Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Modelování systémů pomocí vlastností

Bc. Petr Navrátil

Diplomová práce

2020

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Petr Navrátil**
Osobní číslo:	**I18246**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Modelování systémů pomocí vlastností**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem práce je vytvořit vývojové prostředí – aplikaci pro vytváření obecných popisů vlastností a objektů, které tyto vlastnosti implementují. Např. schopnost změnit polohu, a pro popis systémů jako kolekce těchto obecně definovaných vlastností, např. automobil je schopen se pohybovat, převážet náklad atd.). V teoretické části bude analyzována problematika ontologií a objektově orientovaného programování/modelování. V praktické části pak student navrhne vývojové prostředí pro vytváření obecných popisů vlastností a chování a pro popis systémů jako kolekce těchto obecně definovaných vlastností v textové, nebo grafické formě. Poté demonstruje jeho funkčnost při vytváření modelu technického systému.

Rozsah pracovní zprávy: **Dle pokynů vedoucího DP**
Rozsah grafických prací: **Dle pookynů vedoucího DP**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

MERUNKA, Vojtěch. Objektové modelování. Praha: Alfa Nakladatelství, 2008. Informatika studium (Alfa Nakladatelství). ISBN 978-80-87197-04-2.
HOEKSTRA, Rinke. Ontology representation: design patterns and ontologies that make sense. Fairfax, VA: IOS Press, c2009. Frontiers in artificial intelligence and applications, 197. ISBN 978-1607500131.

Vedoucí diplomové práce: **doc. Ing. Tomáš Brandejský, Dr.**
Katedra softwarových technologií

Datum zadání diplomové práce: **5. listopadu 2019**
Termín odevzdání diplomové práce: **15. května 2020**

Univerzita Pardubice Česká republika
L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

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

V Pardubicích dne 15. listopadu 2019

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, 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. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 5. 11. 2019

Petr Navrátil

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu mé diplomové práce panu doc. Ing. Tomášovi Brandejskému, Dr. za odborné vedení a cenné rady, které mi poskytl v průběhu práce.

ANOTACE

V této práci je nejprve popsána problematika ontologií a sémantického webu. Následně je popsáno objektově orientované programování a jazyk UML, jakožto představitel jazyka pro objektově orientované modelování. Potom je popsána myšlenka modelování systémů pomocí vlastností. Následně je realizována desktopová aplikace umožňující vytváření obecných popisů vlastností a objektů, které tyto vlastnosti implementují.

KLÍČOVÁ SLOVA

modelování, systém, vlastnost, objekt, sémantika, syntaxe

TITLE

System modeling by properties

ANNOTATION

In this work is first described ontology problematics and semantic web. Next is described object-oriented programming and the UML language, as representative of language for object-oriented modeling. At the end of the theoretical part is described the idea of modeling of systems using properties. In the practical part is designed and implemented desktop application which allows creating of universal description of properties and objects, which implements these properties.

KEYWORDS

modeling, system, property, object, semantics, syntax

OBSAH

Seznam obrázků **10**

Seznam zdrojových kódů **11**

Seznam zkratek **12**

Úvod **13**

1 Problematika ontologií **14**

1.1 Ontologický datový model 14

1.2 Ontologické jazyky 14

1.3 Nástroje pro práci s ontologiemi 15

1.4 Praktické aplikace ontologií 15

2 Syntaxe a sémantika **16**

2.1 Syntaxe 16

2.2 Sémantika 16

3 Sémantický web **18**

3.1 RDF 18

3.1.1 RDFS 19

3.2 OWL 19

3.3 Shrnutí sémantického webu 19

4 OOP **20**

4.1 Třída 20

4.2 Objekt 20

4.3 Atribut 20

4.4 Metoda 20

4.5 Zapouzdření dat 21

4.6 Dědičnost 21

4.7 Rozhraní 21

4.8 Shrnutí OOP 21

5 UML **23**

5.1 Historie jazyka UML 23
5.2 Stavební bloky 23
5.2.1 Předměty 24
5.2.2 Relace 24
5.2.3 Diagramy 24
5.3 Nástroje pro práci s UML 25
5.4 Shrnutí UML 25
6 Modelování systémů pomocí vlastností 26
6.1 Multiphysics 26
7 Vývojové prostředí (aplikace) 28
7.1 Jazyk Java 28
7.2 Knihovna JavaFX 28
7.2.1 JavaFX Scene Graph 29
7.3 Datový model 30
7.3.1 Rozhraní `ISystemComponent` 31
7.3.2 Třída `AbstractSystem` 31
7.3.3 Třída `SimpleSystem` 32
7.3.4 Třída `ComplexSystem` 32
7.3.5 Rozhraní `IPropertyComponent` 32
7.3.6 Třída `AbstractProperty` 32
7.3.7 Třída `SimpleProperty` 33
7.3.8 Třída `ComplexProperty` 33
7.3.9 Výčtový typ `ValueType` 33
7.3.10 Třída `Attribute` 33
7.3.11 Třída `Method` 33
7.3.12 Třída `Parameter` 34
7.3.13 Pomocné třídy 34
7.4 Aplikace pro modelování systémů 34
7.4.1 Architektura a součásti aplikace 34
7.4.2 Hlavní balíček aplikace 35
7.4.3 Balíček `dialog` 36

7.4.4 Balíček `graphics` 37
7.4.5 Balíček `util` 38
7.4.6 Balíček `serialization` 39
7.5 Uživatelské rozhraní aplikace 40
8 Demonstrace funkcí aplikace 42
Závěr 43
Použitá literatura 44
Přílohy 46

SEZNAM OBRÁZKŮ

Obrázek 1 - předpoklady realizace Sémantického webu, zdroj: vlastní 18
Obrázek 2 - typy UML diagramů podle standardu UML 2.2, zdroj: vlastní 25
Obrázek 3 - porovnání tříd a systémů s vlastnostmi, zdroj: vlastní 26
Obrázek 4 - ukázka JavaFX Scene Graph, zdroj: vlastní 30
Obrázek 5 - diagram tříd balíčku abstraction datového modelu, zdroj: vlastní 30
Obrázek 6 - architektura aplikace a použité knihovny, zdroj: vlastní 35
Obrázek 7 - struktura grafického uživatelského rozhraní aplikace, zdroj: vlastní 41

SEZNAM ZDROJOVÝCH KÓDŮ

Zdrojový kód 1 - metoda testující, zda systém obsahuje danou vlastnost 31
Zdrojový kód 2 - vytvoření vizuální ikony vlastnosti, pokud uživatel potvrdil dialog 39

SEZNAM ZKRATEK

API	Application Programming Interface
B2B	Business to Business
B2C	Business to Customer
GUI	Graphical User Interface
OMG	Object Management Group
OMT	Object Modeling Technique
OOP	Object-oriented Programming
OSMC	Open Source Modelica Consortium
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
UML	Unified Modeling Language
URI	Uniform Resource Identifier
XML	Extensible Markup Language

ÚVOD

Cílem této práce je navrhnout a implementovat vývojové prostředí neboli aplikaci, která bude umožňovat tvorbu a popis vlastností a dále tvorbu systémů jako kolekce vybraných definovaných vlastností. V teoretické části práce budou popsány a vysvětleny některé důležité pojmy z oblasti objektového programování a modelování. Nejdříve bude vysvětlen pojem ontologie v oblasti informačních systémů. Následovat bude vysvětlení rozdílů a ujasnění pojmů syntaxe a sémantika, na které naváže část zabývající se sémantickým webem. Další část bude věnována objektově orientovanému programování a následovat bude část o jazyku UML, který je představitelem jazyka pro objektově orientované modelování. V samostatné kapitole bude vysvětlena myšlenka modelování systémů pomocí vlastností. Druhá část teoretické části se bude zabývat popisem návrhu a implementace samotné aplikace. Nejdříve budou stručně popsány použité technologie, následovat bude popis navrženého datového modelu, a nakonec bude popsána aplikace s grafickým uživatelským rozhraním využívající navržený model. Poslední kapitola bude demonstrovat funkcionalitu aplikace při modelování vybraného systému.

1 PROBLEMATIKA ONTOLOGIÍ

Ontologie je filozofická disciplína, která se zabývá jsoucnem, bytím jako takovým a základními pojmy. Řecký filozof Aristotelés pro ni používá označení první filozofie, která je součástí metafyziky a zabývá se nejobecnějšími otázkami [7].

V oblasti informačních technologií má ontologie ale jiný význam. Jedná se o výslovný a formalizovaný popis určité problematiky. Je to formální a deklarativní reprezentace, která obsahuje definici pojmů a definici vztahů mezi jednotlivými pojmy. Je to teda vlastně jakýsi slovník, sloužící k uchovávání a předávání znalostí týkajících se určité problematiky [6].

1.1 Ontologický datový model

Ontologie se používají v oblastech umělé inteligence, softwarového inženýrství, sémantického webu a systémového inženýrství jako datový model, který reprezentuje určitou znalost nebo její část. Obecně datový model ontologie obsahuje následující čtyři typy prvků:

1. **Entita** (instance, objekt, jedinec) – základní stavební prvek datového modelu ontologie. Entita může být konkrétní nebo abstraktní.
2. **Třída** (kategorie) – množina entit určitého typu. Podmnožinou třídy je podtřída. (podkategorie). Třída může obsahovat zároveň entity i podtřídy.
3. **Atribut** – určitá vlastnost, charakteristika nebo parametr entity. Každý atribut má nejméně název a hodnotu a slouží k uložení určité informace vztahující se k dané entitě.
4. **Vazba** – jednosměrné nebo obousměrné propojení dvou entit. Vazba je vlastně určitým typem atributu, jehož hodnotou je jiná entita v ontologii.

1.2 Ontologické jazyky

Pro reprezentaci ontologie se používají tzv. ontologické jazyky [9]. Ontologický jazyk je v informatice seznam určitých výrazů popisující nějakou ontologii. Ontologických jazyků je velké množství, jako příklad lze uvést jazyk CycL, který byl použit v projektu Cyc, což je jeden z úplně prvních projektů zabývající se zachycováním znalostí. Dalším důležitým jazykem je jazyk SHOE (Simple HTML Ontology Extension), což je první jazyk, který vznikl specificky za účelem přidání sémantiky k webovým stránkám. Tento jazyk vznikl v roce 1996 a umožňuje

do zdrojového kódu webových stránek začleňovat metadata o objektech, kterých se tyto stránky týkají a také umožňuje začleňovat samotné ontologie, které definují sémantiku metadat.

