VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

WEBOVÉ UŽIVATELSKÉ ROZHRANÍ PRO INSPEKCI A EDITACI MODELŮ A SIMULACÍ

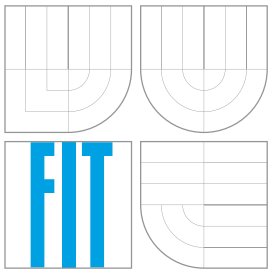
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

SVATOPLUK ŠPERKA

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

WEBOVÉ UŽIVATELSKÉ ROZHRANÍ PRO INSPEKCI A EDITACI MODELŮ A SIMULACÍ

WEB USER INTERFACE FOR INSPECTION EDITING OF MODELS AND SIMULATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — SVATOPLUK ŠPERKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. VLADIMÍR JANOUŠEK, Ph.D.
SUPERVISOR

BRNO 2007

Zadání bakalářské práce

Řešitel: Svatopluk Šperka

Obor: Informační technologie

Téma: Webové uživatelské rozhraní pro inspekci a editaci modelů a simulací

Kategorie: Modelování a simulace

Pokyny:

1. Seznamte se s prostředím Squeak Smalltalk, s jeho webovými servery a frameworky pro tvorbu webových aplikací.
2. Seznamte se s prostředím SmallDEVS.
3. Navrhněte webové uživatelské rozhraní pro inspekci a editaci modelů a simulací v prostředí SmallDEVS.
4. Navržené prostředky realizujte a demonstrujte na vhodných příkladech.

Literatura: dle pokynů vedoucího

Vedoucí: Janoušek Vladimír, Ing., Ph.D., UITS FIT VUT

Datum zadání: 1.listopadu 2006

Datum odevzdání: 15. května 2007

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Při reálném nasazení simulace vytvořené v modelovacím a simulačním prostředí SmallDEVS, např. na vestavěném zařízení, vyvstává otázka, jak tuto simulaci sledovat, spravovat či případně měnit její chování. Nativní uživatelské rozhraní je v takovém případě nedostupné a příkazová řádká není z důvodu velice omezených možností vizuální interpretace informace vhodná. Internet a s ním spojené technologie hypertextových dokumentů jsou dnes již natolik rozšířené a prostředky pro jejich dynamickou tvorbu sofistikované, aby toto médium mohlo být využito jako platforma pro implementaci vzdáleného rozhraní. Tato práce popisuje návrh a implementaci takového webového uživatelského rozhraní pro inspekci a editaci modelů a simulací.

Klíčová slova

modelování, simulace, uživatelské rozhraní, web, DEVS, Smalltalk, SmallDEVS

Abstract

In real situation / industry setting of simulation created using modeling and simulation framework SmallDEVS, for example on embedded system, problem how to monitor, administer and change behaviour of this simulation arises. Native graphic user interface is inaccessible and command line interface is inappropriate because of its limitations in visual interpretation of information. Internet and hypertext document technologies are widespread and tools for their dynamic generation so sophisticated that this medium is suitable as a platform for implementing remote user interface. This document describes design and implementation of such web user interface for inspection and editing of models and simulations.

Keywords

modeling, simulation, user interface, web, DEVS, Smalltalk, SmallDEVS

Citace

Svatopluk Šperka: Webové uživatelské rozhraní pro inspekci a editaci modelů a simulací, bakalářská práce, Brno, FIT VUT v Brně, 2007

Webové uživatelské rozhraní pro inspekci a editaci modelů a simulací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Vladimíra Janouška

.....

Svatopluk Šperka

14. května 2007

Obsah

Kapitola 1

Úvod

SmallDEVS je systém pro modelování a simulaci založený na formalismu DEVS a prototypově orientovaném přístupu k objektovému modelování, s cílem umožnit interaktivní a evoluční návrh systémů. Jeho uživatelské rozhraní je postaveno na možnostech Squeaku, ve kterém je celý SmallDEVS implementován. Toto rozhraní je ideální ve fázi vytváření a ladění modelu. Nabízí vysokou interaktivitu, okamžitou reakci na požadavky uživatele a pohledy na systém, které jsou synchronizovány s jeho skutečným stavem.

Výchozími body této práce, jako právě oborem modelování a simulace a formalismem DEVS se zabývá kapitola 2, samotným frameworkem SmallDEVS, jakožto základním kamenem potom kapitola 3.

Při reálném nasazení již ale není nezbytné a často ani možné k zařízení, na kterém simulační model běží (např. vestavěné zařízení či vzdálený systém), fyzicky přistoupit a využít tak výše zmíněného uživatelského rozhraní. Je ale nezbytné mít možnost sledovat stav systému a případně upravovat model či parametry simulace. Tímto vzniká potřeba rozhraní, které by vzdáleně zpřístupňovalo maximum informací o modelech, simulacích a případně dalších objektech spravovaných systémem SmallDEVS a nabízelo základní množinu operací pro práci s nimi.

Počítačové sítě, potažmo Internet, jsou dnes všudypřítomné a nabízí relativně standardizované technologie pro přenos a prezentaci informací. S využitím dynamického generování hypertextových dokumentů je možno zpřístupnit poměrně sofistikované aplikace. Pokud jsme ochotni přistoupit na nižší interaktivitu uživatelského rozhraní, máme k dispozici ověřenou platformu s klienty dostupnými napříč širokým spektrem operačních systému a zařízení. Této problematice se dále věnuje oddíl 2.3, tvorbě webových aplikací v prostředí Squeak Smalltalk potom část 4.2.

Kapitola 4 se zabývá přesnou specifikací požadavků na webové uživatelské rozhraní a návrhem přístupu k jeho realizaci. Samotnou realizací navržené architektury se potom zabývá 5. kapitola. Poslední kapitola hodnotí přínos celé práce a navrhuje potenciální možnosti vylepšení a rozšíření.

Kapitola 2

Výchozí body

V této kapitole bude ve stručnosti podán základ problematiky, ve které tato práce vzniká. Bude nastíněn obor modelování a simulací a jeho význam obecně. Popsán bude formalismus DEVS, na kterém je postaven framework SmallDEVS. Další část bude věnována technologiím spjatým s hypertextovými dokumenty a jejich přenosem po internetu. V závěru se podíváme na současný vztah webových technologií a oboru simulací.

2.1 Modelování a simulace

V teorii systémů se modelováním se rozumí vytváření modelu (napodobeniny) systému. Tento model a jeho simulace (experimentování s reprezentací simulačního modelu) vede v konečném důsledku k rozšíření znalostí o původním systému [14].

Modelování je hojně využíváno především při návrhu nových systémů, protože je ekonomicky výhodnější mít možnost analyzovat systém bez nutnosti ho stavět. Často se ale využívá i při zkoumání systémů, které nejsou z různých důvodů postižitelné jinými metodami. Jedná se například o procesy, které nejsme schopni precizně matematicky formulovat či neznáme metody analytického řešení, děje jejichž chování je roprostřeno na řádově rozdílných časových intervalech, než jaké je člověk a jeho technika schopna analyzovat či zachytit.

2.2 Discrete Event System Specification

DEVS je formalismus vyvinutý Bernardem P. Zeiglerem (1976), který poskytuje prostředky pro specifikaci matematického objektu zvaného systém [2]. V tomto formalismu je model buď atomický nebo sdružený[1] (potom odpovídá klasické definici systému, jakožto množině entit, které mezi sebou nějakým způsobem interagují – existují mezi nimi vazby).

Atomický model je definován jako sedmice:

$$M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \tau)$$

[1] Označováno také jako spojovaný.

Jednotlivé složky mají následující význam:

- X je množina externích událostí.
- S je množina sekvenčních stavů.
- Y množina výstupních hodnot (událostí).
- $\delta_{int} : S \to S$ je přechodová funkce vnitřního stavu.
- $\delta_{ext} : Q \times X \to S$ je přechodová funkce vnejšího vstupu, kde $Q = \{(s, t_e) \mid s \in S, t_e \in \langle 0; \tau(s) \rangle\}$. Q specifikuje aktuální stav systému a dobu, po kterou se v tomto stavu nachází.
- $\lambda : S \to Y \bigcup \{\epsilon\}$ je výstupní funkce, kde ϵ reprezentuje nulový výstup.
- τ je časová postupová funkce, která vrací životnost stavu.

Je možné se v literatuře setkat i s mírně odlišným značením. τ bývá označnováno „ta“ a δ_{int} s δ_{ext} bývá sdružováno do δ označované jako kvasi-přechodová funkce (např. [4]).

Sdružené modely modely zajišťují hiearchičnost celého systému, kdy na různých úrovních pohledu jsou skryty detaily o dílčích subsystémech. To zajišťuje relativní srozumitelnost modelů.

