Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Diskrétní simulace pro metodu BORM v nástroji OpenCABE

Veronika Larionova

Vedoucí práce: Ing. Robert Pergl, Ph.D.

11. května 2015

Poděkování

V rámci této možnosti bych chtěla poděkovat vedoucímu mé práce, panu Ing. Robertu Perglovi, Ph.D., za velmi cenné rady a znalosti, kterých jsem nabyla během konzultací, panu Mgr. Martinu Podlouckému za poskytnuté informace o nástroji OpenCABE. V neposlední řadě chci poděkovat své rodině, která mě po celou dobu studia podporovala.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 11. května 2015 .....

Odkaz na tuto práci

Larionova, Veronika. *Diskrétní simulace pro metodu BORM v nástroji OpenCABE*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato bakalářská práce se zabývá zkoumáním možností, jak rozšířit procesní modely Object Relation Diagram (ORD) v metodě BORM o schopnost simulace. Zároveň poskytuje návrh, jak lze propojit modelovací nástroj OpenCABE s knihovnou pro diskrétní simulaci.

V textu je rozebrána jak samotná technika diskrétní simulace, tak i knihovny, které ji implementují v programovacím jazyce Java. Kromě toho je krátce představená metoda BORM.

Práce hledá řešení rozšíření ORD ve verzi vyvinuté v CCM (Centre for Conceptual Modelling) FIT o prvky a parametry pro podporu diskrétní simulace. Na základě navržených rozšíření a poznatků z rešeršní části bakalářské práce je představen návrh začlenění knihovny pro diskrétní simulaci do nástroje OpenCABE.

Ve výsledku vznikly dva návrhy. První pokrývá rozšíření procesních modelů ORD o konstrukty umožňující diskrétní simulaci. Druhý návrh obsahuje popis způsobu začlenění knihovny Desmo-J do nástroje OpenCABE. Součásti řešení jsou též doporučení pro následnou implementaci.

Klíčová slova diskrétní simulace, BORM, ORD, OpenCABE.

Abstract

This thesis covers the research on the possibilities of the extension of BORM's Object Relation Diagrams (ORD), in version of CCM FIT, with elements which will enable discrete-event simulation of ORD. At the same time it gives the description of the design which introduces the integration of chosen discrete-event simulation library written in Java into modeling tool OpenCABE.

Text provides the research not only on the topic of discrete-event simulation, but also introduces the analyses of non-commercial discrete-event simulation libraries. The smaller chapter describes the methodology BORM and its ORD.

Based on the collected data the design of the extensions of ORD is presented. Taking the extensions in consideration the possible integration of discrete-event library is introduced. The thesis is concluded with recommendations for following implementation and evaluation of the achievements.

Keywords discrete-event simulation, BORM, ORD, OpenCABE.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Podniky v dnešní době mají čím dál tím víc složitější strukturu. V rámci nich probíhají nejrůznější podnikové procesy. Jelikož je v zájmu vedení firem jejich prosperita, musí být procesy výhodné a efektivní, proto jsou zkoumány, analyzovány a vylepšovány.

Některé podnikové procesy lze vyhodnocovat za pomoci matematických vzorců. Bohužel daná metodika analýzy se dá aplikovat pouze na menší množinu jednodušších problémů (např. řízení zásob). Častěji se v praxi setkává s komplikovanými procesy, kde velmi důležitou roli hraje jejich nedeterministický a dynamický charakter chování. Pro dané případy je volena simulace. V rámci ní se napodobuje chování podnikových systémů a provádějí se série pokusů, jejichž výsledky posléze napomáhají k optimalizaci procesů. Nejčastěji se pro simulování chování firem volí diskrétní simulace.

Metoda BORM představuje nástroj, kterým lze podnikové procesy zachytávat a modelovat. Poskytuje pro tyto účely jednoduše pochopitelné prostředky, ke kterým například patří procesní modely Object Relation Diagram.

Cíle a metodika

Práce má za cíl propojit metodu BORM a diskrétní simulaci. Daný úkol je rozložen do tří kroků:

1. návrh rozšíření ORD tak, aby byla umožněna simulace procesních modelů,

2. návrh rozšíření nástroje OpenCABE, který bude reflektovat návrh rozšíření ORD,

3. návrh integrace vybrané knihovny pro diskrétní simulaci do modelovacího nástroje OpenCABE na funkční rovině. Nakonec se zformulují závěry a doporučení.

Průběh naplnění cílů se dělí na dvě části: rešeršní, kde jsou shrnuté znalosti potřebné k tvorbě řešení, a návrhovou, ve které se představí návrhy popisující možnost řešení úkolů. Práce bude postupovat následovně:

1. Nejprve se provede rešerše techniky diskrétní simulace. Daná část bude rozdělena do dvou kroků. Zpočátku se zpracují informace o simulaci jako celku. 1 Posléze se přejde k rešerši vlastností, které přísluší výhradně diskrétní simulaci.2

2. Dále se zjistí informace o metodice BORM. 3 Jedná se o získaní přehledu o jejích vlastnostech a podstatě. Nejdůležitějším poznatkem z dané části bude popis ORD a jeho prvků, který posléze bude tvořit základ pro návrh.

3. V dalším kroku se vytvoří přehled knihoven pro diskrétní simulaci, na základě kterého se zvolí zástupce pro následný návrh. 4

4. Posléze se přejde do stadia návrhu. Nejprve se provede návrh rozšíření ORD. 5 Zde budou aplikovány poznatky z výše popsaných kroků 1 a 2.

5. Následně se přejde k návrhu integrace vybrané knihovny do modelovacího nástroje OpenCABE. 6 Daná část se rozdělí na tři fáze:

   a) navržení rozšíření vybrané simulační knihovny o modelovací prvky, které zastupují elementy ORD,

   b) návrh rozšíření nástroje OpenCABE tak, aby podporoval představené změny v ORD,

   c) návrh propojení knihovny pro diskrétní simulaci a OpenCABE.

6. Nakonec se zformulují závěry a doporučení.7

Část I

Rešerše

Kapitola 1

Simulace

Pojem simulace může mít mnoho významů, které patří do různých vědeckých sfér. Nejčastějším výkladem je však, že se jedná o metodu získávání nových znalostí o objektech, prostředí nebo situaci, které zkoumáme, tím že se provádí pokusy nad jejích modelem. [1] Simulaci se vlastně napodobuje chování systému.

Choi a Kang ve své publikaci [2] rozšiřují tuto definici o další vlastnost: simulace může sloužit i k vzdělávaní či zábavě lidí. Tento poznatek dovoluje vymezení dvou druhů metody:

- *analytické simulace* (analytic simulation), která se zaobírá experimentováním s modelem za účelem získávání nových poznatků,
- *simulace virtuálního prostředí* (virtual environment simulation), jejímž účelem je vzdělávání či zábava.

Pro účely této práce je podstatný pojem analytické simulace.

1.1 Původ simulace

Simulace představuje disciplínu, která vznikla z metody Monte Carlo. Postupem času se simulace oddělila a stala se samostatnou disciplínou, ale i přesto je s metodou často spojována. Jejich rozdílnost lze vidět na účelu použití. Zatímco metoda Monte Carlo se zabývá statistickými odhady, simulace napomáhá ke zkoumání složitých dynamických systémů.[3]

1.1.1 Metoda Monte Carlo

„*Metodou Monte Carlo rozumíme numerické řešení pravděpodobnostních i deterministických úloh pomocí statistického experimentu.*“[3] Metoda je založena na opakovaných experimentech a jejich vyhodnocení. S využitím statistického

aparátu lze posléze získat odhad zkoumané hodnoty. Platí, že s počtem pokusů roste i přesnost odhadu.

Častým příkladem metody Monte Carlo je určení Ludolfova čísla π. Pro popis se používá kruh o průměru 1 jednotky, který je umístěn doprostřed čtverce o straně rovné 1 jednotce. Následně jsou prováděné „hody“ do čtverce. Hodnota π je potom daná podílem počtu hodů, které padly do kruhu, ku celkovému množství pokusů.[2]

Metoda se používá v různých odvětvích, od vědeckých po ekonomická. Častým příkladem aplikace je výpočet N-rozměrných integrálů.[1]

1.2 Systém

Jeden ze základních pilířů simulace představuje pojem *systém*. Jedná se o množinu objektů s vymezenými vztahy mezi nimi. Tento popis lze rozšířit o faktory, které ovlivňují systém z vnějšku. Zmíněné externí faktory tvoří okolí systému (system enviroment). [4]

1.2.1 Komponenty systému

Základními „součástkami“ systémů jsou *prvky* (entity), které reprezentují zkoumané objekty. Můžou to být pacienti, auta, disky. Pro popis charakteru jednotlivých elementů se používají *atributy*. Podoba atributů nabývá různých hodnot od čísel, přes pravdivostní hodnoty, až do textových popisů.

Jelikož podstatnou vlastnost objektů tvoří vazby mezi nimi, které též mají podobu atributů, rozlišují se dva druhy atributů:

- referenční, jenž odkazuje na další prvek (zákazník obsluhovaný číšníkem),
- standardní udávající určité hodnoty objektů, např. věk, výšku, barvu očí.

V systémech, které se v čase mění, existují dva typy objektů, jenž se v nich objevují. Buď se entity nachází v systému stále (permanentní objekty), nebo jim jen prochází. Daný druh se nazývá transakci.[5]

1.2.2 Stav systému

Stav systému je dán souhrnem *stavových proměnných* a *proměnných systému* v určitém časovém okamžiku. Stavové proměnné se vztahuji k jednotlivým objektům. Jedná se o vazby mezi prvky nebo i o jednotlivé vlastnosti. Na druhou stranu proměnné systému se vážou k systému jako celku. Příklady takových proměnných můžou být počet zákazníků nebo množství volných pokladen v obchodě. [2] a [1]

1.2.3 Dělení systémů

Kategorizace systémů může probíhat na základě různých kriterií, přičemž jednotlivých druhů dělení je mnoho a zároveň zkoumaný systém může být z hlediska různých pohledů řazen do více kategorii.

Pokud pro popis systému není zahrnuto časové hledisko, mluví se o *statických systémech*. Opačnou skupinu tvoří *systémy dynamické*, které jsou základem pro techniku simulace.[5]

Další dělení může být ovlivněno tím, zda systém, se kterým se pracuje, existuje. Tehdy se hovoří o *reálném systému*. Lze se ale střetnout s případy, kdy jeho realizace není možná. K danému případu se pojí pojem *koncepční systém*.[6]

Kriteriem zařazování můžou být i změny stavů systému. Zkoumá se predikovatelnost následujícího stavu. [4]

- *Deterministické systémy* se chovají zcela předvidatelně. Na základě předchozího stavu a aktivity, která bude pro přechod vykonána, je již jasné, jaký bude stav následující.

- *Stochastické systémy* se vyznačují tím, že ze známých údajů nelze stanovit další krok. Důležitou roli hraje míra náhodnosti. Stochastický přístup se používá pro popis mnoha reálných situací, např. obsluha lidí.

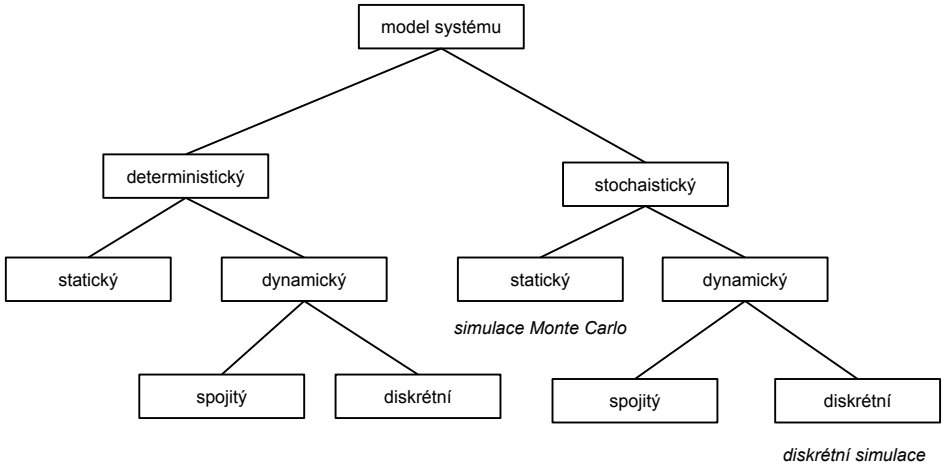
1.3 Model

Pod pojmem model se schovává forma popisu vymezeného systému. Proces tvorby modelu, tj. modelování, představuje aplikaci zobrazení, které převede prvky a atributy vybrané části systému na objekty modelů.[1] Musí zároveň platit, že model zachovává vlastnosti modelovaného systému. Jednoduchým příkladem takového popisu je mapa, která představuje zjednodušenou reprezentaci části zeměkoule.[5] Obvykle hranice mezi pojmy model a systém může být opomenuta v momentě, kdy se vyhodnotí, že model představuje dobrý obraz systému.[7]

Zobrazení však není jednoznačné, což má za následek to, že pro jeden modelovaný systém lze nalézt více systémů modelujících. Tento vztah platí i naopak, jelikož v důsledku zobrazení mohou být zanedbané některé detaily systému.

Model může mít různou formu. Systém někdy lze vyjádřit s pomoci matematických prostředků v podobě diferenčních rovnic, nebo lze použit i jednoduché grafické znázornění. V souvislosti s analytickou simulaci figuruji tyto druhy modelů [1]:

- *abstraktní model*, ve kterém jsou dány jednotlivé prvky a relace mezi nimi



Obrázek 1.1: Klasifikace modelů (převzáto z [8])

- *simulační model*, kdy se jedná o proveditelnou podobu abstraktního modelu. Proveditelnost říká, zda lze s modelem provádět experimenty za pomoci simulačního programu. Model může mít podobu kódu nebo i grafickou podobu Petriho sítě, pokud se nalezne vhodný simulátor, který si stouto podobou poradí.

1.3.1 Klasifikace modelů

Modely lze dělit dle různých kriterií, proto je složité uvést jednoznačnou klasifikaci modelů. Jako tradiční dělení uvádí Peringer [1] skupiny modelů, které se liší charakterem změn stavů jejich proměnných v čase.

- *Modely spojité* – představují množinu, pro níž platí, že změny stavů proměnných probíhají spojitě v průběhu času, tj. mění se plynule. Pro popis lze zvolit diferenční rovnice.

- *Modely diskrétní* – v této skupině modelů se stav mění skokově v určitých časových okamžicích. Jako podobu modelu lze zvolit konečné automaty nebo ACD (activity cycle diagramm).

- *Modely kombinované* – pro skupinu platí, že součásti modelu jsou prvky jak s diskrétním chováním, tak i se spojitým.

Další možnost klasifikace je znázorněna obrázkem 1.1. Zde je patrné slučování klasifikace systémů a modelů, které je dáno zaměňováním pojmů, jež bylo zmíněno dříve 1.3.

1.3.2 Validace a verifikace modelu

Validace a verifikace představují procesy, které jsou nedílnou součásti simulace, ale též se objevují v dalších disciplínách spojených s modelováním.

Validace je proces, při kterém se ověřuje, zda abstraktní model odpovídá modelovanému systému. Validita se ověřuje iterativně, aby rozdíl mezi realitou a jejím zobrazením byl nejmenší.

Model převážně nereprezentuje celý systém, proto lze jen těžko mluvit o úplné validitě. Spíše je zmíněna míra přesnosti, kterou model má. Existuje několik stupňů míry validity[9]:

- replicatively valid (validní z hlediska replikace) – data generovaná modelem odpovídají datům systému,
- predictively valid (predikovatelně validní) – stanovuje, že výstupní data simulace odpovídají hodnotám, které budou získané z systému,
- stucturally valid (strukturně validní) – se vztahuje k vlastností, kdy model nejen reprodukuje chování systému, ale i reflektuje způsob, jakým je chování získáváno.

Verifikace modelu zas řeší již před samotnou simulací. Zde se ověřuje, zda simulační model odpovídá abstraktnímu z hlediska struktury a chování.

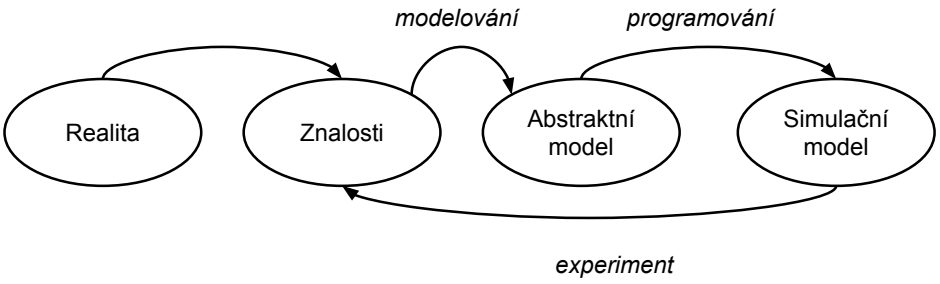
1.4 Princip simulace

Proces simulace je podmíněn tím, že nejprve se stanoví, co se bude simulovat, a posléze se na základě detailních znalostí vytvoří model. Obrázek 1.2 ukazuje, jak probíhá proces získávaní poznatků s pomocí simulace.[1] Zpočátku na základě znalostí, které jsou často zprostředkovány experty daného oboru, vytvoří abstraktní model. V něm se zformulují informace podstatné pro experiment. Posléze se abstraktní model převede na simulační. V této fázi by nemělo docházet k vypuštění žádné informace z abstraktního modelu, i když se to vlivem šetření času a prostředků někdy stává. Simulační model vstupuje do procesu simulace. Výsledkem tohoto kroku jsou nějaká data, která se zobecněním převedou na poznatky.

1.4.1 Proces modelování a simulace

Daná část poskytuje podrobnější popis postupu. Proces má spíše iterativní charakter za účelem udržování pravdivé informace. V praxi se jednotlivé kroky mohou upravovat, přehazuje se jejich pořadí nebo se některé části vynechávají. [10]

1. *Formulace problému.* Rozpoznání problému a určení cílů patří k stěžejnímu kroku procesu. Zde se vymezuje systém, jeho objekty a vztahy.



Obrázek 1.2: Princip simulace (inspirováno [1])

Provádí se analýza řešení na základě dřívějších projektů a vybere se nejvhodnější varianta. Může se provést analýza rizik, která rozhodne, zda má smysl pokračovat.

2. *Tvorba konceptuálního modelu.* Jedná se tvorbu základní představy o modelu. Definují se objekty, jejich atributy a vztahy. Stanovují se stavové proměnné a určuje se jejich relevantnost. Zároveň se hledají vlastností, které by mohli mít kritický dopad na systém, např. uváznutí, možné nestability.

3. *Sběr a analýza dat.* Krok vyhodnocuje informace nabyté z získaných dat (shromažďování dat probíhá v kroku 1). Stanovují se stochastické a deterministické parametry. Pokud nejsou k problému žádné naměřené hodnoty, využívají se zkušeností s podobnými případy. Vyhodnocuje se aplikace statistických rozdělení.

4. *Tvorba simulačního modelu.* Vybírají se simulační technologie. Dochází k transformaci konceptuálního modelu do podoby spustitelného simulačního modelu. V průběhu se můžou odhalit chyby, které nebyly v předchozích krocích objeveny.

5. *Validace a verifikace.*(viz 1.3.2) Nejedná se přesně o určitý krok. Spíš představuje integrovanou část procesů. Validace a verifikace musí být podložené dokumentací, jelikož v dalších iteracích není vyloučeno, že bude potřeba modely měnit a tím i opakovat jejich ověření. Dokumentace v daných případech zjednoduší orientaci ve změnách. Posléze může sloužit podkladem pro jiné simulace.

6. *Experiment.* Krok začíná tvorbou experimentu. Následně se na základě stanovených cílů a požadavků provádí pokusy. Posléze se vyhodnocují výstupy a upravují se parametry.

7. *Analýza výstupů.* Nakonec jsou výstupní data vyhodnocená za účelem pochopení chování systému. Dobrou pomůckou pro tuto fázi je vizualizace, která přispívá k lepšímu pochopení systému než pouze statistická analýza.

Nesmí se též zapomenout na důkladnou dokumentaci projektu a závěrečnou aplikaci výsledků. [1] a [10]

1.5 Simulace a čas

V rámci simulace se rozlišují dva hlavní druhy časů:

- reálný čas, který odpovídá času skutečného světa,
- simulární čas vyskytující se v průběhu simulace.

Simulární čas je reprezentovaný nějakou proměnnou experimentu s rostoucím charakterem. Pojetí jednotky daných časů nemusí být navzájem stejné. Zatímco obvykle den představuje 24 hodin, simulace ho může pojmout za 1 sekundu. [5]

1.6 Modelování vstupních dat

Data, která vstupují do simulace, rozhodují o jejím budoucím vývoji. Proto je velmi důležitá disciplína zabývající se jejich sběrem a analýzou. Zdrojem jednotlivých údajů mohou být reálná měření, záznamy z jiných projektů, specifikace výrobce atd.. Při zkoumání různých vstupních dat je důležité určit jejich povahu, tj. zda mají deterministický charakter, či pravděpodobnostní.

Deterministické veličiny se vyznačují svou stálou hodnotou, proto pro jejich stanovení je dostačující jen jedno měření. Jedná se o rychlosti výrobních linek, intervaly pro údržbu, ...

