VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technologií

Ústav inteligentních systémů

Ing. Radek Kočí

METODY A NÁSTROJE PRO IMPLEMENTACI OTEVŘENÝCH SIMULAČNÍCH SYSTÉMŮ

Methods and Tools for Implementation
of Open Simulation Systems

ZKRÁCENÁ VERZE PH.D. THESIS

Obor: Informační technologie
Školitel: Doc. Ing. Zdena Rábová, CSc.
Oponenti: Prof. Ing. Jiří Šafařík, CSc.
Prof. Ing. Ivo Vondrák, CSc.

Datum obhajoby: 29. října 2004

KLÍČOVÁ SLOVA
otevřené implementace, reflektivita, Objektově orientované Petriho sítě, modelování, simulace, prototypování, simulační systém

KEYWORDS
open implementation, reflection, Object Oriented Petri Nets, modelling, simulation, prototyping, simulation system

Práce je uložena na vědeckém oddělení FIT VUT v Brně

ISBN 80-214-2798-1
ISSN 1213-4198

Obsah

1 Současný stav řešené problematiky

1.1 Úvod

Moderní simulační systémy jsou stále komplikovanější a komplexnější. S tímto trendem roste i potřeba prostředků pro snadnou, příp. automatickou konfiguraci a adaptaci schopností systémů pro různé aplikační podmínky. Dalším trendem v modelování je využívání multiparadigmatu. Modely jsou většinou vytvářeny s použitím jednoho typu paradigmatu (např. Petriho sítě [26, 36, 30, 40, 13], SIMLIB [33], Simula [34] apod.) Občas je však vhodné určité části systému modelovat pomocí různých paradigmat, která jsou pro vyjádření dílčího problému vhodnější.

Jako použitelné řešení se jeví koncept tzv. otevřených implementací, které zpřístupňují některé aspekty interní implementace systémů a tím umožňují jejich kontrolu. Základním prostředkem pro vyjádření otevřené implementace je zajištění obecného implementačního rámce, který usnadňuje práci uživatelů, ať už v podobě navrhování či používání otevřených implementací. Jedním z takových rámců je metaúrovňová architektura a s ní spojený metaobjektový protokol (MOP). MOP zajišťuje řešení založené na objektové orientaci, které může být jednoduše integrováno do klasických vývojových procesů. MOP je velmi široce uplatňován v moderních operačních systémech a jazycích, kde nabízí elegantní a uniformní způsob programování založený na reflektivních konceptech.

Modelování obsahuje obecně dvě úrovně: modelování statické struktury systémů a modelování jejich chování. Simulaci, tj. experimentování s modelem, pak chápeme jako interpretaci chování modelu na základě popisu chování jeho struktur. Pro popis dynamických aspektů modelu lze použít různá paradigmata a formalismy. Tato práce vychází z Objektově orientovaných Petriho sítí (OOPN), které byly vyvinuty na Fakultě informačních technologií Vysokého učení technického v Brně. OOPN s výhodou kombinují přednosti Petriho sítí (formální základ, přirozený popis paralelismu) a objektové orientace (strukturalizace modelu, přirozená tvorba instancí podstruktur).

Předkládaná práce vychází z myšlenky otevřené implementace simulačního systému konvergujícího do zjednodušené formy (ve smyslu bezpečnosti) operačního systému, který slouží pro modelování, simulaci a prototypování složitých problémů. Výhody otevřené implementace se projeví především ve velké flexibilitě systému umožňující inkrementální vývoj modelů, jednoduchou aplikovatelnost speciálních simulačních technik či možnost adaptace prostředí vzhledem k měnícím se požadavkům, a to včetně možnosti měnit či kombinovat různé vyjadřovací prostředky (paradigmata), jako např. OOPN, Smalltalk apod. Důležitým rysem uvedené myšlenky je i možnost začlenit vyvíjený model do reálného prostředí. Takto pojatý model pak může sloužit jako část prototypu či přímo aplikace. Smyslem celého projektu PNtalk je ověřit diskutované vlastnosti otevřených systémů v praxi a jejich

uplatnění v oblasti modelování a prototypování agentů, umělé inteligence či návrhu komplexních systémů.

1.2 Objektově orientované přístupy k Petriho sítím

Petriho sítě byly vyvinuty v šedesátých a sedmdesátých letech 20. století a brzy se ukázaly být vhodným jazykem pro popis a analýzu synchronizace, komunikace a sdílení zdrojů mezi paralelními procesy. Přesto se při praktickém použití naráželo na dvě zásadní nevýhody. Vytvářený model byl často nepřiměřeně velký, protože veškerá manipulace s daty musela být reprezentována přímo v síťové struktuře. Petriho sítě také neobsahovaly koncept hierarchie, takže bylo nemožné vytvářet skutečně rozsáhlé modely jako množinu separátních submodelů. Tyto problémy byly odstraněny při vývoji vysokoúrovňových Petriho sítí a hierarchických Petriho sítí v osmdesátých letech.

V devadesátých letech 20. století se začaly dostávat do popředí zájmu objektově orientované varianty Petriho sítí. Od poloviny 90. let se na Fakultě informačních technologií Vysokého učení technického v Brně vyvíjí systém PNtalk a s ním spojené paradigma Objektově orientovaných Petriho sítí (OOPN). Základní formalismus paradigmatu OOPN je tvořen vysokoúrovňovou variantou Petriho sítí, které umožňují přirozeným způsobem popsat tok řízení modelu a paralelismus. OOPN jsou založeny na stejném chápání objektu jako objektově orientovaný jazyk Smalltalk [8] s jedním podstatným rozdílem – procedura (tj. metoda objektu) není popsána jako sekvence příkazů, ale prostřednictvím vysokoúrovňových Petriho sítí. Metodu pak chápeme jako vzor struktury. Při vyvolání metody dojde k vytvoření instance příslušného vzoru (sítě), tzn. že se vytvoří jeho kopie, nad kterou pak probíhá evoluce metody.

Nejznámější projekty a nástroje, vyvíjené na jiných pracovištích a univerzitách, jsou např. *Renew* [35], *Petri Net Kernel* [18], *Objektové Petriho sítě* [26, 25] a *Kooperativní sítě* [36]. Uvedené ukázky různých přístupů k objektově orientovaným Petriho sítím nebo objektovým prostředím pro vývoj různých variant Petriho sítí se svou podstatou blíží variantě Objektově orientovaných Petriho sítí (OOPN). Zásadní rozdíl mezi OOPN a jinými formalismy lze spatřovat v následujících bodech:

- Větší provázanost Petriho sítí (OOPN) a inskripčního jazyka, která umožňuje tvorbu prototypových aplikací s řízením na základě Petriho sítí, zjednodušení tvorby workflow aplikací apod.

- Vysoký stupeň dynamičnosti OOPN, který umožňuje vývoj sítí za běhu modelu (simulace, běh prototypu) samotným modelem nebo vnějšími zásahy.

