VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

GENEROVÁNÍ SEKVENČNÍCH DIAGRAMŮ Z MODELŮ PETRIHO SÍTÍ

SEQUENCE DIAGRAM GENERATION FROM OBJECT-ORIENTED PETRI NETS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ADAM BEŇO**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK KOČÍ, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148291

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Beňo Adam**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Generování sekvenčních diagramů z modelů Petriho sítí**
Kategorie:	Softwarové inženýrství
Akademický rok:	2022/23

Zadání:

1. Prostudujte koncept formalismu Objektově orientovaných Petriho sítí (OOPN) a sekvenčních diagramů z jazyka UML. Seznamte se s existující implementací editoru a simulátoru OOPN.
2. Navrhněte mechanismus transformace scénářů modelů popsaných formalismem OOPN do sekvenčních diagramů.
3. Implementujte modul do editoru OOPN pro generování sekvenčních diagramů. Modul musí umožnit mapování aktivit sekvenčních diagramů zpět do modelů OOPN.
4. Vytvořte sadu testovacích příkladů.
5. Analyzujte možné problémy a omezení navrženého řešení. Pro vybrané problémy specifikujte jejich podstatu, důsledky a možná řešení.

Literatura:

- V. Janoušek: Modelování objektů Petriho sítěmi. Disertační práce. VUT v Brně, 1998.

Při obhajobě semestrální části projektu je požadováno:
První dva body zadání.

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

Vedoucí práce:	**Kočí Radek, Ing., Ph.D.**
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	3.11.2022

Abstrakt

Práca sa zaoberá problémom generovania sekvenčných diagramov zo scenárov modelov Objektovo orientovaných Petriho sietí (OOPN). Cieľom je implementovať modul do existujúceho editoru OOPN, ktorý bude umožňovať zobrazovať sekvenčné diagramy z jednotlivých simulačných behov a bude prepojený so simulovaným modelom. Užívateľ tak získa celistvejší pohľad o priebehu simulácie a dokáže lepšie detekovať chyby v modeloch OOPN.

Abstract

The thesis focuses on the issue of sequence diagram generation from scenarios modelled by object-oriented Petri nets (OOPN). It aims on module implementation into existing OOPN editor which enables showing sequence diagrams from individual simulations and is connected with simulated model. The user thus gets complete overview of the simulation process and is able to detect errors in OOPN models.

Kľúčové slová

Petriho siete, Objektovo orientované Petriho siete, OOPN, PNtalk, sekvenčné diagramy, UML, Java, JavaFX, MVC

Keywords

Petri nets, Object-oriented Petri nets, OOPN, PNtalk, sequence diagrams, UML, Java, JavaFX, MVC

Citácia

BEŇO, Adam. *Generování sekvenčních diagramů z modelů Petriho sítí.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Kočí, Ph.D.

Generování sekvenčních diagramů z modelů Petriho sítí

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Radka Kočího, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Adam Beňo
9. mája 2023

Poďakovanie

Rád by som poďakoval svojmu vedúcemu práce za zhovievavý a odborný prístup počas celej doby riešenia a za čas, ktorý venoval spätnej väzbe pri jej dokončievaní. Taktiež by som rád poďakoval svojej rodine a priateľke za podporu počas celej doby štúdia a pri písaní tejto práce.

Obsah

Kapitola 1

Úvod

Objektovo orientované Petriho siete sú formalizmus, ktorý si za cieľ určil prepojiť obecne známe a používané Petriho siete s princípmi objektovej orientácie. Objektová orientácia odstraňuje problém so štruktúrovaním bežných Petriho sietí, ktoré pri zložitejších problémoch značne narastajú a stávajú sa neprehľadné.

Objektovo orientované Petriho siete umožňujú modelovať systémy na väčšej úrovni abstrakcie. Jednotlivé objekty v OOPN komunikujú prostredníctvom volania metód. Tak vzniká interakcia medzi modelovanými objektami. Tieto modely je možné následne simulovať a overiť tak ich správnosť. Keďže pomocou Petriho sietí modelujeme systémy, ktoré sú diskrétne a paralelné, rôzne simulačné behy môžu prebiehať rozličným spôsobom. K tomu, aby sme dokázali zachytiť komunikáciu objektov a vzájomné volanie metód nám slúžia sekvenčné diagramy, ktoré zobrazujú interakciu medzi objektami.

Na Fakulte informačných technológií v Brne bola vytvorená konkrétna implementácia objektovo orientovaných Petriho sietí s názvom PNtalk. Spolu s tým je vyvíjaný grafický editor, umožňujúci vytvárať modely v jazyku PNtalk a simulačný server schopný tieto modely simulovať a krokovať.

Cieľom práce je vytvoriť modul do existujúceho editoru, ktorý na základe priebehu simulácie modelov vytvorených v tomto editore a simulovaných na simulačnom servery postupne generuje sekvenčné diagramy. Tie sú prepojené s modelmi OOPN vytvorenými v editore.

Kapitola 2 popisuje modelovacie formalizmy, na ktorých táto práca zakladá. Obsahuje úvod do problematiky Objektovo orientovaných Petriho sietí a popisuje jazyk PNtalk ako konkrétnu implementáciu tohto formalizmu. Taktiež obsahuje popis sekvenčných diagramov a ich grafickej notácie v jazyku UML. Kapitola 3 sa zameriava na praktickú časť práce. Popisuje použité nástroje a požiadavky na aplikáciu spolu s princípom generovania sekvenčných diagramov. Taktiež je v nej uvedený návrh architektúry. Kapitola 4 obsahuje implementačné detaily výslednej aplikácie. V kapitole 5 sú zobrazené konkrétne výsledky vo forme vhodne zvolených príkladov demonštrujúcich funkčnosť, ale aj obmedzenia výsledného riešenia a zoznam chýb editoru, ktoré sa pri práci s ním vyskytli.

Kapitola 2

Modelovacie formalizmy

V tejto kapitole budú predstavené dva modelovacie formalizmy, na ktorých táto práca zakladá. Prvým sú Objektovo orientované Petriho siete a druhým sekvenčné diagramy.

2.1 Objektovo orientované Petriho siete

Aby bolo možné pochopiť Objektovo orientované Petriho siete (OOPN), je nutné priblížiť základné princípy objektovej orientácie a Petriho sietí. Následne bude predstavený jazyk PNtalk ako konkrétna implementácia OOPN.

2.1.1 Objektová orientácia

Objektová orientácia vychádza z prirodzeného chápania sveta ako súboru navzájom interagujúcich objektov. Táto prirodzenosť je hlavným dôvodom, prečo sa objektová orientácia stala veľmi obľúbenou nielen v programovaní, ale je využívaná aj pri modelovaní, návrhu alebo v databázových systémoch a ďalších.

Každý objekt má svoje vlastnosti a schopnosti. Vlastnosti objektov sa nazývajú atribúty a uchovávajú stav objektu. Schopnosti sa nazývajú metódy a predstavujú množinu operácií, ktorým objekt rozumie. Objekty je možné na základe podobnosti ich vlastností a schopností združovať do skupín nazývaných triedy. Triedy slúžia ako obecný predpis pre vytváranie konkrétnych objektov (inštancií).

Takou triedou môže byť napríklad auto. Auto má svoju farbu a značku. To sú jeho vlastnosti. V triede nie je definované akej konkrétnej farby a značky auto je, iba hovorí, že objekt vytvorený na základe triedy auto bude mať nejakú farbu a značku. Podľa úrovne abstrakcie by autá mohli zaradené do tried ako motorové vozidlá alebo dopravné prostriedky.

Dedičnosť, ako dôležitý princíp objektovej orientácie, umožňuje triedam dediť atribúty a metódy iných tried. Spoločné vlastnosti môžu byť združené do rodičovskej triedy, z ktorej potom ostatné triedy dedia a bližšie sa špecifikujú. Takto vzniká hierarchia dedičnosti. Čím vyššie je trieda v tejto hierarchii, tým je obecnejšia.

Dôležitou vlastnosťou objektovej orientácie je zapuzdrenie. Zapuzdrenie umožňuje skryť informácie o vnútornej štruktúre a sprístupniť iba vybrané operácie, skrz ktoré prebieha interakcia s objektom. Takto je zabezpečená integrita a bezpečnosť stavu objektu.

Objekty rôznych tried môžu na rovnakú správu odpovedať rozdielnym správaním. Táto vlastnosť je nazvaná polymorfizmus. Dôležité je to najmä pri hierarchii dedičnosti, kde podtriedy dedia metódy rodičovskej triedy a v niektorých prípadoch musia byť schopné odpovedať rozdielnym spôsobom. [2, 6]

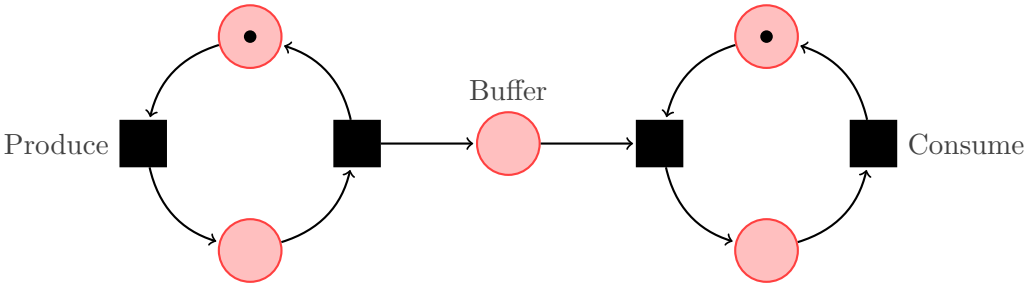
2.1.2 Petriho siete

Petriho siete sú matematický modelovací jazyk pre popis diskrétnych systémov. [11] Ich tvorca, nemecký matematik Carl Adam Petri, ich detailne popísal vo svojej dizertačnej práci *Kommunikation mit Automaten* v roku 1962.

Grafická reprezentácia Petriho siete je bipartitný orientovaný graf. Obsahuje štyri druhy objektov: miesta, prechody, orientované hrany a značky. Orientované hrany spájajú miesta s prechodmi a naopak. Nikdy však nespájajú miesto s miestom alebo prechod s prechodom. Miesta sú reprezentované krúžkom, prechody štvoruholníkom. Značky sú zakreslené ako bodky v miestach a vyjadrujú okamžitý stav systému. Prítomnosť značky v mieste vyjadruje, že daný stav systému je aktívny. Každý prechod má definovanú množinu vstupných a výstupných miest. Aby bol prechod vykonateľný, všetky vstupné miesta musia obsahovať značku. Tá je pri vykonaní prechodu odobraná a zapísaná do všetkých výstupných miest. [10] Na obrázku 2.1 je uvedený príklad jednoduchej Petriho siete. Zobrazuje problém producenta a konzumenta s neobmedzeným zásobníkom. Producent (na ľavej strane) produkuje značky a ukladá ich do zásobníka (miesto s názvom *Buffer*). Konzument (na pravej strane) tieto značky zo zásobníka odoberá a konzumuje.

Najjednoduchšie Petriho siete, nazývané C/E siete (Condition/Event Nets), povoľujú, aby každé miesto obsahovalo najviac jednu značku. Vyjadrovacia sila takéhoto modelu je rovná sile konečných automatov. Komplexnejšie P/T siete (Place/Transition Nets) umožňujú do každého miesta vložiť viac značiek a priradiť hranám ohodnotenie. Takto je zaistená možnosť presúvať väčšie množstvo značiek. P/T siete môžu byť rozšírené o kapacitu miest, priority prechodov, testovacie a inhibičné hrany. Každé rozšírenie zvyšuje vyjadrovaciu silu Petriho sietí. Ďalšie typy Petriho sietí, ktoré vznikli rozšírením P/T sietí [4]:

- **Časované Petriho siete** – Zavádzajú pojem času a umožňujú modelovať deje trvajúce nejaký časový úsek. Trvanie dejov môže byť deterministické (konštantný časový úsek), stochastické (náhodný časový úsek) alebo môže kombinovať obe (niektoré prechody trvajú konštantný čas zatiaľ čo iné náhodný).
- **Farebné Petriho siete** – Umožňujú modelovať rôzne typy značiek, ktoré môžu nadobúdať rozličné hodnoty (farby). Rozširujú Petriho siete o prvky ako premenné, deklarácia typov, stráže či akcie prechodov.
- **Hierarchické Petriho siete** – Zavádzajú možnosť hierarchického štruktúrovania.
- **Objektové Petriho siete** – Spájajú výhody objektovej orientácie s Petriho sieťami.



Obr. 2.1: Príklad jednoduchej Petriho siete.

2.1.3 Jazyk PNtalk

V tejto časti bude predstavený jazyk PNtalk [3], založený na Objektovo orientovaných Petriho sieťach (OOPN). Základ vychádza z definície OOPN a je inšpirovaný jazykom Smalltalk. Konkretizuje niektoré skutočnosti, ktoré zostali v definícii otvorené. Prináša tiež syntaktické vylepšenia zľahčujúce zápis. Názov je odvodený od *"Petri Nets & Smalltalk"*.

Programovanie v jazyku PNtalk je postavené na vytváraní tried neprimitívnych objektov a ich zaraďovaní do hierarchie dedičnosti. Triedy sú definované množinami Petriho sietí, skladajúce sa z miest, prechodov a hrán.