Data náhodného charakteru představují reprezentanty rozsáhlejší množiny vstupů. Jejich hodnoty nejsou předem známé. Při simulaci je zastupuji generátory náhodných veličin. Nejčastější se jedná o doby mezi příchody a délky trvání obsluhy. [11]

1.6.1 Náhodné veličiny

Náhodné veličiny představují aparát, který dovoluje vyjádřit neurčitost. Naměřené hodnoty jsou zanalyzované a přiřazené k rozdělení, které udává pravděpodobnost jednotlivých jevů.

1.6.1.1 Určování rozdělení

Rozdělení existuje mnoho, a proto je důležité zvolit nejvíce odpovídajícího zástupce. Při rozhodování můžou nastat dvě situace:

- budou dostupna data z předchozích měření,
- nebo z nějakého důvodu (např. neexistence systému) nebude možné data změřit.

Pokud se pracuje se získanými hodnotami, tak lze pro vyhodnocení kandidáta na rozdělení využit konstrukci histogramu. Z vizuálního vyhodnocení lze získat několik vhodných kandidátů, které posléze budou otestováni testem dobré shody. Nesmí se též opomenout i znalost systému, na základě které lze též omezit množinu možných rozložení.

V případech, kdy data nejsou poskytnutá, musí se vycházet ze znalostí podobných systémů nebo využít rady expertů.[3]

1.6.1.2 Příklady rozdělení

Jednotlivá rozdělení lze řadit do dvou skupin: na základě toho, zda pracují pouze s celými čísly (diskrétní), nebo používají hodnoty z celého intervalu (spojitá).

Jelikož rozdělení tvoří obsáhlou množinu, bude představeno jen několik častěji používaných zástupců. Z důvodů velkého rozsahu dané tématiky příklady obsahuji jen popis aplikace a přehled parametrů. Pro hlubší znalosti se doporučuje obrátit například na [2] nebo další literaturu zabývající se tématem statistiky.

Spojitá rozdělení Spojitá rozdělení umožňuji generovat hodnoty z cele číselné osy. Nejčastěji se aplikují na simulaci intervalů mezi příchody, dob obsluhy a činností.

Exponenciální rozdělení Exponenciální rozdělení představuje vhodného kandidáta pro modelování časových intervalů mezi nezávislými událostmi nebo pro procesy, které „nemají paměť“, tj. to co se stalo do daného momentu, nijak neovlivní následující krok. Rozdělení má jeden parametr λ, který udává počet příchodů za časovou jednotku. Jeho hodnota musí být kladná.

Rovnoměrné rozdělení Pro rovnoměrné rozdělení platí, že každá hodnota je stejně pravděpodobná. Tato vlastnost pomáhá k modelování náhodností, a proto je rozložení vhodné pro doby trvání činnosti. Jako parametry se udávají krajní hodnoty intervalu hodnot, které mohou nastat, tj. horní a dolní hranice.

Normální rozdělení Normální rozdělení lze použit pro situaci, kdy se na celku podílejí dílčí operace (např. proces výroby auta, kdy jednotlivé kroky trvají určitou dobu). Používá se pro modelování chyb ekonomických pozorování a fyzikálních měření. Rozdělení disponuje dvěma parametry: střední hodnotou μ a rozptylem σ^2.

Erlangovo rozdělení Erlangovo rozdělení pomáhá získávat nezáporné hodnoty exponenciálního charakteru. Používá se pro modelovaní dob příchodu nebo též slouží pro uřčení doby života do k-té poruchy.

Diskrétní rozdělení Generátory diskrétních rozdělení zprostředkovávají celočíselné hodnoty. Vyjadřují počty zakázek nebo osob.

Geometrické rozdělení Geometrické rozdělení slouží po popis rozložení počtů jevů, jejíž výsledkem je příznivý jev. Je charakterizováno jen jedním parametrem, který určuje pravděpodobnost p, s jakou nastane žádaná situace.

Binomické rozdělení Binomické rozdělení ukazuje, jak jsou rozložené příznivé výsledky v n pokusech. Jeho parametry jsou pravděpodobnost příznivého jevu p a počet experimentů n.

Poissonovo rozdělení Poissonovo rozdělení lze aplikovat na modelování nezávislých jevů, které nastanou za jednotku času, např. počty příchodů zákazníku do obchodu, počet vadných výrobků, ... Rozdělení je charakterizováno parametrem λ, který udává průměrný počet jevů za časovou jednotku. Jednotlivé doby mezi příchody událostí jsou dány exponenciálním rozdělením se střední hodnotou $1/\lambda$.

1.6.1.3 Generátory náhodných čísel

Téma simulace a náhodných veličin se úzce pojí na pojem generátoru. Jelikož pro stochastickou simulaci jsou potřeba náhodné hodnoty, zabudovávají se do programů generátory náhodných čísel, které potom vrací hodnoty v závislosti na pravidlu (rozdělení).

Daná problematika je velmi rozsáhlá a pro účely práce není primární, jelikož se operuje s hotovými knihovnami pro diskrétní simulaci. Z tohoto důvodu se v případě zájmu doporučuje obrátit na jinou literaturu (např. [3]).

1.7 Typy simulací

Simulaci lze dělit v závislosti na použitých modelech a technologiích. Nejčastější dělení představuje kategorizace založená na typech modelů. Rozlišuji se tyto druhy: *diskrétní*, *spojitá* a *kombinovaná* simulace.

Lze se setkat i s pojmy číslicové, analogové a fyzikální simulace, kdy kriteriem řadění do skupin jsou použité simulační prostředky. Možné je též dělení, které zohledňuje míru přesnosti zpracování výsledků. Za zmínku stojí vnořená simulace. Model, který je pro simulaci použit, obsahuje komponenty, u kterých jsou zohledněný výsledky z vnořených modelů.[1]

1.8 Účel simulace

Důvody, které primárně vedou na aplikaci simulace se dají shrnout v 4 bodech. [11]

- V praxi existují velmi složité systémy, které nelze pochopit statickým zkoumáním. Je zapotřebí sledovat dynamiku modelu. Často se jedná o systémy výroby.
- Dále se simulace nasazuje v případech, kdy systém již existuje, ale požaduje se jeho vylepšení. Rozhoduje se o managementu personálu a lidí, vylepšují se podnikové procesy.
- Na druhou stranu simulace přináší možnost, jak lze testovat nové myšlenky. Mnohdy umožňuje zmenšit ztráty, které by nastaly reálným nasazením systému.
- V neposlední řadě simulace umožňuje získávaní nových znalostí bez zasahování do samotného systému.

1.9 Výhody simulace

S účely simulace souvisejí i výhody jejího použití. Snad nejvýznamnějším důvodem pro volbu simulace je cena. Experimenty s reálným systémem jsou převážně finančně a materiálně náročné, proto se nasazuje technika simulace. Výsledky potom můžou být ověřený jediným pokusem nad skutečnými objekty. [1]

Další benefit simulace se schovává v možnost manipulace s časem. Samotné procesy v reálném světě trvají i roky, což velice ztěžuje jejich výzkum. Při simulaci může rok být vnímán jako jedna vteřina. Zároveň lze manipulovat s rychlosti běhu simulárního času. [11]

Mnohé nástroje pro simulaci poskytují i vizualizační prostředky, které zprostředkovávají větší názornost na chování systému. To umožňuje lepší zachytávaní chyb modelu a nedostatků systému.

Simulace dovoluje snížit i počet analytiku a prostředků, které jsou zapotřebí pro výzkum. Daný benefit se posléze odrazí na nižších nákladech.

1.10 Nevýhody a problémy simulace

I když simulace přináší mnoho pozitivních aspektů, má i svá úskalí. Prvním problémem je tvorba modelu. Konstrukce modelu složitého systému představuje časově a personálně náročnou činnost, kdy se očekává spoluúčast expertů z různých oborů pro formulaci znalostí. Mnohdy je zapotřebí i vyškoleného analytika pro konstrukci simulačního modelu. Zároveň je často opomíjená stránka modelování vstupních dat, která má za následek nesprávné výstupy.[11]

Další nevýhodu představuje problematika validity modelu, kdy experiment s chybným modelem může způsobit katastrofální následky.[1]

Rozsáhlé systémy jsou zas velice náročné na zpracování jejich modelů. V takových případech se musí počítat s nutnosti výkonných počítačů.

Zároveň je simulace často přeceňována a očekává se, že poskytne pro všechno řešení. Opak je bohužel pravdou. Simulace nabízí pouze možnou alternativu. Zda výsledek opravdu použít závisí na managementu a analyticích.

1.11 Alternativní přístup

Simulace je často výkonnostně velmi náročná, a proto se někdy za vhodné shledává použití analytické postupy. Výsledkem jejich aplikace je přesná hodnota. Sice analytické řešení je rychlejší a jednodušší, nelze ho aplikovat vždy. Pro případy systémů, které zahrnují prvky náhodnosti a reflektují průběh času, představuje simulace vhodnějšího kandidáta.[3]

Kapitola 2

Diskrétní simulace

Jak již bylo dříve zmíněno diskrétní simulace zkoumá diskrétní modely/systémy. Systémy jsou dány jako množina statických a dynamických objektů, které jsou mezi sebou propojené. Dynamické objekty (entity) mají stanovené své chování dané jejich vlastnostmi a aktivitami, na základě kterého se pohybují po systému. K dispozici mají i statické objekty, což jsou zdroje, které jsou nutné pro aktivity. Jelikož zdroje nejsou nekonečné jsou nutné fronty, ve kterých entity čekají na jejich uvolnění. [3]

2.1 Modelovací techniky a umístění diskrétní simulace

Modelovací techniky, kterými jsou například Petriho síť, konečné automaty nebo Markovovy řetězce, mohou být klasifikovány na základě reprezentace stavových proměnných a času. Oba dva parametry mohou nabývat dvou hodnot: diskrétních a spojitých.

Spojitý čas se vyvíjí postupně a může být reprezentován reálným číslem. V diskrétním případě se čas hýbe po skocích, které lze vyjádřit celočíselnou hodnotou. Stav je zas dán množinově a pro jeho spojitou podobu platí, že se jedná o množinu spojitých hodnot/reálných čísel. Na druhou stranu diskrétní stav je seskupení celočíselných hodnot.

Tím vznikají čtyři možné skupiny, kam lze modelovací techniky zařadit 2.1.

- DESS (Differential equation system specification). Jedná se o skupinu kam patří spojíte dynamické systémy.

- DTSS (Discrete time system specification). Do této kategorie náleží dynamické systémy s diskrétním časem. Lze jimi popisovat měření veličin v určitých časových okamžicích.

Tabulka 2.1: Metody popisu na základě času a stavu

	spojitý čas	**diskrétní čas**
spojitá množina stavů	DESS	DTSS
diskrétní množina stavů	**DEVS**	Automaty

- DEVS (Discrete event system specification). Sem spadají dynamické systémy diskrétních událostí, k výzkumu kterých se právě plouživá diskrétní simulace.

- Automaty. Kategorie zahrnuje diskrétní dynamické systémy, které lze vnímat jako podmnožinu systému diskrétních události s tím, že události probíhají ve fixních okamžicích.

[10]

Diskrétní simulace tedy představuje techniku, která vnímá čás spojitě, ale množina stavů, kterých může simulovaný systém dosáhnou, je diskrétní.

2.2 Modelovací metody pro diskrétní simulaci

Pro popis dynamických systémů diskrétních událostí lze použít velkou množinu metod. Jejich aplikace je dána vlastnostmi systémů a závisí na expertovi, kterou metodu zvolí. V této části bude představeno pár zástupců.

2.2.1 Konečné automaty

Systém je popsán konečnou množinou stavů, které jsou propojeny přechody. Stavy určují vývoj systému. Přechody jsou zas spojené s podmínkou, která umožňuje jejich provedení. Se vstupem do stavu a výstupem z něho může být propojena činnost. Kromě toho lze definovat reakci na vstup.

Konečný automat je dán počátečním stavem, přechodovou funkci, která určuje jak se dostat do dalšího stavu, a funkci výstupu, jež na základě vstupu a aktuálního stavu určí výstup. Automaty se dělí na dva druhy:

- Mealy, kdy pro určeni výstupu jsou podstatné kromě stavových proměnných i vstupní hodnoty,

- Moore, pro něhož platí, že vstupy neovlivňuji výstupy.

Existují též i časované konečné automaty, které přechodům přiřazují čas. [10] Konečné automaty lze aplikovat k popisu sekvenčních logických obvodů. [1]

2.2.2 Petriho síť

Jedná se o speciální typ orientovaného grafu. Skládá se z dvou typu uzlů, míst a přechodů, propojených hranami. Pro účely znázornění dynamiky je graf rozšířen o značky (tokens) umístěné do míst. Funkce přechodu zajisti přenos značky z vstupního místa do výstupního.

S pomocí Petriho sítě lze zkoumat problémy uváznutí nebo hladovění. Existuje více druhů síti, např. barevné nebo stochastické.

2.2.3 Markovovy řetězce

Markovovy řetězce představují stochastické systémy s diskrétním časem. Mohou být popsané s pomoci grafu s uzly a hranami. Metoda nemá paměť, tj. pravděpodobnost dalších stavů nezávisí na stavech předchozích. Přechody mají stanovené pravděpodobnosti. Markovské řetězy se aplikuji pro výpočet pravděpodobnosti přechodů do dalších stavů.[10]

2.2.4 Sítě hromadné obsluhy

Síť hromadné obsluhy je založena na vztahu zákazníka a obsluhy. Zákaznici přicházející do systému a generuji požadavky, které jsou řazeny do zásobníku obsluhy a následně obsluhovány. Časy příchodů a obsluhy jsou definovány stochastickými modely.

2.2.5 DEVS (Discrete Event Specified system)

DEVS zastupuje matematickou modelovací techniku, která byla vyvinuta z teorie systému. Dovoluje konstrukci hierarchických a modulárních systému. Má nadefinovaný propojovací mechanismus, který dovoluje konstrukci složitých systémů z menších jednotek.

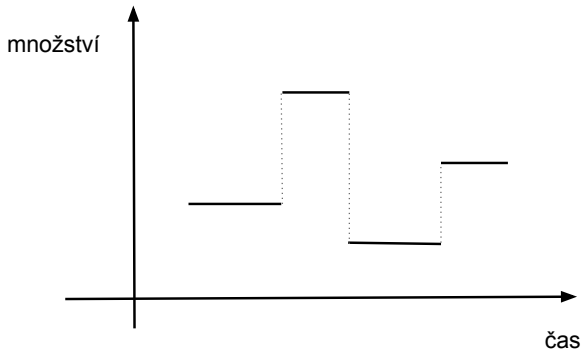
Umožňuje popis systému, jejichž chování závisí na sekvenci události. Metodu lze též aplikovat i na spojité systémy[1].

2.3 Událost

Diskrétní simulace představuje zkrácenou verzi názvu pro simulaci diskrétních *událostí*. Pravě událost představuje ústřední pojem dané techniky.

O události se mluví, pokud se něco stane, například dorazí zákazník, stiskne se zvonek nebo začne zkouška. Z formálního hlediska se s její pomocí vyjadřuje změna stavu systému.

V modelech tyto změny mohou mít různou podobu. Pokud se jedná o grafické znázornění systému s pomocí Petriho sítě, bere událost na sebe podobu přechodu. V simulačním modelu je zas změna definovaná programovým příkazem.



Obrázek 2.1: Ukázka skokového charakteru změn

Událost probíhá naráz, tj. je atomická, a pro účely simulace má nulovou dobu trvání. I když v reálném světě operace může trvat určitý časový úsek, v případě simulace tento čas se zanedbává.[1]

2.3.1 Next-Event Time Advance

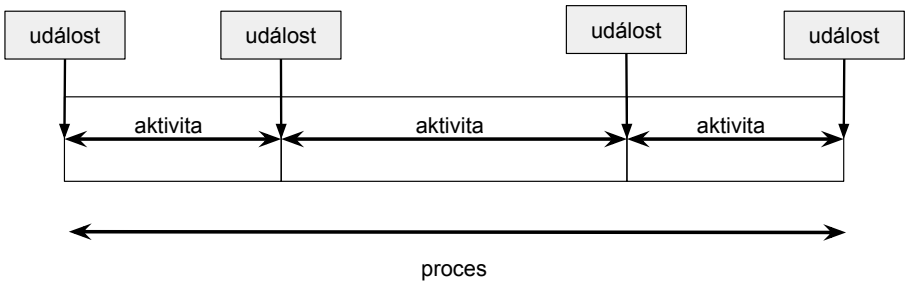
„Pohyb“ diskrétní simulace v čase dopředu je dán událostmi. Jelikož mezi tím stav systému zůstává konstantní, lze doby mezi jednotlivými událostmi zanedbávat. Na základě dané vlastnosti většina počítačových simulačních jazyků přiklání k next-event approach (přístup na základě následující události). Po proběhnutí všech změn simulární čas stávající události „skočí“ na čas další naplánované události. Tím lze vynechat časové doby, během kterých není vykazována žádná činnost, což v reálném světě není možné.[12] Na obrázku 2.1 je patrné, jak se mění stav systému na základě změny množství produktu na skladě.

2.3.2 Kalendář událostí

Kalendář událostí představuje strukturu náležící programu pro simulaci, která obsahuje prostředky pro plánování událostí.[5] Kalendář vytváří a aktualizuje posloupnost událostí probíhajícího simulačního experimentu. Jednotlivé položky jsou seřazeny vzestupně dle doby, kdy přijdou na řadu. V případě více událostí se stejným časem je výběr řízen prioritami.

2.4 Proces

Proces/transakce představuje další podstatný pojem diskrétní simulace. Je definován jako sled událostí, které navazuji na sebe dle určité logiky.[5] Jednotlivé



Obrázek 2.2: Vzah aktivita-proces-událost

událostí v procesu můžou být oddělené od sebe prostředky, které vyjadřují čekání 2.2 (aktivity reprezentující čekání ve frontě nebo operace mytí auta).

Procesy v systému probíhají paralelně a mohou spolu i komunikovat. Někdy několik procesů mohou být zapouzdřeny do objektu, který se nazývá agent. Ten posléze projevuje inteligentní chován za účelem dosažení svého cíle.

Pro zajištění paralelizmu procesů se díky jednodušší implementaci často používá tak zvaný *kvaziparalelismus*. Jedná se o techniku, kdy pro řešení paralelních procesu jsou zavedeny priority, na základě kterých jsou jednotlivé události seřazené v kalendáři a posléze jsou vykonávaný jedním procesorem.[1]

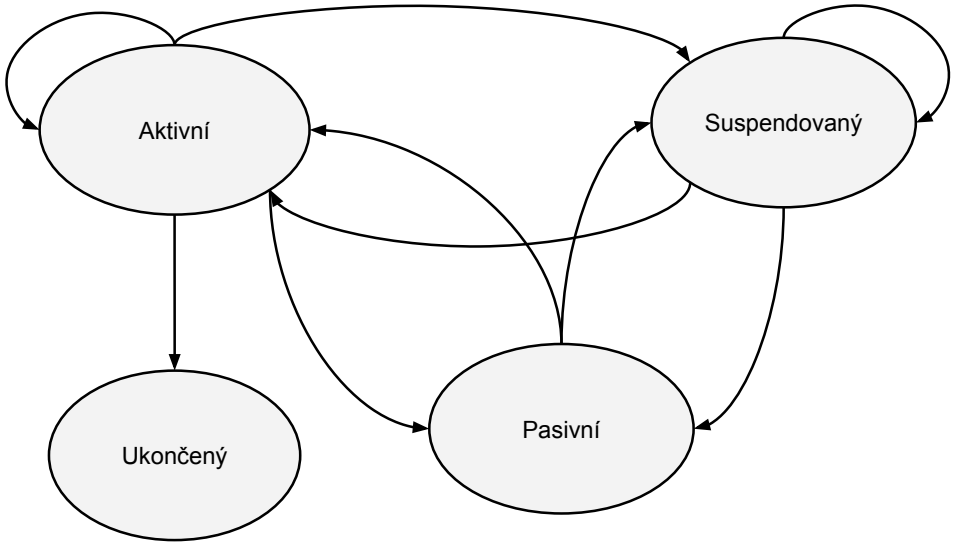
2.4.1 Stavy procesu

Proces má definován stavy vůči okolí a v průběhu svých aktivit může mezi nimi přecházet.2.3 Základem jsou stavy *aktivní* a *pasivní*. Proces je aktivní, pokud vykonává nějakou činnost. V momentě, kdy musí svou činnost přerušit a „uspat se“, např. čeká na obsluhu či prostředky, přejde do stavu pasivního. V tento moment není proces naplánován a neví se, kdy bude znovu obnoven.