1.3 Nástroje pro práci s ontologiemi

Prakticky všechny ontologické jazyky lze zpracovávat běžným textovým editorem, ale existuje také množství specializovaných nástrojů – editorů. Některé nástroje jsou určené pouze pro nějaký konkrétní ontologický jazyk, ale existují i takové nástroje, které nejsou bezprostředně spojeny s jedním konkrétním jazykem. Jedním z velice propracovaných systémů, který není spojen s žádným jazykem, ale lze z něj provádět export do všech hlavních formátů, je nástroj **Protégé** vyvinutý v *Stanford Center for Biomedical Informatics Research*. Tento nástroj je zdarma a s otevřeným zdrojovým kódem a obsahuje také velké množství pluginů, které ještě rozšiřují jeho funkcionalitu.

1.4 Praktické aplikace ontologií

Ontologie se v současnosti využívají v několika oblastech a jsou to např. následující oblasti [5]:

- **Znalostní management ve firmách** – s pomocí ontologií, lze zachytit informace a znalosti interního i externího původu, zabezpečit jejich konzistenci, usnadnit jejich vyhledávání a tyto znalosti mohou využívat různí pracovníci dané organizace;
- **Elektronické obchodování** – může se jednat o obchodování typu B2C nebo B2B. Zákazníkovi může ontologie usnadnit vyhledání požadovaného produktu a firmám může pomoci s vyhledáním potenciálního partnera;
- **Zpracování přirozeného jazyka** – ontologie mohou pomáhat při překladu nebo také při sumarizaci textů;
- **Inteligentní integrace informací**;
- **Inteligentní výukové systémy**.

2 SYNTAXE A SÉMANTIKA

Tato kapitola popisuje dva důležité pojmy z oblasti informačních technologií, které hrají důležitou roli při programování a modelování, a to jsou *syntaxe* a *sémantika*. Protože tyto pojmy budou ještě několikrát použity v dalších kapitolách, věnuje se jim tato kapitola.

2.1 Syntaxe

Syntaxe programovacího jazyka představuje množinu pravidel, které definují kombinace symbolů, které jsou považovány za správně strukturovaný dokument nebo fragment daného jazyka. Toto platí jak pro programovací jazyky, kde dokument reprezentuje zdrojový kód, tak i pro značkovací jazyky, kde dokument reprezentuje data.

V mnoha běžných programovacích jazycích jako jsou např. C++, Java, C# jsou jednotlivé příkazy programu (zdrojového kódu) ukončeny středníkem. Pokud středník nenapíšeme, vznikne syntaktická chyba, tzv. *Syntax Error* a kompilace nebo spuštění našeho programu selže. V současné době dokáže velké množství různých vývojových prostředí a editorů rozpoznávat syntaktické chyby a upozornit nás na ně.

Kromě syntaxe v oblasti informačních technologií je také syntax lingvistická disciplína, která se zabývá vztahy mezi slovy ve větě, správným tvořením větných konstrukcí a slovosledem. (což je vlastně velmi podobné tomu co známe z programovacích jazyků)

2.2 Sémantika

Sémantika představuje skutečný **význam** daného programovacího jazyka, jeho klíčových slov, datových struktur nebo obecně význam dat.

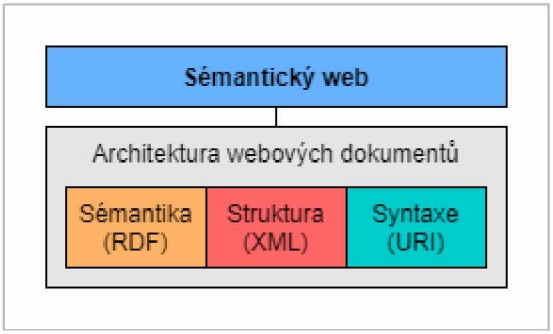
Syntakticky správný program, který lze úspěšně zkompilovat a spustit nemusí být sémanticky správně. Například můžeme v nějakém programovací jazyku vytvořit funkci pro sčítání dvou čísel, která má jako parametry daná čísla, ale ve vlastním těle funkce tyto čísla místo sečtení odečteme a vrátíme výsledek. Tato funkce bude syntakticky správně a budeme jí moct používat, ale sémanticky bude špatně. Dalším příkladem může být značkovací jazyk HTML, používaný k tvorbě webových stránek. Tento jazyk obsahuje množství tagů a každý tag má nějaký určený význam. Pokud bychom chtěli vytvořit nadpis k nějakému textu a použili bychom tag `<bold></bold>`, tak to bude syntakticky správně, ale sémanticky špatně, protože tento tag

slouží k tvorbě tučného textu, ale pro tvorbu nadpisů je určen například tag `<h1></h1>`. Odhalování sémantických chyb je obecně složitější než odhalování chyb syntaktických.

Podobně jako syntax je také sémantika součástí lingvistiky, jako nauka o významu výrazů z různých strukturních úrovní jazyka, například slov, slovních spojení a vět.

3 SÉMANTICKÝ WEB

Sémantický web je koncept webu založený na obsahu, který je vytvořen a strukturován podle určitých pravidel a standardů a umožňuje efektivnější a snadnější vyhledávání informací[1]. Realizace sémantického webu předpokládá implementaci standardů pro sémantickou, strukturální a syntaktickou složku architektury webových dokumentů. Výsledkem této aplikace uvedených standardů bude konzistentní logická struktura dat, která bude implicitně vyjadřovat **význam zaznamenaných informací**.



Obrázek 1 - předpoklady realizace Sémantického webu, zdroj: vlastní

Myšlenka sémantického webu je tady od roku 2001, kdy ji poprvé vyslovil sám autor „klasického" webu Tim Berners-Lee. Problém s klasickým webem je ten, že většinu tvoří dokumenty, které mohou číst lidé, ale počítačové programy, které by s daty mohly manipulovat si s nimi poradit neumí. počítače jsou schopny rozpoznat, která část dokumentu je hlavička, která je odkazem na jiný dokument, ale už nejsou schopné rozeznat, že jde o domovskou stránku nemocnice a odkaz vede k životopisu lékaře.

Cílem sémantického webu je posunout strojovou interpretovatelnost informací uložených na webu. Pro počítače je velmi těžké zjistit o čem webová stránka je, tím že se budou dívat na její strukturu. Prvním krokem k sémantickému webu je specifikace způsobu, jak přidávat relativně jednoduchá metadata k dokumentům na webu.

3.1 RDF

Resource Description Framework představuje flexibilní způsob reprezentace metadat na webu. Je to jednoduchý jazyk, který definuje způsob tvorby tvrzení o věcech na webu. Tyto tvrzení

[1] více o sémantickém webu https://www.w3.org/2001/sw/ nebo https://www.w3.org/standards/semanticweb/

mají formu trojic označovaných <s, p, o> skládajících se z podmětu (*subject*), predikátu (*predicate*) a předmětu (*object*). Tyto trojice jsou definovány tak, že predikát p je relace mezi s a o. Každá část trojice, tzv. RDF jméno, představuje zdroj (*resource*). To, jak je zacházeno se jmény záleží na jejich syntaktické formě. Zdroje mohou být identifikovány pomocí URI (*Uniform Resource Identifier*), pomocí čistých textových řetězců, nebo pomocí řetězců s XML Schema datovým typem. RDF zdroj může být *cokoliv* a nemusí to existovat na webu.

Kolekce propojených RDF trojic představuje směrovaný graf, kde uzly jsou podměty a předměty a hrany jsou vlastnosti.

RDF grafy mohou být uchovávány v různých formátech. Existují tři oficiální serializační syntaxe pro RDF: RDF/XML, N-Triple a Terse RDF Triple Language (Turtle).

3.1.1 RDFS

Přestože RDF může být použito pro popis komplexních grafových struktur, poskytuje toho málo ve smyslu sémantiky a automatického odvozování. RDF Schema (RDFS) rozšiřuje RDF o základní primitiva, která nám umožňují vyjádřit více obecné znalosti.

3.2 OWL

Web Ontology Language (OWL) je rodina jazyků pro popis ontologií. Tyto jazyky jsou postaveny na RDF standardu. OWL rodina obsahuje mnoho druhů, syntaxí a specifikací se stejnými jmény. OWL a OWL2 odkazují na specifikace z roku 2004 a 2009.

3.3 Shrnutí sémantického webu

Přestože od vyslovení myšlenky uplynulo již skoro 20 let a je sémantický web již řadu let propagován, tak se zatím nedočkal výrazného rozmachu. Jednou z jeho nevýhod a také pravděpodobných příčin jeho malého rozšíření je to, že je oproti stávajícímu webu dosti komplikovaný. Vznikají také další a jednodušší způsoby, jak do stávajícího webu (snadno) přidat sémantickou informaci.

4 OOP

V této kapitole bude popsáno Objektově orientované programování (OOP), které v současné době představuje jedno z nejvíce používaných programovacích paradigmat. Podpora jednotlivých principů objektově orientovaného programování se může v rámci různých programovacích jazyků lišit, proto v této kapitole budou popsány hlavně obecné koncepty a vlastnosti. Ze zástupců objektově orientovaných programovacích jazyků můžeme jmenovat například: C++, C#, Java, Python, Swift, Groovy, Kotlin, Scala, PHP a mnoho dalších.

4.1 Třída

Třída je základním kamenem OOP. Představuje soubor proměnných a podprogramů. Proměnným se říká členské proměnné (*member variables*) nebo datové složky nebo atributy a slouží k uchovávání stavu objektu. Podprogramům se říká metody *(methods*) a ty manipulují s členskými proměnnými a tím mění stav objektu. Třída představuje jakousi šablonu, na základě, které můžeme vytvářet jednotlivé konkrétní objekty. Třída reprezentuje nějaký objekt reálného světa jako je člověk, auto, židle, dům a další.

4.2 Objekt

Objekt je datový prvek, který je vytvořen podle nějaké třídy. Používá se také termín *instance třídy*. Podle dané třídy lze vytvořit libovolné množství objektů, které mají stejné schopnosti, ale mohou mít různé hodnoty členských proměnných.

4.3 Atribut

Atribut nebo také členská proměnná nebo datová složka uchovává stav daného objektu. Každý objekt má svoje vlastní hodnoty atributů. Každý atribut má nějaký datový typ, který určuje, jaké hodnoty, nebo co vlastně daný atribut uchovává.