1.3 Modelování a prototypování

Specifické oblasti využití OOPN jsou v modelování aplikací a prototypování. Model lze obecně chápat jako abstrakci uvažovaného systému. Smyslem modelu je ověřit požadované vlastnosti modelovaného systému či porozumět uvažované problémové doméně. Pro ověřování vlastností existuje více přístupů, od formálních verifikací a validací až po simulace. Pokud chceme porozumět řešené problémové doméně (tj. nechceme provádět verifikaci vlastností), můžeme použít některý z formalismů pro modelování systémů jako např. UML [39] nebo přímo manipulovat s modelem systému, který není tak abstraktní jako simulační model, ale zároveň není plnohodnotnou implementací problému – tedy prototypovat.

Obecně se dá říci, že při použití grafických modelů (diagramů) vzniká problém porozumění pramenící právě z jejich grafické podstaty. Tento problém by se dal vyjádřit následovně: Krásná vlastnost techniky grafického popisu je, že jí rozumí každý. Špatná vlastnost je, že jí rozumí každý jinak. Tento problém u Petriho sítí (včetně OOPN) není. Grafické vyjádření má svou podstatu ve formální definici. OOPN lze použít pro modelování statických i dynamických aspektů systémů a díky svému formálnímu základu je možné provádět experimenty (simulace) *přímo s těmito modely*, příp. verifikace vybraných částí modelu. Bližší diskuzi na dané téma a případovou studii OOPN modelování lze nalézt v [20, 23].

Prototypování [24] můžeme chápat jakou součást životního cyklu vývoje aplikace. Prototypování bylo poprvé představeno a použito v osmdesátých letech, tedy ve stejné době, kdy se objektově orientované přístupy staly populární. Základní myšlenkou prototypování je vývoj funkčního modelu aplikace, aniž jsou předem známy veškeré požadované aspekty a vlastnosti modelovaného systému. Přesto je někdy výhodné mít k dispozici zjednodušenou verzi modelu, která umožňuje provádět experimenty, testovat a validovat různá možná řešení daného problému a lépe rozhodovat, co je skutečně zapotřebí a jak toho přesně dosáhnout.

Hranice mezi některými modely UML, simulačním modelem a prototypováním nemusí být zcela jednoznačné. Prezentovaná práce se věnuje modelování, simulaci a prototypování a především systému pro práci s dynamickými simulačními modely a prototypy. Jednou z klíčových myšlenek je spojit nebo alespoň maximálně zjednodušit vazbu mezi modelem a prototypem, tj. spojit výhody rychlé tvorby prototypů, jaké nabízí systém Smalltalk, a paradigmatu založeného na formálním popisu modelu.

1.4 Související práce

Problematikou modelování, objektově orientovaných Petriho sítí a reflektivních architektur se zabývá mnoho projektů a prací ve světě. Následující výčet lze chápat jako inspirativní zdroje, které jsem využil ve své práci. V oblasti modelování můžeme najít různé varianty metod a přístupů, např. z aktivit na Fakultě informač-

ních technologií Vysokého učení technického v Brně to jsou techniky a nástroje pro tvorbu multimodelů [11, 12], simulační jazyk HELEF [10] nebo simulační knihovnu SIMLIB [33] pro jazyk C++. V oblasti objektově orientovaných Petriho sítí můžeme najít shrnutí různých přístupů, variant a použití v [6, 42]. Myšlenka spojení objektové orientace a Petriho sítí vznikla v první polovině devadesátých let. Jednotlivé přístupy k vytvoření různých variant objektových Petriho sítí si můžeme uvést podle jejich představitelů: dr. Charles A. Lakos [26], Christophe Sibertin-Blanc [36], dr. Daniel Moldt [30] nebo profesor Rüdiger Valk [40]. Objektově orientované Petriho sítě společně se systémem PNtalk byly poprvé představeny v polovině devadesátých let 20. století [4, 13, 14].

Myšlenka objektově orientované výpočetní reflektivity[1] (*computational reflection*), včetně strukturálních změn a změn chování během evoluce, byla představena v sedmdesátých letech [8], ale kořeny těchto konceptů můžeme vysledovat mnohem dříve (jazyk Lisp, Univerzální Turingův stroj apod.) V současné době se problematikou reflektivních systémů zabývají profesor Gregor Kiczales [17] (aspektově orientované programování) a dr. Pattie Maes [28] (umělá inteligence), projekty Actalk [3] (souběžná práce ve Smalltalku prostřednictvím reflektivity, představení aktorů), Apertos [41] a TUNES [37] (vývoj vysoce flexibilních operačních systémů). V oblasti modelování a simulace můžeme zmínit přístupy k zavedení reflektivity (umožňující strukturální změny modelů) do systému DEVS [1], a to např. práce Fernanda J. Barrose [2] nebo profesora Uhrmachera [38]. Ve světě Petriho sítí můžeme nalézt aktivity související s kombinací objektové orientace, Petriho sítí a reflektivních vlastností např. u dr. Lakose [27], který zdůrazňuje výhody takového přístupu v čisté, flexibilní a efektivní implementaci a také v možnosti představit alternativní schémata strategií plánování apod.

2 Cíle disertační práce

Disertační práce vychází z konceptu systému PNtalk pro modelování a simulaci prostřednictvím paradigmatu Objektově orientovaných Petriho sítí (OOPN). Jejím cílem je rozšířit výzkum OOPN a vytvořit nástroj umožňující začlenit vyvíjený model do reálného prostředí, tj. nástroj pro prototypování. Práce se věnuje přístupům k vyjádření dynamických aspektů modelování, simulace a prototypování. Jako základní přístup je diskutován princip otevřených implementací, který je aplikován na architekturu otevřeného simulačního systému. Otevřený simulační systém umožňuje uvažovat dynamické změny modelů a dynamické manipulace s modely, včetně spojení modelů popsaných OOPN s jinými paradigmaty. Cíle disertační práce můžeme shrnout do tří bodů:

[1] Někdy bývá ve spojitosti s reflektivními systémy použit pojem *reflexe*. Podle mého mínění je český termín *reflektivita* korektní ekvivalent anglického termínu *reflection*. Reflexe pak vyjadřuje jednu z vlastností reflektivních systémů, tj. schopnost odrážet (reflektovat) vybrané aspekty objektů (či jiných entit).

1. *Návrh nové koncepce otevřené architektury simulačního systému*. Architektura vychází z principů otevřených implementací a objektové orientace. Koncepce návrhu simulačního systému má umožnit spojovat různá paradigmata, provádět dynamické manipulace s modely i simulačním systémem a provádět různé druhy simulačních experimentů.

2. *Zobecnění přístupu k reprezentaci doménových paradigmat*. Otevřený simulační systém má umožnit kombinaci různých paradigmat na úrovni objektů i metod a jejich přizpůsobení měnícím se požadavkům modelování. Přístup k popisu doménových paradigmat vychází z vlastností popisu a evoluce OOPN.

3. *Návrh metodiky a zhodnocení aplikovatelnosti navrženého otevřeného simulačního systému*. Výhody otevřeného simulačního systému se projeví především v jeho velké flexibilitě umožňující adaptovat systém podle požadovaných potřeb, kombinovat různé vyjadřovací prostředky pro specifikaci modelů a vytvářet prototypy reálných aplikací.