Proces může též nabývat *suspendovaného* a *ukončeného* stavu. Jak lze intuitivně předpokládat, ukončený stav přísluší takovému procesu, který již provedl všechny své činnosti a už nikdy víc nebude naplánovaný. Suspendovaný stav zas říká, že uspaný proces je již naplánován v kalendáři a bude v brzké době aktivován.[5]

2.5 Druhy diskrétní simulace

Pro diskrétní simulaci jsou podstatné proces, aktivita a událost 2.2. Tyto pojmy dávají základní podnět k třem modelovacím technikám pro diskrétní



Obrázek 2.3: Možné přechody mezi stavy procesu [5]

simulaci. Jejich vývoj byl spojen s pohledy na svět simulačních programovacích jazyků. I přesto, že každý přístup je logicky správně, většina simulačních programů přiklání k implementaci událostního a procesního přístupu. Aplikace modelování na základě aktivit se ukázala být nevýkonnou ve většině simulačních scénářů. [12]

2.5.1 Event-Scheduling

Přístup je založen na modelování s pomoci událostí, které popisují změnu stavu systému. Pokud do obchodu dorazí zákazník, zvýší se počet lidi na provozovně, což způsobí změnu stavu. Obdobným případem je i odchod. Začátek a konec obsluhy též mají charakter událostí, jelikož se mění stav pokladních / pokladny z volné na obsazenou a opačně.

Po tom, co je vymezeno zadání/systém a určené možné vstupní veličiny, nastává moment, kdy se musí určit entity a jejich vlastností. Příkladem je pokladní, který může mít atributy volný a obsazený.

Jelikož modelování probíhá na základě událostí, stanoví se všechny možné stavové změny. Událostí mohou být dvojího charakteru:

- podmíněné – nastávají pouze pokud jiná událost napomůže dosažení požadovaného stavu,
- nepodmíněné – nejsou závisle na stávajícím stavu.

Jednoduché událostí lze potom spojovat do *složených událostí* (compound events), např. událost příchodu zákazníka na poštu vyvolá buď zařazení do fronty, nebo započetí obsluhy u volné přepážky. Představuje to tak možnost jak vykonat několik akcí za jednu simulační dobu (dáno atomickou vlastnosti událostí 2.3).

Diskrétní simulace následně představuje množinu seřazených složených událostí. Prostředek, jak řazení zajistit, je dán plánováním (scheduling). Událostí spolu s přiřazeným simulárním časem jsou umisťovány do kalendáře událostí (Future Event List). Potom kontrolu přebírá podprogram (time routine), který zajistí správné provedení událostí.

1. Vybere první položku kalendáře.
2. Zajistí inkrementaci simularního času na dobu události.
3. Provede událost, tj. předá kontrolu bloku s naprogramovanou logikou změny stavu, která se provede.
4. Odstraní nebo jinak zneplatní záznam události.
5. Pokud kalendář není prázdný, tak pokračuje znovu od 1.

Inkrementace simulárního času může být prováděna dvěma způsoby: pevně stanovenými časovými intervaly (fixed-time advance) nebo intervaly rozdílné délky (variable-time advance). První případ představuje jednodušší možnost k implementaci, ale pokud se modelují reálné situace, intervaly proměnlivé délky se hodí víc.

Pokud se dodržuje next-event přístup, potom blok, který zajišťuje provedení události, musí též obsahovat naplánování události další, proto příklad příchodu zákazníka na poštu bude rozšířen o vytvoření záznamu o dalším příchodě zákazníka. [12]

2.5.2 Process-Interaction

Přístup na základě procesů je mnohem jednodušší než modelování událostmi, jelikož se pracuje s něčím, co je podobné objektům. Na druhou stranu nevýhodou je menší flexibilita a nižší programová kontrola.

Každá entita systému je zastoupena procesem, který postupně prochází systémem i časem. Proces představuje posloupnost činností, např. vezmi košík, nakupuj, zaplať. Někdy proces musí vykonávat operaci čekání (např. pokud není volná přepážka na poště).

Přístup se výrazně liší tím, že na rozdíl od atomického provedení bloku události, jednotlivé části procesů (aktivity) můžou začít a skončit v různých okamžicích. Pokud se během „životního cyklu“ procesu vyskytne aktivita, která má danou dobu trvání t, vytvoří se položka s časem $t_0 + t$, kde t_0 udává

stávající simulární čas. Do položky ze přidává atribut, který stanoví místo odkud potom pokračovat (reactivation point). Po jejím vložení kontrolu získává podprogram time routine. Moment, kdy proces čeká na dokončení činnosti, se nazývá deactivation point (deaktivační bod). Pokud se v úvahu vezme, že proces obsahuje podobných míst více, lze pozorovat, jak čas prochází jednotlivými entitami.

Reaktivace procesu nemusí nastávat v pouze jasně daných okamžicích. Pokud se zákazník musí zařadit do fronty, není možné stanovit přesně, kdy na něho přijde řada. Ví se pouze, že bude obsloužen až se přepážka uvolní a před nim už nebude nikdo čekat. Z tohoto důvodu se rozlišují dva typy reaktivace: podmíněná a nepodmíněná. Nutnost podmínky je často způsobená nutnosti vyjádření „boje“ o zdroje.

2.5.3 Activity-Scanning

Activity Scanning představuje nejméně používaný princip modelování, který je založen na přístupu skrze aktivity. Splnění jednotlivých aktivit je dáno podmínkou. Základem jsou dvě fáze:

1. Simulární čas je postupně inkrementován po malých diskrétních úsecích. Během tohoto procesu se též provádí kontrola přiřazené podmínky.

2. Pokud se najde aktivita, jejíž podmínka je splněná, dojde k jejímu zpracování a obnoví se stav systému.

[13]

2.5.3.1 Three Phase Approach

Inkrementace po diskrétních úsecích je velmi nevýhodná. Někdy sice malé intervaly jsou nutnosti, ale jsou situace, kdy se provede zbytečně mnoho inkrementací. V důsledku toho byl představen přístup, který kombinuje Activity Scanning s plánováním událostí. Z první metody zůstala kontrola podmínek aktivit v určitý okamžik. Pohyb v čase se ale zajišťuje plánováním událostí.

Lze se střetnout s 2 typy aktivit:

- aktivity, které musí nastát v momentě, kdy jsou naplánované,

- podmíněné aktivity, jež v závislosti na splnění podmínky mohou nastát v čas naplánované události.

Přístup ale přináší velkou nevýhodu v podobě komplikovaného a hůře srozumitelného modelu.[13]

2.6 Příklady diskrétních systémů

Diskrétní simulace je aplikovaná v mnohých sférách: podnikání, výroba, telekomunikace, ... Tato část práce představí několik druhů diskrétních systémů, které lze v praxi potkat.

2.6.1 Systém hromadné obsluhy (Queueing system)

Systémy hromadné obsluhy (SHO) jsou určeny k popisu situací, kdy vlivem nedostatku zdrojů je entita (osoba, datový tok, dopravní prostředek, ...) nucena čekat na jejích uvolnění. Jedná se o velice častý druh modelu.

2.6.1.1 Prvky systému

Základ systému tvoří tři složky: entity, zdroje (resources) a fronty. Entity (transakce) zastupují osoby, zprávy, stroje. Zdroje (např. lidí, stroje) představují objekt, který po určitou dobu zdrží transakci při jejím průchodu (číšník příjme objednávku zákazníka). Jsou součásti pevné struktury, tzv. *báze* modelu. Zdroje, nebo-li *obslužné linky*, mohou být dvou typů: *zařízení* a *sklady*. Rozdíl těchto prvků je dán jejich kapacitou. Zatímco zařízení mohou být obsazovány nejvýše jedním procesem, sklady jsou přístupné více transakcím najednou. Prvky báze tvoří stálou strukturu, po které se pohybují procesy.

K zdrojům se pojí *fronty*. Jelikož kapacita stanovuje omezení, musí transakce čekat, až na ně přijde řada, což je zajištěno zařazováním do front.

Dalším možným prvkem systému je *brána* (gate). Jejím účelem je nahromadit určité množství transakcí, než budou moci postoupit dál. Tento prvek je ale málokdy zařazen do simulačních programů.

Do báze se též zahrnují *generátory* a *spotřebitele*. Generátory jsou zodpovědné za reprezentaci vstupů transakcí do systému (např. příchod zákazníka do obchodu). Někdy se dají použít pro reprezentaci dělení procesů, které má za následek vznik nového jedince. Spotřebitele zas představuji úbytek/odchod procesů pryč.[5]

2.6.1.2 Fronty

Samotné zařazení do fronty musí mít svá pravidla (queueing discipline). Mezi základní řadicí algoritmy patří [1]:

- **FIFO** (First-in-first-out). Pravidlo říká, že prvky jsou obsluhovány na základě doby příchodu. Čim dřív proces dorazí do fronty, tím rychleji se dostane k obsluze.

- **LIFO** (Last-in-first-out). Zde jsou zas transakce obsluhovány od konce. Pokud prvek dorazí jako poslední, odejde obsloužen jako první.

- **SIRO** (Service in Random Order). Zařazení do fronty je řízeno náhodně, a proto není předem jasné, komu se dříve poštěstí odejít.
- **Prioritní fronta**. Hlavní roli pro dané fronty hraje priorita. Představuje hlavní kriterium, dle kterého je transakce umisťována do fronty. V závislosti na implementaci prvky buď s nižší, nebo s vyšší prioritou jsou upřednostňovány před procesy s nižšími privilegií.

Vybíraní transakcí z fronty probíhá většinou ze začátku. Existuji však i jiné možnosti odebíraní, např. fronta s netrpělivými požadavky, kdy požadavky mohou frontu opustit po uplynutí určité doby. Fronta může mít stanovena omezení na délku. Převážně se pracuje s neomezeně velkými frontami, ale kapacitně omezené varianty nejsou výjimkou. Ohraničení se dá použít pro případy, kdy se pracuje s omezeními v prostoru. Zvláštní případ představuje fronta o nulové délce, kdy není povoleno čekat na obsloužení a transakce musí odejít.

Nemusí vždy platit, že k jednomu zdroji patří jedná fronta. Jednoduše lze napojit jednu frontu k více obslužným linkám a naopak. K jednomu zdroji se přiřadí několik front. [7]

2.6.1.3 Priority

Priority představují nějakou číselnou hodnotu, která je určená k rozlišování privilegovanosti. Kromě priority procesů 2.6.1.2 lze též pracovat i s prioritou obsluhy, která dovoluje přerušit probíhající obsluhu s nižší prioritou. Proces, jehož činnost byla přerušena může systém opustit, nebo se vrátí zpět do fronty.

2.6.1.4 Obslužná síť

Obslužnou síť tvoří vícero obslužných linek propojených mezi sebou. Dle vztahu k okolí lze rozlišit několik typů síti [1]:

- *Otevřená obslužná síť* provádí komunikaci s okolím v podobě posílaní požadavků.
- *Uzavřená obslužná síť* udržuje všechny požadavky v rámci systému.
- Existuje též určitý kompromis mezi prvními dvěma typy, což je *síť smíšená*. Pro ní platí, že její chování je určeno druhem požadavku. Pro nějaké se chová jako otevřená, pro druhé zas jako uzavřená.

2.6.1.5 Účely simulace SHO

Systémy hromadné obsluhy lze nasadit do různých prostředí. Používá se pro modelování nemocnic, obchodů a též i komunikaci prostřednictvím počítačové sítě. Během simulačního experimentu se pozoruje doba, kterou stráví transakce

v systému. Zkoumá se čas, po který čeká proces ve frontě, či se stanovuje vytíženost obslužné linky. Po statistickém zpracování údajů z experimentů, lze získané znalostí aplikovat pro vylepšení procesů (např. stanoví se odhad optimálního počtu otevřených pokladen pro určitou denní dobu).

2.6.2 Systémy podobné SHO

Další systémy jsou též založené na principu omezených zdrojů. Opět se jedná o případ, kdy dochází k umisťování entit do front na základě nedostatku obsluhy.

2.6.2.1 Počítačové systémy (Computer systems)

Po systému se pohybují entity, které mají podobu úkolů (tasks) a práce (jobs). Soutěží o procesory, např. disky, klávesnice, CPU. Pokud žádný zdroj není k dispozici, entity se řadí do front, které představují integrovanou část systému. Zde čekají, až na ně přijde řada.

2.6.2.2 Comunication Systems (Komunikační systémy)

Dalším obdobným systémem je komunikační systém. Entity zastupují zprávy, volání, pakety, které jsou generovány externím zdrojem (uživatelem). Cílem transakcí je dorazit ke spotřebiteli. Na své cestě prochází různými přepínacími zařízení (switching equipment), které symbolizují zdroje. Mechanismus, který zajišťuje spravedlivé a vhodné přidělování obslužných zařízení, se nazývá protokol.[7]

Kapitola 3

BORM

BORM (Business Object Relation Modelling) je metoda, která se vyvíjí již od roku 1993. Jedním z podnětů pro její tvorbu byl pojem tzv. „koncepční mezery“ (conceptual gap)[14]. Tato problematika spočívá v rozdílu mezi znalostmi a myšlenkami uživatele/experta a jejich interpretaci analytikem, který zkoumá a zaznamenává podnikové procesy. BORM se snaží překonat danou mezeru a nabízí postupy a prostředky, které mají za úkol zjednodušit integraci uživatele do procesu analýzy.

I když původně byla metoda koncipovaná k tvorbě objektově orientovaných systémů, lze ji též využít k analýze požadavků na systém a na modelování a vyhodnocování business procesů. [15] BORM kombinuje objektově orientovaný přístup s procesním, což zjednodušuje komunikaci s lidmi ze sféry podnikání.[14]

V kapitole se krátce představí metoda jako celek. Dále se zaměří na možností, které BORM poskytuje pro zkoumání objektů reálného světa, tj. metodu OBA 3.2 a procesní modely ORD 3.3.

3.1 Metodika BORM

BORM využívá spirální model životního cyklu, který je typický pro objektově orientované metodiky.[14] Cyklus pokrývá všechny stadia vývoje softwaru od tvorby business modelů až po samotnou implementaci.

3.1.1 Přístup metody BORM

Přístup metody dává objektům tři vlastnosti (atributy) [14], které jsou nazývané *dimenzemi*. Jedná se o data, chování a historii. Dimenze dat seskupuje viditelné vlastností objektů (např. typ a vlastnosti serveru) a vztahy mezi nimi. Chování je dáno aktivitami, které objekt vykonává. Stavy objektu a přechody mezi nimi zas udávají historii. Každý z těchto „pohledů“ představuje důležitou součástku. BORM kombinuje všechny dimenze do jedné grafické podoby,

což přináší větší názornost na problematiku, tj. není potřeba velkého množství oddělených diagramů jako v metodě UML (Unified Modeling Language).

Základní pojmy, které se k metodě vážou jsou *proces* a *objekt*. Pojem objektu je klíčový a v průběhu vývojového cyklu se jeho význam mění.

1. Zpočátku se jedná a *business objekt* (objekt reálného světa). Zde se zkoumají činnosti, komunikace a vztahy objektů.

2. Posléze se objektu s pomoci jasně daných technik mění na *konceptuální objekt*. Ten již obsahuje pojmy odpovídající objektově orientovanému paradigmatu.

3. Poslední podobou, kterou na sebe bere, je *softwarový objekt*. Zde jsou již důležité vlastností spojené s vybranými implementačními technologiemi.

3.1.2 Životní cyklus

Životní cyklus se skládá ze šesti fázi 3.1, které lze rozdělit do tři části o dvou krocích. V jednotlivých fázích se používá jen určitá množina pojmů, což je založeno na tom, že jednotlivé pojmy se v průběhu transformuji. Zároveň jsou používané termíny pro daný krok relevantní (př. není potřeba vědět, co je jednoduchá dědičnost, pokud se jedná o počáteční analýzu). [15] Jedná se o tyto fáze [15] a [16]:

1. *Strategická analýza* je založena na vymezení problému, určení rozsahu a stanovení procesů, které se odehrávají uvnitř a vně systému.

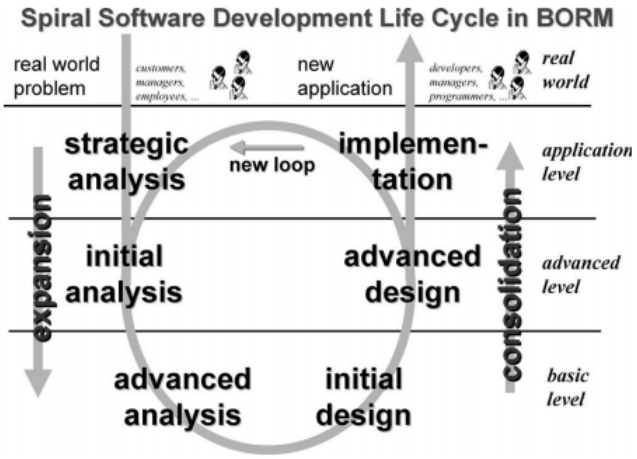
2. *Úvodní analýza* spočívá v mapovaní procesů a atributů (vlastností) objektů. Zaobírá se už situaci, jaka má být.

3. *Podrobná analýza* vede na detailní rozpracováni dosavadní analýzy, určují se „typy“ objektů (např. třídy objektů) a vazeb (dědičnost, agregace, ...).

4. *Úvodní návrh* představuje fázi, kde se již počítá s důsledným vyplnění předchozích kroků. Dochází k přípravě na implementaci.

5. *Podrobný návrh* již bere v úvahu cílové implementační prostředí a transformuje model do takové podoby, která se podřídí jeho požadavkům.

6. *Implementace* zahrnuje tvorbu a testování softwaru.



Obrázek 3.1: Životní cyklus BORM (převzáto z [14])

3.1.2.1 Stadia expanze a konsolidace

Jednotlivé výše uvedené fáze lze rozdělit do dvou skupin/stadií. První tři fáze patří do tzv. stadia *expanze*, které je ukončeno detailní analýzou. Výsledkem je konceptuální model popisující řešení z pohledu požadavků.

Další stadium *konsolidace* pokrývá zbylé tři fáze. Zde dochází ke zpracování myšlenek vzniklých expanzi do podoby produktu (aplikace). Vstupem do stadia je konceptuální model, který se postupně upravuje a transformuje. [14]

3.2 OBA (Object Behavioral Analysis)

Metoda OBA představuje techniku, s jejíž pomoci se získávají podklady pro tvorbu objektového modelu. Postup lze rozdělit do pěti kroků:

1. Zpočátku se na základě interview vymezí požadované funkce systému a identifikují se základní objekty. Dochází zde k rozpoznávají procesů a vymezení zadání.

2. Následně se tvoří scénáře z poznatků o zjištěných procesech a objektech. Určují se vlastnosti objektů.

3. Třetím krokem se definují vztahy mezi objekty a procesy. K tomuto jsou využity modelové karty.

4. V kroku modelování procesů se určí životní cyklus objektů, tj. specifikují se stavy a přechody mezi nimi.

5. Nakonec se provede validace a verifikace modelu.

Informace získané danou metodou lze znovu použít v podobných případech, zároveň je lze využít pro dokumentaci projektu.

3.3 ORD (Object Relationship Diagram)

OR diagramy slouží k modelování procesů pro metodu OBA. Jedná se o poměrně jednoduchou notaci, která kromě zachycení procesů z hlediska jednotlivých objektů, modeluje i průběh z hlediska systému.

3.3.1 Prvky diagramu

Diagram manipuluje s malým množstvím pojmů a symbolů. Následující část obsahuje popis ORD, který vychází z myšlenek vypracovaných v rámci CCM FIT.

3.3.1.1 Participant, stav a aktivita

Základním „stavebním kamenem“ diagramu je participant. Jedná se o objekt, který zachycuje osoby, organizace nebo systémy. Element obaluje další prvky, což jsou stavy a aktivity.

Stavy umožňují zachycení historie změn svého participanta v čase. Mezi nimi jsou vymezeny speciální typy stavů: počáteční a koncový stav. Mohou též obsahovat „podproces“ (vnořenou posloupnost aktivit a stavů), nebo jen jednu aktivitu.

Aktivity zas udávají chování objektů. Dá se na ně pohlížet buď jako na činnosti, které generují výstup, nebo jako na prostředek umožňující dosažení dalšího stavu.

3.3.1.2 Přechody a komunikace

Propojení stavů zajišťují přechody, na kterých se můžou nacházet akce. Jedná se o orientované spojení, kdy směr přechodu říká, jakých stavů lze z daného bodu dosáhnout.

Komunikace znázorňuje závislost mezi aktivitami a určuje pořadí jejich provádění. Je podobná komunikačnímu kanálu, kterým zároveň lze posílat zprávy – data (data flows). Nejprve iniciátor má možnost poslat data putující ve směru komunikace (input data flow), na které lze očekávat odpověď (output data flow).

Pergl a Podloucký [17] ještě zavádí rozdělení komunikací na přímou a nepřímou. Základem je, že interakce se provádí ve stejný moment, což představuje přímou variantu. Pokud však se modeluje situace, kdy si kamarádi píšou SMS, je potřeba vyjádřit, že zpráva nebude hned přečtena, a odesílatel nečeká po odeslání na odpověď, ale pokračuje dál ve své činnosti. To symbolizuje nepřímá komunikace.

3.3.1.3 Podmínky

Podmínky jsou v ORD umísťovány na přechody a komunikace. Jedná se o slovní vyjádření situace, kdy dochází k jejich provedení (např. prší/neprší, je pracovní den/je víkend, ...).

