledném diagramu vykresleny, umožňují vyjádřit potřebu vzájemné pozice těchto nepropojených struktur. Algoritmus pro rozvržení pak může zajistit jejich relativní uspořádání v souladu s požadavky. Avšak pro globální případ je potřeba podrobnější zmapování preferencí prvků v nepropojených podgrafech, případné další specifikace na úrovni těchto podgrafů. Tento přístup komplikuje výpočetní složitost, a proto je v základním návrhu předpokládaná pouze lokální úroveň pro specifikaci preferencí, která platí pro pozici prvků vůči ostatním dosažitelným vrcholům.

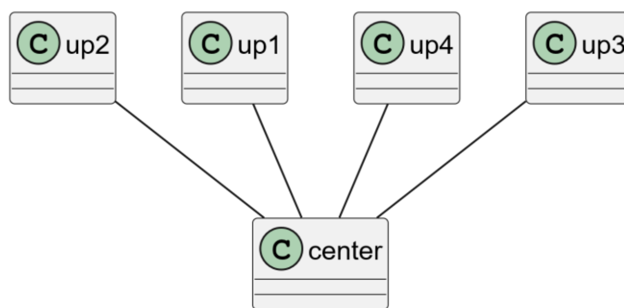
Obecně platí, že pokud mají být dva prvky umístěny v horní části diagramu a zároveň propojeny, je vhodné mezi nimi zavést horizontální propojení. Tím se snižuje riziko, že automatické rozvržení posune některý z prvků do nižší vrstvy hierarchie. Podobně to platí pro prvky, jejichž preferovaná poloha je v dolní části diagramu, či pro prvky umístěné na levé či pravé straně. Tato pravidla jsou shrnuta v Tabulka 4, která ukazuje závislost mezi základní preferovanou pozicí prvního a druhého prvku a odpovídajícím zápisem ve vstupním kódu. Pro přehlednost je výsledný zápis demonstrován na jednoduchých asociacích, jelikož u ostatních typů vztahů se uplatňují analogické postupy.

Mezi preferovanými pozicemi existuje určitá hierarchie, která má přímý vliv na výsledné rozmístění prvků v diagramu. Nejvyšší prioritu má poloha `up`, následují polohy `left` a `right`, a teprve poté poloha `down`. Tato hierarchie je patrná také z Tabulka 4, kde je možné pozorovat vliv této priority na výsledný zápis vztahu.

Tabulka 4: Pravidla pro formulaci vztahu podle preferovaných pozic, zdroj: vlastní

Pozice A	Pozice B	Výsledný zápis	Komentář
up	up	A - B	Horizontální propojení
down	down	A - B	Horizontální propojení
left	left	A -- B	Vertikální propojení
right	right	A -- B	Vertikální propojení
up	down;left;right;none	A -- B	Vertikální propojení, A má v hierarchii přednost
down	up;left;right;none	B -- A	Vertikální propojení, B má v hierarchii přednost
left	right;none	A - B	Horizontální propojení, A má v hierarchii přednost
right	left;none	B - A	Horizontální propojení, B má v hierarchii přednost

Uvedené pozice v Tabulka 4 jsou základními a jednoduššími pro realizaci. Komplikovanějším případem je požadavek pro umístění prvku uprostřed. Pro splnění preferované centrální pozice, je nezbytné, aby v bezprostředním okolí daného prvku existoval dostatek vrcholů, jejichž preferované pozice nejsou v rozporu s centrální pozicí. V případě konfliktu umístění prvku doprostřed s některou ze základních jednoduchých preferencí má základní pozice přednost, a proto výsledné uspořádání nebude odpovídat požadované centrální pozici. Příkladem konfliktu může být propojení centrálního prvku pouze se čtyřmi prvky stejné základní preference, což vyobrazuje Obrázek 34. Pokud jsou však podmínky splněny, algoritmus vyhodnotí okolní propojené prvky a zajistí, aby tyto prvky obklopovaly centrální prvek.



Obrázek 34: Ukázka konfliktních preferencí pro centrální prvek, zdroj: vlastní

Uvedené mechanismy ukazují, že možnost ovlivňovat automatické rozvržení v PlantUML je nepřímá a je založena na cílené manuální manipulaci se vstupním textovým formátem. Rozšíření o nápovědu v navrhovaném jazyce proto představuje praktické zjednodušení. Uživateli je umožněno deklarovat preferovanou pozici prvků přímo, aniž by byl nucen využívat nepřímé techniky a složitě přeuspořádávat zdrojový kód. Pokud se později rozhodne rozvržení změnit, postačí pouze úprava příslušné anotace, nikoli kompletní přepracování vstupního souboru. Nápovědy jsou tedy významným zjednodušením zápisu, který vhodně využívá interních mechanismů.

Pro vztahy mezi prvky je klíčové slovo `hint` pouze jiným značením původního označení směru vykresleného propojení z PlantUML, které se píše v těle syntaxe pro propojení ve hranatých závorkách.