3 Přístup k řešení

Teoretické úvahy a praktické aspekty implementace otevřeného simulačního systému se opírají o koncepci otevřených implementací, zejména metaúrovňových architektur s reflektivními vlastnostmi. Problematika dynamických modelů spočívá v možnosti změn, tj. změn chování či struktury modelu za jeho běhu. Vhodným prostředím pro realizaci těchto požadavků jsou dynamické systémy založené na objektově orientovaných principech, jako jsou např. Self nebo Smalltalk. Základní princip těchto systémů spočívá v rozdělení modelovaného problému na nezávislé jednotky (objekty), které spolu komunikují zasíláním zpráv. Množina zpráv (či operací), které objekt nabízí, se označuje jako rozhraní. Rozhraní bývá někdy označováno jako protokol (termín protokol zavedl jazyk a systém Smalltalk [8]).

Základní ideou otevřené implementace je zpřístupnění vnitřních struktur objektů, abychom mohli systém modifikovat a rozšiřovat. Klasickým případem otevřené implementace jsou metaúrovňové architektury. Můžeme si je představit tak, že ke každému klasickému objektu existuje nějaký jiný (meta) objekt, který poskytuje protokol pro inspekci a změnu vybraných aspektů klasického objektu.[2] Základními úrovněmi jsou *doménová úroveň* (*domain level*)[3] a *metaúroveň* (*meta level*) [28]. Doménová úroveň představuje prostor, který popisuje řešený problém v daném paradigmatu (příp. paradigmatech). Na této úrovni probíhá evoluce *doménových*

[2] Takto pojatému klasickému objektu budeme nadále říkat *doménový objekt*. S tímto objektem je spojeno *doménové paradigma*, které je použito pro jeho definici (např. Smalltalk nebo OOPN).

[3] V literatuře bývá tato úroveň označována také jako základní (*base level*).

objektů. Metaúroveň vytváří prostor pro evoluci systémových (tj. řídicích, implementačních) metaobjektů. S jednotlivými úrovněmi jsou svázány protokoly objektů – doménový protokol a metaobjektový protokol (MOP) [9, 43, 16]. Obecně pak v metaúrovňových architekturách existují různé úrovně, kde vždy jedna úroveň je řízena jinou úrovní. Principy otevřených implementací umožňují nejen ovlivňovat vlastní struktury objektů, ale i jejich výpočetní chování, tzn. způsob, jak reagují na zprávy, jaké další operace se vykonají v důsledku zaslání či přijetí zprávy apod.

S otevřenými implementacemi a metaúrovňovými architekturami úzce souvisejí jejich reflektivní vlastnosti. Reflektivní systém je takový systém, který má sám v sobě prostředky pro modifikaci své struktury či svého chování, a to *pouze prostřednictvím* těchto prostředků [29]. Obecně pak můžeme říci, že metaúroveň řídí doménovou úroveň ve dvou krocích [32]: Doménový objekt volá metaobjekt s žádostí o změnu sémantiky svých operací nebo struktury, tj. dochází k operaci *reifikace* (*reification*)[4] implementace nebo sémantických aspektů objektu. Druhý krok představuje předání řízení z metaobjektu zpět doménovému objektu, tj. dochází k procesu *reflexe* nebo také *reflektování* (*reflecting*) provedených změn zpět do doménového objektu.

Formalismem diskutovaným jako základní paradigma simulačního systému jsou Objektově orientované Petriho sítě, které jsou svou podstatou splynutím Petriho sítí a jazyka Smalltalk. Architektura systému prezentovaného v této práci proto vychází z konceptů, které nabízí Smalltalk, přičemž je založena na myšlence maximální úrovně otevřenosti. Výhody takto koncipovaného systému se projeví především ve velké flexibilitě umožňující adaptovat systém podle požadovaných potřeb, a to včetně možnosti měnit či kombinovat různé vyjadřovací prostředky.

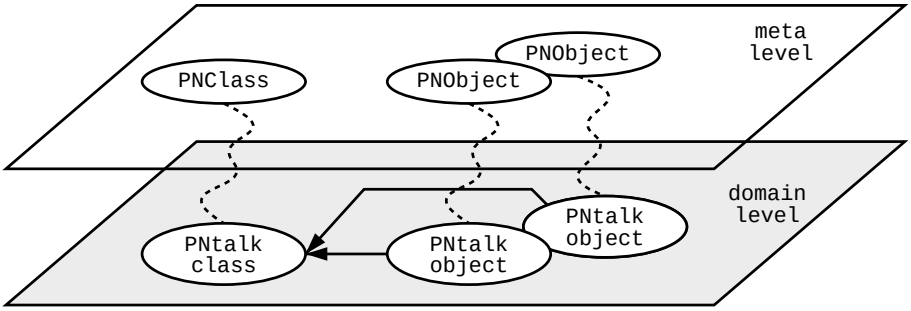
4 Dosažené výsledky

Otevřený simulační systém musí poskytovat metainformace o doménových objektech, ale také (alespoň částečně) metainformace o systému samotném. Návrh simulačního systému je proto koncipován jako metaúrovňový, jehož nejvyšší úroveň zapadá do metaúrovňové architektury systému Smalltalk [15, 21, 22].

4.1 Klasifikace úrovní simulačního systému

Architektura simulačního systému zavádí novou metaúroveň mezi doménovou třídou a objektem na jedné straně a Smalltalkem na straně druhé. Představme si nyní jednotlivé úrovně v systému a jejich význam (vztah doménové úrovně a metaúrovně je naznačen na obrázku 1):

[4] *Reify* podle výkladového slovníku znamená *consider an abstract concept to be real*. V našem pojetí to můžeme chápat jako vytváření konkrétních dat či informací z abstraktních konceptů objektu, které mohou být kontrolovány metaobjektem a tím mohou být kontrolovány příslušné aspekty objektu.



Obrázek 1: Základní úrovně simulačního systému

Doménová úroveň (domain level): Tato úroveň je představována třídami OOPN a jejich instancemi (objekty). Z hlediska doménové úrovně jde o klasický objektově orientovaný přístup k tvorbě modelů. Doménové objekty a třídy jsou *konceptuální objekty*, se kterými se manipuluje při tvorbě modelů v doménovém paradigmatu (v našem případě OOPN). V simulačním systému však tyto objekty reálně neexistují a jsou zastoupeny svými *metaobjekty.*

Metaúroveň (metalevel): Metaúroveň nad doménovou úrovní je reprezentovaná množinou instancí tříd *PNClass* (popisuje chování tříd OOPN) a *PNObject* (popisuje chování živých objektů, tj. instancí tříd OOPN). Tyto instance budeme obecně nazývat *metaobjekty*. Pro každý doménový objekt existuje právě jeden metaobjekt na metaúrovni (a naopak každý metaobjekt na metaúrovni reprezentuje právě jeden doménový objekt). Metaobjekt nad doménovým objektem je *aktor*. Jeho součástí je proces, který serializuje všechny přístupy a požadavky na objekt. Tento proces nazvěme *řídicí proces metaobjektu* (*metaobject control process*).

Meta metaúroveň (meta metalevel): Metaúroveň nad metaúrovní. Je reprezentována třídami *PNClass* a *PNObject* (tj. instancemi svých metatříd ve Smalltalku). Jak už bylo řečeno, systém předpokládá možnost změny doménového paradigmatu (tato možnost je diskutována v kapitole 4.5). V takovém případě budeme rozlišovat *typ doménového objektu*. Pro každý *typ doménového objektu* existuje právě jedna třída na meta metaúrovni. Pro daný typ doménového objektu jsou pak všechny metaobjekty doménových objektů instancemi příslušné třídy na meta metaúrovni.

