VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

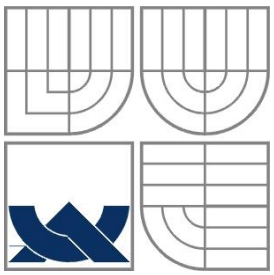
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELIGENT SYSTEMS

MODELEM ŘÍZENÝ NÁVRH KONFERENČNÍHO SYSTÉMU

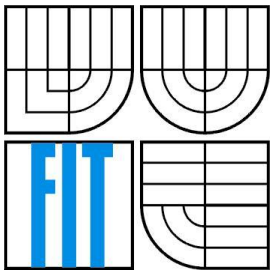
SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE Bc. MATĚJ CAHA
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELIGENT SYSTEMS

MODELEM ŘÍZENÝ NÁVRH KONFERENČNÍHO SYSTÉMU

MODEL BASED DESIGN OF THE CONFERENCE SYSTEM

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE
AUTHOR

Bc. MATĚJ CAHA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK KOČÍ, Ph.D.

BRNO 2013

Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém jazyce.

Abstract

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém jazyce, oddělená čárkami.

Keywords

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Citace

Caha Matěj: Modelem řízený návrh konferenčního systému, semestrální projekt, Brno, FIT VUT v Brně, 2013

Modelem řízený návrh konferenčního systému

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením...
Další informace mi poskytli...
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Matěj Caha
9.1.2013

Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

Obsah

1. Úvod

nepiste o predmetech, tady se venujete DP - OK

citace se uvadeji do [] - vyresim az na DP

Tato studie vznikla jako projekt do předmětu SIN na VUT FIT v Brně. Smyslem této práce je v rozumné míře shrnout dosavadní metodiku modelem řízeného návrhu a zajít až k těm aktuálně vyvíjeným. Popíši zde historii návrhu systémů, dále princip metodiky MDA (Model Driven Architecture) dle specifikace skupiny OMG (1), poté navážu na metodiku a technologii vyvíjenou na naší fakultě výzkumnou skupinou modelování a simulace systémů při ústavu inteligentních systémů (více v (2)).

Softwarové inženýrství se již spoustu let zabývá vývojem software a řešením problémů, které při vývoji vznikají, společně s vyvíjením nových technik k usnadnění a urychlení vývoje systémů. V současné době existuje mnoho metod vývoje SW, ze kterých si můžeme připomenout klasický Vodopádový model, nebo pokročilejší Spirálový model, dále z těch pokročilejších to je třeba inkrementální iterační model vývoje, nebo unifikovaný RUP a v neposlední řadě zmíním agilní metodologie jako extrémní programování, nebo SCRUM.

Všechny zmíněné metodiky ve svých fázích vývoje více či méně využívají modely, jako reprezentaci systému, nebo jeho částí. Modelování je pro nás důležité, neboť nám umožňuje postihnout ony vlastnosti systému, které nás zajímají. Pravděpodobně nejznámější modelovací prostředek je UML (Unified Modeling Language, (3)), který je však pouze informativní (vizualizační), lépe řečeno, neumožňuje ve své základní podobě provádění vytvořených modelů.

V oblasti softwarového inženýrství došlo v poslední době k rozvoji přístupu, který využívá modelování ve smyslu simulovatelných modelů a to v takém rozsahu, že je lze považovat za programovací jazyk (4). Tento přístup dostál mnoha označení, z nichž některá jsou *Model-based design* (MBD), také *Model-driven design* (MDD) a *Model-driven architecture* (MDA), nebo trochu komplexnější název *Model and Simulation based design* (MSBD). Všechny ale v podstatě značí to, že vývoj počítačového systému je podložen proveditelným modelem, čili například *xUML* (Executable UML, viz. kap. 3.1), nebo jinými formalismy, jako *Petriho sítě*, případně *DEVS*, které jsou probrány v kapitole 4. Tento proveditelný model může být testován již v průběhu analýzy, což je nesporná výhoda oproti klasickým metodám vývoje, kdy se systém testoval až po fázi implementace. Fáze implementace u modelem řízených metod návrhu může být potom prováděna manuálně, kdy programátoři vytvářejí kód na základě odsimulovaného modelu systému, nebo také (polo)automaticky, kdy musí být prvkům modelu jasně stanoveny pravidla transformace na zdrojový kód. Tuto transformaci mohou IT analytici částečně uzpůsobovat svému vkusu pomocí zavedení *Metaúrovní*, jako například *Meta-Object Facility* (MOF) u UML. Tato problematika transformace modelu je částečně popsána v kapitole 3.

Kapitola 4 potom pojednává o významu modelování a simulace při vývoji systému a diskutuje výzkum probíhající na fakultě informačních technologií na VUT v Brně (viz. (4)).

v nazvu kapitoly neuvadejte zkratky - OK?

2. Historie - cesta k MBD

nepisete pribeh, ale technickou zpravu, budte ve formulaci myslenek trosku strizlivejsi

Na vývoj softwarového průmyslu se dá dívat jako na příběh o vrstvení abstrakcí. V počátku psaní počítačových programů, mluvíme-li o programech ukládaných do paměti počítače, kdy měly počítače paměť počítanou v kB a výkon udávaný v kHz existovala v podstatě pouze jedna vrstva abstrakce, která mapovala jedna k jedné instrukční sadu jazyka na přímé provádění strojového kódu (5). Podobně jako tento vývoj softwarového průmyslu je i cesta návrhu systémů dlážděna zvyšováním počtu abstrakčních vrstev a vývojem různých metodik k ulehčení, ale hlavně urychlení a tedy snížení ceny vývoje. Pojďme si tou cestou tedy projít od začátku. Následující řádky popisující historii návrhu systémů jsou volně převzaty z (5).

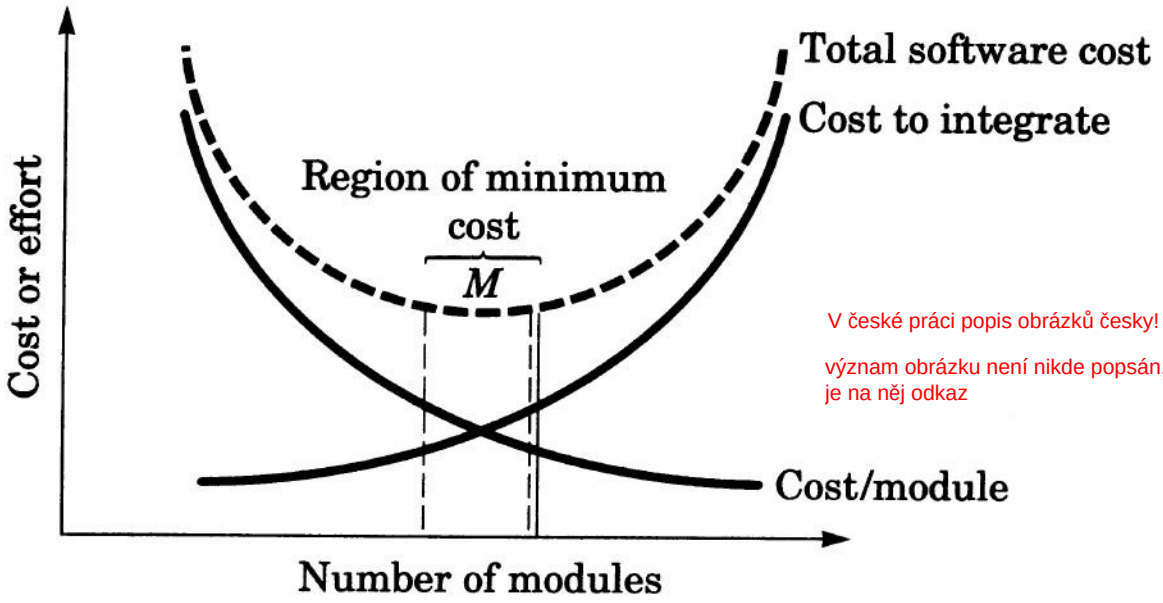
na pocatku ceho?

