FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Zadání bakalářské práce

Název:	Implementace simulace BORM ORD diagramů na platformě OpenPonk
Student:	Antonín Jirásek
Vedoucí:	doc. Ing. Robert Pergl, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Informační systémy a management
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je implementace simulace BORM ORD diagramů na platformě OpenPonk.

1. Seznamte se platformou pro konceptuální modelování OpenPonk, jazykem Pharo a existující implementací BORM ORD diagramů.
2. Seznamte se s existujícími přístupy k simulaci BORM ORD diagramů.
3. Navrhněte a implementujte algoritmy pro simulaci BORM ORD diagramů a interakce s uživatelem. Inspirujte se existujícími implementacemi simulací FSM a Petri Net. Dle potřeby rozšiřte metamodel a UI pro modelování BORM ORD diagramů.
4. Řešení zdokumentujte, otestujte a demonstrujte na případové studii. Diskutujte přínosy pro výuku a praxi.

Elektronicky schválil/a Ing. David Buchtela, Ph.D. dne 19. dubna 2023 v Praze.

Bakalářská práce

IMPLEMENTACE SIMULACE BORM ORD DIAGRAMŮ NA PLATFORMĚ OPENPONK

Antonín Jirásek

Fakulta informačních technologií
Katedra teoretické informatiky
Vedoucí: doc. Ing. Robert Pergl, Ph.D.
16. května 2024

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2024 Antonín Jirásek. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení, je nezbytný souhlas autora.

Odkaz na tuto práci: Jirásek Antonín. *Implementace simulace BORM ORD diagramů na platformě OpenPonk*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Obsah

Poděkování vi

Prohlášení vii

Abstrakt viii

Seznam zkratek ix

Úvod 1

I Rešerše 2

1 Konceptuální modelování 3
1.1 Známé standardy a metodiky konceptuálního modelování . . . 3
1.1.1 UML . . . 3
1.1.2 BPMN . . . 4
1.1.3 Petriho sítě . . . 4

2 Simulace 5
2.1 Simulace v konceptuálním modelování . . . 5
2.2 Známé simulátory konceptuálních modelů . . . 5

3 BORM 8
3.1 BORM ORD . . . 8
3.1.1 OBA - Object Behaviour Analysis . . . 10
3.1.2 Pravidla BORM ORD . . . 10

4 BORM ORD simulace 11
4.1 Teorie simulace BORM ORD . . . 11
4.2 Existující implementace . . . 12
4.2.1 Craft.CASE . . . 12
4.2.2 Portál pro optimalizaci BORM procesů . . . 13

5 Využité technologie 14
5.1 Smalltalk . . . 14
5.2 Pharo . . . 14
5.2.1 SUnit testing . . . 14
5.3 OpenPonk . . . 14
5.3.1 OpenPonk Simulation . . . 15
5.3.2 Petriho sítě na platformě OpenPonk . . . 15

II Praktická část 16

6 Analýza 17
6.1 Funkční požadavky . . . 17
6.2 Nefunkční požadavky . . . 17
6.3 Případy užití . . . 17

7 Implementace 19
7.1 Logika simulace . . . 19
7.1.1 Algoritmus pro procházení diagramem . . . 19
7.1.2 Input a output podmínky . . . 19
7.1.3 Rozdělení input podmínek . . . 20
7.2 Hierarchie a architektura . . . 22
7.3 Nutné změny BORM ORD modelu . . . 23
7.4 OPBormSimulator . . . 24
7.5 OPBormElementSimuator . . . 24
7.6 OPBormSimulationGUI . . . 25
7.7 OPBormConstraintMenuGUI . . . 26
7.8 Zobrazení simulace na diagramu . . . 27

8 Testování 29
8.1 BormSimulationTest nadtřída . . . 29
8.2 Jednoduché Unit testy . . . 30
8.3 Testování output podmínek . . . 30
8.4 Testování input podmínek . . . 30

9 Případová studie 33
9.1 Výhody využití simulace . . . 33
9.1.1 Nalezení chyb – diagram eshopu . . . 33
9.1.2 Jednodušší porozumění procesů – diagram získání nového hráče . . . 35
9.2 Porovnání simulace s nástrojem Craft.CASE . . . 37
9.2.1 Simulace podpory . . . 37
9.2.2 Simulace odpálení bomby . . . 37

10 Instalace a návod k použití 40
10.1 Instalace . . . 40
10.2 Návod k použití . . . 40

11 Závěr 42

Obsah příloh 45

Seznam obrázků

2.1 Camunda BPMN simulator . . . 6
2.2 BPSimulator . . . 6
2.3 Yawl, získáno z: https://yawlfoundation.github.io/ . . . 7

3.1 Tabulka BORM ORD [8] . . . 9

4.1 Craft.CASE simulace . . . 12
4.2 BORM portál simulace . . . 13

5.1 Třídový diagram implementace simulace Petriho sítí [17] . . . 15

6.1 Use case diagram aplikace . . . 18

7.1 Příklad pro input podmínky – *Any* a *All* . . . 21
7.2 Příklad pro input podmínky – *All* a *All* . . . 22
7.3 Třídy package OPBormSimulation . . . 23
7.4 Dědičnost mezi třídami balíčků ModelSimulation a BormSimulation . . . 23
7.5 Okno simulace . . . 25
7.6 Okno simulace s podmínkami k nastavení . . . 26
7.7 Okno pro přednastavení podmínek . . . 27
7.8 Příklad různých barev dle okolností . . . 28

8.1 Třídy testů BORM simulace . . . 29
8.2 Diagram modelu pro testování output podmínek . . . 30
8.3 Diagram modelu pro testování input podmínek . . . 32

9.1 Diagram pro eshop . . . 34
9.2 Opravený diagram pro eshop . . . 35
9.3 Diagram podpory namodelovaný v nástroji Craft.CASE . . . 36
9.4 Diagram pro eshop . . . 37
9.5 Diagram odpálení bomby v Craft.CASE . . . 38
9.6 Simulace odpálení bomby v nástroji Craft.CASE . . . 39

10.1 Okno BORM editoru . . . 41
10.2 Okno simulace s podmínkami k nastavení . . . 41

Seznam výpisů kódu

7.1 Metoda pro nastavení správného krokování . . . 24
7.2 Metody využité pro rozhodnutí krokování . . . 24
7.3 Input podmínky pro přechody . . . 24
7.4 Output podmínky pro přechody . . . 25

8.1 exampleModel metoda pro vytvoření modelu pro testy output podmínek . . . 31
8.2 Helper metoda pro testování input podmínek . . . 32
8.3 Test input all podmínek . . . 32

Chtěl bych poděkovat především svému vedoucímu práce, doc. Ing. Robertu Perglovi, Ph.D. za dobré rady a konzultace. Dále také děkuji Ing. Janu Blizničenkovi za zasvěcení do platformy OpenPonk a za jeho cenné doporučení k implementaci. Také můj vděk patří všem, co se přede mnou podíleli na vývoji platformy OpenPonk. Závěrem bych také rád poděkoval mé rodině a mým přátelům, za jejich podporu během psaní bakalářské práce a celého mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 16. května 2024

Abstrakt

Tato práce se zaměřuje na implementaci simulace BORM ORD (Business Objects Relation Modelling Object Relation Diagram) na platformě OpenPonk. OpenPonk je metamodelovací platforma implementovaná v dynamickém prostředí Pharo, zaměřena na podporu činností v oblasti softwaru a podnikového inženýrství. Výsledek této bakakářské práce je rozšíření platformy OpenPonk o simulaci BORM ORD, který je přínosný nejen pro studenty FIT ČVUT využívající platformu OpenPonk, ale i pro kohokoli, kdo má zájem modelovat BORM ORD diagramy. Součástí této práce je také studium existujících přístupů k simulaci BORM ORD diagramů, upřesnění pravidel simulace, případová studie a diskuse o přínosu simulace BORM ORD.

Klíčová slova konceptuální modelování, simulace diagramů, Pharo, Smalltalk, OpenPonk, procesní inženýrství, BORM, BORM simulace, použití ve výuce

Abstract

This paper focuses on the implementation of the BORM ORD (Business Objects Relation Modelling Object Relation Diagram) simulation on the OpenPonk platform. OpenPonk is a metamodelling platform implemented in the Pharo dynamic environment, aimed at supporting software and business engineering activities. The result of this bachelor thesis is the addition of BORM ORD to the OpenPonk platform, that will be beneficial not only for FIT CTU students using the OpenPonk platform, but for anyone interested in modelling BORM ORD diagrams as well. The study of existing approaches to simulating BORM ORD diagrams, refinement of the simulation rules, a case study and a discussion of the benefits of BORM ORD simulation are also part of this thesis.

Keywords Conceptual modelling, diagram simulation, Pharo, Smalltalk, OpenPonk, process engineering, BORM, BORM simulation, teaching

Seznam zkratek

BORM	Business Objects Relation Modelling
FSM	Finite State Machine / Konečný automat
ORD	Object Relation Diagram
OBA	Object Behavioural Analysis
FIT	Fakulta informačních technologií
GUI	Graphical user interface
UML	Unified Modeling Language
OMG	Object Management Group
BPMN	Business Process Model and Notation
BFS	Breadth-first search
CASE	Computer Aided Software Engineering

Úvod

BORM ORD (Business Objects Relation Modelling Object Relation Diagrams) jsou diagramy mapující bussiness procesy. Využívají se zejména k vizualizaci těchto procesů, která pomáhá k jejich rychlému a jednoduchému zformalizování. Tyto diagramy mohou například pomoci k pochopení těchto procesů. Čím jsou BORM diagramy komplexnější, tím je ale složitější je pochopit. Tento problém může být značně redukován přidáním možnosti simulace, díky které si uživatel může znázornit průběh procesu.

Tato bakalářská práce se zabývá právě simulací BORM diagramů. Teoretická část bakalářské práce se zabývá metodikou a přínosem simulace BORM diagramů a případovou studií. Její praktická část pak obsahuje implementaci simulace BORM ORD pro platformu OpenPonk, vyvíjenou zejména na Fakultě informačních technologií ČVUT.

Výsledek bakalářské práce bude přínosný nejen pro studenty Fakulty informačních technologií ČVUT studujících předmět BI–ZPI (základy procesního inženýrství), ale má hodnotu i pro kohokoliv, kdo využívá BORM ORD modelování, včetně profesionálů v této specifické oblasti procesního inženýrství.

Cílem této bakalářské práce je navrhnout a implementovat simulaci BORM ORD modelů pro platformu Openponk. V rámci bakalářské práce budou implementovány algoritmy pro simulaci BORM ORD diagramů a interakci s uživatelem, inspirované již existujícími implementacemi simulací FSM a Petriho sítí v OpenPonk. Bude rozšířen existující metamodel a UI pro modelování BORM ORD diagramů. Řešení bude zdokumentováno, otestováno a demonstrováno v případové studii. Součástí práce bude i analýza a diskuse přínosu simulace BORM ORD modelů pro modelování business procesů jak ve výuce, tak i v praxi.

Část I

Rešerše

Kapitola 1
Konceptuální modelování

V oblasti vývoje informačních systémů je proces konceptuálního modelování klíčovou fází, která poskytuje strukturovaný přístup k reprezentaci a pochopení složitých problémových oblastí. Tato kapitola se zabývá významem konceptuálního modelování a objasňuje jeho základní principy, metodiky, techniky a nástroje. Prostřednictvím zkoumání různých jazyků konceptuálního modelování a jejich aplikací si tato kapitola klade za cíl poskytnout vhled do úlohy konceptuálního modelování při usnadňování efektivní komunikace, porozumění a analýzy v kontextu vývoje informačních systémů.

Konceptuální modelování představuje formální popis vybraných aspektů fyzického i sociálního prostředí, který slouží k porozumění a komunikaci. Tyto popisy, označované jako konceptuální schémata, vyžadují přijetí formálního zápisu, označovaného jako konceptuální model. Konceptuální schémata zahrnují relevantní aspekty dané domény, jako je kancelářské prostředí a s ním spojené činnosti, a usnadňují dosažení shody mezi zúčastněnými stranami, jako jsou pracovníci kanceláří, kteří potřebují společné chápání domény. Konceptuální schémata navíc pomáhají zprostředkovat tuto společnou perspektivu nováčkům prostřednictvím různých grafických a jazykových rozhraní.[1]

1.1 Známé standardy a metodiky konceptuálního modelování

Sekce 1.1 se zaměřuje na konceptuální modely, které jsou pro tuto bakalářskou práci důležité.

1.1.1 UML

Unifikovaný modelovací jazyk (UML) je základním kamenem softwarového inženýrství a nabízí standardizovaný přístup k návrhu, vizualizaci a dokumentaci softwarových systémů. Ve složitém prostředí velkých podnikových aplikací se jazyk UML stává důležitým nástrojem pro systémové architekty i vývojáře. Jeho hlavním cílem je usnadnit vytváření dobře strukturovaných systémů, které se vyznačují škálovatelností, bezpečností a spolehlivostí. Díky vizuálnímu znázornění softwarové architektury pomáhá UML pochopit a sdělit složitosti návrhu systému a zajistit, aby všechny zúčastněné strany byly v souladu s cíli a požadavky specifického projektu. Kromě toho UML podporuje opakované použití kódu a modularitu při programování, což vývojářům umožňuje vytvářet aplikace jako soubor samostatných komponent, a tím zvyšuje efektivitu a udržitelnost kódu. Díky různým typům diagramů a flexibilitě je jazyk UML vhodný pro široké spektrum aplikací bez ohledu na základní technologie nebo platformy. Od modelování obchodních

procesů až po specifikaci systémových interakcí slouží UML jako univerzální rámec, který vývojářům umožňuje efektivně řešit výzvy moderního vývoje softwaru. Jako základ modelově řízené architektury (MDA) OMG pomáhá UML nejen při návrhu systému, ale také usnadňuje bezproblémovou integraci napříč různými middlewarovými platformami. Tak zajišťuje přizpůsobivost a dlouhou životnost softwarových řešení. UML v podstatě překračuje rámec pouhé dokumentace; ztělesňuje systematický přístup k vývoji softwaru a umožňuje týmům transformovat abstraktní koncepty do robustních, škálovatelných a udržovatelných softwarových systémů. [2]

1.1.2 BPMN

BPMN, neboli Business Process Model and Notation, slouží jako standardizovaný grafický jazyk pro zobrazování podnikových procesů v rámci Business Process Diagrams (BPD). BPMN vychází z tradičních metod vývojových diagramů. Jeho cílem je poskytnout intuitivní a zároveň komplexní systém notace, který je určen jak technickým, tak obchodním uživatelům. [3]

BPMN 2.0 nabízí grafické znázornění, dále sémantiku provádění a vykazuje kompatibilitu s dalšími prováděcími jazyky, jako je BPEL. To z něj činí univerzální nástroj pro komunikaci a spolupráci mezi zainteresovanými stranami v oblasti podnikání, včetně analytiků, vývojářů a manažerů. [3]

BPMN v podstatě funguje jako společný jazyk, který usnadňuje bezproblémovou komunikaci a porozumění v celém spektru návrhu a implementace podnikových procesů.