Metaobjekt řídí operace nad doménovým objektem na základě *řídicích zpráv*. Komunikace doménových objektů (zasílání *synchronních zpráv*) je adaptována na komunikaci metaobjektů prostřednictvím řídicích *asynchronních zpráv*. Metaúrovňovou architekturu simulačního systému si můžeme klasifikovat následovně:

- Architektura kombinuje statickou reflektivitu (tj. reifikační informace jsou získány při překladu modelu) a dynamickou reflektivitu (tj. reifikační infor-

mace mohou být získávány dynamicky za běhu modelu) [7]. Veškeré operace mohou být kdykoliv během evoluce systému inkrementálně změněny.

- Architektura kombinuje strukturální reflektivitu (tj. reifikaci struktury modelu) a reflektivitu chování (tj. reifikaci způsobu provádění operací apod.) [7].
- Architektura implementuje reflektivitu založenou na objektech (tj. metaobjekty reprezentují doménové objekty při jejich evoluci) [5]. V kombinaci se Smalltalkem pak lze využít i reflektivitu založenou na třídách (tj. metaobjekty ovlivňují skutečné doménové objekty).

4.2 Reprezentace doménového objektu

Doménový objekt je v systému reprezentován metaobjektem, který nazveme *doménový metaobjekt* (viz metaúroveň). Evoluce doménového metaobjektu je řízena na základě událostí, které jsou generovány ve vláknech procesů objektu. Vlákno a proces jsou reprezentovány speciálními metaobjekty.

4.2.1 Základní struktury

Vlákno Základní úrovní při běhu (simulaci) modelu je *vlákno*, které je součástí *procesu*. Princip procesů a vláken vychází z předpokladu, že metoda tvoří vzor pro instanciaci nějaké struktury (tj. sítě OOPN, kódu ve Smalltalku apod.) Instance této struktury, tedy *proces*, pak může být složena z více podprocesů, tedy *vláken*. V pojetí OOPN je vlákno chápáno jako *instance přechodu*, tj. přechod popisuje chování vlákna.

Systémový proces Objekt obsahuje množinu procesů, tj. vyvolaných metod. S každým procesem se váže implementace použitého paradigmatu, tj. *interpret*. Interpret provádí interpretaci paradigmatu, generuje události, provádí vybrané události atd. Proces je pak tvořen množinou vláken a množinou událostí, které jsou potenciálně proveditelné. Nyní si definujeme systémový proces jako komponentu na metaúrovni.

Systémový proces *je struktura*

$$P = (Calendar, Threads, Events)$$

kde

- $Calendar$ – *seznam aktivačních záznamů událostí,*[5]

[5] Význam kalendáře a práce s časem jsou popsány k kapitole 4.3.

- $Threads$ – *seznam vláken procesu,*
- $Events$ – *seznam aktuálně proveditelných událostí procesu.*

Třídu všech systémových procesů označíme Π.

Takto specifikovaný systémový proces P je obecný. Každé použité paradigma má definovaný svůj vlastní proces, který musí splňovat požadavky kladené na obecný proces (základní struktura a operace nad strukturou). Ostatní části pak reflektují požadavky použitého paradigmatu a jeho interpretu. Můžeme tedy rozlišovat *typ systémového procesu*. Proces OOPN obsahuje navíc seznamy míst a přechodů.

Kontext metody Každá asynchronní zpráva je zpracována tak, že se vytvoří meta-objekt *kontext metody*, který udržuje informace o vyvolané a zpracovávané metodě. Podle typu zprávy, tj. zda je z doménového protokolu nebo z metaobjektového protokolu, se vytváří kontext metody – buď je reprezentován *systémovým procesem*, nebo speciálním metaobjektem.

Procesní vazba Doménové objekty navzájem komunikují prostřednictvím zasílání zpráv. Pokud nějaký objekt (*sender*) zašle zprávu jinému objektu (*receiver*), objekt *receiver* zprávu zpracuje a předá zpět zdrojovému objektu *sender* návratovou hodnotu. Zpráva je zasílána z vlákna t objektu *sender*. Na základě zprávy se vytvoří proces P v objektu *receiver*. Vlákno t čeká na dokončení procesu P. Mezi vláknem a procesem existuje vztah, který nazveme *procesní vazba*. *Procesní vazbu* definujeme jako dvojici $(thread, process)$, kde $thread$ reprezentuje závislé vláko a $process$ reprezentuje vyvolaný proces.

Systémová zpráva Klíčem pro uchování procesní vazby v doménovém meta-objektu je speciální metaobjekt *systémová zpráva*. Metaobjekt *systémová zpráva* zajišťuje potřebné informace nutné pro korektní zpracování asynchronní zprávy mezi metaobjekty. Systémová zpráva je v rámci systému jedinečná (má jedinečnou identifikaci).

4.2.2 Doménový metaobjekt

Operace na metaúrovni většinou manipulují s interními daty metaobjektů, a proto musí být zajištěna synchronizace těchto operací. Synchronizace je implementována jako vzájemné vyloučení (mutual exclusion) jednotlivých *obsluh událostí* v objektu a vnějších *asynchronních zpráv*. Každá událost v objektu může být provedena pouze na základě speciální asynchronní řídicí zprávy. Pro zajištění vzájemného vyloučení byl proto zvolen způsob serializace příchozích *asynchronních* zpráv, které jsou řazeny do fronty.

Doménový metaobjekt *je struktura*

$$PNObject = (Queue,\, invP,\, depT,\, class,\, LP,\, Processes)$$

kde

- $Queue$ – *fronta delegovaných zpráv,*
- $invP$ – *seznam vyvolaných procesů,*
- $depT$ – *seznam závislých vláken,*
- $class$ – *reference na třídu doménového objektu,*
- $LP \in \Pi$ – *životní proces objektu,*
- $Processes \subset \Pi$ – *seznam procesů doménového objektu.*

Třídu všech doménových metaobjektů označíme Θ.

Procesní vazba je v doménovém metaobjektu uložena prostřednictvím dvou seznamů – *seznamu závislých vláken* a *seznamu vyvolaných procesů* (viz $depT$ a $invP$). Prvkem seznamu závislých vláken je dvojice $(message, thread)$, kde $message$ je systémová zpráva, která reprezentuje zprávu zaslanou z vlákna $thread$. Prvkem seznamu vyvolaných procesů je dvojice $(message, process)$, kde $message$ je systémová zpráva, která byla přijata doménovým metaobjektem a $process$ je proces vyvolaný na základě přijaté zprávy.

4.3 Evoluce systému

4.3.1 Čas v systému

Důležitým aspektem simulace je modelový čas. V prezentovaném simulačním systému si každý objekt udržuje vlastní kalendář událostí, který uchovává informace o časově vázaných událostech. Manipulace s časovými událostmi v doménovém modelu jsou prováděny jako zasílání zpráv vlastnímu objektu (*self*). Např. *self hold: 100* představuje suspendování aktivního vlákna na dobu 100 časových jednotek. Tato zpráva je na doménové úrovni chápána jako synchronní, při jejímž zpracování dojde k suspendování vlákna. Na metaúrovni je tato zpráva zpracována jako zpráva z doménového protokolu. Z principu zpracování této zprávy vyplývá, že vytváří kontext metody. Tento kontext je v obsluze zprávy *hold:* asociován s uvedeným časem a poté vložen do kalendáře. V okamžiku dosažení požadovaného času se tento kontext uvolní a ukončí zpracování metody. Tím se se uvolní původní čekající vlákno, tj. čekání vlákna je ukončeno.

