Využití UML a dalších metod formálního popisu aplikací pro generování zdrojového kódu aplikací a reverzní softwarového inženýrství

Application of UML-based formal design methods for source code generation and reverse software engineering

Martin Malaník

Bakalářská práce
2009

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2008/2009

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin MALANÍK**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Využití UML a dalších metod formálního popisu aplikací pro generování zdrojového kódu aplikací a reverzní softwarové inženýrství.**

Zásady pro vypracování:

1. **Vytvořte rešerši na téma využití UML a dalších metod formálního popisu aplikací pro generování zdrojového kódu aplikací a reverzní softwarové inženýrství.**
2. **Srovnejte současně dostupné vývojové CASE nástroje (komerční i volně dostupné) a technologie z hlediska jejich schopnosti produkovat programový kód (generování programového kódu).**
3. **Srovnejte současně dostupné vývojové CASE nástroje (komerční i volně dostupné) a technologie z hlediska schopnosti analýzy existujícího zdrojového kódu (reverzní inženýring).**
4. **Vytvořte sadu příkladu demonstrujících práci se srovnávanými vývojovými prostředími.**
5. **Srovnejte tyto nástroje: Sparx Enterprise Architect, Visual Paradigm for UML, MagicDraw UML, Poseidon for UML, IBM Rational Software Architect, SmartDraw, ArgoUML, BOUML, Eclipse, Eclipse UML2Tools, Eclipse PyUML, UML Designer.**
6. **Vytvořte přehlednou srovnávací tabulku vlastností testovaných produktů a dosažené výsledky testů komentujte.**

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **ARLOW, J., NEUSTADT, I. UML a unifikovaný proces vývoje aplikací. Brno: Computer Press, 2003. 387 s. ISBN 80-7226-947-X.**
2. **KANISOVÁ, H., MÜLLER, M. UML srozumitelně. 2. akt. vyd. Brno: Computer Press, 2006. 176 s. ISBN 80-251-1083-4.**
3. **Computer-aided software engineering na Wikipedii. URL: (http://en.wikipedia.org/wiki/Computer-aided_software_engineering).**
4. **Reverse engineering na Wikipedii. URL: (http://en.wikipedia.org/wiki/Reverse_engineering).**
5. **Sparx Enterprise Architect. URL: (http://www.sparxsystems.com).**
6. **Visual Paradigm for UML. URL: (http://www.visual-paradigm.com).**
7. **MagicDraw UML. URL: (http://www.magicdraw.com).**
8. **IBM Rational Software Architect. URL: (http://www-01.ibm.com/software/awdtools/swarchitect/websphere).**
9. **SmartDraw. URL: (http://www.smartdraw.com).**
10. **Eclipse, Eclipse UML2Tools. URL: (http://www.eclipse.org).**
11. **Eclipse PyUML. URL: (http://sourceforge.net/projects/eclipse-pyuml).**
12. **UML Designer. URL: (http://www.greenbirdsoftware.com).**

Vedoucí bakalářské práce: **Ing. Michal Bližňák, Ph.D.**
Ústav aplikované informatiky

Datum zadání bakalářské práce: **20. února 2009**

Termín odevzdání bakalářské práce: **1. června 2009**

Ve Zlíně dne 13. února 2009

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Cílem této bakalářské práce je srovnat současné nástroje pro softwarové inženýrství z pohledu generování zdrojového kódu a reverzního inženýrství. Teoretická část se skládá z úvodu do modelovacího jazyka UML, principu generování zdrojových kódů, reverzního inženýrství a základního popisu CASE nástrojů. Praktická část se pak zabývá srovnáním šesti komerčních a šesti volně dostupných CASE nástrojů z hlediska generování zdrojových kódů a reverzního inženýrství z programovacího jazyka Java.

Klíčová slova: generování zdrojového kódu, reverzní inženýrství, CASE nástroje, UML

ABSTRACT

The aim of this bachelor project is to compare actual development tools for software engineering from code generating and reverse engineering point of view. The theoretical part consists of introduction to UML, code generating and reverse engineering fundamentals and basic description of CASE tools. The practical part is concerned with testing of six commercial and six freeware CASE tools from view of code generating and reverse engineering from Java programming language.

Keywords: code generating, reverse engineering, CASE tools, UML

Na tomto místě bych rád poděkoval vedoucímu své bakalářské práce, panu Ing. Michalu Bližňákovi, Ph.D. za vedení při mé práci. Dále přátelům a spolužákům za jejich pomoc a spolupráci. A také svým rodičům za celoživotní podporu nejen při studiu.

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.

Ve Zlíně …….

podpis diplomanta

OBSAH

ÚVOD

V současnosti jsme obklopeni nejrůznějšími rozsáhlými informačními systémy, se kterými se setkáváme prakticky na každém kroku (internetové bankovnictví, IS systém vysoké školy apod.). Základem takového stabilního a dobře fungujícího systému je jeho počáteční návrh, model.

Dříve stačilo, aby analytik vzal tužku do ruky a na kus papíru načmáral jednoduchý vývojový diagram a měl vymyšlenou strukturu programu. S vývojem doby se však klade důraz i na vývoj systému – na jeho rozsah a schopnosti. Dnešnímu analytikovi by tedy už nestačil ani celý blok se sadou tužek.

Proto vznikly tzv. CASE nástroje. Tyto nástroje jsou určeny k tomu, aby pomáhaly s vývojem nejen dalších počítačových systémů, ale také např. pro modelování firemních procesů a jejich optimalizaci.

Takových nástrojů je obrovské množství. Některé dokážou pokrýt celý životní cyklus projektu (počáteční návrh, průběh vývoje, konečná realizace), jiné zase pouze kreslení zmíněných „vývojových diagramů“. Základem těchto nástrojů je tedy vytvoření modelu, který je postaven na požadavcích zadavatele, a který je možno v budoucnu snadno měnit upravovat.

Projektovým manažerům, analytikům, vývojářům i testerům se tak dostává do rukou neskutečně silná zbraň, která dokáže uchovávat a třídit jejich myšlenky a postup ve vývoji na daném projektu.

Cílem této práce je zhodnotit několik současných CASE nástrojů ze dvou hledisek. Prvním je schopnost generování zdrojových kódů z již vytvořeného modelu. Druhým pak možnost reverzního inženýrství z již napsaných zdrojových kódů (např. za účelem zdokumentování či přepracování).

I. TEORETICKÁ ČÁST

1 UML – UNIFIED MODELING LANGUAGE

1.1 Co to je?

Jazyk UML, neboli Unified Modeling Language, můžeme označit jako sjednocený modelovací jazyk. To znamená, že na rozdíl od jiných, spíše textově orientovaných jazyků, má tento i svou grafickou syntaxi a sémantiku. Syntaxí jsou pravidla pro postupné sestavování jednotlivých elementů do větších celků, z nichž se skládá výsledný projekt a sémantikou se označuje význam jednotlivých syntaktických výrazů.[6]

V současnosti je základem kvalitního softwarového systému především jeho dobře promyšlený objektově-orientovaný návrh. Tento návrh je pak předpokladem pro úspěšnou a rychlou implementaci celého projektu. Právě zde nachází UML své nejsilnější uplatnění. Původní návrh sice můžeme realizovat pomocí nejrůznějších podpůrných prostředků, zejména odlišných typů diagramů, UML má však nespornou výhodu v tom, že přesně definuje, co má který diagram obsahovat. Toto pak zaručuje jistou konzistenci a přesně danou sémantiku, což je velmi důležité při sdílení informací mezi jednotlivými lidmi v týmu (a to ať už se jedná o vývojáře, analytiky, projektové manažery či testery).[6]

Dalšími z předností jazyka UML (a díky němu vytvořených diagramů) je také použití otevřeného a rozšiřitelného standardu, podpora kompletního vývojového cyklu aplikace či jiného systému a velká podpora pro různé aplikační oblasti. Rozvoj v jeho šíření mu zaručuje také fakt, že je podporován celou řadou nástrojů. Jedná se jak o nástroje pevně určené pro práci s UML, tak o integrované vývojové prostředí (IDE), které mnohdy dovolují provádět i převod mezi UML diagramem a algoritmem zapsaným v některém z rozšířených programovacích jazyků (a samozřejmě i opačně – což ale není vždy úplně realizovatelné).[6]

1.2 Historie

Prvopočátky objektově-orientovaných jazyků sahají až do druhé poloviny sedmdesátých let. Kolem roku 1980 se počet těchto jazyků rozšířil na desítky. Bohužel právě toto příliš nepřispělo k šíření žádného z nich. Až později po roce 1990 začaly vznikat další verze jazyků, které od sebe navzájem přejímaly některé vlastnosti.[22]

Pravá historie jazyka UML začala až v roce 1994, kdy pánové Grady Booch a Jim Rumbaugh z firmy Rational Software Corporation (dnes patřící pod IBM) začali pracovat na sjednocení metod Booch a OMT (Object Modeling Technique). Když se pak na podzim roku 1995 přidal k Rational Software Ivar Jacobson s jeho metodou Objectory, začala se konečně rýsovat kvalitní metodika pro objektově-orientované softwarové inženýrství.[22]

Snaha všech tří pánů nakonec přinesla ovoce už v červnu roku 1996, kdy bylo představeno UML 0.9 (o pár měsíců později UML 0.91). Během tohoto roku pak autoři sbírali názory uživatelů, což znamenalo, že vývoj se rozhodně zastavit nehodlá.[22]

Rational Software tedy tvrdě pracoval na sjednocování UML. Mezitím však začal být kladen důraz i na ustavení standardizovaného modelovacího jazyka. O toto začali usilovat pánové Ivar Jacobson a Richard Soley (Chief Technology Officer Object Management Group - OMG).[22]

OMG je mezinárodní otevřené neziskové společenství, které napříč celým průmyslem vyvíjí nejrůznější standardy pro odlišné technologie. V červnu roku 1995 se představitelé hlavních metologií shodli na potřebě hledání standardu (samozřejmě pod záštitou OMG).[22]

V roce 1996 se několik společností shlédlo v UML a začaly na něm stavět svůj byznys. Rational Software založil konsorcium „UML Partners“, které sdružovalo společnosti, které byly ochotny vynaložit prostředky pro tvorbu silné definice UML 1.0. Mezi těmito společnostmi byly například DEC, HP, Microsoft, Oracle, IBM či Unisys. Toto společenství nakonec pomohlo ke vzniku UML 1.0. Ke vzniku jazyka, jenž byl dobře definován, byl dostatečně expresivní, silný, ale též velmi dobře aplikovatelný v široké škále oblastí. Tento úspěch zapříčinil připojení dalších společností k UML Partners, které chtěly přispět svými nápady. Zanedlouho bylo tedy vytvořeno UML 1.1, jež se zaměřovalo především na srozumitelnost sémantiky. Tuto verzi přijalo OMG ještě na podzim roku 1997. UML poté prošlo ještě mnoha změnami (ty většinou pouze opravovali chyby předchozích verzí). UML 1.4.2. se pak v roce 2005 stalo i mezinárodním standardem (ISO/IEC 19501:2005).[22]

Aktuálně oficiální verzí podporovanou OMG je UML 2.0. Některé CASE nástroje však už podporují neoficiální verzi UML 2.1.[21]

1.3 Popis UML

Jak už bylo předesláno, UML nám nedefinuje kompletní proces vytvoření úspěšného projektu (tedy například informačního systému). UML je pouhou množinou nástrojů, které se k tomuto procesu používají – je dobré strávit dostatek času nad výběrem toho správného. Důležité je též vybrat správnou metodiku, která nám bude definovat kdo a jak má který nástroj používat. Teprve toto nám pak dovolí naplno využít všech možností UML. Z objektových metodik můžeme jmenovat například Unified Process, Catalysis (vhodná pro tvorbu systémů stojících na principech CBD – Component-based Development) či metodika Select Perspective, která pokrývá celý životní cyklus projektu.[23]

UML nástroje pro modelování nám standardně oddělují statické a dynamické stránky systému. Statické stránky nám ukazují, jak systém vypadá, zatímco dynamické popisují, jak se systém chová v čase a jak reaguje na nejrůznější změny. Dohromady nám tyto části umožňují pochopit celkovou architekturu systému. Vhodný výběr modelovacích nástrojů nám tedy umožní vytvářet systém ze všech možných pohledů. Toto je velmi důležité pro úspěch celého projektu. Díky správné volbě totiž můžeme vytvořit přehledný model dílčí i celkové architektury vyvíjeného systému a zaručit tak jak jeho kvalitní a rychlou implementaci, tak i úspěšné reálné nasazení.[23]

V drtivé většině případů (ne-li snad úplně ve všech) je počátečním krokem při vývoji analýza. Jejím prvním úkolem je vymezení hranic modelovaného systému. Toto je samozřejmě základem jakéhokoli objektově-orientovaného vývoje. Tedy poznat zkoumaný předmět ze všech jeho stran a následně na základě získaných poznatků učinit další kroky. Těmi pak může být například syntéza (etapa designu a implementace).[23]

Po úspěšném vymezení hranic systému následuje krok další. Tím je nalezení vhodných objektů v systému a určení všech vztahů mezi nimi. Toto je též velmi důležitý krok, jehož chybné provedení může v budoucnu způsobit nemalé problémy. Takto vytvořený objektový model poté prochází dalšími etapami vývoje. Jednou z nich pak je vytvoření předlohy pro implementaci, se kterou nám s využitím automatického generování kódu může pomoci vhodně zvolený CASE nástroj.[23]

Vytvořený model pomocí UML se skládá z různých diagramů, které reprezentují různé pohledy na modelovaný systém. Tyto pohledy jsou pak souhrnem specifikací systému, jenž vymezuje prostor, ve kterém se budou patřičné osoby během vývoje pohybovat. Diagram

ve vizuální formě představuje právě jednu konkrétní situaci, která může v systému nastat. Jazyk UML rozeznává pět základních pohledů na systém:[7]

Pohled případů užití – případy užití (use cases) vyjadřují základní požadavky, které jsou kladeny na systém a definují také hranice pro ostatní pohledy.