1.1.3 Petriho sítě

Petriho sítě slouží jako teoretický rámec pro modelování toku informací a nabízí k němu formalizovaný přístup. Jejich vlastnosti, koncepty a metodiky se neustále vyvíjejí a jejich cílem je poskytnout intuitivní a efektivní prostředky pro popis a zkoumání dynamiky informací a řízení v systémech. Petriho sítě jsou zvláště cenné pro systémy s asynchronními a souběžnými operacemi. Jejich hlavní uplatnění spočívá v modelování událostních systémů, kde jsou souběžné události proveditelné, avšak podléhají omezením týkajícím se jejich načasování, pořadí nebo četnosti výskytu. [4]

Kapitola 2
Simulace

„Simulace je výzkumná technika, jejíž podstatou je náhrada zkoumaného dyn. systému jeho simulátorem s tím, že se simulátorem se experimentuje s cílem získat informace o původním zkoumaném dynamickém systému.“ [5]

2.1 Simulace v konceptuálním modelování

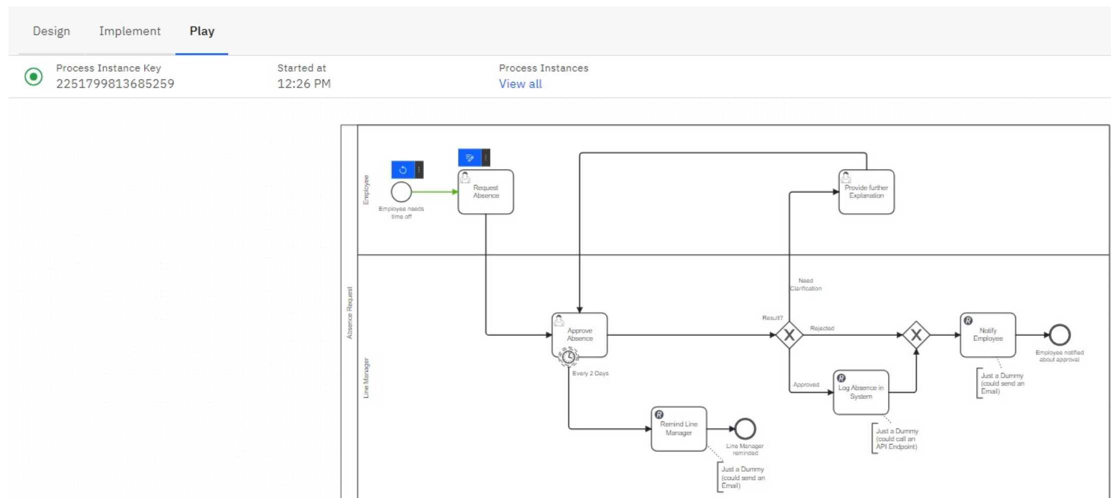
Simulace je v konceptuálním modelování velmi důležitá. Umožňuje totiž uživateli co nejrychleji pochopit daný model. Zároveň také pomáhá návrháři se ujistit, že model opravdu odpovídá reálnému světu.

Čím je model větší, tím jsou tyto vlastnosti simulace přínosnější. Zvlášť ve velkých modelech, které mohou v tištěné formě zabrat i několik listů papíru formátu A1, je zorientování uživatele bez simulace velmi obtížné. Simulace mu tak může uspořit značné množství času.

2.2 Známé simulátory konceptuálních modelů

- Camunda

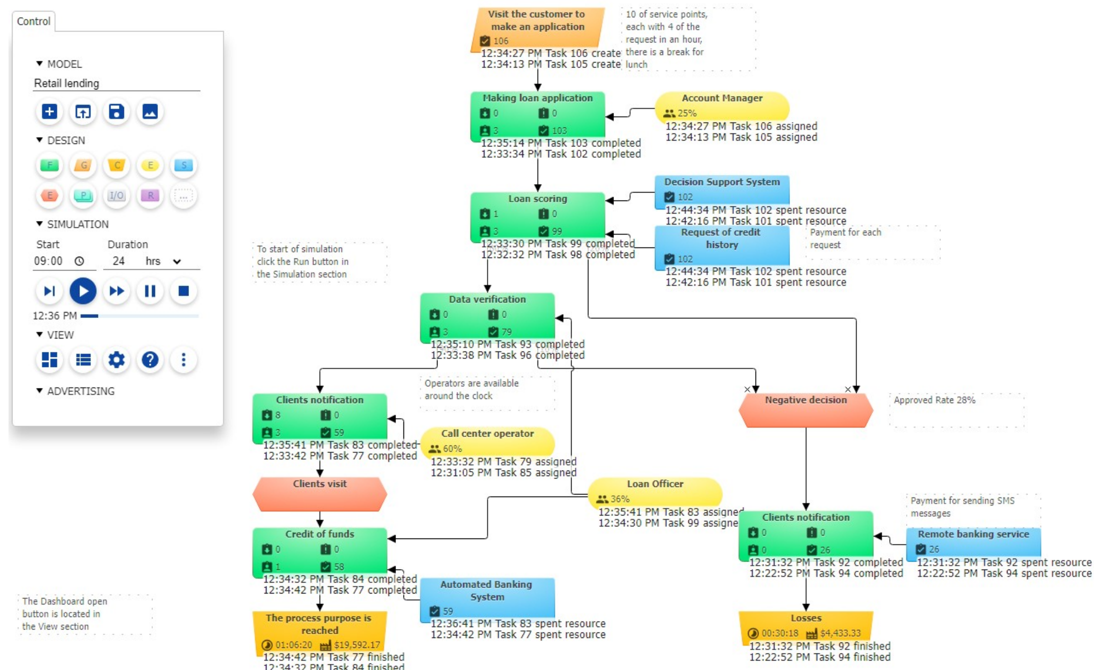
  Camunda nabízí velmi silnou webovou aplikaci pro simulaci BPMN diagramů. Mezi její přednosti patří poměrně jednoduchý webový interface a různé přidané hodnoty k BPMN modelům (např. lze při simulaci vynutit vyplnění tasku, jako je napsání emailu). Další hodnotná funkce, kterou kromě simulace Camunda nabízí, je verifikace modelů, která vám již během tvorby BPMN diagramu může bez simulace napovědět, zda se řídíte dle pravidel BPMN.



Obrázek 2.1 Camunda BPMN simulator

- BPSimulator

  BPsimulator je webová aplikace podporující eEPC, BPMN a Visio modelování.

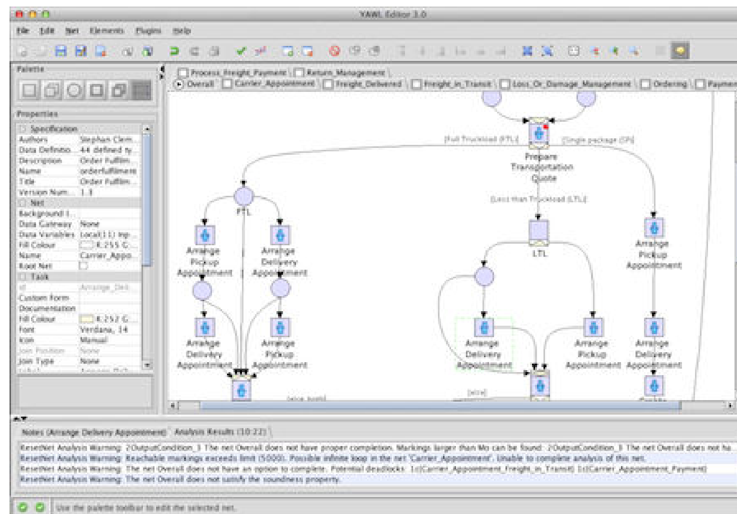


Obrázek 2.2 BPSimulator

- YAWL BPM

  YAWL BPM, neboli Yet Another Workflow Language, je open-source systém pro BPM (Bussiness Process Management). Nabízí výkonné funkce pro modelování, automatizaci a optimalizaci podnikových procesů. Mezi klíčové aspekty YAWL patří robustní modelovací jazyk a bezproblémové zpracování složitých transformací dat. YAWL umožňuje organizacím efek-

tivně řídit pracovní postupy, zpracovávat data a řídit služby v rámci jejich podnikových procesů.[6]



Obrázek 2.3 Yawl, získáno z: https://yawlfoundation.github.io/

O simulátorech metodiky BORM pojednává část příští kapitoly.

Kapitola 3
BORM

BORM (Business Object Relation Modelling) je vývojová metodika pro zachycení znalostí typických business systémů. Vyvíjí se od roku 1993,a z ohlasů odborné veřejnosti lze usuzovat, že je oblíbená jak u uživatelů, tak u vývojářů. Získaná účinnost BORM je do značné míry výsledkem jednotné a jednoduché metody prezentace všech aspektů příslušného modelu. [7]

BORM začínal se záměrem vytvořit metodiku podporující tvorbu softwarových systémů založených na objektově orientovaném přístupu, dále na čistých objektově orientovaných programovacích jazycích a v neposlední řadě i na nerelačních objektových databázích, ale postupně vyšlo najevo, že některé definované techniky pro modelování procesů lze využít i obecně. BORM dospěl do takové metodiky, že pokrývá nejen celý vývoj softwaru, ale i oblasti analýzy požadavků na projektovaný systém a modelování obchodních procesů.

Jedním z hlavních cílů metodiky BORM je přiblížit k sobě uživatele a návrháře v procesu tvorby informačních systémů. Častý problém při vývoji informačních systémů je právě konceptuální mezera, tzv. „conceptual gap“, neboli rozdíl pochopení a znalostí řešení problému mezi vývojářem/designerem a uživatelem či zákazníkem. BORM svojí jednoduchostí a názorností je pochopitelný i pro laika v oblasti řešeného problému, což pomáhá výše zmíněnou mezeru minimalizovat. Další výhoda BORM metodiky je kombinace procesně a objektově orientovaného vývoje, která také dále zužuje konceptuální mezeru. [7]

BORM, stejně jako podobné OOAD (objektově orientovaná analýza a design) metody je založen na spirálním modelu pro vývoj životního cyklu.

3.1 BORM ORD

BORM ORD (Business Object Relation Modelling Object Relation Diagram) je model pro znázornění business procesů.

pojem	symbol	popis
Objekt = Participant	Obdélník se jménem zobrazeným uvnitř v levém horním rohu.	Objekt (v úvodních fázích BORMu je označován jako "Participant") představuje účastníka modelovaného procesů, jak byl rozpoznán s pomocí modelových karet.
Stav	Menší obdélník odlišený barvou a typem písma pro pojmenování stavu kreslený dovnitř symbolu pro objekt. (Pro počáteční a koncový stav se používají symboly shodné s UML)	Stavy vyjadřují postupné změny participantů v čase.
Asociace	Silná černá šipka s plným zakončením mezi participanty a nebo stavy. U šipky se píše popis, který blíže specifikuje charakter vazby. Pojmy přirozeného jazyka zde mají přednost před programátorskými označeními typu "dědí", "skládá", ...	Asociace vyjadřují datově orientované vztahy mezi participanty a nebo jejich stavy, protože se mohou v čase měnit. Asociace vyjadřují jednotným způsobem vztahy, které mohou být později upřesněny jako skládání, dědění nebo závislost objektů.
Aktivita	Ovál propojený čárou s participantem nebo jeho stavem. Ovály mohou být kresleny také dovnitř k nim příslušných objektů.	Aktivity reprezentují jednotlivé aspekty chování objektů tak, jak byly rozpoznány pomocí scénářů v modelových kartách.
Komunikace	Šipka, která propojuje aktivity mezi sebou. Malé pojmenované šipky kreslené rovnoběžně k hlavní šipce komunikace vyjadřují datové toky.	Komunikace vyjadřují sled provádění a vzájemnou závislost aktivit různých objektů mezi sebou. datové toky mohou být vedeny oběma směry.
Přechod	Šipka s nevyplněným trojúhelníkovým zakončením, která propojuje aktivity a stavy jednoho objektu.	Součástí přechodu je také aktivita, ze které přechod vychází. Přechod tedy představuje činnost, kterou je třeba vykonat, aby objekt změnil svůj stav.
Podmínka	Přeškrtnutí s textovým popisem u komunikace nebo u propojení aktivity a objektu.	Podmínkou se vyjadřuje omezená platnost komunikace nebo aktivity.

Obrázek 3.1 Tabulka BORM ORD [8]

Obrázek 3.1 shrnuje ty pojmy, které tvoří BORM ORD. Je ale důležité zmínit, že symboly pro různé pojmy se mohou lišit dle implementace. Například v platformě OpenPonk nejsou aktivity odlišeny oválným tvarem, ale obdélníkem se zakulacenými rohy a jiným odstínem barvy. Přechody přitom mají vyplněná trojúhelníková zakončení.

Jelikož tato práce se zabývá simulací BORM ORD, je důležité se zaměřit na OBA (Object Behaviour Analysis).

3.1.1 OBA - Object Behaviour Analysis

Úkolem metody analýzy chování objektů (OBA) je primárně pochopit popis problému a sekundárně formulovat tento popis z perspektivy více vzájemně se ovlivňujících objektů. Tyto objekty plní klíčové systémové role a odpovědnosti tím, že poskytují a přijímají dobře definované služby, jež ovlivňují chování systému. Výsledky této analýzy by měly být srozumitelné pro koncové uživatele a vhodné pro následný návrh a implementaci. Přitom musí být v souladu s cíli a úkoly systému. Na základě metody OBA lze formulovat odhady dalšího vývoje projektu na základě počtu identifikovaných chování účastníků a iniciátorů a vztahů mezi těmito stranami. Metodika OBA sestává z pěti základních kroků. Kroky jsou indexovány od nuly, kde první krok je zdůrazněn neobvyklým označením „nulový“, a to z důvodů, že často přesahuje obvyklý rozsah analýzy.[9]

Pět kroků analýzy chování objektů je [9]:

- Nastavení kontextu pro analýzu,
- pochopení problému se zaměřením na chování,
- definování objektů, které vykazují chování,
- klasifikace objektů a identifikace jejich vztahů,
- modelování dynamiky systému.

3.1.2 Pravidla BORM ORD

Procesy se v BORM ORD znázorňují pomocí participantů. Každý z těchto participantů může „vlastnit“ několik stavů a aktivit. Aktivity a stavy se v rámci jednoho participanta propojují pomocí přechodů, mezi vícero participanty pak pomocí komunikací. Komunikace navíc mohou přenášet data díky „data flows“. Dále mohou být součástí modelu i podmínky, které hrají právě v simulaci BORM ORD modelů důležitou roli.

Detailnější popis jednotlivých částí BORM ORD je uveden v obrázku 3.1.

Kapitola 4
BORM ORD simulace

BORM původně nevznikl s úmyslem simulace, tedy simulace BORM ORD modelů nebyla poměrně dlouhou dobu dobře definována. S návrhem pro pravidla a logiku simulace BORM ORD modelů přišli autoři článku „Towards Formal Foundations for BORM ORD Validation and Simulation“ [10] Martin Podloucký a Robert Pergl. Tento článek byl představen na ICEIS 2014 - 16th International Conference on Enterprise Information Systems. Článek dle mých zjištění obsahuje první formální zápis pravidel a logiky simulace BORM ORD, a to i přesto, že základní simulace byla již dříve nabízena v softwaru Craft.CASE (v roce 2005 se již plánovalo její přidání [11]). Zbytek této kapitoly bude tedy parafrázovat tento článek a primárně z něj vycházet.

4.1 Teorie simulace BORM ORD

V článku „Towards Formal Foundations for BORM ORD Validation and Simulation“ [10] se autoři zaměřili na zlepšení porozumění a formalizaci objektových relačních diagramů (ORD) v rámci Bussiness Object Relation Modelling (BORM). Zavedli dva klíčové principy pro řešení problémů v sémantice a simulaci BORM ORD – princip simultánnosti a princip závislosti (the simultaneity and the dependancy principles).

Princip simultánnosti stanovuje, že v rámci procesu definovaného pomocí BORM ORD nemůže žádný účastník vykonávat více aktivit současně. Tento princip je v souladu s ontologií Mealyho strojů, z níž ORD vychází, a objasňuje interpretaci paralelních větví v diagramu. Jeden účastník se sice může nacházet v několika stavech, nemůže však vykonávat více úkolů zároveň. Dle již stanovených pravidel BORM ORD ale není definováno, v jakém pořadí by se aktivity měly provádět. Tento fakt umožňuje virtuálně paralelizovat provedení aktivit jednoho participanta.

Princip simultánnosti doplňuje princip závislosti, který zdůrazňuje vzájemnou provázanost úkolů v rámci procesu. Úkoly mohou před svým provedením záviset na dokončení jiných úkolů. Explicitním vymezením závislostí úkolů zajišťuje princip závislosti souvislý průběh procesu a zabraňuje ontologicky extravagantním situacím. Tento princip řeší potřebu přesnosti a jednoznačnosti při určování pořadí provádění úloh v rámci procesu.