6.2 Implementace aplikace

Následující podkapitola se zaměřuje na popis realizace navrženého řešení formou konzolové aplikace, která je implementovaná v programovacím jazyce Java. Aplikace využívá knihovnu PlantUML pro generování výsledných diagramů. Pozornost je věnována především nejdůležitějším třídám hlavních částí systému – datovému modelu, sytaktickému analyzátoru, aplikátoru nápovědy a generátoru. Důraz je kladen na popis tříd tvořících základní stavební bloky jednotlivých modulů a na objasnění způsobu, jakým jsou implementovány algoritmy zajišťující požadovanou funkcionalitu. Na závěr jsou představeny doplňkové úpravy, které rozšiřují původní návrh o praktické funkce, jako je efektivnější a poholnější generování snímků nebo možnost využívat neviditelné hrany.

Datový model

Datový model představuje základní vrstvu implementace a jeho třídy se nachází v balíčku `model`, s výjimkou třídy `PlantUmlSlidesSource` umístěné v balíčku `slide`. Vytvořením vnitřní reprezentace domény umožňuje oddělit logiku zpracování od vstupních formátů, sjednocuje práci s daty napříč moduly a podporuje uchovávání změn v čase. Tím poskytuje stabilní základ pro další klíčové části systému.

Na nejvyšší úrovni datového modelu stojí třída `PlantUmlSlidesSource`, která uchovává celý analyzovaný diagram. Jednotlivé prvky jsou reprezentovány instancemi třídy ElementInfo. Každý takový prvek je určen jednoznačnou identifikací (`IElementIdentification`) a má přiřazen seznam parametrů (`IElementParams`). Identifikace zachycuje neměnné vlastnosti prvku, zatímco parametry vyjadřují jeho proměnlivé charakteristiky v čase. Konkrétní třídy pro identifikaci a parametry tvoří nejnižší úroveň, do níž jsou prvky diagramu dekomponovány.

Rozhraní `IElementIdentification` určuje jednoznačnou identifikaci každého prvku a současně uchovává informace o jeho preferované pozici a prioritě, což umožňuje ovlivňovat rozmístění prvků. Identifikace implementuje metody `equals` a `hash`, aby bylo možné je efektivně využít jako klíč ve vyhledávacích strukturách.

Rozhraní `IElementParams` definuje rozsah platnosti parametrů v podobě třídy `SlideInterval`. V rámci tohoto rozsahu se udržuje informace nejen o začátku a konci platnosti konkrétních vlastností, ale také o tom, zda byl konec platnosti specifikován, nebo může být později přepsán.

Konkrétní typy prvků, které datový model podporuje, jsou:

- klasifikátory,
- jmenné prostory,
- poznámky,
- vztahy,
- členy klasifikátorů:
    - atributy,
    - operace,
    - nespecifikované členy, pro které není možné řádně udržovat historii editací.

Každý z těchto typů disponuje vlastní implementací rozhraní `IElementIdentification` a `IElementParams`.

Syntaktický analyzátor

Syntaktická analýza vstupu je implementována v balíčku `parser`. Jejím cílem je převést textový vstupní soubor do vnitřní reprezentace diagramu, s níž dále pracují ostatní části systému. Hlavní třídou je `SlideComponentsParser`, která prostřednictvím metody `parse` postupně zpracovává jednotlivé řádky textu, získává instance tříd datového modelu a udržuje jejich kolekci pro vytvoření cílové struktury diagramu `PlantUmlSlidesSource`.

Pro každý z podporovaných elementů je implementován samostatný analyzátor, jenž kontroluje shodu s definovanými vzory regulárních výrazů pro daný typ. Základní konstrukce pro sestavení regulárních výrazů jsou dostupné ve statické třídě RegexGroups, aby bylo možné jejich znovupoužití. Při úspěšném rozpoznání řádkové definice je vytvořen odpovídající `IElementParams` a případně i `ElementInfo`. Podle posloupnosti prvotních definic konkrétních instancí prvků je přiřazena priorita pro automatické rozvržení.

Aplikátor nápovědy pro pozici prvků

Po dokončeném načtení vstupu se ještě v metodě `parse` volá aplikátor nápovědy pro rozmístění prvků. Třídy pro tuto funkcionalitu jsou v balíčku `hints`. Aplikátor implementuje rozhraní `IHintApplier`, které je realizováno buď v lokální, nebo globální variantě. Ve výchozím nastavení je využíván `LocalHintApplier`.

Rozhraní aplikátoru deklaruje metodu `applyPositionInfo`, jejíž činností je úprava priority prvků, jejich pořadí ve vztazích a prostorové směřování hran. Globální varianta algoritmu navíc do diagramu přidává neviditelná propojení, která ovlivňují relativní polohu nepropojených podgrafů. Celkově jde o proceduru, která modifikuje kolekci prvků před sestavením objektu `PlantUmlSlidesSource`.