4.4 Metoda

Metoda představuje nějaký podprogram, který může manipulovat s atributy daného objektu. Metody mohou mít vstupní parametry a také návratovou hodnotu. Pomocí vstupních parametrů můžeme metodě předat nějaké hodnoty, které potřebuje pro svůj výpočet a pomocí návratového hodnoty metody můžeme získat výsledek provedeného výpočtu.

4.5 Zapouzdření dat

Zapouzdření dat (*data encapsulation*) je jednou z důležitých vlastností objektově orientovaných jazyků. Pomocí zapouzdření dosáhneme toho, že k datům dané třídy lze přistupovat a manipulovat s nimi jenom s využitím metod dané třídy. Důsledkem zapouzdření je tedy vlastně **autorizovaný přístup k datům**.

4.6 Dědičnost

Dědičnost umožňuje vytvořit třídu, která rozšiřuje nějakou jinou třídu o další atributy a metody. Tato třída má všechny atributy a metody třídy, ze které dědí a zároveň může definovat různé další atributy a metody. V souvislosti s dědičností se používají termíny *předek* a *potomek*. Některé programovací jazyky umožňují tzv. vícenásobnou dědičnost, to znamená že třída může dědit z více různých tříd. Vícenásobná dědičnost přináší určité výhody, ale také problémy. Určitou obdobou vícenásobné dědičnosti v některých jiných programovacích jazycích jsou tzv. rozhraní.

4.7 Rozhraní

Rozhraní (*interface*) představují určitou obdobu vícenásobné dědičnosti. Najdeme je například v jazycích Java a C#. V rozhraních můžeme deklarovat hlavičky metod, tj. název metody, jednotlivé vstupní parametry, jejich typy a návratový typ metody. Následně můžeme vytvořit třídu, která bude tzv. implementovat dané rozhraní a to znamená, že daná třída musí nadefinovat chybějící těla a tím pádem funkcionalitu metod deklarovaných v daném rozhraní. Obecně může třída implementovat více než jedno rozhraní a rozhraní může být implementováno různými třídami, kde jednotlivé třídy mohou nadefinovat rozdílnou funkcionalitu.

4.8 Shrnutí OOP

Objektově orientované programování představuje velmi dobře uchopitelný a představitelný přístup k programování, čemuž odpovídá jeho současná popularita. Jedním z problémů různých programovacích paradigmat a také OOP je to, že nelze nijak obecně nebo snadno kontrolovat sémantiku určité části kódu. Pokud bychom měli například deklarovanou metodu `vykresli(...)` (buď v rozhraní nebo třeba v abstraktní třídě), tak při implementaci této metody v nějaké třídě budeme jen těžko moci zkontrolovat, že tato metoda opravdu provádí

třeba vykreslení nějakého objektu nebo tvaru na obrazovku a že naopak nedělá něco jiného. Musíme věřit programátorovi, který implementoval danou metodu, že dodržel *kontrakt* (slovní popis toho co má daná metoda dělat).

5 UML

Jazyk UML (Unified Modeling Language), česky unifikovaný modelovací jazyk je **univerzální** jazyk pro **vizuální modelování** systémů. Nejčastěji je spojován s modelováním objektově orientovaných softwarových systémů, má ale mnohem širší využití díky jeho zabudovaným rozšiřovacím mechanismům. Byl navržen proto, aby spojil nejlepší existující postupy modelovacích technik a softwarového inženýrství. Velkou výhodou je to, že diagramy vytvořené v jazyku UML jsou **srozumitelné pro lidi**, ale mohou je také snadno interpretovat specializované programy. Jazyk UML poskytuje vizuální syntaxi, kterou můžeme využít při sestavování svých modelů [4].

5.1 Historie jazyka UML

Před rokem 1994 existovalo několik objektově orientovaných modelovacích metod. Soupeřilo mezi sebou několik jazyků pro vizuální modelování a také metodik. Více než polovina tehdejšího trhu byla rozdělena mezi metody Booch (jejím autorem byl p. Booch) a OMT (Object Modeling Technique, jejím autorem byl p. Rumbaugh). Bylo několik pokusů o sjednocení, které ale selhaly. Autoři zmíněných metod se spojily ve firmě Rational Corporation, která pracovala na tvorbě jazyka UML. Práce na jazyku UML začaly v roce 1995. V roce 1996 navrhlo sdružení OMG (Object Management Group) specifikaci pro objektově orientovaný jazyk pro vizuální modelování. Toto sdružení pak v roce 1997 přijalo jazyk UML a vznikl tak **první průmyslový standard objektově orientovaného jazyka pro vizuální modelování**. V roce 2005 nahradila revize UML 2.0 současnou verzi UML 1.5 a v současné době (rok 2020) je aktuální specifikace UML 2.5.1 [4].

5.2 Stavební bloky

Jazyk UML je sestaven ze tří základních stavebních bloků:

- **předměty** (*things*) – samotné prvky modelu;
- **vztahy** (*relationships*) – spojení mezi předměty, určují, jak spolu předměty významově souvisí;
- **diagramy** (*diagrams*) – pohledy na modely UML, ukazují kolekce předmětů a jsou naším způsobem vizualizace toho, *co* systém bude dělat a *jak* to bude dělat.

5.2.1 Předměty

Předměty nebo také „věci“ nebo abstrakce dělíme v jazyku UML na:

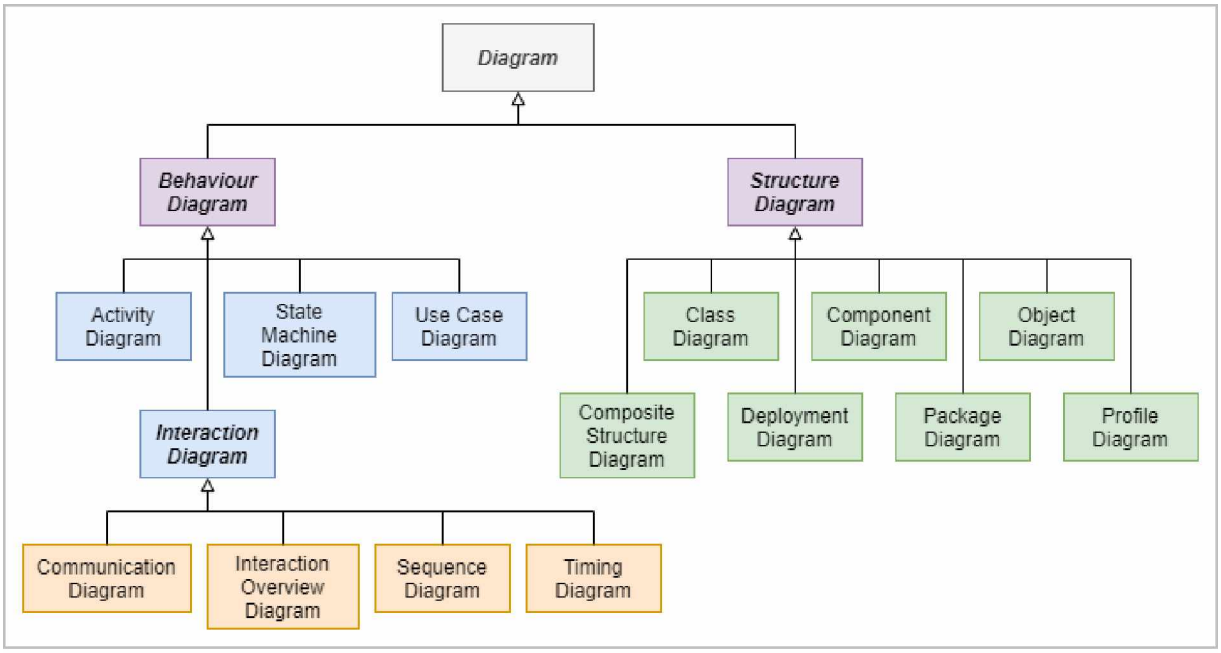
- **strukturní abstrakce** *(structural things)* – podstatná jména modelu UML, jako jsou třídy, rozhraní, spolupráce, případ užití, komponenta;
- **chování** (*behavioural things*) – slovesa modelu UML, jako například interakce, stav;
- **seskupení** (*grouping things*) – balíčky používané k seskupování sémanticky souvisejících prvků modelu;
- **poznámky** (*annotational things*) – anotace, které lze k modelu připojit s úmyslem zachytit informaci sestavenou jen k tomuto účelu.

5.2.2 Relace

K zachycení vztahu mezi dvěma předměty modelu slouží relace. Umožňují zachytit sémantický vztah mezi předměty. Při modelování v jazyku UML je důležité porozumět přesné sémantice jednotlivých typů relací. Mezi nejpoužívanější relace patří například Asociace (*Association*), Agregace (*Aggregation*) nebo Zobecnění (*Generalization*).

5.2.3 Diagramy

Diagramy představují různé pohledy na model. Model obsahuje všechny předměty a relace, ale jednotlivé diagramy mohou obsahovat jen vybrané předměty a relace a také při odstranění předmětu z určitého diagramu nedochází k jeho odstranění z modelu. Diagramů je v jazyce UML značné množství a některé z nich byly přidány až v průběhu vývoje. Následující obrázek znázorňuje jednotlivé typy diagramů.



Obrázek 2 - typy UML diagramů podle standardu UML 2.2, zdroj: vlastní

Diagramy se dělí na dvě základní skupiny, a to jsou diagramy struktury (*Structure Diagram*) a diagramy chování (*Behaviour Diagram*). Diagramy chování ještě obsahují podkategorii diagramů interakce (*Interaction Diagram*). Mezi jednoznačně nejpoužívanější diagramy struktury patří diagram tříd (*Class Diagram*) a objektový diagram (*Object Diagram*). Z diagramů chování se nejvíce používá diagram případů užití (*Use Case Diagram*) a diagram aktivit (*Activity Diagram*).

5.3 Nástroje pro práci s UML

Existuje velké množství softwarových nástrojů pro modelování v jazyku UML, které se liší svojí funkcionalitou a také cenou. Některé nástroje určené zejména pro komerční použití jsou placené, ale mnoho jich je zdarma nebo dokonce s otevřeným zdrojovým kódem. Mezi placené a komerčně nejrozšířenější patří nástroj **Enterprise Architect** a do kategorie zdarma a s otevřeným zdrojovým kódem patří například nástroj **Modelio**.