4.3.2 Svět systému

Doménové metaobjekty jsou v systému vždy přiřazeny pod řídicí metaobjekt, který nazveme *svět*. Pro metaobjekt *svět* nebo *doménový metaobjekt* jsou definovány následující pojmy:

- *Stav*. Stav metaobjektu je odvozen od aktivity při zpracovávání asynchronních (tj. delegovaných) zpráv:
  - *pracující* (*busy*) – metaobjekt zpracovává řídicí zprávu / zprávy
  - *připravený* (*ready*) – metaobjekt je ustálený, tzn. že nemá žádnou zprávu ke zpracování
- *Aktivita*. Každý doménový metaobjekt může být aktivní nebo neaktivní. Aktivita objektu znamená pouze jeho označení, zda je zahrnut do evolučního cyklu světa nebo ne. Podobně je definovaná aktivita i pro celý svět – pokud je svět neaktivní, žádný objekt nemůže provádět kroky ve své evoluci.
- *Živost*. Pokud neexistuje v doménovém metaobjektu proveditelná událost, říkáme, že je daný objekt *neživý*. Pokud neexistuje ve světě objekt, který by byl živý, a kalendář událostí je prázdný, pak říkáme, že je svět *neživý*.

Každý objekt je součástí světa, který řídí jeho evoluci v koordinaci s ostatními objekty světa. Svět zajišťuje synchronizaci modelových časů jednotlivých objektů a je iniciátorem podnětů pro kroky evoluce.

Svět *je struktura*

$$PNtalkWorld = (Objects,\ Active,\ Busy,\ Executable,\ Calendar)$$

kde

- $Objects \subset \Theta$ – *seznam objektů světa,*
- $Active \subseteq Objects$ – *seznam aktivních objektů světa,*
- $Executable \subseteq Objects$ – *seznam objektů obsahujících alespoň jednu událost,*
- $Busy \subseteq Objects$ – *seznam pracujících objektů,*
- $Calendar$ – *seznam aktivačních záznamů událostí ve světě.*

Třídu všech světů simulačního systému označíme Ξ.

Řízení objektů světa Algoritmus řízení evoluce objektů světa si můžeme demonstrovat následujícím způsobem (algoritmus předpokládá, že uvažovaný svět je aktivní). Algoritmus je prezentován deklarativní formou, kde pro každou splněnou podmínku $[\ldots]$ (případně konjunkci podmínek $[\ldots] \wedge [\ldots]$) se provede akce $\Rightarrow$.

- $[\exists\, o \in Objects : o \in Busy] \Rightarrow$ svět je ve stavu $busy$
- [svět není $busy$] $\wedge$ $[\neg\exists o \in Objects : o \in Executable] \wedge [\neg\exists t \in Calendar]$ $\Rightarrow$ svět je *neživý*
- [svět není $busy$] $\wedge$ $[\neg\exists o \in Objects : o \in Executable] \wedge [\exists t \in Calendar] \wedge$ [$\neg\exists$ atomická zpráva] $\Rightarrow$ svět vybere nejmenší čas z kalendáře, posune svůj čas a zašle jeho hodnotu všem svým objektům
- [svět není $busy$] $\wedge$ [svět je živý] $\wedge$ $[\exists o \in Objects : o \in Active \wedge o \in Executable]$ $\Rightarrow$ svět zašle zprávu *step* každému objektu $o \in Active \wedge o \in Executable$

Pokud objekt po otestování proveditelnosti přechodů zjistí, že se nemůže vyskytnout v daném stavu žádná událost, zašle tuto informaci metaobjektu *world* prostřednictvím řídicí zprávy. Příslušná asociovaná metoda zruší tento objekt z množiny $Executable$. Z hlediska řízení evoluce světa to znamená, že tomuto objektu není potřeba zasílat řídicí zprávu *step*.

4.3.3 Spojené světy

Ve specifických aplikacích simulačního systému (migrace objektů, spojení s jinými systémy apod.) je někdy vhodné mít více oddělených výpočetních prostorů (světů), které ovšem sídlejí modelový čas. Tuto situaci řeší koncept *spojených světů*. Základní řídicí jednotkou spojených světů je *řídicí svět*. Řídicí svět má jediný úkol: synchronizace modelového času spojených světů.

Řídicí svět *je struktura*

$$WorldController = (Worlds,\, Busy,\, Calendar)$$

kde

- $Worlds \subset \Xi$ *– seznam spojených světů,*
- $Busy \subseteq Worlds$ *– seznam pracujících světů,*
- $Calendar$ *– seznam aktivačních záznamů událostí spojeného světa.*

Algoritmus řízení objektů světa se od své původní verze odlišuje ve dvou bodech: při možnosti posunu času pouze informuje řídicí svět a skutečný posun nastává až na základě podnětu od řídicího světa. Algoritmus řízení spojených světů je podobný algoritmu řízení objektů světa, tj. pokud je spojený svět ve stavu $busy$, nemůže dělat nic, pokud není ve stavu $busy$ a existuje alespoň jeden záznam v kalendáři událostí, vybere záznam s nejmenší hodnotou a vybranou hodnotu zašle všem svým světům. Pokud se svět překlopí ze stavu $busy$ do stavu $ready$, pak zašle tuto informaci řídicímu světu, který zruší příslušný svět ze seznamu $Busy$. Pokud je nyní seznam prázdný, znamená to, že všechny spojené světy čekají na posun času. Z kalendáře se vybere nejmenší čas a jeho hodnota se zašle všem spojeným světům.

4.4 Interoperabilita

Interoperabilitu můžeme chápat jako plně transparentní přístup mezi objekty různých prostředí, v našem případě mezi objekty OOPN a Smalltalku. Hlavní motivací této vlastnosti je možnost použít Petriho sítě (OOPN) jako část programu napsaného ve Smalltalku (a naopak). Problém interoperability mezi objekty OOPN a Smalltalku spočívá v různém výpočetním chování těchto objektů. Ve Smalltalku probíhá komunikace prostřednictvím přímého zasílání zpráv, tzn. že objekt, který zaslal zprávu jinému objektu, čeká na dokončení zpracování zprávy. Objekty OOPN jsou reprezentovány svými metaobjekty, které spolu komunikují prostřednitvím asynchronních (delegovaných) zpráv. Doménové zprávy objektu ve Smalltalku jsou k dispozici přímo jako metody objektu, doménové zprávy objektu OOPN jsou k dispozici prostřednictvím speciální řídicí zprávy. Při komunikaci objektů Smalltalku a OOPN zde proto musí existovat vrstva upravující výpočetní chování na požadovanou podobu. Tuto vrstvu reprezentuje metaobjekt, který nazveme *proxyobjekt*. Prezentovaná problematika je také diskutována v [19].