Logický pohled – tento pohled je určen pro klienta (zadavatele). Jsou zde užity pojmy a vzájemné vztahy z problémové domény.

Procesní pohled – znázorňuje chování systému, které musí splňovat požadavky a omezení z případů užití, jež jsou kladeny na průběh procesů.

Pohled nasazení – mapuje komponenty na množinu fyzických výpočetních uzlů v cílovém prostředí.

Implementační pohled – ukazuje, jak budou jednotlivé komponenty a jejich propojení v aplikaci rozděleny.

Veškeré pohledy jsou pak využity v diagramech – jejich popis bude uveden v samostatné kapitole.

1.4 Základní pojmy UML

Jazyk UML je rozdělen na tři základní stavební bloky. Těmi jsou předměty, diagramy a relace.

1.4.1 Předměty

Předměty jsou elementy zpracovávaného modelu. Tyto jsou pak členěny do dalších podkategorií. Jednou z takových podkategorií jsou *Strukturní abstrakce* – to jsou například nejrůznější programové třídy, aplikační či objektové rozhraní, komponenty, uzly a případy užití. Takový diagram by však měl obsahovat minimálně dvě takové abstrakce. Pro jednu abstrakci totiž modelování ztrácí svůj smysl.[7]

Další podkategorií jsou takzvaná *Chování*. Ta v diagramu zastupují interakce – neboli komunikaci mezi jednotlivými předměty. Tato podkategorie nám pak dovoluje vytvořit model, u nějž se stavy dají specifikovat pomocí přechodů, událostí a aktivit.[7]

Seskupení patří též mezi podkategorie předmětů. Dle potřeby (přehlednost) seskupuje různé části diagramu. Seskupení však nejsou použitelná u všech pomocných nástrojů, což může být v případě rozsáhlejších grafů problémem.[7]

Nejjednodušší podkategorií jsou *Poznámky*. Tyto většinou blíže specifikují chování elementů v diagramu.[7]

1.4.2 Diagramy

Diagramy zachycují různé stránky systému. Tyto diagramy mohou být například pomocí balíčků různě sdružovány do hierarchicky organizovaného celku. Typů diagramu je celá řada, a proto budou popsány v samostatné kapitole.

1.4.3 Relace

Relace popisují nejrůznější vztahy mezi jednotlivými předměty. Tyto vztahy se pak můžou různě ovlivňovat. UML rozlišuje více druhů stavů.[7]

Jedním z nich je například *Asociace* neboli obecná souvislost předmětů. UML tuto relaci popisuje jako „měnící se populaci vztahů daných propojením objektů“. Speciálními variantami asociace jsou pak kompozice a agregace.[7]

Další relací je *Závislost*. Jako už plyne z jejího názvu – změna v jednom předmětu způsobí také určitou změnu v předmětu druhém. Dobře zmapované závislosti nám můžou v budoucnu ušetřit mnoho práce s hledáním nejrůznějších chyb. Pokud je ale relace tohoto druhu přemíra, může vést k znepřehlednění systému a tím i jeho budoucí obtížnou údržbu.[7]

Generalizace nebo také dědičnost. Touto relací je míněno, že jeden předmět je speciální verzí druhého. Vztah generalizace je velmi důležitý pro veškeré objektově-orientované modelování. Opět ale přemíra jeho užití může vést k nemalým problémům.[7]

Realizace je druh vztahu, při kterém jeden předmět představuje dohodu, za jejíž plnění je odpovědný jiný předmět (předměty). Podobné je například využití rozhranní v programovacím jazyce Java.[7]

1.5 Případy užití (use cases)

V první fázi analýzy se ještě neřeší veškeré technologické aspekty modelovaného systému. Neřešíme tedy, na jaké platformě bude náš systém postaven či jaké výhody či nevýhody přináší různé typy databází. Co však zjistit musíme je, které procesy bude budoucí systém podporovat a jací uživatelé s ním budou pracovat. Případy užití (Use Case) jsou psány z pohledu zákazníka (zadavatele) a podávají vůbec první představu a rozsahu budoucího projektu. Využívá se zde pojmů přirozeného jazyka a termínů z problémové domény. Toho se užívá především kvůli srozumitelnosti vůči zadavateli – pomůže nám to tak co nejsrozumitelněji načrtnout kostru systému. Příklady případů užití mohou být:[7]

- Zobrazení podrobností o zboží
- Zadání kontaktních údajů zákazníka
- Přihlášení do systému

V reálné analýze se většinou setkáváme s velkým počtem případů užití s mnohdy ještě větším počtem nejrůznějších závislostí. Toto by při přímém zobrazení zničilo jakoukoli vypovídající hodnotu UML diagramu. Proto se využívá nejrůznějších úprav (označení opakování některých vztahů, nepovinného chování, společných rysů, …) ke zjednodušení daného diagramu. Popis těchto úprav už je však nad rámec této práce a proto zde nebudou blíže popisovány.[7]

1.6 UML diagramy (UML diagrams)

Diagramy jsou základem celého modelování v UML. Z čeho se skládají, již bylo popsáno výše, nyní následuje bližší popis jejich několika typů.

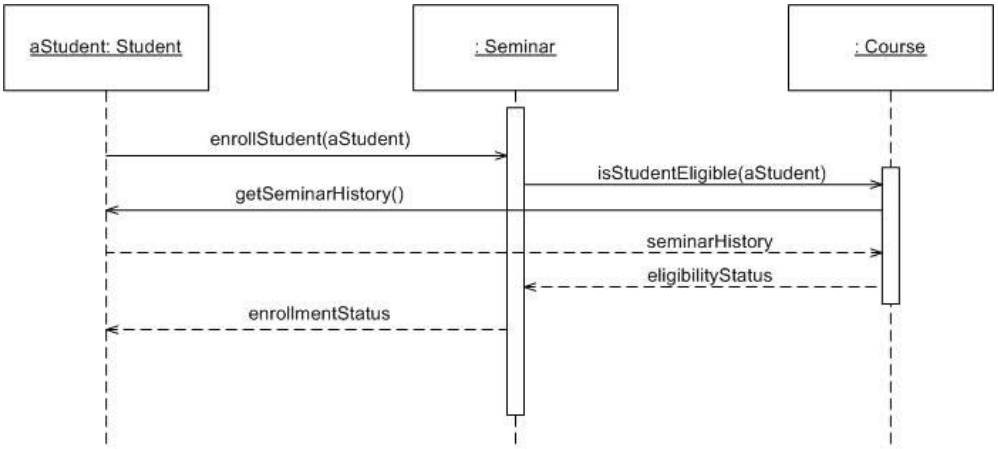
1.6.1 Diagram případů užití (use case diagram)

Vytyčení hranic systému je v UML podpořeno dynamickým pohledem modelovaným prostřednictvím diagramů případů užití. Tento diagram je užíván i pod jinými názvy jako jsou například: diagram typových činností, jednání atd. V rámci tvorby těchto diagramů tedy modelujeme vztahy systémem a okolím (aktoři, externí systémy) a nikoliv jejich vzájemnou interakci, tzn. způsob, jakými jsou jednotlivé případy užití zajištěny interními funkcemi systému.[23]

1.6.2 Sekvenční diagram (sequence diagram)

Sekvenční diagramy jsou vytvářeny většinou přímo z diagramu případu užití. Modelují interakci objektů v rámci komunikace mezi aktorem a systémem (čili popisují dynamickou stránku systému). Z tohoto je zřejmé, že k jednomu případu užití může existovat více sekvenčních diagramů. Sekvenční diagramy jsou vhodné pro nalezení a rozlišení objektů v systému. Často se totiž stává, že modely nalezené v počáteční fázi modelování, se ne tak úplně kryjí se softwarovými objekty (implementované v některém z programovacích jazyků). Proto postupem času (zejména během fáze designu) dochází k redukci objektových modelů.[23]

Sekvenční diagram se skládá ze dvou dimenzí pohledu. V horizontální dimenzi jsou zobrazovány jednotlivé objekty. Vertikální pohled zase znázorňuje tok času. Zprávy, které jsou posílány mezi jednotlivými objekty, mohou mít různý charakter – asynchronní, vnořené, reprezentující návratové hodnoty atd. Zprvu nás však zajímá pouze charakter obecné komunikace, podrobnější rozpracování se vytváří až v pozdějších fázích vývoje. V případě nutnosti se v sekvenčních diagramech používají i cykly a větvení. Pokud nám v průběhu rozpracovávání sekvenční diagram neúměrně narůstá, může to značit přílišnou obecnost některého z případů užití, ke kterému je daný diagram vytvářen.[23]

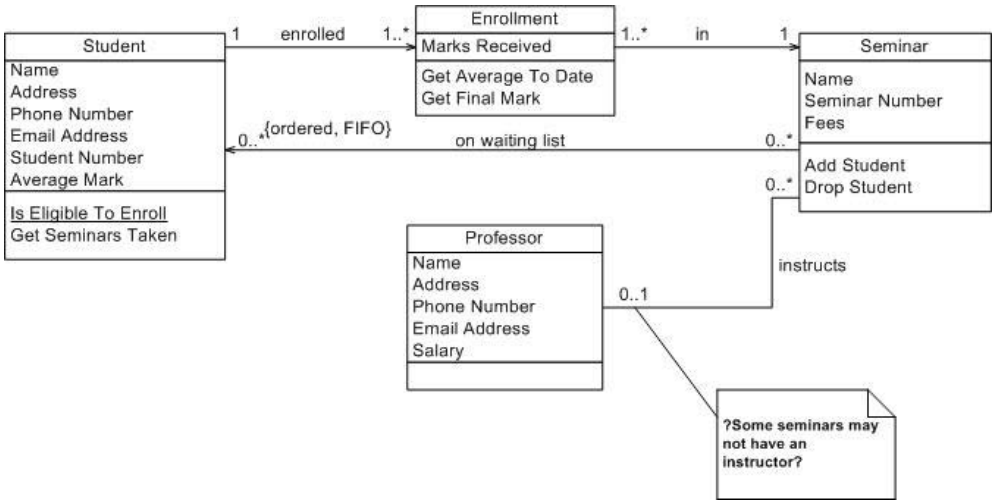


Obrázek 1.1 – Příklad sekvenčního diagramu [20]

1.6.3 Diagram tříd (class diagram)

Jedním z nejčastěji používaných nástrojů je právě diagram tříd. Jde vlastně o jakýsi základ všech prostředků objektové analýzy a designu. S postupným přibližováním k fázi implementace jsou tyto diagramy přehodnocovány a v ideálním případě přepracovány do

podoby grafického modelu daného zdrojového kódu. Diagramy tříd zobrazují, na rozdíl od předešlých diagramů, statickou stránku systému. UML pak striktně rozlišuje více druhů tříd (parametrizovatelné třídy, …) a také množství vztahů, kterými jsou jednotlivé třídy propojeny (asociace, agregace, kompozice, generalizace, …).[23]

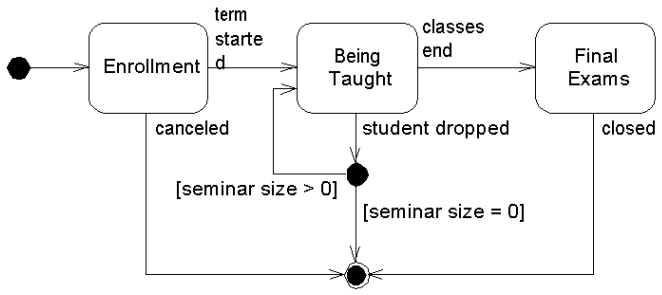


Obrázek 1.2 – Příklad diagramu tříd [20]

Tvorba tohoto typu diagramů je naprosto klíčovým problémem. Zvládnout objektový přístup totiž znamená správně používat i právě tento typ diagramů (používá se totiž v průběhu celého životního cyklu projektu). Při tvorbě je pak nepsaným pravidlem využití 7+2, což znamená, že jeden diagram obsahuje 7-9 tříd. To bude mít za následek hlavně přehlednost. Vzhledem k velkému množství modelovacích prvků, jež můžeme použít v tomto diagramu, může se velice snadno sklouznout k tvorbě formálně přesných, avšak těžko srozumitelných modelů.[23]