Termy

Termy sú základné a najjednoduchšie výrazy a jazykovo reprezentujú objekty PNtalku. Medzi termy patria:

1. **Literály** – Identifikujú významné primitívne objekty:
   - **Čísla** – Objekty reprezentujúce číselné hodnoty. Reagujú na správy požadujúce výsledky matematických operácií. Zapisujú sa ako sekvencia číslic, ktorá môže obsahovať znamienko mínus, desatinnú bodku alebo písmenko e predstavujúce notáciu s exponentom. Napr. `5, -25, 23.456, -0.02, 1.34e2`.
   - **Znaky** – Objekty reprezentujúce jednotlivé symboly abecedy. Sú uvedené znakom `$`. Napr. `$1, $+, $$`.
   - **Reťazce** – Objekty reprezentujúce sekvenciu znakov. Reagujú na správy požadujúce prístup k jednotlivým znakom a porovnanie s inými reťazcami. Zapisujú sa v apostrofoch. Apostrof vnútri reťazca musí byť zdvojený. Napr. `'hi', 'mustn"t'`.
   - **Symboly** – Objekty používané typicky ako mená. Rovnaké symboly reprezentujú ten istý objekt. Zapisujú sa so znakom `#` na začiatku. Napr. `#e, #z34`.
   - **Booleovské konštanty** – Vyhradené identifikátory `true` a `false`.
   - **Nedefinovaný objekt** – Vyhradený identifikátor `nill`.

   Okrem vyššie zmienených pripúšťa implementácia PNtalku aj literály pole a blok z dôvodu zaistenia konzistencie s jazykom Smalltalk.
2. **Premenné** – Ich hodnotu možno programovo ovplyvniť a tak môžu v priebehu výpočtu reprezentovať rôzne objekty. Identifikátory sa uvádzajú s malým počiatočným písmenom. Napr. `res, x`.
3. **Pseudopremenné** – Existujú dve, `self` (odkaz na samotný objekt) a `super` (odkaz na rodičovskú triedu). Ich hodnota nie je programovo ovplyvniteľná a závisí od okamžitého stavu výpočtu.
4. **Mená tried** – Jedná sa o konštanty, ktoré sa v priebehu výpočtu nemenia. Identifikátory sa uvádzajú s veľkým počiatočným písmenom. Napr. `PN, Participant`.

Zasielanie správ

Zaslanie správy je výraz, ktorý sa môže objaviť ako súčasť stráže a akcie prechodu. Syntax zaslania správy je *<adresát> <správa>*. Správa pozostáva zo selektorov a prípadných argumentov. Podľa typu selektoru rozlišujeme následovné typy správ:

1. **Unárna správa** – Správa bez argumentov. Selektor správy je identifikátor bez dvojbodky, napr. `Participant new, 2.8 rounded`.

2. **Binárna správa** – Správa s jedným argumentom. Selektor správy je jeden zo symbolov: `+` (sčítanie), `-` (odčítanie), `/` (delenie), `*` (násobenie), `=` (rovnosť), `~=` (nerovnosť), `<` (menšie než), `>` (väčšie než), `<=` (menšie alebo rovné), `>=` (väčšie alebo rovné), `&` (logický súčin), `|` (logický súčet), `//` (celočíselné delenie), `\\` (zvyšok po celočíselnom delení), `,` (konkatenácia), `==` (rovnosť identity), `~~` (nerovnosť identity).

3. **Správa s kľúčovými slovami** – Správa s jedným alebo viac kľúčmi. Kľúč je identifikátor zakončený dvojbodkou s vlastným argumentom. Napr. „`calc add: 5 with: 2`“, kde `calc` je adresát správy, `add:with:` sú selektory a `5` a `2` sú argumenty.

V správach, kde sú adresát správy aj argumenty termy, ide o jednoduché zaslanie správy. Pokiaľ je príjemca správy alebo argument opäť zaslanie správy, ide o zložené zaslanie správy. Zložené správy sú vyhodnocované postupne zľava doprava. Poradie ich vyhodnocovania možno ovplyvniť zátvorkami.

Siete

Siete v jazyku PNtalk sú popísané graficky. Existuje aj textový popis pre potreby strojového spracovania. Siete sa skladajú z miest, prechodov a hrán.

Miesto siete je reprezentované kružnicou alebo elipsou. Každé miesto má svoje meno a môže mať počiatočné značenie. Počiatočné značenie má formu hranového výrazu a môže obsahovať premenné, ktoré musia byť vyčíslené v počiatočnej akcii miesta. Počiatočná akcia je syntakticky totožná s akciou prechodu.

Prechod je reprezentovaný štvoruholníkom. Každý prechod má svoje meno a môže mať špecifikovanú stráž a akciu. Stráž prechodu je tvorená sekvenciou výrazov oddelených čiarkou, z ktorých každý výraz je zaslanie správy. Tieto výrazy sú vo vzájomnej konjunkcii, čiže stáž je platná, pokiaľ sú platné všetky čiastkové výrazy. Akcia, narozdiel od stráže môže obsahovať výraz priradenia v tvare ⟨premenná⟩ := ⟨zaslanie správy⟩, napr. `res := x * y`. Stráž prechodu je podčiarknutá.

Hrana spája miesto s prechodom. Ku každej hrane je pripojený hranový výraz. Hranový výraz reprezentuje multimnožinu v tvare n_1‘c_1, n_2‘$c_2, ..., n_m$‘c_m, kde n_i je term a c_i je term alebo zoznam. Zoznam má tvar $e_1, e_2, ..., e_m$, kde e_i je opäť term alebo zoznam. Príkladom hranového výrazu je „`5’7, x’3, y’(z, $e)`“. Tento výraz reprezentuje multimnožinu obsahujúcu päť výskytov čísla `7`, `x` výskytov čísla 3 a `y` výskytov dvojice `(z, $e)`. Rozlišujeme tri typy hrán:

- **Vstupná hrana** – Smeruje z miesta do prechodu. Udáva značky odobrané prechodom z príslušného miesta (vstupná podmienka).

- **Výstupná hrana** – Smeruje z prechodu do miesta. Udáva značky umiestnené prechodom do príslušného miesta (výstupná podmienka).

- **Testovacia hrana** – Obojstranná hrana. Testuje prítomnosť značiek v príslušnom prechode (podmienka prechodu).

Rozsah platnosti mien miest a prechodov je obmedzený na sieť ich výskytu. Toto pravidlo neplatí pre miesta siete objektu, ktoré sú viditeľné vo všetkých sieťach metód objektu.

Rozlišujeme dva typy sietí:

- **Sieť objektu** – V podobe miest reprezentuje atribúty a definuje implicitnú aktivitu objektu. Každá trieda má práve jednu takúto sieť.
- **Sieť metódy** – Špecifikuje reakciu objektu na prichádzajúcu správu. Každá sieť metódy obsahuje vzor správy, ktorej obdržaním objekt vyvolá dynamické vytvorenie odpovedajúcej inštancie siete. Vzor správy sa skladá zo selektoru a parametrov. V rámci siete metódy potom existuje podmnožina miest reprezentujúcich parametrové miesta, ktorých názvy odpovedajú formálnym parametrom vo vzore správy. Takto je zaistené predávanie parametrov metódam. Každá sieť metódy obsahuje jedno výstupné miesto s názvom `return`, pomocou ktorého je navrátený výsledok vyhodnotenia správy. Prechody sietí metód môžu byť pomocou hrán prepojené s miestami siete objektu, a tak ku nim pristupovať a meniť ich dáta. Každá trieda môže mať definovaných viac sietí metód.

Konštruktory

Objekty v jazyku PNtalk sú vytvárané zaslaním správy `new` príslušnej triede. Tejto správe rozumie každá trieda a v reakciu na ňu vytvorí inštanciu triedy inicializovanú počiatočným značením siete objektu. Jedná sa o implicitný konštruktor zabudovaný priamo do jazyka.

PNtalk umožňuje aj dodatočnú alebo prípadne parametrizovanú inicializáciu objektov pomocou špeciálnych metód nazvané konštruktory. Sú podobné ostatným metódam objektu, avšak v ich špecifikácii sa nachádza kľúčové slovo `constructor`. Týmto správam rozumie trieda aj jej inštancie. Trieda najprv vytvorí novú inštanciu pomocou správy `new` a následne zašle novovzniknutému objektu rovnakú správu. Objekty reagujú rovnakým spôsobom ako pri volaní ostatných metód. Konštruktor vždy vracia hodnotu `self`.

Synchrónne porty

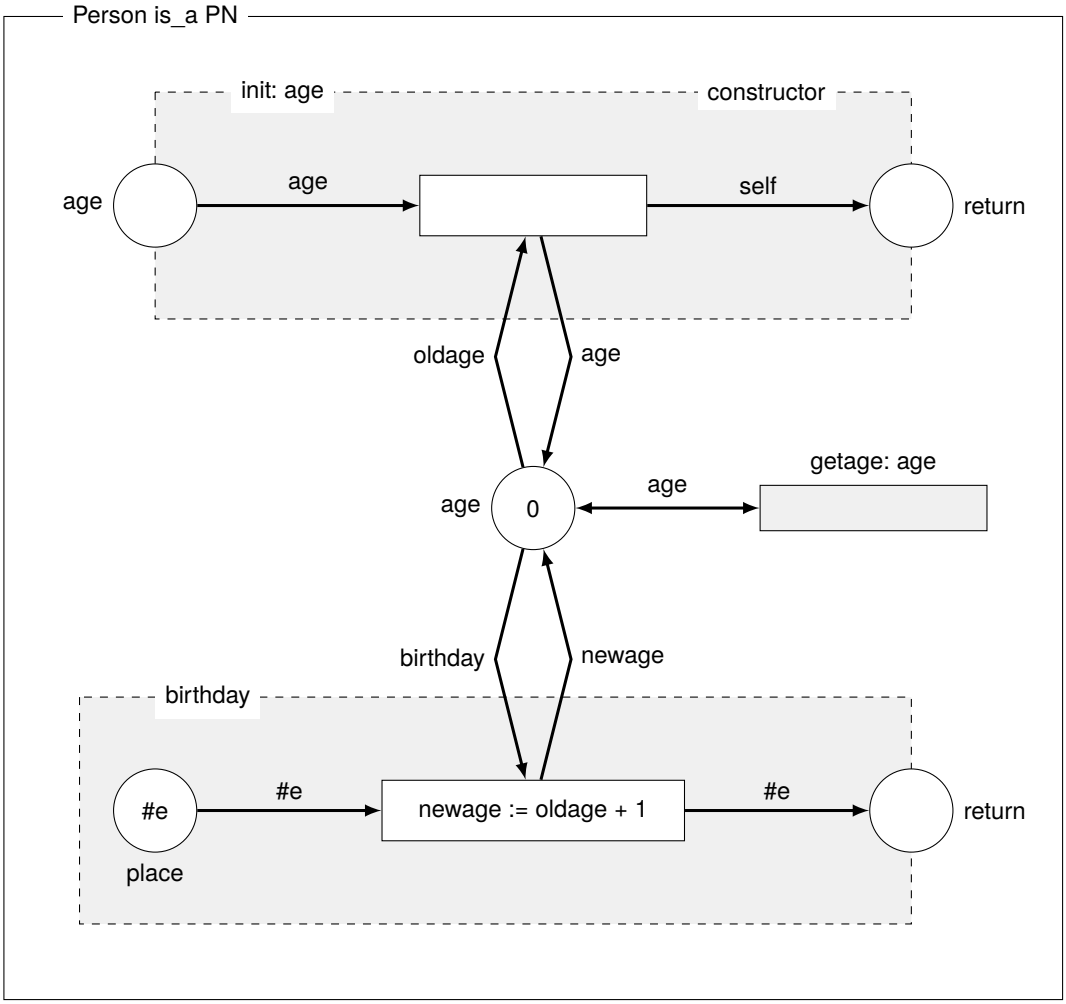
Pomocou synchrónnych portov je možné testovať a meniť stav objektu. Podobne ako prechody, môžu byť prepojené hranami s miestami siete objektu a môžu obsahovať stráž. Obsahujú vzor správy, na ktorú reagujú. Synchrónne porty je možné volať zaslaním správy zo stráže ľubovoľného prechodu. Môžu byť volané s voľnými alebo dopredu vyhodnotený parametrami a sú buď vykonateľné alebo nevykonateľné pre určité naviazanie premenných.

Synchrónny port, ktorý je prepojený s miestami siete objektu len pomocou testovacích hrán, je nazývaný predikát, pretože neovplyvňuje jeho stav. Grafická reprezentácia synchrónneho portu je štvoruholník a stráž nemusí byť podčiarknutá.

Triedy

Model v PNtalku sa skladá z množiny tried, z ktorých jedna je označená ako počiatočná trieda. Trieda je zložená zo siete objektu, množiny sietí metód a konštruktorov a zo synchrónnych portov.

Jazyk PNtalk podporuje jednoduchú dedičnosť. To znamená, že každá trieda má práve jedného priameho predchodcu. Na vrchole hierarchie dedičnosti je trieda `Object`. Jej priamymi potomkami sú triedy primitívnych objektov, ktoré sú najčastejšie dostupné v podobe liberálov, a trieda `PN`, ktorá je vrcholom hierarchie pre všetky triedy definované Petriho sieťami. Trieda `PN` nie je popísaná Petriho sieťami, jej objektová sieť je prázdna. Trieda dedí od svojho predchodcu štruktúru siete objektu, metód, konštruktorov a synchrónne porty.