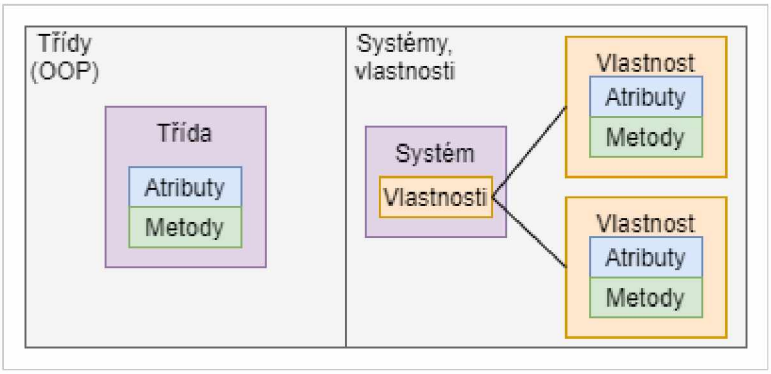
5.4 Shrnutí UML

Jazyk UML představuje velmi mocný nástroj pro modelování systémů. Mezi jeho přednosti patří to, že je to vizuální jazyk a také jeho rozšiřitelnost o další prvky, čímž můžeme získat další nástroje pro modelování různých specializovaných systémů. Jeho určitou nevýhodou může být právě jeho rozsáhlost a s tím spojený čas na jeho dostatečné pochopení.

6 MODELOVÁNÍ SYSTÉMŮ POMOCÍ VLASTNOSTÍ

Tento způsob modelování vychází z toho, že základním elementem jsou vlastnosti. Vlastnosti jsou ve větší či menší míře abstrakcí různých vlastností, které známe z reálného světa. Například automobil má vlastnost barva, nebo vlastnost polohy v prostoru, či vlastnost pohybu. Dalším příkladem může být člověk, který má vlastnosti jako je váha, výška, věk nebo také poloha v prostoru. Vlastnosti jsou univerzální a jsou nositeli (požadovaných) atributů. Zároveň také vlastnosti s ohledem na svůj význam a atributy definují (požadované) funkce (metody). Konkrétní systémy nebo také objekty, nebo třídy vznikají jako množiny, tvořené z konkrétních vlastností. Tyto množiny odpovídají matematickým množinám, to znamená, že každá množina může obsahovat danou vlastnost právě jednou.

Při použití tohoto přístupu se v první řadě zaměřujeme na správný popis vlastností. Vlastnosti by měli být popsány tak, aby obsahovaly jen nezbytně nutné atributy a metody. Jednotlivé systémy pak samy o sobě žádné atributy a metody nemají, jsou pouze tvořeny určitou množinou požadovaných vlastností. Následující obrázek zobrazuje rozdíl mezi klasickými třídami (objekty) z pohledu OOP a mezi systémy a vlastnostmi.



Obrázek 3 - porovnání tříd a systémů s vlastnostmi, zdroj: vlastní

6.1 Multiphysics

V současné době se také začíná používat pojem *multiphysics*. Definice tohoto pojmu se poměrně liší, jedna z nich ho definuje například takto: spojené procesy nebo systémy zahrnující více než jednu současně se vyskytující fyzikální oblast a studium znalostí o těchto procesech a systémech [10]. V oblasti modelování a simulací si výzkumníci uvědomili, že různé objekty mají různé (fyzikální) soubory vlastností, z nichž každý řeší jiné simulační nástroje a metody. Například letadlo má kromě aerodynamických vlastností také třeba různé mechanické vlastnosti, které zaručují, že se letadlo nerozpadne, nebo třeba elektrické vlastnosti, díky nimž

můžeme případně nemůžeme letadlo vidět na radaru. Co se týče softwarových nástrojů zaměřených pro tento druh modelování a simulace můžeme zmínit prostředí **OpenModelica**, o které se stará organizace Open Source Modelica Consortium (OSMC) a z komerčních nástrojů můžeme zmínit například **COMSOL Multiphysics**.

7 VÝVOJOVÉ PROSTŘEDÍ (APLIKACE)

Cílem této práce bylo vytvořit vývojové prostředí – aplikaci umožňující vytváření obecných popisů vlastností a také objektů, které tyto vlastnosti implementují. Pro vývoj aplikace jsem si zvolil objektově orientovaný přístup a programovací jazyk Java 8, jehož součástí je knihovna JavaFX 8, která je určená k tvorbě grafického uživatelského rozhraní pro desktopové aplikace. Použil jsem vývojové prostředí JetBrains IntelliJ IDEA Ultimate a pro správu projektu jsem použil nástroj Gradle, který slouží pro automatizaci sestavování programu. Vývoj aplikace jsem si rozdělil do dvou hlavních etap:

1. návrh a implementace datového modelu,
2. návrh a implementace aplikace s GUI využívající a rozšiřující datový model.

7.1 Jazyk Java

Tento programovací jazyk je v současnosti jedním z nejpoužívanějších objektově orientovaných programovacích jazyků. Mezi jeho hlavní přednosti patří to, že je multiplatformní, automatická správa paměti a syntaxe velmi podobná jazyku C++. Co se týče vlastností OOP, tak jazyk Java podporuje většinu z nich. Jazyk podporuje jednoduchou dědičnost, místo vícenásobné dědičnosti se zde používá dříve zmíněný koncept rozhraní.

7.2 Knihovna JavaFX

Knihovna JavaFX je tvořena množinou zejména grafických balíčků, které umožňují vývojářům designovat, vytvářet, testovat, ladit a nasazovat klientské aplikace, které fungují stejně na různých platformách. Mezi její klíčové vlastnosti patří následující:

- **Java API** – knihovna se skládá z tříd a rozhraní, které jsou vytvořeny pomocí jazyka Java;
- **FXML and Scene Builder** – FXML je deklarativní značkovací jazyk, vycházející z jazyka XML, který slouží k tvorbě grafického uživatelského rozhraní JavaFX aplikací. Kód FXML je možné psát ručně, nebo použít nástroj Scene Builder, který umožňuje interaktivní návrh a následné vygenerování FXML kódu;
- **Built-in UI controls and CSS** – knihovna obsahuje všechny hlavní ovládací prvky potřebné k vývoji plnohodnotné aplikace jako například: tlačítka, zaškrtávací políčka,

seznamy, tabulky a další. Vzhled jednotlivých ovládacích prvků je možné upravit pomocí jazyka CSS;

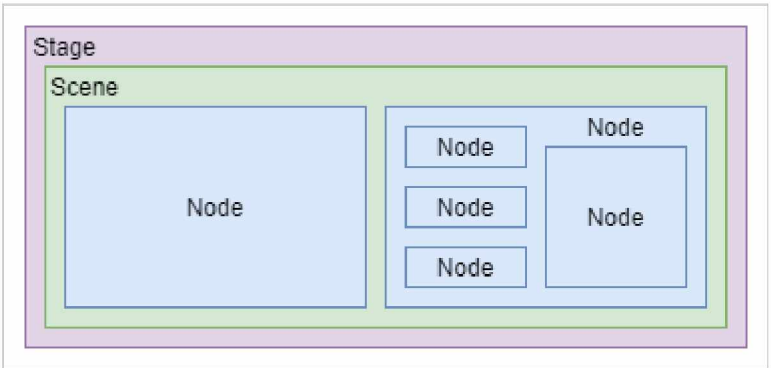
- **3D Graphics Features** – knihovna obsahuje třídy pro práci s 3D grafikou jako jsou `Box`, `Cylinder`, `MeshView`, `Material`, `AmbientLight`, `PointLight` a další;
- **Canvas API** – umožňuje přímé kreslení do určité oblasti v rámci JavaFX scény;
- **Printing API** – podpora pro tisk;
- **Hardware-accelerated graphics pipeline**;
- **High-performance media engine**;
- **Self-contained application deployment model**.

7.2.1 JavaFX Scene Graph

Důležitou součástí knihovny JavaFX je tzv. JavaFX Scene Graph, který definuje hierarchickou strukturu jednotlivých elementů nebo také uzlů grafického uživatelského rozhraní JavaFX aplikace. Základem je třída `Stage` (jeviště), která představuje JavaFX kontejner nejvyšší úrovně. Každá aplikace má tzv. primární jeviště, které je vytvořeno platformou (operačním systémem) při startu JavaFX aplikace, a v rámci běhu aplikace můžeme vytvářet dle potřeby další jeviště. Další třídou je třída `Scene` (scéna), která představuje kontejner pro všechny elementy grafického uživatelského rozhraní. Abstraktní třída `Node` (uzel) představuje základ pro všechny elementy grafického uživatelského rozhraní. Jedním z přímých potomků třídy `Node` je další abstraktní třída `Parent` (rodič) představuje základ pro všechny elementy nebo také uzly, které mohou mít nějaké potomky. Tato třída má dva přímé potomky:

- `Group` – aplikuje různé efekty a transformace na kolekci svých dceřiných elementů;
- `Region` – základní třída pro všechny ovládací prvky grafického uživatelského rozhraní a kontejnery, které umožňují různé rozložení ovládacích prvků.

Typicky JavaFX aplikace obsahuje jeviště, které obsahuje scénu, tato scéna obsahuje jednoho rodiče (tzv. *root*) a ten představuje kořen hierarchické struktury. Další třídy už jsou třídy reprezentující jednotlivé ovládací prvky, nebo různé kontejnery umožňující různé rozložení ovládacích prvků. Následující obrázek ilustruje popsanou hierarchickou strukturu.