1.6.4 Stavový diagram (state transition diagram)

Tento diagram slouží pro modelování životního cyklu části systému a svým rozsahem odpovídá jednomu objektu. Změna stavu pak bývá vyvolána signálem z vnějšího okolí - například nějaká forma zprávy zaslaná danému objektu. Jednoduše se toto dá popsat tak, že změna stavu nastane při změně některého z atributů objektu. Samozřejmě že objekt musí mít svůj start i konec. Start by měl být v objektu jen jeden, zatímco konců může být více. Z implementačního hlediska je startem například zavolání konstruktoru dané třídy, ukončení pak obstará destruktor či nějaký automatický prostředek správy paměti.[23]

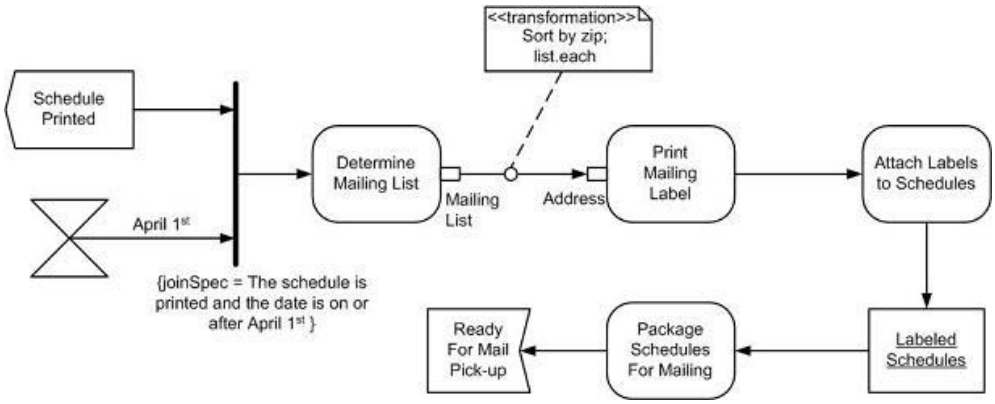


Obrázek 1.3 – Příklad stavového diagramu [20]

1.6.5 Diagram aktivity (activity diagram)

Diagram aktivity je jistou obdobou stavových diagramů. Tento typ diagramů se zpravidla vztahuje k jednomu případu užití, případně k jedné metodě objektu. Pomocí diagramu aktivit pak modelujeme dynamický tok, jenž už není řízený vnějšími událostmi, ale interními podněty.[23]

Stejně dobře jako pro procesní modelování je možné používat diagramy aktivit i pro základní schematickou tvorbu grafického uživatelského prostředí, to nám může následně odhalit různé nesrovnalosti například v návaznosti jednotlivých GUI obrazovek.[23]



Obrázek 1.4 – Příklad diagramu aktivit [20]

1.6.6 Další diagramy

Mezi další diagramy patří například:[23]

- Diagram spolupráce (collaboration diagram)
- Diagram komponent (component diagram)
- Diagram nasazení (deployment diagram)

2 GENEROVÁNÍ ZDROJOVÉHO KÓDU

2.1 Automatické generování zdrojového kódu / generátory

Automatickým generováním zdrojového kódu se rozumí přepis modelu jazyka UML do jednoho z objektově-orientovaného programovacího jazyka. Generátorem zdrojového kódu pak nazýváme aplikaci, která tento přepis provádí.[3]

Automatické generování však není žádnou novinkou. V softwarovém inženýrství je využíváno již řadu let. Při denní práci takové generování používáme běžně. Příkladem jsou třeba generátory dokumentů ve formě kompilátorů (generování strojového kódu z některého z vyšších programovacích jazyků), generátory dokumentace (např. Javadoc) nebo třeba generátory kódu pro webové stránky (HTML). Obecně můžeme tedy generátory označit jako programy, které tvoří dokumenty či jiné programy (především programové skelety).[5]

2.2 Metody generování kódu

2.2.1 Generování kódu pomocí šablon a filtrů

Metoda šablon a filtrů je jednou ze základních metod generování zdrojového kódu. Postup při jejím použití je následující:[3]

- Pomocí některé z šablon (např. XSLT) se definuje předpis pro generování všech částí modelu (metody, třídy, balíčky, …), které jsou stejného typu
- Poté se z modelu vyberou důležité části vhodné ke generování kódu a ty se následně převedou do textového formátu (nejčastěji XML/XMI)
- Na takový zjednodušený model se aplikují dříve vytvořené šablony
- Výstupem je generovaný zdrojový kód (nebo jeho část)

Příklad použití šablony a filtrů - výše zmíněný XMI popis vizuálního modelu filtrací zjednodušíme a následně aplikujeme XSLT šablonu:[5]

```
<xsl:template match="class">
      public class <xsl:value-of select="name"/> {
      <xsl:apply-templates select="attribute">
      }
</xsl:template>
<xsl:template match="attribute">
      private <xsl:value-of select="@type"/>
      <xsl:value-of select="@name"/>;
</xsl:template>
```

Výsledný kód v jazyce Java by mohl vypadat třeba takto:[5]

```
Public class NazevTridy {
      private int nazevAtributu;
}
```

2.2.2 Generování kódu pomocí šablon a metamodelů

Tato metoda je úpravou metody předchozí. Postup je v jejím případě tento:[3]

- Definuje se určitý metamodel (předpis, jenž určuje jaké prvky a jaké vztahy lze v modelu použít) tak, aby byl totožný pro všechny modely, z nichž se chystáme generovat zdrojový kód
- Opět se vytvoří šablony pro generování kódu (viz předchozí metoda)
- Na vytvořený metamodel se aplikují šablony = vygenerování části kódu

2.2.3 Generování kódu pomocí generativních gramatik

Jedná se o metodu, která vychází z teorie formálních jazyků. Řekněme, že symbol Y lze přepsat na slovo y, jestliže dvojice (Y; y) je z množiny určitých přepisovacích pravidel. Řekneme, že gramatika G generuje jazyk L, jestliže lze po konečném počtu použití našich přepisovacích pravidel odvodit z inicializačního symbolu gramatiky G libovolné slovo jazyka L. Pro takový způsob generování kódu se používá tento postup:[3]

- Definuje se gramatika G jazyka UML tak, aby generovala jeho textovou formu
- Následně se definuje gramatika L cílového programovacího jazyka
- Nakonec se vhodným způsobem vytvoří gramatika T, která zkombinuje gramatiky G a L. Tato gramatika bude mít inicializační symbol (symbol, ze kterého jsou odvozena generovaná slova) gramatiky G a generovat bude slova gramatiky L

2.3 Výhody a nevýhody generování kódu / generátorů

Výhody:[5]

- Kvalita generovaného kódu je konzistentní (použitím šablon atd.)
- V případě změny je možnost opětovné generace kódu jen za pomoci úpravy šablon
- Úspora času, který může být využit na analýzu a návrh
- Zjištění možných chyb již při vytváření šablon
- Zvýšení produktivity

Nevýhody:[5]

- Nutnost použití některého z generátorů (různá kvalita, možnosti, licence)
- Nutnost dobrého analytického myšlení při tvorbě šablon
- Generátor nelze použít kdykoli a na cokoli
- Tvorba jen části kódu -> nutno dopsat zbytek kódu ručně

3 REVERZNÍ INŽENÝRSTVÍ

Reverzní inženýrství je proces sloužící k objevení technologických principů mechanické aplikace skrze analýzu jeho struktury, funkcí a operací. V softwarovém pohledu nám tedy může sloužit například k opětovnému vytvoření ztracené dokumentace k některé aplikaci. Reverzního inženýrství je využito i v mnoha dalších odvětvích (např. v armádě, kdy se kopírují technologie jiných národů).[25]

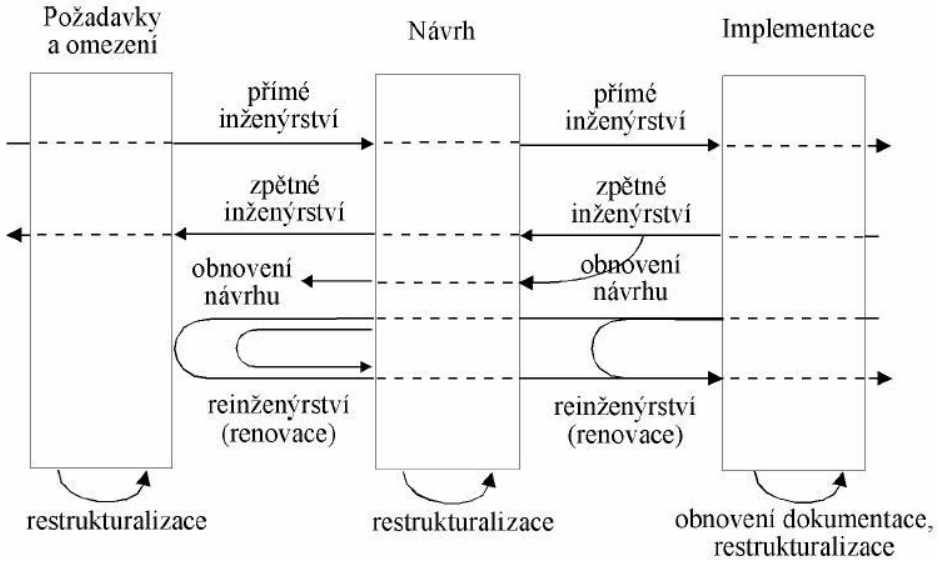
3.1 Reverzní inženýrství dle Chikofského

Elliot J. Chikofsky definoval reverzní inženýrství jako proces vedoucí k porozumění struktury a vnitřních vztahů systému. To znamená, že při reverzním inženýrství jde především o prozkoumání daného systému. Úpravy a jakékoli jiné formy jeho restrukturalizace tedy do této definice už nepatří.[4]

Výsledný popis by však měl být na jistém vyšším stupni abstrakce. Ke zdůraznění tohoto rozdílu využívá Chikofsky označení obnovení návrhu (design recovery) a obnovení dokumentace (redocumentation). Obnovením dokumentace se míní například přeformátování zdrojového kódu do upořádaného tvaru (konstrukce vývojových diagramů aj.). Obnovení návrhu se od obnovení dokumentace liší vyšším stupněm abstrakce – výsledkem je pouze popis systému.[4]

Reverzního inženýrství je také často využito během změn už používaného systému. Zde se opět rozlišují dva druhy činností a to restrukturalizace (restructuring) a reinženýrství (reengineering). První jmenovaný se týká transformace systému z jedné reprezentace do jiné při zachování stejné úrovně abstrakce. Reinženýrství pak znamená skutečnou změnu daného systému (či dokumentace).[4]

Restrukturalizace i reverzní inženýrství mohou být prováděny manuálně, jde však o obtížnou a časově velmi náročnou proceduru. Proto bylo vyvinuto množství nejrůznějších nástrojů, aby pomohli zautomatizovat tento proces (procesy). Nicméně i zde platí pravidlo, že ne vždy můžeme použít cokoli na cokoli. Lidský element tedy zůstává i nadále nutnou součástí těchto procesů.[4]



Obrázek 3.1 – Chikofského model [4]

4 NÁSTROJE PRO PRÁCI S UML (CASE NÁSTROJE)

Computer-Aided Software Engineering, zkráceně CASE, v překladu znamená počítačem podporované softwarové inženýrství. Jiným slovem se dá říci, že se jedná o vývoj softwaru za využití počítačové podpory. CASE nástroje potom umožňuji modelování žádaného systému za pomoci nejrůznějších diagramů, které většina lidí chápe srozumitelněji než psané slovo. Některé dále umožňují generování zdrojového kódu z modelu (ušetření práce programátorům), zpětnou tvorbu modelu z už existujícího zdrojového kódu (reverzní inženýrství), synchronizaci modelu a zdrojového kódu (tzv. round-trip) či vytvoření jednoduché dokumentace modelu. Tyto nástroje jsou většinou postaveny tak, aby umožňovaly týmovou práci na projektu (sdílení rozpracovaných částí, správa vývoje. Automatizace některých procesů, dodržování zvolených metodik).[24]

4.1 Současnost

V současné době se využívá nespočet rozličných CASE nástrojů. Nepomáhají pouze vývojářům, ale své uplatnění nacházejí například v procesu řízení firmy, kdy se zanalyzují stávající firemní procesy a za pomoci CASE nástrojů se tyto vymodelují tak, jak by měly tyto procesy správně fungovat.[24]

U většiny současných nástrojů je jádrem univerzální modelovací jazyk UML. V grafickém rozhraní je možno modelovat nejrůznější typy diagramů (diagramy tříd, stavové diagramy, …). Důvodem pro vznik těchto nástrojů je právě tato podpora, která při vyšší složitosti systému už nelze provádět za pomoci pouhé tužky a kusu papíru.[24]

Při podpoře návrhu a vývoje softwarové aplikace (za pomoci některého z CASE nástrojů) je opět nutno se řídit určitými pravidly a postupy, které zaručí, že náš projekt nebude odsouzen k záhubě hned na počátku. Takové souhrny postupů (metodiky) nám přesně říkají kdy a co řešit. Podpora metodiky prostředkem CASE vyžaduje, aby tento prostředek pokrýval všechny fáze životního cyklu projektu. Tyto prostředky se pak nazývají integrovaná prostředí pro vývoj aplikací. Přičemž jejich hlavní charakteristikou je pevná svázanost s konkrétní metodikou. Tyto nástroje mají pak výhodu integrity všech metod, technik a nástrojů v kontextu celého životního cyklu projektu.[24]

4.2 Komponenty CASE nástrojů

CASE nástroje se dělí podle jejich funkcí a vlastností. Správný nástroj pak vybíráme dle našich požadavků na tyto komponenty. Takovými komponentami jsou:[24]

- Konzistence GUI (jednotný vzhled obrazovek, popisků, tlačítek, jednotné ovládání, …)
- Jednotná (centrální) databáze pro veškeré informace o objektech, které jsou v systému použity (zaručení jednotných informací v kterémkoli kroku vývoje)
- Prostředky verifikace konzistentnosti dat a podpora normalizace dat (textový editor pro popis jednotlivých objektů - pro účely technické a uživatelské dokumentace systému, či možnost jejího přímého generování ze systému)
- Možnost rychlého návrhu uživatelských obrazovek (včetně vstupů a výstupů)
- Generátor zdrojových programů (pro opakované použití daného kódu)
- Export a import dat (pro práci s modely či dokumentací, které mohly být vytvořeny v externích nástrojích)

4.3 Hlavní přínosy CASE nástrojů

Nejpodstatnější výhody CASE nástrojů můžeme shrnout do několika bodů:[24]

- Vyšší produktivita práce (časová úspora)
- Nižší chybovost (přehlednost)
- Snadná údržba a další vývoj (provázanost)
- Kvalitnější dokumentace (možnost její automatické generace)
- Umožnění týmové spolupráce (univerzálnost)

II. PRAKTICKÁ ČÁST

5 TESTOVÁNÍ

5.1 Co se testuje

CASE nástroje nebudou srovnávány jako celky, nýbrž primárně ze dvou pohledů.