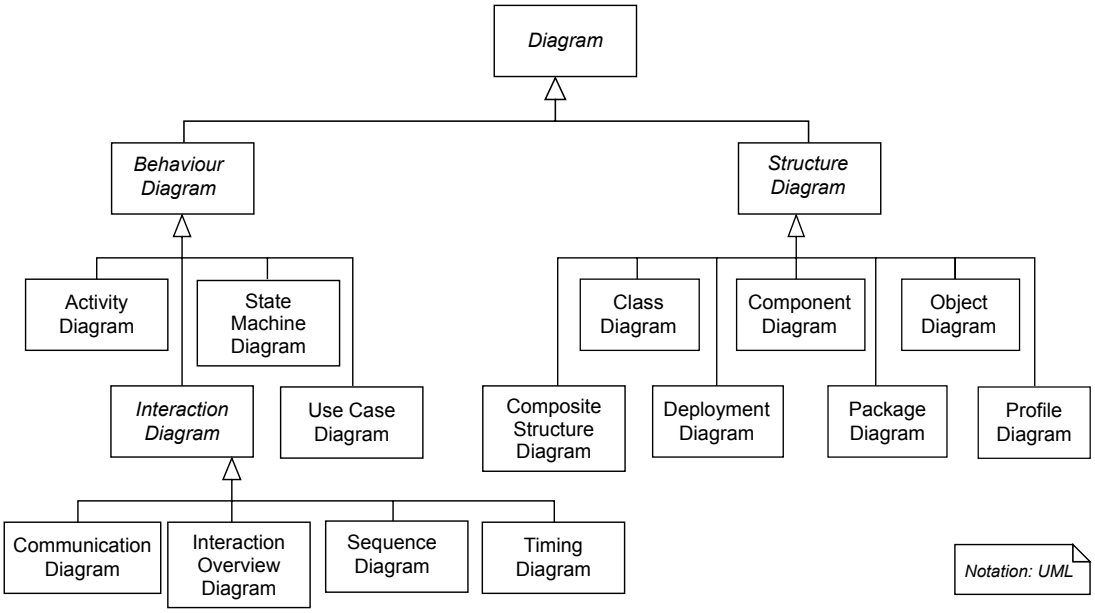
Obr. 2.2: Príklad triedy OOPN.

Príklad triedy v jazyku PNtalk je uvedený na obrázku 2.2. Zobrazuje triedu `Person`, ktorá dedí priamo od triedy `PN`. Jej objektová sieť obsahuje jedno miesto s názvom `age`. Definovaný je synchrónny port `getage: age`, ktorý vracia stav miesta objektovej siete `age`. Je k nemu pripojený pomocou testovacej hrany. Jedná sa teda o synchrónny port nazvaný predikát, pretože nemení stav objektu. Trieda obsahuje konštruktor nazvaný `init: age`. V sieti konštruktoru je vidieť parametrové miesto `age` a návratové miesto `return`. Konštruktor triedy vždy vracia hodnotu `self`. V rámci triedy je tiež definovaná metóda `birthday`, ktorá neprijíma žiadne parametre a jej úlohou je zvýšiť hodnotu miesta siete objektu s názvom `age` o jedna.

2.2 Sekvenčné diagramy

Sekvenčné diagramy sa radia k diagramom interakcie z modelovacieho jazyka UML (Unified Modeling Language). Slúžia k modelovaniu komunikácie medzi objektami. Zobrazujú presnú postupnosť správ, ktoré si objekty medzi sebou vymieňajú.

2.2.1 Jazyk UML



Obr. 2.3: Rozdelenie diagramov jazyka UML. Prevzaté z [12].

UML je modelovací jazyk, ktorý bol vytvorený so zámerom poskytnúť jednotný štandard pre vizualizáciu procesov analýzy, dizajnu a implementácie softvérových systémov. Jeho využitie sa však neobmedzuje len na oblasť informačných technológií a takisto dobre je využiteľný v bankovníctve, finančníctve, zdravotníctve a mnohých ďalších odvetviach.

UML definuje množstvo rôznych diagramov, ktoré sú rozdelené na 3 základné typy: diagramy štruktúry, správania a interakcie. Diagramy štruktúry reprezentujú statické prvky a najčastejšie sú využívané k modelovaniu architektúry výsledného systému. Diagramy správania reprezentujú dynamické prvky a sú využívané k zobrazeniu funkcionality. Diagramy interakcie, ktoré sú podmnožinou diagramov správania, popisujú kontrolný a dátový tok. [12] Celkové rozdelenie diagramov je zobrazené na obrázku 2.3.

2.2.2 Základné prvky

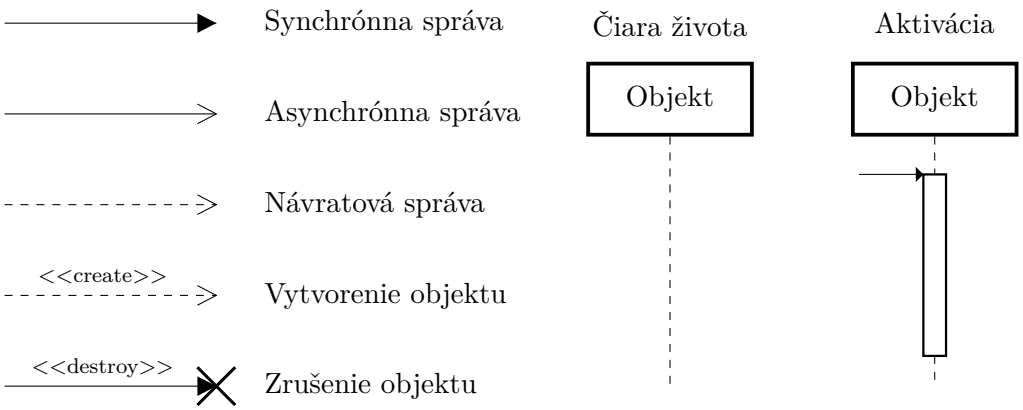
Prvky zobrazované v horizontálne rovine predstavujú objekty systému zúčastňujúce sa interakcie. Vertikálna rovina predstavuje čas, v ktorom táto interakcia prebieha. Nejde pritom o zobrazenie presných časových úsekov interakcií, ale ich sekvenčnú postupnosť. [13] Medzi základné prvky sekvenčných diagramov patria:

- **Čiara života** – Predstavuje objekt modelovaného systému s jeho časovou líniou. Je reprezentovaná obdĺžnikom, v ktorom môže byť informácia o identifikácii objektu a z ktorého vo vertikálnom smere vychádza čiara predstavujúca jeho dobu života. Táto čiara môže, ale nemusí byť prerušovaná.
- **Správy** – Modelujú komunikáciu medzi objektami. Rozlišujeme viacero typov správ:
  - **Synchrónna správa** – Zaslanie správy s tým, že odosielateľ musí čakať na odpoveď.

  - **Asynchrónna správa** – Zaslanie správy s tým, že odosielateľ nemusí čakať na odpoveď a môže pokračovať v ďalšej činnosti.
  - **Návratová správa** – Je spojená s volaním synchrónnej správy a signalizuje jej ukončenie.
  - **Vytvorenie objektu** – Správa je zaslaná čiare života a predstavuje začiatok jej doby života.
  - **Zrušenie objektu** – Správa je zaslaná čiare života a predstavuje koniec jej doby života. Typicky je zakončená krížikom.

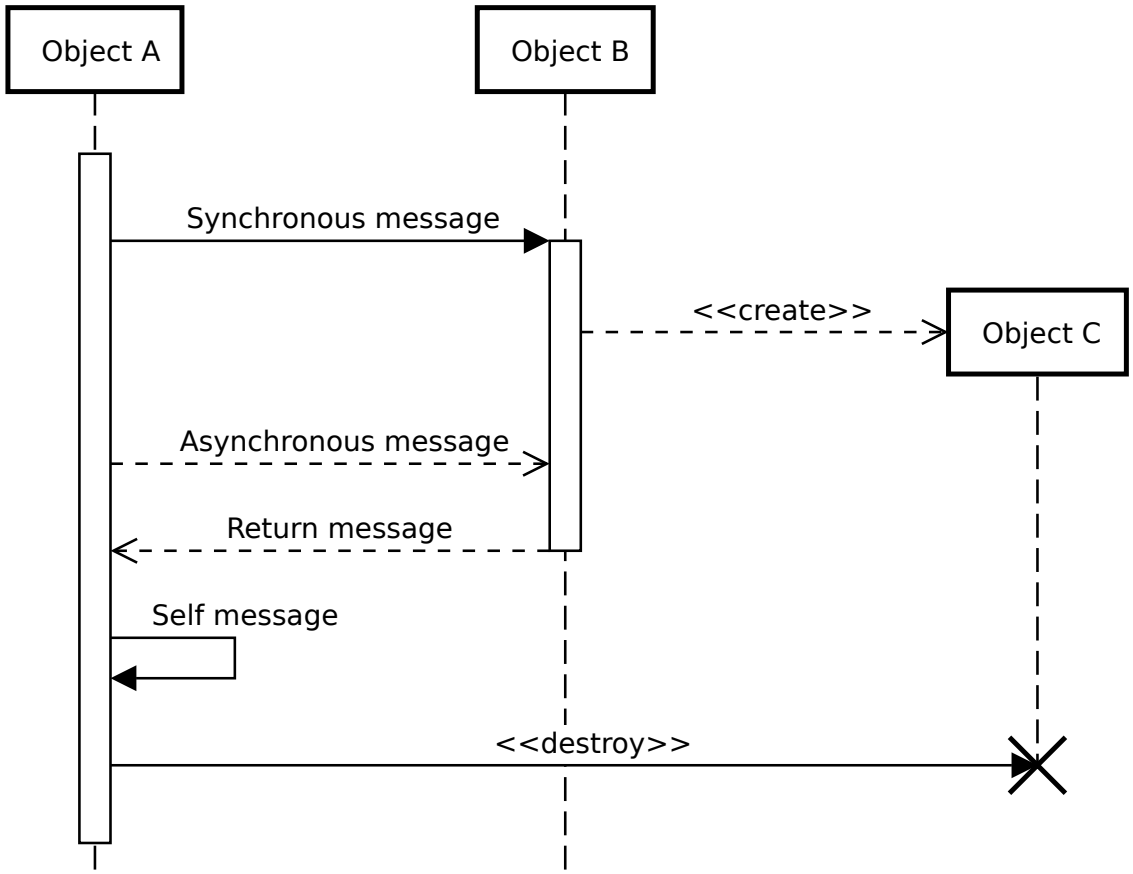
- **Aktivácia** – Predstavuje dobu, po ktorú je objekt aktívny. Je reprezentovaná obdĺžnikom nakresleným ponad časť čiary života.

Grafická notácia vyššie spomenutých prvkov v jazyku UML je znázornená na obrázku 2.4. Obrázok 2.5 následne zobrazuje príklad sekvenčného diagram s popísanými prvkami. Ide o demonštračný príklad, ktorého cieľom je zobraziť jednotlivé prvky v diagrame.



Obr. 2.4: Grafická notácia základných prvkov sekvenčných diagramov.

Medzi ďalšie komponenty využívané hlavne v komplexnejších sekvenčných diagramoch patria stráže, predstavujúce podmienky, a kombinované fragmenty. Pomocou kombinovaných fragmentov možno vyjadriť napríklad alternatívny výber a voliteľné, cyklické alebo paralelné správanie. [1]



Obr. 2.5: Príklad jednoduchého sekvenčného diagramu s popísanými prvkami.

Kapitola 3

Analýza požiadavkov a návrh aplikácie

V tejto kapitole budú ako prvé predstavené n8stroje použité v tejto práci. Následne budú analyzované požiadavky na výslednú aplikáciu a popísaný návrh architektúry.

3.1 Použité nástroje

Práca nadväzuje na grafický editor modelov OOPN a simulačný server schopný tieto modely simulovať. Popísaná bude základná práca s editorom bez implementačných detailov a náležitosti potrebné k práci so simulačným serverom.