Jak jsem psal v úvodu této kapitoly, na počátku byly počítače a vývoj software dosti omezené. Každý programátor psal programy v podstatě ve strojovém kódu a vytvářel tak malé programy, které byly jednoduše srozumitelné jednomu člověku a každý si raději hrabal na svém písečku. Nicméně, počítače začaly být větší a výkonnější a společně s tím rostla i složitost vyvíjených systémů. S příchodem programovacích jazyků třetí generace (jako třeba C nebo C++) vzrostla abstrakce programování a programátoři se občas museli spojit do týmů a pracovat společně. Tím vznikla potřeba stanovit jistá pravidla psaní kódu v týmu. Program potřeboval být více strukturovaný, čímž na začátku 70~~-desátých~~ let 19. století vznikl trend zvaný *Strukturované Programování*. Tento trend však nepřinesl kýžený výsledek. co je kýžený výsledek? - OK přepracováno

Možná tedy nebyl problém v implementaci kódu, ale v dřívější fázi životního cyklu softwaru (dále SW). Edward Yourdon a Larry Constantine to takto napadlo a udělali tah směrem k *Strukturovanému Návrhu* SW. Navrhli dvě klíčové kritéria kvality návrhu: *propojení* a *koheze*. Obě dvě kritéria jsou míněny ve smyslu *nezávislosti*. Šlo jim tedy o to, aby při návrhu vznikly moduly, které jsou na sobě co nejméně závislé a to jak ve smyslu propojení, tak ve smyslu soudružnosti dat v modulu. Počet modulů je potřeba volit rozumně (viz Obrázek 2.1). Obě tyto vlastnosti jsou klíčové také pro modelem řízený návrh, ke kterému směřujeme. Dříve zmínění pánové také nabídli svétu návrhový formalismus zvaný *Diagram Struktury,* který návrhářům umožnil grafickou vizualizaci navrhovaného kódu.

Zaměření na dřívější etapu životního cyklu SW pokračovalo dále a vyústilo ve *Strukturovanou Analýzu*, která byla popularizovaná lidmi Tom DeMarco (1978) a Edward Yourdon (1989). Principem bylo oddělení specifikace problému od jeho řešení, což vedlo k vytvoření dvou primárních modelů. První, zvaný *Základní Model*, sloužil jako representace systému, bez zaměření na implementaci. Druhý, zvaný *Implementační Model*, byl funkční realizací základního modelu s důrazem na organizaci hardware, software a zdrojového kódu. Zde má základy princip MDA se svým platformě nezávislým a platformě specifickým modelem, ale k tomu se dostaneme v další kapitole. Při vytváření modelů v této etapě jsme se mohli setkat s abstrakcí typu *DFD* (Data Flow Diagram), *SD* (State diagram) a *ERD* (Entity Relationchip Diagram), které dnes v mírně upravené podobě potkáváme v modelovacím jazyce *UML* (Unified Modeling Language). V této době vznikl také *událostmi řízený* přístup, o který se zasloužili pánové Ward a Mellor (1985), kteří řekli, že systém nemusí být rozdělen s*hora-dolů,* ale *zvenku-dovnitř*. Jejich pohled byl založen na porozumění

prostředí ve kterém systém běží a vytvoření specifikace chování tak aby vyhověla tomuto prostředí. Tento přístup se dá připodobnit k *Use-Case* diagramům, které opět najdeme v *UML*.



V české práci popis obrázků česky!

význam obrázku není nikde popsán, pouze je na něj odkaz

Obrázek 2.1 Závislost ceny SW na počtu modulů. Převzato z (6).

Mezitím, co byly vyvíjeny strukturované metody, si návrháři uvědomili, že mohou existovat i jiné způsoby rozdělení. Přišli na to, že objekty v systému jsou mnohem více stabilní, než funkce nad nimi prováděné. Vznikly tedy *Objektově Orientované* (OO) *Metody*, které se snaží přizpůsobit reálnému světu tím, že implementují objekty, které mají své vlastnosti a funkce, neboli metody, které mohou provádět. Knihy o OO programování autorů Booch (1986), Mayer (1988) a dalších byly následovány knihami o *OOD (Object-oriented Design)* a *OOA (Object-oriented Analysis)*. Tyto přístupy byly aplikovány hned v počátku životního cyklu SW, kde jsou prováděna nejdůležitější rozhodnutí ve formě specifikace chování a pravidel systému. MDA se ztotožňuje s touto aplikací a zavádí ji také do prvotní fáze vývoje.

Objektově orientovaný přístup se stal populárním a lidé začali vymýšlet, jak by šel vývoj za pomocí objektů ještě zjednodušit. Booch v publikaci *Object Oriented Analysis and Design with Applications* (1993) detailně popsal metodu OO vývoje. Principy užívání *vzorů*, nebo *archetypů*, jako základ pro platformě specifickou implementaci, byly obhajovány i mnoha dalšími (např.: Beck, Shlaer, Mellor, Buschmann, Meunier, Coad). Celé to završila čtveřice Gamma, Helm, Johnson a Vlissides, kteří roku 1994 publikovali knihu *Design Patterns*, která je považována za nejznámější v oboru *návrhových vzorů*. Ivor Jacobson se svým přístupem řízeným *případy užití* (angl. Use-Case) roku 1992 pozvednul povědomí o důležitosti organizačních požadavků ve smyslu lehce modelovatelných a sledovatelných v průběhu životního cyklu. Roku 1991 vyvinul Dr. James Raumbaugh s kolektivem techniku zvanou *OMT (Object Management Technique)* ve které navrhli

soubor notací pro podporu vytváření modelů analýzy a návrhu. Tato technika se v 90. letech stala dominantní.

Shlaer a Mellor s jejich *rekurzivním návrhem* zdůraznili význam vytváření precizních *PIM* (Platform Indipendent Model), které mohou být systematicky, případně automaticky, překládány[1] až k vygenerování zdrojového kódu cílového systému. Dále prosadili nový přístup strukturování systému a to na základě tematických celků, čili sdružení tematicky shodných prvků do jednoho modulu zvaného *doména*. Výsledkem tohoto rozdělení systému jsou moduly vysoce nezávislé na ostatních, tak jak bylo požadováno již před 20~~-ti~~ lety při *Strukturovaném Návrhu* pány Yourdon a Constantine. Podobným směrem se vydal kolektiv autorů v čele s p. Bran Selic, kteří v roce 1994 představili metodu nazývanou *Real-time Object-oriented Modeling (ROOM)* zaměřenou na vývoj RT systémů s důrazem na definování rozhraní a komunikačních protokolů. Oba tyto přístupy umožňují grafické znázornění, které je dnes naštěstí standardizováno v UML.