I když je DEVS původně určen pro diskrétní systémy, existují jeho úpravy i pro modelování spojitých procesů a mnoho dalších úprav a rozšíření – např. Parrallel DEVS (Himmelspach, Ewald, Leye, Uhrmacher 2007) pro paralelní systémy s distribuovaným simulátorem či Cell-DEVS (Wainer, Giambiasi 2001), který propojuje celulární automaty s formalismem DEVS.

2.3 Technologie spojené s tvorbou a šířením webových stránek

Hypertextové dokumenty se liší od klasických dokumentů tím, že obsahují hyperlinky, které odkazují na jiné dokumenty. Tím dochází k propojení těchto dokumentů a vytvoření navázností mezi informacemi v nich obsažených. Na tomto konceptu je postaven i World Wide Web (Berners-Lee, Cailliau 1990), což je systém hypertextových dokumentů, běžící nad internetem. Ten měl pomocí jednotného rozhraní (browseru) zpřístupnit dokumenty různých typů jako síť uzlů, ve které se může uživatel dle vlastní vůle pohybovat [10].

Pro popis struktury a obsahu hypertextových dokumentů na Webu byl vytvořen jazyk HTML a později XHTML, který ma stejné vyjadřovací schopnosti jako HTML (Hypertext Markup Language) pouze jeho syntaxe odpovídá jazyku XML. Pro definování vzhledu dokumentů se používá jazyk CSS (Cascading Style Sheets). Spoluprací těchto jazyků je možno oddělit obsah dokumentu od jeho konečného vzhledu. To zlepšuje přístupnost dokumentu (ve smyslu dostupnosti lidem s postiženími), přizpůsobitelnost vzhledu možnostem zařízení

či média, na kterém se zobrazuje a také umožňuje použít stejný styl na více dokumentů, což v konečném důsledku vede k redukci práce potřebné ke sjednocení vzhledu dokumentů.

Pro přenos hypertextových dokumentů na internetu se používá protokol HTTP (Hypertext Transfer Protocol). Je to bezstavový protokol, postavený na konceptu dotaz/odpověď (angl. request/response). Jedná se tedy čistě o přenos dat mezi klientem a serverem bez uchovávání kontextu proběhnuvší komunikace.

Internet je dnes široce dostupné médium a kromě zpřístupňování dokumentů začíná být využiván i jako platforma pro tvorbu aplikací. K tomu se využívá dynamického generování stránek, což je označení pro vytváření obsahu stránek na základě kontextu a podmínek, ve kterých je stránka požádována. Uchování kontextu řeší různé nástroje pro dynamické generování stránek různě – v principu se ale jedná o skrytá pole ve formulářích nebo parametry adresy (URL), pomocí kterých je identifikován kontext dotazu.

Výhodami webových aplikací je jednožnačně dostupnost z libovolného místa, vyžadující pouze běžné programové vybavení, snadné sdílení dat v těchto aplikacích vytvořených i aktuálnost verze dané aplikace. Omezení jsou ale poměrně značná, především co se týká možností uživatelského rozhraní. Celé prostředí nebylo navrženo s ohledem na interaktivitu. Server nemůže aktualizovat data u klienta bez toho, aby si toto klient vyžádal [9].

Tyto problémy jsou do značné míry řešeny skripty na straně klienta v jazyce Javascript, který umí nejrozšířenější prohlížeče na trhu interpretovat. Na něm je postavena technologie AJAX (Asynchronous Javascript and XML), která využívá Javascriptu k aktualizaci částí stránky pomocí komunikace se serverem na pozadí. Jiným řešením jsou technologie třetích stran jako Flash společnosti Adobe či WPF/E (nově pod označením Silverlight) společnosti Microsoft. Dostupnost těchto technologií je již ale nižší co do spektra podporovaných zařízení a platforem. Při jejich použití je nutno, pokud nechceme snížit přístupnost, zajistit, aby takové stránky byly zobrazitelné i na nepodporovaných platformách, což de facto znamená implementovat jednu funkcionalitu dvakrát.

2.4 Web-based simulation

Web-based simulation je souhrnný název pro fenomén propojování webu a jeho technologií s odvětvím modelování a simulací. Toto propojení potenciálně umožňuje integraci, která může klíčové ovlivnit budoucí výzkum na poli simulací [7].

Nabízí se několik směrů. Jedná se především o paralelní a distribuovanou simulaci a distribuované uložiště modelů. Pro spolupráci mezi různými systémy je třeba řešit formalizaci přenášených informací. Objevují se snahy využít k přenosu takových dat jazyk XML. Jako nejzajímavější se jeví na něm postavený *Sémantický web* a *Webové služby* [8].

V případě distribuovaného uložiště modelů je žádoucí uložiště integrovat s ostatními informacemi na webu. Velmi důležitým krokem je možnost adresovat modely v takových uložištích jedinečnou URL.

Co se ale nejvíce dotýká této práce je zpřístupnění modelovacích a simulačních fra-

meworků prostřednictvím webových prohlížečů. Příkladem takového rozhraní může být např. *M & MEMS*, což je webové rozhraní pro přístup k MEMS[2] simulátoru Sugar[3] běžícím na clusteru umístěném na univerzitě v Berkeley. Ten umožňuje pomocí webového rozhraní implementovaném v PHP, nahrávat do simulátoru popisy zařízení, spouštět jejich simulace a prohlížet jejich výsledky [3]. Neumožňuje tedy editace modelů vzdáleně.

Příkladem rozhraní, které takovou editaci umožňuje je *DYNAST*[4] vyvíjený na ČVUT v Praze. Je to systém pro simulaci a analýzu obecných lineárních i nelineárních systémů. Ten nabízí Java applet *DynCAD*, ve kterém je možno systém drag&drop způsobem poskládat z grafických symbolů [11]. Ten je možno následně i vzdáleně simulovat.

Dalším podobným je webové rozhraní pro přistup k systému postaveném na simulačním jazyce *SIMAN*[5]. Ten umožňuje simulaci diskrétních i spojitých systémů. Dovoluje přidávat, mazat a textově editovat modely a vzdáleně je kompilovat a spouštět [1].

Velice zajímavým rozhraním je *WebCell*[6], určený pro správu kvantitativních a kvalitativních informací o buněčných sítích, dále pro průzkum jejich stavu a dynamického chování. Umožňuje konstrukci, vizualizaci a analýzu modelů. Tak usnadňuje kinetické modelování a simulaci biologických systémů [6]. *WebCell* je implementován kombinací Java Server Pages a Javových appletů.

[2]Microelectromechanical Systems, Mikroelektromechanické systémy
[3]http://www-bsac.eecs.berkeley.edu/cadtools/sugar/sugar/
[4]http://icosym-nt.cvut.cz/cacsd/msa/onlinetools.shtml
[5]http://mat.gsia.cmu.edu/simul/node21.html#SECTION00080000000000000000
[6]http://webcell.kaist.ac.kr/

Kapitola 3

SmallDEVS

SmallDEVS je prostředí pro modelování a simulaci, postavené, jak název napovídá, na formalismu DEVS. Jeho snahou je přiblížit tvorbu softwaru tvorbě fyzického díla a vzdálit ji tak klasickému programování. Vyvíjet systém v kontaktu s prostředím – za simulace, možnost zkoumat jeho chování a složitější systémy tvořit z těch jednodušších. Umožnit tvůrci přímý a co nejkonkrétnější přístup ke všem jeho prvkům a usnadnit mu tak jeho lepší pochopení.

Základními koncepty, kterými se tohoto snaží dosáhnout, jsou otevřenost a reflektivnost. Otevřeností se myslí možnost zkoumat a manipulovat se strukturou i aktuálním stavem všech prvků v systému. Tím získáme o systému poznatky, které můžeme dále využít k inkrementálnímu vývoji systému. Tuto interaktivní a inkrementální konstrukci modelů autor nazývá *exploratory modeling* [12].

Reflektivnost je důležitá k tomu, aby všechny operace, které nad modelem může provést člověk, mohl nad sebou provést i model sám. Formálněji to znamená, že jazyk, v tomto případě SmallDEVS, je zároveň svým metajazykem, tzn. že konstrukce jazyka jsme schopni tímto jazykem samotným popsat.

SmallDEVS je implementován ve Squeak Smalltalku, který je již sám o sobě na výše zmíněných konceptech postaven. Jakožto dynamický jazyk umožňuje, aby atomické modely byly za běhu modifikovány. U staticky typovaných jazyků jsou změny za běhu omezeny na ty strukturální ve sdružených modelech – při změně atomických modelů je nutné znovu zkompilovat příslušnou třídu [12]. To je omezující při tvorbě reflektivních, sebemodifikujících se a adaptabilních systémů.

Současné implementace DEVSu jsou postaveny na třídních objektově orientovaných jazycích jako např. C++ (Zeigler 1996), Java (Zeigler 1997) nebo Python (Bolduc, Vangheluwe 2002). I tento přístup má svá omezení – především při modifikacích konkretních, jednotlivých instancí. Zde je pak nutno vytvářet nové specializace tříd a těmi nahradit původní instance.