Proxyobjekty s sebou nesou ještě jednu neméně důležitou vlastnost. Budují vrstvu mezi vysílajícím objektem a jeho příjemcem. Tato vlastnost se uplatní především v okamžiku distribuovaného běhu modelu, kde spolu mohou komunikovat vzdálené objekty, případně kde objekty mohou migrovat mezi jednotlivými uzly simulačního systému.

4.5 Kombinace paradigmat

Jednou z klíčových myšlenek prezentované otevřené architektury je kombinace různých vyjadřovacích prostředků, tj. přístup založený na multiparadigmatu. Při modelování v simulačním systému můžeme použít multiparadigma dvěma různými způsoby. Buď zařadíme zvolené paradigma přímo do simulačního systému, nebo spojíme objekty simulačního systému s objekty jiného systému (tj. vytvoříme *multimodel*). Pro potřeby předložené disertační práce si nyní představíme tyto přístupy

v úvahách nad kombinací se Smalltalkem [8] a se systémem DEVS [1] a rovněž v možnosti začlenění jiného paradigmatu přímo do prostředků simulačního systému.

Smalltalk a OOPN Hlavní problémy se spojením objektů modelu v systému s jeho okolím jsou v zásadě tyto dva: Protože okolí systému nepodléhá jeho evolučnímu řízení, je nutné zajistit *kompatibilitu výpočetního chování* (tj. zasílání a zpracování zpráv) a je nutné zajistit *synchronizaci modelového času*.

- *Kompatibilita výpočetního chování*. Pro zajištění transparentní komunikace mezi objekty simulačního systému a objekty Smalltalku byly zavedeny speciální proxyobjekty (viz kapitola 4.4). Každá reference na objekt, který je součástí simulačního systému, nebo se kterým komunikují objekty simulačního systému, je realizována prostřednictvím proxyobjektů.

- *Synchronizace časů*. Uvažovaný objekt musí implementovat základní rozhraní *doménového metaobjektu* a být zařazen do vybraného světa. Pak může pro manipulaci s časem využít metaobjektový protokol světa. Pokud budeme uvažovat situaci, kdy je množina externích objektů řízena speciálním objektem (např. DEVS), můžeme tento objekt abstrahovat nejen jako speciální doménový metaobjekt, ale i jako svět v simulačním systému. Pro zajištění této vlastnosti lze s výhodou využít koncepci *spojených světů*.

Změna doménového paradigmatu Architektura systému, tak jak byla prezentována, umožňuje poměrně jednoduchým způsobem začlenit do systému nové doménové paradigma, případně provést modifikaci stávajících paradigmat. Začlenění nového paradigmatu má význam v případě, že chceme běh části modelu, která je popsána tímto paradigmatem, plně podřídit simulačnímu systému. Každá interpretace doménového paradigmatu simulačního systému je svázána se systémovým procesem. Změnu doménového paradigmatu můžeme provést na základě změny systémového procesu. Každý proces objektu je vytvořen v reakci na zaslání zprávy, a tedy invokaci příslušné metody. Každá metoda je svázána se svým doménovým paradigmatem, a tudíž dokáže determinovat systémový proces, který se má vytvořit. Takže pouhým začleněním příslušných informací do reprezentace metody můžeme dosáhnout multiparadigmatu na úrovni jednotlivých metod objektu. S každým typem doménového paradigmatu je svázán jeho překladač a interpret příslušného systémového procesu, který musí dodržovat protokol definovaný pro systémový proces. Interpret a překladač musí být implementovány a zařazeny do meta metaúrovně simulačního systému. Při vytvoření instance nového procesu je pak nový systémový proces automaticky zařazen do struktur doménového metaobjektu.

Začlenění DEVS do simulačního systému. V této části si pouze přiblížíme možnosti kombinace DEVS a simulačního systému. Jejich podrobnější popis, včetně

popisu systému DEVS, je uveden v disertační práci. V kapitole 4.1 jsme definovali pojem *typ doménového objektu*. Ke každému typu doménového objektu existuje speciální doménový metaobjekt. Atomický nebo spojovaný DEVS může být chápán jako speciální typ doménového objektu, který je reprezentován speciálním doménovým metaobjektem (nazveme jej *doménový metaobjekt DEVS*). Tento metaobjekt, který je začleněn do světa jako jeho komponenta, nahrazuje kořenový koordinátor v DEVS. Při komunikaci se svou podřizenou komponentou uplatňuje původní zprávy DEVS simulátorů. Neprovádí však rozhodování o posunu času, ale tvoří spojovací článek mezi DEVS modelem a světem. Pro komunikaci se světem pak platí následující pravidla:

- Po simulačním kroku zjistí doménový metaobjekt DEVS čas následující události a předá ji světu.
- Pokud svět rozhodne o posunu času, informuje všechny své komponenty. Pokud je nový čas světa t_w roven času t doménového metaobjektu DEVS, pokračuje ve svém původním algoritmu evoluce DEVS (tj. informuje podřízenou komponentu o novém čase).

Začlenění simulačního systému do DEVS. V tomto případě musíme uvažovat spojovaný DEVS (*coupled DEVS*), do kterého se začlení *svět* simulačního systému jako jeden z atomických DEVS. Protokol světa musí být upraven tak, aby odpovídal dané implementaci spojovaného DEVS. Evoluce času světa by se podřídila spojovanému DEVS, tzn. že impulzy k posunu času musí vycházet od řídicího objektu spojovaného DEVS (tj. od jednoho z DEVS koordinátorů).

4.6 Aplikovatelnost otevřeného simulačního systému

Podstatným důsledkem otevřeného návrhu simulačního systému je možnost kontroly doménových objektů prostřednictvím jejich metaobjektů a možnost tuto kontrolu provádět z doménové úrovně. Tento koncept umožňuje provádět (mimo jiné) různé simulační techniky. Jako ukázku aplikace vnořené simulace si můžeme uvést následující kód akce přechodu:

world ← *self meta world.*
newWorld ← *world clone.*

Operace *self meta* představuje získání reference na metaobjekt daného doménového objektu, a tedy doménový objekt může provádět reifikační operace nad metaobjekty. Zprávou *world* z metaobjektového protokolu pak získáme referenci na metaobjekt reprezentující svět, do kterého je objekt vložen. Voláním metaobjektové operace *world clone* vytvoříme *klon* našeho prostředí, které po dokončení vytvoření

pokračuje v evoluci jako původní svět. Protože v průběhu reifikačního procesu, vyvolaného metodou *clone*, nelze reflektovat stav této operace do nově vytvářeného klonu, je v klonu tato operace ukončena s hodnotou *nil*. Pro úplnost dodáváme, že operace *clone* je asynchronní, čímž je zajištěno její výlučné zpracování.

Pokud bychom chtěli dosáhnout jistých změn před vlastní evolucí klonu, můžeme místo operace *clone* použít operaci *cloneAndSuspend*, která po dokončení operace klonování zastaví evoluci vytvořeného světa. Následující ukázka vytváří suspendovaný klon světa, získává referenci na klon objektu, ve kterém je tento kód obsažen (proměnná *clone*, lze toho dosáhnout díky tomu, že *lokální* identifikace objektů v původním a nově vytvořeném světě je stejná), a vkládá novou instanci objektové sítě. Poté nově vytvořený svět aktivuje.

```
world ← self meta world.
newWorld ← world cloneAndSuspend.
newObject ← newWorld componentID: (self meta id).
newObject compile: 'zdrojový kód objektové sítě'.
newWorld resume.
```

5 Závěr

Předkládaná disertační práce se zabývá modelováním, simulací a prototypováním složitých systémů s ohledem na dynamické aspekty vytvářených modelů a prototypů. Diskutuje objektově orientované přístupy k modelování a prototypování. Jedním z možných přístupů k řešení této problematiky je využití objektově orientované modifikace Petriho sítí, která kombinuje prostředky pro popis struktury (objektová orientace) a prostředky pro popis chování na formálním základě (Petriho sítě). Objektově orientované Petriho sítě (OOPN), díky své obecnosti, tvoří základní paradigma tvorby modelů, které bylo uvažováno v této práci.