Prvním pohledem bude schopnost daného CASE nástroje produkovat programový kód z předem vymodelovaných diagramů (diagramy tříd, stavové diagramy, …). Hodnocena pak bude kvalita (úplnost) výstupního kódu a jeho rozsáhlost (zda jsou generovány pouze hlavičky tříd, či dokáže nástroj generovat i obsah nejrůznějších metod). Hodnocen bude také import potřebných knihoven či zpracování poznámek z modelu.

Druhý pohled zahrnuje tzv. reverzní inženýrství neboli schopnost CASE nástroje, která umožňuje z už napsaného kódu generovat samotné diagramy či programovou dokumentaci. Zde bude hodnocena opět kvalita vytvořených diagramů, jejich typ, přehlednost a práce s nimi.

Jako doplnění pak bude hodnocen i komfort při modelování jednotlivých diagramů, celková srozumitelnost GUI a svižnost celého nástroje.

Hodnocení těchto pohledů bude uvedeno v podobě autorových poznatků během testování u jednotlivých nástrojů.

U každého testu bude také uveden krátký závěr a celkové hodnocení všech tří pohledů. Na konci práce pak bude uvedena souhrnná tabulka s podrobným hodnocením vlastností všech CASE nástrojů.

5.2 Testované nástroje

CASE nástrojů je obrovské množství. Tato práce se však zabývá srovnáním výběru pouze 6-ti komerčních a 6-ti volně dostupných nástrojů. Těmito nástroji jsou:

Komerční CASE nástroje:

- Sparx Enterprise Architect
- Visual Paradigm for UML
- MagicDraw UML
- Poseidon for UML
- IBM Rational Software Architect
- SmartDraw

Volně dostupné CASE nástroje:

- ArgoUML
- BOUML
- StarUML (Eclipse je testováno v rámci Eclipse UML2Tools)
- Eclipse UML2Tools
- Eclipse PyUML
- UML Designer

5.3 Testovací příklad

Ačkoli je celá řada programovacích jazyků, které CASE nástroje nativně podporují, testování bude pro jednoduchost probíhat pouze v jazyce Java SE (především verze 1.5). Tento jazyk je v současnosti jedním z nejrozšířenějších objektově-orientovaných programovacích jazyků. Zároveň má autor této práce nejvíc praktických zkušeností právě s Javou.

Testovacím příkladem byl zvolen jednoduchý program pro seřazení jednotlivých řádků v CSV souboru dle zadaných kritérií. Program pracuje na úrovní příkazové řádky (zde se i zadávají parametry pro nastavení třízení).

Program samotný se skládá ze dvou balíčků (packages). První package se jmenuje input a obsahuje pouze jednu třídu - SortCSV. Tato třída spouští samotný program. To znamená, že zpracuje vstupní parametry a vytvoří instanci virtuálního CSV souboru. Této instanci nastaví zpracované parametry, vloží do ní řádky z fyzického souboru, spustí třízení a výstup vypíše do souboru (či do konzole -> záleží na nastavení). Druhý package s názvem csv pak obsahuje dvě třídy - třídu CSV, jež reprezentuje virtuální CSV soubor a třídu RadekCSV, které reprezentuje jeden řádek CSV souboru. Třída CSV zároveň obsahuje vnitřní třídu typu enum. Tento balíček a hlavně třída RadekCSV bude primárně sledována při generování zdrojového kódu.

Kompletní zdrojové kódy jsou uvedeny na konci práce (ZK I, ZK II, ZK III).

5.4 Testovací podmínky

Testování probíhá na laptopu HP Compaq 6715b (AMD Sempron 3600+ 2.00GHz, 2GB RAM, 80GB HDD). Jeho softwarové vybavení pak tvoří Microsoft Windows Vista Business SP1 EN 32bit, Microsoft Internet Explorer 7.0 EN, Mozilla Firefox 3.0.8 EN, Eclipse SDK 3.2.2 EN, JRE (1.6.0_13) a JDK (1.6.0_07).

6 KOMERČNÍ CASE NÁSTROJE

Testy vybraných komerčních nástrojů.

6.1 Sparx Enterprise Architect

6.1.1 Základní informace

Název: Enterprise Architect 7.5
Verze: 7.5.843, Professional
Autor: Geoffrey Sparks (Sparx Systems)
Licence: Trial (30 dní)
Stažení: http://www.sparxsystems.com.au/bin/easetup.exe

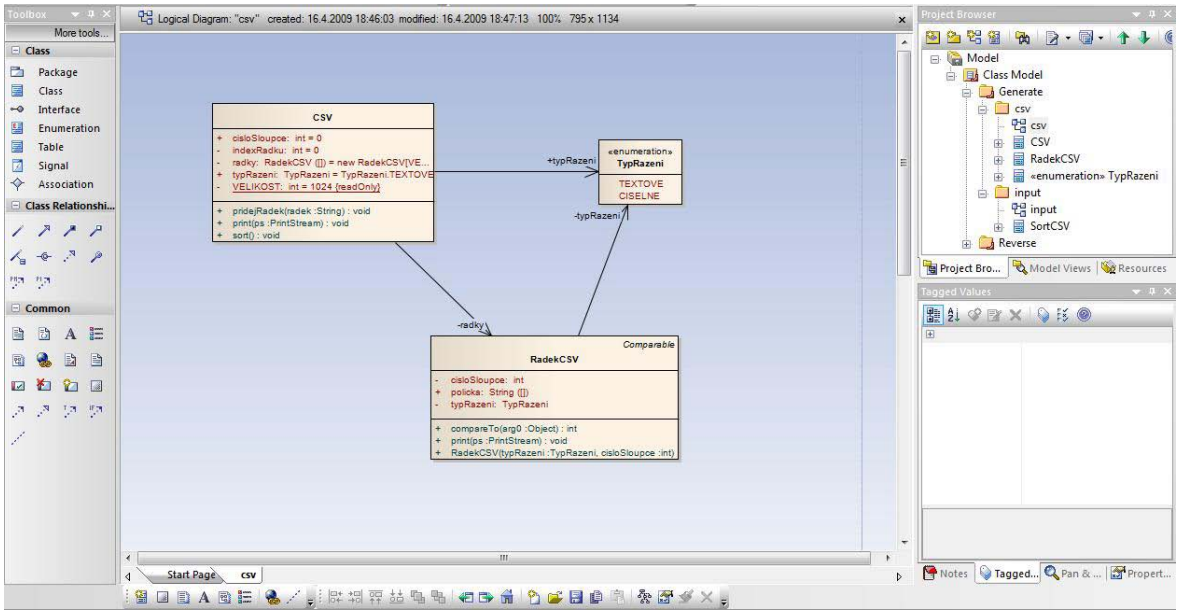
Sparx Enterprise Architect je vyvíjen australskou firmou Sparx Systems, která na trhu působí od roku 1996. Tato firma se specializuje především na vývoj nástrojů pro modelování, plánování, design a tvorbu softwarových systému.[16]

Jedním z takových nástrojů je právě i Enterprise Architect (EA). EA je nástrojem především pro UML modelování, která v poslední verzi 7.5 plně podporuje UML 2.1. Jeho rozsah pokrývá životní cyklus projektu už od zpracování základních požadavků zadavatele, přes dodatečnou úpravu, až po generování zdrojového kódu či dokumentace k modelům. Zároveň poskytuje možnosti spolupráce celého týmu v rámci jednoho nástroje.[16]

6.1.2 Schopnost generování zdrojového kódu

EA, ačkoli je velmi robustním nástrojem pro modelování, tak do verze 7.5 stále nepodporuje generování zdrojového kódu ze stavových diagramů či diagramů aktivit. Uživatel si proto musí vystačit pouze s diagramem tříd. I přes tento nedostatek má však EA v tomto směru stále co nabídnout.

Při vytváření diagramů se EA uživateli snaží jeho práci všemožně usnadnit. Proto při přidávání metod či atributů člověk vyplňuje jen patřičná okna a položky. To se může zdát zdlouhavé, avšak výhodu to přináší především v přehlednosti a pořádku ve všech vložených informacích (nehledě na jednoduchost pozdějšího dohledání). Na základě takto vložených údajů pak EA modeluje daný diagram podle svých vlastních konvencí. Ty jsou, ostatně jako celé GUI programu, přehledně zpracované (Obrázek 6.1).



Obrázek 6.1 – Prostředí tvorby diagramu tříd v EA

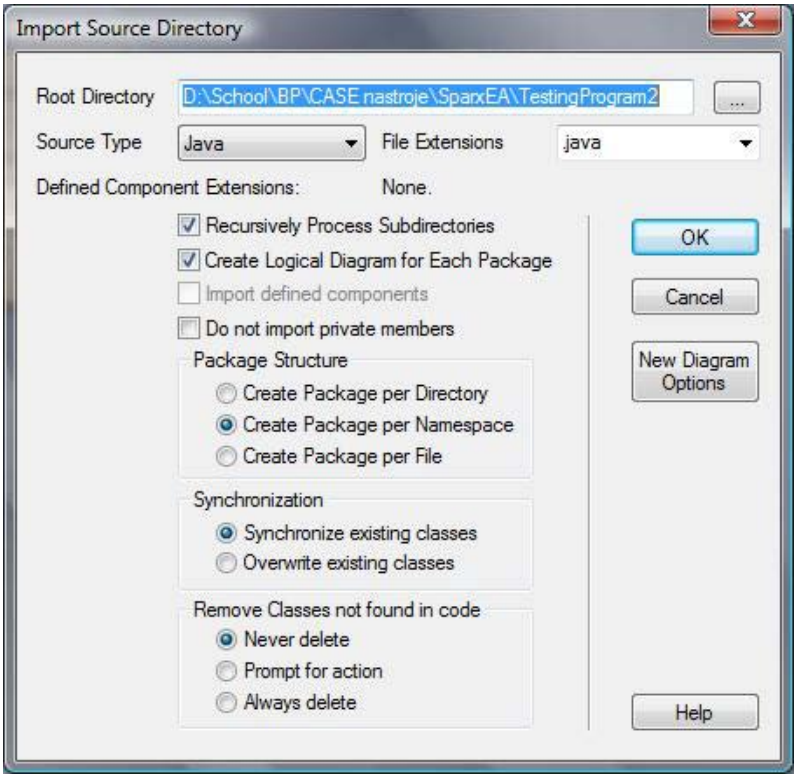
Při generování zdrojového kódu jsou výstupem jednotlivé Java soubory, které obsahují veškeré údaje vložené do modelu. Vytvoří se tedy kostra dané třídy, nadefinují se metody i atributy (včetně případných inicializačních hodnot). EA navíc v každé třídě generuje automaticky její konstruktor a také destruktor (ten však není v případě Javy úplně nutný). Dále dokáže z modelu exportovat komentáře ke všem prvkům dané třídy. Bohužel však pokulhává v importu standardních Java packages jakými jsou třeba java.io atd., které jsou součástí JDK (Java Development Kit). Tento neduh však dokáže vyřešit většina současných vývojářských nástrojů (Eclipse, NetBeans, …) na pár kliknutí. Výsledný zdrojový kód jedné z generovaných tříd znázorňuje obrázek (Obrázek 6.2).

```
package csv;

/**
 * Trida reprezentujici jeden radek v CSV objektu
 * @author eNko
 * @version 1.0
 * @created 16-IV-2009 17:38:12
 */
public class RadekCSV implements Comparable {

    private int cisloSloupce;
    public String policka[];
    private TypRazeni typRazeni;

    public RadekCSV(){

    }

    public void finalize() throws Throwable {

    }

    /**
     * Kontruktor - nastavi zpusob razeni a cislo sloupce, podle ktereho se bude
     * tridit
     *
     * @param typRazeni
     * @param cisloSloupce    cisloSloupce
     */
    public RadekCSV(TypRazeni typRazeni, int cisloSloupce){

    }

    /**
     * Porovnani 2 radku
     *
     * @param arg0
     */
    public int compareTo(Object arg0){
        return 0;
    }

    /**
     * Vypis radku na vystup
     *
     * @param ps    ps
     */
    public void print(PrintStream ps){

    }

}
```