Aktuální stav návrhu a vývoje systémů je považován za sjednocení těch nejlepších metod z historie a může být popsán následujícími vlastnostmi:

1. Rozdělení systému na domény, obsahující prvky/objekty zapadající do stejného tématu, které jsou téměř nepropojené a kohezní;
2. Oddělení platformě nezávislého chování od platformě specifického chování, jako rozšíření dříve definovaného oddělení *základního modelu* od *implementačního modelu*;
3. Definice vzorem řízeného mapování pro systematické vytváření *platformě specifických* modelů z *platformě nezávislých* modelů. Toto mapování může být manuální, ale je snaha o plnou automatizaci;
4. Používání abstraktního, ale úzce sémanticky definovaného formalismu *xUML* (executable UML), které zajišťuje modelům preciznost a možnost jejich testování/simulaci již v době modelování systému.

Tímto jsme prošli historií návrhu systémů a v další kapitole se z blízka podíváme na metodologii MDA.

[1] Ve smyslu překladu programu. tato poznámka je možná zbytečná

Metodika

3. MDA (Model Driven Architecture)

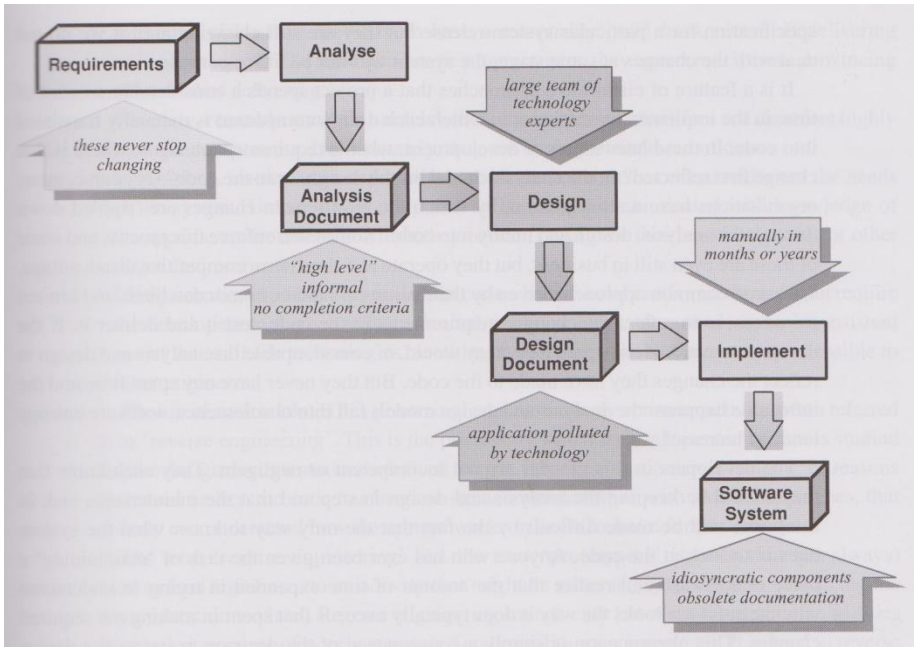
MDA je metodika vývoje SW založená na proveditelných modelech. Je vyvíjena skupinou OMG (1), která má pod sebou i mnoho dalších metodik a technologií zaměřených na interoperabilitu a portabilitu objektově-orientovaných aplikací, uveďme pár známějších např. UML, XML, XMI, COBRA, atd. Pro zájemce uvádím adresu na které najdou další informace: http://www.omg.org/. Dovolím si citovat doslovný souhrn konceptu MDA dle skupiny OMG z (5):

je uz v [1]

"MDA development focuses first on the functionality and behavior of distributed application or system, undistorted by idiosyncrasies of the technology or technologies in which it will be implemented. MDA divorces implementation details from business functions. Thus, it is not necessary to repeat the process of modeling an application od system's functionality and behavior each time a ew technology (e.g., XML/SOAP) comes along. Other architectures are generally tied to a particular technology. With MDA, functionality and behavior are modeled once and only once. Mapping from a PIM (Platform Independent Model) through a PSM (Platform Specific Model) to the supported MDA platforms will be implemented by tools, easing the task of supporting new or different technologies."

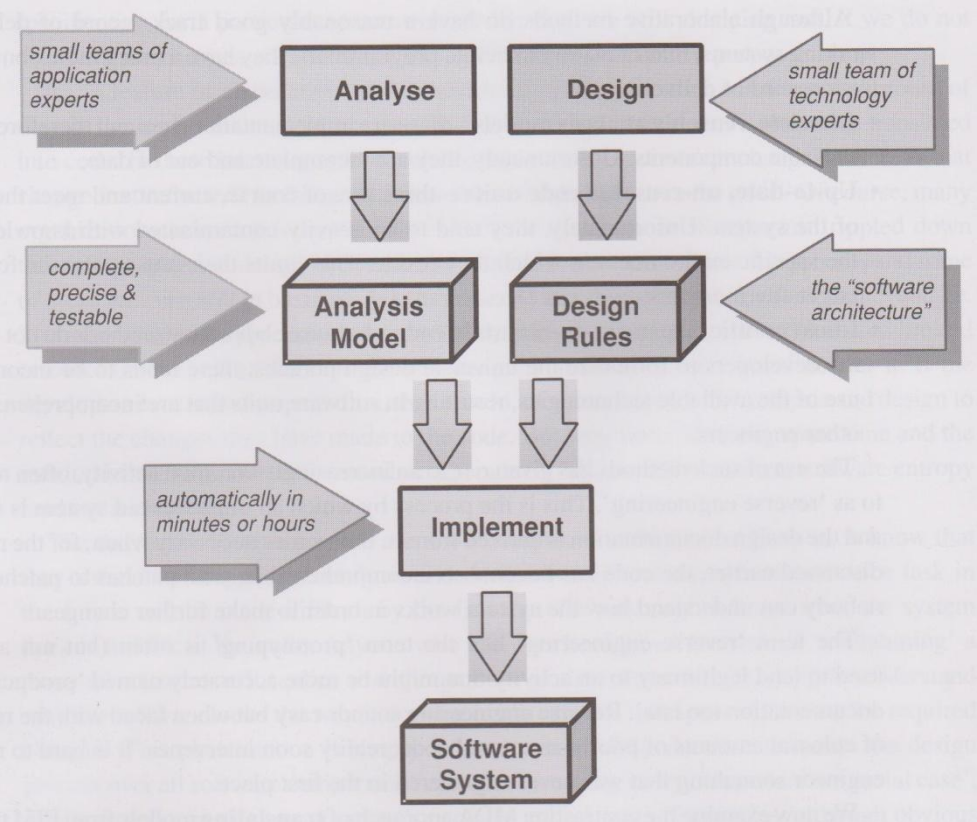
citace [5]

Citovaný odstavec volně přeložen říká, že MDA je technika vývoje, kdy je striktně oddělen model systému ve smyslu požadovaného chování a funkcionality od implementačních detailů závislých na použitých technologiích. Model funkcionalit a chování systému, zvaný *PIM* (viz. kap. 3.2), stačí vytvořit pouze jednou. Tento model je poté mapován pomocí překladových pravidel na *PSM* (viz. kap. 3.3) a konkrétní implementaci na dané platformě, zvanou *PSI* (Platform Specific Implementation). Pro jeden *PIM* tedy může existovat mnoho *PSM*, které definují transformaci modelu *PIM* na platformě závislou implementaci *PSI*. Nutno říct, že princip MDA spočívá také právě ve formě modelu *PIM*, který je proveditelný a tedy testovatelný v průběhu vývoje.