Celým textem práce se prolínají myšlenky dynamické evoluce modelů (tj. schopnost modifikovat strukturu a chování modelu během jeho evoluce) a prototypování. Dynamická evoluce představuje výborný a jednoduchý prostředek pro aplikaci speciálních simulačních technik, modelování agentů apod. Ve spojení s prototypováním umožňuje inkrementální vývoj modelů softwarových systémů, jednoduchou převoditelnost formálního modelu na funkční prototyp apod. Stěžejním cílem práce byla diskuze principů otevřených systémů, jejich aplikace na principy evoluce OOPN a návrh otevřené architektury simulačního systému, který umožňuje uvažovat dynamické manipulace s modely.

Za zásadní výsledek lze považovat vybudování konceptuálního základu reflektivního systému, který umožňuje dynamický vývoj modelů, dynamickou evoluci modelů a kombinaci různých vyjadřovacích prostředků. Otevřená architektura navíc umožňuje zásahy do řídicích algoritmů a struktur, a to i z běžícího modelu.

Pro dosažení požadovaných vlastností byly využity reflektivní vlastnosti otevřených implementací a zavedena interoperabilita objektů OOPN se svým okolím, tj. s objekty Smalltalku. Prezentovaná architektura zahrnuje klasifikace výpočetních úrovní v systému, entit jednotlivých úrovní a vztahů mezi těmito entitami. Klasifikace struktur popisujících běžící model (a tomu odpovídajících struktur popisujících model) na objekty, procesy a vlákna, včetně metaobjektového protokolu a algoritmů evoluce, umožňuje začleňovat nová paradigmata pod plnou kontrolu simulačního systému, kombinovat tato nová paradigmata se stávajícími (na úrovni objektů i metod) a dynamicky je přizpůsobovat měnícím se požadavkům. Navržené algoritmy a principy byly prototypově ověřeny.

Klasifikace struktur simulačního systému a klasifikace výpočetního chování simulačního systému přináší možnost multimodelování, kde jsou submodely vytvářeny s využitím různých prostředků a systémů (např. PNtalk, procesně orientovaná simulace ve Smalltalku, DEVS). Současně je tento koncept použitelný při tvorbě prototypů, kdy je výhodné část prototypu implementovat např. objekty ve Smalltalku a část prototypu modelovat např. Petriho sítěmi. Reflektivní architektura simulačního systému (dynamické změny chování a struktur) ve spojení s prototypováním umožňuje reálné nasazení systému v oblastech s vysokým stupněm dynamičnosti vazeb jako např. agentní systémy [31], proces návrhu aplikací [20], prototypování aplikací apod.

Reference

[1] T. Kim B. Zeigler and H. Praehofer. *Theory of Modeling and Simulation*. Academic Press, 2nd edition, 2000.

[2] F. J. Barros. Modeling formalisms for dynamic structure systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 7(4):501–515, 1997.

[3] J. Briot. Actalk : A Framework for ObjectOriented Concurrent Programming - Design and Experience, 1999.

[4] M. Češka, V. Janoušek, and T. Vojnar. PNtalk – A Computerized Tool for Object-Oriented Petri Nets Modelling. *Computer Aided Systems Theory and Technology – EUROCAST'97*, 1333:591–610, 1997.

[5] S. Chiba. *A study of Compile-Time Metaobject Protocol*. PhD thesis, 1996.

[6] R. Esser, J. Janneck, and M. Naedele. Applying an object-oriented petri net language to heterogeneous systems design. In *In Petri Nets in Systems Engineering*, 1997.

[7] J. Ferber. Computational reflection in class based object oriented languages. In *Proceedings of the 4th Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA'89)*, volume 24, pages 317–326. ACM Press, 1989.

[8] A. Goldberg and D. Robson. *Smalltalk 80: The Language*. Addison-Wesley, 1989.

[9] B. Gowing and V. Cahill. Making Meta-Object Protocols Practical for Operating Systems. In *4^{th} International Workshop on Object Orientation in Operating Systems*, pages 52–55, Lund, Sweden, 1995.

[10] M. Hrubý. *Prostředí pro modelování heterogenních systémů*. PhD thesis, 2004.

[11] M. Hrubý and R. Kočí. Modern Simulation Techniques and Tools. In *Proceedings of the International Workshop MOSMIC'2003*, pages 7–15, Zilina, SK, 2003. FRI ZU.

[12] M. Hrubý, R. Kočí, P. Peringer, and Z. Rábová. Tools for Creating of Multimodels. *Kybernetes: The International Journal of Systems & Cybernetics*, 31(9):1391–1400, 2002.

[13] V. Janoušek. PNtalk: Object Orientation in Petri nets. In *Proc. of European Simulation Multiconference ESM'95*, pages 196–200, Prague, Czech Republic, 1995.

[14] V. Janoušek. *Modelování objektů Petriho sítěmi*. PhD thesis, 1998.

[15] V. Janoušek and R. Kočí. PNtalk: Concurrent Language with MOP. In *Proceedings of the CS&P'2003 Workshop*, pages 271–282, Warsawa, PL, UW, 2003.

[16] G. Kiczales, J. des Rivieres, and D. G. Bobrow. *The Art of the Metalevel Protocol*. MIT Press, 1991.

[17] G. Kiczales, J. Lamping, A. Menhdhekar, Ch.Maeda, C. Lopes, J.M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[18] E. Kindler and M. Weber. The Petri Net Kernel - An infrastructure for building Petri net tools. *International Journal on Software Tools for Technology Transfer*, 3(4):486–497, 2001.

[19] R. Kočí. Prospects of the PNtalk system in the Smalltalk environment. In *Proceedings of 7th Conference Student FEI 2001*, pages 338–342, Brno, CZ, 2001. VUT.

[20] R. Kočí. The PNtalk System - a Technique for Object Oriented Modelling. In *Proceedings of XXIIIrd International Autumn Colloquium*, pages 151–158, Ostrava, CZ, 2001. MARQ.

[21] R. Kočí. The Open Architecture of the PNtalk System. In *Proceedings of the International Conference and Competition - Student EECIT 2003*, pages 358–362, Brno, CZ, 2003.

[22] R. Kočí and Z. Rábová. Purposes of the PNtalk System. In *Proceedings of International Conference MOSIS '03*, pages 149–156, Ostrava, CZ, 2003. MARQ.

[23] R. Kočí and T. Vojnar. A PNtalk-based Model of a Cooperative Editor. In *Proceedings of the 35th Spring International Conference on Modelling and Simulation of Systems – MOSIS 2001*, pages 165–172, Hradec nad Moravicí, CZ, 2001. MARQ.