V souvislosti s možnosti simulace ORD jsou zavedené vstupní a výstupní podmínky (input/output conditions)[17], které se vážou na případy vícero výstupů z jednoho stavu a obdobně i vstupů. Jejich základ vychází ze dvou principů:

- simultaneity principle (princip současného provádění), jehož podstata spočívá v tom, že participant nemůže vykonávat několik činností naráz. Výsledkem je, že jednotlivé „větvé“ vystupující ze stavu jsou na sobě nezávislé a jejich pořadí provedení je libovolné. Též lze jejich provádění prokládat.
- dependency principle (princip závislosti), který udává to, že možnost přechodu ze stavu může být ovlivněna splněním určitých větví vstupujících do něho.

Vstupní a výstupní podmínky jsou pravdivostní výrazy, které jsou přiřazovány stavu. Výstupní podmínka stanovuje, jaké kombinace větví mohou pokračovat z daného stavu. Vstupní podmínky říkají, kdy lze z daného stavu pokračovat (reflektuje 3.3.1.3). Jejich vyhodnocení je podmíněno doběhnutím všech větví, které jsou spojené v daném stavu.

Kapitola 4

Přehled knihoven pro diskrétní simulaci

V dané části práce se představí knihovny a frameworky pro diskrétní simulaci v programovacím jazyce Java, které jsou volně dostupné. Následně se vybere jeden ze zástupců, který bude podrobněji zanalyzován z hlediska cíle práce.

4.1 Přehled knihoven a frameworku

Pro všechny představené knihovny platí, že tvorba simulačního modelu je prováděna skrze převádění myšlenek do podoby simulačního modelu v jazyce Java. Při evaluaci jednotlivých zástupců bude přihlíženo na:

1. podporu modelování na základě procesů, jelikož se tento přístup nejvíce hodí pro stávající podobu ORD. Pokud by v budoucnu došlo k striktnějšímu vymezení použití stavů a aktivit, který by více reflektoval podobu automatů typu Mealy, lze zvážit využití principu modelování na základě událostí,

2. poskytnutí generátorů alespoň základních náhodných rozdělení (např. exponenciálního, Poissonova, normálního, Bernoulliho, rovnoměrného),

3. způsob modelování (míra abstrakce modelovacích objektů, postupy prováděné během tvorby simulačního modelu),

4. aktuálnost (aktualizuje se),

5. poskytování výsledků simulace a případně i jejich statistické vyhodnocení,

6. možnost přidaní vizualizace experimentu pro větší názornost preferována.

4.1.1 J-Sim [18]

J-Sim představuje knihovnu pro procesně orientovanou diskrétní simulaci, která má své počátky na Západočeské univerzitě v Plzni. Primárně je produkt cílen na simulaci obslužných síti, ale jelikož obsahuje základní konstrukty (kalendář událostí, fronty, procesy, ...) lze J-Sim používat i pro jiné systémy.

K dispozici jsou pouze 4 generátory náhodných rozdělení: exponenciální, rovnoměrné, Gaussovo a Bernoulliho (v podobě generování pravdivostních hodnot). Lze též generovat i objekty v zadaném poměru.

Modelování je založeno na tvorbě modelů procesů a určení jejich chování prostřednictvím definování metod. Dále se programuje i průběh simulace: inicializace, vytváření front a procesů, ukončování, odchytávaní a ošetřování výjimek.

V rámci knihovny nejsou dány prostředky pro sběr dat o experimentu a jejich vyhodnocení. Lze sice získat informace o frontě (průměrná délka a doba čekání). Chybí ale zde možnosti sbíraní uživatelem definovaných dat.

Výstupy z simulace jsou buď do konzole, nebo lze interaktivně provádět simulaci skrze GUI (Graphical user interface).

Poslední aktualizace proběhla v roce 2006. Knihovna je distribuována pod licenci Academic Free Licence 3.0.

4.1.2 SimJava2 [19]

Knihovna SimJava2 je vyvíjena na Edinburské univerzitě (University of Edinburgh). Přístup modelování využivá procesního pohledu na svět. Jednotlivé entity jsou propojeny porty, skrze které se posílají zprávy 4.1.

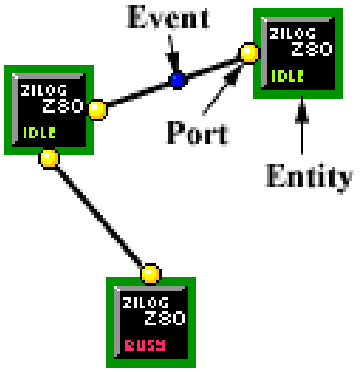
Balíček generátoru rozdělení obsahuje 5 diskrétních rozdělení a 16 spojitých. V rámci experimentu lze sbírat data, které se posléze zobrazí ve zprávě. K dispozici je zobrazovaní měřených hodnot do grafu. Lze též aplikovat analýzu výstupu z dat z vícero běhu simulace.

Uživatel má k dispozici prohlížeč grafů (SimJava Graph Viewer). Zároveň lze vizualizovat simulaci v podobě appletu spustitelného přes prohlížeč. Knihovna podléhá licenci The University of Edinburgh Academic Non-Commercial License.

4.1.3 Desmo-J [20]

Framework Desmo-J zprostředkovává simulaci a modelování na základě procesního a událostního pohledu na svět. Vývoj této knihovny probíhá na Hamburské univerzitě (University of Hamburg) již od roku 1999. Základním principem je oddělení modelu od experimentu. Uživatel tvoří programový model, který je posléze propojen s pokusem.

K modelování kromě základních tříd (proces, fronta, zdroje) jsou poskytnuté i abstraktnější objekty: složený proces (ComplexSimProcess), prostředky



Obrázek 4.1: Princip modelování v SimJava2 (převzáto z [19])

pro modelování výrobních procesů, podmíněná fronta, synchronizační struktury pro vztah producent/konzument.

Náhodné veličiny lze generovat 25 generátory. Modelovací objekty (fronty, zdroje, ...) mají sami o sobě zabudovaný statistický sběr dat, kromě toho ale lze definovat vlastní zkoumané parametry. Údaje se následně vyobrazí ve výstupní zprávě (html a xml formát).

Framework je rozšířen o možnost 2D a 3D vizualizace experimentu. Též lze využít provádění pokusu přes GUI. Produkt podléhá licenci Apache License, Version 2.0. Poslední aktualizace proběhla v listopadů 2014.

4.1.4 JavaSim [21]

JavaSim je knihovnou, jejíž existence je datována již od roku 1997. Předlohou byla knihovna C++SIM. Jako i její předchůdce vychází z poznatků tříd knihovny SIMULA, která se snažila o objektový přístup k simulaci. Poskytuje základní modelovací prostředky pro procesní přístup.

Kromě základních tříd pro simulaci, uživateli je umožněno určovat údaje pro statistické vyhodnocení a zobrazovat je v podobě histogramů. Zároveň jsou poskytnuty prostředky pro synchronizaci (semafory).

Pro generování náhodných veličin je dáno 7 generátorů. Pokrývají Bernoulliho a 5 spojitých rozdělení. Další generátor je určen pro získávaní náhodných hodnot bez daného pravidla.

Produkt podléhá licenci GNU Lesser General Public License, v. 2.1. Poslední aktualizace proběhla v roce 2015.

4.1.5 SSJ [22]

SSJ (Stochastic Simulation in Java) je nástroj pro stochastickou simulaci v Javě. Nabízí jak procesně orientovanou diskrétní simulaci, tak i modelování

na základě událostí. Za vývoj knihovny zodpovídá univerzita v Montrealu (Université de Montréal). Knihovna též poskytuje možnost aplikování metody Monte Carlo.

Pro provedení simulace procesů lze použít dvě možností: provádění paralelního běhu procesů prostřednictvím vláken, nebo použití funkcí jiného produktu – knihovnu DSol (viz 4.1.6). Pro modelování jsou poskytnuté kromě procesů a front třídy reprezentující zdroje.

Knihovna obsahuje i implementace stochastických procesů jako Brownův pohyb. Disponuje velkou listinou generátorů náhodných veličin. Obsahuje přes 60 možných generátorů diskrétních, spojitých a empirických rozdělení.

SSJ poskytuje přístup k sběru dat z experimentu a k jejich základnímu statistickému vyhodnocení (průměry, odchylky, atd.). Zprostředkovává možnost aplikace testů shod. Knihovna je distribuovaná pod licencí GNU General Public License. Poslední aktualizace byla provedena v červnu 2014.

4.1.6 DSol [23]

D-SOL (Distributed Simulation Object Library) se zaměřuje na distribuovaný a obslužně založený způsob modelování. Zaměřuje se na animaci prostřednictvím webových technologii a spolupráci s externími informačními systémy (např. databázemi). Umožňuje jak diskrétní, tak i spojitou simulaci. Primárně se zaměřuje na modelovací techniku založenou na událostech, ale podporuje i procesní pohled.

Samotné modelování se skládá ze dvou kroků: tvorba modelu v Javě a definice experimentu v XML. Pro modelování náhodných veličin je k dispozici 21 druhů rozdělení (8 diskrétních a 13 spojitých). Lze též získávat statistické výstupy v podobě histogramů, grafů a záznamů o údajích.

Základní výstup simulace je zprostředkován skrze GUI, které umožňuje jednoduchou manipulaci s experimentem (zastavování, obnovení běhu, ...). Lze ale naprogramovat i výstup do konzole nebo na jiné zařízeni.

DSol poskytuje možnost doprogramování grafického zobrazení experimentu přes interface daný knihovnou. Distribuce probíhá pod licenci BSD-style license. Poslední aktualizace proběhla v roce 2014.

4.1.7 JSL [24]

JSL (Java Simulation Library) je zástupcem frameworků pro diskrétní simulaci, který podporuje jak pohled z hlediska událostí, tak i z hlediska procesů. Podobně jako Desmo-J (viz 4.1.3) se snaží dodržovat oddělení modelu a simulačního experimentu. K dispozici jsou, kromě prvků proces a fronta, elementy spojené s obslužnými sítěmi a systémy zásobovaní. Aktivity procesů jsou brány jako položky seznamu, které se vykonávají postupně za sebou. Za jejich provedení zodpovídá funkce (coroutine).

Knihovna zprostředkovává mechanizmy pro sběr dat během experimentu a jejich vyhodnocení (průměr, minimum, maximum, atd.). Data lze též zpracovat do podoby histogramu. Výstupem simulace mohou být textové zprávy.

Uživatel má k dispozici 9 generátoru spojitých rozdělení a 8 diskrétních. Zástupci plně pokrývají množinu nejčastěji používaných rozdělení.

Knihovna je primárně určena pro vědu a výzkum v oblasti simulace, proto podléhá licenci GNU Public License. Poslední aktualizace proběhli v roce 2013.

4.1.8 Výběr simulační knihovny

Na základě analýzy stávajících volně dostupných produktů pro diskrétní simulaci byl zvolen framework Desmo-J.

Samozřejmě nelze opomenout předností konkurenčních knihoven. SSJ má velké zázemí v oblastí generování náhodných veličin, které je výborné pro simulaci stochastických procesů. DSol přináší výbornou myšlenku v oddělení popisu modelu od experimentu. Zároveň spolu s JSL uvádí nový přístup k paralelní simulaci procesů, který není založen na technologii vláken. Zajímavou myšlenkou je i komunikace skrze porty, která se vyskytuje u knihovny SimJava2.

I přes klady konkurujících produktů, zvolení Desmo-J je podmíněno přehledností poskytnutých informací o něm a dostupností abstraktnějších modelovacích konstruktů. Zároveň se nesmí opomenout i fakt, že již byl využít pro podobné účely (simulace BPMN) [25]. Benefitem je i jeho neustálý vývoj, výstupem kterého je například možnost vizualizace a parametrizovatelné GUI pro provádění experimentů.

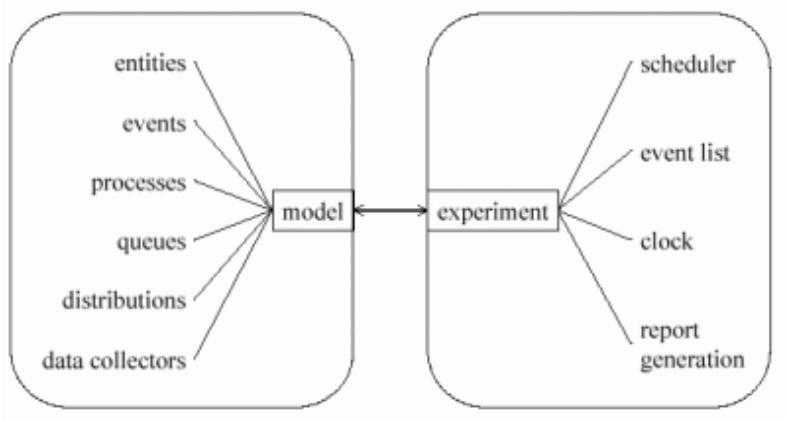
4.2 Analýza Desmo J

Cílem dané části je podrobněji zanalyzovat možnosti poskytnuté frameworkem Desmo-J. Vysvětlí se jeho struktura. Primární snahou bude popsat částí zabývající se procesním modelováním, které budou potřebné pro další vývoj práce.

Jak bylo zmíněno již v přehledu, framework podporuje oddělení modelu a experimentu 4.2, proto pro vykonání pokusu postačí definice modelu a jeho komponent, které se posléze propojí s experimentem. Daná vlastnost umožňuje znovupoužití každé ze dvou částí, tj. lze provádět více různých pokusů se stejným modelem a naopak.

4.2.1 Základní balíček

Podstatné třídy, které umožňují modelování jsou shromážděný v základním balíčku (desmoj.core), který se dále dělí na:



Obrázek 4.2: Desmo-J: oddělení modelu od experimentu (převzáto z [20])

- `desmoj.core.simulator` – obsahuje třídy nutné k modelování a následnému provázaní s experimentem,
- `desmoj.core.dist` – obsahuje třídy pro modelování statistických rozdělení. Proces generování je založen na lineárním kongruentním generátoru náhodných čísel. Můžou být vráceny pravděpodobnostní a celočíselné hodnoty nebo čísla v pohyblivé řádové čárce. Lze generovat i entity,
- `desmoj.core.exception` – obsahuje třídy výjimek, které jsou potřebné pro interní použití knihovny,
- `desmoj.core.report` – balíček obsahuje funkce k automatickému generování zpráv,
- `desmoj.core.statistic` – nalézají se zde třídy pro sběr statistických údajů z experimentu,
- `desmoj.core.advancedModellingFeatures` – obsahuje třídy, které symbolizují abstraktnější modelovací prvky, tj. podmíněné fronty, objekty pro modelování spolupráce, konstrukty pro vyjádření situace konzument/producent,
- `desmoj.core.util` – zde se nalézají třídy, které pomáhají parametrizovat a kontrolovat experiment.

4.2.1.1 Balíček desmoj.core.simulator

Na obrázku A lze vidět zjednodušenou strukturu převážně modelovacích objektů (jsou zde například vypuštěny třídy pro modelování událostí).

Model První podstatný objekt reprezentuje třída `Model`. Jedná se o abstraktní třídu, od které se dědí při tvorbě vlastního modelu. Parametry, které se do potomka přidávají, mohou být počty statických procesů, fronty, statistická rozdělení. Účel jejich umístění do třídy modelu spočívá v jejich viditelností dalšími prvky. Třída `Model` je zodpovědná za jejich inicializaci.

Proces Dalším důležitým objektem je proces, který zastupuje třída `SimProcess`. Opět při modelování uživatelem definovaný proces dědí od ní. Proces obsahuje metody aktivace (zajišťuji změnu stavu procesu na aktivní), které jsou viditelné z venku. Možné varianty „probuzení“ jsou například: aktivace v daný moment, po nějakým procesu nebo před nim. Pro další změny stavů procesů vlastní třída metody pro uspání a reaktivaci. Pro řazení do front lze procesu stanovit prioritu.

Aktivity procesu Snad nejdůležitější metodou je `lifeCycle()`. Slouží k popisu činností vykonaných samotným procesem. Metoda musí být implementována uživatelem. Aktivitu, jejíž vykonání je závisle pouze na samotné instanci, lze modelovat s pomoci metody `hold(TimeSpan dt)`, která zajistí jeho uspaní po stanovenou dobu.

Potomci procesu Desmo-J má již připravený generátor vstupních procesů – `ArrivalProcess`. Stačí mu říct frekvenci, s jakou poskytuje procesy, a druh procesů, který generuje. Další abstraktní podtřídu představuje složený proces (`ComplexSimProcess`). Slouží jako kontejner procesů. Dokud je aktivní, jeho komponenty spí.

Fronta Fronta je implementována třídou `ProcessQueue`. Řadí do sebe procesy nejprve na základě jejich priorit, a posléze dle daného pravidla (FIFO, LIFO, náhodně).

Podmínky V rámci balíčku figurují dvě abstraktní třídy, které reprezentují podmínky. První se váže na entitu/proces a je určena k filtrování pro zařazování do front. Druhá je spojena s modelem (`ModelCondition`) a stanovuje podmínku ukončující experiment.

Třídy pro experiment Součásti baličku jsou třídy zastupující základní součásti simulace: kalendář událostí (`EventList`) s položkami (`EventNote`) a simulační hodiny (`SimClock`) s časovými jednotkami (`TimeInstant`). Nad vším nahlíží plánovač (`Scheduler`), který kontroluje celý běh simulace.

Třída `Experiment` Třída `Experiment` vlastní prostředky, které umožňuji provedení experimentu s modelem. Přes instanci této třídy se nastavuje doba běhu pokusu nebo generovaní nějakých výstupů (report, trace, debug).

4.2.2 Statistická rozdělení

Balíček `desmoj.core.dist` obsahuje třídy zajišťující generování náhodných hodnot. Na základě navrácené hodnoty se rozlišují tři skupiny rozdělení:

- numerická,
- rozdělení entit,
- pravdivostní (generuje pravdivostní hodnoty).

Možná rozdělení jsou shrnuta v tabulce B.1. Pro ošetření záporných náhodných hodnot (např. u dob příchodů) lze stanovit, že jsou požadováné pouze kladné hodnoty (`setNonNegative(true)`).

4.2.3 Prostředky pro statistický sběr dat

Balíček `desmoj.core.statistic` poskytuje třídy, které umožňují statistický sběr dat, jenž lze uplatnit pro vyhodnocení uživatelem sledovaných parametrů.

- `Count` je třída, která slouží k počítáni jevů. Jejím prostřednictví lze zjistit, kolik lidí dorazilo do obchodu, počet objednávek, atd..
- Třída `Tally` umožňuje sledovat průměr a standardní odchylku pro číselné posloupnosti. Možnost aplikace lze hledat například v průměrných dobách aktivit (obsluha na pokladně).
- `Accumulate` připomíná třídu `Tally`, ale hodnoty jsou vyvažované časovým intervalem.
- `Třída` Histogram řadí jednotlivá pozorování do přehrádek a ve výsledku generuje jednoduchý graf. Zároveň zpráva poskytuje informaci o průměru, odchylce a krajních hodnotách.
- `Regression` představuje konstrukt, který vyhodnocuje dvě množiny dat s pomocí regresního testu.
- `TimeSeries` je třída, která umožňuje zaznamenávat dvojici čas-hodnota. Výstup lze posléze použít pro konstrukci grafu závislosti hodnoty na čase.

4.2.4 Možnosti grafického výstupu

Simulaci lze provádět jednoduše přes konzoli. Ale ne vždy je to pohodlné. Proto v rámci rozšíření základních tříd Desmo-J (balíček `desmoj.extensions`) vznikl tzv. Experiment Starter, který poskytuje GUI pro manipulaci s simulaci. Dostupné jsou dvě podoby: applet a aplikace.

Pro 2D animaci jsou k dispozici modelovací prvky, které dědí od základních tříd pro modelování. Z toho plyne to, že se musí vytvoří vlastní model, který umožní vizualizaci. Průběh grafického vyobrazení probíhá až po skončení experimentu.

Část II

Návrh

Kapitola 5

Rozšíření ORD

Nyní budou navržena rozšíření stávající podoby OR diagramu, která umožní simulaci procesních modelů. Základními patrnými problémy znemožňující provádění experimentů jsou:

- Diagram udává jeden průběh procesem. Nebere v úvahu existenci několika instancí participanta.
- Lze jen stěží dosáhnou simulace reálné situace, pokud nejsou k dispozici data o čase.

Body však nepředstavují plný výčet. V každé podkapitole budou proto situace v diagramu rozebrány více.

5.1 Generátory a aktivátory

Jak bylo zmíněno na začátku: diagram pracuje jen s jedním procesem. Pro simulaci je proto potřeba vyjádřit existenci vícero instancí simulovaných participantů. Z hlediska systému se rozlišují dva typy objektů: permanentní a dočasné. Pro každý druh platí jiný způsob vstupu do systému.

5.1.1 Generátor

V kapitole o SHO 2.6.1 byl představen generátor procesů, který modeloval příchod entit do systému. Tento prvek lze též využít pro OR diagram.

Generátor byl určen pro dočasné procesy (transakce) procházející systémem. Z toho plyne, že se v generátoru musí stanovit způsob, jak participanti dorazí. Dle znalostí chování entit lze stanovit dobu mezi příchody např. s pomocí exponenciálního rozdělení. V úvahu ale připadá i možnost, že by doba mezi novými jedinci byla konstantní.

Zároveň může uživatel mít zájem počet vygenerovaných instancí participanta omezit, proto by generátor měl obsahovat parametr stanovující číselnou hranici.