Generátor diagramů

Balíček `generator` obsahuje implementaci generování diagramů, jejíž ústřední třídou je `DiagramGenerator`. Inicializaciování této třídy vyžaduje `PlantUmlSlidesSource`, cestu k výstupní složce a základní řetězec pro názvy výstupních souborů. Metoda `generateAll` zajišťuje postupnou tvorbu diagramů pro všechny použité snímky. Hlavní fáze generování diagramu jsou konstrukce textového popisu v jazyce PlantUML a následné volání PlantUML pro vytvoření výstupní grafiky ve formátu SVG.

Sestavení textového vstupu pro PlantUML probíhá ve třídě `ComponentDefinitionGenerator`. Formulace syntaktických konstrukcí, které zajišťují zvýraznění změn mezi jednotlivými verzemi diagramu, je implementována ve třídě `HighlightUtils`.

Během generování je nutné určit stav každého parametru prvku na základě porovnání s parametry téhož prvku z předchozího snímku. Základní stavy, jejich typ ve zdrojovém kódu a jejich příslušná forma reprezentace shrnuje Tabulka 5.

Tabulka 5: Souhrn typu vykreslení pro konkrétní stav parametrů aktuálního snímku, zdroj: vlastní

Stav parametrů	**Výčtový typ (`ParamsState`)**	**Forma**
Nezměněné	`UNCHANGED`	Normální vykreslení
Nové	`NEW`	Zvýrazněná (zelená)
Smazané	`DELETED`	Zvýrazněná (červená, případně přeškrtnutí)
Editované	`EDITED`	Zvýrazněná (žlutá)
Chybí	*Není ve výčtu*	Schovaný prvek – pro členy stačí nezahrnout do popisu, pro ostatní je třeba definovat a schovat pomocí `hide`

K volání PlantUML se využívá třída `SourceStringReader` a její metoda `outputImage`, která zprostředkuje uložení výsledného obrázku v operačním systému.

Praktické úpravy

Během vývoje byly do programu doplněna vylepšení, která přispívají k efektivnějšímu zpracování a k praktičtějšímu využití nástroje.

První z těchto úprav představuje zavedení kurzoru nad seznamem parametrů prvků. Kurzorem lze pro zadané číslo snímku efektivně vybrat právě aktuální parametry, aniž by bylo nutné opakovaně procházet všechny záznamy parametrů.

Další úprava se týká evidence čísel použitých a potřebných snímků. Aplikace vynechává generování nadbytečných snímků. Nadbytečné jsou po sobě jdoucí snímky, mezi nimiž nedošlo k žádné změně. Výsledná sekvence tak obsahuje pouze relevantní stavy diagramu, což zlepšuje přehlednost výstupu i efektivitu generování.

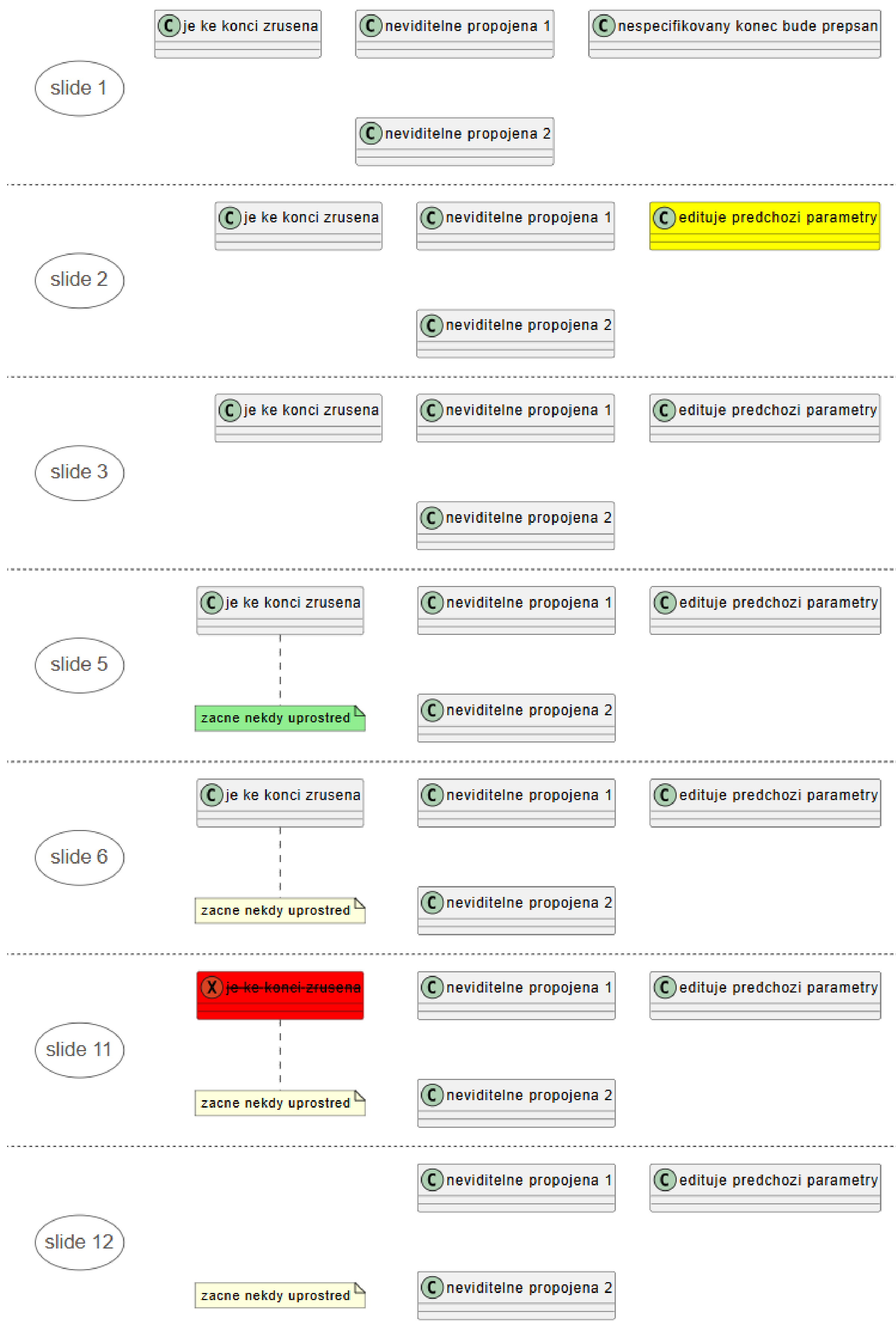
S předchozí úpravou souvisí i možnost definovat parametry s otevřeným intervalem platnosti. Pokud je určeno pouze číslo počátečního snímku, konec jeho platnosti není určen a odvíjí se od případných následujících parametrů daného prvku. Tento mechanismus je v kódu označován jako redukce snímků a slouží k tomu, aby se průběžně přepisovaly neupřesněné konce a odstraňovaly se parametry, které by nikdy neplatí.