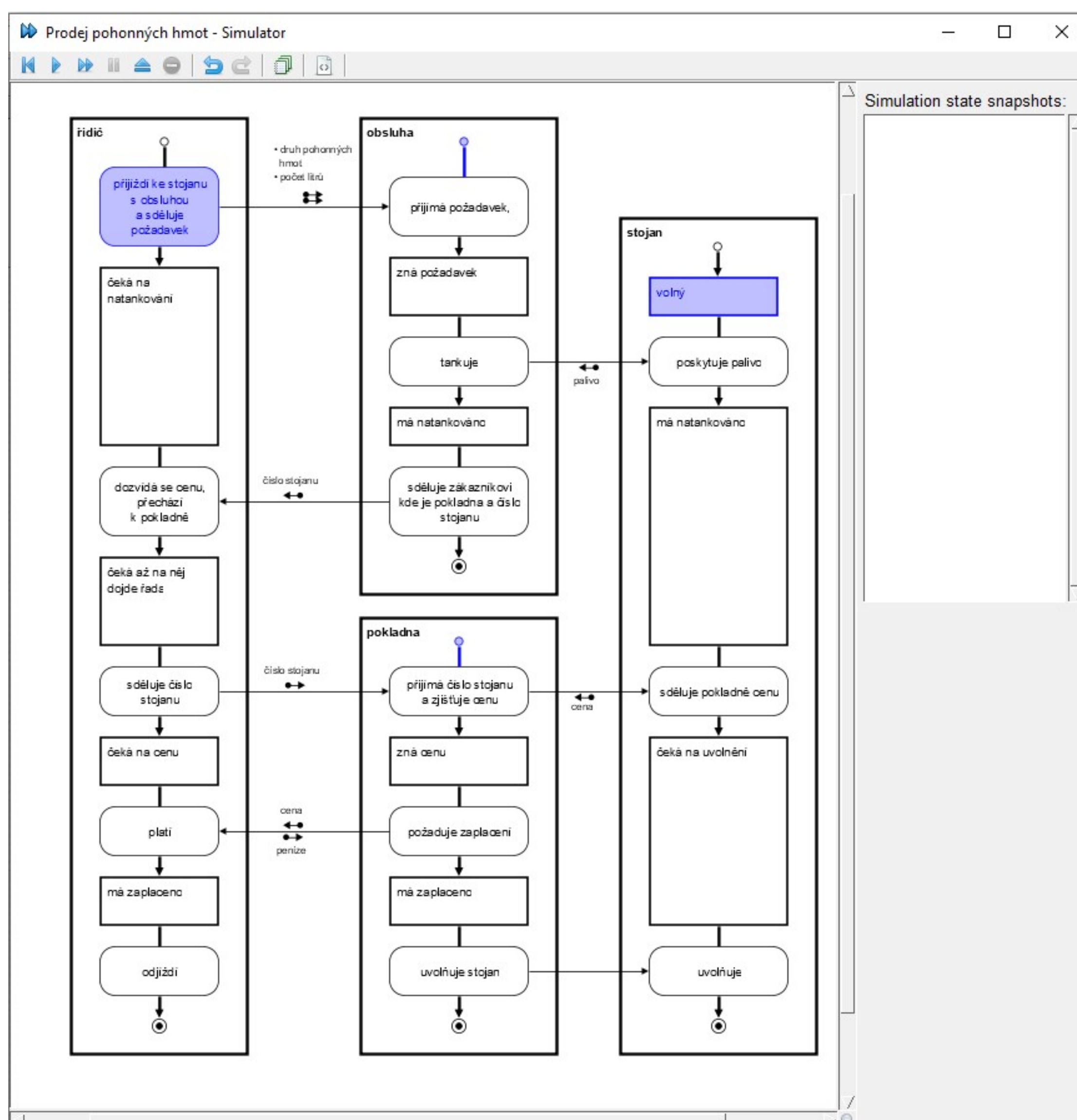
Principy simultánnosti a závislosti společně tvoří rámec pro formalizaci sémantiky BORM ORD. Poskytují vodítko pro interpretaci diagramů procesů, specifikaci závislostí úloh a simulaci provádění procesů. Začleněním těchto principů do formálních základů metodiky BORM autoři zvýšili vyjadřovací schopnost, srozumitelnost a praktickou využitelnost metodiky v kontextu analýzy a návrhu systémů.

4.2 Existující implementace

4.2.1 Craft.CASE

Craft.CASE® je originální český nástroj pro modelování a analýzu CASE, který podporuje metodu BORM. Nástroj byl vyvinut ve společnosti e-Fractal s.r.o. na zakázku mezinárodní poradenské a konzultační firmy Deloitte & Touche, která je uživatelem a spolutvůrcem metody BORM. Program byl implementován v prostředí VisualWorks/Smalltalk a byl prvotně určen pro použití ve Windows 2000 a XP. [11]

Tento nástroj bohužel nepodporuje nastavení input a output podmínek, tedy jeho využití pro simulaci procesů je značně limitované. Mezi další jeho nevýhody patří v dnešní době již zastaralý a pro laika velmi nepřehledný user interface.

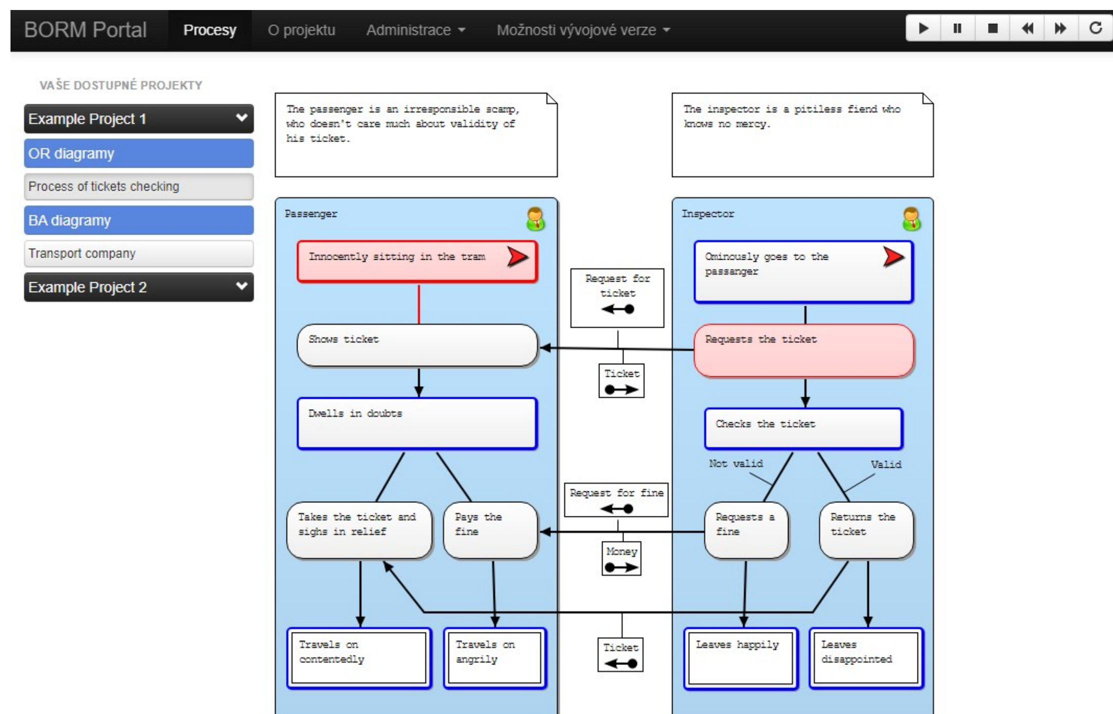


■ **Obrázek 4.1** Craft.CASE simulace

4.2.2 Portál pro optimalizaci BORM procesů

Portál pro optimalizaci BORM procesů [12] byl vyvinut panem Michalem Baldou v rámci jeho bakalářské práce v roce 2013. Tento portál bohužel také nepodporuje nastavení input a output podmínek. Oproti Craft.CASE však disponuje přívětivějším a přehlednějším GUI. Jeho nevýhody ve srovnání s Craft.CASE jsou ale znatelné:

- Složitější prvotní instalace – jedná se o webový portál, tedy uživatel musí prvotně spustit Squeak server, na který se pak připojí,
- nemožnost editace diagramů přímo v portálu. Uživatel musí nejprve vytvořit model v nástroji OpenCABE, teprve poté ho může nahrát na server a odsimulovat.



Obrázek 4.2 BORM portál simulace

Kapitola 5

Využité technologie

5.1 Smalltalk

Smalltalk je objektově orientovaný jazyk, který byl poprvé vydán v roce 1972. Mezi jeho přednosti patří jednoduchá syntaxe – všechny pravidla by se vešly na obálku běžného dopisu. [13]

5.2 Pharo

Pharo je programovací jazyk a vývojové prostředí, které upřednostňuje jednoduchost a zpětnou vazbu v reálném čase. Striktně dodržuje objektově orientované principy a eliminuje potřebu konstruktorů, deklarací typů, rozhraní nebo primitivních typů. Navzdory své jednoduchosti zůstává Pharo výkonným jazykem s kompaktní syntaxí.

Vývojové prostředí poskytované jazykem Pharo zajišťuje okamžitou zpětnou vazbu v průběhu celého procesu vývoje, testování i ladění. Díky tomu odpadá nutnost časově náročných kroků kompilace a nasazení, a to i v produkčním prostředí.

Jednou z pozoruhodných vlastností jazyka Pharo jsou jeho možnosti ladění, které vývojářům umožňují procházet kód krok za krokem, restartovat a vytvářet metody za běhu. Je to praktický nástroj pro zvýšení produktivity a efektivity řešení problémů.

[14]

Pharo udržuje oddaná komunita přispěvatelů, díky nimž neustále přibývají četné frameworky a knihovny. Je plně open-source, uvolněný pod licencí MIT a dostupný na GitHubu, kde ho může kdokoli využít a přispívat do něj.

5.2.1 SUnit testing

SUnit je výkonný testovací framework, který podporuje vytváření a nasazení testů. Jak lze vytušit z jeho názvu, SUnit je zaměřen na testy jednotek, ale lze jej použít i pro integrační a funkční testy. [15]

5.3 OpenPonk

OpenPonk je metamodelovací platforma a modelovací workbench implementovaný v dynamickém prostředí Pharo, který je zaměřen na podporu činností spojených se softwarovým a obchodním inženýrstvím jako je modelování, simulace, generování zdrojových kódů atd.[16]

V době psaní této bakalářské práce OpenPonk podporuje modelování modelů BORM, OntoUML, UML, Petriho sítí, konečných automatů a Markovových řetězců. Markovovy řetězce a konečné automaty nabízí své vlastní simulátory, Petriho sítě mají simulátor založený na balíčku OpenPonk–ModelSimulation.

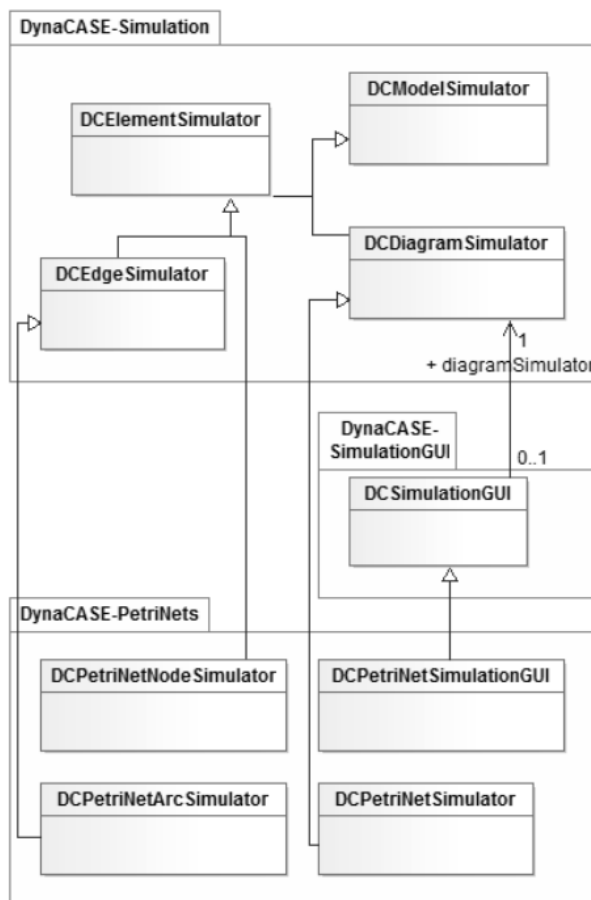
5.3.1 OpenPonk Simulation

Podporu simulace v platformě OpenPonk zajišťuje balíček OpenPonk–ModelSimulation, který byl prvně vyvinut panem Janem Blizničenkem v rámci jeho bakalářské práce [17]. Tento balíček obsahuje několik způsobů pro simulaci v podstatě jakéhokoli diagramu, a to především tím, že je jednoduše rozšiřovatelný.

Pro simulace je balíček OpenPonk–ModelSimulation využíván Petriho sítěmi, jejichž implementace byla také značnou inspirací pro tuto bakalářskou práci.

5.3.2 Petriho sítě na platformě OpenPonk

Implementace Petriho sítí, jakožto první model využívající právě balíček OpenPonk–ModelSimulation, sdílí s mojí implementací simulace BORM ORD velkou část aplikačního designu. Tento design je právě založen na dědičnosti od tříd balíčku OpenPonk–ModelSimulation, která je znázorněna na obrázku 5.1.



Obrázek 5.1 Třídový diagram implementace simulace Petriho sítí [17]

Část II

Praktická část

Kapitola 6

Analýza

6.1 Funkční požadavky

Jako funkční požadavky pro simulaci BORM ORD jsem zvolil:

- F1 – aplikace bude simulovat průchod diagramem dle pravidel stanovených v této práci, tedy rozšířenými pravidly zmíněnými v sekci 4.1,
- F2 – simulace se bude zobrazovat pomocí barev v diagramu, a to jednoznačně.

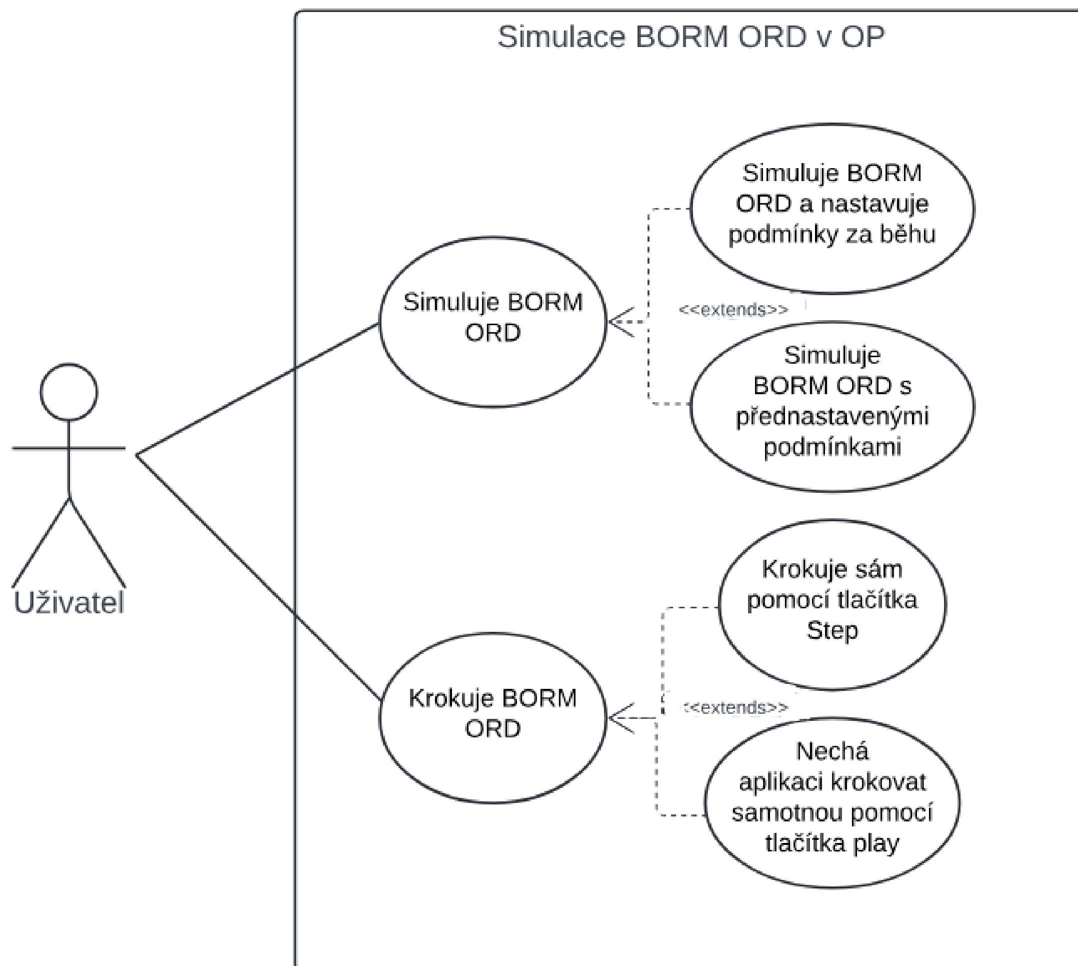
6.2 Nefunkční požadavky

Jako nefunkční požadavky pro simulaci BORM ORD byly zvoleny:

- Simulace bude jednoduše pochopitelná a ovladatelná,
- pravidla simulace (například input a output podmínky) budou jednoduše upravitelné a rozšiřitelné,
- aplikace bude mít podobný GUI k již naimplementovaným simulačním GUI na platformě OpenPonk.