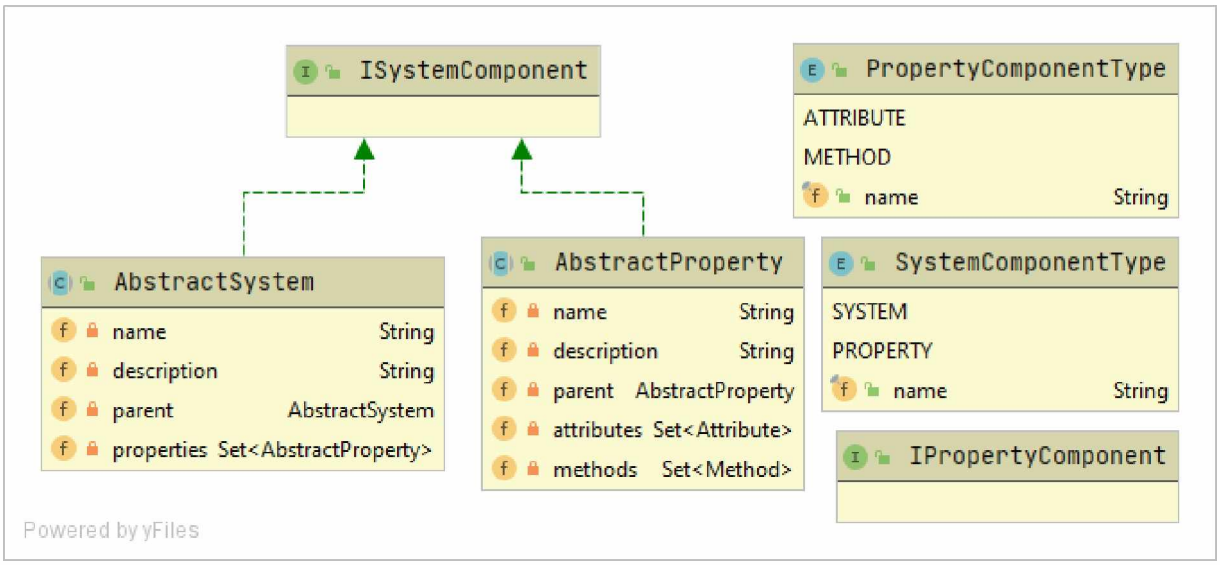


Obrázek 4 - ukázka JavaFX Scene Graph, zdroj: vlastní

7.3 Datový model

V první etapě jsem se zabýval návrhem a implementací datového modelu, který bude umožňovat určitým způsobem popisovat systémy a vlastnosti. Pro datový model jsem vytvořil ve vývojovém prostředí samostatný projekt a od počátku jsem ho vytvářel jako Java knihovnu, kterou bude možné použít v různých aplikacích nezávislou na žádném grafickém uživatelském rozhraní.

V následující části bude stručně popsán navržený a implementovaný datový model, jeho komponenty, tj. třídy, rozhraní a některé jejich důležité metody. Na následujícím obrázku je UML diagram tříd v balíčku `abstraction` datového modelu.



Obrázek 5 - diagram tříd balíčku abstraction datového modelu, zdroj: vlastní

7.3.1 Rozhraní `ISystemComponent`

Toto rozhraní představuje komponentu modelovaného systému. Rozhraní deklaruje jednu důležitou metodu `getComponentType()`, která vrací objekt výčtového typu `SystemComponentType`, který může nabývat buď hodnoty `SYSTEM` nebo `PROPERTY`. Toto rozhraní je implementováno třídami reprezentujícími systém a vlastnosti.

7.3.2 Třída `AbstractSystem`

Tato třída slouží jako základ systému (objektu), který se skládá z různých vlastností. Jedná se o abstraktní třídu, která definuje určité atributy a metody, ale nelze přímo podle ní vytvářet instance, místo toho slouží jako společný základ pro její potomky, z nichž se budou moci vytvářet instance. Nejdůležitějším atributem této třídy je atribut `Set<AbstractProperty> properties`, který představuje množinu vlastností ze kterých se systém skládá. `Set` je v jazyce Java datová struktura odpovídající matematické množině, která se skládá prvků určitého typu, ale každý prvek v ní může být pouze jednou. Dále tato třída obsahuje atributy jako jsou jméno systému nebo jeho jednoduchý textový popis a také atribut představující odkaz na rodičovský systém. Datový model je navržen tak, že umožňuje jednoduchou dědičnost a to znamená, že určitý systém může být potomkem právě jednoho jiného systému a tím že je potomek nějakého systému automaticky získává jeho vlastnosti. Hlavními metodami této třídy jsou metody pro přidávání, nebo odebírání vlastností systému a také metoda `compareTo(...)` sloužící k porovnávaní systémů nebo testování zda jsou systémy stejné. V této abstraktní třídě je testována shoda systému na základě jejich jména.

```
public boolean containsProperty(AbstractProperty property) {
    if (Objects.isNull(property)) throw new NullPointerException();
    if (this.properties.contains(property)) {
        return true;
    } else if (Objects.nonNull(this.parent)) {
        return this.parent.containsProperty(property);
    } else {
        return false;
    }
}
```

Zdrojový kód 1 - metoda testující, zda systém obsahuje danou vlastnost

7.3.3 Třída `SimpleSystem`

Jedná se o potomka třídy `AbstractSystem`, která její funkcionalitu nijak neupravuje, ale na základě této třídy už lze vytvářet instance.

7.3.4 Třída `ComplexSystem`

Další potomek třídy `AbstractSystem`, který je připraven pro složitější implementaci systému, ale v aktuální podobě datového modelu a aplikace pro modelování systémů není nijak využit.

7.3.5 Rozhraní `IPropertyComponent`

Toto rozhraní představuje komponentu vlastnosti systému. Rozhraní deklaruje jednu důležitou metodu `getComponentType()`, která vrací objekt výčtového typu `PropertyComponentType`, který může nabývat buď hodnoty `ATTRIBUTE` nebo `METHOD`. Toto rozhraní je implementováno třídami reprezentujícími systém a vlastnosti.

7.3.6 Třída `AbstractProperty`

Tato třída slouží jako základ vlastností, ze kterých jsou složeny systémy. Podobně jako u třídy `AbstractSystem` se jedná o abstraktní třídu, sloužící jako společný základ pro její potomky, ze kterých se budou vytvářet instance. Nejdůležitějšími atributy této třídy jsou kromě názvu vlastnosti také množina potřebných atributů a metod, které vlastnost potřebuje. Například pokud bychom měli vlastnost Poloha, tak tato vlastnost bude vyžadovat atributy x, y, z (v případě 3D prostoru), které budou právě určovat danou polohu. Vlastnosti stejně jako systémy umožňují jednoduchou dědičnost a znamená to, že daná vlastnost zdědí všechny potřebné atributy a metody definované v její rodičovské vlastnosti. Například vlastnost Pohyb může být potomkem vlastnosti Poloha (v prostoru). Hlavními metodami této třídy jsou metody pro přidávání, nebo odebírání potřebných atributů a metod vlastnosti a také metoda `compareTo(...)` sloužící k porovnávaní vlastností nebo testování zda jsou vlastnosti stejné. V této abstraktní třídě je testována shoda vlastností stejně jako u systémů na základě jejich jména.

7.3.7 Třída `SimpleProperty`

Potomek třídy `AbstractProperty`, který neupravuje její funkcionalitu, ale umožňuje vytváření instancí. Tato třída je používána v rámci aplikace pro modelování systémů.

7.3.8 Třída `ComplexProperty`

Druhý potomek třídy `AbstractProperty`, který je připraven pro složitější implementaci vlastnosti, ale v aktuální podobě datového modelu a aplikace pro modelování systémů není nijak využit.

7.3.9 Výčtový typ `ValueType`

Tento výčtový typ představuje různé fyzikální typy hodnot, kterých mohou nabývat atributy vlastností, vstupní parametry metod, nebo návratové hodnoty metod. Jednotlivé fyzikální typy jsou například: metry, kilogramy, metry za sekundu a další. Kromě klasických typů obsahuje výčtový typ hodnotu `UNSPECIFIED`, která představuje nespecifikovaný typ, použitelný, pokud nám není při modelování/návrhu známo jakých hodnot bude daná komponenta vlastnosti nabývat a také hodnotu `NO_TYPE`, která vyjadřuje, že daný atribut nepoužívá žádný fyzikální typ. Příkladem může být systém „Dopravní prostředek“, který bude mít vlastnost „Kapacita“, která bude definovat atribut počet osob, tak hodnoty tohoto atributu budou pouze čísla udávající počet, která nebudou mít žádné fyzikální typy.

7.3.10 Třída `Attribute`

První ze dvou komponent vlastností představuje potřebný atribut. Tato třída obsahuje dva atributy, a to jsou název atributu a typ jeho hodnot. Testování shodnosti dvou atributů probíhá na základě jejich jména.

7.3.11 Třída `Method`

Druhá komponenta vlastností představuje potřebnou metodu. Kromě atributu název metody, dále obsahuje typ návratové hodnoty metody, a hlavně seznam vstupních parametrů metody. Metody této třídy slouží k přidávání a odebírání parametrů ze seznamu vstupních parametrů metody s požadavkem na možnost přidávání (odebírání) na začátek, na konec, nebo na určité konkrétní místo v seznamu parametrů.

7.3.12 Třída `Parameter`

Tato třída představuje vstupní parametr metody. Každý vstupní parametr má název a typ jeho hodnot. Tato třída je téměř identická s třídou `Attribute`, ale z důvodu přehlednosti a případné budoucí úpravy jsem vytvořil tyto dvě třídy zvlášť.

7.3.13 Pomocné třídy

V předchozí části byly popsány všechny hlavní komponenty navrženého datového modelu a zbývá už jen několik pomocných tříd. Třída `UtilConstants` jak z jejího názvu vyplývá definuje několik pomocných statických konstant, které jsou využívány různými třídami datového modelu a pomáhají se vyhnout problému tzv. *magic numbers* nebo také *magic constants*. Další třídou je třída `SystemXmlReaderWriter`, která umožňuje uložení modelovaného systému do XML souboru a opačně také načtení systému z XML souboru. Poslední pomocnou třídou je třída `XmlReaderWriterUtil`, která definuje řetězcové konstanty pro jednotlivé XML tagy používané v XML souboru pro uložení modelovaného systému.

Součástí projektu datového modelu je také množství tzv. jednotkových testů využívajících knihovnu JUnit 4, které slouží k testování správnosti implementace tříd datového modelu.

Datový model jsem tedy vytvářel hlavně samostatně v první etapě, ale některé další úpravy jsem v něm dělal také při práci na prostředí pro modelování systémů, které tento model využívá, a objevil jsem při jeho vývoji několik nedostatků, nebo chyb v datovém modelu.

7.4 Aplikace pro modelování systémů

Ve druhé etapě jsem vytvářel samotnou aplikaci pro modelování systémů a tato aplikace bude popsána v následujících kapitolách.