Za doplňkovou úpravu je možné považovat i povolení definici neviditelných propojení, což je mechanismus PlantUML pro ovlivnění výsledného rozmístění prvků. V realizované aplikaci není zachována původní podoba syntaxe, ale používá se zápis `@slide(0,0)`, který je vhodnější vzhledem k existenci této konstrukce v gramatice přijímaného jazyka.

Při kombinaci použití nenavazujících čísel snímků a nespecifikování konce platnosti, může uživatel vytvářet vývoj diagramu bez konkrétní znalosti konečného počtu výstupů. Příklad využití tohoto chování programu a použití neviditelných propojení zobrazuje Obrázek 35, který reprezentuje výstup pro vstupní popis z Kód 7.

Kód 7: Popis diagramu s využitím doplňkových funkcionalit

```
note "zacne nekdy uprostred" as note1 @slide(5)
class trida1 as "je ke konci zrusena" @slide(0,10)
class "neviditelne propojena 1" as trida2
class "nespecifikovany konec bude prepsan" as trida3 @slide(0)
class "edituje predchozi parametry" as trida3 @slide(2)
class "neviditelne propojena 2" as trida4
trida2 - trida4 @slide(0,0)
```



Obrázek 35: Ukázka využití doplňkových funkcionalit, zdroj: vlastní

Poslední ukázka, která je složena z Kód 8 a Obrázek 36, znázorňuje možné využití aplikace, způsoby vykreslení a zvýraznění jednotlivých prvků a kombinaci rozšířené funkcionality v podobě postupného rozvoje a preferovaných pozic.