3.1.1 Editor OOPN

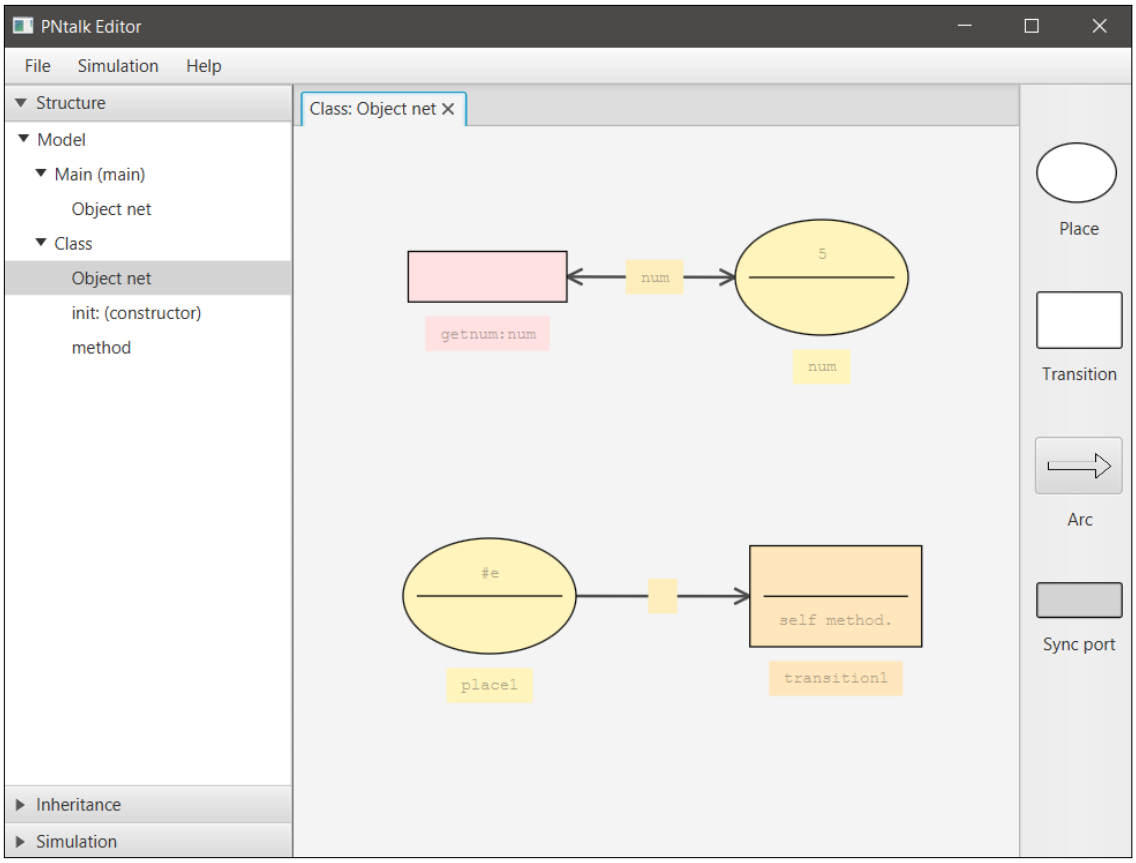
Grafický editor modelov jazyka PNtalk vytvoril a popísal vo svojej diplomovej práci Ing. Daniel Švub. [14] Editor je implementovaný v jazyku Java a pre užívateľské rozhranie je použitá platforma JavaFX.

Grafické rozhranie

Grafické rozhranie aplikácie nasleduje typický dizajn používaný v aplikáciách s užívateľským prostredím. Je rozdelené do štyroch častí:

- `top` – Predstavuje panel menu s položkami `File`, `Simulation` a `Help`.
- `left` – Obsahuje panel pre štruktúru modelu (`Structure`), hierarchiu dedičnosti modelu (`Inheritance`) a štruktúru simulovaného modelu (`Simulation`).
- `right` – Obsahuje grafické prvky siete, ktoré je možné pridávať do sietí.
- `center` – Priestor pre otváranie a editovanie sietí a zobrazovanie stavu simulovaných sietí.

Tieto časti je možné vidieť na obrázku 3.1. V editore je otvorený ukážkový model a otvorená objektová sieť jednej z tried. Tá obsahuje dve miesta, prechod a synchrónny port.



Obr. 3.1: Grafické užívateľské rozhranie editoru.

Vytváranie a editácia modelov

Nový model je možné vytvoriť kliknutím na menu `File > New`. Po zadaní názvu bude v panely štruktúry v ľavej časti vytvorená nová položka s názvom modelu. Tá predstavuje celý, aktuálne editovaný model. V kontextovom okne tejto položky sú tlačidlá pre pridanie novej triedy `Add class` a premenovanie modelu `Rename`.

Trieda je vytvorená spolu s jej sieťou objektu. Kontextové okno položky s názvom triedy umožňuje pridávať nové metódy (tlačidlo `Add method`) a konštruktory (tlačidlo `Add constructor`). Nachádza sa tu aj tlačidlo `Set as main`, ktoré nastaví danú triedu ako počiatočnú.

Všetky siete možno otvoriť dvojklikom na ich názov v panely štruktúry modelu alebo v ich kontextovom okne kliknutím na tlačidlo `Open`. Siete novo vytvorených metód a konštruktorov automaticky obsahujú návratové miesto s názvom `return` a miesta s názvami parametrov, pokiaľ boli zadané. Tieto miesta nemôžu byť zmazané ani premenované. Kontextové okno položiek s názvami metód a konštruktorov umožňuje do ich sietí pridávať miesta zo siete objektu triedy, ktorej patria (položka okna `Show object places` a výber konkrétneho miesta ktoré má byť pridané).

Do aktuálne otvorenej siete je možné pridávať prvky z panelu na pravej strane rozhrania. Miesto (`Place`), prechod (`Transition`) a synchrónny porty (`Sync port`) je pridaný potiahnutím do siete. Synchrónny port môže byť pridaný len do siete objektu. Hrana (`Arc`) sa pridáva kliknutím na príslušný obrázok a zvolením dvoch prvkov, ktoré spája. Kontextové

okno hrany dovoľuje meniť smer hrany na vstupnú (`Change to input`), výstupnú (`Change to output`) alebo testovaciu (`Change to test`). Taktiež umožňuje vytvoriť zlomový bod (`Create break point`) pre lepšiu grafickú názornosť.

Na ľavej strane aplikácie, v panely s názvom `Inheritance`, je zobrazená hierarchia dedičnosti tried v aktuálne otvorenom modely. Vrcholom je trieda `PN`, ktorá predstavuje vrchol všetkých užívateľsky definovaných tried. Kontextové okno položiek tried v tejto hierarchii umožňuje zmeniť rodičovskú triedu (tlačidlo `Change parent class`).

Ukladanie a načítanie modelov

Modely vytvorené v editore je možné uložiť pre budúcu prácu. Editor má definovaný vlastný formát ukladaných modelov, založený na formáte XML (eXtensible Markup Language). K uloženiu do posledného otvoreného súboru pre zápis slúži tlačidlo menu `File > Save`. Tlačidlo `File > Save as` uloží model do nového súboru. Modely je možné načítať kliknutím na `File > Open`.

Export v textovej podobe PNtalku a vo formáte SVG

Editor umožňuje exportovať aktuálne otvorený model do súboru v jeho textovej podobe kliknutím na položku menu `File > Export to PNtalk`.

Každá otvorená sieť môže byť exportovaná ako obrázok vo formáte SVG (Scalable Vector Graphics) kliknutím na položku `Export net as SVG` v kontextovom menu okna siete.

Simulácia modelov

Modely je možné simulovať priamo z editoru. Adresu a číslo portu, pomocou ktorých sa editor pripája ku simulačnému serveru, je možné zmeniť kliknutím na položky menu `Simulation > Change server address` a `Simulation > Change port number`.

Simulácia je zahájená kliknutím na tlačidlo menu `Simulation > Start simulation`. Pokiaľ model neobsahuje chybu a úspešne je nahraný na server, je zahájená simulácia. V ľavej časti rozhrania, v panely `Simulation`, je vytvorená prvá sieť zobrazujúca stav simulácie. Po spustení je to sieť objektu počiatočnej triedy s jej počiatočným značením. Siete zobrazujúce stav simulácie nemožno editovať ani ukladať. Ďalší krok simulácie je vykonaný kliknutím na `Simulation > Next step` a k ukončeniu simulácie slúži tlačidlo menu `Simulation > Quit simulation`.

3.1.2 Simulačný server

Simulačný server modelov PNtalku implementoval vedúci práce Ing. Radek Kočí, Ph.D. v prostredí Pharo. Pharo je dynamický, objektovo orientovaný jazyk, založený na jazyku Smalltalk, s vlastným integrovaným vývojovým prostredím. [8]

Server má definovaný svoj vlastný komunikačný protokol (popísaný ďalej) a formát výpisu stavu simulácie uvedený v prílohe B. Ich popis je prevzatý z materiálov poskytnutých autorom.

Komunikácia so serverom

Prednastavené číslo portu, na ktorom server načúva, je 9999. Táto hodnota môže byť zmenená. Protokol definuje notáciu zápisu správ, ktorým server rozumie nasledovne:

```
("názov požiadavku" "parameter1" "parameter2" ...) => ("výsledok")
```

Na ľavej strane sa nachádza požiadavok zasielaný na sever, na pravej strane je odpoveď. Prvý riadok odpovede je stav vykonania požiadavku: OK alebo FAILED. V zápise preto nie je uvádzaný. Medzi základné príkazy nevyhnutné k simulácii modelov patria:

- `(addClass "reťazec s definíciou triedy") => ()` – Pridanie triedy do modelu na simulačnom servery. Ako parameter prijíma reťazec s definíciou triedy v textovej podobe jazyka PNtalk.
- `(delClass "názov triedy") => ()` – Odstránenie triedy z modelu na simulačnom servery. Ako parameter prijíma reťazec s názvom triedy, ktorá má byť odstránená.
- `(newSim "názov počiatočnej triedy") => ("id simulácie")` – Inicializácia simulácie. Ako parameter prijíma reťazec s názvom počiatočnej triedy. Navracia id simulácie na simulačnom servery.
- `(stepSim "id simulácie") => ()` – Vykonanie jedného simulačného kroku modelu. Ako parameter prijíma id simulácie, ktorej krok má byť vykonaný.
- `(destroySim "id simulácie") => ()` – Ukončenie simulačného behu. Ako parameter prijíma id simulácie, ktorá má byť ukončená.
- `(getState "id simulácie") => ("reťazec s výpisom stavu")` – Získanie stavu simulácie. Ako parameter prijíma id simulácie, ktorej stav má byť navrátený.

3.2 Analýza požiadavkov

Cieľom práce je navrhnúť a implementovať mechanizmus transformácie scenárov popísaných formalizmom OOPN do sekvenčných diagramov. Výsledkom je modul, dopĺňajúci existujúci editor (popísaný v sekcii 3.1.1) o funkcionalitu generovania sekvenčných diagramov z jednotlivých simulačných behov modelov vytvorených v tomto editore. Modul musí umožniť mapovanie aktivít sekvenčných diagramov späť do modelov OOPN.

3.2.1 Generovanie sekvenčných diagramov

Jeden simulačný beh predstavuje jeden vygenerovaný sekvenčný diagram. OOPN modelujú paralelné systémy s diskrétnym časom, takže simulácia prebieha v krokoch a v jednom kroku sa môže udiať viacero udalostí. Z tohto dôvodu je vhodné v sekvenčných diagramoch zobrazovať k jednotlivým aktivitám číslo kroku simulácie v ktorom prebehli.

Generovanie prebieha na základe stavu simulácie získaného zo simulačného serveru v každom jej kroku. Formát stavu simulácie je popísaný v prílohe B. Tento stav následne podlieha analýze, ktorej výsledkom je aktualizácia generovaného sekvenčného diagramu.

Základnou jednotkou OOPN sú objekty. Po spustení simulácie je vytvorený prvý objekt počiatočnej triedy. Objekty sú v sekvenčných diagramoch reprezentované čiarami života. Nové objekty vznikajú zaslaním správy `new` príslušnej triede z akcie prechodu. To by bolo možné v sekvenčných diagramoch zobraziť pomocou správy vytvorenia objektu, avšak výpis stavu simulácie tieto správy nezaznamenáva. Novo vzniknutý objekt sa však objaví vo výpise a tak možno zaznamenať vznik objektu v konkrétnom simulačnom kroku. Rušenie objektov je v OOPN riešené implicitne, pomocou garbage-collectoru. Z tohto dôvodu sa vo výsledných sekvenčných diagramoch nebudú vyskytovať žiadne správy rušenia objektu.

Interakcia medzi objektami prebieha prostredníctvom vzájomného volania metód z akcií prechodov. Jedná sa o neúplné vykonanie prechodu. Značky zo vstupných miest sú odobrané a je vytvorená nová inštancia siete odpovedajúcej metódy. Prechod ostáva rozpracovaný, pokým nedôjde k ukončeniu metódy, čiže zrušeniu inštancie siete a navráteniu výslednej hodnoty. Z tohto dôvodu je volanie metód reprezentované v sekvenčných diagramoch pomocou synchrónnych správ. Pri každom volaní metódy pribudne vo výpise nová rozpracovaná správa s informáciami o objektoch na oboch stranách správy a prípadných parametroch, s ktorými bola zavolaná. Navrátenie z metódy je možné v sekvenčných diagramoch reprezentovať návratovou správou. Vo výpise je návrat z metódy zaznamenaný zrušením rozpracovanej správy. Zrušenie rozpracovanej správy a inštancie siete metódy z výpisu prebieha už v kroku, kedy je výsledok zapísaný do návratového miesta metódy, a tak z neho nie je možné získať výslednú hodnotu navrátenú z metódy. Špeciálnym prípadom je volanie metódy v rámci toho istého objektu (adresát je `self`). Sekvenčné diagramy obsahujú ku každej správe volania metódy odpovedajúcu návratovú správu.

```
(((8439278 #add:with: 1 2 #domain (#obj->5 #obj->9))) ('Main' 1
(('object' 1 (('place1' ()) ('place2' ()) ('place3' ()))
(('transition2' 7 1 ('o'->#ref->2 #self->#ref->1) ())))))
((8439278 1 'transition2' 7)) ()) ('Calculator' 2 (('object' 1
() ()) ('add:with:' 2 (('x' (1->#obj->5)) ('y' (1->#obj->9))
('return' ())) ())) () (8439278->2)))
```

Výpis 3.1: Príklad stavu simulácie získaného zo serveru.