Obrázek 6.2 – Kód vygenerovaný nástrojem Sparx Enterprise Architect

6.1.3 Schopnost reverzního inženýrství

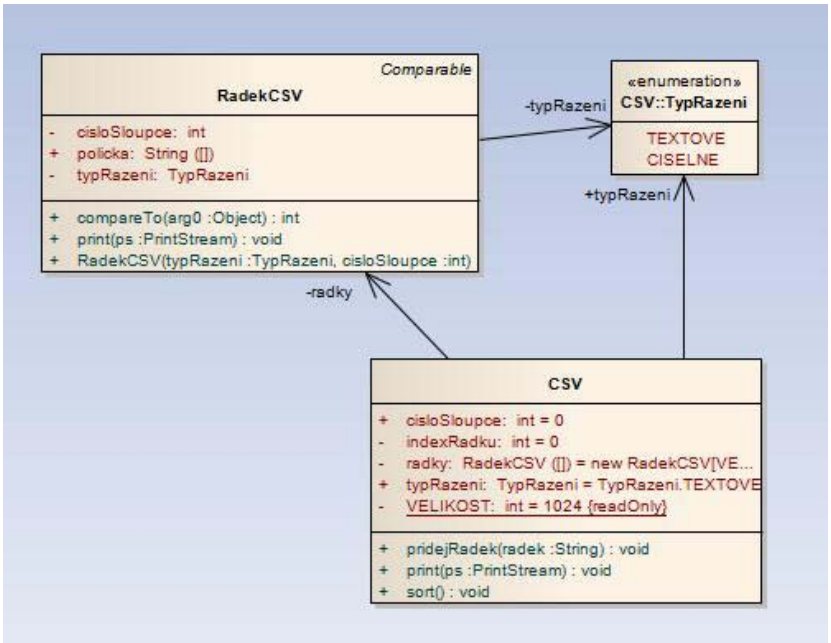
Při reverzním inženýrství máme v EA dva možné postupy. Prvním z nich je importování celé složky (package) se zdrojovými kódy. V tomto případě EA zachová strukturu package a vytvoří diagramy tříd dle zjištěných údajů (obsah jednotlivých tříd, jejich propojení atd.). Dialogové okno tohoto postupu znázorňuje obrázek (Obrázek 6.3). Je zde na výběr typ zdrojového kódu a některé další možnosti způsobu jeho importu.



Obrázek 6.3 – Dialogové okno při reverzním inženýrství v EA

V druhém případě EA umožňuje import jednotlivých Java souborů, kdy pouze vytvoří diagram třídy a jejího obsahu (čili bez návaznosti na třídy okolní).

V obou případech je obsah importu přesný a správný – standardně EA importuje názvy tříd, metod a atributů včetně jejich inicializačních hodnot. Bohužel chybí propojení mezi některými třídami. To je způsobeno tím, že EA se nezabývá lokálními proměnnými v metodách, a proto se nástroj řídí jen obsahem globálních parametrů. EA také zvládá import veškerých komentářů uvedených v kódu.



Obrázek 6.4 – Importovaný package csv do prostředí EA

6.1.4 Hodnocení

Sparx Enterprise Architect	
Svižnost a přehlednost u tohoto nástroje je opravdu dobrá. Generování kódu v tomto nástroji je sice do značné míry rychlé a přehledné, bohužel však nedokáže generovat kód z jiných než diagramů tříd. Reverzní inženýrství generuje přehledné a bezchybné diagramy. Na škodu je absence detekce lokálních proměnných.	
Generování kódu	●●○○○
Reverzní inženýrství	●●○○○
Práce s nástrojem	●●●●○

Tabulka 6.1 – Hodnocení Sparx Enterprise Architect

6.2 Visual Paradign for UML

6.2.1 Základní informace

Název: Visual Paradigm for UML
Verze: 6.4 (build sp2_20090212), Enterprise Edition
Autor: Visual Paradigm International
Licence: Evaluation Copy (30 dní)
Stažení: http://www.visual-paradigm.com/download/ (vyžaduje registraci)

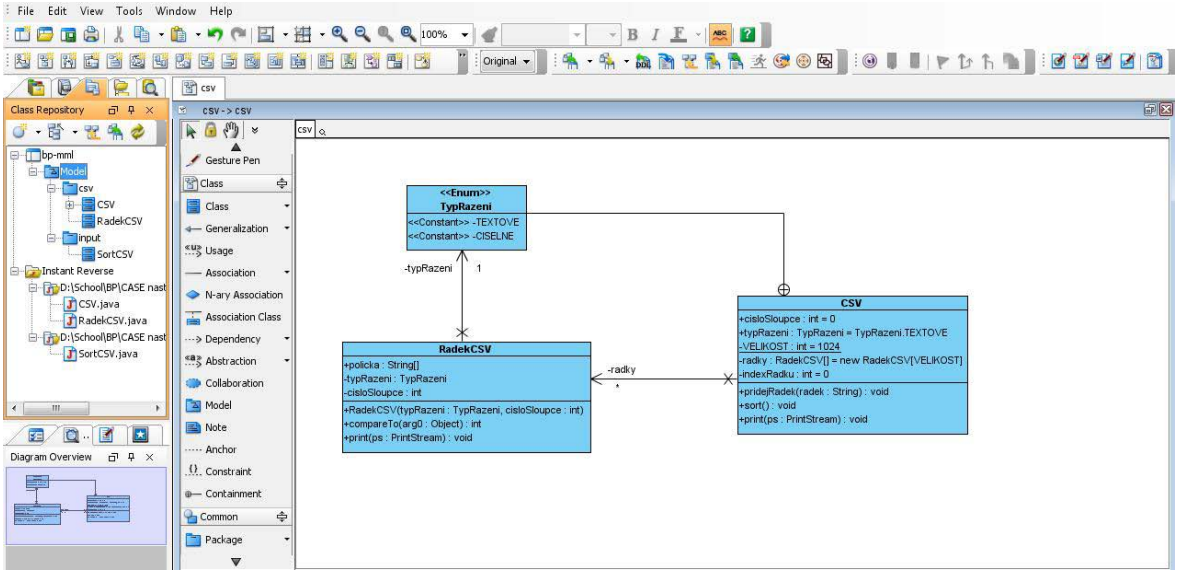
Visual Paradigm for UML (VP) pochází z dílny firmy Visual Paradigm International. Tato firma se sídlem v Hong Kongu se zabývá hlavně vývojem nejrůznějších řešení pro firmy, které chtějí vyšší produktivitu za méně peněz.[19]

Jedním takovým řešením je i modelovací nástroj Visual Paradigm, ve kterém je možnost vytváření jak kompletních firemních procesů, tak i modelování nejrůznějších aplikací. V našem případě jde tedy i o schopnost generování zdrojového kódu či možnost reverzního inženýrství. Bohužel stejně jako firma Sparx Systems (Sparx Enterprise Architect) i Visual Paradigm International na svých stránkách uvádí, že jejich nástroj v těchto směrech pracuje pouze s diagramy tříd.[19]

6.2.2 Schopnost generování zdrojového kódu

Prostředí VP už při prvním spuštění dává díky mnoha ikonám a tlačítkům jasně najevo, že se jedná o další z robustních modelovacích nástrojů. Bohužel při běhu je nástroj v některých chvílích poněkud pomalejší.

Při modelování uživatel standardně vybere typ diagramu, který chce modelovat a může okamžitě začít pracovat. Samotné modelování probíhá jako u předchozího nástroje – jednoduchá nabídka se základními prvky po levé straně pracovního okna (Obrázek 6.5) a následné vkládání všech údajů o metodách a atributech skrze příslušná okna a kolonky. VP si pak sám modeluje grafický model celé třídy. Zajímavým „pomocníkem“ je možnost modelování pomocí gestikulace myší (kreslení), pro které však uživatel potřebuje notnou dávku cviku.



Obrázek 6.5 – Visual Paradigm for UML GUI

Po dokončení modelu (v našem případě diagramu třídy) je v tomto nástroji možnost vygenerování zdrojového kódu. Po vyvolání menu pro tuto akci se zobrazí přehledné okno s možností dodatečného nastavení – po výběru jazyka se zobrazí příslušné nastavení (v případě Javy je to například verze JDK, možnost nastavení prefixu pro atributy a argumenty, …).

Kód samotný je úplný dle diagramu tříd (Obrázek 6.6). Milým překvapením je import potřebných JDK knihoven. Import dalších tříd však není uveden v hlavičce dané třídy, ale přímo v deklaraci proměnných (typy jsou uvedeny ve formě kompletních cest). Konstruktor a destruktor se generuje jen v případě, že je uveden i ve vytvořeném diagramu. Nástroj také vkládá do všech metod generování výjimky - upozornění, že metoda zatím nemá svůj obsah. Bohužel chybí implementace rozhraní, kterou se mi nepodařilo vložit ani do původního modelu.

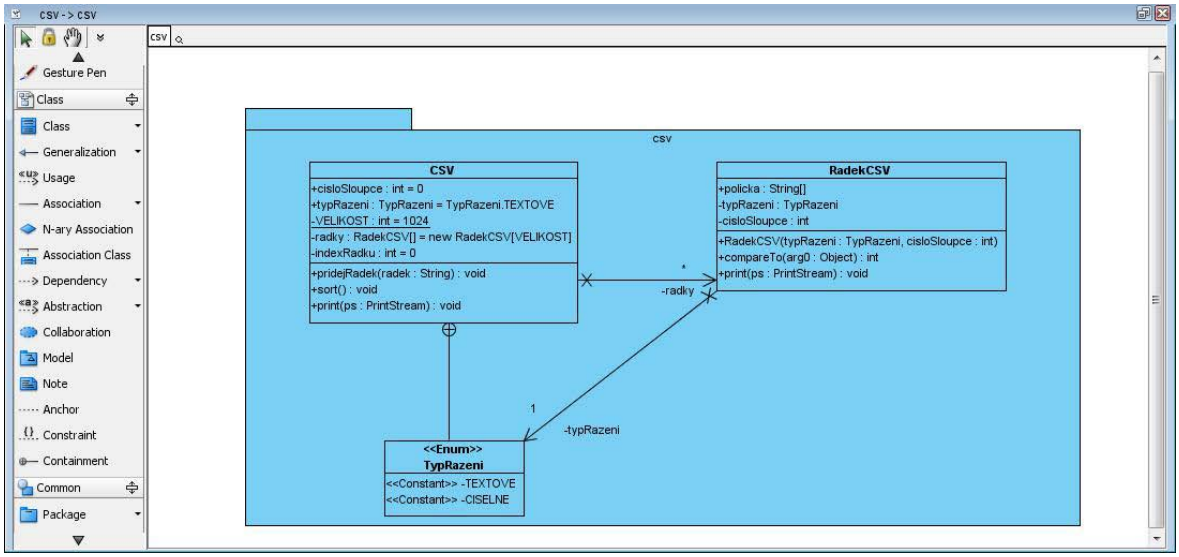
```
package csv;

import java.io.PrintStream;

/**
 * Trida reprezentujici jeden radek v CSV objektu
 */
public class RadekCSV {
    public String[] policka;
    private csv.CSV.TypRazeni typRazeni;
    private int cisloSloupce;

    /**
     * Kontruktor - nastavi zpusob razeni a cislo sloupce, podle ktereho se bude
     * tridit
     *
     * @param typRazeni
     * @param cisloSloupce
     */
    public RadekCSV(csv.CSV.TypRazeni typRazeni, int cisloSloupce) {
        throw new UnsupportedOperationException();
    }

    /**
     * Porovnani 2 radku
     */
    public int compareTo(Object arg0) {
        throw new UnsupportedOperationException();
    }

    /**
     * Vypis radku na vystup
     *
     * @param ps
     */
    public void print(PrintStream ps) {
        throw new UnsupportedOperationException();
    }
}
```

Obrázek 6.6 – Zdrojový kód třídy RadekCSV vygenerovaný pomocí VP

6.2.3 Schopnost reverzního inženýrství

VP podporuje reverzní inženýrství pouze v generování diagramů tříd. Tato funkce je v tomto nástroji stejně jednoduchá jako generování zdrojových kódu. Uživatel v tomto směru pouze vybere složky se zdrojovými kódy (či celé packages). Hierarchie těchto složek je mu pak po celou dobu k dispozici v levé části nástroje (Obrázek 6.5). Kdykoliv si pak uživatel může nechat vygenerovat potřebný diagram tříd a ten si importovat do zvoleného místa ve svém modelu.

Vytvořený model obsahuje veškeré informace ze zdrojových souborů (názvy metod a atributů včetně jejich typů, propojení mezi třídami, komentáře, …). Bohužel však opět chybí procházení obsahu metod. Chybí tedy asociace mezi třídami v rámci lokálních proměnných. VP dokáže rozpoznat násobnost propojení mezi třídami a tu pak zobrazit přímo v diagramu. Dalším nedostatkem je opomenutí znázornění implementace rozhraní.