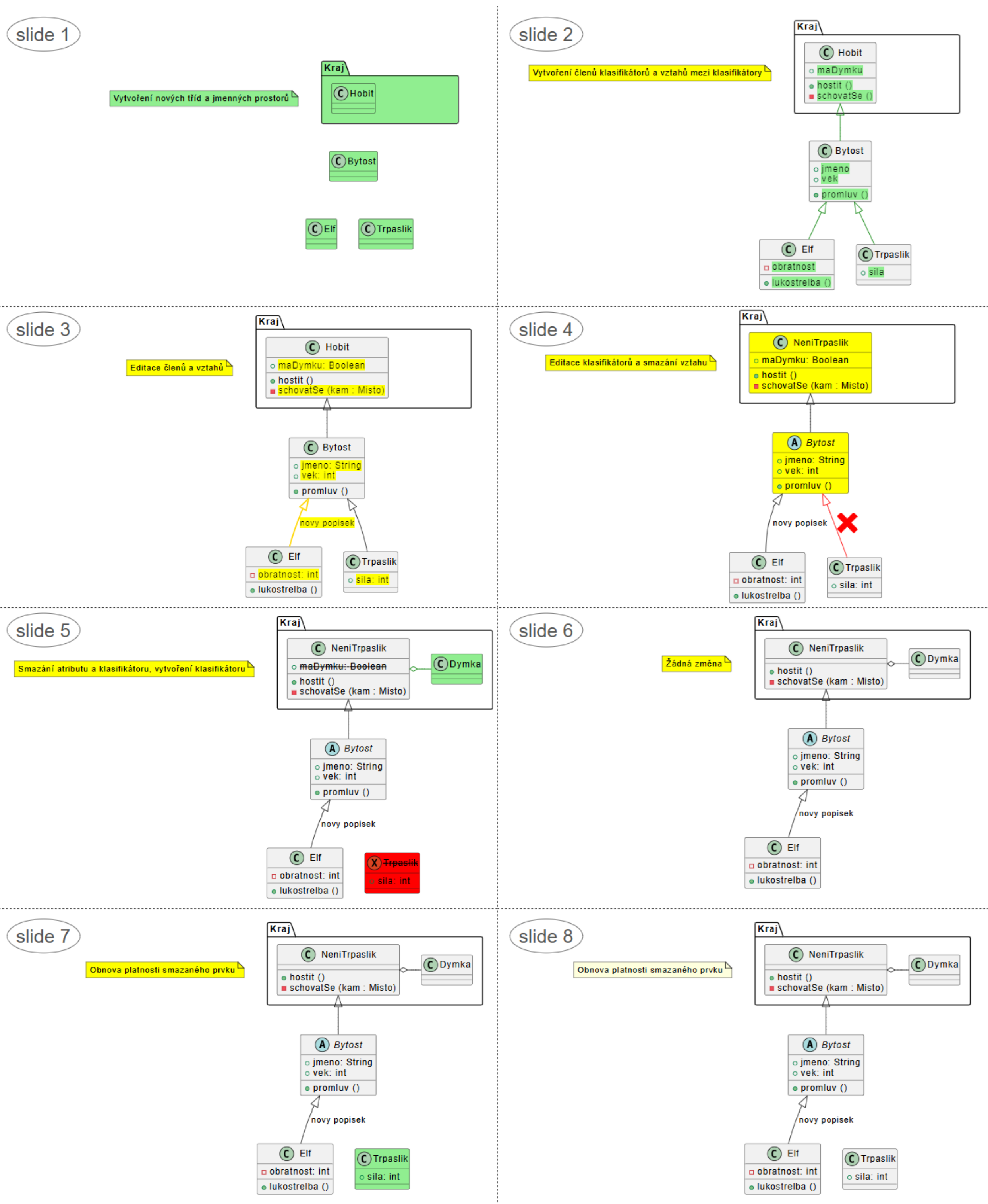
Kód 8: Ukázka použití aplikace

```
namespace Kraj @slide(1) {
class Hobit @slide(1)
class Hobit as "NeniTrpaslik" @slide(4)
Hobit : +maDymku @slide(2)
Hobit : +maDymku: Boolean @slide(3,4)
Hobit : +hostit() @slide(2)
Hobit : -schovatSe() @slide(2)
Hobit : -schovatSe(kam : Misto) @slide(3)
class Dymka @slide(5)
Hobit o- Dymka @slide(5)
}

class Elf @slide(1)
Elf : -obratnost @slide(2)
Elf : -obratnost: int @slide(3)
Elf : +lukostrelba() @slide(2)
class Trpaslik @slide(1,4)
class Trpaslik @slide(7)
Trpaslik : +sila @slide(2)
Trpaslik : +sila: int @slide(3)
class Bytost @slide(1)
abstract Bytost @slide(4) @hint(up)
Bytost : +jmeno @slide(2)
Bytost : +jmeno: String @slide(3)
Bytost : +vek @slide(2)
Bytost : +vek: int @slide(3)
Bytost : +promluv() @slide(2)

Bytost <|-- Elf @slide(2)
Bytost <|-- Elf @slide(3) : novy popisek
Bytost <|-- Trpaslik @slide(2,3)
Bytost <|-- Kraj.Hobit @slide(2)

note n as "Vytvoření nových tříd a jmenných prostorů" @slide(1) @hint(up-
left)
note "Vytvoření členů klasifikátorů a vztahů mezi klasifikátory" as n
@slide(2)
note "Editace členů a vztahů" as n @slide(3)
note "Editace klasifikátorů a smazání vztahu" as n @slide(4)
note "Smazání atributu a klasifikátoru, vytvoření klasifikátoru" as n
@slide(5)
note "Žádná změna" as n @slide(6)
note "Obnova platnosti smazaného prvku" as n @slide(7)
```



Obrázek 36: Ukázka použití aplikace, zdroj: vlastní

6.3 Spuštění aplikace

K běhu aplikace je nutné mít nainstalován program Graphviz, na kterém je závislé automatické rozvržení grafu. Další prerekvizitou je JDK ve verzi 21, která byla použita pro vývoj aplikace.