Obrázek 3.1 Diagram vývoje systému pomocí klasické vývojové metody Vodopád. *Elaborative procces*. Převzato z (5).

Klasické vývojové metody (např. Vodopád) stanoví problém (specifikace systému), který dále rozvíjí zaváděním návrhových a implementačních detailů, až dojdou k řešení, které představuje výsledek vývoje SW (viz Obrázek 3.1). Oproti tomu MDA stanoví problém ve formě modelu *PIM*, který je následně překládán a transformován až k výsledné implementaci systému, jak je vidět na Obrázek 3.2. (5)



Obrázek 3.2 Diagram vývoje systému pomocí MDA. *Translation-based procces.* Převzato z (5).

Dalším problémem klasických metod softwarového inženýrství je převod statického modelu systému (nejčastěji vytvořeného pomocí UML) na funkční implementaci SW. Tuto konverzi provádějí programátoři ve fázi implementace. V jejich práci jim mohou pomáhat různé (polo)automatizované transformace vybraných modelů, ze kterých lze získat kostru programu, či přímo kompletní kód nějakého modulu. Zde se objevuje jedno velké nebezpečí, které vzniká právě jednosměrností transformace (4). Přiznejme si, kdo při dodatečných úpravách vyvíjeného systému zanesl úpravy do všech artefaktů vývoje, čili analýzy, návrhu a samotné implementace? Je to velmi řídký jev. Tím vzniká nekonzistence vygenerovaného a upravovaného kódu vůči dříve navrhnutým modelům. U metodik řízených modelem se vždy upravuje model samotný, který je opět formálně verifikován a případně znovu generován do zdrojového kódu. (5)

Koncept MDA využívá, rozšiřuje a integruje většinu již existujících a používaných specifikací skupiny OMG, jako třeba UML (Unified Modeling Language), MOF (Meta-Object Facility), XML (Extensible Markup Language) a IDL (Interface Definition Language). (7)

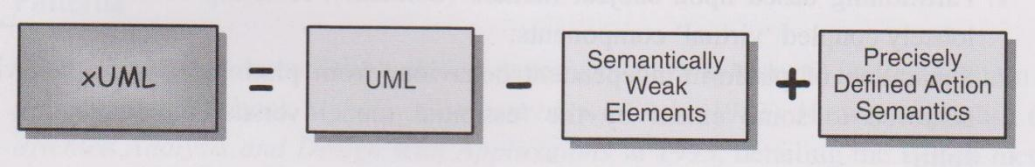
3.1 ~~UML vs. xUML~~ Srovnani UML a xUML

MDA se od klasických vývojových metod odlišuje také použitím modelovacích jazyků a modelu obecně. Zatímco u klasických metod je ve fázi analýzy a návrhu využíván model pouze coby reprezentativní a informativní formalismus, u MDA je tento model obdařen proveditelností, je tedy možné ho otestovat ihned po vytvoření. Tento přístup pomáhá odhalit chyby návrhu hned na počátku vývoje systému. (5)

UML (Unified Modeling Language) je modelovací jazyk, který poskytuje sjednocenou notaci pro reprezentaci různých aspektů objektově-orientovaných systémů. UML bylo také specifikováno skupinou OMG a jeho kompletní specifikace je dostupá zde (3). Náplní této studie však není rozbor jazyka UML, ~~proto to nechám na individuální aktivitě čtenáře.~~ Na dalších řádcích se předpokládá alespoň základní znalost UML.

Jak už bylo řečeno, UML je pouze notační nástroj. UML samotné nemá přesně definovanou sémantiku značek, umožňuje použití různých značek v jiném významu v jiných diagramech (např.: stavový model může být použit k popsání Use-Case diagramů, diagramů objektu, nebo podsystémů) (3). Tato vlastnost je pro MDA nežádoucí, modelování v MDA potřebuje striktní a precizní syntaxi a sémantiku notačních elementů, aby bylo u každého modelu přesně jasné co znamená a jak se má chovat. Toho bylo dosaženo vytvořením specifikace xUML (Executable UML), která je založena na jádře jazyka UML a doplněna Action Semantics (OMG, 2002), specifikující přesné chování používaných UML modelovacích elementů a odstraňuje mnoho nejasností elementů jazyka UML. (5)

Shrňme si tedy vztah xUML a UML následujícím obrázkem.



Obrázek 3.3 Vztah xUML a UML. Převzato z (5).

3.2 PIM (Platform Independed Model)

V MDA je pojem *platforma* používaná pro odkázání na technologické a návrhové detaily, které jsou irelevantní pro funkcionalitu SW. Platformě nezávislý model (PIM) je formální model popisující funkcionalitu systému, nebo jeho dané oblasti. PIM řeší koncepční otázky, ale nezohledňuje, jak bude vypadat technologická implementace. Pomocí dříve popsaného xUML tak můžeme vytvořit formální, precizní a proveditelný model PIM. Testování vlastností systému na úrovni PIM modelu přináší urychlení odladění případných chyb. K PIM modelu se často přidává PIM Metamodel do kterého mohou IT analytici vložit instrukce k překladu na PSM, aby výsledná implementace splňovala jejich výkonnostní požadavky. (5)(7)