Teraz bude uvedená krátka analýza stavu simulácie a získanie informácií z nej. Vo výpise 3.1 je zobrazený stav simulácie v kroku, kedy bola volaná metóda objektu iným objektom. Hneď na začiatku, v zozname rozpracovaných správ pribudlo volanie metódy s id `8439278`. Odosielateľom je objekt s id `1` a prijímateľom objekt s id `2`. Metóda má názov `add:with:` a bola volaná s parametrami `5` a `9`. Nasleduje objekt triedy `Main`, ktorého id je `1` a je teda odosielateľom správy. Objekt má invokovanú len svoju sieť objektu. V zozname spustených prechodov objektu možno vidieť, že volanie správy s id `8439278` bolo zrealizované z prechodu s názvom `transition2`, ktorý sa nachádza v sieti s id `1`, ktorú predstavuje objektová sieť. Druhým objektom je objekt triedy `Calculator` s id `2`, ktorý je prijímateľom správy. Ten má okrem svojej siete objektu invokovanú aj sieť volanej metódy `add:with:`. V zozname závislostí tohto objektu možno vidieť, že táto sieť metódy bola vytvorená v reakcii na prichádzajúcu správu. Výsledný sekvenčný diagram teda bude obsahovať dve čiary života reprezentujúce jednotlivé objekty a správu od jednej k druhému. Neskôr v simulácii dôjde k ukončeniu volanej metódy, ktoré bude znázornené návratovou správou.

3.2.2 Mapovanie aktivít sekvenčných diagramov

Dôležitým požiadavkom je prepojenie sekvenčných diagramov s modelmi OOPN. Analýza výpisu stavu simulácie umožňuje nasledujúce tri možnosti spätného mapovania:

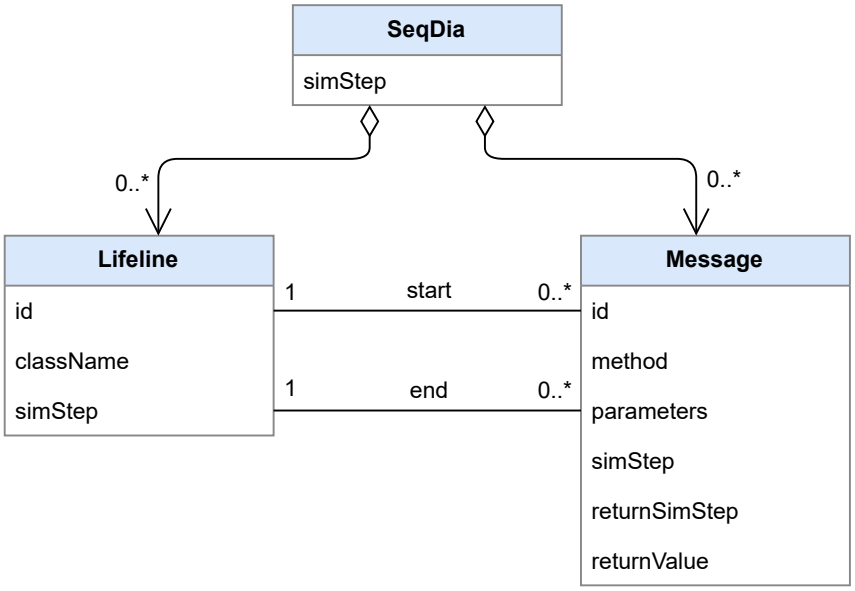
- Prepojenie čiary života so sieťou objektu triedy, z ktorej je objekt reprezentovaný touto čiarou vytvorený. Vo výpise sa pri každom objekte nachádza aj názov príslušnej triedy.
- Prepojenie správy so sieťou metódy, ktorej volanie reprezentuje. V rámci výpisu každá rozpracovaná správa obsahuje názov volanej metódy.

- Prepojenie správy s prechodom, z ktorého bolo volanie metódy realizované. Výpis obsahuje pre každý objekt zoznam správ, ktoré z neho boli zaslané. V ňom možno nájsť názov prechodu a id siete a podľa toho vyhľadať príslušný prechod.

3.3 Architektúra aplikácie

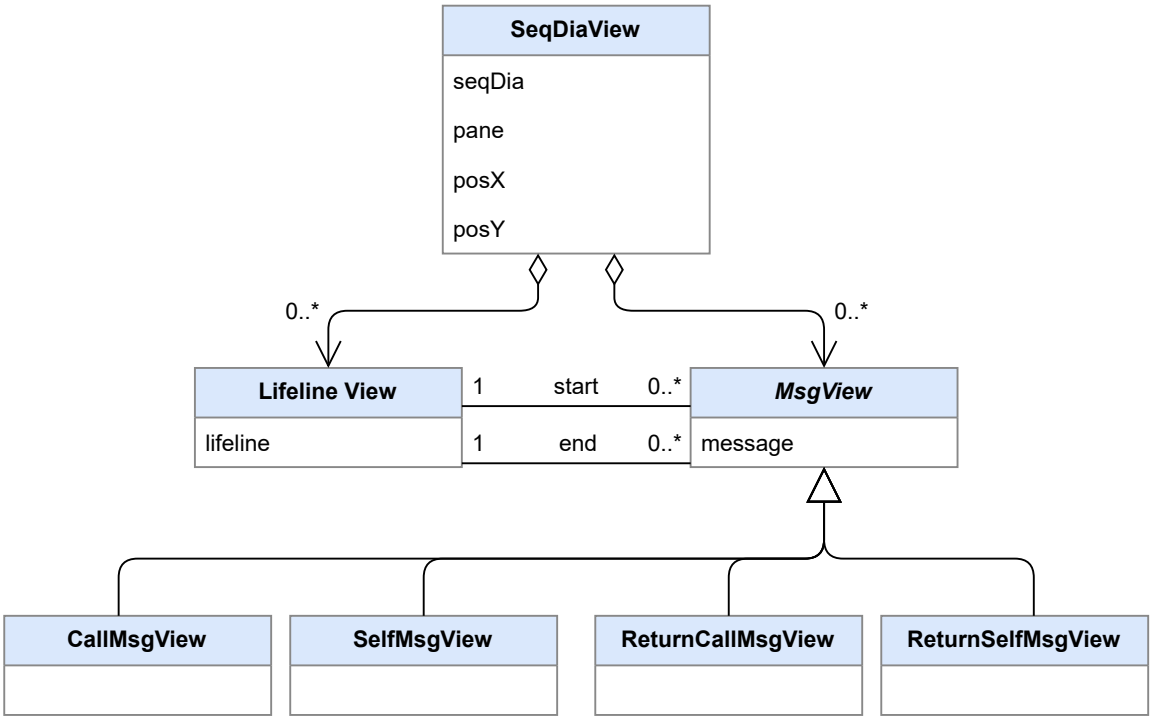
Výsledný modul je navrhnutý ako samostatná jednotka schopná generovať sekvenčné diagramy. Tieto diagramy je potreba zobraziť užívateľovi. Z toho dôvodu je architektúra modulu postavená na návrhovom vzore MVC, ktorý je tradične využívaný v aplikáciách s grafickým užívateľským rozhraním. Ten rozdeľuje aplikáciu do troch častí: Model, View a Controller. [5] Popis architektúry uvedený v tejto sekcii je obecný. Detailný popis jednotlivých tried a metód možno nájsť v implementačnej časti v kapitole 4.

Model predstavujú dáta a ich logika nezávisle od toho, čo vidí užívateľ. Slúži k uchovávaniu informácií o štruktúre a jednotlivých prvkoch generovaných sekvenčných diagramov. Na obrázku 3.2 je zobrazený diagram tried modelu.



Obr. 3.2: Diagram tried sekvenčných diagramov uchovávaných v modeli.

View je časť aplikácie prezentujúca dáta užívateľovi. Obsahuje triedy reprezentujúce grafické prvky sekvenčných diagramov. Každá z nich vlastní referenciu na odpovedajúci prvok modelu. Diagram tried view sa nachádza na obrázku 3.3. Oproti modelu možno vidieť zmenu v triede reprezentujúcej správy sekvenčných diagramov. Tá je rozdelená do niekoľkých tried podľa konkrétneho typu správy ktorý je treba zobraziť.



Obr. 3.3: Diagram tried grafických prvkov sekvenčných diagramov.

Controller slúži k obsluhe aplikácie. Prijíma požiadavky od užívateľa a na ich základe upravuje dáta modelu a zobrazuje príslušné prvky view. Poskytuje funkcionalitu generovania sekvenčných diagramov. Jeho metódy sú volané z metód modulu editoru. Detailný popis komunikácie medzi modulmi, analýza stavu simulácie a riešenie spätného mapovania sú popísané v kapitole 4.

Kapitola 4

Implementácia

Táto kapitola sa zaoberá popisom implementačných detailov výslednej aplikácie. Vytvorený modul poskytuje funkcionalitu generovania sekvenčných diagramov zo simulačných behov modelov OOPN a spätné mapovanie aktivít sekvenčných diagramov do týchto modelov. Architektúra je založená na návrhovom vzore MVC (viď 3.3), ktorého jednotlivé časti budú popísané ďalej. Obsahuje tiež samostatnú komunikáciu so simulačným serverom. Tento modul je následne využitý v grafickom editore modelov OOPN (viď 3.1.1), avšak implementácia sa snaží byť čo najviac nezávislá a je možné ju využiť aj v iných projektoch.

4.1 Jazyk Java a JavaFX

Práca je realizovaná prostredníctvom jazyka Java, v ktorom je implementovaný aj spomínaný editor. Java je objektovo orientovaný programovací jazyk, ktorý vytvoril James Gosling v roku 1995. Jeho syntax vychádza z jazyka C. Java je kompilovaný a interpretovaný jazyk zároveň. Kód je prekladaný do medzistupňa, nazvaného byte code, ktorý je platformovo nezávislý. Ten je následne interpretovaný pomocou Java Virtual Machine. [9]

K tvorbe rozhrania je použitá platforma JavaFX, umožňujúca tvorbu grafických rozhraní v desktopových, mobilných a vstavaných systémoch postavených na jazyku Java. [7] Poskytuje širokú ponuku grafických prvkov a možností prispôsobenia ich vzhľadu.

4.2 Model

Model udržiava dáta o vygenerovaných sekvenčných diagramoch. Nachádza sa v balíčku `cz.vut.fit.sqdia.model`. Obsahuje následujúce triedy:

- `SeqDia` – Reprezentuje celý sekvenčný diagram. Vo forme zoznamov udržiava informácie o čiarach života a správach, ktoré do neho patria. Obsahuje atribút `simStep`, čo je číslo aktuálneho simulačného kroku.
- `Lifeline` – Reprezentuje čiaru života. Obsahuje id objektu na simulačnom servery, názov triedy objektu a simulačný krok v ktorom bola vytvorená. Špeciálnym je atribút `processes`, ktorý slúži k uchovávaniu procesov objektu spolu s ich id na servery v aktuálnom simulačnom kroku. Slúži ako pomocný atribút v rámci analýzy stavu simulácie.
- `Message` – Reprezentuje volanie metódy objektu (správu sekvenčného diagramu). Obsahuje id správy na simulačnom servery, simulačný krok v ktorom bola zaslaná a re-

ferenciu na počiatočnú a koncovú čiaru života. Udržiava tiež názov volanej metódy s prípadnými parametrami a názov prechodu a siete, z ktorého bola správa zaslaná. Metóda `getMethodWithParameters` vráti reťazec zostavený z názvu správy a parametrov. Špeciálnym je atribút `returnSimStep`, ktorý indikuje v ktorom simulačnom kroku bola navrátená výsledná hodnota vyhodnotenia siete metódy. Pre úplnosť obsahuje trieda aj atribút `returnValue`, ktorý avšak nie je použitý z dôvodu nemožnosti získať navrátenú hodnotu popísaného v sekcii 3.2.1.