Prototypový přístup, inspirovaný jazykem Self (Ungar, Smith 1986), umožňuje provádět změny přímo v konkrétních instancích, protože ty nejsou spjaty s žádnou třídou, která

by předepisovala jejich vnitřní strukturu. Squeak Smalltalk, ve kterém jsou SmallDEVS implementovány je standardně založený na třídách, ale díky otevřenosti celého systému a sémantice založené na metaobjektovém protokolu, existuje i balík Prototypes, který myšlenku prototypových objektů realizuje.

3.1 Architektura

Protože výstupem této práce má být uživatelské rozhraní zpřístupňující entity SmallDEVSu, které reprezentují modely a simulace, je nezbytné prozkoumat jeho architekturu, abychom tyto poznatky mohli dále využít při návrhu samotného rozhraní.

3.1.1 Modely a simulace

V počátku hiearchie tříd reprezentující modely fakticky leží třída *BaseDEVS*, která definuje protokol pro práci s obecným DEVS modelem, tj. aspekty které jsou společné atomickým a sdruženým objektům. Jedná se především o vstupy a výstupy modelu, které jsou reprezentovány tzv. porty, přičemž každý port je charakterizován svým jménem. Na vstupní porty lze pomocí metody *poke:to:* zapsat objekt a z výstupních metodou *peekFrom:* objekt přečíst. Dále je zde definována dvojice metod umožňující nastavovat (*parent:*) a zjišťovat (*parent*) nadřazený model – to umožňuje tvořit hiearchické modely. Poslední důležitou schopností modelu poskytnout objekt zajišťující jeho simulaci.

Specializacemi *BaseDEVS* jsou potom třídy *AtomicDEVS*, resp. *CoupledDEVS*, reprezentující atomický model, resp. sdružený model. *AtomicDEVS* definuje zbytek protokolu dle formalismu DEVS – metody reprezentující přechodovou funkci vnitřního stavu (*intTransition*), přechodovou funkci externího vstupu (*extTransition*), výstupní funkci (*outputFnc*) a časovou postupovou funkci (*timeAdvance*). Nutno zdůraznit, že se jedná skutečně jen o definici protokolu – tyto metody postrádají konkrétní implementaci. Ta se očekává od specializací této třídy a zde je tudíž cesta pro modelování systémů pomocí třídního přístupu.

CoupledDEVS definuje protokol (a v tomto případě i implementuje funkcionalitu) pro práci s vnořenými modely. Jedná se o operace přídávání, odebírání, poskytování obsažených modelů a vytváření a rušení propojení (angl. coupling) mezi nimi.

Dvěmi výše zmíněnými třídami je umožněno modelovat systémy třídním způsobem – pomocí instancí tříd. Na další úrovni hiearchie jsou třídy *AtomicDEVSPrototype* a *CoupledDEVSPrototype*, které předefinovávají chování svých předků v tom smyslu, že vnitřně pracují s instancí třídy PrototypeObject. Ta umožňuje měnit strukturu a chování jednotlivých instancí a to tak, že definuje protokol pro editaci slotů (interních proměnných reprezentujících stav modelu) a metod konkrétního prototypu. Sdílené chování je zde realizováno tzv. traity[1]. V uživatelském rozhraní se pro ně občas používá označení „delegate“.Ty obsahují metody, případně konstanty, které mají být sdílené mezi různými prototypy. Tyto na

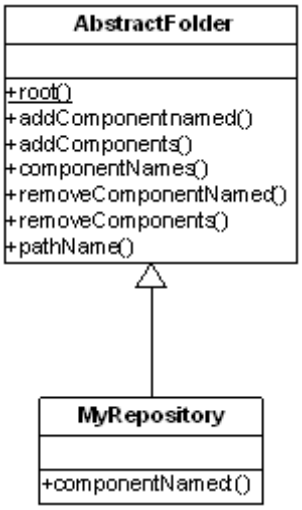
[1] Český překlad v tomto kontextu není ustálený, volně by se dalo přeložit jako charakteristický či povahový rys.

ně potom delegují zprávy, pokud je samy neumí zpracovat. Pro přesnost je nutno uvést, že se nejedná přímo o třídu *PrototypeObject*, ale specializaci, která umí uchovávat texty metod včetně komentářů a implementuje několik metod souvisejících se SmallDEVS.

Jak bylo již zmíněno, každý model poskytuje objekt umožňující jeho simulaci. Základem pro tyto objekty je typ *AbstractSimulation*, který definuje protokol pro spouštění, zastavování a změny parametrů simulace. Implementaci simulačního kroku, ale nechává na svých podtřídách. *DEVSRootSolverRT* tuto operaci nad DEVS modelem implementuje.

UML model jádra SmallDEVS najdete v příloze A.

3.1.2 MyRepository



Obrázek 3.1: UML model MyRepository

Squeak je tvořen sítí komunikujících objektů, ovšem modely resp. simulace tvoří uzavřené systémy, které, pokud nejsou z vnějšku referencovány, by byly zrušeny garbage collectorem. Tímto a také snadnou dostupností je opodstatněna existence MyRepository. Je to centrální uložiště postavené na konceptu adresářových souborových systémů, kde objekty jsou zanořeny ve stromové hiearchii adresářů a jsou jednoznačně identifikovatelné cestou v této struktuře.

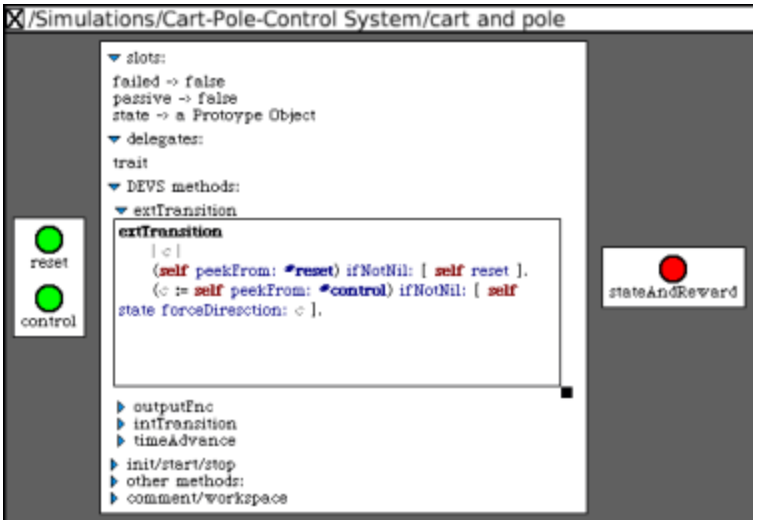
SmallDEVS tuto strukturu využívá pro uložení modelů, simulací a pomocných objektů (např. traitů). Ovšem My Repository je dostupný obecně z celého systému přes třídu *MyRepository*. Do této struktury je možno uložit jakýkoliv objekt, který implementuje protokol předepsaný právě třídou *MyRepository*, resp. jí nadřazenou *AbstractFolder*. Sdružené modely tento protokol implementují, aby vnořené modely modely byly snadno přítupné.

3.2 Grafické uživatelské rozhraní

Protože jedním z cílů navrhovaného webového rozhraní je přiblížit se nativnímu rozhraní, budou zde jednotlivé komponenty tohoto rozhraní popsány, aby poznatky takto získané mohly být dále využity.

Protože primární přístup k modelování systémů ve SmallDEVS je realizován přes prototypy (tedy beztřídně) je i grafické uživatelské rozhraní tímto směrem orientováno (třídní přístup není podporován). I toto rozhraní je do značné míry inspirováno jazykem a prostředím Self. Každý objekt tak může být, bez ohledu na situaci, vizualizován a manipulován pomocí nezávislé GUI komponenty dle svého typu. Uživatelské rozhraní má několik komponent, přičemž každá odpovídá nějaké komponentě z domény modelu. Konkrétně se jedná o prohlížeč MyRepository, inspektor atomický modelů, inspektor sdružených modelů a komponentu pro ovládání simulací.