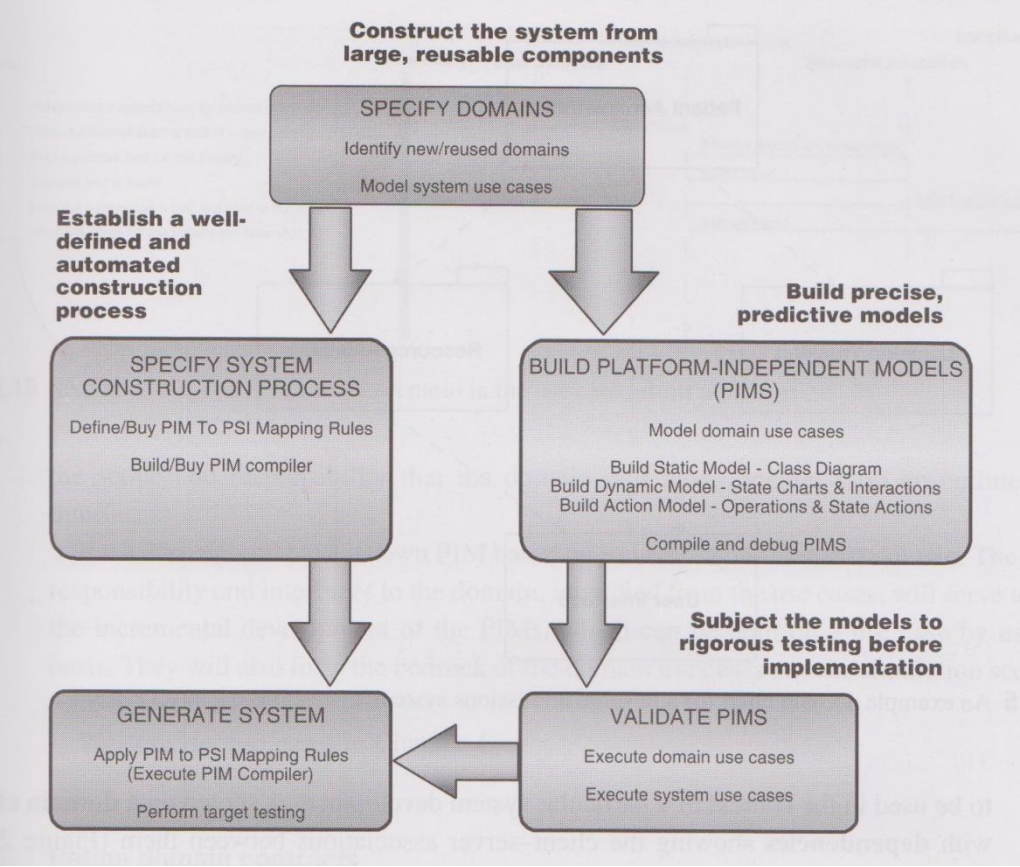
Základ PIM (jako modelu jedné domény) je diagram tříd, od kterého se odvíjejí další diagramy jako stavový diagram, nebo diagram aktivit. (5)

V MDA je model PIM vytvářen pomocí xUML. Tento přístup ale obecně není jediný, jak uvidíme v kapitole 4.

3.3 PSM (Platform Specific Model)

Platformě specifický (závislý) model (PSM) reprezentuje veškerou funkčnost systému z PIM modelu s přidanou vazbou na konkrétní realizační platformu (např. COBRA, C++, JAVA). PSM je vytvořen z PIM aplikováním transformačních pravidel, tento proces se nazývá *mapování*. Mapování samotné je také pospáno formálním modelem. Pro jeden PIM může existovat více PSM modelů. (5)

Existuje více přístupů generování PSM modelů. V prvním z nich, je PSM generován lidmi, týmem odborníků, který z PIM vytvoří PSM na základně cílové platformy. Tento přístup je však neefektivní. Přechodem k MDA jsme chtěli dosáhnout urychlení vývoje SW. Další přístup generování PSM je automatizované generování. Tento přístup již splňuje naše požadavky a je vhodný pro svou rychlost. Základem je, aby mapovací proces byl velmi dobře a precizně popsán/namodelován. Potom je možné úpravy v PIM modelu ihned přeložit a otestovat. Nedoporučuje se dělat zásah do vygenerovaného kódu, neboli PSI (Platform Specific Implementation), protože tím ztratí svou absolutní soudržnost s PIM modelem. (5)



Obrázek 3.4 MDA proces. Převzato z (5).

3.4 Souhrn MDA

MDA je proces vývoje systémů, který je dobře specifikovaný pod záštitou skupiny OMG, a ve světě používaný. Reprezentuje promyšlenou syntézu dobře otestovaných metodik softwarového vývoje. Proces vývoje systémů pomocí MDA se dá zjednodušeně popsat následujícími kroky, případně kresbou Obrázek 3.4.

1. Vytvořit precizní, prediktivní modely;
2. Podrobit modely důkladnému testování před samotnou realizací;
3. Sestrojit automatický, dobře definovaný konstrukční proces (mapování);
4. Konstruovat produkt ze znovu-použitelných komponent.

4. Modelování a simulace v procesu vývoje systémů

V předchozí kapitole jsem se zaměřil na existující světově uznávaný modelem řízený princip návrhu a vývoje SW zvaný Model Driven Architecture. V této kapitole poukážu na možnost zajít s teorií modelování a simulací ještě dále, než tomu je u MDA. Inspirací pro tuto kapitolu mi byl výzkum výzkumné skupiny Modelování a Simulace na fakultě FIT VUT v Brně ~~viz~~ (2). Tato skupina, kromě jiného, zkoumá možnosti vývoje počítačových systémů přístupem kombinace modelem řízeného návrhu (angl. model-based design) s interaktivním inkrementálním vývojem (angl. exploratory programming).

kapitole

Jak již bylo naznačeno v ~~kategorii~~ 2, aktuální vývoj v oblasti softwarového inženýrství spočívá v posuvu od statických k dynamickým modelům a jejich simulaci, které umožňují efektivní analýzu procesů odehrávajících se ve vyvíjeném systému. Výše zmíněná výzkumná skupina se snaží dodržet tento trend s tendencí zachovat proveditelný model v průběhu celého vývoje až k cílovému nasazení. Zde si všimněme rozdílnost od metodologie MDA (popsané v kap. 3), která využívaný model systému nakonec transformuje do klasického kódu. K udržení proveditelného modelu až po nasazení systému do reálného prostředí často pomohou techniky simulace *HIL (hardware-in-the-loop simulation)* a *SIL (software-in-the-loop simulation)*, které umožňují propojení reálných a simulovaných komponent v průběhu vývoje systému (4).

s tou verifikaci je to komplikovanejsi, to jeste probereme

Vyvíjený model systému, který je v průběhu vývoje simulován, jak s virtuálními prvky, tak s reálnými, je potřeba formálně verifikovat. Výzkumná skupina Modelování a Simulace se proto snaží nahradit tradiční programovací jazyky formálními modely. Zaměřili se na formalismy jako *vysokoúrovňové objektově-orientované Petriho sítě* (OOPN), *DEVS* a stavové diagramy (angl. state charts). Pro první dva formalismy implementovali nástroje, které kombinují programování a formální modelování, z čehož vzniká tzv. *programované modelování*, které umožňuje jednodušší práci s modely. (4)

tento vyraz je odkud?

Oba tyto formalismy a k nim vytvořené nástroje blíže proberu v následujících podkapitolách.