6.3 Případy užití

Případy užití jsou znázorněné na use–case diagram na obrázku 6.1.



Obrázek 6.1 Use case diagram aplikace

Kapitola 7
Implementace

7.1 Logika simulace

Logika simulace je značně inspirována článkem „Towards Formal Foundations for BORM ORD Validation and Simulation“[10], který byl již diskutován v kapitole BORM ORD simulace. Během testování a užívání ale vyšlo najevo, že i tato definice pro simulaci má menší nedostatky. V tomto článku se počítalo při simulaci pouze s přechody a na komunikace se nebral ohled. Komunikace jsou z jedna z nejdůležitějších stránek BORM ORD a proto jsem se rozhodl pozměnit pravidla simulace přidáním samostatné input podmínky pro simulaci..

7.1.1 Algoritmus pro procházení diagramem

Jako správný a jednoduchý algoritmus jsem vyhodnotil BFS využívající svého OPBormToken tokenu. Pomohla mi možnost využití části simulace již zabudované v balíku OpenPonk–Simulation. Jednoduché vysvětlení využitého algoritmu je:

- Nejprve se přidají při spuštění simulace tokeny počátečním stavům. Počáteční stavy jsou vyhodnoceny jako stavy, do kterých nevedou žádné hrany nebo jsou označené *initial*.
- Pokaždé, když aplikace dělá krok, algoritmus projde všechny stavy a aktivity a podívá se, které z nich mají splněné input (jak pro přechody, tak i komunikace) a output podmínky.
- Všechny stavy a aktivity, co splňují uvedené input a output podmínky, udělají krok, tzn. že pošlou své tokeny / kopie svých tokenů dál po všech možných hranách z těchto stavů a aktivit vedoucích.

7.1.2 Input a output podmínky

Input a output podmínky jsou základem komplexnější simulace BORM ORD. Rozhodl jsem se pro možnost nastavení podmínek na tři hodnoty: *Any*, *All* a *None*. Tyto tři hodnoty by měly zahrnout většinu možných požadavků logiky simulace.

- Input podmínky

  Input podmínky řeší, zda může simulace pokračovat z dané aktivity či stavu dál, podle toho, zda získaly všechny potřebné tokeny. Při nastavení *All* se čeká, dokud daný stav nebo aktivita nezískají tokeny ze všech příchozích hran. Nastavení *Any* pustí jakýkoli token dál. Nastavení *None* funguje podobně jako nastavení *Any* a bylo přidáno kvůli rozdělení input podmínek. Podrobněji se tímto zabývá sekce „Rozdělení input podmínek“7.1.3.

- Output podmínky

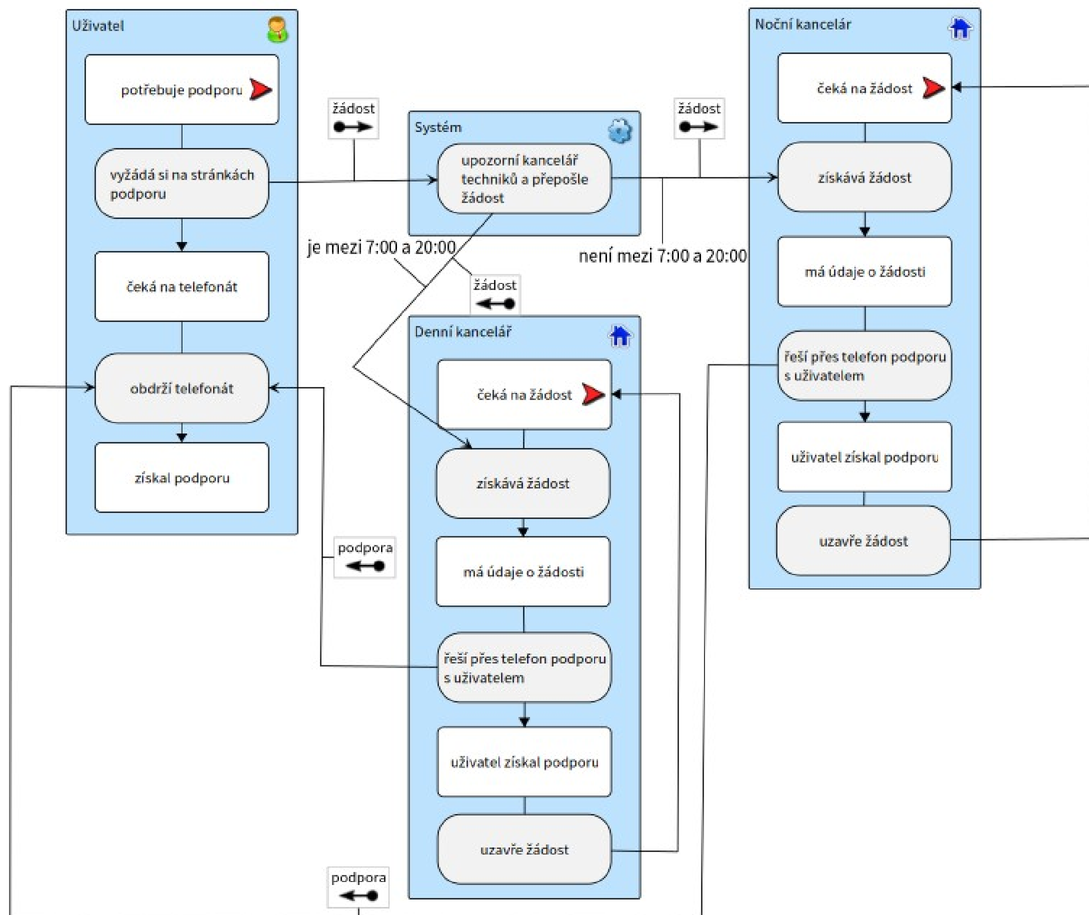
  U output podmínek se řeší podmínky náležící výchozím hranám. V případě nastavení *None* se podmínky ignorují, tedy se simulace chová jako by všechny tyto podmínky byly nastaveny na hodnotu *true*. Nastavení *All* pustí token dál pouze v případě, pokud všechny podmínky mají hodnotu *true*, zatímco nastavení *Any* pustí token dál po právě těch hranách, které nemají podmínku nebo mají podmínkou nastavenou na hodnotu *true*.

7.1.3 Rozdělení input podmínek

Prvně jsem implementoval simulaci tak, že existovala pouze jedna input podmínka, a ke komunikacím jsem se choval stejně jak k přechodům. Toto ale nebyl dobrý přístup, jelikož v mnoha modelech může být vyžadováno získat signál ze všech přechodů ale pouze z jedné komunikace. Proto jsem se rozhodl input podmínku rozdělit na input podmínku pro přechody a input podmínku pro komunikace.

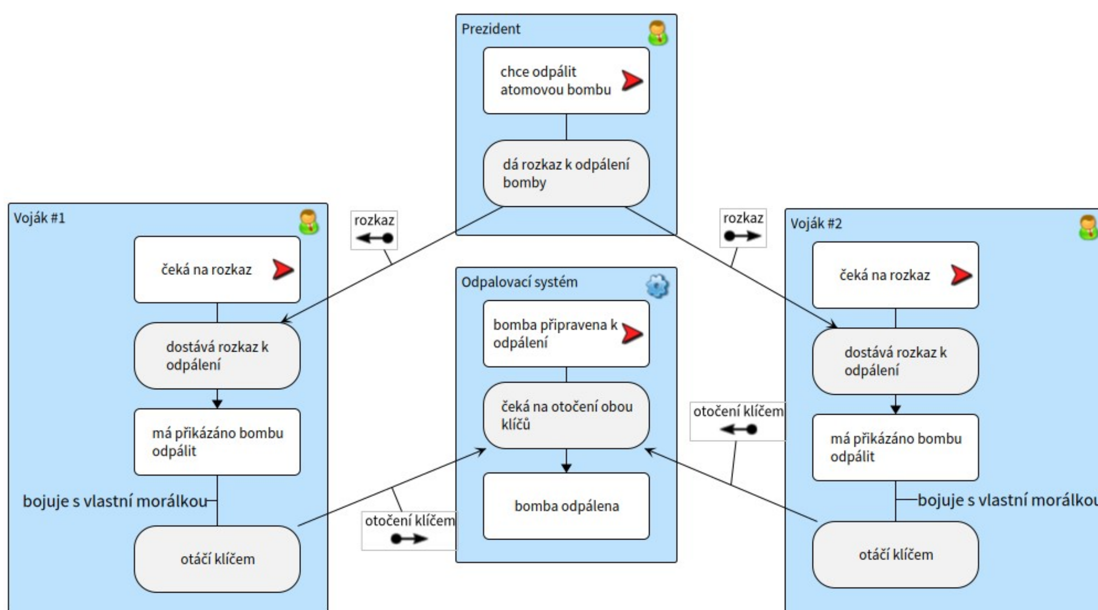
Na obrázku 7.1 je jednoduchý diagram, který znázorňuje problém, který může nastat, pokud je simulace implementovaná pouze s jednou input podmínkou zahrnující jak přechody, tak komunikace. Uživatel zde žádá přes systém podporu, systém upozorní dle toho, kolik je hodin, buď denní nebo noční kancelář, od které pak obdrží uživatel telefonát. V tomto případě je uživateli jedno, ze které kanceláře mu zavolají, ale nutnoui podmínkou toho, aby pokračoval z aktivity *obdrží telefonát* do stavu *získal podporu*, je obdržení alespoň jednoho telefonátu.

Pokud by v tomto případě existovala pouze jedna input podmínka pro přechody i komunikace, znasmenalo by to, že by se uživatel v této aktivitě pokaždé „zasekl“. Přidáním možnosti nastavení podmínek zvlášť pro přechody a zvlášť pro komunikace, lze nastavit podmínku pro komunikace *Any* a podmínku pro přechody *All*. Pak bude průchod diagramem fungovat dle očekávání.



Obrázek 7.1 Příklad pro input podmínky – *Any* a *All*

Tente diagram můžeme porovnat s diagramem odpálení bomby, znázorněném na obrázku 7.2. Zde pro odpálení bomby, tak jak mnoho hollywoodských filmů ukazuje, je potřeba aby dva vojáci zároveň otočili klíčem. Zde je tedy input podmínka pro komunikace nastavena na *All*. Pokud si tedy alespoň jeden z těchto vojáků rozmyslí odpálení bomby, tokeny nikdy nepokročí z aktivity „čeká na otočení obou klíčů“ a bomba se neodpálí.



Obrázek 7.2 Příklad pro input podmínky – *All* a *All*

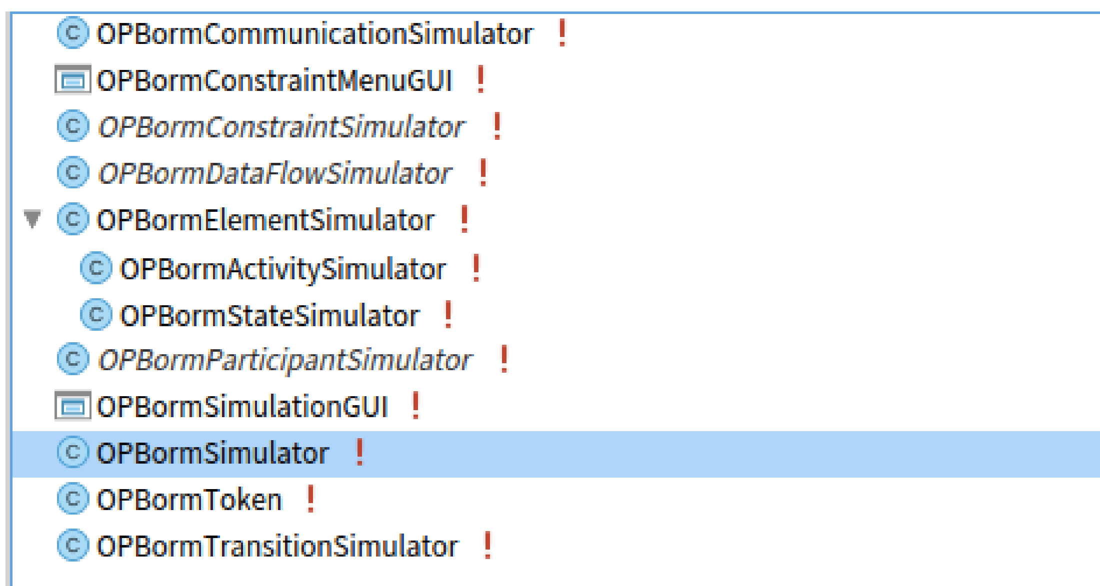
Rozdělení input podmínek představuje ale také jiné komplikace. V první verzi simulace, kde nebyly input podmínky rozděleny, nebyla potřeba možnost nastavení input podmínky na hodnotu *None*. Tuto možnost jsem přidal z důvodu zachování předchozí funkčnosti, kde při nastavení *Any* stačil jakýkoli token bez ohledu na to, zda přišel po přechodu či komunikaci. Po rozdělení input podmínek a nastavení obou na hodnotu *Any* se bude vždy čekat na alespoň jeden token, který přišel po přechodu a další nejméně jeden, který přišel po komunikaci (pokud tedy do dané aktivity nebo stavu vede alespoň jedna komunikace a jeden přechod).

Z těchto důvodů bylo tedy potřeba přidat možnost nastavení podmínek na hodnotu *None*. Pokud jsou obě input podmínky nastaveny na tuto hodnotu, token může pokračovat dále, hned jak dorazí. Hodnotu *None* lze kombinovat s ostatními možnostmi nastavení. Například když input podmínka pro přechody má hodnotu *Any* a input podmínka pro komunikace má hodnotu *None*, token může jít dál, právě když získal alespoň jeden token přes přechod, bez ohledu na zisk tokenu z komunikace.