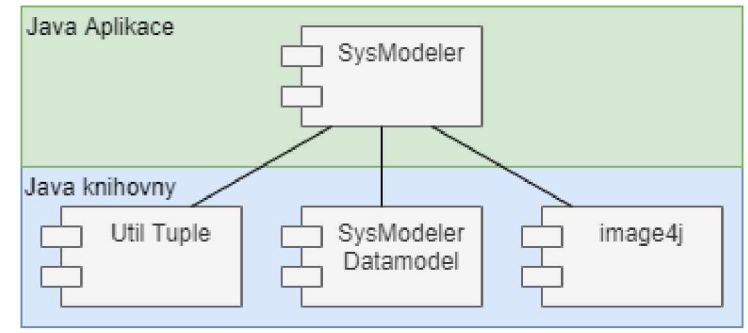
7.4.1 Architektura a součásti aplikace

Pro samotnou aplikaci k modelování systémů jsem vytvořil samostatný projekt. Aplikace využívá 3 následující Java knihovny (soubory, respektive archivy typu `.jar`):

- `cz-ptnavratil-sysmodeler-datamodel` – tato knihovna obsahuje kompletní datový model navržený a implementovaný v první etapě;

- `cz-ptnavratil-util-tuple` – moje vlastní knihovna, která obsahuje tzv. entice (*tuple*), které slouží jako jednoduché, po inicializaci neměnitelné kontejnery pro uchovávání 2-8 objektů různých typů, které samotný jazyk Java 8 neposkytuje;
- `image4j` – knihovna, která je zdarma a s otevřeným zdrojovým kódem a umožňuje číst ikony (obrázky ve formátu `.ico`), které v aplikaci používám, protože Java aplikace jsou bez ikon špatně rozpoznatelné na hlavním panelu operačního systému, nebo ve správci aplikací.

Kromě těchto tří knihoven nepotřebuje aplikace nic jiného než běhové prostředí jazyka Java 8. Na následující obrázku je znázorněna popsaná architektura aplikace.



Obrázek 6 - architektura aplikace a použité knihovny, zdroj: vlastní

Pro návrh a tvorbu grafického uživatelského rozhraní jsem použil dříve popsaný Scene Builder. Aplikace je tvořena jednou scénou, která je definována v souboru `MainScene.fxml`. Tato scéna je stejná po celou dobu běhu aplikace a ovládání aplikace je realizováno pomocí ovládacích prvků na této scéně nebo s využitím dialogových oken. Zdrojový kód aplikace je rozdělen do několika Java balíčků, které budou stručně popsány v následující části.

7.4.2 Hlavní balíček aplikace

Třída `SysModelerApplication` obsahuje vše, co je potřeba pro spuštění jakékoliv JavaFX aplikace. Je potomkem třídy `Application`, která představuje JavaFX aplikaci, dále definuje metodu `main(String[] args)`, která obsahuje volání `launch(args);` a také definuje metodu `start(Stage primaryStage)`. Metoda `main` obecně představuje vstupní bod každé Java aplikace. Metoda `start` slouží k definici toho, co se má provést po startu JavaFX aplikace. V tomto konkrétním případě tato metoda provádí načtení hlavní scény

z FXML souboru, inicializaci primárního jeviště pomocí hlavní scény, načtení ikon aplikace, potom pár dalších věcí, a nakonec zobrazení primárního jeviště. Volání metody `launch` provádí spuštění JavaFX aplikace a v rámci této metody je volána námi definovaná metoda `start`.

Druhou třídou obsaženou v hlavním balíčku je `MainSceneController` a jak její název napovídá je to tzv. kontrolér hlavní scény. Tato třída obsahuje deklaraci všech ovládacích prvků grafického uživatelského rozhraní hlavní scény, které jsou při spuštění aplikace načteny z daného FXML souboru a také definici všech metod, které představují obslužné rutiny událostí, které mohou nastat při používání daných ovládacích prvků. Typickou událostí, která může nastat je kliknutí myší na nějaké tlačítko.

7.4.3 Balíček `dialog`

Tento balíček obsahuje kontroléry dialogových oken pro úpravu definovaných systémů a vlastností.

Pro práci se systémy slouží třída `SystemDialogController`, která se stará o dialog definovaný v souboru `SystemDialog.fxml`. Tento dialog umožňuje práci se systémem na základě datového modelu definovaného v první fázi vývoje aplikace. Umožňuje nastavit jméno systému, případně jeho textový popis a možnost nastavení rodičovského systému, ze kterého tento systém bude dědit vlastnosti. Jestliže je vybrán a nastaven rodičovský systém, v dialogu se zobrazuje seznam zděděných vlastností. Samotné přiřazování vlastností se v tomto dialogu neprovádí. Dialog slouží jednak k vytvoření nového systému nebo také k úpravě již existujícího systému.

Pro práci s vlastnostmi slouží třída `PropertyDialogController`, která se stará o dialog definovaný v souboru `PropertyDialog.fxml`. Na rozdíl od dialogu pro systém, je tento dialog komplexnější a umožňuje kompletní správu vlastnosti na základě definovaného datového modelu. Dialog je rozdělen do tří částí. V první části se nastavuje jméno vlastnosti a je zde tabulka obsahující definované potřebné atributy a formulář pro definovaní nových atributů. Ve druhé části dialogu se nachází tabulka s definovanými potřebnými metodami a zase formulář pro definování nových metod. V poslední části dialogu se nachází tabulka se vstupními parametry vybrané metody a formulář pro úpravu těchto parametrů. Následující obrázek ukazuje dialog pro úpravu vlastnosti.

Právě popsané třídy umožňují práci s dříve definovaným datovým modelem, kromě přiřazování jednotlivých vlastností k systémům. Přiřazování vlastností se provádí s využitím vizuálního nástroje, který bude popsán dále.

7.4.4 Balíček `graphics`

V tomto balíčku se nachází většina kódu, který tvoří vizuální nástroj pro modelování. Tento vizuální nástroj zobrazuje jednotlivé systémy a vlastnosti jako obdélníky s jejich názvy a přiřazení jednotlivých vlastností k systémům, případně dědičnost mezi jednotlivými systémy zobrazuje pomocí čar spojujících tyto obdélníky. Jednotlivé obdélníky pak obsahují referenci na konkrétní systémy a vlastnosti z datového modelu.

Základ tvoří abstraktní třída `RectangularIcon`, která představuje obdélník s nějakým textem, barvou, šířkou, výškou, pozicí vůči svému rodičovskému kontejneru a dalšími různými parametry. Tato třída je potomkem JavaFX třídy `Group` a má atributy standartních (grafických) JavaFX typů jako jsou Rectangle nebo Text a nejdůležitější vlastností je to že jelikož je potomkem `Group`, tak ve chvíli, kdy je někde v okně aplikace vykreslena `Group`, tak jsou vykresleny všechny její dceřiné elementy a tímto způsobem lze v JavaFX aplikacích vytvářet libovolné komplexní grafické (vizuální) obrazce a tvary, které se pak velice snadno vykreslují na určité místo v aplikaci. Součástí této třídy je atribut typu `ConnectionAnchor`, který obsahuje dvě souřadnice provázané se středem obdélníku a do tohoto místa se uchycují jednotlivá spojení mezi různými obdélníky. Tato třída má dva přímé potomky, a to jsou třídy `SystemIcon` a `PropertyIcon`.

Třída `SystemIcon` představuje grafickou reprezentaci systému. Jedná se o obdélník s názvem daného systému. Tato třída obsahuje obslužné rutiny pro různé události, např. kliknutí myší na obdélník. Pokud je třeba v aplikaci zvolen nástroj výběru, tak dvojitým kliknutím levým tlačítkem myši na obdélník se systémem dojde k vyvolání dialogového okna pro úpravu daného sytému.

Obdobně jako třída `SystemIcon` funguje třída `PropertyIcon`, jen s tím rozdílem, že tato třída reprezentuje vlastnost. Zase se jedná o obdélník, tentokrát s názvem dané vlastnosti a třída zase obsahuje hlavně obslužné rutiny různých událostí.

Abstraktní třída `SimpleIconsConnection` představuje spojení mezi různými obdélníky. Stejně jako třída `RectangularIcon` je tato třída potomkem třídy `Group` a obsahuje atribut

JavaFX typu `Line`, který slouží ke kreslení čar. Začátek a konec této čáry spojující dané obdélníky je vždy provázán s dříve zmíněným atributem typu `ConnectionAnchor` u jednotlivých obdélníků a tím je dosaženo, že pohybu jednotlivých obdélníků se automaticky pohybuje i jejich spojení. Tato třída má dva potomky, kteří představují jednotlivé typy spojení.

Třída `SystemToPropertyConnection` představuje spojení systému a vlastnosti. Ve chvíli, kdy je v aplikaci zvolen nástroj pro vytváření spojení se při kliknutí na obdélník systému a následně na obdélník vlastnosti vytvoří spojení mezi nimi. Při vytváření spojení se pomocí reference na daný systém u toho systému zavolá metoda `addProperty()` a jako parametr se použije vlastnost získaná referencí z obdélníku, na který uživatel klikl.

Spojení `SystemToSystemConnection`, které představuje spojení mezi dvěma systémy, tj. spojení mezi potomkem a rodičem se vytváří automaticky, když v dialogu pro úpravu systému nastavíme jeho rodičovský systém.

Třída `PropertyToPropertyConnection` představuje spojení mezi dvěma vlastnostmi, které jsou propojené dědičností. Toto spojení se vytváří automaticky, když v dialogu pro úpravu vlastnosti nastavíme její rodičovskou vlastnost.

V rámci grafického znázornění jsou jednotlivé typy spojení reprezentovány čárami s různou barvou.

7.4.5 Balíček `util`

V tomto balíčku se nachází různý pomocný kód, nebo kód, který se nehodil do jiného balíčku. Výčtový typ `ToolType` představuje různé typy nástrojů, které vizuální nástroj nabízí. Jedná se o nástroje pro výběr, pohyb, mazání, přidávání systémů nebo vlastností, upravování systémů nebo vlastností, nebo přidání spojení.

Nejdůležitější třídou v balíčku `util` je třída `VisualModelerContextHolder`, která udržuje všechny informace a data potřebná při modelování. Tato třída udržuje seznamy všech grafických obdélníků systémů vlastností a spojení mezi nimi. Dále udržuje informaci o právě zvoleném nástroji, o právě vybraném obdélníku systému, nebo vlastnosti a k tomu obsahuje potřebné metody. Všechny třídy z balíčku `graphics` nějakým způsobem pracují s touto třídou. Například když je vytvořeno nové spojení, tak musí být přidáno do seznamu všech existujících spojení, který uchovává tato třída.