[24] P. Krief. *Prototyping with objects*. T.J.Press Ltd, Padstow, Cornwall, UK, 1996.

[25] C. Lakos. Object Oriented Modelling with Object Petri Nets. In *Advances in Petri Nets, LNCS*, Berlin, 1997. Springer.

[26] C. A. Lakos. From Coloured Petri Nets to Object Petri Nets. In *Proceedings of the Application and Theory of Petri Nets 1995*, volume 935, pages 278–297. Springer-Verlag, Berlin, Germany, 1995.

[27] Ch. Lakos. Towards a Reflective Implementation of Object Petri Nets. In *Proceedings of TOOLS Pacific*, pages 129–140, Melbourne, Australia, 1996.

[28] P. Maes. Computational reflection. Technical report, Artifical Inteligence Laboratory, Vrije Universiteit Brusel, 1987.

[29] H. Masuhara, S. Matsuoka, T. Watanabe, and A. Yonezawa. Object-oriented concurrent reflective languages can be implemented efficiently. In Andreas Paepcke, editor, *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, volume 27, pages 127–144, New York, NY, 1992. ACM Press.

[30] D. Moldt. OOA and Petri Nets for System Specification. In *Application and Theory of Petri Nets; Lecture Notes in Computer Science, 16th International Conference*. Italy, 1995.

[31] D. Moldt and F. Wienberg. Multi-Agent-Systems Based on Coloured Petri Nets. In *ICATPN*, pages 82–101, 1997.

[32] R. Pawlak. Metaobject Protocols For Distributed Programming. Technical report, Laboratoire CNAM-CEDRIC, Paris, 1998.

[33] P. Peringer. *Hierarchické modelování na bázi komunikujících objektů*. PhD thesis, 1996.

[34] R.J. Pooley. *An Introduction to Programming in Simula (Computer Science Texts)*. Alfred Waller Ltd, 1987. Kniha dostupná na: http://www.cee.hw.ac.uk/˜rjp/bookhtml/.

[35] The Renew Home Page, http://www.renew.de/, 2004.

[36] C. Sibertin-Blanc. Cooperative Nets. In Robert Valette, editor, *Proceedings of Application and Theory of Petri Nets*, volume 815, pages 471–490. Springer-Verlag, Berlin, Germany, 1994.

[37] The TUNES Project for a Free Reflective Computing System, http://tunes.org, 2004.

[38] A. M. Uhrmacher. Dynamic structures in modeling and simulation: a reflective approach. In *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, volume 11, pages 206–232. 2001.

[39] The UML Home Page, http://www.uml.org, 2004.

[40] R. Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In *Jorg Desel, Manuel Silva (eds.): Application and Theory of Petri Nets; Lecture Notes in Computer Science*, volume 1420, pages 1–25. Springer-Verlag, Berlin, 1998.

[41] Y. Yokote. The apertos reflective operating system: The concept and its implementation. In Andreas Paepcke, editor, *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, volume 27, pages 414–434, New York, NY, 1992. ACM Press.

[42] M. Zapf and A. Heinzl. Techniques for Integrating Petri Nets and Object-Oriented Concepts.

[43] C. Zimmermann. *Advances in Object-Oriented Metalevel Architectures and Reflection*. CRC Press, 1996.

Ing. Radek Kočí

Curriculum Vitae

Základní údaje

Datum narození:	23. 4. 1977
Místo narození:	Moravský Krumlov

Vzdělání

2000 – 2004	Fakulta informačních technologií VUT v Brně student doktorského studijního programu Informační technologie státní závěrečná zkouška složena dne 24.6.2002
1995 – 2000	Fakulta elektrotechniky a informatiky VUT v Brně Ing. v oboru Výpočetní technika a informatika státní závěrečná zkouška složena dne 27.6.2000

Výzkumné projekty

- Automatizované metody a nástroje pro vývoj spolehlivých paralelních a distribuovaných systémů, GAČR GA102/04/0780 (člen týmu)
- Systém PNtalk pro prototypování aplikací, FRVŠ MŠMT, FR1959/2002/G1 (vedoucí týmu)
- Prostředí pro vývoj, modelování a aplikaci heterogenních systémů, GAČR, GA102/01/1485 (člen týmu)
- Modelování, verifikace a prototypování distribuovaných aplikací s využitím Petriho sítí, GAČR,GA102/00/1017 (člen týmu)

Methods and Tools for Implementation of Open Simulation Systems

Abstract

Modern simulation systems work with dynamic models. These models can be modelled by different kinds of paradigms, these models can change during the simulation life-cycle, the used paradigms can be evolved, etc. Thus, modern simulation systems become more and more complex. Recent trend in modern systems for complex application support is not only to allow applications to use services offered by the certain system, but also to offer means to control how these services are provided and processed. Such systems are often called open systems.

The open systems approach can seem to be contrary to the more traditional approach – the black-box abstraction – which says that each abstraction of entities (objects) should expose its functionality but hide its implementation details. The black-box abstraction has many attractive qualities and brings a possibility of portability, reusing or simplicity of the design process. Nevertheless, it does not allow to adapt parts of the system according to changing requirements, to develop applications during its life-time etc. The open implementation principles offer a solution of these requirements. It is needed to remark that the principles of the open implementation approach should be rather understood like the framework intended for more flexible design and use of black-boxes. The foundation to expression of open implementation is to provide a general implementation framework making easier the user's goals, whether their intention is to design or to use open implementations. One of that frames is the metalevel architecture linked to the metaobject protocol (MOP). MOP provides a solution based on object orientation which can be integrated to the standard development processes in a simple way.

Simulation models are often made in the one formalism with using the one methodology. The models are then interpreted (simulated) by means of the uniform simulation technique. Of course, when we want to work with complex models, it is better to combine various kinds of suitable paradigms or suitable simulation techniques. Modelling generally consists of two levels: the modelling of static structures and the modelling of dynamic aspects (behavior) of a modeled system. Simulation can be understood as interactions of those structures based on the defined behavior. To describe different aspects of the modelled world we can use different paradigms and formalisms. Since 1994 the research group at Brno University of Technology has developed an Object Oriented Petri Nets (OOPNs) formalism. OOPNs benefit from the features of Petri nets (formal nature, suggestive description of parallelism, theoretical background) as well as object-orientedness (structured organization, dynamic creation of instances of substructures, etc.) OOPNs has been developed as a part of the PNtalk project. The intent of this project is to combine Petri nets and the

Smalltalk system as consistently as possible. Nowadays the PNtalk system is becoming to be not only the means for modelling and simulation of complex concurrent systems but also the means to prototype such systems.

This Ph.D. thesis deals with an open implementation of the simulation system which converges to a lite form of an operating system serving for modelling, simulation, and prototyping of complex systems. The advantages of open implementations should approve in a robust flexibility of the system enabling to adapt environment to required needs, including a possibility to change or to combine different paradigms. One of possible application domain is artificial intelligence, especially the area of intelligent multi-agent systems. The process of the agent reasoning can be characterized by its inner structures changes. Moreover, the whole structure of a multi-agent system can be highly dynamic. The goals of this Ph.D. thesis are to check the promising features of discussed open systems in the simulation environment and to outline an applicability of such system in the area of the model specification or the complex system design.