Program ve formě spustitelného archivu SlideParsingGenerator.jar je dostupný v přílohách práce. Spouští se z příkazové řádky v adresáři, který tento archiv obsahuje, příkazem:

```
java -jar ./SlideParsingGenerator.jar [<předvolby>] <vstup> <výstup>.
```

Parametr <vstup> určuje cestu k textovému souboru s definicí diagramu. Parametr <výstup> specifikuje cestu, která určuje výstupní adresář a prefix názvu pro vygenerované soubory. Pokud adresář neexistuje, aplikace jej vytvoří. K zadanému základnímu názvu je při ukládání doplněno příslušné číslo snímku pro vygenerovanou verzi diagramu.

Nepovinná část <předvolby> je určena pro pokročilejší nastavení programu, které je vhodné v budoucnu rozšířit. V základní verzi aplikace je dostupný přepínač --global či -g pro použití hintů na globální úrovni, neboť defaultní aplikátor hintů pracuje pouze na úrovni lokální.

6.4 Možnosti rozšíření aplikace

Implementovaná aplikace splňuje definované požadavky a poskytuje funkční nástroj pro generování diagramů diagramů tříd. V průběhu vývoje se ukázalo několik oblastí, v nichž lze funkcionalitu nástroje dále rozšířit. Kontrola výstupů zároveň odhalila případy, kdy výsledná podoba diagramů neodpovídala očekávání z důvodu chyb ve vykreslování knihovnou PlantUML. Tato kapitola proto shrnuje návrhy na opravy a doplňující rozšíření, která přesahují původní návrh a mohou významně přispět k efektivitě i uživatelskému komfortu aplikace.

Oprava specifických případů

V tomto oddíle jsou probrány situace, kdy PlantUML neposkytuje očekávané výstupy při skrývání a přebarvení některých prvků. Tyto nedostatky byly zjištěny v případě generování diagramů s příkazem pro schování poznámek a hran.

U poznámek problém nastává v případě, že z poznámky vychází pouze jediná hrana. PlantUML v tomto scénáři vykresluje poznámku odlišně od situace, kdy má více hran nebo žádnou. Poznámka je automaticky skryta s ukrytím prvku, ke kterému patří, což nerozporuje praktickému použití. Avšak není možné skrýt samotnou poznámku nebo její jedinou hranu. V implementaci je tento problém řešen pomocí automatického přidání neviditelné reflexivní hrany ke každé, aby z poznámky vycházelo více hran a skrývání i přebarvení hrany fungují podle očekávání. Nevýhodou tohoto postupu je méně přehledný zdrojový kód a vliv reflexivní vazby na rozmístění prvků, neboť i neviditelná hrana ovlivňuje výsledné rozvržení.

Další problém se týká hran, které se účastní propojení asociační třídy. Jedná se o hranu vztahu, k němuž je asociační třída připojena, a hranu tohoto připojení. Tyto hrany nelze v PlantUML samostatně skrýt ani barevně zvýraznit. Částečné skrývání těchto hran je uskutečněno pouze v případě skrytí prvku, z něhož hrana vychází. V takovém případě je skryta část hrany od daného prvku do bodu průniku propojených hran. Pro tuto situaci dosud nebyl nalezen žádný funkční zástupný mechanismus. V případě potřeby vizualizace přidávání nebo odebírání asociačních tříd je proto nutné počítat s nefunkčním skrýváním hran a nemožností jejich barevné úpravy. Alternativním řešením může být využití jiných notací pro modelování asociací, které tyto nedostatky obcházejí.

Funkční rozšíření

Další oblastí možného vývoje aplikace je rozšíření její funkční výbavy. Tyto návrhy směřují k obohacení mechanismů pro ovlivnění rozmístění prvků a zpřístupnění užitečných možností generování z PlantUML.

V současné implementaci jsou pouze dvě varianty aplikátoru a jejich volbu lze ovlivnit použitím přepínače pro explicitní volbu globální varianty nápovědy. Zadefinováním přepínače pro volbu konkrétního použitého aplikátoru nápovědy pro pozice, by bylo vhodné pro zpřístupnění více způsobů ovlivnění rozvržení. Potom by bylo možné přidat nové varianty aplikátorů, z kterých by si uživatel mohl vybrat konkrétní aplikátor podle svých potřeb.

Další funkční rozšíření by si vyžádala modifikaci jazyka vstupního popisu. Jedná se o možnost specifikovat preferovanou vzájemnou polohu konkrétních dvou prvků. Další takové rozšíření je přidání nového prvku představující nadpis diagramu. Základní podpora tohoto prvku je již v PlantUML dostupná, implementace by tak spočívala zejména v jeho integraci do vnitřní reprezentace. V diagramu může být přítomen pouze jednen nadpis, ale díky specifikaci platnosti proměnných parametrů nadpisu by bylo možné přepisovat v animaci jeho obsah, obdobně jako u editace ostatních prvků.