4.1 ~~OOPN (~~Objektově-orientované Petriho sítě~~)~~

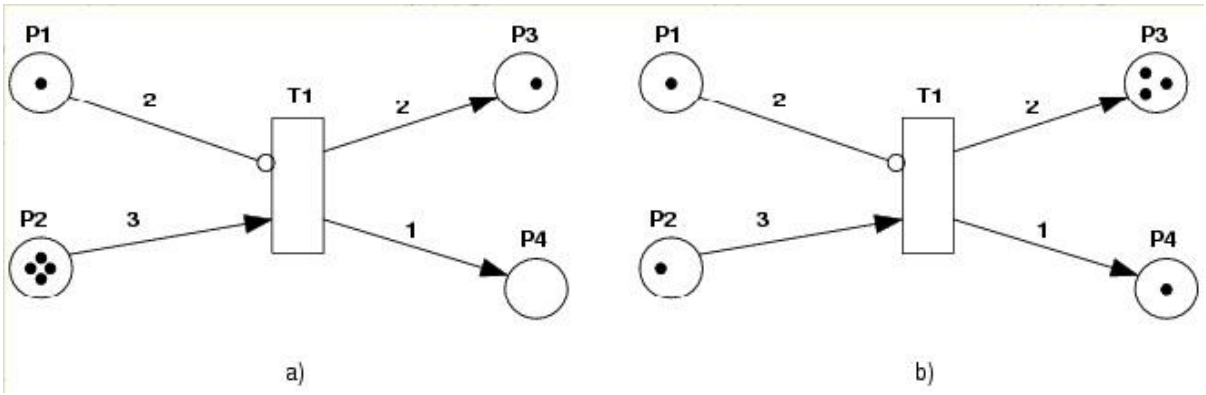
Petriho sítě představují dobrý formalismus pro modelování diskrétních systémů, s možností grafického znázornění a simulace s dobrou formální analyzovatelností. Populárnost Petriho sítí je dána jejich jednoduchostí. Model je sestaven z *míst*, které obsahují stavovou informaci ve formě *značek*, dále *přechody*, vyjadřující možné změny stavu a *hranami*, které propojují *místa* a *přechody*. Petriho sítě se dělí do různých typů, které vznikaly snahou zvýšit modelovací schopnosti tohoto formalismu při zachování koncepční jednoduchosti (8). Různé typy tedy jsou: C-E (Condition-Evets) sítě, P/T sítě, dále vysokoúrovňové (High-Level) sítě, jako například predikátové (Predicate-Transition), nebo barvené (Coloured) sítě, které umožňují pohodlně modelovat i zpracování a tok dat

blizsi seznameni (nebo tak neco) - OK blizsi sez

(8). Širší popis základních Petriho sítí není obsahem této práce, pro ~~zájemce~~ doporučuji publikaci (9), která popisuje použití Petriho sítí v softwarovém inženýrství.

Protože Petriho sítě ve své původní podobě neposkytují strukturovací mechanismy známé z programovacích jazyků omezovalo se použití Petriho sítí pouze na nepříliš rozsáhlé systémy. Podíváme-li se na rozvoj programovacích jazyků a hlavně objektově-orientovaného přístupu a postavíme jej vedle jednoduchých Petriho sítí, zjistíme, že tyto přístupy se zdají být ve svých výhodách a nevýhodách komplementární (8). Není divu, že lidé začali vymýšlet jak tyto přístupy propojit a získat tak od každého výhody a odstranit nevýhody, čili získat dobře strukturovaný, paralelní a hlavně formální (v podstatě matematický) programovací nástroj. Zavedením objektové orientace do Petriho sítí a následný modelovacího formalismu *PNtalk* je popsán panem Janouškem v jeho disertační práci (8) publikované v roce 2008. Spojením Petriho sítí s objektově-orientovaným přístupem se dosáhne toho, že aktivita objektu je modelována v objektu samotném, nikoliv nadřazenou strukturou, jak je tomu u klasických OO programovacích jazycích jako třeba Java, nebo C++, které často umožňují kvaziparalelizmus. Tímto se objekty stávají samostatnějšími a opravdu paralelními entitami, které spolu komunikují předáváním zpráv.

1996 nebo 1998 - OK



Obrázek 4.1 Příklad jednoduché P/T Petriho sítě a jejího jednoho přechodu a) => b). Převzato z (10).

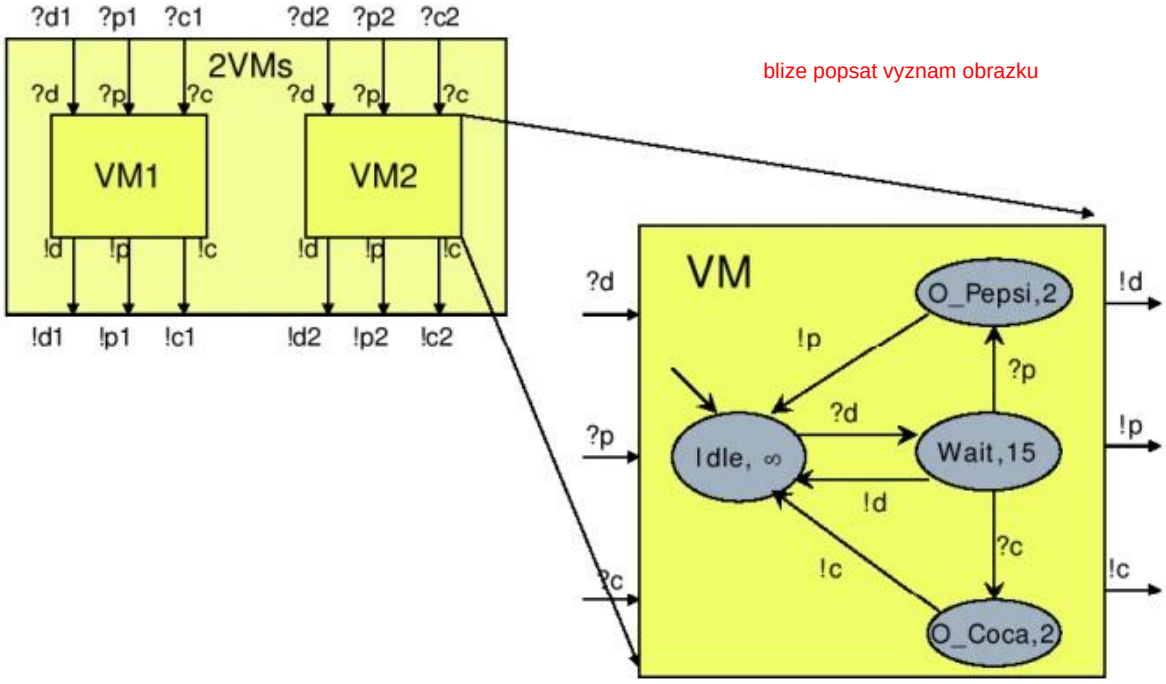
Tím, že se Petriho sítě nehodí pro modelování rozsáhlých systémů, užívají se spíše pro modelování jeho určitých ucelených částí, které jsou dále skládány dohromady, jak si ukážeme v následující podkapitole o formalismu DEVS.

4.2 DEVS

DEVS (Discrete EVent System specification) je formalismus pro specifikaci modelů založený na teorii systémů. DEVS formalismus rozlišuje atomické a složené komponenty (viz. Obrázek 4.2), kde složené komponenty mohou obsahovat další složené, nebo atomické komponenty. Atomické komponenty reprezentují jednu modulární jednotku s rozhraním tvořeným vstupními a výstupními *porty*. Na DEVS mohou být mapovány i jiné formalismy (např. konečné automaty, Petriho sítě, atp.), vzniká tak hierarchicky definovaný systém popsaný různými formalismy, dle vhodnosti aplikace. Atomická komponenta modeluje vlastní aktivitu v podobě reakce na externí (vstupní) události externí přechodovou funkcí a na interní (časově plánované) události interní přechodovou funkcí. Výstup komponenty je závislý na stavu komponenty. Pokud je systém komponován ze subsystémů, je jeho

blize vysvetlit

stav určen stavem všech jeho komponent. Množina DEVS je uzavřena vůči operaci skládání. (4)(11) (12)



Obrázek 4.2 Příklad systému popsaného DEVS formalismem. Převzato z (12).