3.2.1 Inspektor atomických modelů



Obrázek 3.2: Inspektor atomickych modelů

Inspektor atomických modelů zobrazuje a umožňuje editovat celkovou strukturu modelu. V centrální části je de facto inspektor prototypových objektů, který umožňuje editaci slotů (vnitřního stavu), delegátů (sdíleného chování) a metod prototypu, které jsou pro přehlednost rozděleny do kategorií:

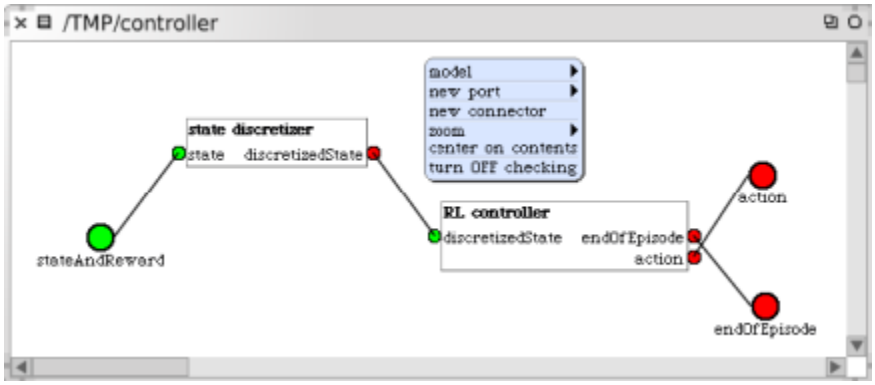
- **Metody DEVSu** – sem patří všechny funkce přímo spjaté s formalismem DEVS (*intTransition*, *extTransition*, *outputFnc*, *timeAdvance*).
- **Init/start/stop** – tři metody zajišťující inicializaci modelu a případné akce, které je nutné pro vést při spuštění/zastavení simulace.

- **Ostatní** – do této kategorie spadají metody, které nepatří ani do jedné z předcházejících. Zpravidla slouží jako pomocné metody.

Poslední položkou v centrální části je potom pracovní plocha umožňující vyhodnocování výrazu v kontextu daného modelu, případně jako prostor pro komentáře.

Po stranách okna jsou umístěny komponenty umožňující editaci vstupních a výstupních portů.

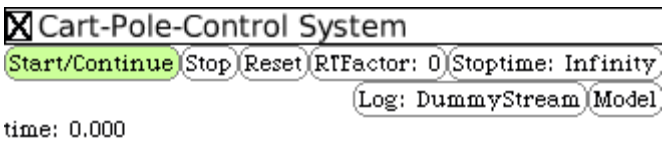
3.2.2 Inspektor sdružených modelů



Obrázek 3.3: Inspektor sdružených modelů

Hlavní část této komponenty tvoří vizuální podoba sdruženého modelu, tzn. systémy v tomto modelu obsažené, jejich propojení a vstupní/výstupní porty systému. Operace, které se provádějí pomocí kontextového menu, zahrnují přidávání a odebírání vnořených modelů, správu jejich propojení a editaci vstupních a výstupních portů celého systému.

3.2.3 Simulace



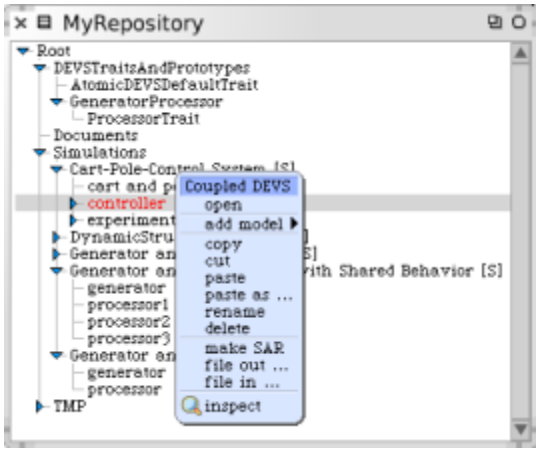
Obrázek 3.4: Samostatný konkrolér simulace

Ovládání simulací se nachazí ve SmallDEVS na několika místech. Jednak je dostupné v průzkumníku MyRepository přes kontextové menu. Další možností je panel na dolním okraji okna v případě, že modelem spjatým se simulací je sdružený model. Poslední možností je otevřít přes kontextové menu speciální kontrolní panel.

Základními operacemi nad simulací je její spuštění/zastavení a uvedení do výchozího stavu – reset. Dále se jedná o nastavení modelového času vzhledem k reálnému (*RTFactor*),

času ve kterém dojde k ukončení simulace a také proudu, který se bude používat pro logování událostí v průběhu simulace.

3.2.4 Průzkumník MyRepository



Obrázek 3.5: Průzkumník MyRepository

Jak bylo řečeno v části 3.1.2 My Repository je užíváno jako uložiště modelů a simulací postavené na konceptu hiearchického stromu objektů. S tím koresponduje i způsob zobrazení, který tvoří strom s rozbalitelnými uzly. S objekty je možno manipulovat pomocí kontextového menu. Základem je možnost objekt otevřít, tím se vytvoří nové okno, které daný objekt reprezentuje. Dalšími standardními operacemi dostupnými nad každým objektem jsou operace se schránkou (vyjmout/kopírovat/vložit, angl. cut/copy/paste) a možnost exportu objektu jako skriptu umožňující jeho zpětnou konstrukci včetně aktuálního stavu.

V případě simulací jsou z kontextového menu přímo dostupné operace pro její ovládání, které byly popsány v předchozí části.

Kapitola 4

Analýza a návrh řešení

V této kapitole budou podrobně rozebrány požadavky na funkčnost uživatelského rozhraní. Bude popsán framework pro tvorbu webových aplikací Seaside, který bude použit jako implementační platforma. Na základě těchto faktů a znalostí o architektuře jádra SmallDEVS bude navržena základní architektura uživatelského rozhraní.

4.1 Specifikace požadavků

Na základě předchozích kapitol je možno shrnout funkční požadavky na webové uživatelské rozhraní do následujících bodů:

- Možnost procházet My Repository v podobě stromu a poskytnout následující operace pro manipulaci s objekty v něm:
  - Otevření příslušného inspektoru, pokud ho daný objekt poskytuje.
  - Přejmenování vybraného objektu.
  - Zpřístupnění schránky pro manipulaci s aktuálně vybraným objektem. Konkrétně potom operace vyjmout (cut), kopírovat (copy) a vložit (paste).
  - Vytváření nových objektů – složek, simulací, atomických i sdružených modelů
  - Export objektu v serializované podobě (ve formě skriptu i XML)
  - Import objektu v serializované podobě a jeho umístění do aktuálně vybraného objektu.
- Umožnit inspekci a editaci atomických modelů. Zpřístupnění slotů, metod a portů.
- Umožnit inspekci a editaci sdružených modelů. Zpřístupnění propojení obsažených modelů a portů sdruženého modelu.
- Umožnit editaci traitů.
- Možnost importu a exportu objektů v serializované podobě.

4.2 Implementační platforma

Nejsofistikovanějším nástrojem pro tvorbu webových aplikací v prostředí Squeak je Seaside (A. Bryant 2001). Základním konceptem je udržování veškerého stavu aplikace na serveru. Ta je modelována pomocí komponent, které definují vzhled a chování části stránky. Z pohledu návrhového vzoru Model-View-Controller je komponenta dvojicí View-Controller. [9] Komponenty jsou zásadní rozdíl oproti např. PHP, kde aplikace je tvořena na úrovni celých stránek. Seaside takto umožňuje kontrolovat více toků řízení na stránce.

Stav sezení mezi serverem a klientem (a tím i jednotlivých komponent) je, jak bylo řečeno, udržován na serveru a to pomocí kontinuací. Kontinuace je reprezentace stavu výpočtu programu (většinou stavu zásobníku a ukazazatele na aktuálně vykonávanou instrukci) [13]. Identifikace aktuálního sezení a kontinuace je zakódována v URL[1]. Při požadavku se potom akce promítají do stavu reprezentovaného kontinuací v adrese. To umožňuje Seasidu se vyrovnat i s používáním tlačítka zpět či duplikací okna prohlížeče.

Tento přístup de facto odstiňuje programátora od limitujících vlastností HTTP protokolu a vytváří úroveň abstrakce pro vývoj srovnatelnou s vývojem desktopových aplikací.

HTML kód je generován pomocí konstrukcí Smalltalku, to umožňuje s ním zacházet jako s jakýmkoliv jiným kódem a budovat tak další úrovně abstrakce, které vedou k minimalizaci objemu kódu a k větší přehlednosti. Seaside generuje validní XHTML 1.0 Transitional kód, což zajišťuje relativně správné zobrazení v prohlížečích, které se blíží k implementaci této normy.

4.3 Základní koncepce

Celé rozhraní se dá rozdělit konceptuálně na dvě poměrně samostantné části. Prvním je Průzkumník MyRepository a druhou zobrazení detailů o objektech v něm, realizované pomocí inspektorů.

Průzkumník MyRepository musí, stejně jako jeho protějšek v nativnim GUI (3.2.4), umožnit procházení této struktury a zpřístupnit obecné operace nad objekty v ní. Tento ovládací prvek by měl tvořit jednu obrazovku uživatelského rozhraní.

Vyběrem objektu z MyRepository, který je schopen se prezentovat pomocí specializovaného inspektoru, uživatelské rozhraní přejde do režimu zobrazení tohoto inspektoru. To zda bude MyRepository v tuto chvíli přístupné nebo ne rozhodnou technické možnosti při *implementaci* vzhledu a funkcionality uživatelského rozhraní.

Je tedy potřeba navrhnout architekturu, která by tyto dvě části realizovala.