4.3 View

View tvoria triedy reprezentujúce grafické prvky sekvenčných diagramov. Každá trieda obsahuje referenciu na odpovedajúci prvok modelu ktorý znázorňuje a logiku jeho vykreslenia. Grafická notácia prvkov je obdobná tomu popísanému v kapitole o sekvenčných diagramoch na obrázku 2.4. Tieto triedy sa nachádzajú v balíčku `cz.vut.fit.sqdia.view` a sú to:

- `SeqDiaView` – Udržiava informácie o všetkých grafických prvkoch sekvenčného diagramu a obsahuje logiku jeho vykreslenia. Obsahuje atribúty `posX` a `posY`, udávajúce x-ovú a y-ovú pozíciu novo vykresľovaných prvkov. Logika vykreslenia prebieha po jednotlivých simulačných krokoch. Ako prvé je vždy zobrazené číslo kroku a prípadná čiara oddeľujúca jednotlivé simulačné kroky. Nasledujú všetky čiary života, ktoré v danom kroku vznikli. Potom sú zobrazené správy volania metód a na záver návratové správy.
- `LifelineView` – Grafické znázornenie čiary života. Z dôvodu, že na simulačnom servery objekty počas simulácie nemajú svoj názov, ale id, je v hlavičke čiary života zobrazené práve toto id a názov triedy objektu, navzájom oddelené dvojbodkou.
- `MsgView` – Abstraktná trieda reprezentujúca obecnú správu sekvenčného diagramu. Všetky konkrétne implementácie jednotlivých správ dedia z tejto triedy. Združuje ich obecné atribúty, ktorými sú referencia na správu z modelu a počiatočné a koncové zobrazenie čiary života.
- `CallMsgView` – Grafická reprezentácia volania metódy. Adresát a prijímateľ sú rozdielne čiary života. Nad čiarou správy je text s názvom volanej metódy. Oproti bežným správam v sekvenčných diagramoch je rozšírená o id správy na simulačnom servery, ktoré je zobrazené pod čiarou správy.
- `SelfMsgView` – Obdoba `CallMsgView`, avšak adresát a prijímateľ je tá istá čiara života. To si vyžaduje rozdielne vykreslenie správy.
- `ReturnCallMsg` – Grafická reprezentácia návratovej správy. Návratová správa vždy patrí ku správe volania metódy. Nad čiarou správy je zobrazené id správy ktorej ukončenie signalizuje.
- `ReturnSelfMsg` – Obdoba `ReturnCallMsg`, avšak adresát a prijímateľ je tá istá čiara života.

4.4 Controller

Controller modulu sa stará o funkcionalitu spojenú s generovaním sekvenčných diagramov. Je to trieda `SeqDiaController` v balíčku `cz.vut.fit.sqdia`.

Poskytuje statickú metódu `generateSequenceDiagram`, ktorá má na vstupe tri parametre. Prvým je reťazec s definíciou tried modelu PNtalku zasielaný na simulačný server, druhým je reťazec s názvom počiatočnej triedy a tretím je list reťazcov s názvami všetkých tried. Na základe toho v tejto triede prebehne celá simulácia modelu a je vygenerovaný sekvenčný diagram. Tento diagram je vrátený z metódy a je možné z neho vytvoriť view.

Controller umožňuje tiež postupnú tvorbu sekvenčného diagramu a možnosť zobrazovať ho v každom kroku simulácie. Slúžia k tomu statické atribúty `seqDia` a `seqDiaView`, ktoré obsahujú aktuálne generovaný sekvenčný diagram a pokiaľ je aj vykreslený tak jeho odpovedajúce zobrazenie. S týmito atribútmi narábajú metódy volané v rôznych fázach simulácie:

- `startSimulation` – Volaná po začatí simulácie. Vytvorí nový sekvenčný diagram a vykoná úvodnú analýzu stavu.
- `stepSimulation` – Volaná v každom kroku simulácie. Posunie počítadlo kroku simulácie v sekvenčnom diagrame a vykoná analýzu stavu.
- `quitSimulation` – Volaná po ukončení simulácie. Statické atribúty triedy sú nastavené na hodnotu `null`.
- `createSeqDiaView` – Na vyžiadanie vytvorí zobrazenie sekvenčného diagramu uloženého v atribúte `seqDia` a uloží ho do atribútu `seqDiaView`.
- `removeSeqDiaView` – Odstráni zobrazenie sekvenčného diagramu uložené v atribúte `seqDiaView`.

Je nutné poznamenať, že funkcionalita postupnej tvorby sekvenčného diagramu zakladá na tom, že nahranie modelu na server, inicializácia, krokovanie a ukončenie simulácie prebieha externe. To je vykonané v rámci editoru a vyššie spomínané metódy sú volané z obdobne nazvaných metód v triede `Controller`.

4.5 Komunikácia so simulačným serverom

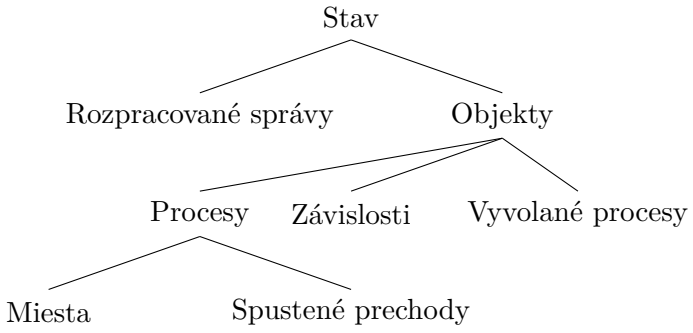
O komunikáciu so simulačným serverom sa stará trieda `SimulationHandler` v balíčku `cz.vut.fit.sqdia.sim`. Implementuje zasielanie požiadavkov na server, popísané v sekcii 3.1.2. Obsahuje statické atribúty `address` a `port`, predstavujúce adresu serveru a číslo portu na ktorom načúva. Názvy statických metód sú totožné s požiadavkami zasielanými na server a prijímajú rovnaké parametre. V balíčku je tiež definovaná vlastná výnimka `SimulationException`, ktorá je vyhodená metódami v prípade, že zo strany serveru je navrátený chybový stav alebo sa k nemu nepodarí pripojiť.

Jedná sa o plne nezávislú implementáciu od zvyšku aplikácie. Tento balíček by mohol byť vyňatý do samostatného modulu pre obecné použitie. Podobná trieda s rovnakým názvom je implementovaná aj v rámci editoru, avšak tá je priamo závislá od implementačných detailov modelov OOPN. V budúcom vývoji by bolo vhodné tento prístup zjednotiť práve pomocou spomínaného obecného modulu, ktorý bude riešiť komunikáciu so serverom.

4.6 Analýza stavu simulácie

Analýza stavu simulácie je implementovaná statickou metódou `parseState` v rámci triedy `SeqDiaController`. Prijíma dva parametre. Prvým je sekvenčný diagram, do ktorého budú

pridané nové prvky, a druhým je stav simulácie. Reťazec na vstupe je postupne rozobraný na menšie časti, medzi ktorými sa cyklí až pokým nie je celý spracovaný. Hierarchické rozčlenenie stavu simulácie na logické časti sa nachádza na obrázku 4.1. V rámci analýzy sú nepotrebné časti výpisu vynechané. Princíp získania nových prvkov je popísaný v sekcii 3.2.1.



Obr. 4.1: Hierarchická štruktúra častí výpisu stavu simulácie.

4.7 Mapovanie aktivít sekvenčných diagramov

Keďže modul generujúci sekvenčné diagramy nemá priamy prístup k modelom Objektovo orientovaných Petriho sietí, bolo treba navrhnúť obecný spôsob umožňujúci spätné mapovanie aktivít sekvenčných diagramov uvedené v sekcii 3.2.2, bez konkrétneho naviazania na prvky modelu.

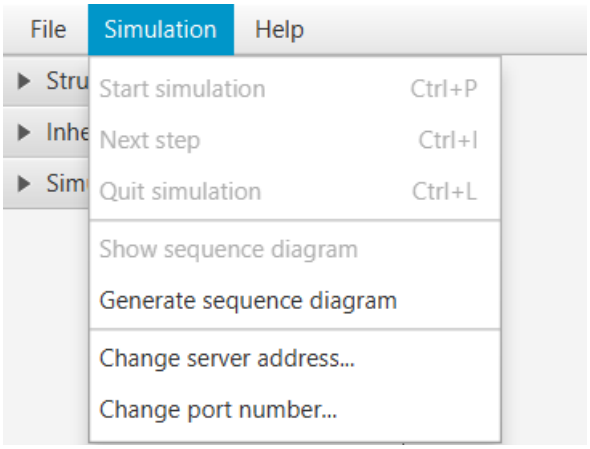
Dosiahnuté to je pomocou troch premenných typu `StringProperty`, ktoré sú predané do konštruktoru triedy `SeqDiaView` pri vytváraní vzhľadu sekvenčného diagramu. Tie predstavujú názov triedy, názov siete a názov prechodu. Pomocou názvu triedy a názvu siete je možné identifikovať konkrétnu sieť modelu. Názov prechodu je nastavený, pokiaľ bola požiadavka na nájdenie konkrétneho prechodu, v ostatných prípadoch je to prázdny reťazec.

Tieto premenné sú nastavované pri kliknutí na položky kontextového menu grafických prvkov sekvenčných diagramov. Pri čiare života je to položka `Show object net`, ktorá otvorí sieť objektu príslušnej triedy. Pri správach volania metódy sú to položky `Show method net`, ktorá otvorí sieť príslušnej metódy, a `Show calling transition`, ktoré otvorí sieť, v ktorom sa nachádza prechod, z ktorého bolo volanie správy uskutočnené a tento prechod je po krátku dobu zvýraznený. Pri návratových správach je táto položka nazvaná `Show return transition`, ale výsledok je rovnaký. Zo spomínaných premenných je názov triedy nastavovaný ako posledný a na zmenu jeho hodnoty možno naviazať funkcionalitu otvárania sietí v editore. To je implementované metódou `reverseMapping` v triede `Controller`, ktorá je súčasťou modulu editoru.

4.8 Prvky grafického rozhrania

Definícia grafického rozhrania sa nachádza v súbore `opne-view.fxml` v rámci modulu editoru. Do tohto súboru boli doplnené prvky, na ktoré je naviazaná funkcionalita generovania sekvenčných diagramov. Zobrazené sú na obrázku 4.2. Ide o následujúce položky v rámci menu `Simulation`:

- `Show sequence diagram` – Umožňuje zobraziť sekvenčný diagram generovaný na základe aktuálneho simulačného behu. Pokiaľ nie je spustená simulácie, tlačidlo je zakázané. Obsluha kliknutia na tento prvok je definovaná metódou `showSeqDia` v triede `Controller`.
- `Generate sequence diagram` – Umožňuje vygenerovať sekvenčný diagram z aktuálne otvoreného modelu. Pokiaľ je spustená simulácia, tlačidlo je zakázané. V reakcii na kliknutie na tento prvok je vyvolaná metóda `generateSeqDia` v triede `Controller`. Tá volá metódu modulu `generateSequenceDiagram` a navrátený sekvenčný diagram zobrazí v novom okne.



Obr. 4.2: Časť rozhrania aplikácie zobrazujúca položky menu `Simulation`.

Kapitola 5

Testovanie

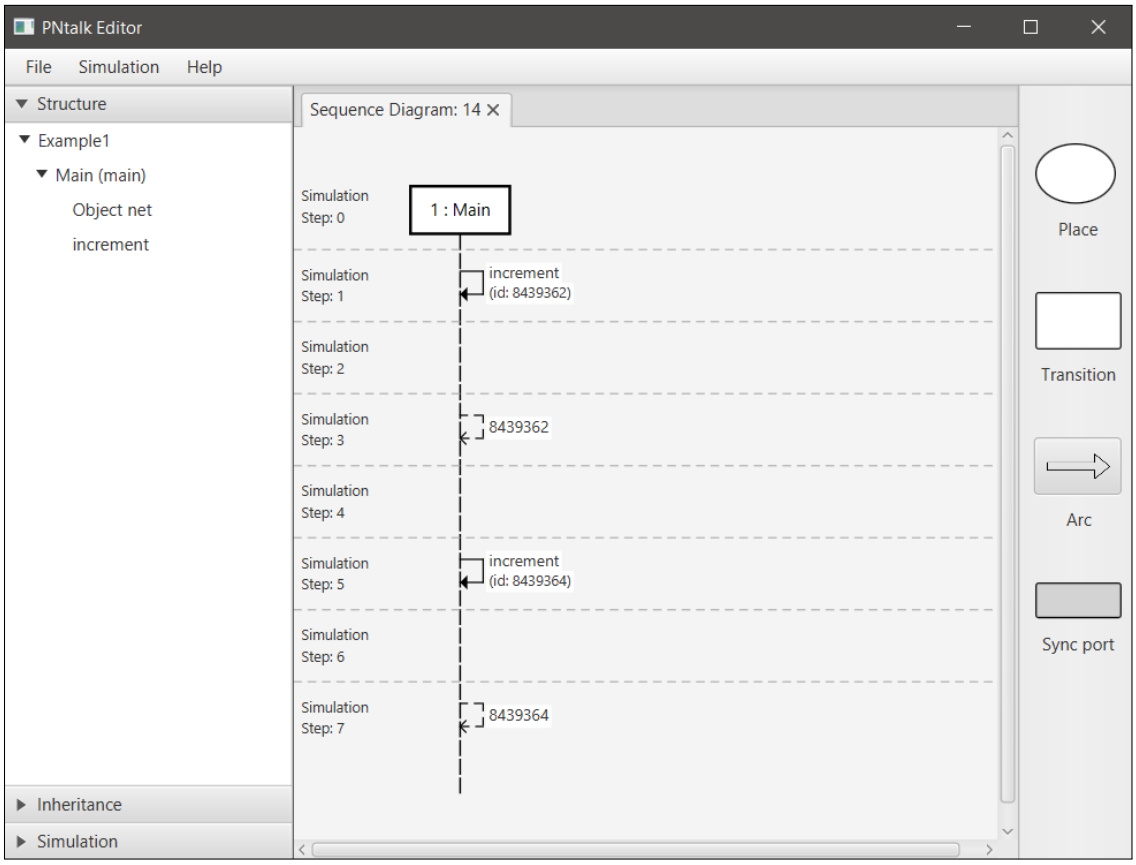
V tejto kapitole sú uvedené príklady demonštrujúce funkcionalitu a grafické zobrazenie výsledného riešenia. Popis každého príkladu obsahuje tri časti: cieľ, model a priebeh simulácie. V časti cieľ je vysvetlený zámer, s ktorým je príklad uvedený. Model popisuje štruktúru modelu OOPN a jeho časti. Priebeh simulácie opisuje priložený obrázok s výsledným sekvenčným diagramom. Uvedené sú tiež možnosti ako otestovať spätné mapovanie aktivít sekvenčných diagramov a na koniec sú popísané chyby, ktoré sa vyskytli pri práci s editorom.