Dále se v tomto balíčku nachází výčtový typ `DialogState`, který je využíván společně s dialogovými okny v rámci aplikace. Tento výčtový typ pomáhá rozlišovat jednotlivé stavy ve kterých se právě jednotlivé dialogy nachází. Při vytváření kontroléru dialogu (v jeho konstruktoru) je atribut tohoto typu nastaven na hodnotu `UNSPECIFIED`. Ve chvíli, kdy je v konstruktoru zavolána metoda `initialize()`, která je automaticky volána při načtení FXML souboru s definicí dialogu, tak je hodnota tohoto atributu nastavena na `INITIALIZED`. Ve chvíli, když se má dialog zobrazit uživateli aplikace tak je atribut nastaven na hodnotu `OPEN`. Ve chvíli, kdy je dialog zobrazen tak může být uživatelem buď potvrzen, pak se nastaví hodnota `CONFIRMED`, nebo může být zrušen a v tom případě se nastaví hodnota `CANCELLED`. Zejména dvě poslední zmíněné hodnoty jsou používané k rozhodnutí, co se má provádět potom co uživatel zavřel dialog. Následující obrázek zobrazuje část zdrojového kódu pro vytvoření vizuální ikony vlastnosti po tom, co uživatel potvrdil dialogové okno.

```
if (propertyDialogState == DialogState.CONFIRMED & createdProperty != null) {
    double startX = mouseX - (RectangularIcon.INITIAL_WIDTH / 2.0);
    double startY = mouseY - (RectangularIcon.INITIAL_HEIGHT / 2.0);
    PropertyIcon propertyIcon = new PropertyIcon(startX, startY,  outReference: null, contextHolder);
    propertyIcon.setData(createdProperty);
    contextHolder.getPropertyIconList().add(propertyIcon);
    paneDrawingArea.getChildren().add(propertyIcon);
    writeMessage(MessageType.OK,  message: "Property '" + createdProperty.getName() + "' was added.");
```

Zdrojový kód 2 - vytvoření vizuální ikony vlastnosti, pokud uživatel potvrdil dialog

Součástí balíčku `util` je také výčtový typ `MessageType`, který definuje jednotlivé typy zpráv, které se zobrazují v informační liště ve spodní části aplikace při práci s aplikací a provádění různých úkonů, jako je například přiřazení vlastnosti k systému. Tento výčtový typ definuje pro každý typ zprávy barvu, takže jednotlivé typy zpráv se zobrazují různou barvou.

Třídy `ApplicationInfo` a `JavaLibrary`, jsou pomocné třídy, které jsou využívané v informačním dialogu „O aplikaci“ a uchovávají informaci o verzi aplikace, datu sestavení a použitých knihovnách.

7.4.6 Balíček `serialization`

Toto je poslední balíček této aplikace. Obsahuje vše potřebné pro serializaci rozpracovaného modelu, tj. uložení v binární podobě na disk počítače a tím zachování stavu aplikace. Jelikož třídy z balíčku `graphics` obsahují různé atributy s datovými typy z knihovny JavaFX, které neumožňují serializaci, tak v tomto balíčku se nacházejí určitým způsobem upravené třídy, které serializaci umožňují. Konkrétně jsou to tyto třídy:

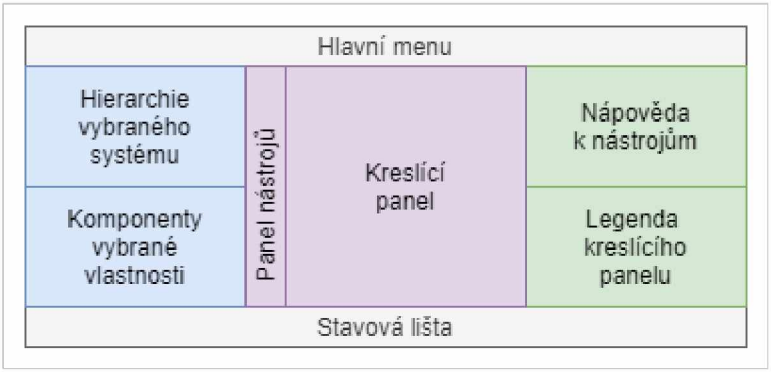
- `SerializablePropertyIcon` – obdélník (ikona) vlastnosti;
- `SerializableSystemIcon` – obdélník (ikona) systému;
- `SerializableSysToPropConnection` – spojení mezi systémem a vlastností;
- `SerializableSysToSysConnection` – spojení mezi dvěma systémy;
- `SerializablePropToPropConnection` – spojení mezi dvěma vlastnostmi.

Potom je zde také třída `SerializationManager`, která provádí samotnou serializaci a její opak, tj. deserializaci. Tato třída implementuje návrhový vzor *Singleton* (jedináček), takže její instance může být v programu jenom jednou. Třída obsahuje čtyři důležité metody. Metoda `prepare()` slouží k přípravě na serializaci. Provádí transformaci jednotlivých objektů tříd z balíčku `graphics`, do jejich upravených variant v balíčku `serialization`. Následující metoda `serialize()` využívá připravené objekty a provádí jejich samotnou serializaci do požadovaného souboru na disku počítače. Pro potřeby deserializace slouží další dvě metody. Metoda `deserialize()` provádí deserializaci z požadovaného souboru na disku a vytváří objekty tříd z balíčku `serialization`. Poslední metodou je metoda `setUpContext()`, která provádí transformaci objektů tříd z balíčku `serialization` na objekty tříd z balíčku `graphics` a ty následně přidá do jednotlivých seznamů, které uchovává požadovaný `VisualModelerContextHolder` a tím je vlastně dokončeno obnovení uloženého stavu aplikace.

7.5 Uživatelské rozhraní aplikace

Aplikace pro modelování má jednoduché uživatelské rozhraní, které má podobnou strukturu, jako uživatelské rozhraní, které známe z jiných běžně používaných aplikací. V horní části aplikace se nachází hlavní menu, které umožňuje například uložit rozpracovaný model, nebo načíst nějaký existující model. V levé části se nachází sloupec, který je rozdělen na dvě části. V horní části se zobrazuje hierarchická struktura právě vybraného systému, tj. jeho vlastnosti včetně jejich rodičovských vlastností a také jeho rodičovský systém. V dolní části tohoto sloupce se zobrazují jednotlivé komponenty právě vybrané vlastnosti, tj. její potřebné atributy a metody. Prostřední a největší část uživatelského rozhraní zabírá panel pro umisťování jednotlivých grafických elementů a v jeho levé části se nachází panel nástrojů. Pravá část vypadá obdobně jako levá – sloupec rozdělený na dvě části. V horní části se nachází jednoduchá textová nápověda pro jednotlivé nástroje a ve spodní části se potom nachází legenda popisující

jednotlivé grafické elementy modelu. Ve spodní části aplikace se potom ještě nachází stavová lišta, která zobrazuje různé informační zprávy v průběhu práce s aplikací. Následující obrázek zobrazuje popsanou strukturu uživatelského rozhraní.



Obrázek 7 - struktura grafického uživatelského rozhraní aplikace, zdroj: vlastní

8 DEMONSTRACE FUNKCÍ APLIKACE

Jedním z cílů diplomové práce bylo demonstrovat funkčnost vytvořené aplikace na vybraném modelu systému. Tato kapitola popisuje postup modelování vybraného systému.

Jako modelový systém (objekt) byl vybrán *automobil*. Když se u automobilu zamyslíme nad jeho rodičovským systémem napadne nás třeba *dopravní prostředek*. Dopravní prostředek může mít vlastnost polohy (v prostoru) a také vlastnost pohybu. Vlastnost pohybu vyžaduje atributy *vx*, *vy*, *vz*, které mohou udávat vektor rychlosti a směru pohybu a jejich hodnoty jsou v metrech za sekundu. Kromě rychlosti může mít tato vlastnost také atributy *ax*, *ay*, *az*, které představují zrychlení a jejich hodnoty jsou v metrech za sekundu na druhou. Jestliže se nějaký objekt nebo systém může pohybovat, tak můžeme logicky usoudit, že by měl mít také vlastnost polohy. V takovéto situaci můžeme využít dědičnosti mezi vlastnostmi a modelovat vlastnost pohybu jako potomka vlastnosti polohy. Tímto způsobem je to vytvořeno v demonstračním modelu. Vlastnost pohybu je tedy přiřazena systému dopravní prostředek, který je rodičem systému auto. Další vlastností, kterou může mít auto je kapacita osob a kapacita nákladu. V demonstračním modelu jsou tedy tyto dvě vlastnosti, kde vlastnost kapacita osob má atribut *počet osob*, který je celým číslem bez jednotky a vlastnost kapacita nákladu má atribut *objem*, který má jako typ metr krychlový. Tyto dvě vlastnosti jsou tedy přiřazeny systému auto. Další vlastností, kterou může mít auto je barva. Barva definuje tři atributy *cervena*, *zelena*, *modra*, které jsou typu celé číslo a představují tři složky barvy v RGB modelu používaném v počítačích. Poslední vlastností v demonstračním modelu je vlastnost rozměry. Tato vlastnost definuje atributy *delka*, *sirka* a *vyska*, které jsou v metrech.

V tomto ukázkovém modelu bylo předvedeno definování systémů a vlastností. Systémy jsou tvořeny množinou vybraných vlastností, které definují atributy a metody. S využitím systémů auto a dopravní prostředek byla demonstrována dědičnost mezi systémy, s využitím vlastností pohyb a poloha zase dědičnost mezi vlastnostmi. Popsaný model je zobrazen v příloze A.

ZÁVĚR

Cílem této diplomové práce bylo vytvořit desktopovou aplikaci umožňující vytváření obecných popisů vlastností a systémů, které tyto vlastnosti implementují.

Nejprve byl vytvořen datový model, který umožňuje popisovat systémy, vlastnosti a přiřazovat jednotlivé vlastnosti k vybraným systémům. Tento model byl vtvořen jako knihovna, která je snadno použitelná v různých aplikacích. Následně byla vytvořena samotná aplikace s grafickým uživatelským rozhraním, využívající vytvořený datový model. Aplikace je vytvořena pomocí jazyka Java 8, knihovny JavaFX a kombinuje v sobě prvky vizuálního modelování a klasických formulářových prvků pro manipulaci s daty. Předností vytvořené aplikace je to, že je multiplatformní a také využívání minimálního počtu dalších knihoven a s tím spojená malá velikost. V teoretické části byly popsány důležité pojmy z oblasti ontologií, byl objasněn rozdíl mezi syntaxí a sémantikou, dále byl popsán sémantický web, objektově orientované programování a modelování a byla popsána myšlenka modelování systémů pomocí vlastností.

Aplikace i datový model splnily všechny požadavky této práce, nicméně v rámci případného dalšího výzkumu a vývoje je možné jejich funkcionalitu a možnosti značně rozšířit. Například by bylo možné vytvořit určitou databázi často používaných vlastností, aby nebylo nutné je v každém modelu definovat znovu.