[1] Unified Resource Locator – adresa konkrétního zdroje v internetu

4.4 Návrh architektury

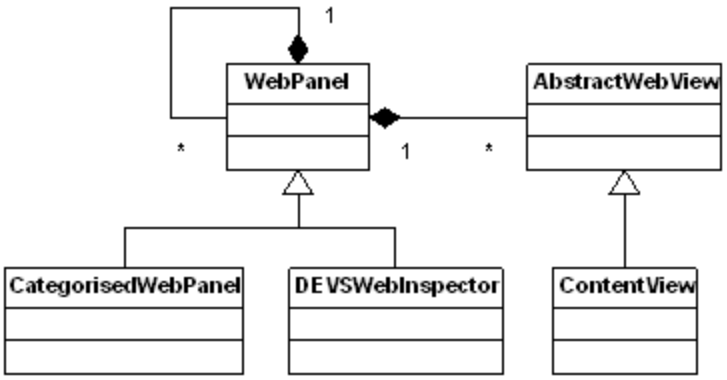
Uživatelské rozhraní budou tvořit specializace komponenty Seasidu, které by měli ideálně izomorfně pokrývat třídy objektů z jádra SmallDEVS – z hlediska uživatelského rozhraní doménu modelu. Postupně se zde pokusíme vyčlenit vlastnosti komponent uživatelského rozhraní a vytvořit tak hierarchii tříd od obecných až po ty konkrétní. Hlavním cílem je vytvořit sadu kompaktních a maximálně znovupoužitelných komponent tak, aby rozšiřování či úpravy systému byli co nejsnažší a aby celá architektura byla přehledná a snadno pochopitelná. Komponenty by měly být zapouzdřené a do značné míry autonomní tak, aby byli použitelné i sami o sobě, ne pouze ve spolupráci s ostatními komponentami navrhovaného UI.

Nejdříve bude proveden návrh inspektorů modelů a simulací a jejich vnitřní struktury. Poté bude následovat MyRepository a komponenty pro manipulaci s ním.

4.4.1 Reprezentace modelů a simulací

Model či simulace je kompozitní objekt sestávající z jedinečných vlastností a množin elementů určitých typů. U atomického modelů to jsou porty, sloty, odkazy na traity a metody. V případě sdružených modelů vnořené modely, porty, propojení. U simulací navíc ještě parametry a stav simulace.

Architektura inspektorů modelů resp. simulací by měla tento kompozitní rys reflektovat. Inspektor by proto měl být tvořen komponentami reprezentujícími jedinečné vlastnosti modelu a zobrazitelnými kolekcemi komponent – množin elementů stejných typů. Tím se nám vyčlenily dvě základní komponenty - obecná komponenta reprezentující konkrétní atomický prvek či aspekt modelu (nazvěme jí *AbtractWebView*) a kolekce obsahující další komponenty (nazvěme jí *WebPanel*).



Obrázek 4.1: Konceptuální UML model architektury inspektorů

AbstractWebView Existují dvě možná použití takové komponenty. Prvním je situace, kdy komponenta reprezentuje či přistupuje k modelu jako k celku. To je případ vizualizace

sdruženého modelu a ovládacího prvku simulací.

ContentView Druhým případem *AbstractWebView* je situace, kdy komponenta reprezentuje element *uvnitř* modelu, ne tedy model jako celek. Takový element je zpravidla reprezentován svým jménem. Pro tuto situaci je vhodné vytvořit specializaci, která bude tomuto účelu přizpůsobena. Pokud budeme postupně procházet nativní uživatelské rozhraní (kapitola 3.2) či jádro SmallDEVS (kapitola 3), narazíme na následující elementy, které mohou být reprezentovány konkrétními výskyty takové komponenty:

- Port (atomický i sdružený model)
- Slot (atomický model)
- Odkaz na trait (atomický model)
- Metoda (atomický model)
- Propojení (sdružený model)

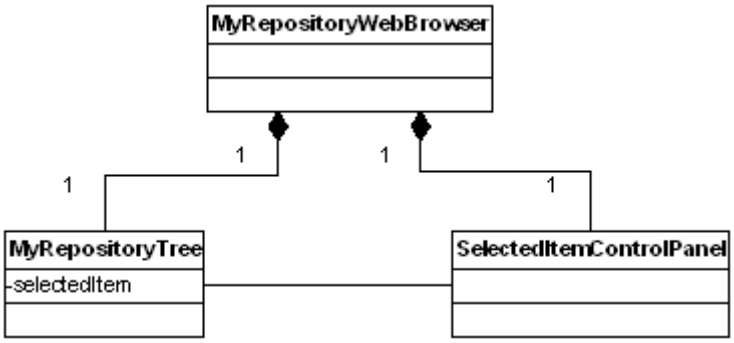
Tato komponenta, kromě toho, že patřičným způsobem zobrazí svůj obsah, musí umět s ním manipulovat. Základní operací je odebrání reprezentovaného elementu z modelu (a tím pádem i sebe z nadřazené kolekce). Obecně nahrazení jakéhokoliv elementu se dá potom realizovat jako odebrání a vytvoření nového elementu. Tím se dostáváme k vytváření, které by měla být schopna realizovat ta samá komponenta. Ta je standardně svázána s modelem a elementem modelu naležejícím. Pro vytváření nových elementů se nabízí varianta, kdy je komponenta volná vzhledem ke konkrétnímu elementu – je tedy nevázaná (angl. unbound). Při dodání patřičných informací bude na modelu daný element vytvořen a komponenta se na něj naváže. Z toho vyplývá, že *ContentView* se musí být schopen prezentovat a chovat dvěmi způsoby – jako nevázaný a jako vázáný.

Každá specializace by měla být schopna poskytnout pro daný model kolekci komponent reprezentující množinu všech výskytů daného elementu na modelu. Tato konstrukce bude výhodná při vytváření inspektorů, kde jde právě o to, poskytnout výčet všech elementů požadovaných typů.

WebPanel Kolekce komponent je, jak bylo uvedeno, zamýšlena k udržování určité množiny komponent (např. množiny reprezentující všechny metody na atomickém modelu), ale i na samotné inspektory modelů se můžeme podívat jako na množinu množin určitých elementů. Z toho vyplývá, že by mohla sloužit i jako základní pro realizaci samotných inspektorů – pouze je třeba je naplnit patřičným obsahem. V případě inspektoru atomických modelů (*AtomicWebInspector*) se bude jednat o kolekce komponent reprezentujících porty, sloty a metody. U inspektoru sdružených modelů (*CoupledWebInspecor*) potom vizuální podoba systému, kolekce portů a propojení. U inspektoru simulací (*RootSolverRTWebInspector*) to bude kontrolní panel simulace a inspektor kořenového modelu.

U atomických modelů je navíc požadavek na třídění metod do kategorií - jedna z nutných specializací je tedy kolekce, která bude umět třídit další komponenty dle názvu do kategorií. Tato komponenta budiž nazývána *CategorisedWebPanel*.

4.4.2 Průzkumník MyRepository



Obrázek 4.2: Konceptuální UML model architektury webového průzkumníka MyRepository

Jak bylo uvedeno v kapitole 3.1.2 je MyRepository stromová hierarchická struktura. Proto je ideální vizualizací, která nejlépe reflektuje povahu této struktury strom s rozbalitelnými uzly. Komponenta (nazývejme jí *MyRepositoryTree*), která bude takovýmto způsobem prezentovat MyRepository zároveň musí poskytovat informaci o aktuálně vybraném uzlu a také informovat o novém výběru. První potřebují operace nad aktuálně vybraným objektem popsané ve specifikacích požadavků (4.1). Druhé je zase potřeba k zobrazení inspektoru právě vybírané komponenty.

Druhou komponentou tvořící průzkumník (*MyRepositoryWebBrowser*) musí být panel poskytující operace nad právě vybraným objektem ve stromové struktuře. Nazývejme tuto komponentu *SelectedItemControlPanel*.

Kapitola 5

Implementace

V této kapitole budou popsány konkrétní kroky spojené s implementací v předchozí kapitole navržené architektury a ústupky či úpravy v ní, spojené s řešením konkrétních problémů. Na konci kapitoly bude čtenář obeznámen s přístupem k designu uživatelského rozhraní.

Názvy tříd přebírají názvy navržených komponent z předchozí kapitoly. Hierarchie tříd musela byt z důvodu nově se objevivších problémů rozšířena, jedná se ale pouze o rozšíření spektra specializací již navržených tříd.

5.1 AbstractWebView a jeho specializace

AbstractWebView plní funkci, jaká vyplynula z analýzy, tj. zobrazuje či operuje s aspektem či elementem modelu, se kterým je svázán. Implementace obsažená v této třídě ale zahrnuje pouze schopnost navázat a rozvázat vztah s nadřazenou komponentou (např. *WebPanel*. Další náležitosti jako definice zobrazované informace a její prezentace je pokryta v specializacích této třídy.