DEVS umožňuje simulaci modelu formou nadřazení simulátoru samotnému modelu v hierarchii komponent a propojením s jeho rozhraním. Lze tak simulovat celý model najednou, nebo pouze jeho části.

Při porovnání DEVS s OO přístupem lze dle (12) zjistit následující vlastnosti:

- OO zvládá dynamický vznik/zánik objektů, DEVS to řeší komplikovaně
- OO čitelně modeluje systém tříd, DEVS naopak pracuje na úrovní instancí
- OO komunikace objektů vyžaduje referenci na adresáta, DEVS definuje komunikační vazby jako relaci nad komponentami
- Některé komponenty systému mají dlouhou životnost, jiné nikoli. Komponenty s delší životností je vhodné modelovat komponentami DEVS, zbytek potom objekty uvnitř atomických komponent.

DEVS je založen na komponentním přístupu a objekty jsou zde používány na jemnější úrovní (v atomických komponentách). (12)

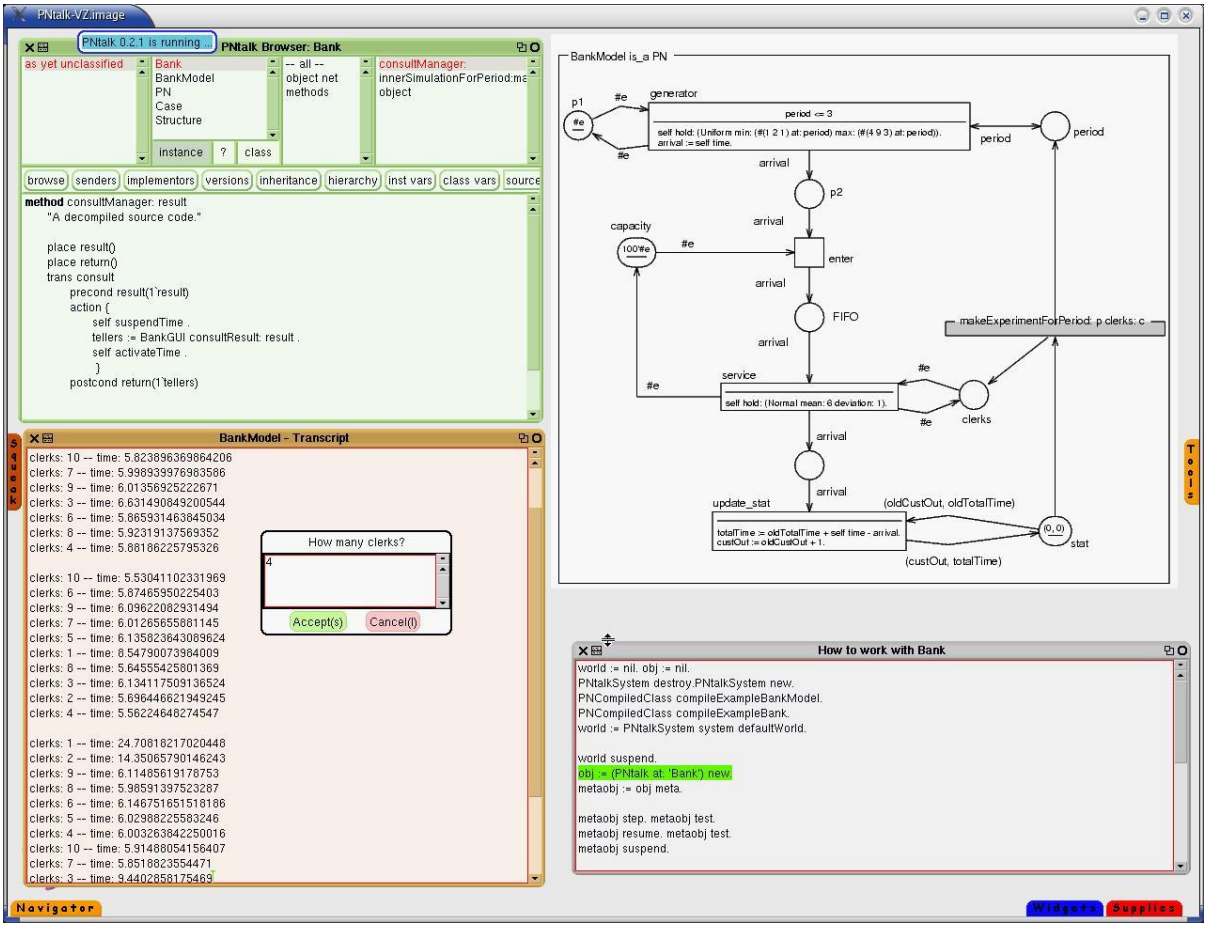
tuto kapitolu vice rozvest, je tu moc primych citaci ....

4.3 Nástroj PNtalk/SmallDEVS

V úvodu této kapitoly jsem zmínil výzkumnou skupinu Modelování a Simulace na FIT VUT v Brně (2). Tato skupina se zabývá nejenom průzkumem v oblasti modelování a simulace, ale také vývojem nových formalismů (viz. OOPN v kapitole 4.1) a nástrojů pro práci s nimi. Pro výše probrané formalismy to jsou nástroje *PNtalk* a *SmallDEVS*. Bystrému čtenáři jistě neunikne podobnost názvu s programovacím jazykem *Smalltalk* na kterém jsou oba nástroje založeny. Vývoj těchto nástrojů původně probíhal odděleně, nyní je však *PNtalk* integrovaný do *SmallDEVS*, respektive *SmallDEVS* slouží pro *PNtalk* jako operační systém. (4)

Pro stručné popsání nástrojů si dovolím doslovně citovat z (4).