Obrázek 6.7 – Výsledek reverzního inženýrství na package csv

6.2.4 Hodnocení

Visual Paradigm for UML	
Tento nástroj je velmi robustní, bohužel však na úkor svižnosti a přehlednosti. Generování bohužel trpí nedostatkem v podobě generování zdrojového kódu pouze z diagramů tříd. Naopak výborná je spolupráce s JDK. Reverzní inženýrství přehledné a rychlé - bohužel také jen s podporou jednoho typu diagramu.	
Generování kódu	●●○○○
Reverzní inženýrství	●●○○○
Práce s nástrojem	●●◎○○

Tabulka 6.2 – Hodnocení Visual Paradigm for UML

6.3 MagicDraw UML

6.3.1 Základní informace

Název: MagicDraw UML

Verze: 16.0 Patch 1, verze Enterprise

Autor: No Magic, Inc.

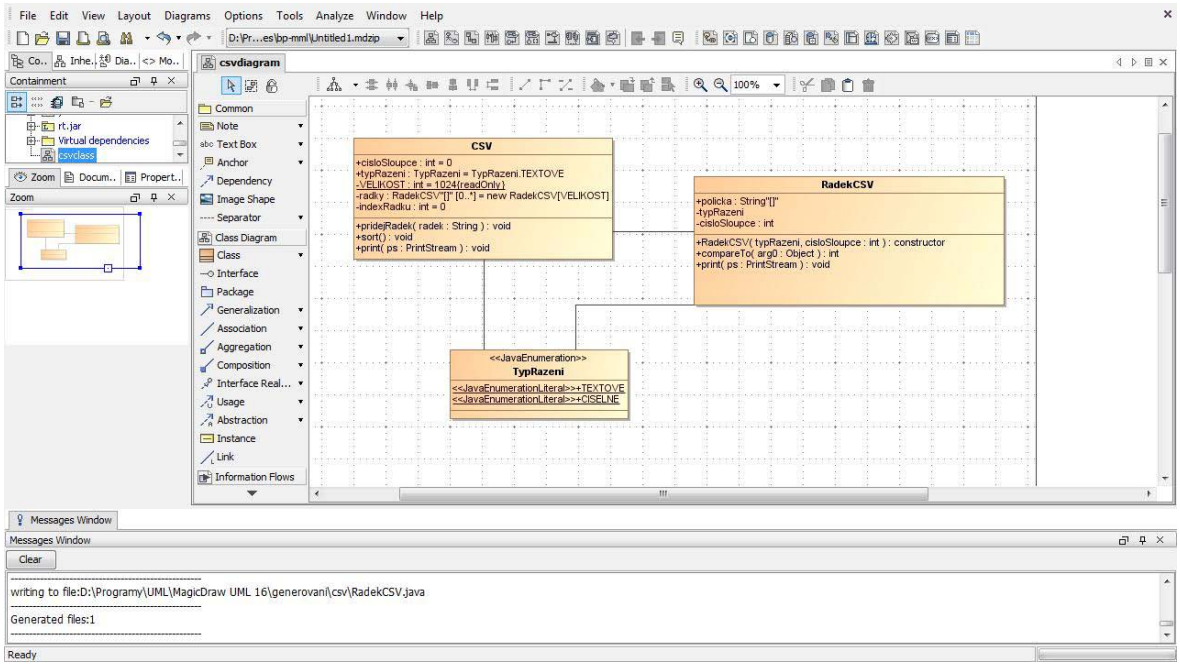
Licence: Evaluation Demo (do 1. července 2009)

Stažení: http://www.magicdraw.com/ (vyžaduje registraci)

Dalším z řady komerčních nástrojů pro modelování je MagicDraw UML. Tento nástroj je schopen modelovat jak business procesy, tak i nejrůznější systémové požadavky či SW návrhy.[13]

6.3.2 Schopnost generování zdrojového kódu

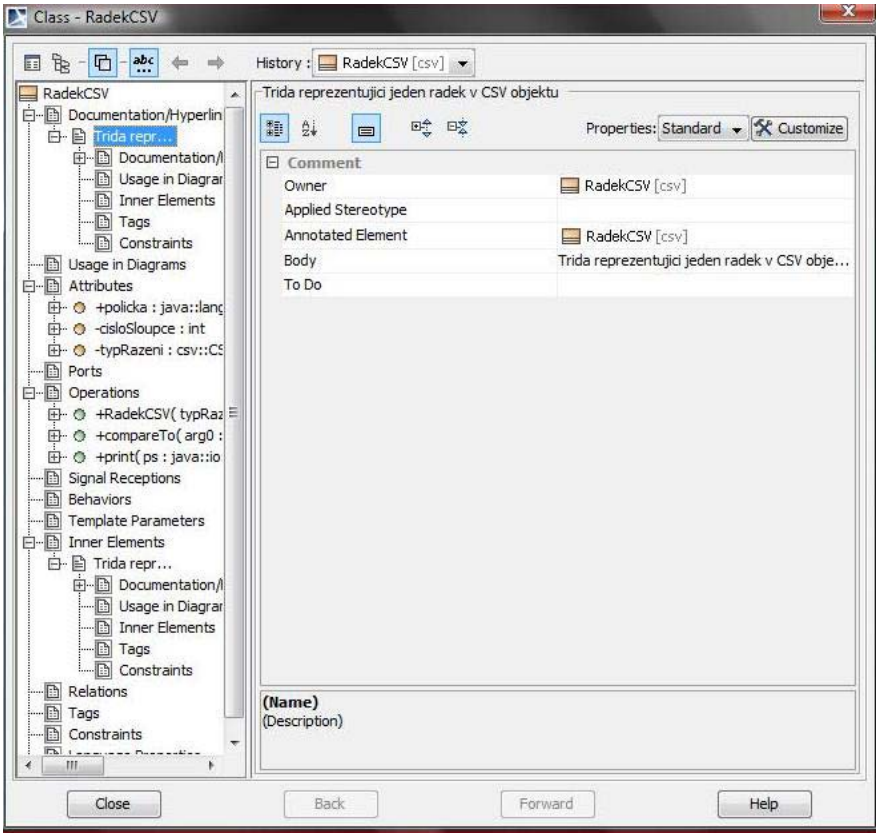
MagicDraw sice nabídku schopnosti generování zdrojového kódu nabízí u většiny z diagramů. Při poklikání na ni se však objeví hláška, že není co generovat. Výjimku tvoří pouze diagram tříd.



Obrázek 6.8 – Prostředí MagicDraw + vytvořený diagram tříd

Uživatelské prostředí tohoto nástroje je příjemné. Lišta nástrojů obsahuje pouze tlačítka pro tvorbu různých diagramů a pro standardní práci s nimi. Vše je velmi intuitivní a přehledně umístěné. Robustnost nástroje je úměrná jiným komerčním nástrojům.

Při modelování přidáváme jednotlivé třídy na pracovní plochu. Pokud modeluje pokročilejší uživatel, může psát názvy atributů, metod a jejich dodatečné parametry přímo do grafických tříd na pracovní ploše. Pokud však uživatel není natolik pokročilý, či chce jen přidat více informací, je zde možnost otevření vlastní specifikace ke každému prvku dané třídy. V této specifikaci pak může zadávat dodatečné parametry k vybranému prvku (Obrázek 6.9). Množství a typ těchto parametrů se liší podle typu prvku (například u metod je nabídka logicky jiná než u atributů). Rozsáhlost těchto specifikací lze částečně nastavit.



Obrázek 6.9 – Specifikace jedné z tříd

Z vytvořeného diagramu tříd (Obrázek 6.8) se pomocí několika kliknutí dá vygenerovat samotný zdrojový kód (Obrázek 6.10).

```
* @(#) RadekCSV.java

package csv;

import csv.CSV.TypRazeni;

/**
 * Trida reprezentujici jeden radek v CSV objektu
 *
 */
public class RadekCSV implements Comparable
{
    public String[] policka;

    private TypRazeni typRazeni;

    private int cisloSloupce;

    /**
     * Kontruktor - nastavi zpusob razeni a cislo sloupce, podle ktereho se bude
     * tridit
     *
     * @param typRazeni
     * @param cisloSloupce
     */
    public RadekCSV( TypRazeni typRazeni, int cisloSloupce )
    {

    }

    /**
     * Porovnani 2 radku
     */
    public int compareTo( Object arg0 )
    {
        return 0;
    }

    /**
     * Vypis radku na vystup
     *
     * @param ps
     */
    public void print( PrintStream ps )
    {

    }


}
```

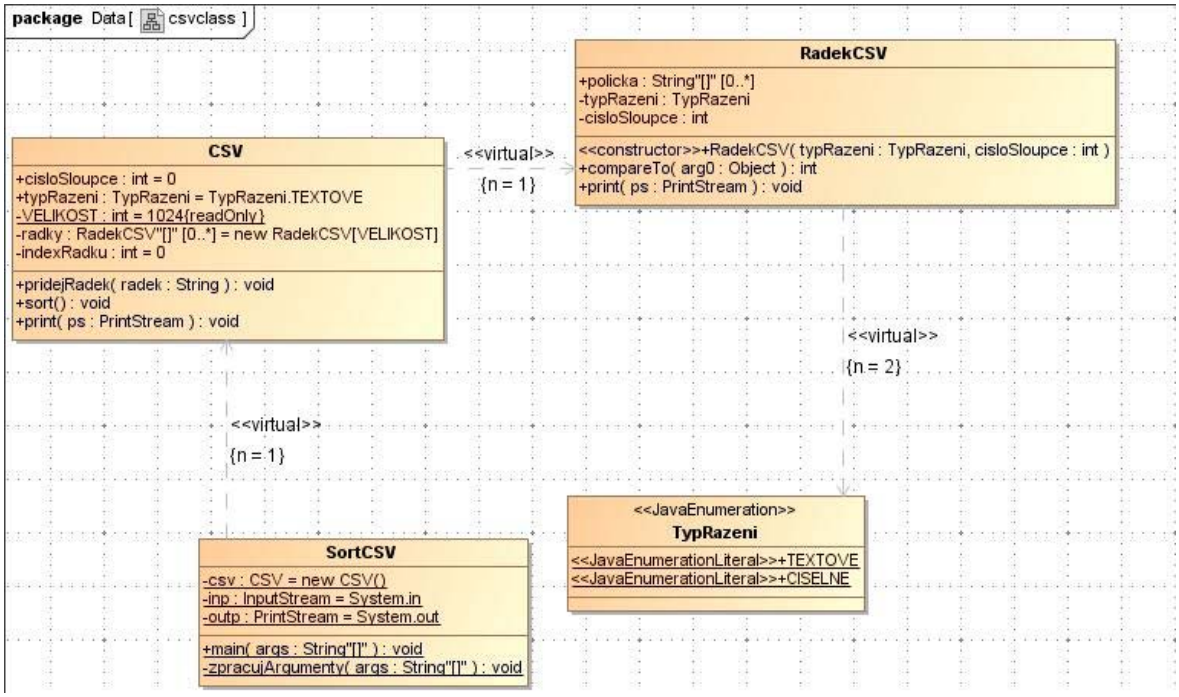
Obrázek 6.10 – Kód vygenerovaný nástrojem MagicDraw

Takový kód jako vždy obsahuje název třídy, deklaraci globálních atributů a všech metod včetně parametrů. Navíc vygeneruje i veškeré komentáře (pokud byly vloženy do diagramu). Správně MagicDraw identifikoval i packages, které jsou ve třídě nutné (včetně standardních JDK packages).

Bohužel MagicDraw, jak se zdá, podporuje generování kódu pouze z diagramu tříd. Musíme se tedy spokojit pouze s kostrou vygenerovaných tříd. Obsahy všech metod již musí vývojáři dopsat sami.

6.3.3 Schopnost reverzního inženýrství

MagicDraw nabízí možnost reverzního inženýrství již při vytváření nového projektu, kdy stačí pouze vybrat kořenový adresář zdrojových souborů a pomocí rekurzivního procházení přidat veškeré třídy.