Na základě těchto poznatků vzniká návrh objektu *Generátor*. V rámci něho je položka určující příchody instancí v podobě statistického rozložení s parametrem/parametry, nebo pevně stanovené hodnoty. V případě, kdy knihovna poskytuje konstantní rozložení (což platí pro Desmo-J), lze výše zmíněné vlastnosti objektu sloučit do jednoho parametru. Zároveň prvek obsahuje volitelný číselný parametr, který udává maximální počet vygenerovaných instancí.

5.1.2 Aktivátor

Permanentní prvky existuji v systému již od jeho vzniku, proto pro ně nelze použít generátor. Na druhou stranu při začátku simulace je potřeba instance „probudit“. Tento proces se nazývá aktivace.

Před spuštěním simulace musí být řečeno, kolik instancí se vytvoří a následně umístí do modelu.

V důsledku daných argumentů lze navrhnout objekt *Aktivátor*, který bude sloužit k aktivaci tolika prvků, kolik bude stanoveno jeho parametrem.

5.1.3 Umístění prvků

Navržené prvky se sice pojí na participanty, na druhou stranu jsou ale součásti modelu. Zároveň z logického hlediska příchody nových jedinců by neměly být ovlivněny již existujícími instancemi. Z tohoto důvodu je nelogické umisťovat navržené prvky do participantů.

Nabízí se možnost, kdy elementy budou náležet modelu jako celku a budou obsahovat informaci, k jakému participantovi se pojí 5.1. Daná myšlenka popisuje návrh umístění dříve představených objektů.

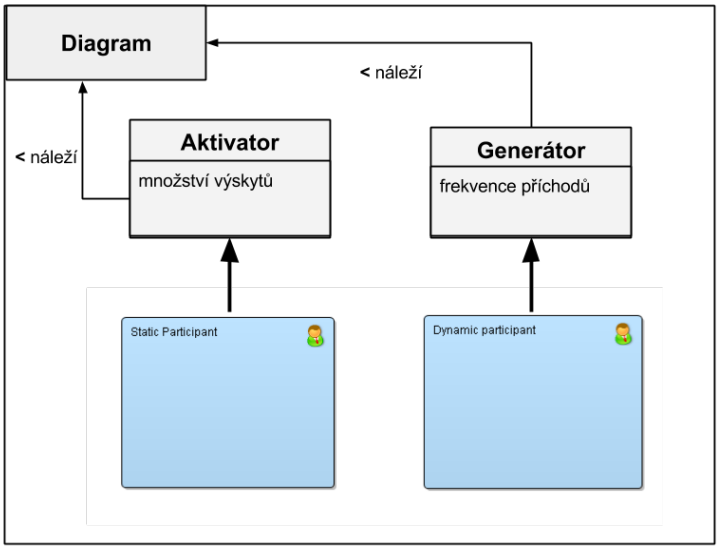
5.2 Stavy a aktivity

Pokud se pohlíží na aktivity jako na činnosti, které mění stav, a na stavy jako na vyjádření historie participanta, potom by čas byl konzumován jen aktivitou. V praxi jsou ale občas tendence přiřazovat stavům činnost, což z části odporuje definici stavu v BORM 3.3.1.1.

5.2.1 Aktivity

Aktivity bezesporu spotřebovávají čas, proto v rámci nich musí být definován atribut, který určí kolik času potřebují.

Samozřejmě lze říct, že některá aktivita trvá pokaždé 10 minut, ale častějšími případy jsou hodnoty z rozdělení. Proto obdobně jako u generátoru je navrženo, aby atribut měl dvě podoby (nebo v případě konstantního rozdělení postačí jedná). Z dané úvahy plyne návrh přidaní parametru času do prvku reprezentujícího aktivitu.



Obrázek 5.1: Vztah diagramu a generatoru/aktivatoru

5.2.2 Stavy

Jak bylo výše konstatováno, stav může mít povahu činnosti. Proto i zde by měl figurovat atribut udávající dobu trvání práce.

Problém ale spočívá v tom, že stav může v sobě obsahovat vnořenou aktivitu či proces. 3.3 Existují proto dvě možnosti, jak přistupovat k modelování času.

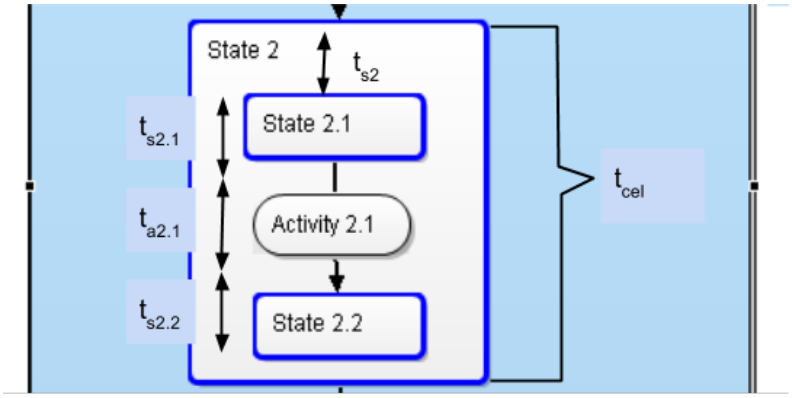
- Lze určit celkový čas provedení stavu, tj. t_{cel} na obrázku 5.2.
- Druhá možnost spočívá v sčítání jednotlivých dob vnořených aktivit a stavů s režii samotného stavu ($t_{s2} + t_{s2.1} + t_{a2.1} + t_{s2.2}$).

První možnost je velice přízniva z hlediska implementace, jelikož lze přeskočit průběh celého vnitřního procesu a provést jen stav.

Na druhou stranu ve vnitřním procesu mohou být prvky s jiným náhodným charakterem. Zároveň je jednodušší určovat vlastnosti menších složek než většího celku. Z těchto důvodu se navrhuje přiřazovat stavům časové údaje, které náleží přesně jim (použití druhé varianty, viz 5.2.2).

5.2.3 Počáteční hodnoty

Jelikož stavy ne vždy vyjadřují činnost, musí být stanovena počáteční hodnota navrženého parametru na 0. Zároveň tuto vlastnost lze použít i u aktivit. Pokud při pokusu se zanedbávají určité momenty (př. předání účtenky), bude hodnota automaticky inicializována na 0.



Obrázek 5.2: Určování časů u stavu s vnořeným procesem

5.3 Komunikace a data

Pokud komunikace je vnímaná jako nevázaní spojení, lze předpokládat, že bude trvat nějaký čas (např. čekání než se ozve volaný). Opět se jedná o přiřazení atributu času prvku. Též bude platit stanovení výchozí hodnoty jako v případě stavů a aktivit.

5.3.1 Data

Zajímavým prvkem jsou data, která putují skrze komunikaci. Jedná se vlastně o objekty, které se předávají. V průběhu průchodu procesem se mohou data přeměňovat na jiná (např. objednávka na produkt se změní v její realizaci). Pokud by při každé komunikaci docházelo k jejich vytváření, potom by tok dat procesem neodpovídal skutečnosti, proto musí data kromě stávající podoby znát i případnou svoji budoucí podobu (transformaci).

Zváží-li se příklad s objednávkou, tak převážně se očekává, že výsledný produkt doputuje k tomu, kdo si o něho požádal. Podobným problémem se v BPMN zabývá korelace (correlation). [26] Uživatel by proto měl mít možnost označit data, u kterých podobna situace nastat může.

5.3.1.1 Návrh

V důsledku předchozí rozvahy se navrhuje rozšířit položku dat o dva parametry:

- možnou budoucí podobu,
- určení, zda je potřeba vědět, ke komu se data vrací.

5.4 Fronty

Fronty jsou častou součástí diskrétních systémů, proto v návrhu nesmějí být vynechány. Jak již bylo dříve zmíněno, jejich existence je podmíněna bojem o zdroje, který nastává při kontaktech mezi zdroji a entitami. V ORD takovým místem je komunikace.

5.4.1 Fronty a participanty

Prvními prvky, které potřebují místo pro čekání na obsluhu jsou participanti. Daná situace odpovídá základnímu konceptu SHO 2.6.1. V praxi můžou nastat dva případy:

1. pro několik zdrojů stejného typu existuje jedna společná fronta, kam se řadí entity,

2. každý zdroj má svoji vlastní frontu a entita si vybírá, kam se zařadí.

Při rozboru daných typů situaci vypadá proces s frontami takto:

1. Entity, které mají zájem o zdroje se kouknou, zda je nějaký zdroj k dispozici. Pokud není, stoupne si do fronty, kde čeká. Zdroje, pokud je jich víc než 1, by měli taky mít místo, kde se nachází, než budou potřeba. Toto opět vede na nutnost fronty, tentokrát ale pro zdroje.

   Z dané úvahy plyne, že se na komunikaci budou nacházet nejvýše dvě fronty: jedna pro entity, druha pro zdroje.

2. V případě, že každý zdroj má svoji vlastní frontu, odpadá nutnost fronty pro ně. Místo, kde čekají, je pevně stanovené přesně před jich frontou V dané situaci si entita vybere frontu, kam se zařadí.

Z výsledné analýzy plyne, že na jedné komunikaci se mohou nacházet nejvýše dvě fronty. Jedna bude pro entity a druha pro zdroje. V případě, že každý zdroj vlastní svoji frontu, je na komunikaci umístěna pouze jedna, která bude určena pro entity.

Ne každá komunikace musí obsahovat frontu. Pokud dva participanti již spojení jednou navázali a nadále během procesu v něm pokračují, neumisťuje se na komunikaci ani jedna fronta.

Fronta může mít stanovenou délku. Primárně by byla neomezená, ale pokud je modelován případ, kdy se ví, že například v místnost se vejde maximálně 10 lidí, potom se stanoví hranice v podobě kapacity.

Kromě délky je důležité určit i jak se budou prvky fronty řadit, což je dáno pravidlem řazení (viz 2.6.1.2).

5.4.1.1 Návrh

V důsledku úvahy se navrhuje objekt *Fronta* určený pro participanty, který bude umisťován na komunikaci. Fronta má přiřazenou znalost o tom, kdo do ní vstupuje. Kromě toho zná svojí délku a pravidlo řazení.

Obrázek 5.3 ukazuje jaké kombinace umístění front mohou nastat v souladu s myšlenkou daného návrhu. Zároveň platí, že i pro případy, kdy k dispozici je pouze 1 zdroj, umisťuje se na komunikaci pro něho fronta.

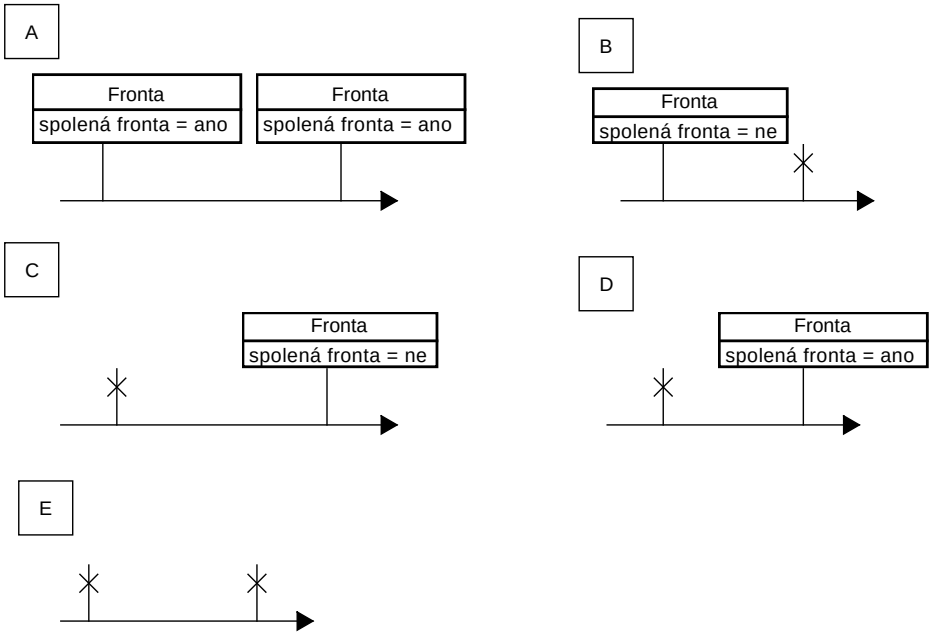
- Případ A z obrázku 5.3 odpovídá situaci, kdy participant navazuje komunikaci s libovolným procesem z druhé fronty.
- Příklad B lze ztotožnit s přepážkami na poště. Ke každé přepážce je připojena svá fronta a zákazník navazuje komunikaci. (poznámka: Při budoucím převodu procesního modelu na simulační model dojde k vytvoření fronty na opačné straně než je umístěna v diagramu. Do nově vytvořené fronty se umístí instance participanta přijímajícího komunikaci s jejich vlastními frontami.)
- Případ C je méně pravděpodobný z hlediska uskutečnění, přesto byl zahrnut, jelikož se nedá vyloučit možnost, že by nastál.

  V daném případě je iniciátorem komunikace obsluha. Každá její instance vlastní frontu se zákazníky, ze které vybírá jedince. Následně je vybraný jedinec osloven obsluhou.
- Příklad D je primárně určen případy, kdy je potřeba vyjádřit, že příjemcem zprávy může být jakákoli instance komunikaci přijímajícího participanta. Firma produkuje identické balíčky a rozdává je bez ohledu na to, jak přicházely objednávky. (poznámka: Při převodu na simulační model se daná situace ztotožní s případem A, tj. vzniknou 2 fronty.)
- Situace popsáná v části E se vztahuje na případy, kdy instance v komunikaci pokračují, nebo se určení příjemce řídí posílanými daty (např. produkt za objednávku).

Při implementaci by mělo být ošetřeno, aby nemohly vznikat jiné případy umístění front. Kontrolu lze provádět ještě na úrovni modelování, kdy uživatel bude na chybu upozorněn, nebo při převodu ORD na simulační programový model, kdy se chyba buď nějak opraví, či opět se upozorní uživatel, který ji ošetří.

5.4.2 Fronty a data

Někdy jsou případy, kdy je potřeba vyjádřit, že se hromadí data (př. e-maily ve schránce). Posléze se vybírají a zpracovávají. Podobné situace nastávají, pokud dochází k nepřímé komunikaci.



Obrázek 5.3: Možné kombinace umístění front (dle návrhu)

Zároveň se musí popsat situace, kdy participant reprezentující obslužnou linku o kapacitě větší než 2 (např. kuchyně), přijímá úkoly, které posléze plní. Průběh procesů se vztahuje jen na data a nemusí zdržovat participanta, který je předává.

5.4.3 Návrh

Tyto dva argumenty vedou k návrhu fronty pro data, která by se umisťovala do participantů. Fronta má znalost o datech v ní skladovaných. Zároveň obsahuje další parametry podobné frontě pro participanty: maximální délka a pravidlo řazení.

5.5 Priority, kapacity a podmínky

5.5.1 Priority

Pokud bude mít uživatel zájem o rozlišení nadřazenosti/podřazenosti dat či participantů, musí být zavedeny priority. Jejich primární účel tkví v přednosti při řazení do front.

Další účel parametru priority se vztahuje na příklad, kdy k více zdrojům stejného typu jsou přiřazené vlastní fronty. Potom priorita určuje zdroj, který bude upřednostněn.

5.5.1.1 Návrh

Návrh spočívá v parametru, který bude přiřazen participantovi nebo datům. Parametr bude říkat, zda priority instancím přiřazovat či ne.

5.5.2 Podmínky

Podmínky, které náleží přechodům a komunikacím nesou charakter náhodné veličiny, která má dvě varianty: buď nastane, nebo nenastane. V důsledku je navrženo přiřadit podmínce pravděpodobnost příznivého výsledků. Při větvení transakcí musí platit, že součet pravděpodobností všech podmínek je roven 1.

5.5.2.1 Výstupní podmínky a procházení větví

S zavedením výstupních a vstupních podmínek byl zmíněn simultaneity principle 3.3.1.3. Pokud po vyhodnocení výstupní podmínky bude zjištěno, že ze stavu pokračují dvě a více větví, potom je potřeba říct, jak budou prováděny:

- náhodně se zvolí jedná a po jejím dokončení se vybere další,
- nebo se činnosti procesů budou náhodně prokládat.

Nastat může i situace, kdy uživatel opravdu chce, aby simulace větví šla paralelně. Tato možnost by měla být též umožněna.

Návrh Výsledkem daných úvah je návrh zavedení dalšího parametru pro stav, který obsahuje výstupní podmínku. Parametr udává způsob, jakým se budou větve provádět (náhodně po jedné, s prokládáním, paralelně.

5.5.3 Kapacity

Kromě kapacity fronty, o které se mluvilo dříve 5.4, rozlišuje se kapacita obslužné linky (viz 2.6.1.1). Kapacita udává maximální počet procesů, které může zařízení obsloužit.

Jelikož danou vlastnost prvky, které jsou modelovány v ORD, mít můžou, navrhuje se participantům přidat parametr udávající jejich kapacitu. Výchozí hodnota parametru je rovna 1.

5.6 Experiment

Kromě diagramu musí mít uživatel i možnost, jak ovlivnit samotnou simulaci, proto do modelovacího procesu je nutné zařadit položku, která bude reprezentovat experiment.

5.6.1 Ukončení pokusu

První, co představuje podstatnou informaci, je znalost o ukončení pokusu. Simulační program musí vědět, kdy svou práci ukončit. Způsob, jak lze informaci reprezentovat má dvě podoby:

- stanoví se pevný časový údaj, po uplynutí kterého musí dojit k ukončení běhu experimentu,
- určí se podmínka, po jejíž splnění dojde k zastavení simulace (např. do obchodu dorazilo 100 zákazníků).

5.6.2 Špatně konstruovaný model

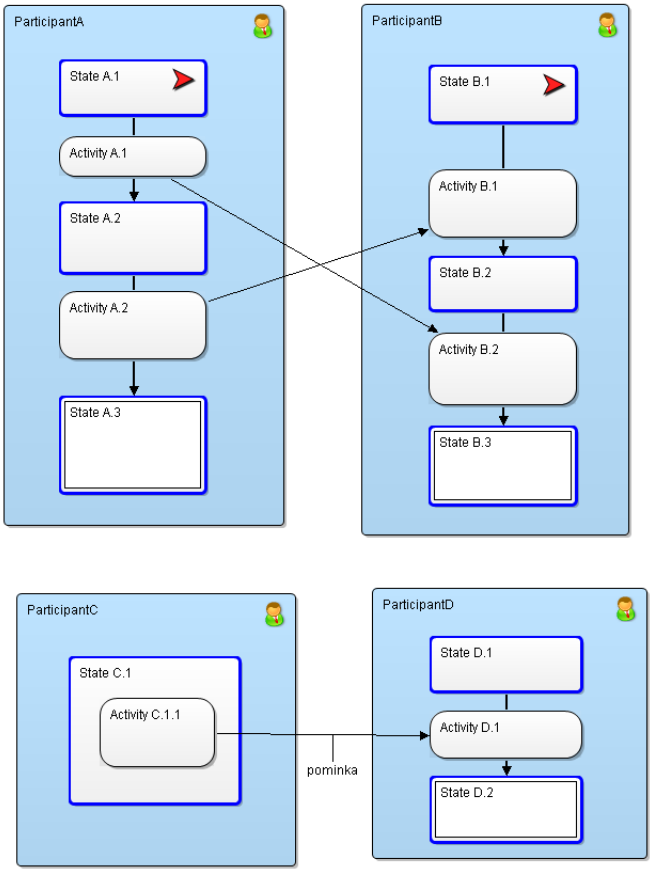
Uživatel může chybovat a vytvořit diagram, který nemůže být proveden. V době psaní dané práce se touto tématikou ve své bakalářské práci zabývá Jaroslav Bambas. V rámci výzkumu nalezl možná uváznutí způsobená špatným modelem (viz příklady 5.4).

V důsledku vyplývá, že musí být stanovena podmínka, kdy nemá smysl dále v experimentu pokračovat, protože je chyba v na straně modelu. Jelikož při uváznutí simulace nevykazuje aktivitu, nabízí se řešení v podobě možnosti stanovení maximální doby „pasivity“ daného experimentu. Po uplynutí stanoveného času by experiment byl násilně ukončen.

5.6.3 Další parametry

Kromě samotného běhu experimentu uživatel by měl mít šanci stanovení podoby průběhu simulace (s vizualizací, do konzole, atd.). Zároveň musí být poskytnut přehled variant výstupů z experimentů a možnost výběru z nich. Dané parametry jsou velice závisle na simulační knihovně. Mezi možné hodnoty parametrů patří (vychází se z možnosti knihovny Desmo-J):

- výstup do konzole,
- přidání indikátoru průběhu (progress bar),
- GUI pro manipulaci s experimentem,
- 2D a 3D vizualizace,
- zprávy (report, debug, trace).



Obrázek 5.4: Příklady uváznutí v ORD (výsledky výzkumu J. Bambase)

5.6.4 Návrh

V důsledku představených úvah se navrhuje do ORD přidat objekt reprezentující experiment. Experiment obsahuje parametr udávající moment, kdy skončí. Parametr může mít dvojí podobu:

- jasný číselný údaj,
- podmínku/podmínky pro ukončení pokusu.