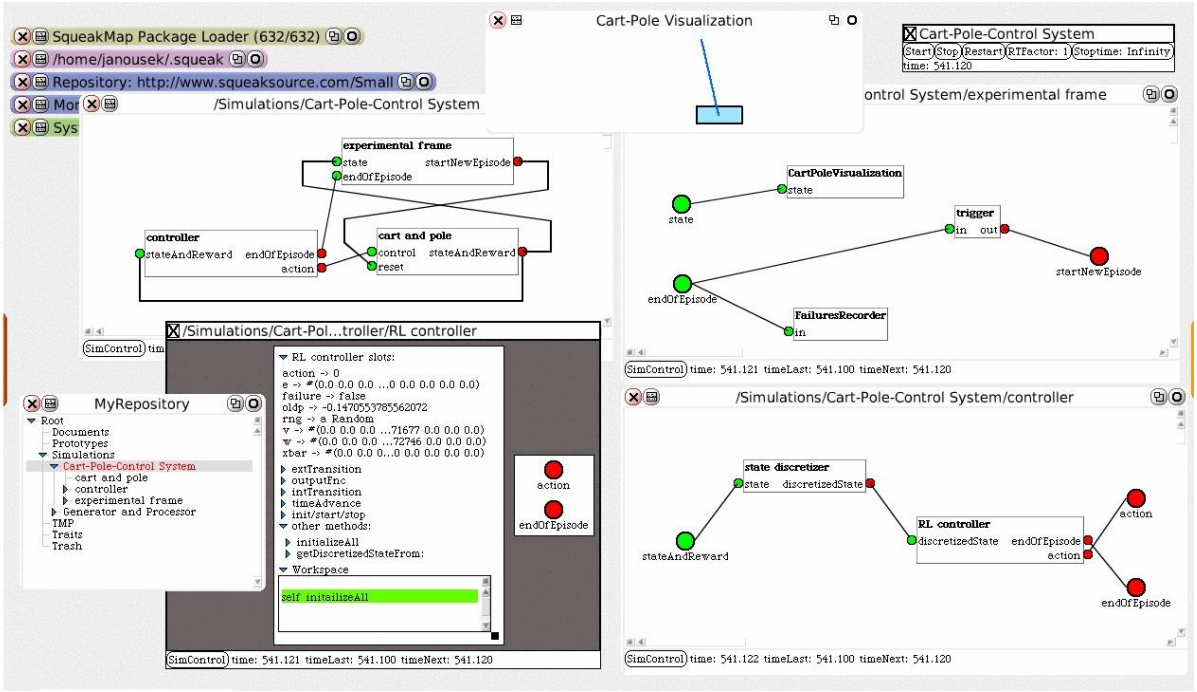
"PNtalk je experimentální modelovací formalismus založený na objektově-orientovaných vysokoúrovňových Petriho sítích (OOPN). Kombinuje vlastnosti Petriho sítí s výhodami objektově orientovaného návrhu systémů. Původním záměrem projektu PNtalk bylo prověřit možnost přímého použití matematického formalismu pro programování. Uvažuje se o aplikacích v oblasti workflow, plánování, logistiky a řízení."



Obrázek 4.3 Ukázka nástroje *PNtalk*. Převzato z (4).

*"SmallDEVS*http://www.fit.vutbr.cz/~janousek/smalldevs *je experimentální implementace formalismu DEVS, zaměřená obecně na modelování a prototypování, ze zvláštním důrazem na*

aplikace v oblasti adaptivních a inteligentních systémů. Umožňuje vývoj struktur i chování komponent za běhu, propojování simulovaných a reálných systémů a kombinaci s jinými formalismy, což je v těchto aplikacích nezbytné. V současné době SmallDEVS slouží jako operační systém pro PNtalk – poskytuje mu vývojové a prováděcí prostředí, uložiště modelů, vstupy-výstupy a komponentní zapouzdření."



Obrázek 4.4 Ukázka nástroje *SmallDEVS*. Převzato z (4).

Vývoj těchto nástrojů je zaměřen na modelem řízený vývoj systémů, přičemž je kladen důraz na ponechání modelu a možnosti jeho simulace až do poslední chvíle, čili k aplikačnímu nasazení. Z toho vyplývá další směr výzkumu a to optimalizace struktury modelů, transformace modelů do výkonnější podoby, případně možnost svázání s jiným programovacím jazykem. (4)

Tyto nástroje poskytují jistou formu uživatelského prostředí, avšak toto GUI ještě není úplně funkční. Vyvíjené nástroje jsou testovány na případových studiích jako třeba: konferenční systém řízený modelem workflow, plánování v agentních systémech, řízení robotických systémů a modelování a simulace senzorových sítí. (4)

5. Závěr

tady by chtelo vice shrnout zavery z teoreticke casti - jake pristupy maji jake vyhody + plan do dalsiho semstru (tj. dosazeni cile DP)

V této práci jsem se zabýval *modelem řízeným návrhem* počítačových systémů, přičemž jsem stručně prošel dosavadní historií vývoje softwarového inženýrství, dále jsem se detailněji zaměřil na metodiku MDA (Model Driven Architecture) vytvořenou skupinou OMG (viz. (1)).

Poté jsem probral význam využití modelování a simulací ve vývoji systémů a poukázal jsem na výzkum probíhající na fakultě informačních technologií na VUT v Brně (viz. stránky výzkumné skupiny (2)), který je zaměřen na využití formalismů jako DEVS a OOPN (Objektově-orientované Petriho sítě) a vývoj nástrojů pro ulehčení práce s těmito formalismy a udržení proveditelného modelu až do fáze nasazení aplikace.

Závěrem lze říci, že softwarové inženýrství se od svého počátku hodně vyvinulo a aktuální stav je uspokojivý. Je zde stále velký prostor pro výzkum a zlepšování metodik vývoje počítačových systémů.

Zvlastni zpusob citaci, ten jste prebiral odkud?

Bibliografie

1. **Object Management Group, Inc.** MDA. *Object Management Group.* [Online] 30. 10 2012. [Citace: 10. 12 2012.] http://www.omg.org/mda/.

2. **Kočí, R., a další.** Modelování a simulace. *Modelování a simulace.* [Online] [Citace: 11. 12 2012.] http://perchta.fit.vutbr.cz/research-modeling-and-simulation.

3. **Object Management Group, Inc.** UML Resource Page. *Object Management Group - UML.* [Online] 28. 11 2012. [Citace: 10. 12 2012.] http://www.uml.org/.

4. **Kočí, R. a Janoušek, V.** Simulace a vývoj systémů. *Modelování a simulace.* [Online] 17. 9 2010. [Citace: 11. 12 2012.] http://perchta.fit.vutbr.cz/research-modeling-and-simulation/3.

5. **Raistrick, Ch., a další.** *Model Driven Architecture with Executable UML.* Cambridge : Cambridge University Press, 2004. ISBN 0 521 53771 1.

6. **Zendulka, Jaroslav.** Projektování programových systémů. [Online] [Citace: 10. 12 2012.] http://www.fit.vutbr.cz/study/courses/PPS/public/pdf/11_2.pdf.

7. **Patočka, Miroslav.** *Řešení IS pomocí Model Driven Architecture.* [pdf] Brno : Masarykova Univerzita, 2008. Diplomová práce.

8. **Janoušek, V.** Modelování objektů Petriho sítěmi. *Disertační práce.* Brno : Fakulta informačních technologií, Vysoké učení technické v Brně, 2008. ISBN 978-80-214-3747-4.

9. **Girault, C. a Valk, R.** *Petri Nets for Systems Engineering. A Guide to Modeling, Verification, and Applications.* Berlin : Springer, 2003. ISBN 3-560-41217-4.

10. Objektová Petriho síť. *akela.mendelu.cz.* [Online] [Citace: 11. 12 2012.] https://akela.mendelu.cz/~petrj/opnml/opn.html.

11. **Peringer, P.** Simulační nástroje a techniky. *Slajdy k přednáškám.* [pdf]. Brno : Fakulta informačních techologií, Vysoké učení technické v Brně, 2011.

12. **Janoušek, V.** Systémy s diskrétními událostmi, formalismus DEVS. *Slajdy k přednáškám.* [pdf].

Seznam příloh