5.1.1 ContentView

ContentView rozšiřuje funkcionalitu *AbstractWebView* s orientací na element obsažený v modelu. Jak bylo popsáno v sekci 4.4.1, *ContentView* je buď svázaný nebo nesvázaný s konkrétním elementem, který je reprezentován tzv. selektorem. Selektor je název či jiná zjednodušená reprezentace elementu, pomocí kterého je možno se dostat k danému elementu.

Element je tedy buď svázaný nebo nesvázaný s elementem na modelu. Fakticky tento stav reprezentuje přítomnost či nepřítomnost selektoru. Pokud selektor není znám, komponenta se vykresluje specifickým způsobem – jako formulář očekávající zadaní údajů potřebných k vytvoření elementu. Při jejich zadání komponenta vytvoří na modelu element a nastaví u sebe odpovídající selektor. Tím dojde k navázání (angl. binding) komponenty.

Samozřejmě *ContentView* neposkytuje konkrétní implementaci přístupu k elementu či

jeho vytváření na modelu, pouze definuje tok řízení a množinu abstraktních metod[1], jejichž implementace se očekává od jeho specializací.

Každá specializace *ContentView* umožňuje pomocí třídní metody získat pro daný model množinu komponent, která reprezentuje množinu elementů příslušného typu vyskytující se na modelu. To je velice uzitečná konstrukce, která bude využita při vytváření inspektorů modelů, kde jde právě o to zobrazit *všechny* elementy požadovaných typů.

V následující části bude popsána komponenta reprezentující propojení modelů, která má některá specifika.

5.1.2 CouplingView

CouplingView slouží k reprezentaci propojení ve sdruženém modelu. V nativním GUI pro sdružené modely se propojení ovládají pomocí kontextového menu ve vizuální reprezentaci modelu (viz. 3.2.2). To ale při použití standardizovaných technologií pro tvorbu webu není možné.

Tato komponenta zobrazuje propojení ve formátu *zdroj : port — cíl : port*. Vytvoření nového spojení probíhá ve dvou fázích. V první dojde k vyběru zdrojové a cílové komponenty. Ve druhé fázi jsou nabídnuty relevantní porty u daných komponent. Po jejich vyběru dojde k vytvoření spojení.

5.2 WebPanel a realizace inspektorů modelů a simulací

Jak bylo navrženo, komponenty všech inspektorů vycházejí z třídy *WebPanel*. Ta implementuje především protokol pro přidávání a odebírání komponent, který odpovídá protokolu standardní kolekce prostředí Squeak. Při přidávání komponenty do panelu je, pokud taková komponenta implementuje příslušný protokol, informována o nově vzniklém vztahu. To může být užitečné např. pokud si obsazená komponenta přeje být z panelu odebrána.

Každý panel má své jméno, to z toho důvodu, aby obsah takového panelu mohl být rozpoznán podle tohoto názvu. Vykreslování panelu je rozděleno do dvou fází – záhlaví s názvem a samotný obsah. Ukázalo se toťiž jako účelné, aby takový panel mohl být sbalen pouze do svého záhlaví a nezabíral tak zbytečně mnoho místa, pokud není momentálně používán.

5.2.1 CategorisedWebPanel

Z důvodu požadavku zobrazení metod v kategoriích (viz. 3.2.1) bylo potřeba implementovat specializaci panelu, který by takovou kategorizaci umožňoval.

Zvolené řešení kategorizuje komponenty na základě jmen přidávaných komponent. Kategorie je vytvořena se svým jménem a názvy komponent, které do této kategorie spadají.

[1] Smalltalk syntakticky abstraktní metody nemá, je možno ale vyvolat vyjímku deklarující nepřítomnost redefinice metody v odvozené třídě.

Při vkládání komponenty s odpovídajícím jménem je tato zařazena na příslušné místo.

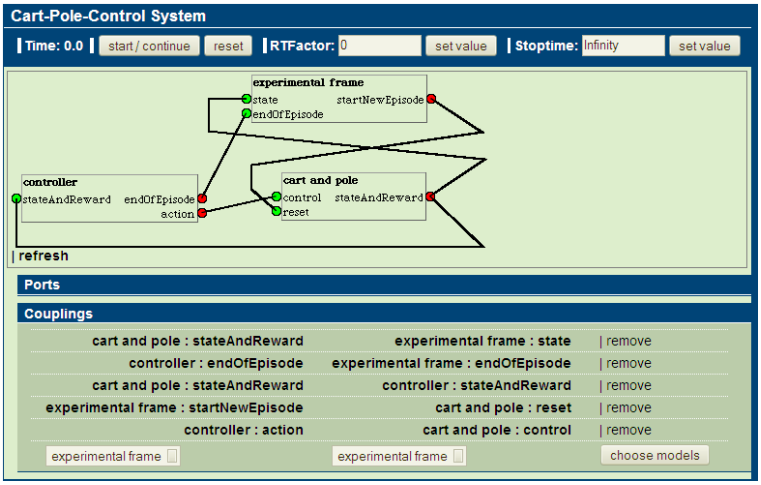
Kategorie jsou reprezentovány vnořenými panely. Při renderování (generování HTML kódu) jsou první kategorie a po nich teprve komponenty, které v panelu do žádné nepatří.

5.2.2 TableWebPanel

WebPanel vykresluje obsažené komponenty pomocí HTML elementu *DIV*. To je ale pro prezentaci některých elementů modelu (např. slotů u atomických modelů) nevhodné. Jedná se o elementy, které obsahují menší množství informací, které je z důvodu funkčního designu vhodné patřičně zarovnat. Pomocí CSS ale není jednoduchým způsobem možné dosáhnout dobrého formátování. Proto vznikl *TableWebPanel*, který pouze předefinovává způsob vykreslování – do tabulky.

Od komponent reprezentujících jednotlivé elementy ale očekává, že se budou vykreslovat jako řádky tabulky. Narušuje to sice jednotnost způsobu vykreslování, ale z hlediska rozvržení (angl. layout) je to řešení odůvodněné. Pokud by bylo cílem zachovat jednotnost, bylo by možné na *AbstractWebView* definovat atribut určující způsob vykreslování. Za stávající situace by ale tato úprava vedla pouze k jednotnosti chování komponent bez dalších přínosů.

5.2.3 Inspektoři modelů a simulací



Obrázek 5.1: Příklad inspektora – inspektor simulace sdruženého modelu.

Všechny komponenty, které slouží jako inspektoři jsou specializacemi *WebPanelu*. Při vytváření nad dodaným modelem (ve smyslu z domény modelu) se inicializují – naplní komponentami reprezentujícími příslušné aspekty a elementy. Jedná se buď o specializované komponenty nebo panely naplněné komponentami reprezentujícími všechny výskyty elementu určitého typu na modelu.

Inspektor atomických modelů je reprezentován třídou *AtomicWebInspector*. Ta se při vytvoření naplní následujícími složkami:

- Panel reprezentující všechny porty – vstupní i výstupní. Tento panel ve skutečnosti obsahuje další dva panely. Jeden je naplněn instancemi třídy *InputPortView*, druhý instancemi *OutputPortView*.
- Tabulkovým panelem s reprezentací všech slotů (instance *SlotView*).
- Kategorizovaným panelem reprezentujícím všechny metody. Metodu u atomického modelu reprezentuje třída *MethodView*.

Inspektor sdružených modelů tvoří třída *CoupledWebInspector*. Při vytváření nad modelem se inicializuje následujícím obsahem:

- Instance třídy *CoupledGraphView*, jejíž výstup je vizualizací sdruženého modelu v podobě grafu s porty a subsystémy. Je to tentýž výstup jako v nativním GUI, pouze přetransformován do obrázku. Není tudíž možné s jeho obsahem jakkoliv manipulovat.
- Panel s porty, který je identický s panelem u atomického modelu.
- Panel s propojeními. Propojení je reprezentováno třídou *CouplingView* (viz. 5.1.2)

Inspektor simulací reprezentuje třída *RootSolverWebInspector*. Při vytvoření se inicializuje pouze následujícími dvěma složkami:

- Ovládací panel simulací (*SimulationControlPanel*), který zobrazuje informace a ovládací prvky týkající se simulace. Konkrétně se jedná o aktuální čas simulace, možnosti jejího zastavení či spuštění (podle toho zda simulace běží nebo je zastavena). Dále je možno měnit poměr rychlosti běhu simulace k reálnému času (*RTFactor*) a konečný čas běhu simulace (*Stoptime*).
- Inspektor kořenového modelu simulace – podle typu inspektor atomických nebo sdružených modelů.

Pro zobrazení traitů byl též vytvořen inspektor, který se naplní pouze jediným dalším panelem, který obsahuje reprezentace metod v něm. Protože se ale k metodám traitů momentálně přistupuje jinak (*methodSourceAt:*) než k metodám u atomických modelů (*MethodSourceFor:*), je zde metoda zastoupena třídou *TraitMethodView*, což je specializace *MethodView*, zmíněné právě u atomických modelů.