Dále objekt obsahuje údaj o maximální době pasivního stavu. V neposlední řadě k dispozici jsou parametry spojené s možnostmi vizualizace a výstupů, které se řídí schopnostmi simulační knihovny.

5.7 Definice sledovaných parametrů

Některé knihovny pro diskrétní simulaci poskytují informace o experimentu v podobě údajů o době běhu, průměrné délce fronty nebo průměrných čekacích dobách ve frontě. Uživatel by ale měl mít možnost stanovit si vlastní parametry, které by chtěl sledovat. V dané sekci se bude počítat s možnostmi zvolené knihovny Desmo-J.

Jak bylo uvedeno v 4.2.3, knihovna poskytuje třídy, kterými lze data sbírat. V důsledku stavu, činnosti nebo dalších faktorů je instance třídy aktualizovaná nějakou hodnotou, například s každým obslouženým zákazníkem se číšníkovi zvýší plat o 50 korun. Uživatel by měl možnost si definovat sledované parametry a určit, kdy a jak dochází k jejich aktualizaci.

V ORD aktualizaci parametrů může provést stav, aktivita, komunikace nebo data. Statistické údaje mohou být sbírány v rámci celého modelu, participanta, instancí participanta ..., proto k parametru kromě znalosti o aktualizaci musí být přidána informace o tom, k čemu se vztahuje.

5.7.1 Návrh

V důsledku úvahy je navržen prvek, který bude existovat v rámci modelu. Jeho parametry budou:

- třída pro sběr statistických dat,
- prvek ORD, ke kterému se vztahuje, a zda se jedná o instanci či o všechny zástupce

Pro aktualizaci hodnot v rámci jednotlivých prvků ORD budou přidávány položky o dvou částech: navržený prvek pro sledovanou hodnotu a způsob aktualizace (např. u položky A s typem `Count` bude záznam vypadat takto: A; +20).

Kapitola 6

Integrace vybrané knihovny do OpenCABE

Tato kapitola se zabývá návrhem, jak propojit vybranou knihovnu Desmo-J a modelovací nástroj OpenCABE s cílem simulace procesních modelů ORD. Jedná se hlavně o zajištění převedení OR diagramů do podoby simulačního modelu v jazyce Java. Zároveň musí dojít k přidaní rozšíření navržených v 5 do nástroje OpenCABE. V neposlední řadě se navrhne spojovací prvek Desmo-J a OpenCABE. V průběhu práce na návrhu se uvažovalo o dvou možnostech, jak dosáhnou cíle.

1. První možnost počítala se složitým modulem (pluginem), který by vyprodukoval kód odpovídající simulačnímu modelu.

2. V druhém případě byla zvážená možnost rozšíření modelovacích tříd Desmo-J tak, aby vznikly prvky, které budou více odpovídat částem ORD. Nové objekty by obsahovaly popis chování prvků, které reprezentují. Daný krok zjednodušuje proces tvorby simulačního modelu, jelikož část logiky již bude definována v rámci nových tříd.

Pro další zpracování byla zvolena druhá možnost, která zároveň líp odpovídá povaze objektově orientovaného programování, proto daný návrh bude více vyhovovat pro implementaci v Javě.

6.1 Rozšíření Desmo-J

Jak již bylo řečeno v úvodu kapitoly, jeden z kroků návrhu propojení představuje rozšíření Desmo-J.

1. Nejprve se navrhne rozšíření modelovacích prvků knihovny (jednoduché znázornění objektů viz A.5).

2. Posléze se představí návrh, jak zajistit zastavení experimentu, pokud byl uživatelem poskytnut neproveditelný model.

3. Nakonec se popíše objekt zastupující model, který zapouzdřuje i logiku, jež bude použita při generování simulačního modelu.

6.1.1 Stav a aktivita

Prvními navrhovanými prvky, které rozšiřují knihovnu Desmo-J, jsou stav a aktivita (objekty `ORDState` a `ORDActivity`). Odpovídají stavům a aktivitám z ORD s navrženými rozšířeními popsanými v 5.2.

V daném případě se na stavy a aktivity pohlíží jako na orientovaný graf. Pro posloupnost aktivit a stavů bude dále používán pojem proces (procesem je i jedna aktivita nebo stav).

Všechny společné vlastnosti aktivity a stavů lze shrnout do pojmu uzel procesu. Z daného důvodu lze říct, že `ORDActivity` a `ORDState` dědí od objektu `ORDProcessNode` (název je inspirován metamodelem BORM A.2 z OpenCABE), který je potomkem třídy `ModelComponent`.

6.1.2 ORDProcessNode

`ORDProcessNode` je rodičem `ORDActivity` a `ORDState`. V něm se nachází všechny společné atributy a chování, které uzlům procesu náleží.

Parametrem objektu je odkaz na statistické rozdělení s příslušným parametrem/parametry. Obsahuje dvě metody:

- První metoda je abstraktní a její chování bude popsáno v potomcích. Jejím účelem je vracení následujícího uzlu, který bude zpracováván.

- Druha metoda sděluje časový údaj pro daný uzel (příklad algoritmu viz 1).

Algorithm 1 Vracení časového údaje uzlem procesu

```
if distribution ≠ NULL then
    return distribution.sample()
else
    return 0
end if
```

6.1.2.1 ORDActivity a ORDCommunication

Od `ORDProcessNode` dědí objekt `ORDActivity`. Jelikož s aktivitou je spjata i komunikace, bude v rámci dané části představen objekt, který uchovává informace o komunikaci (`ORDCommunication`).

ORDCommunication Účelem objektu je reprezentace komunikací. V rámci něho existují dané parametry:

- informace, zda vystupuje, nebo vstupuje do aktivity,
- údaj, zda je komunikace přímá, či nepřímá,
- fronta pro participanty na začátku komunikace,
- fronta pro participanty na konci komunikace,
- položky předávaných dat (0 – 2 položky)
- odkaz na Bernoulliho rozdělení s parametrem daným pravděpodobnosti splnění podmínky,
- odkaz na rozdělení, určující dobu navázaní komunikace,
- druhy participantů, které jsou s komunikaci spojené.

Pokud komunikace nemá nějaký z popsaných parametrů specifikovaný, doplní se výchozí hodnoty (NULL, 0).

Chování vyhodnocení podmínky proveditelnosti komunikace je dáno tímto postupem:

1. pokud rozdělení je přiřazeno, vrátí výsledek z něho,
2. jinak vždy vrací, že je komunikace proveditelná.

ORDActivity Kromě seznamu komunikací, který aktivita může obsahovat, disponuje prvek informaci z rodiče a odkazem na další prvek. Tím má i povinnost definovat chování metody, která vrací následovníka. Vracen je uzel, na který se odkazuje aktivita.

Pokud aktivita obsahuje více komunikací, zajišťuje určování pořadí jejích zpracování. Uchovává informace o tom, co již bylo předáno k provedení a co ještě bude. Navrhovaná logika určení pořadí je dána těmito pravidly:

1. komunikace vstupující do aktivity má přednost,
2. komunikace s dočasným procesem má přednost,
3. přímá komunikace má přednost,
4. komunikace s daty má přednost,
5. pokud jsou si komunikace rovné, je výběr řízen náhodně.

Kontrola pravidel postupuje dle pořadí až do splnění některého z bodů.

6.1.2.2 ORDState

Navrhovaný objekt `ORDState` na rozdíl od objektu aktivity může obsahovat proces v podobě parametru (odkaz na počáteční uzel procesu). Z tohoto důvodu je i složitější návrh metody poskytující následovníka. Kroky metody vypadají takto:

1. pokud parametr podprocesu je prázdný nebo podproces je splněn, vrátí se prvek, na který ukazuje odkaz,
2. pokud podproces existuje, zavolá se tatáž metoda na aktuální uzel podprocesu a vrátí se jeho výsledek. Pokud podproces se ukončil, bude navrácen odkaz na následovníka.

Daný algoritmus ale ještě není úplný. Stavu může náležet víc než jeden odkaz na následující uzel, proto pro daný parametr bude použit pojem seznamu následovníků. Každé položce v seznamu může být přiřazena pravděpodobnost toho, že k přechodu dojde. Daná vlastnost je zaznamenána seznamem Bernoulliho rozdělení o odpovídajících parametrech. Zároveň stav má určený způsob provádění větví (viz 5.5.2.1).

Pokud stav obsahuje pouze jednoho podmíněného následovníka, rozšíří se algoritmus jen o vyhodnocení podmínky před přechodem. Pokud podmínka nabude pozitivní hodnoty, vrátí se následující uzel, jinak se vrátí indikátor ukončení procesu (např. NULL). To bude mít za následek konec procesu nebo větve. Pokud by stav nebyl koncovým, doporučuje se upozornit na to uživatele (např. s pomocí výjimky), jelikož nejspíš došlo k chybě v logice modelu.

Vstupní a výstupní podmínky Pokud ale stav obsahuje větvení (obsahuje vstupní a výstupní podmínky) bude obsluha probíhat jinak (logika zpracování bude vysvětlena u návrhu objektů participanta 6.1.4). Pro tyto účely se navrhuje metoda, která zajistí vyhodnocení výstupní podmínky a vrátí možnou množinu větví, které podmínku splňuji. Vyhodnocení by mělo probíhat dokud výraz výstupní podmínky nebude pravdivý. Zároveň by měla existovat metoda zajišťující obdobné vyhodnocení vstupní podmínky.

6.1.3 Data

Pro reprezentaci dat se navrhuje objekt `ORDDataFlow`. Kromě vlastního jména obsahuje parametry:

- název položky dat, na kterou se mění,
- volitelnou prioritu pro řazení,
- volitelnou množinu instancí participantů, které s daty pracovali (účel parametru je vysvětlen na příkladě s objednávkou, o které se diskutovalo v 5.3.1),

- rozlišení, zda jsou data vstupní nebo výstupní (přejata vlastnost z metamodelu BORM A.2).

6.1.4 Participant a jeho chování

Nejsložitější z hlediska popisu chování je participant. Navrhuje se zastoupit tento prvek objektem `ORDParticipant`, který dědí od `SimProcess`.

Z hlediska systému se rozlišují prvky permanentní a dočasné, proto vymezení jen jedné třídy pro participanty je nedostačující. Sice zpracování stavu, aktivit, komunikaci, atd. je stejné, permanentní participanti však po ukončení jednoho průchodu procesem začínají opakovat postup znovu, zatímco dočasní participanti ze systému jednoduše zmizí. Z daného důvodu se z objektu `ORDParticipant` navrhuje podědit objekty `ORDPermanentParticipant` (viz 6.1.4.3) a `ORDDynamicParticipant` (viz 6.1.4.4).

Dále se v této části rozebere struktura a chování objektu, který reprezentuje zařízení o kapacitě větší 1 (sklad, viz 2.6.1.1). Procesem z hlediska vybrané knihovny je i generátor participantů. Při popisech chování v dané části práce bude pro instanci participanta používán i pojem proces.

6.1.4.1 ORDParticipant

Všechny společné rysy se navrhuje umístit do objektu `ORDParticipant`. Participant obsahuje ukazatel na začátek grafu, který odpovídá posloupnosti stavů a aktivit participanta z ORD. Zároveň si udržuje seznam dat, se kterými pracuje, a seznam procesů, se nimiž komunikuje. Volitelnou složkou objektu je fronta dat a fronta procesů. V rámci svého životního cyklu, vlastní proces seznam instancí participantů, se kterými komunikuje, a ukazatel na zpracovávaný uzel. V daném návrhu je proces zodpovědný za:

- zpracování činnosti stavu,
- zpracování činnosti aktivity,
- vyřešení komunikace,
- zajištění průchodu větvemi.

Základní osnova pro vyřešení jednoho uzlu vypadá takto (jedná se vlastně o navrhovanou metodu objektu `ORDParticipant`):

1. pokud uzel není aktivitou, která obsahuje vstupující komunikaci, **zpracuje se uzel** 6.1.4.1,
2. pokud je daný uzel aktivitou, ověří se, zda obsahuje komunikaci,
   a) v případě, že komunikace není k dispozici, pokračuje se dál na 4,
   b) jinak **se zpracuje komunikace** 6.1.4.1,

3. pokud je daný uzel stavem, zjisti se, zda se v dalším kroku větví (už je proveden vnitřní proces a ze stavu vystupuje víc než 1 přechod)

   a) v případě, že k větvení nedochází, pokračuje se dál 4,

   b) jinak **se zpracuje větvení** 6.1.4.1 a přeskočí se žádost o následovníka,

4. aktuální uzel se požádá o jeho následovníka.

V popise jsou zvýrazněná „chování“, jejíž podoba bude dále rozebrána. Jedná se o návrh dalších metod objektu `ORDParticipant`.

Zpracování uzlu Daný příkaz zastupuje zpracování časové hodnoty, která byla přiřazena stavům a aktivitám v rámci návrhu 5.2. Knihovna Desmo-J poskytuje metodu, která zajistí pozdržení procesu na určitou dobu 4.2.1.1. Právě danou metodou se modeluji činnosti, které v reálném životě spotřebovávají čas. Návrh metody je potom velmi prostý. Stačí jen uspat proces na dobu, kterou vrátí zpracovávaný uzel.

Zpracování komunikace Pro daný návrh bude pro začátek komunikace použit pojem výstupní komunikace a pro její konec (v diagramu část komunikace, kde je hlavička šipky) vstupní komunikace. Návrh metody spočívá ve zpracování každého případu zvlášť.

1. Dokud aktivita obsahuje nevyřízené komunikace, zjistí se, zda komunikace vrácena aktivitou je vstupní,

   a) pokud ano, **zpracuje se vstupní komunikace**,

   b) jinak **se zpracuje výstupní komunikace**.

2. Aktuální aktivita se zažádá o další komunikaci.

Jednotlivé kroky příkazů zpracování vstupní komunikace a zpracování výstupní komunikace jsou z důvodu lepší přehlednosti zobrazeny diagramy A.3 a A.4. Doporučuje se při implementaci kroky rozdělit do více metod. Zpřehlední to kód a zároveň při vhodném členění lze metody znovu používat. Daný návrh se řídí možnými kombinacemi front, které byly vysvětleny v 5.4.1.1 (bere se v potaz situace front po převodu do simulačního modelu, tj. platí informace z poznámek).

Shrnutí zpracování vstupní komunikace A.3 Pokud je komunikace vstupní, musí být příjemce komunikace uspán. V pasivním stavu čeká na signál od iniciátora. V závislosti na druhu komunikace (přímá nebo nepřímá) se liší i její zpracování.

- Přímá komunikace je založena na propojení obou účastníku komunikace (iniciátor zajistí vzájemnou výměnu odkazů). Příjemce přebírá kontrolu nad zpracováním v momentě, kdy iniciátor navázal komunikaci a poslal data. Následně příjemce provede aktivitu a její výstup (výstupní data), pokud se ke komunikaci pojí, pošle iniciátorovi. Iniciující proces je posléze probuzen.

- Nepřímá komunikace je v daném návrhu spojena s existencí fronty dat u příjemce. Pokud k dispozicí není, vede to na nahlášení chyby. Při nepřímé komunikaci proces na vstupu postupně zpracovává data z fronty. Pokud ve frontě nic není, proces se uspí. Je probuzen, když data budou zprostředkována iniciátorem komunikace.

Zpracování dat z datové fronty se provádí vyjmutím jedné položky a jejím zařazením do seznamu zpracovávaných dat. Seznam dat by měl odpovídat FIFO frontě (nové položky se přidávají na konec).

Shrnutí zpracování výstupní komunikace A.4 Iniciátor komunikace je zodpovědný za aktivaci procesu umístěného na straně vstupu komunikace. Pokud však komunikace obsahuje fronty, může nastat situace, že proces navazující komunikaci se ve frontě uspí (není nikdo, s kým se dá spojit). Proces je probuzen příchodem nového příjemce (viz A.3).

Navázaní spojení V kontextu daného návrhu navázaní spojení představuje činnost, kterou vykoná iniciátor. Jedná se o uspání zodpovědného procesu na dobu, která je udána při modelování 5.3.

Zajištění aktivace příjemce Tato metoda by měla zajistit probuzení příjemce. Jelikož každý proces v určitou dobu prochází uzly jinou rychlostí, může se stát, že proces iniciující komunikaci dorazí ke komunikaci dříve a signál aktivace se ztratí. Metoda má zajistit, že daná situace nenastane. Využívá se vlastnosti, že události, které procesy produkují, jsou uložené v kalendáři událostí. Tím se procesy stávají naplánovanými. Pokud se proces uspí na neurčito, nebude naplánován.

V rámci navrhované metody se proces iniciátora neustále ptá, zda je příjemce naplánován, pokud získá kladnou odpověď, naplánuje svoji reaktivaci na dobu naplánovaní příjemce. V opačné případě může proces probudit.

Doporučuje se pro jistotu stanovit plánovací prioritu iniciátora nižší než příjemce. Potom bude naplánovaná položka iniciátora komunikace za položkou příjemce. Zajistí to správnou funkčnost navrženého algoritmu.

Zpracování větvení Při zpracování větvení důležitou informací představuje způsob procházení jednotlivými větvemi (viz návrh rozšíření 5.5.2.1). Navrhuje

se zpracování rozdělit na 3 popsané varianty. V závislosti na hodnotě parametru získaného z modelu bude vybrána vhodná metoda.

V případech, kdy není potřeba paralelního provádění, se navrhuje zajistit vykonávaní větví procesem, kterému náleží (bude navržena metoda/sada metod, které větve provedou). Řešení paralelního provádění větví bude představeno v 6.1.4.2.

Ze stavu se po vyhodnocení výstupní podmínky vrátí seznam odkazu na začátky větví, které budou provedeny. Pro provádění větví náhodně nebo prokládaně bude zavolána vhodná metoda, která zajistí jejich provedení.

Provedení větví náhodně Danou metodu lze popsat takto:

1. Náhodně se vybere větev ze získaného seznamu.
2. Dokud se ve větvi nenarazí na vstupní podmínku nebo se větev neukončí (koncový stav bez přechodů nebo navrácený indikátor konce), budou prováděny uzly větve.
3. Po ukončení provedení se větev zařadí do seznamu vyřízených větví.
4. Pokud získaný seznam není prázdný, vrátí se v postupu na krok 1.
5. Po skončení provádění větví se vyhodnotí vstupní podmínka.
   a) Pokud je podmínka splněna, vrátí se uzel s vstupní podmínkou.
   b) Jinak se ohlásí chyba.

Provedení větví prokládaně Metoda má některé společné rysy s prováděním větví náhodně. Postup je následující:

1. Ze seznamu se vybere náhodně větev.
2. Zpracuje se její jeden uzel. Pokud větev nebude ukončena vstupní podmínkou nebo se nenarazí na koncový stav bez možnosti přechodu, zařadí se větev zpět do seznamu.
3. V případě ukončení provedení se větev zařadí do seznamu vyřízených větví.
4. Pokud získaný seznam není prázdný, vrátí se provádění ke kroku 1.
5. Po skončení provádění větví se vyhodnotí vstupní podmínka.
   a) Pokud je podmínka splněna, vrátí se uzel s vstupní podmínkou.
   b) Jinak se ohlásí chyba.

6.1.4.2 Paralelní provádění větví

Návrh paralelního provedení spočívá v přidělení procesu (ve významu použitým v rámci diskrétní simulace) jedné větve, kterou bude provádět než narazí na vstupní podmínku nebo indikátor konce (konečný stav bez možnosti přechodů). Dané procesy budou shromážděny do složeného procesu, který na ně bude dohlížet (využíti třídy `ComplexSimProcess` z Desmo-J).

V rámci návrhu se přidá nový modelovací prvek, který zastoupí kontejner procesů větví. Jedná se o objekt, který podědí od `ComplexSimProcess`. Bude vlastnit předaný seznam větví.

Procesy pro provádění větví se navrhuje podědit od objektu `ORDParticipant`). Z instance participanta, která je zaúkoluje získají všechny potřebné informace (seznam dat a seznam procesů). Začátek jimi vykonávaného procesu odpovídá prvnímu uzlu větve. Jejich životní cyklus vypadá takto:

1. Dokud se nenarazí na vstupní podmínku nebo indikátor ukončení, zpracovávají se jednotlivé uzly.

2. Po ukončení se probudí příslušný „kontejner“ a proces se zařadí do něho.

Instance participanta, která chce zpracovávat větve paralelně, vytvoří kontejner, kterému předá větve, které chce provádět. Do kontejneru se vloží nově vytvořené procesy pro provádění větví a instance participanta se uspí.

Jelikož je třída `ComplexSimProcess` navržena tak, že její vnitřní procesy spí dokud jsou v ní, budou neaktivní i procesy pro větve. Kontejner každému procesu přidělí jeho větev, vypustí probuzené procesy ven ze sebe a uspí se. Ty začnou posléze pracovat dle výše popsaného postupu.

Kontejner je postupně probouzen procesy, které se do něho vrací. Při každém probuzení si zkontroluje, zda už dorazily všechny. Pokud ne, opět se uspí.

Když se všechny procesy vrátí, vyhodnotí kontejner vstupní podmínku a výsledek předá spolu s aktivací příslušné instanci participanta a uspí se.

Pokud se vstupní podmínka vyhodnotila kladně, požádá proces participanta kontejner o stav s vstupní podmínkou. Následně probudí kontejner, který skončí svou činnost. V opačném případě kontejner pouze probudí a sám nahlásí chybu.