5.1 Volanie vlastnej metódy

Cieľ. Cieľom príkladu je ukázať volanie vlastnej metódy a odpovedajúcej návratovej správy.

Model. Model obsahuje len jednu triedu nazvanú `Main`. Objektová sieť tejto triedy obsahuje miesto `counter` s počiatočným značením `0`. Trieda má definovanú metódu `increment`, ktorá zvýši hodnotu v tomto mieste o `1`. Sieť metódy teda pristupuje k miestu siete objektu a mení jeho dáta. Sieť objektu obsahuje tiež logiku volania tejto metódy v nekonečnom cykle. Medzi volaním je vždy pauza jeden simulačný krok, ktorý je modelovaný prázdnym prechodom medzi jednotlivými volaniami.

Priebeh simulácie. Zachytený na obrázku 5.1. Na začiatku simulácie je vytvorený objekt počiatočnej triedy. To je zachytené čiarou života reprezentujúcou tento objekt, ktorá začína v simulačnom kroku 0. Následne je volaná vlastná metóda `increment`. Zobrazená správa začína aj končí v tej istej čiare života. Pri správe je zobrazený názov volanej metódy a jej id na simulačnom servery. O dva kroky neskôr je zachytené ukončenie metódy, ktoré opäť začína aj končí v tej istej čiare života. Pri tejto správe je zobrazený text s id správy ktorej ukončenie reprezentuje. V ďalšom kroku je vykonaný prázdny prechod siete objektu a tento cyklus sa opakuje teoreticky do nekonečna.



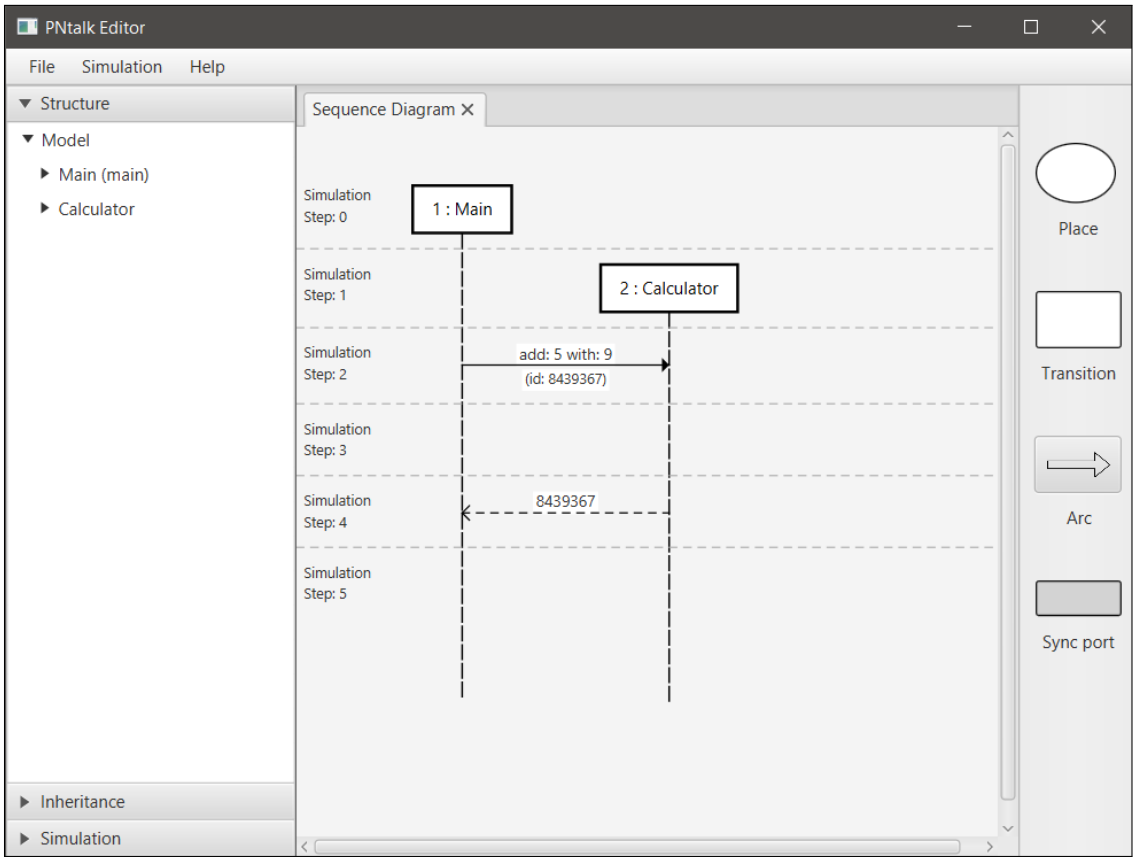
Obr. 5.1: Príklad vygenerovaného sekvenčného diagramu z modelu, ktorého cieľom je zobraziť volanie vlastnej metódy.

5.2 Vytvorenie objektu a volanie jeho metódy

Cieľ. Cieľom príkladu je ukázať vytvorenie druhého objektu počas simulácie a volanie jeho metódy. Súčasťou je aj zobrazenie príslušnej návratovej správy.

Model. Model OOPN obsahuje dve triedy: `Main` a `Calculator`. Trieda `Calculator` má jednu sieť metódy reagujúcu na správu `add: x with: y`. Táto metóda prijíma dva číselné parametre, ktorých súčet je vrátený. Jej sieť sa skladá z parametrových miest, jedného prechodu, v ktorom je vykonaný súčet, a z návratového miesta `return`. Sieť objektu tejto triedy je prázdna. Sieť objektu triedy `Main` obsahuje logiku tohto príkladu. V prvom prechode je vytvorený objekt triedy `Calculator` a v druhom je tomuto objektu zaslaná správa `add: 5 with: 9`.

Priebeh simulácie. Zachytený na obrázku 5.2. Sekvenčný diagram začína akotypicky vytvorením objektu počiatočnej triedy na začiatku simulácie. V prvom kroku pribudne nová čiara života, reprezentujúca vznik nového objektu. V ďalšom kroku je volaná jeho metóda. To je znázornené správou od jednej čiary života k druhej. Nad touto čiarou správy je zobrazený názov metódy spolu s parametrami, s ktorými bola volaná. Pod čiarou sa nachádza id správy na simulačnom servery. O dva kroky neskôr dôjde k ukončeniu metódy. Návratová správa obsahuje opäť text s id správy ktorej ukončenie reprezentuje. Týmto je celá simulácia ukončená.



Obr. 5.2: Príklad vygenerované sekvenčného diagramu, ktorého cieľom je zobraziť vytvorenie objektu a volanie jeho metódy.

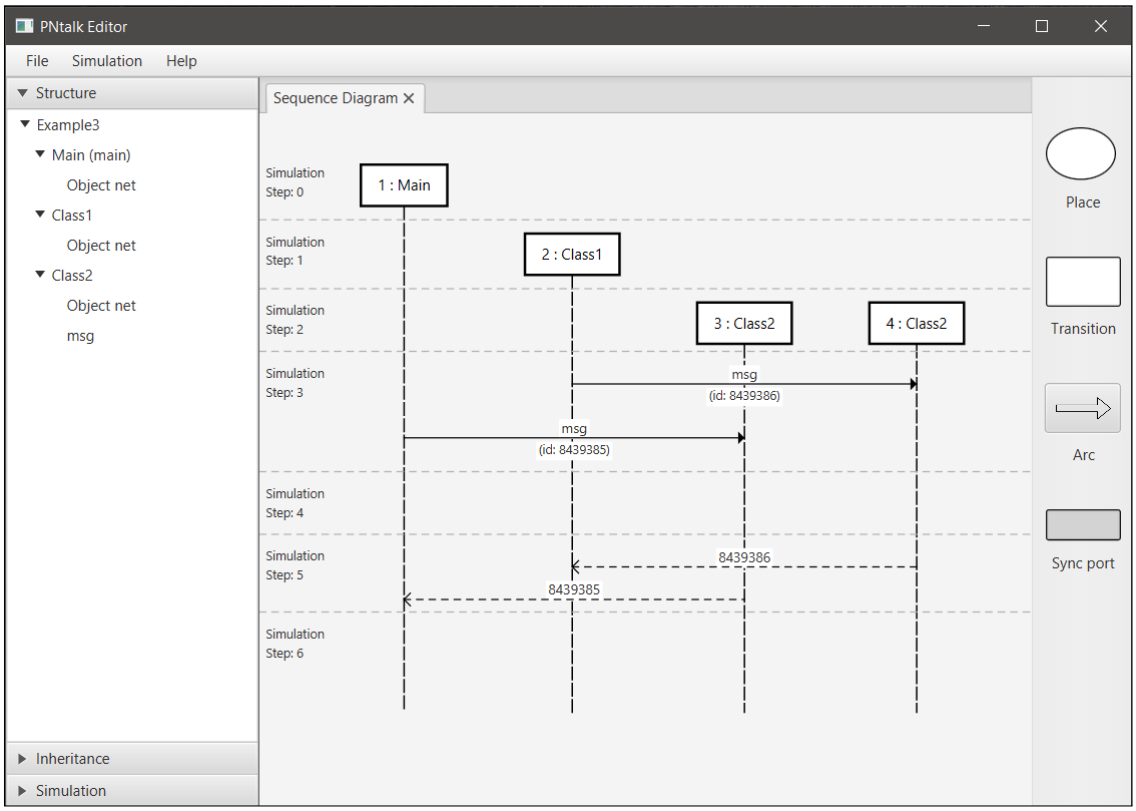
5.3 Paralelizmus v modeloch OOPN

Cieľ. Cieľom príkladu je zachytiť a zobraziť paralelné vykonávanie akcií v modeloch Objektovo orientovaných Petriho sietí. Príklad zachytáva vytvorenie dvoch objektov v jednom simulačnom kroku, volanie dvoch metód v jednom simulačnom kroku a tiež návrat z týchto metód v jednom simulačnom kroku. Príklad bol vytvorený s cieľom zobraziť všetky spomenuté možnosti a nemodeluje žiadny scenár reálneho sveta.

Model. Model obsahuje tri triedy: `Main`, `Class1` a `Class2`. Trieda `Main` je počiatočnou triedou modelu. V rámci jej siete objektu je najprv vytvorený objekt triedy `Class1`, potom objekt triedy `Class2` a nakoniec je volaná metóda `msg` druhého z vytvorených objektov. V sieti objektu triedy `Class1` je najprv vytvorený nový objekt triedy `Class2` a volaná jeho metóda `msg`. Sieť objektu triedy `Class2` je prázdna. Táto trieda však obsahuje metódu `msg`, ktorá neprijíma žiadne parametre a obsahuje len prázdne vykonanie prechodu.

Priebeh simulácie. Zachytený na obrázku 5.3. Zaujímavé časti začínajú v druhom simulačnom kroku, kde je možné vidieť vznik dvoch objektoch naraz. Pokiaľ by výpis stavu obsahoval aj správy vytvorenia objektu, bolo by možné priradiť novo vytvorené objekty k tým, z ktorých boli vytvorené. Takto je možné len poznamenať, že objekt s id `3` bol vytvorený prechodom vykonaným v objekte triedy `Main` a objekt s id `4` vykonaním prechodu v objekte triedy `Class1`. Táto dedukcia vyplýva zo znalosti modelu a následného zasielania

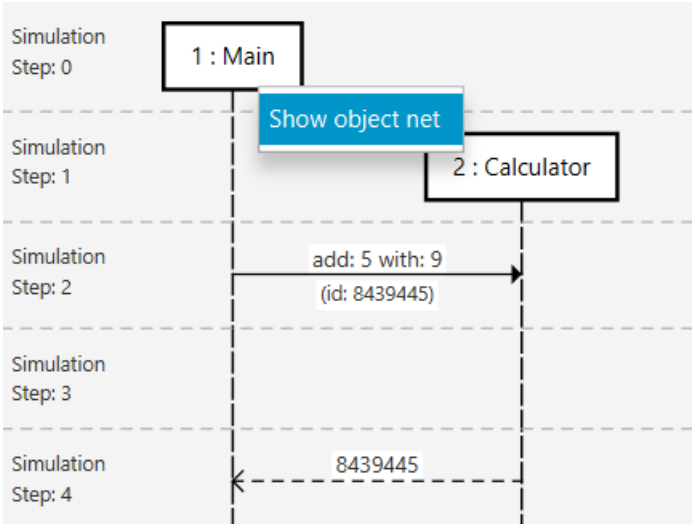
správ medzi objektami v simulácii. V následujúcom kroku je zachytené volanie dvoch metód v jednom kroku. O dva kroky neskôr sú zobrazené návratové správy k týmto volaniam.