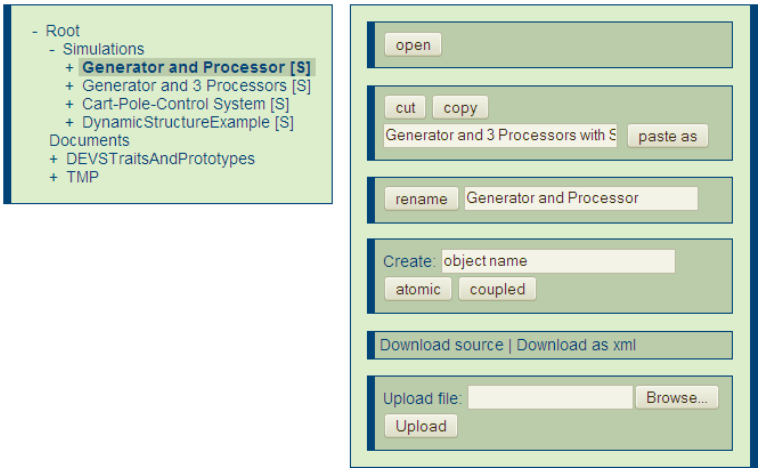
Entity jádra SmallDEVS inspektory poskytují metodou *webView*.

5.3 MyRepositoryWebBrowser

Třída *MyRepositoryBrowser* je zastřešující komponentou tvořící průzkumník MyRepository ve smyslu, jak byl popsán v části 4.4.2. Je složen ze dvou komponent – *MyRepositoryTree* a *SelectedItemControlPanel*.

Strom je založen na standardní komponentě Seasidu *WATree*, která umožňuje velice flexibilně přizpůsobení v chování a prakticky neklade nároky na konkrétní implementaci stromové struktury. Strom slouží k procházení hieararchické struktury, přičemž aktuálně vybraná položka je zvýrazněna. Jednotlivé uzly, pokud obsahují další, je možno rozbalovat.

MyRepositoryWebBrowser umí zacházet s MyRepository a umozňuje vyvolat nad objekty v něm různé akce. Pro zobrazení inspektorů u komponent, které toto umozňují, je potřeba dodat referenci na komponentu, která toto zobrazení zajistí. Tento vztah je možno definovat při vytváření průzkumníka.



Obrázek 5.2: Webová podoba MyRepository (vpravo panel s operacemi)

5.3.1 SelectedItemControlPanel

Druhá komponenta *SelectedItemControlPanel* poskytuje operace nad vybraným objektem. Je realizována jako *WebPanel*, naplněný skupinami akcí. Každá skupina je implementována samostatně, jednak kvůli logickému seskupení souvisejících akcí ve struktuře kódu ale také při samotném výstupu v podobě HTML stránky. Bázová třída všech skupin poskytuje operace, které mohou potenciálně využít všechny specializace. Jedná se např. o implementaci shránky, zjisťování existence objektů se zadaným jménem ve vybrané lokaci apod. Konkrétně se potom jedná o následující skupiny:

- *OpenActions* – Je zobrazena pouze pokud vybraná komponenta poskytuje inspektor. Při výběru dojde k otevření tohoto inspektoru a k minimalizaci průzkumníka MyRepository.

- *RenameActions* – Umožní přejmenovat vybraný objekt. Dojde k zachování všech existujících vztahů (např. u vnořeného modelu).
- *ClipboardActions* – zpřístupňuje operace se schránkou (vyjmout, kopírovat, vložit). Při pokusu vložit objekt do lokace, kde již objekt se stejným jménem existuje, umožní vložit objekt pod jiným jménem.
- *CreationActions* – Umožňuje vytvářet objekty ve vybraném uzlu. Jedná se o složky, simulace, a sdružené i atomické modely. Umožní vytvářet jednotlivé objekty pouze tam, kde je to legální (např. nelze nic vytvořit v atomickém modelu).
- *ExportActions* – Zpřístupňuje ke stažení serializovanou podobu vybraného objektu ve formě skriptu a ve formátu XML.
- *ImportActions* – Umozňuje nahrát soubor obsahující serializovanou podobu objektu (skript nebo XML). Upozorní pokud ze souboru nejde objekt zkonstruovat a zajistí případnou změnu jména, pokud objekt se stejným jménem v dané lokaci již existuje.

5.4 Kořenová komponenta

Webová aplikace psaná s použitím frameworku Seaside musí mít jednu kořenovou komponentu. V případě uživatelského rozhraní SmallDEVS je touto komponentou *SmallDEVSWeb*. Ta po vytvoření obsahuje průzkumník MyRepository u kterého definuje sebe jako komponentu, které zajišťuje zobrazení detailů o komponentách vybraných v průzkumníku – těch které poskytují inspektor pro své zobrazení.

5.5 Design rozrhaní a jeho realizace

Z hlediska vzhledu je celé rozhraní navrženo minimalisticky, s důrazem na maximální funkcionalitu a s orientací na obsah. Použity jsou odstíny dvou barev – zelené a modré – s občasným použitím bílé jako zvýrazňujícího prvku. Je použita například jako barva písma v záhlaví panelů, jejichž pozadí je modré. Při výskytu kurzoru myši nad většinou prvků tyto mění barvu svého pozadí. Tím dochází k optickému zvyraznění, které pomáhá orientaci v uživatelském rozhraní zvláště pak za přítomnosti mnoha ovládacích prvků.

Vzhled celého rozhraní je definován pomocí kaskádových stylů (CSS). Každá komponenta definuje svoje styly sama. Například panel jich definuje více, protože umožňuje třídu kaskádového stylu nastavovat dynamicky (např. kvůli odlišení pozadí vnořených panelů). Styly jsou definovány v metodě *style* jako řetězec, který Seaside automaticky, pokud je komponenta na aktuální stránce použita, zařadí do stránky formou okazu na styl.

Styly, které se nevztahují k žádné konkrétní komponentě jsou potom definovány v tzv. knihovně stylů (třída *SmallDEVSStyleLibrary*). Jedná větsinou o styly definující vzhled standardních elementů HTML.

K vytvoření uživatelského rozhraní není použit žádný obrázek a je tak minimalizována velikost jednotlivých stránek. Zvažováno bylo např. použití ikon místo tlačítek, které spouštějí různé operace. Navrhnutí takových ikon, aby jasně komunikovali svůj účel není jednoduché a špatný design by mohl vést k celkově horší orientaci v uživatelském rozhraní [5], což je samozřejmě nežádoucí.

5.5.1 MyRepository při zobrazení inspektora

Při přechodu mezi zobrazením MyRepository a otevřením inspektoru byla zvolena cesta minimalizace MyRepository do podoby tlačítka v levém horním rohu. Bez této úpravy docházelo při větší šířce inspektoru ke zlomení rozvržení a tím k degradaci celého rozhraní. Pro návrat do pohledu na MyRepository stačí kliknout na zmíněné tlačítko.

Touto úpravou nedošlo k žádnému omezení funkcionality, protože mezi průzkumníkem MyRepository a otevřeným inspektorem nelze realizovat žádnou přímou interakci.

Kapitola 6

Závěr

V rámci řešení se podařilo realizovat všechny požadavky specifikované v části 4.1. Rozhraní zpřístupňuje veškeré informace, které se tykají existujících modelů a simulací v prostředí SmallDEVS a umožňuje zkoumat aktuální stav systémů na úrovni formalismu DEVS.

Toto rozhraní může být velice užitečným doplňkem aplikačního uživatelského rozhraní u konkrétních aplikací při jejich reálném nasazení. Nepostradatelným nástrojem se potom stává pro inspekci a konfiguraci systémů nasazených na vestavěných systémech.

Navržená architektura se ukázala jako dostatečně modulární, znovupoužitelná a poměrně snadno implementovatelná. Komponenta *WebPanel* a její specializace dokonce překračují rámec tohoto konkrétního projektu, protože jsou obecně použitelné v jakékoliv webové aplikaci jako vizuální kolekce – jejich design konfigurovatelný a funkčnost upravitelná.

6.1 Širší perspektiva

Tato práce, resp. její programový výstup, je jedním z možných využití architektury umožňující na simulacích založený evoluční design systémů. Ta poskytuje reflektivní rozhraní k frameworku SmallDEVS, potažmo k modelům a podporuje interaktivní a programovou modifikaci modelů a simulací. Kromě webového uživatelského rozhraní, nativního GUI a skriptů, které již SmallDEVS zpřístupňují, je další snahou zpřístupnit ho jako službu v SOA[1]. To by umožňilo propojení a spolupráci s dalšími simulátory či tvorbu klientů v podobě desktopových aplikací.

6.2 Možnosti dalšího vývoje

V této části bude popsáno několik možných rozšíření systému webového uživatelského rozhraní a jeho funkcionality.