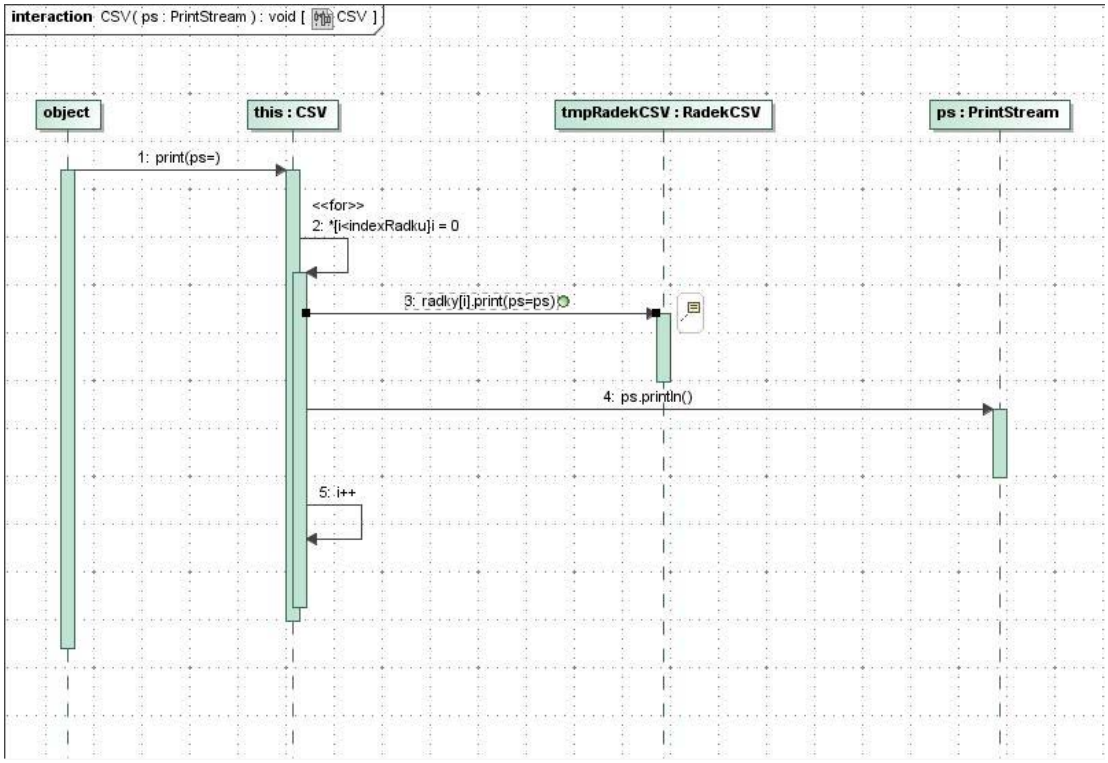


Obrázek 6.11 – Reverzní inženýrství – diagram tříd

Z takto vybraných souborů se pak vygeneruje diagram tříd (Obrázek 6.11). Tento diagram standardně obsahuje několik tříd, které jsou vzájemně propojeny závislostmi a obsahují výčet metod a atributů. Konstruktor dané třídy je jasně označen prefixem <<constructor>>.

Ve specifikaci dané třídy lze pak nalézt i komentáře k jednotlivým prvkům třídy, jejich viditelnost či inicializační hodnoty. Bohužel chybí extrakce poznámky v hlavičce třídy.

Jako jeden z mála podporuje MagicDraw i generování sekvenčních diagramů. Stačí vybrat některou z metod vygenerované třídy a v místní nabídce zvolit reverzní implementaci. Výsledný diagram takového postupu znázorňuje obrázek (Obrázek 6.12). V tomto diagramu MagicDraw správně označil posloupnost všech kroků i třídy, se kterými metoda pracuje (včetně standardních knihoven JDK).



Obrázek 6.12 – Reverzní inženýrství – sekvenční diagram

Toto je však z reverzního inženýrství vše, co nám MagicDraw v této verzi může nabídnout. I tak ale mnohé z ostatních nástrojů předčí právě v možnosti generování sekvenčních diagramů.

6.3.4 Hodnocení

MagicDraw UML	
MagicDraw je i přes svou robustnost přehledným a svižným nástrojem. V generování zdrojového kódu podporuje pouze diagramy tříd. V reverzním inženýrství však získává body navíc za schopnost generování sekvenčních diagramů.	
Generování kódu	●●○○○
Reverzní inženýrství	●●●○○
Práce s nástrojem	●●●⊙○

Tabulka 6.3 – Hodnocení MagicDraw UML

6.4 Poseidon for UML

6.4.1 Základní informace

Název: Poseidon for UML

Verze: 6.0.2-0, Professional Edition

Autor: Gentleware AD

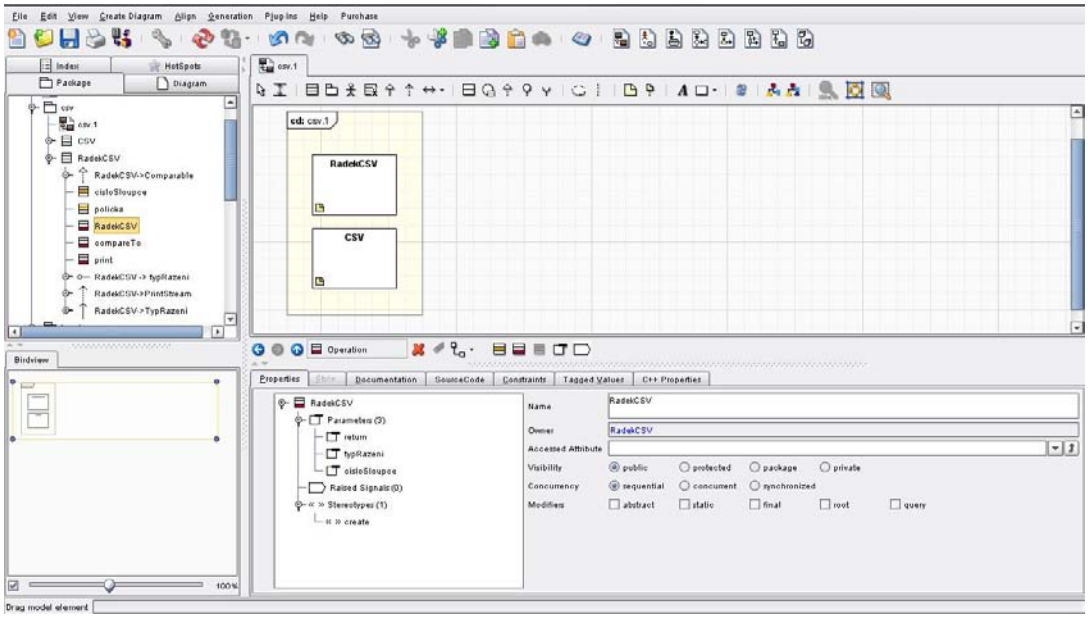
Licence: Evaluation License (30 dní)

Stažení: http://www.gentleware.com/uml-software-pe.html (vyžaduje vyplnění dotazníku)

Poseidon for UML vyvinula německá firma Gentleware, která byla založena roku 2004 v Hamburku. Zakladatelé této společnosti jsou podepsání i pod, tentokráte nekomerčním modelovacím nástrojem, ArgoUML. Poseidon for UML je jeden z nejstahovanějších komerčních UML nástrojů (přes milión kopií, které jsou distribuovány ve více než 100 zemích světa).[11]

6.4.2 Schopnost generování zdrojového kódu

Modelování v Poseidonu se trochu liší od ostatních modelovacích nástrojů. V pracovní oblasti diagramu tříd jsou sice znázorněny jednotlivé třídy. Jejich skutečná provázanost a obsah je však znázorněna v levé části GUI nástroje (Obrázek 6.13). Podrobnosti k jednotlivým prvkům jsou umístěny pod pracovní plochou pod jednotlivými kartami.



Obrázek 6.13 – Modelování třídy a jejích atributů

Rozsáhlost údajů, které můžeme do jednotlivých prvků třídy vložit, je průměrná. Vše se děje v jednotlivých kartách, které daný prvek při svém zvolení nabídne (Obrázek 6.13).

Poseidon for UML nabízí generování kódu pouze z diagramu tříd. Čili opět pouze generování koster daných tříd. Tento nedostatek však částečně nahrazuje tím, že při vkládání údajů ohledně jednotlivých metod máme v nabídce i kartu Source Code. Zde si uživatel může prohlédnout kostru metody a případně k ní doplnit i zdrojový kód. Takový kód je pak součásti vygenerovaného zdrojového kódu (Obrázek 6.14).

```
package csv;
import csv.CSV.TypRazeni;
import java.io.PrintStream;

/**
 * Trida reprezentujici jeden radek v CSV objektu
 *
 */
public class RadekCSV implements Comparable {

/**
    public String[] policka;
/**
    private CSV.TypRazeni typRazeni;

/**
    private int cisloSloupce;

/**
 * Kontruktor - nastavi zpusob razeni a cislo sloupce, podle ktereho se bude
 * tridit
 *
 *
 * @param typRazeni
 * @param cisloSloupce
 */
    public  RadekCSV(CSV.TypRazeni typRazeni, int cisloSloupce) {
        this.typRazeni = typRazeni;
        this.cisloSloupce = cisloSloupce;
    }

/**
 * Porovnani 2 radku
 *
 *
 * @return
 * @param arg0
 */
    public int compareTo(Object arg0) {

        return 0;
    }

/**
 * Vypis radku na vystup
 *
 *
 * @param ps
 */
    public void print(java.io.PrintStream ps) {
        // your code here
    }
}
```

VYGENEROVANÝ KÓD

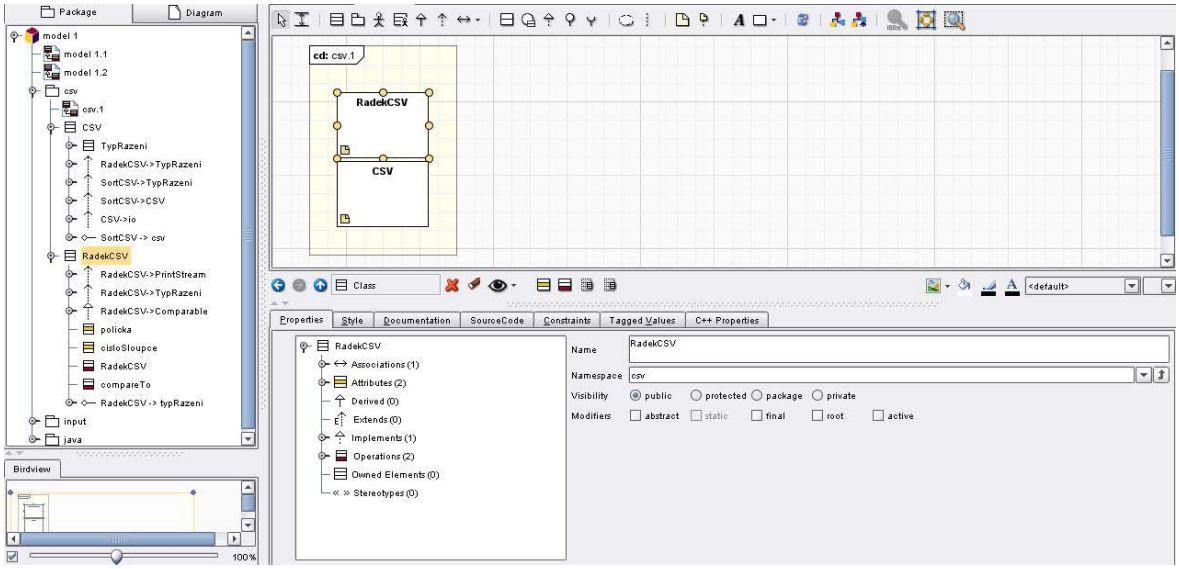
Obrázek 6.14 – Zdrojový kód vytvořený pomocí Poseidon for UML

6.4.3 Schopnost reverzního inženýrství

Poseidon UML umožňuje reverzní inženýrství pouze z Java programů (z Java souborů i JAR souborů – zkompilovaný program). V našem testovacím programu Poseidon při

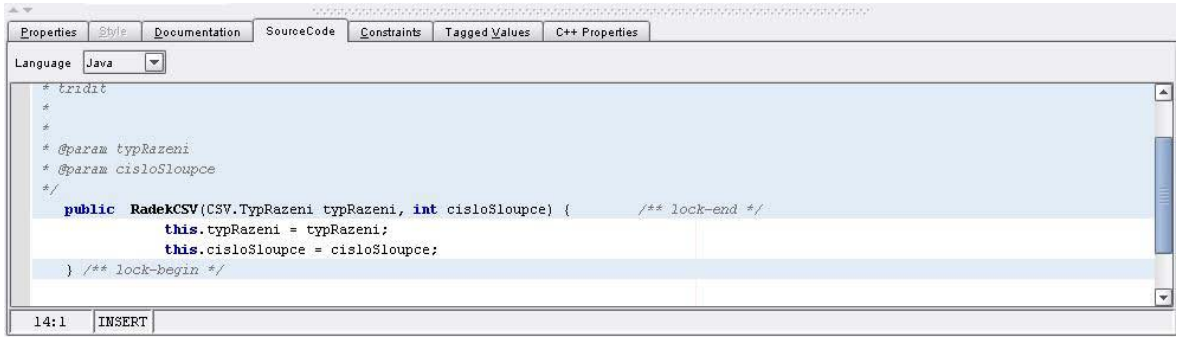
importu upozornil několik „chyb“, které se však při bližším přezkoumání zdrojového kódu projevily jako falešné. Bohužel kvůli této chybě jsou veškeré generované diagramy nekompletní.

Zbylá část importu proběhla bez sebemenších problémů. Obrázek (Obrázek 6.15) znázorňuje v jeho levé části výsledek, který obsahuje hierarchii vygenerovaného modelu. Typicky byly vytvořeny pouze digramy tříd (generování jiných typů diagramů není podporováno).



Obrázek 6.15 – Vygenerovaný diagram včetně popisu atributů a metod

Tento nástroj navíc dokázal importovat i celý zdrojový kód. Tudíž při výběru některé z tříd máte možnost upravovat její zdrojový kód (Obrázek 6.16). Toto může být velmi užitečné při dodatečných úpravách modelů a jejich následném zanesení zpět do kódu (tzv. RoundTrip).