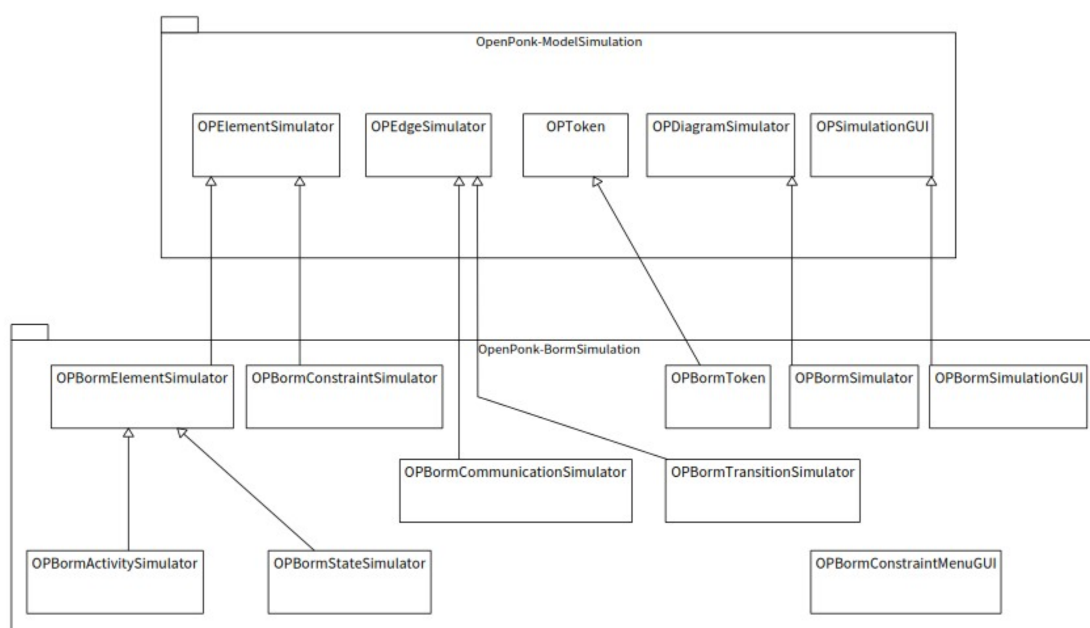
7.2 Hierarchie a architektura

Architektura tříd vypadá tak, jak je znázorněna na obrázcích 7.3 a 7.4 díky využití balíku OpenPonk–ModelSimulation. OpenPonk–ModelSimulation vyžaduje vytvoření simulátoru pro každou součást modelu, který je následovně na ni namapován. Tyto simulátory poté řeší logiku jednotlivých kroků při simulaci. Každý simulátor má metody *isSteppable* a *isSimulable*, které poté sdělují třídám balíčku OpenPonk–ModelSimulation, jak s nimi zacházet. Mezi nejdůležitější simulátory patří OPBormSimulator, který zastupuje celý diagram, a OPBormElementSimulator, který zastupuje stavy a aktivity.

Další důležité třídy jsou OPBormSimulationGUI a OPBormConstraintMenuGUI, které se starají o user interface.



Obrázek 7.3 Třídy package OPBormSimulation



Obrázek 7.4 Dědičnost mezi třídami balíčků ModelSimulation a BormSimulation

7.3 Nutné změny BORM ORD modelu

BORM ORD model byl na platformě OpenPonk napsán dle definovaných BORM pravidel, ale při jeho prvotní implementaci nebyla brána v potaz simulace. To znamenalo potřebu změny modelu tak, aby podporoval input a output podmínky. Každému BormProcessNode (tedy nadtřídě BormActivity a BormState) se tedy přidaly get a set metody inputTransitionSimulationCondition, inputCommunicationSimulationCondition a outputSimulationCondition, které nastavují a vrací

nastavenou instanci potomků třídy BormNodeSimulationCondition.

7.4 OPBormSimulator

Třída OPBormSimulator dědí od třídy OPDiagramSimulator z balíku OpenPonk–Simulation. Její hlavní role je nastavení krokování a zajištění správné inicializace tokenů v příslušných počátečních stavech.

Metoda *setDefaultStepping* zobrazená ve výpisu kódu 7.1 říká třídě OPDiagramSimulator, jak se má diagram simulovat. V kódu lze vidět, že krokovat se budou jen ty simulátory, které vrací v metodě *isSteppable* hodnotu true, dále že jejich pořadí může být náhodné a že při každé iteraci mají udělat kroky všechny simulátory, které tuto možnost mají.

```
1  setDefaultStepping
2    "sets default stepping settings"
3
4    super setDefaultStepping.
5    self
6      fromSteppableElements;
7      orderAny;
8      selectAll;
9      resetActions;
10     addAction: [ :item | item step ]
```

Výpis kódu 7.1 Metoda pro nastavení správného krokování

7.5 OPBormElementSimuator

OPBormElementSimulator zajišťuje logiku simulace stavů a aktivit. Je to nadtřída OPBormActivitySimulator a OPBormStateSimulator. Hlavní úkol OPBormElementSimuator je vyhodnotit, zda se má sám krokovat, tedy zda je „steppable“. Aby simulátor zjistil, zda toto splňuje, musí vyhodnotit, zda jsou splněny jak input tak output podmínky. Input podmínka je splněna pouze tehdy, když jsou splněny input podmínky pro dané přechody a komunikace. Tato základní logika je znázorněna ve výpisu 7.2.

```
1  isSteppable
2
3      ↑ self inputCondition and: [ self outputCondition ]
4
5
6  inputCondition
7
8      ↑ self inputCommunicationCondition and: [
9              self inputTransitionCondition ]
```

Výpis kódu 7.2 Metody využité pro rozhodnutí krokování

Hlavní logiku řešení tedy obsahují metody *outputCondition*, *inputTransitionCondition* a *inputCommunicationCondition*. Metody *inputTransitionCondition* a *inputCommunicationCondition* si jsou navzájem velmi podobné. Jediný rozdíl mezi nimi je to, zda sledují příchozí komunikace nebo příchozí přechody a kontrolují počet jim přidružených tokenů.

```
1  inputTransitionCondition
2
```

```
3    (model inputTransitionSimulationCondition nameAsInput = 'Any' and: [
4       self incomingTransitions isNotEmpty ]) ifTrue: [
5      ↑ self transitionTokenCount > 0 ].
6    model inputTransitionSimulationCondition nameAsInput = 'All' ifTrue: [
7      ↑ self transitionTokenCount >= self incomingTransitions size ].
8    ↑ true
```

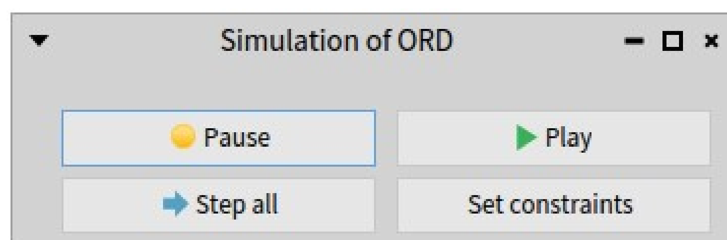
Výpis kódu 7.3 Input podmínky pro přechody

Metoda *outputCondition* oproti předchozímu sleduje výchozí hrany a jejich podmínky. Pokud vyhodnotí, že dle nastavení output podmínky by měl být simulátor krokovatelný, vrátí hodnotu metody *baseCondition*. Ta řeší, zda daný simulátor má výchozí hrany a má alespoň jeden token. Pokud by tyto podmínky nebyly splněny, nedává smysl simulátor krokovat.

```
1  outputCondition
2
3    | constraintSimulator |
4    model outputSimulationCondition nameAsOutput = 'Any' ifTrue: [
5      self outgoing do: [ :each |
6        each model constraint ifNotNil: [
7          constraintSimulator := self diagramSimulator simulatorOf:
8                                    each model constraint.
9          constraintSimulator value ifTrue: [ ↑ self baseCondition ] ] ].
10     ↑ false ].
11   model outputSimulationCondition nameAsOutput = 'All' ifTrue: [
12     self outgoing do: [ :each |
13       each model constraint ifNotNil: [
14         constraintSimulator := self diagramSimulator simulatorOf:
15                                   each model constraint.
16         constraintSimulator value ifFalse: [ ↑ false ] ] ].
17     ↑ self baseCondition ].
18   ↑ self baseCondition
```

Výpis kódu 7.4 Output podmínky pro přechody

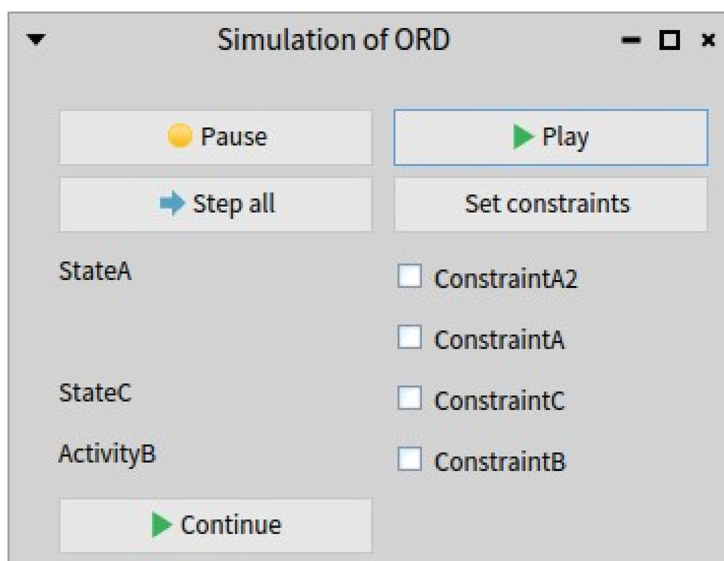
7.6 OPBormSimulationGUI



Obrázek 7.5 Okno simulace

Jako základ pro grafický interface je využita třída OPSimulationGUI podobně jako v simulaci Petriho sítí. Pro simulaci BORM byla ale odebrána tlačítka load a save state, protože při simulaci BORM ORD postrádají smysl (tato tlačítka jsou důležitá při ukládání stavu Petriho sítí, kde tokeny jsou přímo součástí modelu).

Jak vyplývá z obrázku 7.5, zbývají čtyři tlačítka: Pause, Play, Step all a Set constraints. Step all udělá všechny možné kroky v danou chvíli a zastaví se. Play udělá všechny možné kroky každou vteřinu a Pause simulaci zastavuje. Set constraints otevře okno pro přednastavení podmínek, o kterém se dále píše v sekci OPBormConstraintMenuGUI 7.7.



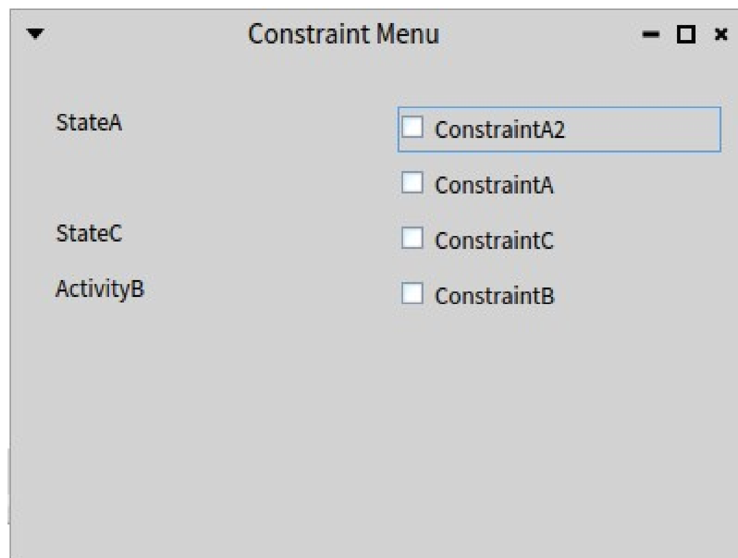
Obrázek 7.6 Okno simulace s podmínkami k nastavení

Pokud dojde k tomu, že simulátor se dostane do stavu nebo aktivity, ze které vede hrana s podmínkou v situaci, kdy podmínky nebyly přednastaveny, simulace se zastaví. Podmínka je poté připsána do listu podmínek v okně simulace vedle jména stavu nebo aktivity, z nichž vychází hrana, na které se tato podmínka nachází. Pro toto zobrazení jsem se rozhodl po otestování několika verzí implementace – nejprve existovalo jen OPBormConstraintMenuGUI pro přednastavení podmínek. Toto řešení ale neumožňovalo nastavení podmínek za chodu, což značně limitovalo využitelnost simulace. V další iteraci se již daly podmínky nastavovat za běhu, ale byla k tomu využita „pop-up“ okna, podobně jako v nástrojích Craft.CASE a v portálu pro optimalizaci BORM procesů. Tato iterace ale také nebyla ideální a to z důvodu, že když simulátor narazil během jednoho kroku na více podmínek, otevřená okna nebyla přehledná.

Prvně byly podmínky v jednom listu v okně simulace. To ale představovalo další problém – pokud by na dvou nebo více místech v modelu byly stejně pojmenované podmínky, např. podmínky „finished“ a „did not finish“, uživatel by nemohl rozeznat, která podmínka v listu odpovídá dané podmínce v diagramu. To by pravděpodobně nastalo v mnoha diagramech. Proto jsem tedy zvolil možnost, kde na levé straně jsou vypsány všechny stavy a aktivity a na pravé straně jsou pro každý stav a aktivitu vypsány podmínky ležící na hranách, které z nich vedou.

7.7 OPBormConstraintMenuGUI

Constraint Menu, neboli menu podmínek, se otevře, pokud uživatel stiskne v okně simulace tlačítko *Set constraints*. V tomto menu je list podmínek, formátovaný podobně jako v okně simulace. Důležité je zmínit, že jakmile se otevře toto menu pro přednastavení podmínek, podmínky se již nebudou zobrazovat v okně simulace.



Obrázek 7.7 Okno pro přednastavení podmínek

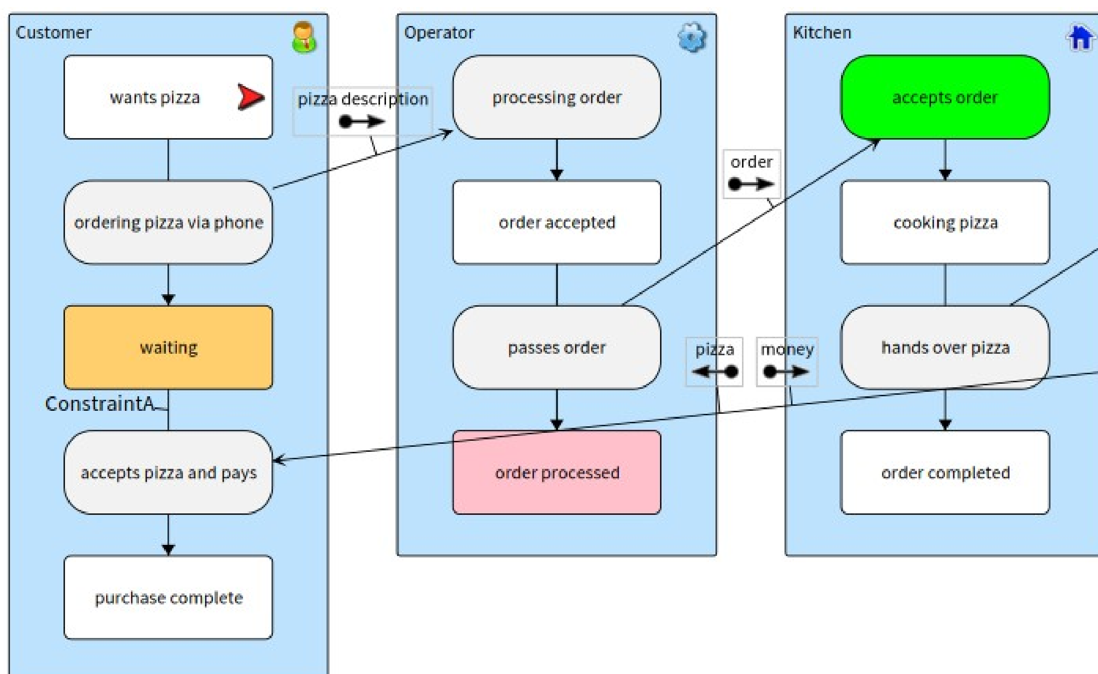
7.8 Zobrazení simulace na diagramu

Je důležité, aby uživatel při simulaci jednoznačně viděl, jak simulace postupuje skrz diagram. Pro tento úkol jsem se rozhodl využití vybarvování tak, aby bylo jednoduše možné pospat všechny okolnosti ve kterých se stavy a aktivity mohou nacházet.