POUŽITÁ LITERATURA

[1] MERUNKA, Vojtěch. *Objektové modelování*. Praha: Alfa Nakladatelství, 2008. Informatika studium (Alfa Nakladatelství). ISBN 978-80-87197-04-2.

[2] HOEKSTRA, Rinke. *Ontology representation: design patterns and ontologies that make sense*. Fairfax, VA: IOS Press, c2009. Frontiers in artificial intelligence and applications, 197. ISBN 978-1607500131.

[3] HEROUT, Pavel. *Učebnice jazyka Java*. 5., rozš. vyd. České Budějovice: Kopp, 2010. ISBN 978-80-7232-398-2.

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

[5] SVÁTEK, Vojtěch. *Ontologie a WWW* [online]. [cit. 2020-05-04]. Dostupné z: https://nb.vse.cz/~svatek/onto-www.pdf. Vysoká škola ekonomická, Fakulta informatiky a statistiky, Katedra informačního a znalostního inženýrství.

[6] Ontologie_(informatika). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-05-04]. Dostupné z: https://cs.wikipedia.org/wiki/Ontologie_(informatika)

[7] Ontologie. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-05-04]. Dostupné z: https://cs.wikipedia.org/wiki/Ontologie

[8] Unified Modeling Language. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-05-04]. Dostupné z: https://en.wikipedia.org/wiki/Unified_Modeling_Language

[9] Ontologické jazyky. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-05-04]. Dostupné z: https://cs.wikipedia.org/wiki/Ontologické_jazyky

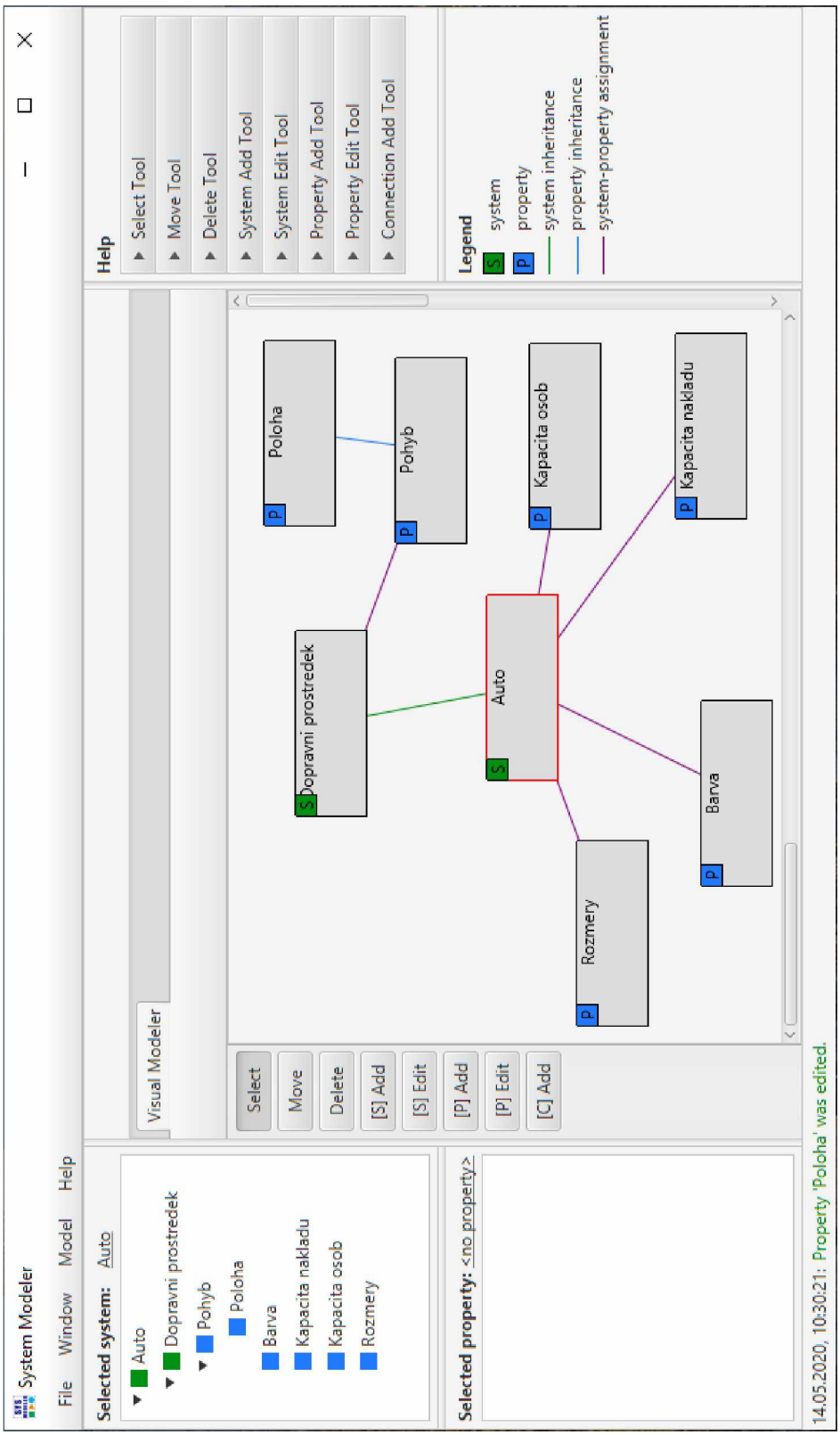
[10] Multiphysics. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-05-17]. Dostupné z: https://en.wikipedia.org/wiki/Multiphysics

PŘÍLOHY

Příloha A – Demonstrační model vytvořený v aplikaci 47
Příloha B – Ukázka dialogových oken aplikace 48
Příloha C – Struktura projektů a zdrojových kódů 49

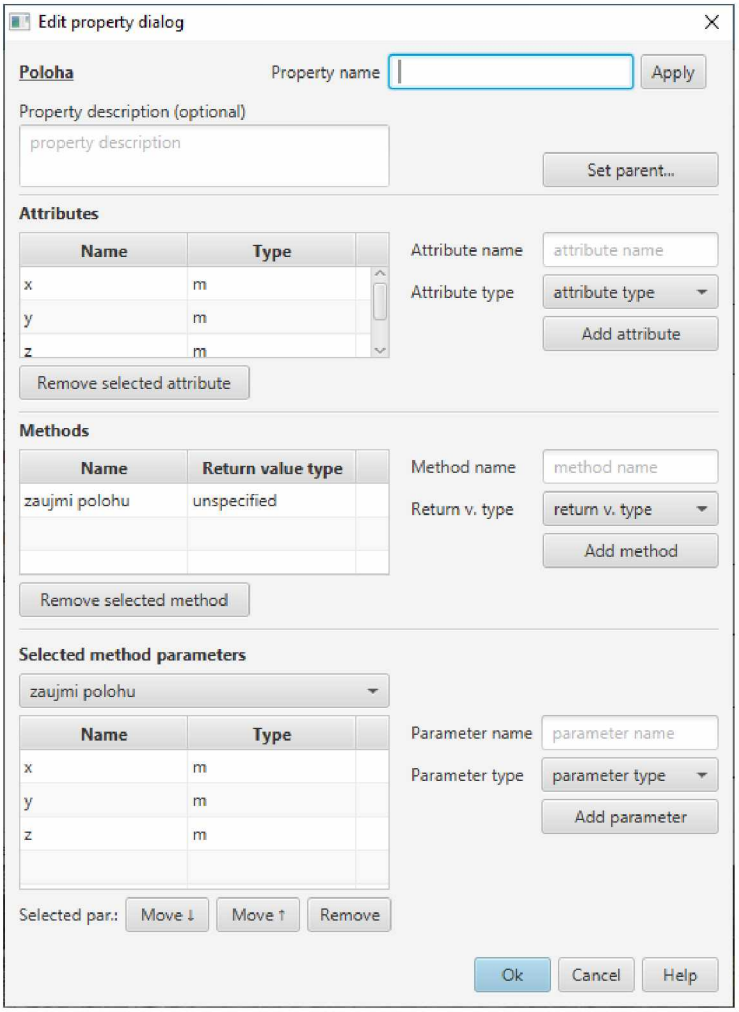
PŘÍLOHA A – DEMONSTRAČNÍ MODEL VYTVOŘENÝ V APLIKACI

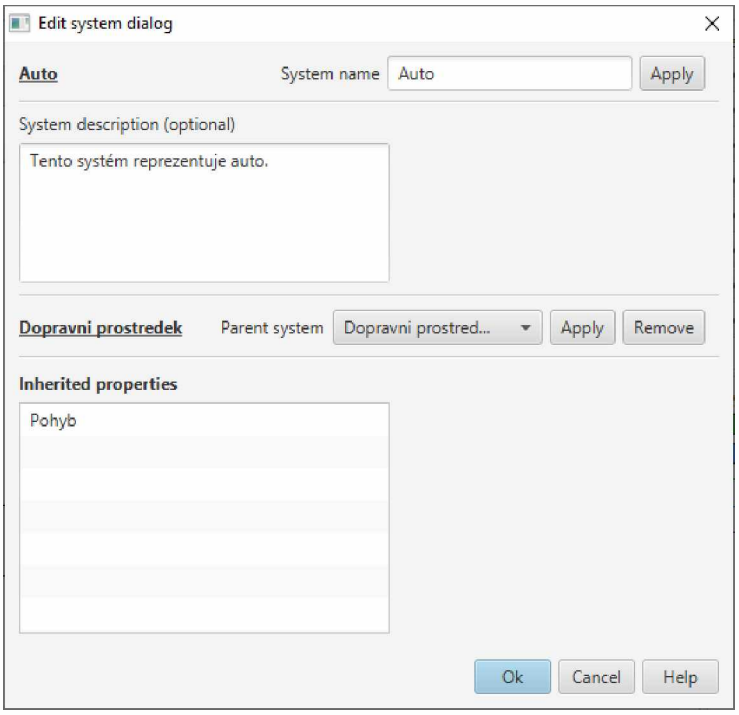
Následující obrázek zobrazuje hlavní okno aplikace společně s vytvořeným demonstračním modelem.



PŘÍLOHA B – UKÁZKA DIALOGOVÝCH OKEN APLIKACE

Následující obrázky ukazují některá dialogová okna aplikace.





PŘÍLOHA C – STRUKTURA PROJEKTŮ A ZDROJOVÝCH KÓDŮ

Následující obrázky zachycují balíčky a třídy projektů datového modelu a aplikace pro modelování.