Vykreslení ortogonálně vedených hran, která je již v PlantUML podporováno, by mohlo být přeneseno i do budoucí verze vyvinuté aplikace. Avšak výsledky tohoto vykreslení nejsou optimální podle metriky křížení hran, a proto je tato možnost vhodná pouze ve specifických případech. Uvažovat by se dalo buď o povolení původního vyjádření v popisném jazyce, nebo o zavedení přepínače. Oba přístupy mají své výhody a nevýhody a teoreticky je možné zavést obě možnosti s tím, že volba z přepínače by měla mít větší prioritu.

Poslední potenciálně nejpřínosnější a zároveň nejkomplikovanější z vyjmenovaných je rozšíření o podporu více typů diagramů. Z pohledu UML by bylo vhodné doplnit diagramy pro modelování chování, jako například sekvenční diagramy, nebo diagramy aktivit.

Rozšíření přizpůsobivosti

Pro zvýšení komfortu a přizpůsobivosti aplikace lze uvažovat o zavedení funkcí, které uživateli poskytnou detailnější kontrolu nad podobou výstupů i nad způsobem práce s prvky. Tato rozšíření by měla být dostupná prostřednictvím přepínačů, což umožní jejich snadné a opakovatelné využití.

Vedle stávajícího generování ve formátu SVG by mohla být dostupná i možnost exportu do dalších formátů, například PNG, které PlantUML nativně podporuje, nebo mezivýstup generátoru v jazyce PlantUML (ve formě jednotlivých souborů i v jediném s komentáři pro určení generovaného snímku).

Flexibilitu zvýší i možnost vybrat, zda má sufix názvů výstupních souborů přiřazen podle pořadí generovaného výstupu, nebo podle čísla snímku. Implementovaná aplikace používá druhý způsob.

Dalším rozšířením je možnost generovat výstup pouze pro konkrétní snímek, nikoli vždy pro celou sekvenci. To je výhodné například při testování dílčích změn.

Přínosem v usnadnění zápisu by byla možnost kaskádového mazání prvků, což by znamenalo, že není nutné uvádět u všech prvků konec platnosti. Při odstranění nadřazeného prvku by se automaticky považovaly za smazané i prvky v něm vnořené a hrany z něj vycházející. Současně by bylo možné nastavit, zda tuto funkci využít, nebo zda provést mazání pouze pro explicitní definici konce platnosti.

Další rozšíření spočívá v zavedení konfiguračního souboru, v němž by uživatel mohl definovat vizuální styl pro stavy prvků (viz Tabulka 4). Tím by se podstatně zvýšila přizpůsobitelnost výsledných výstupů. Modifikace stylu pro konkrétní stav by mohlo specifikovat i případ, v kterém není pro uživatele žádoucí daný stav prvků ve výsledku zvýraznit.

ZÁVĚR

Cílem této diplomové práce bylo vytvořit aplikaci pro generování diagramů tříd na základě textového popisu. Aplikace podporuje tvorbu postupně se vyvíjejících diagramů, automatické zvýrazňování změn a začlenění preferovaných pozic prvků do výpočtu rozvržení grafu.

V teoretické části byly shrnuty základní koncepty jazyka UML a principy automatického rozvržení grafů, které tvořily nezbytný podklad pro návrh vlastního řešení. Analýza existujících nástrojů prokázala, že knihovna PlantUML představuje nejvhodnější základ pro implementaci řešení.

Rozšiřující požadavky na popisný jazyk vedly k modifikaci jazyka PlantUML a k definici formální gramatiky této modifikace. Na základě poznatků z předcházejících částí práce bylo možné navrhnout a implementovat funkční řešení, které kombinuje syntaktickou analýzu textového vstupu, aplikaci nápovědy pro rozmístění prvků na vnitřní reprezentaci diagramu a generování výstupů prostřednictvím PlantUML.

Implementovaná aplikace byla obohacena o vlastní mechanismy zvyšující efektivitu a přehlednost výsledků, například možnost generovat pouze verze diagramů se změnami ke zvýraznění či se vznikem ustáleného stavu prvků, nebo možnost nemuset definovat konec platnosti parametru, který může být automaticky odvozen z navazujících změn.

Hlavní přínos práce spočívá v návrhu a realizaci nástroje, který umožňuje sledovat vývoj diagramů v čase, zjednodušuje tvorbu diagramů s požadovaným rozvržením a eliminuje nutnost rozsáhlých manuálních úprav vstupního popisu podle pravidel automatického rozvržení v PlantUML.

Nad rámec implementovaného řešení byla formulována doporučení pro další rozvoj, například opravy chyb vykreslování v PlantUML, rozšíření variant aplikátorů nápovědy, zavedení přepínačů pro spouštění programu, přidání prvku nadpis, podpora více výstupních formátů, kaskádové mazání závislých prvků nebo využití konfiguračního souboru pro pro přizpůsobení stylu jednotlivých stavů.

Výsledná aplikace nemusí být pouze jednorázové řešení konkrétního problému, ale také základ pro další rozvoj. Rozšíření podpory na širší spektrum UML diagramů by významně zvýšilo její praktickou využitelnost při tvorbě animací, které reprezentují vývoj diagramu a jeho klíčové verze v čase.