6.1.4.3 ORDPermanentParticipant

Objekt `ORDPermanentParticipant` dědí většinu vlastností od rodičovského objektu `ORDParticipant`. Jelikož se objekty v systému nachází celou dobu jeho existence, musí průchod ORD procesem být opakován stále do kola. Z tohoto důvodu „životní cyklus“ permanentního participanta vypadá takto:

1. Stále se opakuje zpracovávání uzlů.

2. Pokud je indikován konec, zpracovávaní začne od začátku ORD procesu.

6.1.4.4 ORDDynamicParticipant

Objekt `ORDDynamicParticipant` systémem jen prochází, proto jeho „životní cyklus“ spočívá v vykonání ORD procesu pouze jednou.

1. Stále se provádí zpracovávání uzlů.
2. Pokud je indikován konec, přestanou se provádět uzly a ukončí se proces.

6.1.4.5 Sklad

V případě skladu, musí být zajištěno, že obsluha bude prováděna paralelně. Z tohoto důvodu bude využito podobné myšlenky jako u paralelního provádění větví 6.1.4.2. Sklad bude představovat potomka `ComplexSimProcess`. K němu se budou vázat pracující vlákna obsluhy, které se navrhuje podědit od objektu `ORDParticipant`.

Pracující vlákna sídli frontu práce, do které se řadí buď procesy nebo data. Sklad rozděluje práci mezi pracující procesy, pokud není co rozdělovat nebo komu přidělovat, sklad se uspí. Je probuzen, když se do něho vrátí proces, který splnil úlohu, nebo bude poskytnutá nová úloha k obsluze.

Pracující procesy dostávají úkol od skladu a začínají ho plnit (prochází ORD procesem popsaným v participantovi). Když úkol splní, obnoví si obsah seznamu dat a seznamu procesů, probudí sklad a vrátí se do něho.

6.1.4.6 Generátor procesů

Generátory jsou v rámci knihovny Desmo-J zastoupeny procesy. Poskytnutá je abstraktní třída `ArrivalProcess`. Její potomek musí popsat druh generovaných instancí a způsob jejich vytvoření.

Navrhuje se vytvořit objekt generátoru, který dědí od dané třídy. Jako volitelný parametr, bude mít počet instancí, které vytvořil. Danou hodnotu, lze potom použít pro podmínku ukončení generování.

Objektu se předají rozdělení s parametrem/parametry a typ objektu, který bude vytvářen. Na základě těchto údajů se začnou vytvářet instance.

6.1.5 Zastavení experimentu

Jak již bylo naznačeno (viz 5.6.2) průchod ORD procesem může uváznout, proto je důležité uváznutí hlásit.

Pokud nastane daná situace, přestanou se generovat položky do kalendáře událostí. V případě knihovny Desmo-J experiment přejde do stavu „zastavený“. Změnu stavu zajistí pouze naplánovaní další položky nebo ukončení experimentu. Pro návrh se nabízí využití dané vlastnosti.

Knihovna Desmo-J poskytuje rozhraní (interface), které dovoluje odposlouchávat změny běhu experimentu (`ExperimentListener`). Návrh spočívá ve vytvoření třídy, která dané rozhraní implementuje.

Třída by obsahovala hodiny, které by sledovaly dobu pasivity simulace. V případě pozastavení experimentu by se hodiny spustily. V moment dosažení zadané maximální hodnoty by došlo k násilnému ukončení experimentu a nahlášení chyby. Pokud by se obnovil běh pokusu, odpočet času by se zastavil a hodiny se vrátily k výchozím hodnotám (nastavily se na 0).

6.1.6 Model

Pro zjednodušení fáze transformace se navrhuje vytvořit i objekt `ORDModel`, který bude mít schopnost vytvoření grafů procesů participantů ze souboru s daty. Model dostane informaci, jaké soubory existuji a k jakému participantovi se pojí.

Zároveň model obdrží znalost o frontách procesů, které zastupují případy, kdy každá instance participanta vlastní svojí frontu. Danou informaci lze například uložit opět do souboru, nebo předat v podobě statického pole navrhovaného objektu modelu. Ta se posléze využije při předání odkazů při aktivaci instancí permanentních participantů.

6.1.6.1 Vytváření grafu procesu

Struktura ukládaní informací je ponechána na implementátorovi. Měla by však mít podobu, ze které lze jednoduše extrahovat informace o tom, jak jsou jednotlivé aktivity, stavy, komunikace a data propojené. Ke každému uzlu v souboru musí být přiřazeny informace o jeho parametrech (rozdělení, parametr rozdělení, ...).

Jelikož je v Desmo-J model zodpovědě za inicializaci generátorů rozdělení a front, budou při vytváření jednotlivých uzlů přidělovány příslušné odkazy na inicializované prvky.

Sestavený graf je modelem posléze předán buď přímo procesům participantů (při aktivaci) nebo generátorům, které odkaz předají procesům, jež vytvoří. Pro všechny instance jednoho participanta bude vytvořen pouze jeden graf, jelikož je struktura v průběhu simulace neměnná.

6.2 Rozšíření ORD modelu v OpenCABE

Následující krok představuje rozšíření modelu ORD v nástroji OpenCABE. Jedná se pouze o začlenění návrhu rozšíření ORD 5 do stávajícího metamodelu. Přidají se následující prvky:

- fronta participantů,
- fronta dat,
- generátor,
- aktivátor,

- experiment,
- sledovaný parametr.

Zároveň musí být dodán přehled rozdělení, která jsou v rámci Desmo-J dostupna. Kromě toho se vytvoří seznam statistických prostředků pro sběr dat (viz 4.2.3).

Navrhuje se rozšířit stávající položky A.2 o navržené parametry 5:

- stav – parametr provádění větvení, čas provádění,
- aktivita – parametr času provádění,
- participant – fronta dat, priority, kapacita,
- data – parametr udávající budoucí podobu, parametr udávající, zda se sleduje, ke komu se data vrací,
- komunikace – údaj o době trvání navázaní spojení,
- podmínky – pravděpodobnost splnění.

6.3 Propojení Desmo-J a OpenCABE

Navrhovaná myšlenka integrace knihovny do OpenCABE spočívá v přidání prvku, který by zajistil převod rozšířeného ORD modelu z OpenCABE do podoby simulačního modelu (navržených prvků v sekci 6.1). Jednoduché znázornění myšlenky je dáno obrázkem 6.1.

Na základě konzultace [1] bylo zjištěno, že nástroj lze jednoduše rozšiřovat s pomocí pluginu. Plugin dle návrhu představuje spojovací prvek mezi stávajícím nástrojem OpenCABE a Desmo-J. V rámci něho by též byli zaregistrovány UI komponenty pro práci se simulaci (spuštění simulace, nastavení adresáře pro generování výstupů z experimentu).

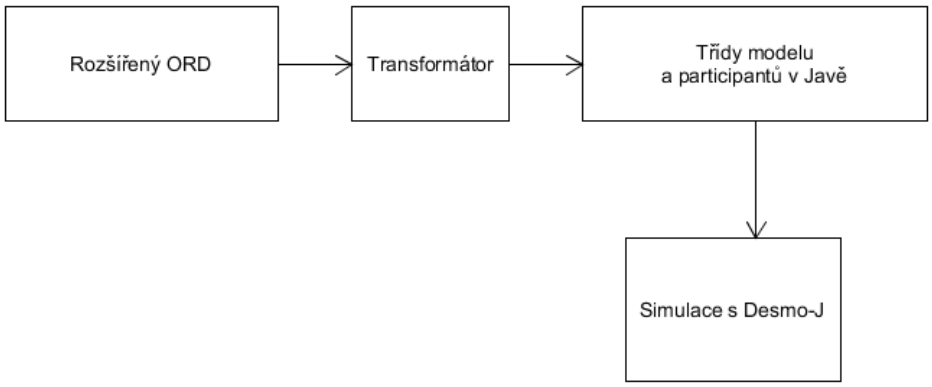
6.3.1 Převod ORD na simulační model

Pro spuštění simulace je zapotřebí model v Javě. Vytváření modelu lze rozdělit na dvě části:

- vytvoření modelů participantů,
- vytvoření zástupce celého modelu.

V momentě, kdy je plugin zaregistrován k OpenCABE, je získán přístup ke všem jeho prvkům. Nejprve se zjistí aktuální okno, ve kterém se nachází ORD určený k simulaci. Získá se ORD příslušnicí danému oknu a následně se provede transformace.

[1] Konzultace s panem Podlouckým 21.4.2015



Obrázek 6.1: Myšlenka propojení OpenCABE a Desmo-J

6.3.1.1 Vytvoření modelů prvků

Vytvoří se popis tříd participantů, které jsou součástí modelů. Jednotlivé vytvářené třídy mohou dědit od jednoho ze třech navržených prvků: permanentní participant, dočasný participant a sklad.

Od skladu bude vytvořená třída dědit v případě, že kapacita participanta bude větší než 1. Dočasný participant se pozná dle toho, že existuje generátor, který na něho odkazuje. Participanti s aktivátory budou potomky permanentních prvků.

Jejich chování je již popsané v rodičovských třídách a předaní parametrů zajistí navržená třída modelu.

6.3.1.2 Zástupce celého modelu

Než se vytvoří grafy jednotlivých procesů participantů, budou postupným procházením ORD diagramu vytvořen soubory s popisem grafů procesů. Zároveň lze již během této fáze sbírat informace o provázaní participantů a front.

Posléze se přejde k vytváření třídy daného modelu, která bude dědit od dříve popsané třídy `ORDModel` 6.1.6.

Z ORD se získají tyto informace:

- všechna použitá rozložení (převedou se na parametry objektu),
- všechny fronty participantů a komunikací (převedou se na parametry objektu),
- počty permanentních participantů (stanou se parametry třídy, na základě kterých se provede vytvoření a aktivace jednotlivých instancí),

- parametry rozdělení (současná znalost rozdělení dovoluje provést jejich inicializaci),
- kapacity front (údaj se použije při inicializaci front).

Na základě toho se do souboru s třídou modelu vygenerují metody inicializace rozdělení a front a metody vytvoření a aktivace generátorů a stálých procesů systému.

6.3.2 Podmínky ukončení experimentu a experiment

Pro podmíněné ukončení experimentu se navrhuje vytvořit objekt, který představuje potomka třídy `ModelCondition` z Desmo-J. Zde se definuje metoda `check()`, do níž se doplní podmínka ukončení. Daný postup se uskuteční v případě, že podmínkou ukončení není pouze maximální doba běhu experimentu.

Následně pokud uživatel nebude požadovat GUI pro průběh experimentu, vytvoří se metoda `main()`, která bude obsahovat vytvoření instance modelu a experimentu a jejich následné propojení. Specifikuji se též výstupní zprávy, které uživatel při modelování určil.

6.3.3 Spuštění simulace

Jak již bylo zmíněno na začátku dané části, prostřednictvím pluginu lze vytvořit vlastní UI prvky a přidělit jim funkčnost.

Navrhuje se vytvořit tlačítko, které zajistí spuštění experimentu. Na základě hodnot daných prvkem Experiment z rozšířeného ORD může spuštění simulace vypadat takto:

- pokud uživatel požaduje průběh simulace bez GUI, zajistí se překlad vytvořených tříd a spuštění experimentu,
- pokud si uživatel vybral GUI pro experiment nebo vizualizaci experimentu, spustí se příslušná aplikace.

Kapitola 7

Závěry a doporučení

Daná kapitola je určena k formulaci závěrů spojených s návrhy představenými v kapitolách 6 a 5. Následně budou poskytnutá doporučení pro následnou implementaci řešení.

7.1 Závěry

Práce měla tři cíle:

1. navrhnout rozšíření ORD, která umožní simulování procesních modelů,
2. poskytnout návrh rozšíření nástroje OpenCABE, které reflektují rozšíření ORD,
3. navrhnout způsob integrace knihovny pro diskrétní simulaci do nástroje OpenCABE (jedná se pouze o návrh na funkční rovině).

Řešení daných úkolů měla vycházet z předchozí rešerše techniky diskrétní simulace a vytvořeného přehledu knihoven pro diskrétní simulaci v Javě.

V rámci kapitoly 5 byly představeny nové prvky a parametry, jejichž cílem je poskytnout další potřebnou informaci, která spolu se znalosti ze základního modelu ORD umožní automatické vytvoření simulačního modelu. Jedná se o parametry reprezentující čas, počty participantů nebo pravděpodobnostní hodnoty. Dále se sem řadí prvky, které zastupují fronty, generátory, experiment nebo aktivátory.

Kapitola 6 pokrývá cíl 2 a 3. V rámci sekce 6.2 jsou shrnuta rozšíření OpenCABE v podobě rozšíření stávajícího metamodelu BORM o parametry a prvky z kapitoly 5.

Pro integraci byla zvolena knihovna Desmo-J. Proces jejího začleňování do OpenCABE je v rámci návrhu rozdělen na dva kroky. První fáze navrhuje rozšíření modelovacích tříd, které knihovna nabízí. Rozšíření tvoří určitou abstrakci, která nabízí objekty zastupující prvky ORD. Objekty mají v sobě popsané potřebné chování a vlastnosti. Z těchto důvodů je ulehčena automatická

tvorba simulačního modelu, jelikož zapotřebí jsou pouze údaje z parametrů rozšířeného metamodelu.

Druhou fázi tvoří návrh pluginu, který zajisti vygenerování simulačního modelu pro procesní diagram umístěný v aktivním okně nástroje. Zároveň navržený plugin má za úkol přidání grafických prvků pro spuštění příslušného experimentu.

Vypracovaný návrh poskytuje popis, který lze převést do podoby implementačního modelu a posléze i realizovat. Snahou bylo zohlednit chování entit v reálném světě, proto návrh obsahuje prvek fronty pro data, který se běžně v diskrétních systémech nevyskytuje. Zároveň je v rámci převodu procesního modelu ORD na simulační zachována struktura procesů participantů. Tím nedochází k úbytkům v informacích o entitách.

7.2 Doporučení

Daná část se věnuje doporučením určeným pro implementaci navrženého řešení. Prvním základním doporučením, je zajistit kontrolu uživatelem poskytnutého modelu. Jedná se například o ověření, zda umístěné fronty na komunikacích spadají pod možné případy popsané v 5.4.1.1. Následný návrh s touto kontrolou částečně počítá.

Doporučuje se vhodně volit strukturu souborů, které se vytvoří při generování simulačního modelu. 6.3.1.2. Proces iniciace prvků by měl proběhnout relativně rychle, proto struktura musí umožňovat poměrně jednoduché získávaní informací bez nutnosti zdlouhavého čtení ze souborů.

Vhodné je též nastavování výchozích hodnot u parametrů objektu. Dané doporučení platí jak pro navržené modelovací prvky v ORD, tak i pro objekty simulačního modelu.

Zároveň pro navržené objekty v rámci Desmo-J by mělo být poskytnuto více konstruktorů. Jejich použití by se řídilo znalostmi získanými z rozšířeného ORD, např. pro komunikaci s 2 frontami bude použít jiný konstruktor než pro komunikaci s jednou frontou.

Při implementaci grafického prvku pro spuštění implementace je vhodnější jako výchozí hodnotu nastavit spuštění GUI pro simulaci. Jedná se o určitý kompromis mezi spuštěním vizualizace, která je výpočetně náročná, a spuštěním bez GUI, které je pro uživatele méně přehledné.

U instancí, které opakují průchod grafem procesu participanta (např. stále prvky systému), musí pokaždé docházet k ověření platnosti udržovaných seznamů. Podstatou je pokaždé projít seznam odkazů na další instance participantů a odstranit zneplatněné záznamy (např. pokud proces opustil systém).

Závěr

Cílem dané práce bylo navrhnout rozšíření procesních modelů ORD, která umožní provádění jejich diskrétní simulace. Zároveň měl být představen návrh rozšíření nástroje OpenCABE reflektující rozšíření ORD. Poslední cíl spočíval ve vytvoření návrhu, který popisuje integraci vybrané knihovny pro diskrétní simulaci napsané v programovacím jazyce Java do nástroje OpenCABE.

Na základě rešeršní části práce I byl vytvořen návrh rozšíření procesních modelů ORD. Myšlenka zakládá na přidávaní parametrů stávajícím prvkům a začlenění nových modelovacích elementů. Návrh vychází převážně ze znalosti o systému hromadné obsluhy 2.6.1, který lze aplikovat na mnohé situace. Mezi rozšíření patří například zohlednění času v podobě parametrů aktivit, stavů a komunikací, přidaní pravděpodobnosti uskutečnění přechodů. Dále do modelu byly začleněny fronty pro participanty a data.

Následně se navrhla integrace knihovny Desmo-J do nástroje OpenCABE. Výběr knihovny se řídil vytvořeným přehledem 4.1 a vyhodnocením knihoven pro diskrétní simulaci napsaných v jazyce Java. V rámci návrhu integrace byly vyčleněny tři kroky:

1. Navrhla se sada objektů reprezentujících prvky ORD, která rozšiřuje modelovací třídy knihovny Desmo-J. V rámci návrhu se kromě samotných objektů popsalo i jejich chování a vlastnosti.

2. Následně se provedl návrh rozšíření nástroje OpenCABE, jež reflektuje navržená rozšíření procesních modelů ORD. Jednalo se o přidaní nových parametrů a elementů do metamodelu BORM, který je v OpenCABE implementován.

3. Na závěr byl navržen plugin, který zajistí propojení mezi Desmo-J a OpenCABE. V rámci pluginu se nachází transformátor, který zprostředkovává převod rozšířeného modelu ORD do simulační podoby v Javě. Zároveň navržený plugin je zodpovědný za registraci grafických prvků pro spuštění simulace.

Spolu s doporučeními 7.2 lze dané návrhy použít jako podklady pro následnou implementaci. Objekty navržené pro Desmo-J nemusí být používány výhradně ve spojení s OpenCABE. S jejich pomocí lze tvořit vlastní simulační modely, které nemusí odpovídat metodice BORM.

Velmi přínosná je též i rešeršní část, která poskytuje dobrý přehled jak o základech simulace jako celku, tak i o samotné technice diskrétní simulace. Kromě toho poskytuje rozbor knihoven pro diskrétní simulaci v Javě, který může být použít jako podklad pro jiné práce.

Daný návrh lze rozšířit o popis objektů a postupů, které zajistí simulovaní modelů s doprovodem vizualizace. Za zvážení stojí úpravy, kdy se například místo objektů reprezentujících komunikace přidají komunikační kanály, kterými budou posílány zprávy. Další směr, kterým lze též ve vývoji návrhu pokračovat, představují vnořené modely.

Literatura

[1] PERINGER, P.: Modelování a simulace [online], prosinec 2012, [cit. 2015-24-04]. Dostupné z: `http://www.mechacek.com/Media/Default/Page/opora-ims.pdf`

[2] CHOI, B. K.; KANG, D.: *Modeling and Simulation of Discrete Event Systems*. John Wiley & Sons, 2013.

[3] DLOUHÝ, M.; FÁBRY, J.; KUNCOVÁ, M.; aj.: *Simulace podnikových procesů*. Brno: Computer Press, druhé vydání, 2011, ISBN 978-80-251-3449-8.

[4] POOCH, U. W.; WALL, J. A.: *Discrete Event Simulation: A Practical Approach*. Computer Science & Engineering, Taylor & Francis, 1992, ISBN 9780849371745.

[5] KŘIVÝ, I.; KINDLER, E.: *Simulace a modelování*. Učební texty, Ostravská univerzita, Přírodovědecká fakulta, 2001, ISBN 9788070428092.

[6] GLOMBÍKOVÁ, V.: Počítačová simulace podniých procesů [online], 2014, [cit. 2015-24-04]. Dostupné z: `http://www.kod.tul.cz/predmety/PSI/Prednasky/prednasky_2011/prednaska_2014_2.pdf`

[7] CASSANDRAS, C. G.; LAFORTUNE, S.: *Introduction to Discrete Event Systems*. SpringerLink Engineering, Springer, 2008, ISBN 9780387333328.

[8] BABULAK, E.; WANG, M.: Discrete Event Simulation: State of the Art. In *Discrete Event Simulation*, editace A. Goti, InTech, 2010, s. 1–20.

[9] KHEIR, N.: *Systems Modeling and Computer Simulation*. Electrical and Computer Engineering, Taylor & Francis, druhé vydání, 1995, ISBN 9780824794217.

[10] WAINER, G. A.: *Discrete-Event Modeling and Simulation*. CRC Press, 2009, ISBN 9781420053364.

[11] CHUNG, C. A.: *Simulation modeling handbook: A practical approach*. CRC Press LLC, 2004, ISBN 0849312418.

[12] FISHMAN, G.: *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer Series in Operations Research and Financial Engineering, Springer New York, 2013, ISBN 9781475735529.

[13] VANGHELUWE, H.: Discrete Event Modelling and Simulation [online], [cit. 2015-30-04]. Dostupné z: `http://www.cs.mcgill.ca/~hv/classes/MS/discreteEvent.pdf`

[14] KNOTT, R.; MERUNKA, V.; POLAK, J.: The BORM methodology: a third-generation fully object-oriented methodology. *Knowledge-Based Systems*, ročník 16, č. 2, 2003: s. 77–89.