Barev využitých pro znázornění okolností stavů je 5:

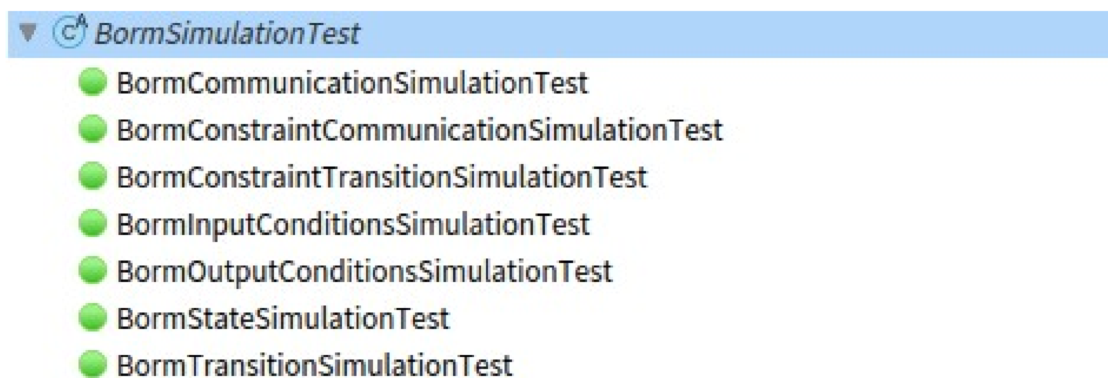
- Základní barvy – bílá pro stavy, lehce šedá pro aktivity. Tyto barvy symbolizují, že daný stav ani aktivita nedisponuje žádnými tokeny.
- Zelená – barva, která znázorňuje, že daný stav či aktivita mají alespoň jeden token a splňují jak input tak output podmínky, tedy jsou připraveny provést krok a token(y) poslat dál.
- Oranžová – v aktivitě či stavu se nachází alespoň jeden token, ale buď output nebo input podmínky nejsou splněny. Tedy je nutné dále čekat a není možné zatím provést krok.
- Růžová – stav či aktivita má splněné podmínky, nevedou z nich ale žádné další hrany. Tedy zde cesta tokenů končí.

Na obrázku 7.8 je uveden příklad použití zvolených barev. Stav *order processed* je růžový, jelikož z něho nevedou žádné hrany a jeho input podmínky jsou splněny. Stav *waiting* je oranžový, jelikož podmínka *ConstraintA* je nastavená na false, tedy nejsou splněny jeho output podmínky a nemůže pokračovat posláním tokenu dál. Aktivita *accepts order* je zelená, protože má token a její input a output podmínky jsou splněny. Tedy může svůj token při dalším kroku poslat dál.



Obrázek 7.8 Příklad různých barev dle okolností

Kapitola 8
Testování



Obrázek 8.1 Třídy testů BORM simulace

Testování balíku simulace BORM ORD je důležité pro vývoj, aby během případného rozšiřování funkčnosti BORM ORD a jeho simulace byla jisté, že správnost a funkčnost kódu je stálá. Tyto testy umožňují rychle otestovat základní i komplexnější funkčnosti simulace BORM ORD a usnadňují budoucí vývoj.

Kromě Unit testů napsaných v rámci vývoje byla i předběžná verze BORM ORD simulace otestována studenty BI–ZPI (předmět Základy procesního inženýrství na Fakultě informačních technologií ČVUT).

8.1 BormSimulationTest nadtřída

BormSimulationTest je abstraktní nadtřída pro všechny testovací třídy balíčku OPBormSimulation. Dědí od další abstraktní třídy BormTest z balíku OpenPonk-BormModel, která dědí od SUnit třídy TestCase. Výhodou dědičnosti od třídy BormTest je možnost využití všech jejích metod pro vytvoření testovacího BORM ORD modelu, čímž se vyhýbáme opakování kódu.

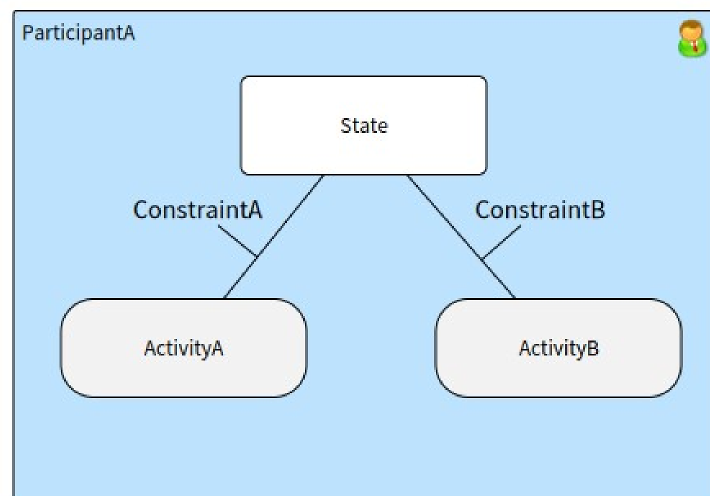
Dále třída BormSimulationTest obsahuje tři metody, z nichž metody *controllerClass* a *exampleModel* jsou zodpovědností jejich potomků, tedy jejich tělo obsahuje pouze `^ self subclassResponsibility`. Třetí metoda je metoda setUp, která zavolá `super setUp` a poté nastaví model získaný z podtřídy a vytvoří pro něj simulátor.

8.2 Jednoduché Unit testy

Nejspíše nejjednoduší test BormStateSimulationTest se zabývá pouze správností přidělení tokenů, tedy, že stav bez jakýchkoli vstupujících přechodů či komunikací při vytvoření simulace získá právě jeden token.

Další poměrně jednoduché testy BormTransitionSimulationTest a BormCommunicationSimulationTest ověřují, zda se při simulaci správně prvně přiřadí tokeny a poté se po jednom kroku správně přesunou po přechodu, respektive po komunikaci.

8.3 Testování output podmínek



Obrázek 8.2 Diagram modelu pro testování output podmínek

V BormOutputConditionsSimulationTest se testuje chování simulace při všech možných kombinacích nastavení chování stavu a nastavení hodnot podmínek na přechodech, které vycházejí z daného stavu. Na obrázku 8.2 lze vidět model, který se vytvoří pomocí exampleModel metody uvedené ve Výpisu kódu 8.1.

8.4 Testování input podmínek

Podobně jako u testování output podmínek je vytvořen složitější model. Tento model je zachycen na obrázku 8.3 a je navržen tak, aby zachytil všechny eventuality, které se mohou na inputu jedné aktivity vyskytnout. Důležitá je zde aktivita *ActivityA*, do které vedou dva přechody a jedna komunikace. Každá z těchto hran má právě jednu svoji podmínku, která se v testech různě nastavuje.

V této třídě také z důvodu zjednodušení kódu existuje metoda *endStateValueForGiven:b:c:numberOfSteps:*. První tři parametry této metody nastavují hodnoty podmínek na vstupních hranách, čtvrtý parametr pak udává, kolik kroků se má provést. Návratová hodnota této metody je počet tokenů v posledním stavu, na obrázku 8.3 znázorněném jako StateEnd. Jak se tato metoda využívá, lze vidět v ukázce kódu 8.3.

```
1  exampleModel
2
3      model := self emptyModel.
4      participantA := self emptyParticipant.
5      state := self emptyState.
6      activityA := self emptyActivity.
7      activityB := self emptyActivity.
8      transitionA := self emptyTransition.
9      transitionB := self emptyTransition.
10     constraintA := self emptyConstraint.
11     constraintB := self emptyConstraint.
12     transitionA from: state to: activityA.
13     transitionB from: state to: activityB.
14     model
15         add: participantA;
16         add: state;
17         add: transitionA;
18         add: constraintA.
19     participantA add: activityA.
20     participantA add: activityB.
21     state
22         add: transitionB;
23         add: constraintB.
24
25     transitionA constraint: constraintA.
26     transitionB constraint: constraintB.
27
28     ↑ model
```

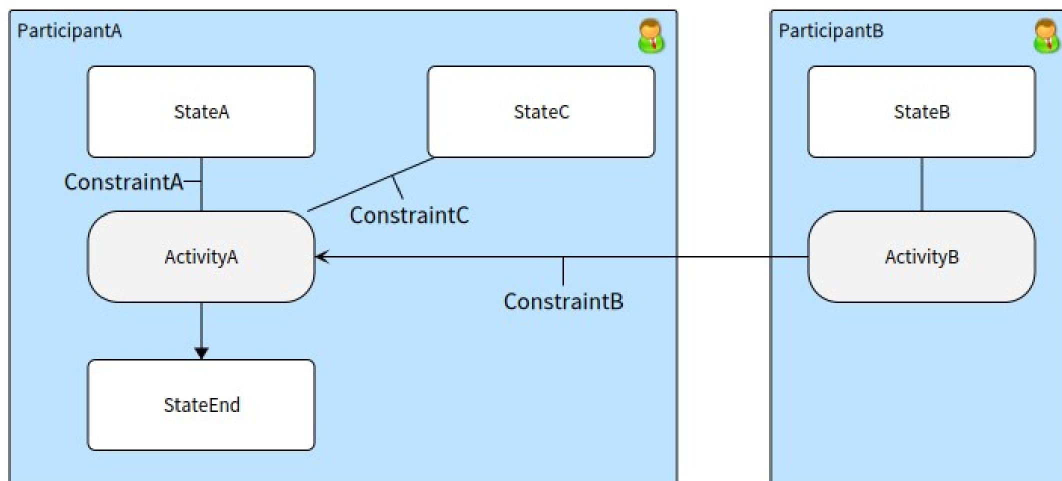
Výpis kódu 8.1 exampleModel metoda pro vytvoření modelu pro testy output podmínek

```
endStateValueForGiven: constraintAValue b: constraintBValue
                        c: constraintCValue numberOfSteps: numberOfSteps

    | constraintSimulatorA constraintSimulatorB
      constraintSimulatorC endStateSimulator |
    endStateSimulator := simulator simulatorOf: stateEnd.
    constraintSimulatorA := simulator simulatorOf: constraintA.
    constraintSimulatorB := simulator simulatorOf: constraintB.
    constraintSimulatorC := simulator simulatorOf: constraintC.
    constraintSimulatorA value: constraintAValue.
    constraintSimulatorB value: constraintBValue.
    constraintSimulatorC value: constraintCValue.
    1 to: numberOfSteps do: [ :index | simulator step ].
    ↑ endStateSimulator tokenCount
```

Výpis kódu 8.2 Helper metoda pro testování input podmínek

```
testAllTwo

    activityA inputTransitionSimulationCondition:
        BormNodeAllNeighborSimulationCondition new.
    self
        assert: (self
                    endStateValueForGiven: true
                    b: true
                    c: true
                    numberOfSteps: 3)
        equals: 1
```

Výpis kódu 8.3 Test input all podmínek



Obrázek 8.3 Diagram modelu pro testování input podmínek

Kapitola 9

Případová studie

Jako případovou studii jsem se rozhodl využít existující semestrální práce studentů předmětu BI–ZPI. Tyto práce zahrnují BORM ORD modely, nejčastěji namodelované v nástroji OpenCABE. Několik takových diagramů jsem tedy znovu namodeloval využívajíc platformu OpenPonk tak, abych mohl diskutovat přínos simulace, která nebyla v nástroji OpenCABE implementována.

Dále jsem se také rozhodl porovnat mnou navrženou simulaci se simulací pomocí portálu pro optimalizaci BORM procesů, který jsem popisoval v sekci kapitoly BORM ORD simulace 4.2.2.

Ze 17 semestrálních jsem prací vybral 10 BORM ORD diagramů, v nichž jsem našel díky simulaci 13 chyb. Chyby v těchto diagramech byly sémantické, tedy se vyznačovaly tím, že diagram nedával dobrý smysl. Syntaxní chyby jsou již odchyceny při modelování. Jak OpenPonk, tak OpenCABE dovolují modelovat pouze syntaxně dobře. V další sekci představím dva z diagramů a budu s jejich pomocí diskutovat přínos simulace. Oba dva diagramy jsou přiloženy ve formátu .opp tak, aby si je čtenář mohl sám spustit na platformě OpenPonk.

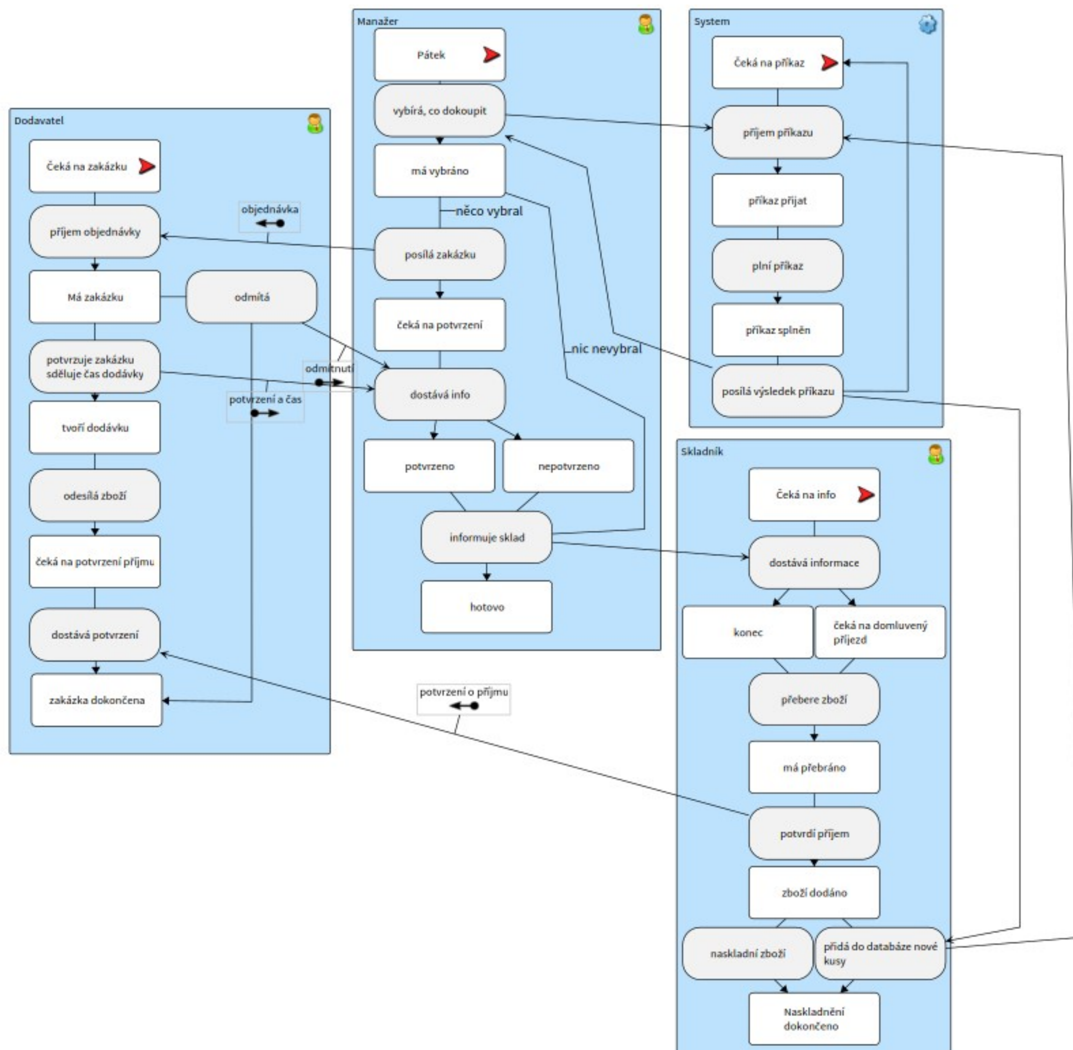
9.1 Výhody využití simulace

9.1.1 Nalezení chyb – diagram eshopu

Diagram eshopu, namodelován v rámci semestrální práce pro předmět BI–ZPI, je dobrým příkladem, jak simulace může pomoci při hledání chyb. Bohužel autoři ve své semestrální práci neuvedli své jména. Na první pohled vypadá diagram logicky a jak bylo zmíněno, již víme že v syntaxi, díky tomu že byl modelován v programu OpenCABE, chyby nejsou. Když ale je diagram prokrokován simulací, zjistíme hned několik problémů.