POUŽITÁ LITERATURA

ARLOW, Jim a Ila NEUSTADT, 2007. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2. aktualiz. a dopl. vyd. Brno: Computer Press. ISBN 978-80-251-1503-9.

Automatic Diagram Layout, c2024. *Visual Paradigm* [online]. [cit. 2025-08-11]. Dostupné z: https://www.visual-paradigm.com/support/documents/vpuserguide/1283/28/6047_automaticdia.html

BOOCH, Grady, James RUMBAUGH a Ivar JACOBSON, 2005. *The unified modeling language user guide*. 2nd ed. Upper Saddle River, NJ: Addison-Wesley. ISBN 03-212-6797-4.

DIDIMO, Walter, c2025. Orthogonal Drawings of Graphs and Their Relatives. In: *Department of Applied Mathematics* [online]. [cit. 2025-08-11]. Dostupné z: https://kam.mff.cuni.cz/conferences/gd2019/part1.pdf

FOWLER, Martin, 2009. *Destilované UML*. Praha: Grada. Knihovna programátora. ISBN 978-80-247-2062-3. Dostupné také z: http://toc.nkp.cz/NKC/200907/contents/nkc20091928930_1.pdf

CHU, I-Ping, C2001-2025. BNF and EBNF. In: *DePaul University* [online]. [cit. 2025-08-14]. Dostupné z: https://condor.depaul.edu/ichu/csc447/notes/wk3/BNF.pdf

KALLIN, Daniel, [2024]. *Nomnoml* [online]. [cit. 2025-08-14]. Dostupné z: https://nomnoml.com/

KANISOVÁ, Hana a Miroslav MÜLLER, 2006. *UML srozumitelně*. 2., aktualiz. vyd. Brno: Computer Press. ISBN 80-251-1083-4.

KOBOUROV, Stephen a Eric WELCH, 2017. Measuring Symmetry in Drawings of Graphs. *Eurographics Computer Graphics Forum* [online]. **36**(3), 341-351 [cit. 2025-08-11]. Dostupné z: https://www2.cs.arizona.edu/~kobourov/symmetries-eurovis17.pdf

KRUCHTEN, P.B., 1995. The 4 1 View Model of architecture. *IEEE Software* [online]. Institute of Electrical and Electronics Engineers (IEEE), **12**(6), 42-50 [cit. 2025-08-05]. ISSN 0740-7459. Dostupné z: doi:10.1109/52.469759

MAZETTI, Viktor a Hannes SÖRENSSON, 2012. *Visualisation of state machines using the Sugiyama framework* [online]. [cit. 2025-08-11]. Dostupné z: https://odr.chalmers.se/items/2b928745-ced6-4fe9-a4d2-d166a19caf22. Diplomová práce. Chalmers University of Technology, University of Gothenburg, Department of Computer Science and Engineering. Vedoucí práce Peter Damaschke.

Mermaid Documentation, [2025]. *Mermaid* [online]. [cit. 2025-08-14]. Dostupné z: https://mermaid.js.org/intro/

PlantUML Language Reference Guide [online], 2025. [cit. 2025-08-14]. Dostupné z: https://plantuml.com/guide

PURCHASE, Helen, 1997. Which aesthetic has the greatest effect on human understanding? *Lecture Notes in Computer Science* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 248-261 [cit. 2025-08-11]. ISBN 9783540639381. ISSN 0302-9743. Dostupné z: doi:10.1007/3-540-63938-1_67

RUMBAUGH, James, Ivar JACOBSON a Grady BOOCH, c2005. *The unified modeling language reference manual*. 2nd ed. Boston: Addison-Wesley. The Addison-Wesley object technology series. ISBN 03-212-4562-8.

SUGIYAMA, Kozo, Shojiro TAGAWA a Mitsuhiko TODA, 1981. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernetics* [online]. Institute of Electrical and Electronics Engineers (IEEE), **11**(2), 109-125 [cit. 2025-08-11]. ISSN 0018-9472. Dostupné z: doi:10.1109/tsmc.1981.4308636

TAMASSIA, Roberto a Isabel F. CRUZ. Graph Drawing Tutorial. In: *Brown University* [online]. [cit. 2025-08-11]. Dostupné z: https://cs.brown.edu/people/rtamassi/papers/gd-tutorial/gd-constraints.pdf

The Hitchhiker's Guide to PlantUML! [online]. [cit. 2025-08-14]. Dostupné z: https://crashedmind.github.io/PlantUMLHitchhikersGuide/

Unified Modeling Language, c2025. *OMG* [online]. [cit. 2025-08-05]. Dostupné z: https://www.omg.org/spec/UML/2.5.1/PDF

1. YFiles for HTML Documentation, c2025. *YFiles* [online]. [cit. 2025-08-11]. Dostupné z: https://docs.yworks.com/yfiles-html/dguide/layout/layout-summary.html