[15] MERUNKA, V.: *Objektové modelování*. Praha:Alfa Nakladatelství, první vydání, 2008, ISBN 9788087197042.

[16] VEJRAŽKOVÁ, Z.: *Business Process Modeling and Simulation: DEMO, BORM and BPMN*. Diplomová práce, České vysoké učení technické v Praze, 2013.

[17] PODLOUCKÝ, M.; PERGL, R.: Towards Formal Foundations for BORM ORD Validation and Simulation. In *ICEIS 2014 - Proceedings of the 16th International Conference on Enterprise Information Systems, Volume 2, Lisbon, Portugal, 27-30 April, 2014*, editace S. Hammoudi; L. A. Maciaszek; J. Cordeiro, SciTePress, 2014, ISBN 978-989-758-028-4, s. 315–322, doi:10.5220/0004897603150322. Dostupné z: `http://dx.doi.org/10.5220/0004897603150322`

[18] KAČER, J.: *J-Sim Tutorial [online]*. [cit. 2015-04-30]. Dostupné z: `http://www.kacer.biz/j-sim/documentation/tutorial/`

[19] The Institute for Computing Systems Architecture: *The SinJava Tutorial [online]*. [cit. 2015-05-01]. Dostupné z: `http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/guide/tutorial.html`

[20] Department of Computer Science, niversity of Hamburg: *The DESMO-J Tutorial [online]*. [cit. 2015-05-01]. Dostupné z: `http://desmoj.sourceforge.net/tutorial/overview/0.html`

[21] LITTLE, M.: *JavaSim, GitHub [online]*. [cit. 2015-05-01]. Dostupné z: `https://github.com/nmcl/JavaSim/blob/master/docs/javasim.pdf`

[22] L'ECUYER, P.: *SSJ User's Guide [online]*. Canada Research Chair in Stochastic Simulation and Optimization,Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Canada, 6 2014,

[cit. 2015-05-01]. Dostupné z: http://www.iro.umontreal.ca/~simardr/ssj/examples/examples.pdf

[23] Faculty of Technology, Policy and Management of Delft University of Technology in Delft: *DSOL [online]*. [cit. 2015-05-01]. Dostupné z: http://simulation.tudelft.nl/simulation/index.php/dsol

[24] ROSSETTI, M. D.: *Java Simulation Library (JSL) [online]*. University of Arkansas, [cit. 2015-04-30]. Dostupné z: http://cavern.uark.edu/~rossetti/research/research_interests/simulation/java_simulation_library_jsl/

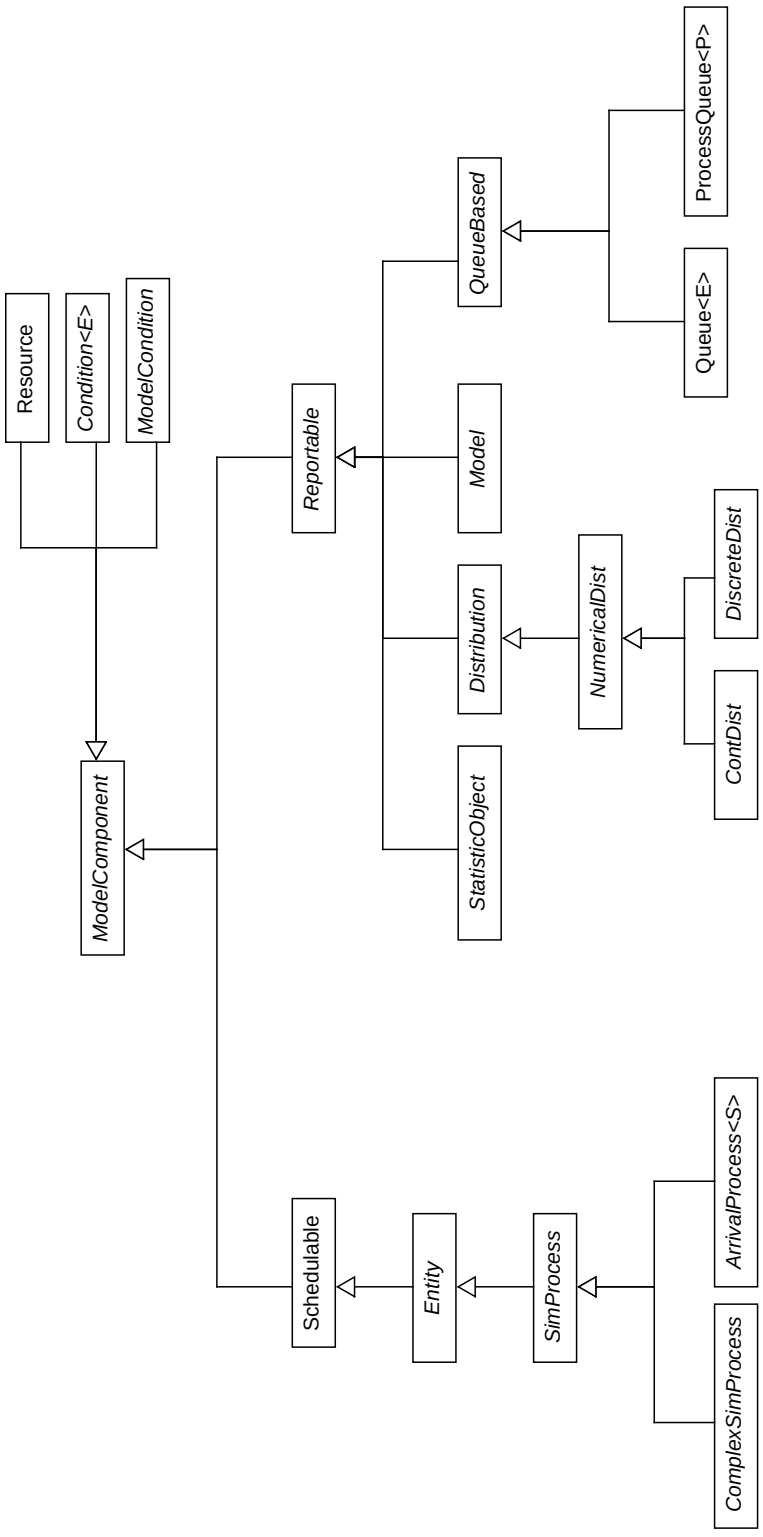
[25] GÖBEL, J.; JOSCHKO, P.; KOORS, A.; aj.: The Discrete Event Simulation Framework DESMO-J: Review, Comparison To Other Frameworks And Latest Development. In *Proceedings of the 27th European Conference on Modelling and Simulation, ECMS 2013, Ålesund, Norway, May 27-30, 2013*, editace W. Rekdalsbakken; R. T. Bye; H. Zhang, European Council for Modeling and Simulation, 2013, ISBN 978-0-9564944-6-7, s. 100–109, doi:10.7148/2013-0100. Dostupné z: http://dx.doi.org/10.7148/2013-0100

[26] Oracle: *Communicating Business Processes Using Correlations [online]*. [cit. 2015-05-02]. Dostupné z: http://docs.oracle.com/cd/E25178_01/doc.1111/e15176/correlations_bpmpd.htm

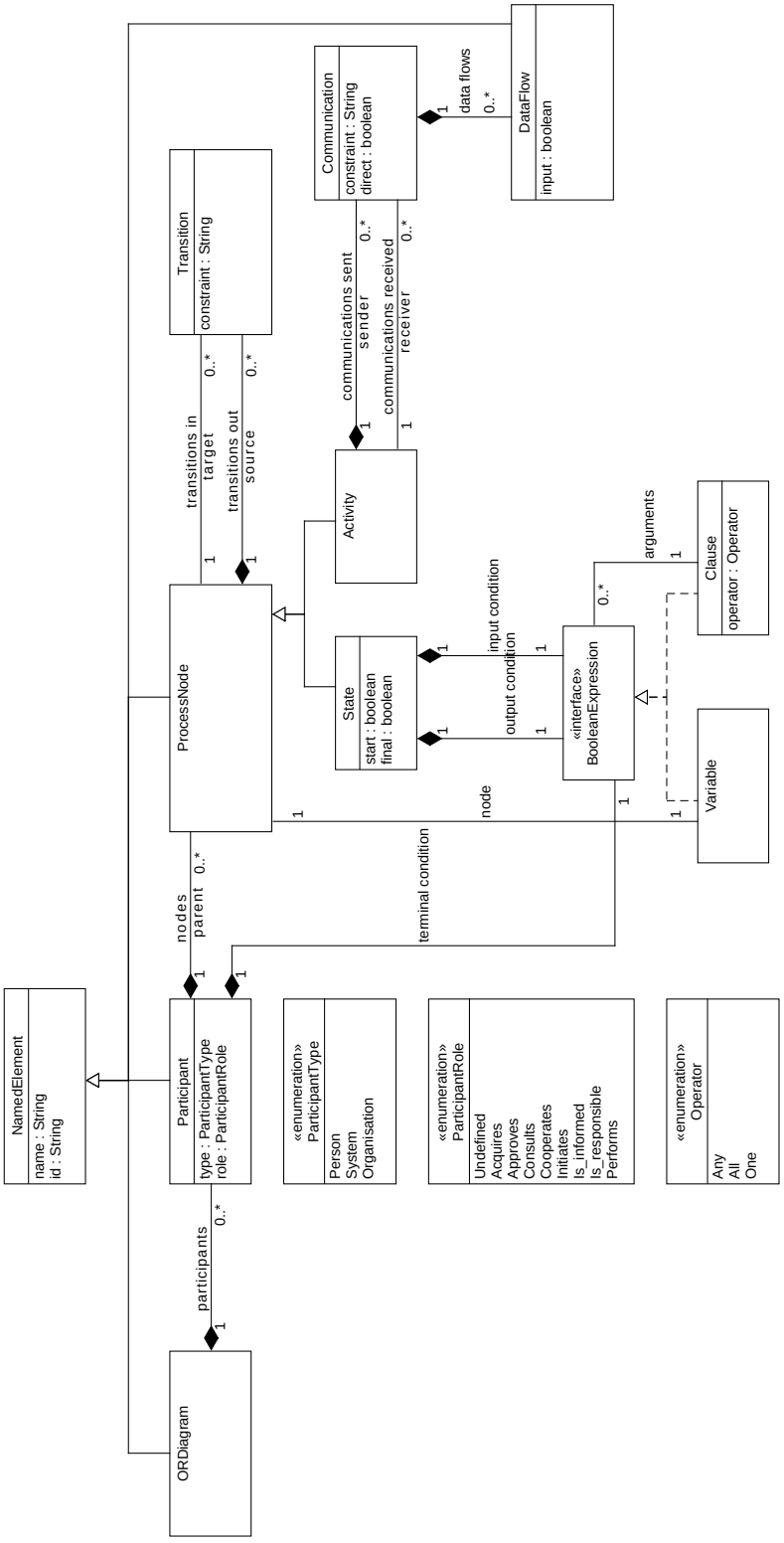
Příloha A

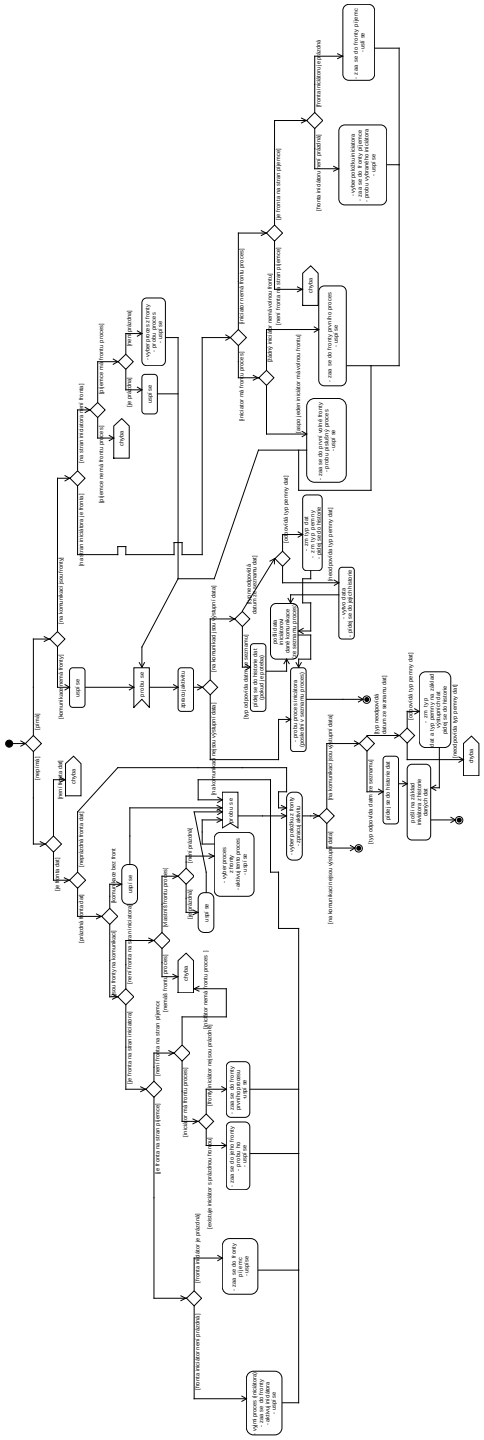
Obrázkové přílohy

Obrázek A.1: Zjednodušená hierarchie modelovacích prvků Desmo-J

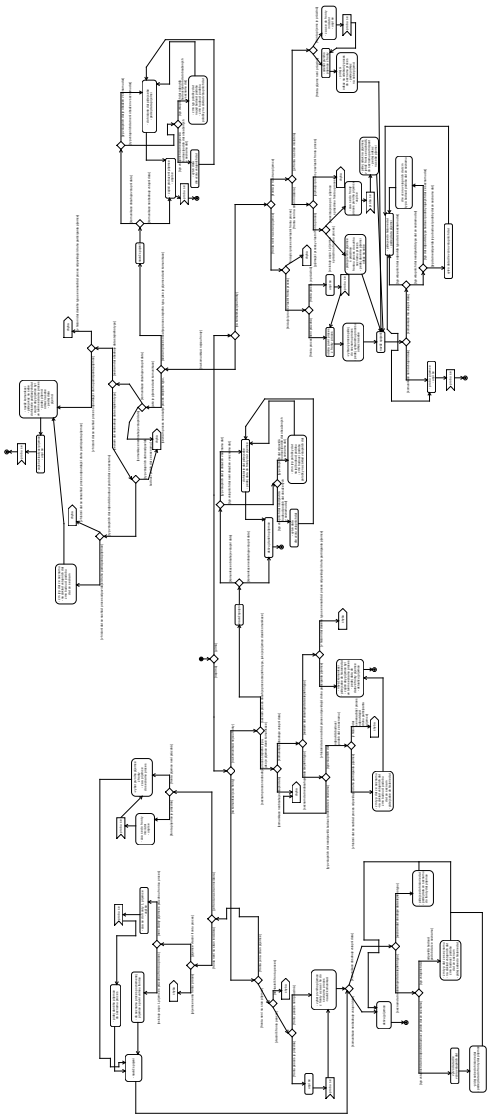


Obrázek A.2: Metamodel BORM ORD

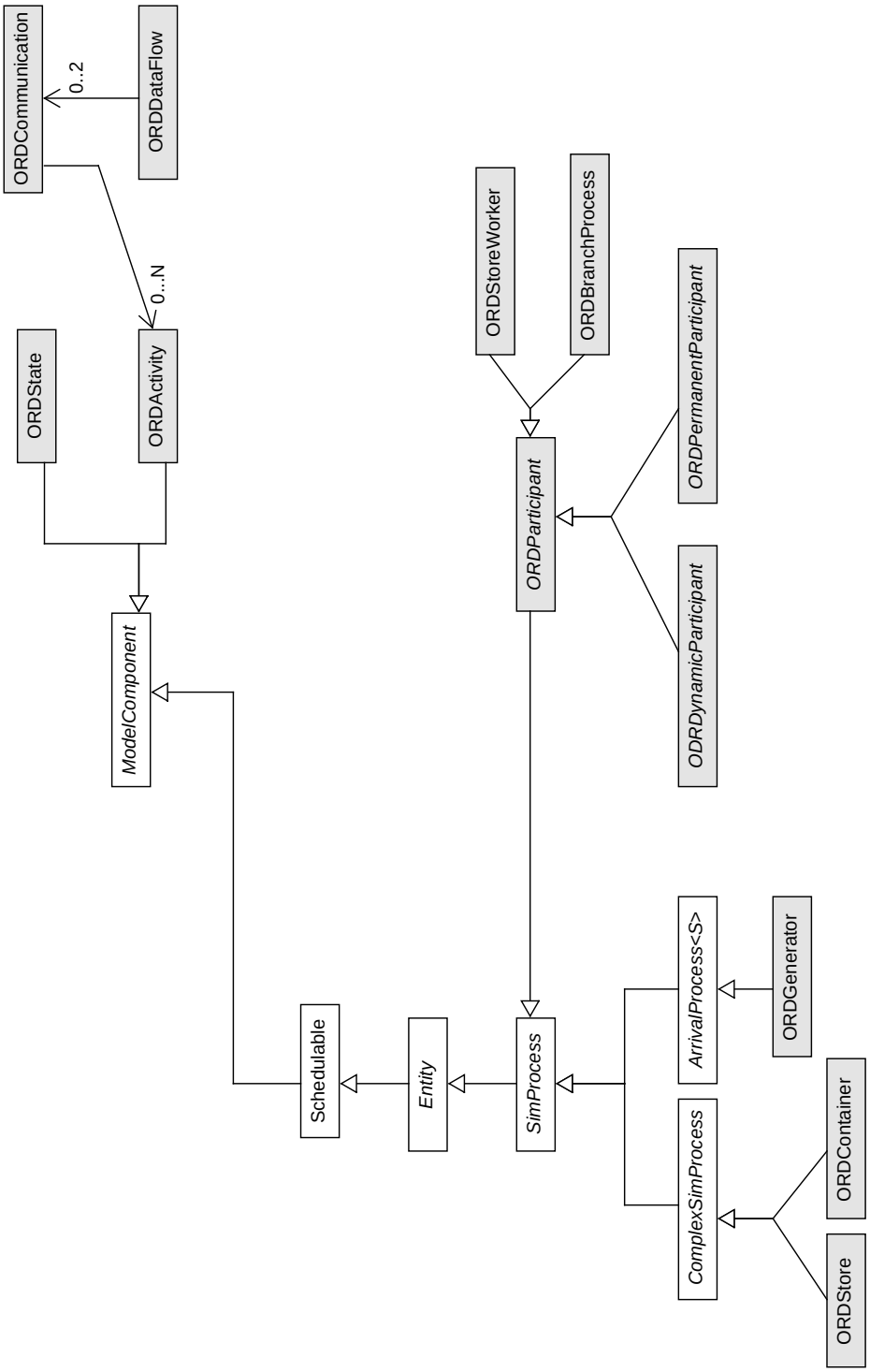




Obrázek A.3: Návrh postupu zpracování vstupní komunikace



Obrázek A.4: Návrh postupu zpracování výstupní komunikace



Obrázek A.5: Návrh rozšíření modelovacích prvků

Příloha B

Přílohy v podobě tabulek

Tabulka B.1: Přehled dostupných statistických rozdělení

BoolDistBernoulli	vrací true s pravděpodobnosti p
BoolDistConstant	konstantně vrací zadanou pravděpodobnostní hodnotu
ContDistEmpirical	generuje hodnoty z empirického rozdělení
ContDistErlang	poskytuje hodnoty z Erlangova rozdělení
ContDistExponential	odpovídá exponenciálnímu rozdělení
ContDistGamma	vrací hodnoty z gamma rozdělení
ContDistNormal	generuje hodnoty z normálního rozdělení
ContDistTriangular	slouží k získání hodnot z trojúhelníkového rozdělení
ContDistUniform	hodnoty odpovídají rovnoměrnému rozdělení
ContDistWeibull	vrací hodnoty z Weibullova rozdělení
DiscreteDistBinomial	slouží k získáni hodnot z Binomického rozdělení
DiscreteDistConstant	vrací stále stejnou hodnotu (testovací účely)
DiscreteDistEmpirical	odpovídá diskrétní podobě empirického rozdělení
DiscreteDistPoisson	generuje hodnoty z Poissonova rozdělení
DiscreteDistGeo	vrací hodnoty z geometrického rozdělení
DiscreteDistUniform	diskrétní varianta rovnoměrného rozdělení
EntityDistEmpirical	empirické rozdělení pro generování entit (pravděpodobnosti se přiřazuji každé entitě zvlášť)
EntityDistUniform	entity jsou vraceny se stejnou pravděpodobnosti

Příloha C

Seznam použitých zkratek

BORM Business Object Relation Modeling

BPMN Business Process Model and Notation

CCM Centre for Conceptual Modelling

DESS Differential Equation System Specification

DEVS Discrete-Event Systems Specification

DTSS Discrete Time System Specification

GUI Graphical user interface

HTML Hyper Text Markup Language

OBA Object Behaviour Analysis

ORD Object Relation Diagram

SHO Systém hromadné obsluhy

UML Unified Modeling Language

UI User interface

XML Extensible markup language

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `diagrams` ..... složka s rozsáhlými diagramy
  - `thesis.pdf` ..... text práce ve formátu PDF