[1] Service Oriented Architecture

6.2.1 Zvýšení interaktivity

Je možno uvažovat implementaci interaktivní vizualizace editoru sdružených modelů s obdobnout funkcionalitou, jakou nabízí nativní GUI. Implementace by pravděpodobně mohla být realizována spoluprácí klientského JavaScriptu a CSS stylů. Ještě vyšší interaktivity a pravděpodobně za menšího úsílí by se dalo dosáhnout využitím technologie třetí strany, např. Adobe Flash.

Interaktivita by se dala dále zvýšit využitím technologie AJAX[2]. Nebylo by potom nutné s každou akcí uživatele načítat celou novou stránku, ale obnovovalo by se pouze příslušné uzel v objektovém modelu dokumentu (DOM). Konkrétně by se takového chování dal využít při přidávání a odebíraní elementů v inspektorech, ale i při operacích se schránkou v MyRepository. Dále např. při rozbalování uzlů MyRepository.

6.2.2 Adresovatelnost MyRepostiory

Momentálně je možno se k modelům či jejich serializovaným podobám dostat pouze postupnou navigací v MyRepository. Pro snadnou dostupnost v případě složitých modelů by bylo vhodné implementovat možnost adresovat modely, případně i serializované podoby, jedinečnou URL adresou. Tím by došlo k užší intergraci MyRepository a webu.

Vhodný formát takové adresy modelu je:

```
http://<zařízení>/<umístění rozhraní>/<cesta v MyRepository>/<jméno modelu>
```

Adresa serializovaných podob by potom byla tvořena přidáním koncovky `.st` pro formu skriptu nebo `.xml` pro XML.

Seaside neumožňuje používání takových adres pro všechny stránky, protože je komponentově orientovaný a adresa je používáná k identifikaci sezení a aktuálního stavu (viz. 4.2). Je možno ale zpracovat dotazovanou URL a inicializovat tak úvodní stránku dle požadavku, dále se již adresa bude řídit zákonitostmi Seasidu – z hlediska uložiště modelů tedy již nebude mít žádnou sémantiku.

6.2.3 Bezpečnost

Z hlediska reálného nasazení je, pokud by SmallDEVS běžel ve veřejně dostupném prostoru, nezbytné implementovat do systému bezpečností prvky. Základním takovým prvek je autentizace uživatelů – přihlašování do webového uživatelského rozhraní. K tomu je možno použít dekorační komponentu *WABasicAuthentication*, která využívá chybového kódu 401 HTTP prokololu (neautorizovaný přístup), na který klienti reagují výzvou o zadání přihl. jména a hesla. Ty jsou poté zpět zasílány v hlavičce požadavků. Touto dekorační komponentou je potom možné některé komponenty obalit a vynutit tak přihlášení před jejich použitím. Do autentizace nebude komponenta zobrazena.

[2] Asynchronous Javascript and XML

Další možností je implementace vlastní přihlašovací komponenty, která bude zobrazena při vstupu do systému. Do úspěšné autentizace nebude uživatel puštěn do samotného webového uživatelského rozhraní.

Informace o uživatelích by mohly být uloženy ve speciální složce v MyRepository.

Literatura

[1] Guru A., Savory P., and Williams R. *A web-based interface for storing and executing simulation models.* Proceedings of the 2000 Winter Simulation Conference, 2000.

[2] Zeigler B.P. *Object oriented simulation with hierarchical, modular Models.* Academic Press, 1990.

[3] Bindel D. and Eaton P.R. *M & MEMS: A web-based interface for Sugar MEMS simulator hosted on Millenium cluster.* University of California at Berkeley, 2000.

[4] Sarjoughian H.S. and Cellier F.E. *Discrete Event Modeling and simulation technologies: A tapestry of systems and AI-Based theories and methodologies.* Springer-Verlag New York Inc., 2001.

[5] Mullet K. and Sano D. *Designing visual interfaces: Communication oriented techniques.* Prentice Hall, 1995.

[6] Dong-Yup Lee, Choamun Yun, Ayoun Cho, Bo Keyeng Hou, Sunwon Park, and Sang Yup Lee. *WebCell: A web-based environment for kinetic modeling and dynamic simulation of cellular networks.* Bioinformatics Advance Access, 2006.

[7] Fishwick P.A. *Web-Based Simulation.* Proceedings of the 1997 Winter Simulation Conference, 1997.

[8] Chandrasekaran S., Silver G., Miller J.A., Cardoso J., and Sheth A.P. *Web service technologies and their synergy with simulation.* Proceedings of the 2002 Winter Simulation Conference, 2002.

[9] Ducasse S., Lienhard A., and Renggli L. *Seaside — A multiple control flow web application framework.* ESUG Conference, 2004.

[10] Berners-Lee T. and Cailliau R. *WorldWideWeb: Proposal for a HyperText Project.* [online; navštíveno 28. 4. 2007] `http://www.w3.org/Proposal.html`, 1990.

[11] Bubnik V., Schirrwagen J., and Sevcenko M. *DynCAD: Web-based schematic editor.* [online; navštíveno 7. 5. 2007] `http://virtual.cvut.cz/dyncad/help/`.

[12] Janoušek V. and Kironský E. *Exploratory modeling with SmallDEVS*. Vysoké učení technické v Brně, 2004.

[13] Wikipedia. *Continuation — Wikipedia, The Free Encyclopedia*, 2007. [online; navštíveno 26. 4. 2007] `http://en.wikipedia.org/wiki/Continuation`.

[14] Rábová Z., Zendulka J., Češka M., Peringer P., and Janoušek V. *Modelování a simulace*. Vysoké učení technické v Brně, 1992.

Seznam použitých zkratek a symbolu

AJAX – Asynchronous Javascript and XML

DEVS – Discrete Event System Specification

CSS – Cascading Style Sheets

GUI – Graphical User Interface

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

PHP – PHP Hypertext Preprocessor

SOA – Service Oriented Architecture

UI – User Interface

UML – Unified Modeling Language

URL – Unified Resource Locator

WPF/E – Windows Presentation Foundation / Everywhere

WWW – World Wide Web

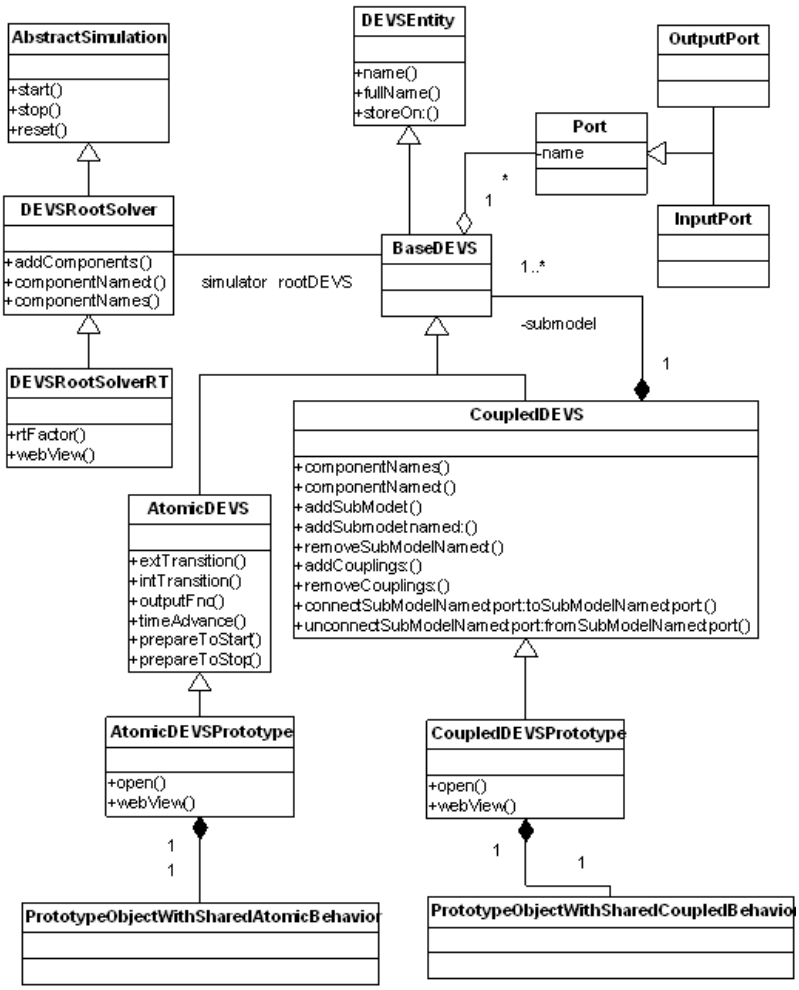
XHTML – Extensible Hypertext Markup Language

XML – Extensible Markup Language

Seznam příloh

Příloha A

Model jádra SmallDEVS



Obrázek A.1: UML model jádra SmallDEVS