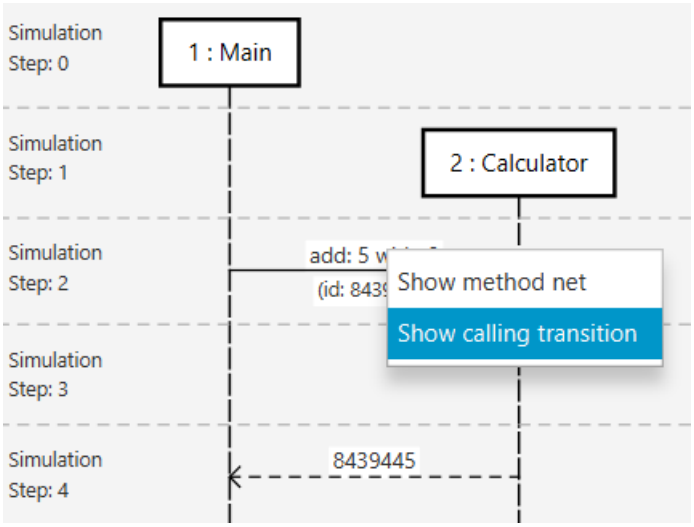
Obr. 5.3: Príklad vygenerované sekvenčného diagramu zachytávajúci paralelné vykonávanie akcií v modeloch OOPN.

5.4 Spätné mapovanie sekvenčných diagramov

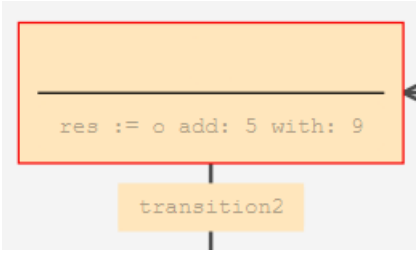
Prepojenie sekvenčných diagramov s modelmi, ktorých scenáre zobrazujú je možné otestovať na ľubovoľnom príklade obsahujúcom aspoň jedno volanie metódy. Po kliknutí na tlačidlo `Show object net` v kontextovom okne čiary života, zobrazeného na obrázku 5.4, je otvorená odpovedajúca sieť objektu triedy. Podobne pri kliknutí na tlačidlo `Show method net` v kontextovom okne správy, zobrazeného na obrázku 5.5, je otvorená sieť metódy danej správy. Tlačidlo `Show calling transition` otvorí sieť obsahujúcu prechod, z ktorého bolo volanie správy realizované a tento prechod je dobu desiatich sekúnd zvýraznený na červeno. To je možné vidieť na obrázku 5.6.



Obr. 5.4: Kontextové okno čiary života obsahujúce možnosť zobrazenia odpovedajúcej siete objektu v modeli OOPN.



Obr. 5.5: Kontextové okno správy sekvenčného diagramu obsahujúce možnosti spätného mapovania do modelu OOPN.



Obr. 5.6: Zvýraznenie prechodu, z ktorého bolo realizované volanie metódy, pri spätnom mapovaní do modelu OOPN.

5.5 Chyby editoru

Pri testovaní príkladov sa vyskytlo niekoľko problémov editoru, ktoré budú v tejto časti spomenuté. Uvedené triedy a metódy sú súčasťou implementácie modulu editoru. Všetky chyby boli v rámci práce opravené a v implementačnej časti sú riadne vyznačené príslušným komentárom.

Najzávažnejším bolo chybné spracovanie stavu simulácie získaného zo serveru. To nepočítalo s možnosťou prázdneho zoznamu v rámci výpisu, napr. pokiaľ sieť objektu neobsahovala žiadne prvky. Prejavovalo sa to zacyklením aplikácie, ktorú bolo následne treba tvrdo ukončiť. Toto spracovanie je realizované metódou `parseState` v triede `SimulationHandler`. V rámci opravy chyby bola tiež vytvorená pomocná metóda `skipActual`, ktorá preskočí logický prvok výpisu stavu.

Ďalšia chyba sa prejavila pri odstránený metódy z panelu štruktúry. Odstránená bola len časť s jej zobrazením, avšak v dátach táto metóda stále existovala. To znemožňovalo vytvoriť metódu s rovnakým názvom a tiež bola táto metóda zaslaná na simulačný server aj keď si užívateľ myslel, že ju už odstránil. V rámci opravy chyby stačilo reflektovať toto odstránenie do dát modelu. To sa nachádza v metóde `deleteItem` v triede `Controller`, ktorá je volaná v reakcii na odstránenie prvku z panelu štruktúry.

Podobná chyba tej predchádzajúcej sa vyskytla pri zmene hrany zo vstupnej na výstupnú a naopak. Táto zmena nebola reflektovaná do dát modelu. To sa však neprejavilo pri zaslaní na simulačný server ale pri zápise do súboru. Po opätovnom načítaní bol smer hrany nezmenený. Táto chyba je opravená v metóde `createContextMenu` triedy `ArcView`.

Posledná chyba sa prejavila pri pokuse o otvorenie kontextového okna panelu štruktúry modelu a panelu hierarchie dedičnosti. Po otvorení aplikácie nie je načítaný žiadny model a editor sa pokúša pristúpiť k neexistujúcemu objektu. Kontrola na existenciu modelu v editore bola doplnená do metód `structureMenu` a `inheritanceMenu` v triede `Controller`.

Kapitola 6

Záver

Cieľom tejto práce bolo navrhnúť mechanizmus transformácie scenárov modelov Objektovo orientovaných Petriho sietí (OOPN) do sekvenčných diagramov. To je dosiahnuté na základe simulovania týchto modelov, získania stavu v každom kroku simulácie a jej následnej analýzy. Práca vychádza z jazyka PNtalk, ktorý je konkrétnou implementáciou OOPN.

Výsledkom je plne funkčný modul v jazyku Java, schopný generovať sekvenčné diagramy na základe dodaného modelu. Vytvorené sekvenčné diagramy zobrazujú pomocou čiar života objekty a správami je modelované vzájomné volanie metód. Modul je zapracovaný do grafického editoru jazyka PNtalk, v ktorom je možné vytvárať a editovať modely a je prepojený so simulačným serverom. Dôležitou vlastnosťou je mapovanie aktivít a prvkov sekvenčných diagramov späť do modelov OOPN. Opravené boli tiež niektoré chyby editoru, ktoré sa počas práce s ním vyskytli.

Vytvorené riešenie je skôr finálnou verziou a poskytuje len menší priestor pre ďalšie vylepšenia. Do vygenerovaných sekvenčných diagramov by bolo možné pridať prvok aktivácie, jedná sa však len o vizuálnu stránku. Taktiež v nich nie sú zobrazené správy vytvorenia objektu, avšak k tomu by bolo nutné upraviť formát stavu simulácie získaného zo serveru. V prepojení s editorom by bolo vhodné do budúcnosti oddeliť modul komunikujúci so simulačným serverom. V tejto chvíli je to riešené v každom module zvlášť.

Modul je navrhnutý s ohľadom na prípadné ďalšie rozšírenia a vďaka jeho implementačnej nezávislosti od editoru je možné ho jednoducho použiť aj v ďalších prácach a projektoch.

Literatúra

[1] Fakhroutdinov, K. *Combined Fragment* [online]. [cit. 2023-04-19]. Dostupné z: https://www.uml-diagrams.org/sequence-diagrams-combined-fragment.html.

[2] Herrity, K. *What Is Object-Oriented Programming (OOP)? A Complete Guide* [online]. 2023 [cit. 2023-22-04]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-object-oriented-programming.

[3] Janoušek, V. *Modelování objektů Petriho sítěmi*. Brno, CZ, 1998. Dizertačná práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a informatiky. Vedúci práce Doc. RNDr. Milan Češka, C.

[4] Kochaníčková, M. *Petriho sítě* [online]. 2008 [cit. 2023-20-04]. Dostupné z: http://phoenix.inf.upol.cz/esf/ucebni/petriho_site.pdf.renamed.

[5] Martin, M. *MVC Framework Tutorial for Beginners: What is, Architecture Example* [online]. 2023 [cit. 2023-26-04]. Dostupné z: https://www.guru99.com/mvc-tutorial.html.

[6] Nyakundi, H. *OOP Meaning – What is Object-Oriented Programming?* [online]. 2022 [cit. 2023-22-04]. Dostupné z: https://www.freecodecamp.org/news/what-is-object-oriented-programming/.

[7] OpenJFX. *JavaFx* [online]. 2023 [cit. 2023-30-04]. Dostupné z: https://openjfx.io/.

[8] Pharo. *About Pharo* [online]. 2023 [cit. 2023-24-04]. Dostupné z: https://pharo.org/about.

[9] Singh, K. *Introduction of Java* [online]. 2022 [cit. 2023-30-04]. Dostupné z: https://www.scaler.com/topics/java/introduction-to-java/.

[10] Wang, J. a Tepfenhart, W. Petri Nets. In:. Jún 2019, s. 201–243. DOI: 10.1201/9780429184185-8. ISBN 9780429184185.

[11] Wikipedia contributors. *Petri net* [online]. 2023 [cit. 2023-20-04]. Dostupné z: https://en.wikipedia.org/wiki/Petri_net.

[12] Wikipedia contributors. *Unified Modeling Language* [online]. 2023 [cit. 2023-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Unified_Modeling_Language.

[13] Wisbey, O. *Sequence diagrame* [online]. 2023 [cit. 2023-04-19]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/sequence-diagram.

[14] Švub, D. *Grafický editor návrhových modelů*. Brno, CZ, 2022. Diplomová práca. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedúci práce Ing. Radek Kočí, P.

Príloha A

Obsah priloženého pamäťového média

- `doc` – Dokumentácia zdrojového kódu vygenerovaná pomocou nástroja Javadoc.
- `examples` – Príklady modelov OOPN popísaných v kapitole 5.
- `latex` – Zdrojové súbory technickej správy.
- `opn-editor` – Modul editoru.
  - `src` – Zdrojové súbory editoru. Všetky doplnené časti sú popísané v tejto správe a v zdrojových súboroch sú riadne vyznačené pomocou komentárov.
  - `pom.xml` – Konfiguračný súbor.
- `sqdia-ext` – Modul sekvenčných diagramov.
  - `src` – Zdrojové súbory sekvenčných diagramov. Predstavujú vlastnú časť práce.
  - `pom.xml` – Konfiguračný súbor.
- `pntalk-simulator` – Obsahuje obraz simulačného serveru použitého pri vypracovávaní tejto práce.
- `opn-editor-1.0.jar` – Spustiteľný multiplatformový JAR súbor.
- `pom.xml` – Konfiguračný súbor nástroja Apache Maven.
- `README` – Súbor s informáciami o aplikácii a pokynmi pre spustenie.
- `xbenoa00-technicka-sprava.pdf` – Elektronická verzia tejto technickej správy vo formáte PDF.

Príloha B

Formát stavu simulácie

Táto príloha obsahuje popis formátu stavu simulácie získaného zo simulačného serveru. Formát sa nachádza vo výpise B.1. Symbol # nie je jeho súčasťou. Text za ním predstavuje komentár k aktuálnemu elementu. V jeho špecifikácii sú použité nasledujúce skratky:

- **N** – číslo, napr. 5
- **S** – reťazec uzatvorený apostrofmi, napr. 'string'
- **C** – symbol, napr. #e
- **T** – typ prvku v mieste:
  - **#obj** – jednoduchý objekt (číslo, symbol, reťazec)
  - **#ref** – referencia na iný objekt
- **V** – hodnota prvku v mieste:
  - **T == #obj** – hodnota reprezentujúca objekt
  - **T == #ref** – id objektu

```
(
  ( % zoznam rozpracovaných správ
    ( % správa
      N % id správy
      C % identifikátor metódy
      N % id volajúceho objektu
      N % id prijímajúceho objektu
      C % typ správy
      ( ... zoznam parametrov/hodnôt ... )
    )
    ( % ďalšia správa ... )
  )

  % zoznam objektov
  ( % objekt
    S % názov triedy
```

```
    N % id objektu
    ( % zoznam procesov (objektová sieť a invokované metódy)
      ( % proces
        S % názov metódy, 'object' pre objektovú sieť
        N % id procesu
        ( % zoznam miest
          ( % miesto
            S % názov miesta
            ( % obsah miesta, jednotlivé prvky oddelené medzerou
              N->T->V % prvok miesta, N je početnosť prvku v mieste
            )
          )
        )
        ( % zoznam spustených prechodov (vlákien)
          ( % vlákno
            S % názov prechodu
            N % identifikátor vlákna
            N % index vykonávanej akcie v prechode
            ( % zoznam premenných a naviazaných hodnôt, oddelené medzerou
              C->T->V % C=premenná, T=typ, V=hodnota
            )
          )
        )
      )
    )
    ( % zoznam závislostí (správy zaslané z tohto objektu)
      ( % závislosť
        N % id správy
        N % id procesu, z ktorého bola správa zaslaná
        S % názov prechodu, z ktorého bola správa zaslaná
        N % id vlákna, z ktorého bola správa zaslaná
      )
    )
    ( % zoznam vyvolaných procesov v reakcii na správu, oddelené medzerou
      N->N % id_správy -> id_procesu
      N->N % id_správy -> id_procesu
      ...
    )
  )
  ( ... ďalší objekt ... )
)
```

Výpis B.1: Formát stavu simulácie.