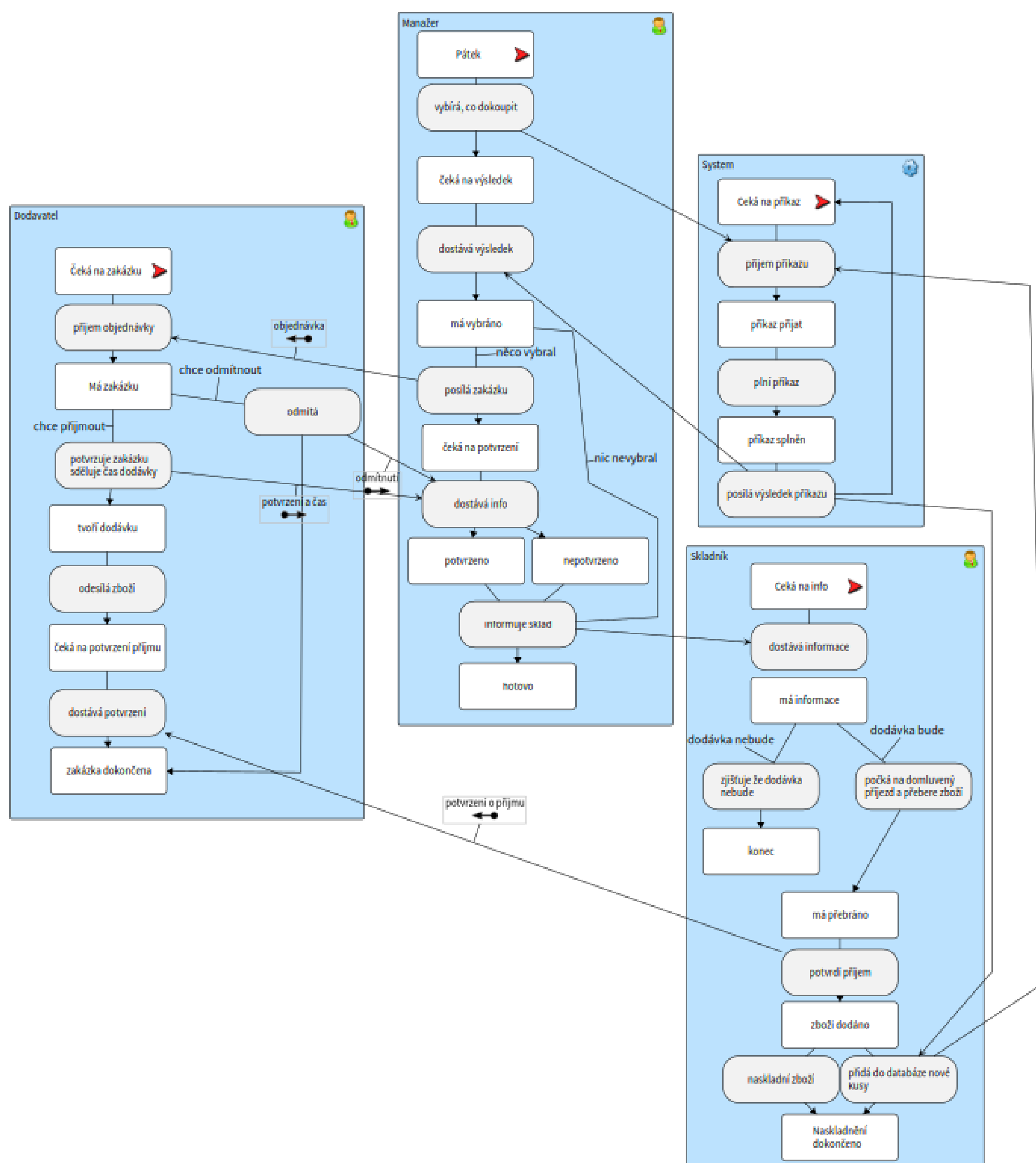
Nejjednodušší problémy k vyřešení jsou chybějící podmínky. Je zřejmé že autoři zapomněli na podmínky u přechodů vedoucích ze stavu „Má zakázku“. Toto lze jednoduše opravit, ale není těžké takovouto chybu při modelování přehlédnout.

Další, poněkud složitější chyba, se odehrává v aktivitě manažera „vybírá co dokoupit“. Zde nastane situace, kde se simulace zasekne. Zasekne se, protože input podmínka aktivity musí být nastavena *All* jak pro přechody, tak pro komunikace. Kdyby byla nastavena jinak, mohlo by nastat, že proces manažera bude pokračovat dál do stavu „má vybráno“, aniž by získal výsledek příkazu od systému. V takovém případě nedává diagram dobrý smysl. Při nastavení obou input podmínek na *All* nastane situace, kde aktivita „vybírá co dokoupit“ nikdy nepostoupí, jelikož proto, aby poslala dál tokeny, musí získat token z komunikace. Token z komunikace ale nezíská, protože proces systému nikdy nepokročí přes aktivitu „příjem příkazu“, která čeká právě na token od aktivity „vybírá co dokoupit“.



Obrázek 9.1 Diagram pro eshop

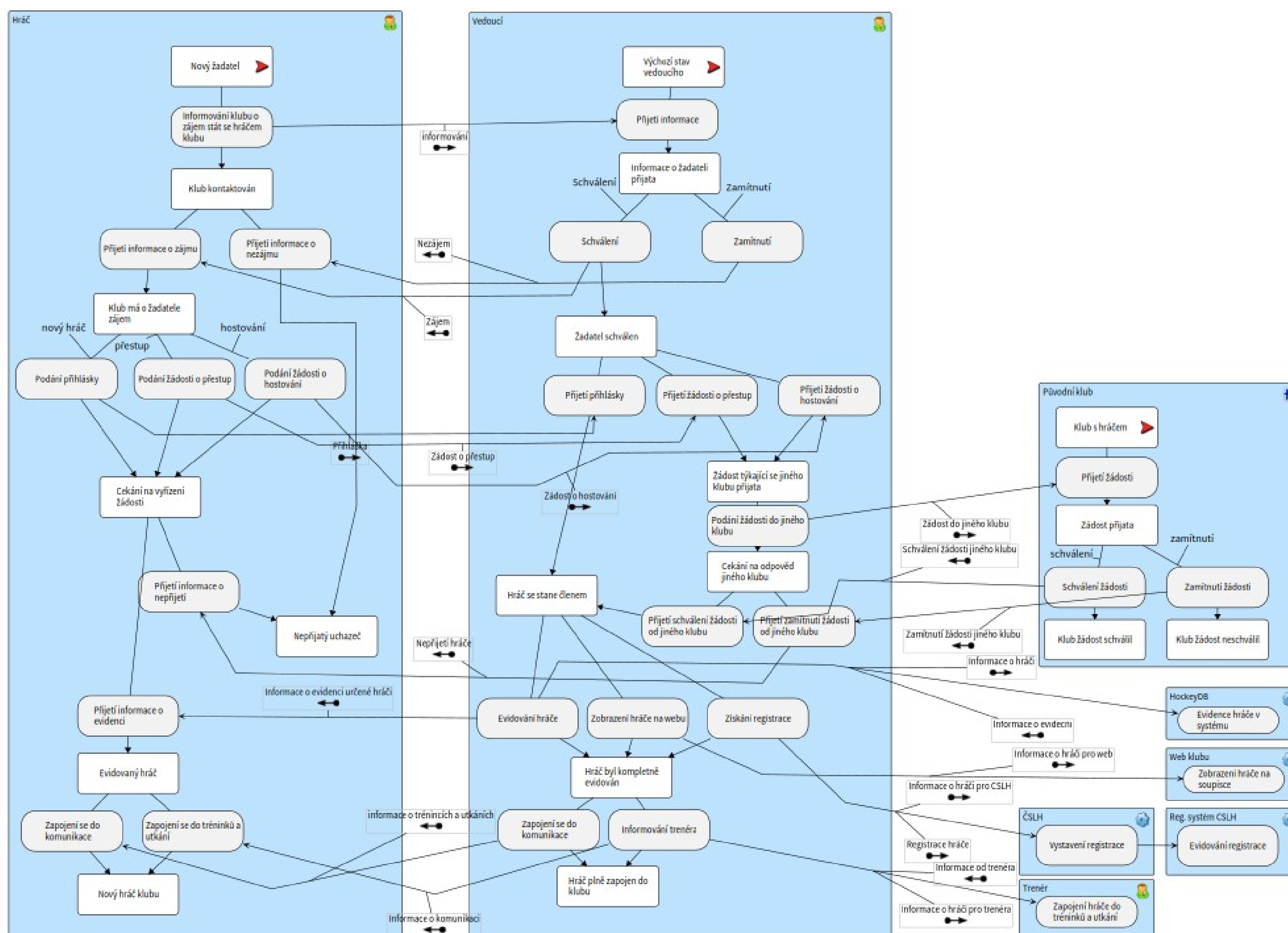
Simulace mi tedy pomohla diagram opravit, tak aby měl dobrý smysl. Opravený diagram je znázorněn na obrázku 9.2.



Obrázek 9.2 Opravený diagram pro eshop

9.1.2 Jednodušší porozumění procesů – diagram získání nového hráče

Diagram získání nového hráče dobře znázorňuje další výhodu simulace – jednoduší a rychlejší pochopení procesů v diagramu. Diagram byl namodelován v rámci semestrální práce „Optimalizace procesů v hokejovém klubu“ Danielem Vrátilem a Matějem Černíkem. Diagram má celkem osm participantů a 51 aktivit a stavů. Když k tomu připočteme velké množství přechodů a komunikací, snadno se v diagramu ztratíme. Simulace pomůže uživateli pochopit diagram krok po kroku a značně sníží čas potřebný k pochopení různých procesů.



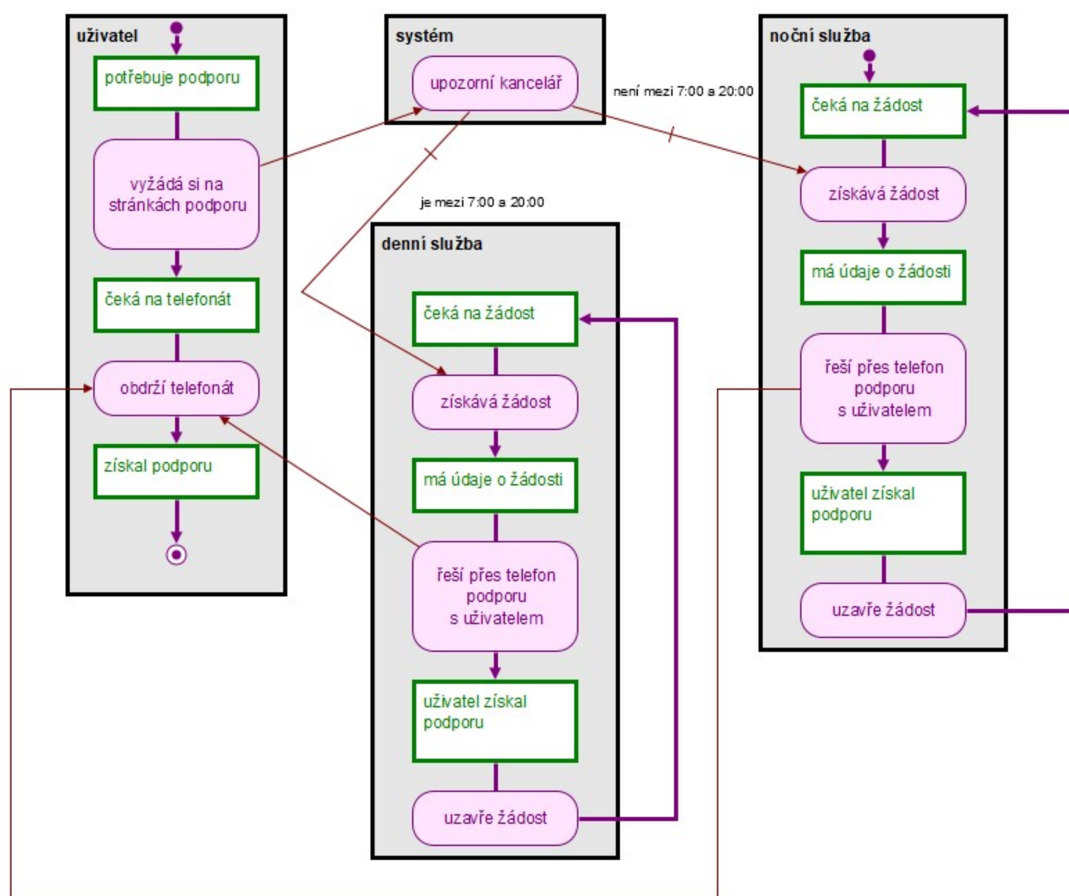
Obrázek 9.3 Diagram podpory namodelovaný v nástroji Craft.CASE

9.2 Porovnání simulace s nástrojem Craft.CASE

Pro porovnání simulace s nástrojem Craft.CASE (verze 2.4.18.1) jsem se rozhodl využít diagram pro podporu a diagram odpálení bomby znázorněné v kapitole Implementace na obrázcích 7.1 a 7.2. Namodeloval jsem je znovu v nástroji Craft.CASE a spustil na nich simulaci.

9.2.1 Simulace podpory

V tomto případě funguje simulace tak jak by uživatel očekával – aktivita obdrží telefonát čeká dokud nedostane signál přes alespoň jednu komunikaci a pak simulace pokračuje dál.

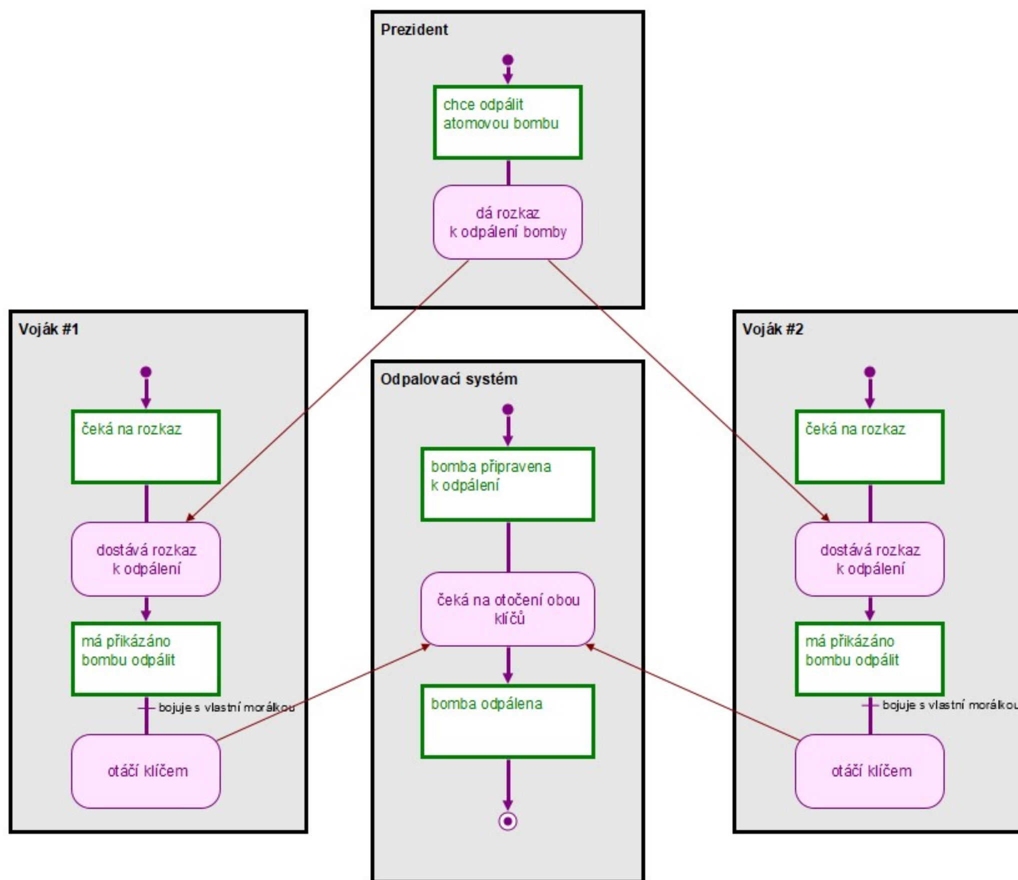


Obrázek 9.4 Diagram pro eshop

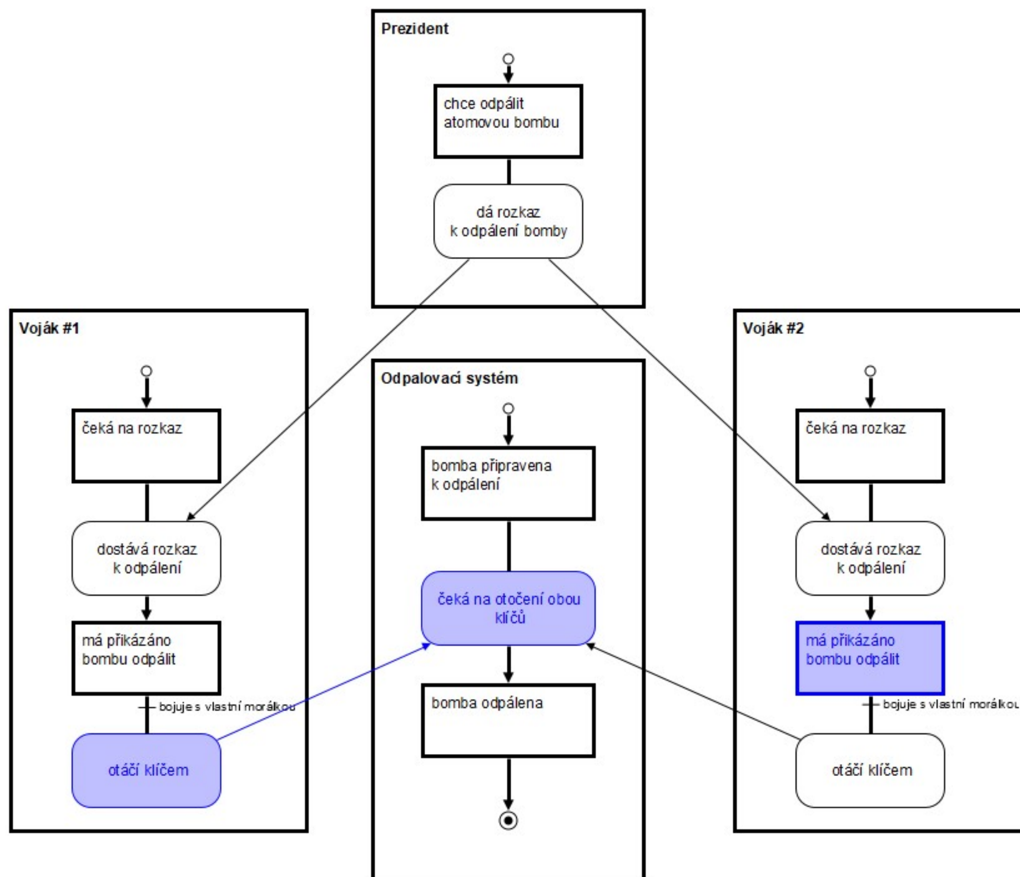
9.2.2 Simulace odpálení bomby

Při simulaci tohoto diagramu vychází najevo nedostatky implementace simulace v nástroji Craft.CASE. Jelikož nelze přenastavit input podmínky, tak input podmínka pro komunikace je vždy nastavena ekvivalentně k podmínce v OpenPonk simulaci *Any*.

V diagramu by mělo dojít k odpálení bomby pouze v případě, že oba vojáci otočili klíčem. V simulaci Craft.CASE ale dojde k odpálení bomby i když otočí klíčem jen jeden voják, tak jak je znázorněno na obrázku 9.6.



Obrázek 9.5 Diagram odpálení bomby v Craft.CASE



Obrázek 9.6 Simulace odpálení bomby v nástroji Craft.CASE

Kapitola 10

Instalace a návod k použití

10.1 Instalace

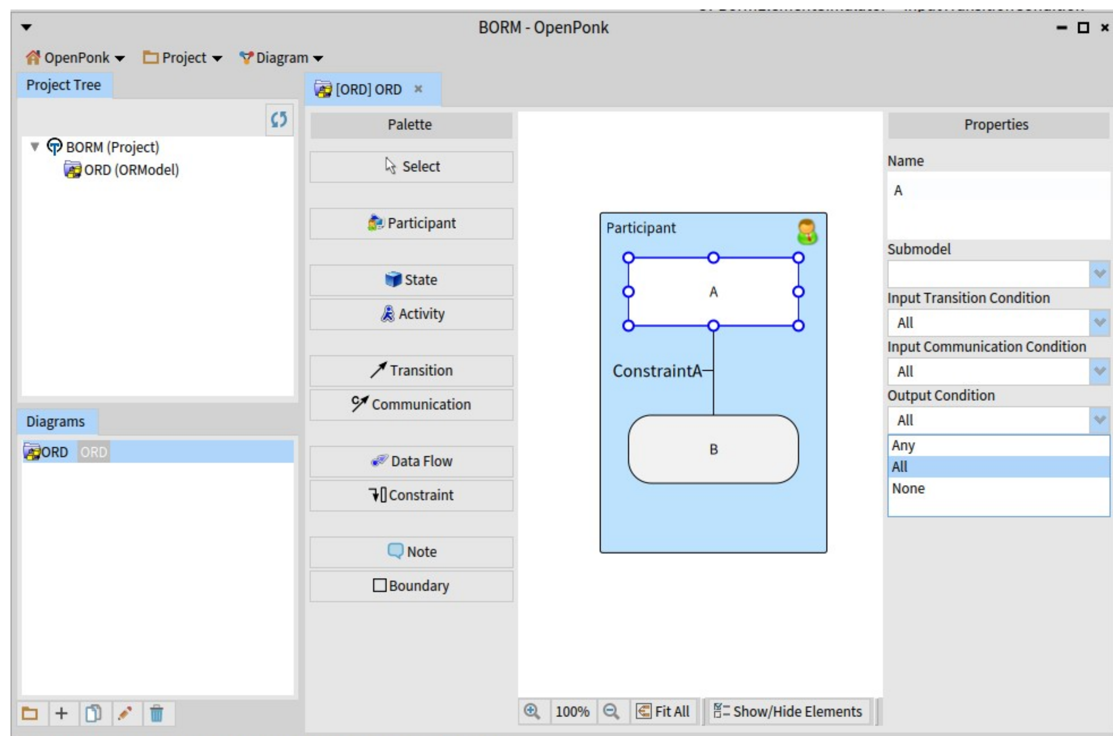
V době odevzdání této bakalářské práce ještě nebyla přidaná simulace BORM ORD do oficiálního vydání OpenPonk, ale doufám, že se tak brzy stane. Do té doby pro instalaci stačí přiloženou image (ze složky exe/image) spustit pomocí Pharo Launcheru dostupného z oficiálních stránek Pharo
(https://www.pharo.org/download-70). Pro systémy window stačí spustit .bat soubor ve složce exe.

10.2 Návod k použití

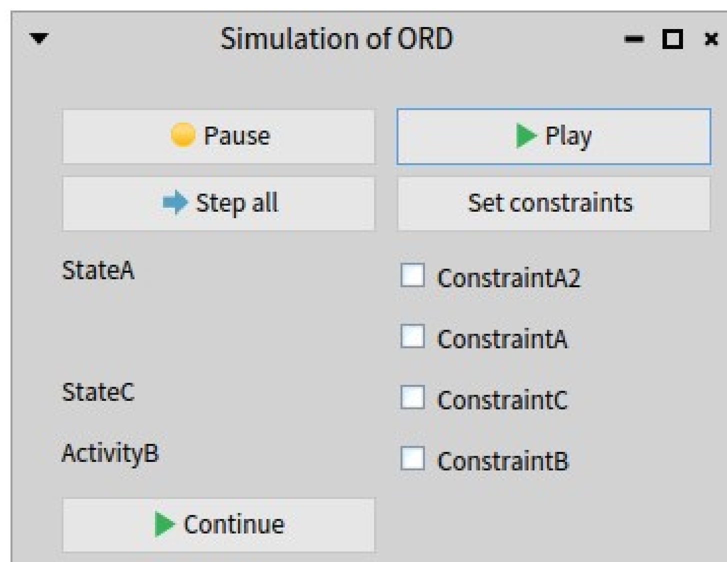
Nejprve musí uživatel spustit image a namodelovat svůj BORM ORD diagram. Ve spuštěné instanci image tedy stačí v horním panelu kliknout na BORM editor. Důležité při modelování je nezapomenout na správné nastavení input a output podmínek. Toto nastavení je vidět na pravém panelu v obrázku 10.1.

Jakmile má uživatel vytvořený diagram a nastavené input a output podmínky, může začít simulovat. Okno simulace se spouští v menu *Diagram* v horním panelu. Pokud chce uživatel podmínky přednastavit před simulací, stačí kliknout na tlačítko *Set constraints* a otevře se mu menu pro přednastavení. V tomto menu uživatel vybere podmínky, u kterých chce nastavit hodnotu pravdivé.

Pro spuštění krokování diagramu stačí kliknout na tlačítko *Play*, které udělá krok každou vteřinu, případně pokud uživatel chce krokovat sám, může využít tlačítko *Step all*. Pokud nebyly podmínky přednastaveny a simulace narazí na podmínku, tak se pozastaví. V tomto případě uživatel může nastavit přímo v okně simulace hodnotu této podmínky a poté kliknout na tlačítko *Continue*, které znovu spustí krokování.



Obrázek 10.1 Okno BORM editoru



Obrázek 10.2 Okno simulace s podmínkami k nastavení

Kapitola 11
Závěr

Cílem této bakalářské práce bylo navrhnout a naimplementovat simulaci BORM ORD modelů pro platformu Openponk.

V rámci bakalářské práce bylo vytvořeno rozšíření platformy OpenPonk, využívající vybrané algoritmy pro simulaci BORM ORD diagramů a interakci s uživatelem. Tyto algoritmy byly inspirovany již existujícími implementacemi simulací FSM a Petri Net. Byl rozšířen existující metamodel a UI pro modelování BORM ORD diagramů tak, aby podporovaly nutné standardy pro smysluplnou simulaci. Řešení bylo otestováno a předvedeno v případové studii. Dále byl zanalyzován a diskutován přínos simulace BORM ORD modelů pro modelování business procesů. Řešení splnilo stanovené funkční požadavky. Nefunkční požadavky byly také splněny – vzhledem k využítí simulace studenty předmětu BI–ZPI lze tvrdit že simulace je jednoduše pochopitelná a ovladatelná. Pravidla simulace jsou jednoduše upravitelná a rozšiřitelná díky objektově orientovanému přístupu k architektuře aplikace a znovu využití GUI Petriho sítí zajišťuje jednotný vzhled aplikace.

Tato práce umožňuje pomocí simulace jednodušší pochopení a orientaci v modelech BORM ORD nejen studentům modelujícím BORM ORD v předmětu BI–ZPI, ale i každému, kdo využívá platformu OpenPonk, včetně profesionálů v oblasti procesního inženýrství.

Ačkoliv tato bakalářská práce zahrnuje funkční implementaci simulace BORM ORD, existují další možné kroky pro vylepšení, které leží mimo rozsah této práce. Mezi tyto kroky patří robustnější vyhodnocování input – output podmínek, například pomocí logických formulí (AND, XOR, OR). Dále by v budoucnu bylo zajímavé rozšířit aplikaci o možnost konverze BORM ORD modelů na Petriho sítě. Dalším možným krokem pro vylepšení simulací na platformě OpenPonk by bylo rozšíření balíku OpenPonk–Simulation o třetí vrstvu mezi simulací a modelem tak, aby se kód pro simulaci mohl lépe řídit principy objektově orientovaného programování.

Bibliografie

1. MYLOPOULOS, John. Conceptual modelling and Telos [online]. [B.r.] [cit. 2024-04-13]. Dostupné z: https://www.researchgate.net/profile/John-Mylopoulos/publication/242177349_Conceptual_Modelling_and_Telos1/links/5564397508ae9963a11f0a53/Conceptual-Modelling-and-Telos1.pdf.
2. *What is UML | Unified Modeling Language — uml.org* [online]. [B.r.]. [cit. 2024-05-16]. Dostupné z: https://www.uml.org/what-is-uml.htm.
3. ROSING, Mark von; WHITE, Stephen; CUMMINS, Fred; MAN, Henk de. Business Process Model and Notation—BPMN. In: *The Complete Business Process Handbook*. Elsevier, 2015, s. 433–457. Dostupné z DOI: 10.1016/b978-0-12-799959-3.00021-5.
4. PETERSON, James L. Petri Nets. *ACM Computing Surveys*. 1977, roč. 9, č. 3, s. 223–252. ISSN 1557-7341. Dostupné z DOI: 10.1145/356698.356702.
5. KŘIVÝ, Ivan; KINDLER, Evžen. *Simulace a modelování* [online]. Ostravská univerzita, Přírodovědecká fakulta, 2001 [cit. 2024-05-16]. Dostupné z: https://vendulka.zcu.cz/Download/Free/SkriptaKindlerMS.pdf.
6. *Modern Business Process Automation: YAWL and its Support Environment*. Springer Berlin Heidelberg, 2010. ISBN 9783642031212. Dostupné z DOI: 10.1007/978-3-642-03121-2.
7. KNOTT, Roger; MERUNKA, Vojtech; POLAK, Jiri. The BORM methodology: A third-generation fully object-oriented methodology. *Knowledge-Based Systems*. 2003, roč. 16, s. 77–89. Dostupné z DOI: 10.1016/S0950-7051(02)00075-8.
8. MERUNKA, Vojtěch; POLÁK, Jiří. BORM – Business Object Relation Modeling [online]. 1999 [cit. 2024-04-13]. Dostupné z: http://prog-story.technicalmuseum.cz/images/dokumenty/Programovani-TSW-1975-2014/1999/1999-25.pdf. Popis metody se zaměřením na úvodní fáze analýzy I.S.
9. RUBIN, Kenneth S.; GOLDBERG, Adele. Object behavior analysis. *Communications of the ACM*. 1992, roč. 35, č. 9, s. 48–62. ISSN 1557-7317. Dostupné z DOI: 10.1145/130994.130996.
10. PODLOUCKÝ, Martin; PERGL, Robert. Towards Formal Foundations for BORM ORD Validation and Simulation. In: *Proceedings of the 16th International Conference on Enterprise Information Systems - Volume 2: ICEIS*, SciTePress, INSTICC, 2014, s. 315–322. ISBN 978-989-758-028-4. Dostupné z DOI: 10.5220/0004897603150322.
11. MERUNKA, Vojtěch. První zkušenosti s modelovacím nástrojem Craft.CASE. *Katedra informačního inženýrství, PEF, ČZU Praha* [online]. 2005 [cit. 2024-04-13]. Dostupné z: http://prog-story.technicalmuseum.cz/images/dokumenty/Programovani-TSW-1975-2014/2005/2005-20.pdf.

12. BALDA, Michal. *Portál pro optimalizaci BORM procesů*. 2013. bakalářská práce. Fakulta informačních technologií. Vedoucí práce Robert PERGL.
13. GEEKSFORGEEKS. *Introduction to Smalltalk* [`https://www.geeksforgeeks.org/introduction-to-smalltalk/`]. 2024. [cit. 2024-05-16].
14. COMMUNITY, Pharo. *About Pharo* [`https://pharo.org/about`]. 2024. [cit. 2024-05-16].
15. DUCASSE, Stéphane; POLITO, Guillermo; ALCOCER, Juan Pablo Sandoval. *Testing in Pharo*. 2023.
16. UHNÁK, Peter; PERGL, Robert. The OpenPonk modeling platform. In: 2016, s. 1–11. Dostupné z DOI: `10.1145/2991041.2991055`.
17. BLIZNIČENKO, Jan. *Podpora simulace a vizualizace v nástroji DynaCASE*. 2015. Dostupné také z: `https://dspace.cvut.cz/handle/10467/63136`. Bakalářská práce. ČVUT FIT. Vedoucí práce Robert PERGL.

Obsah příloh

```
readme.txt ........................ stručný popis obsahu média
exe ............................... adresář se spustitelnou formou implementace
src
  thesis .......................... zdrojová forma práce ve formátu LaTeX
diagrams .......................... využité diagramy ve formátu .opp
text .............................. text práce
  thesis.pdf ...................... text práce ve formátu PDF
```