Obrázek 6.16 – Možná úprava kódu po importu zdrojových souborů

6.4.4 Hodnocení

Poseidon for UML	
Poseidon for UML se nad ostatní nástroje jemu podobné řadí hlavně díky možnosti importu zdrojových kódů přímo do diagramu tříd. Bohužel však při reverzním inženýrství vyhazuje nesmyslné chyby. Při generování kódu dokáže generovat pouze z diagramu tříd.	
Generování kódu	●●○○○
Reverzní inženýrství	●●○○○
Práce s nástrojem	●●●●○

Tabulka 6.4 – Hodnocení Poseidon for UML

6.5 IBM Rational Software Architect

6.5.1 Základní informace

Název: Rational® Software Architect™

Verze: 7.5.2, Standard Edition

Autor: IBM corp. and others

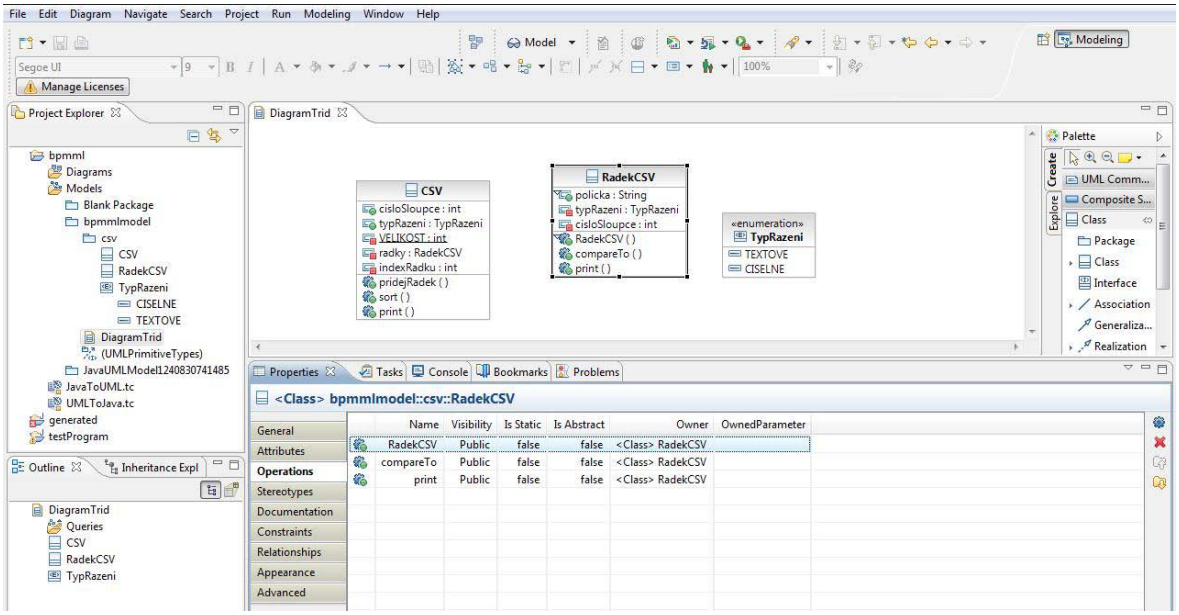
Licence: Trial (30 dní)

Stažení: http://www.ibm.com/developerworks/downloads/r/rsd (vyžaduje IBM download managera)

Rational Software Architect (RSA) je jedním z mnoha produktů, které vyvíjí softwarový gigant IBM. RSA je dalším z řady komerčních CASE nástrojů, které pokrývají celý životní cyklus projektu. Bohužel samo IBM v některých tutoriálech označuje generování zdrojového kódu z diagramů jako „deprecated“ čili zavrhované. Proto se v této oblasti od tohoto nástroje nedá moc očekávat.[12]

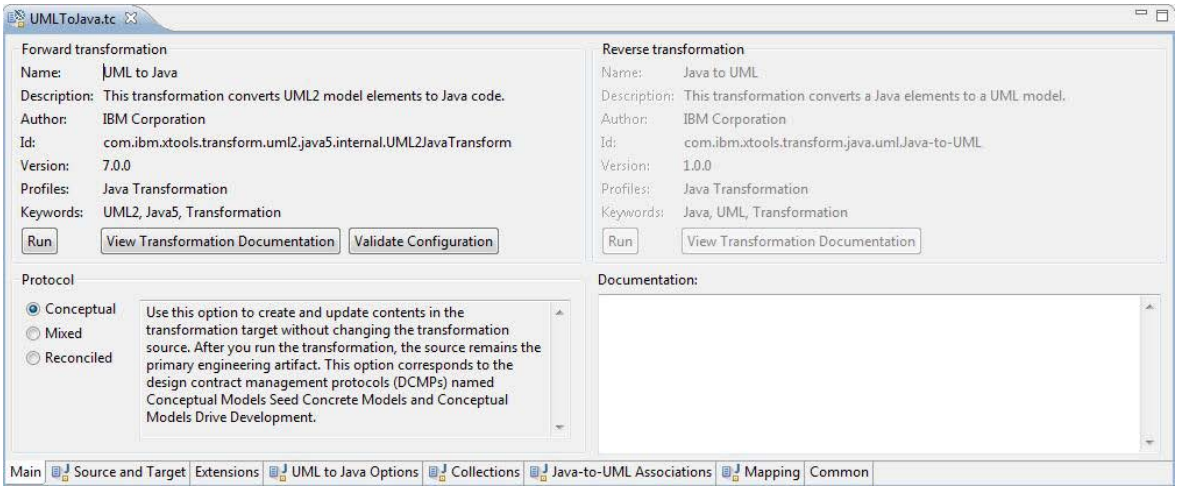
6.5.2 Schopnost generování zdrojového kódu

Již při prvním spuštění RSA se každému, kdo má zkušenosti s programováním v prostředí Eclipse, otevře velmi povědomé GUI (Obrázek 6.17). IBM tímto krokem zřejmě chtělo ušetřit všem vývojářům čas při zvykání si na nové prostředí.



Obrázek 6.17 – Ukázka diagramu tříd a výpisu metod třídy RadekCSV

Modelování v RSA je podobné modelování v jiných CASE nástrojích. Ve složce Diagrams se vytvoří prázdný diagram (v našem případě diagram tříd) a následně se na pracovní plochu umisťují jednotlivé prvky a v okně pod ní se pak vyplňují veškeré další potřebné údaje (Obrázek 6.17). Třídy, které jsou umístěny v diagramu tříd, se zároveň ukládají do složky Models do patřičného package.



Obrázek 6.18 – Nastavení šablony pro generování zdrojového kódu

Pro generování kódu je nutné vytvořit tzv. transformační šablonu (Obrázek 6.18). Taková šablona však není schopná generování zdrojových kódů z jednotlivých diagramů. Lze ji tedy spustit pouze na package ve složce Models. Nastavení transformační šablony zahrnuje zadání zdroje (package ve složce Models) a cíle (nejlépe package v nějakém již

vytvořeném Java projektu). Dále je v případě Javy možnost automatického generování Set a Get metod k jednotlivým atributům a možnost definování základního rozdělení kolekcí.

```
/**
package csv;

import csv.CSV.TypRazeni;
import java.io.PrintStream;

/**
 * <!-- begin-UML-doc -->
 * Trida reprezentujici jeden radek v CSV objektu<br>
 * <!-- end-UML-doc -->
 * @author eNko
 * @generated "UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
 */
public class RadekCSV implements Comparable {
    /**
     * <!-- begin-UML-doc -->
     * <!-- end-UML-doc -->
     * @generated "UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
     */
    public String[] policka = null;
    /**
     * <!-- begin-UML-doc -->
     * <!-- end-UML-doc -->
     * @generated "UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
     */
    private int cisloSloupce;

    /**
     * <!-- begin-UML-doc -->
     * Kontruktor - nastavi zpusob razeni a cislo sloupce, podle ktereho se
     * <!-- end-UML-doc -->
     * @param typRazeni
     * @param cisloSloupce
     * @generated "UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
     */
    public RadekCSV(csv.CSV.TypRazeni typRazeni, int cisloSloupce) {
        // begin-user-code
        // TODO Auto-generated constructor stub
        // end-user-code
    }

    /**
     * <!-- begin-UML-doc -->
     * <!-- end-UML-doc -->
     * @generated "UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
     */
    private TypRazeni typRazeni;

    /**
     * <!-- begin-UML-doc -->
     * Porovnani 2 radku
     * <!-- end-UML-doc -->
     * @param arg0
     * @return
     * @generated "UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
     */
    public int compareTo(Object arg0) {
        // begin-user-code
        // TODO Auto-generated method stub
        return 0;
        // end-user-code
    }

    /**
     * <!-- begin-UML-doc -->
     * Vypis radku na vystup<br><br>@param ps
     * <!-- end-UML-doc -->
     * @param ps
     * @generated "UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
     */
    public void print(PrintStream ps) {
        // begin-user-code
        // TODO Auto-generated method stub

        // end-user-code
    }
}
```

Obrázek 6.19 – Vygenerovaný zdrojový kód nástrojem RSA

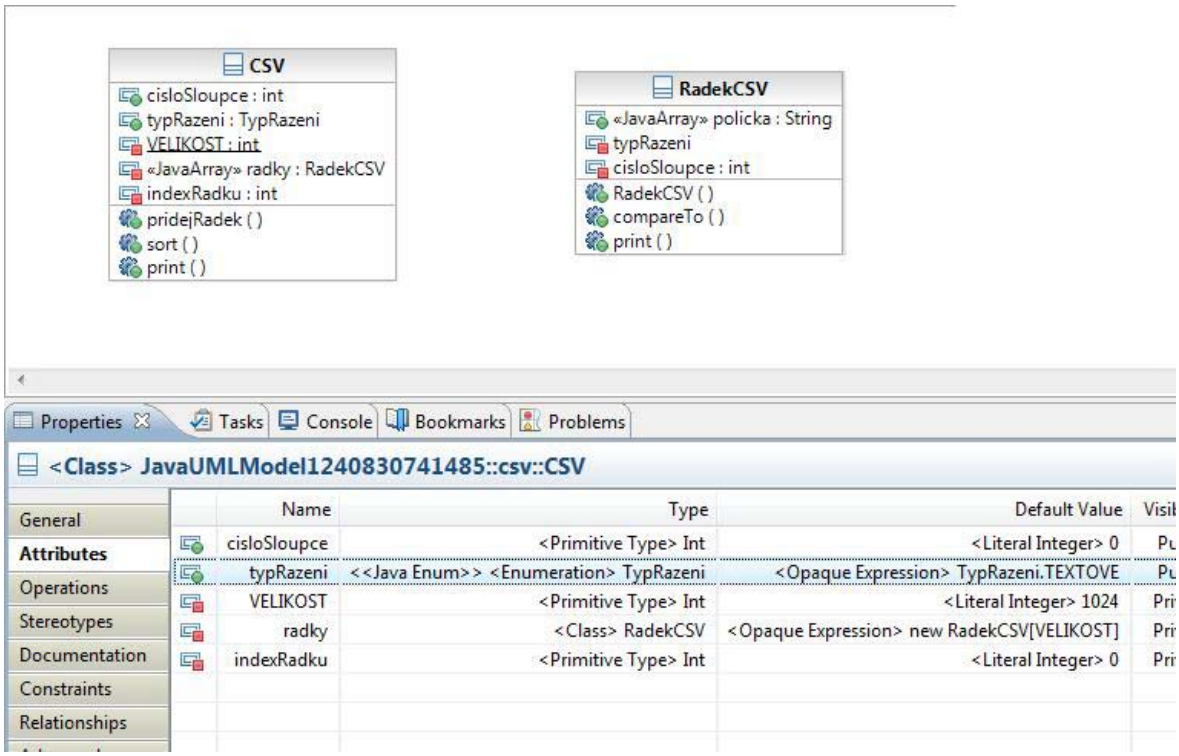
Vygenerovaný kód znázorňuje obrázek (Obrázek 6.19). Tento kód je správný, ovšem možná až zbytečně přeplněný nejrůznějšími komentáři. Další problémy byly s generováním samotné enumeration třídy, kdy RSA občas generoval třídu typu interface místo enum.

Všechny tyto problémy jsou nejspíš způsobeny rozšířeními (extensions), které jsou při transformaci do Javy použity, a které zřejmě nejsou úplně doladěny.

6.5.3 Schopnost reverzního inženýrství

Pro reverzní inženýrství, stejně jako pro generování kódu, se opět používá transformačních šablon. Jejich nastavení je velmi podobné jako v případě generování kódu.

Po spuštění reverzní šablony se nám importují dané package do složky s modely. Generování kteréhokoli UML diagramu se však nekoná. Tuto záležitost si už musí uživatel obstarat sám – například vytvořením diagramu tříd a následným „přetáhnutím“ jednotlivých třídy ze složky Models. Výsledek takového počinu znázorňuje obrázek (Obrázek 6.20).



Obrázek 6.20 – Vygenerované třídy včetně popisu atributů třídy CSV

Bohužel RSA nebyl schopen oddělit vnitřní třídu TypRazeni od třídy CSV. Tato třída je sice v deklaraci proměnných uvedena (Obrázek 6.20), součástí složky Models však není – uživatel v případě, že ji chce využít například v diagramu tříd, si ji musí vymodelovat sám.

6.5.4 Hodnocení

IBM Rational Software Architect	
Ačkoli za tímto nástrojem stojí gigant jako je IBM, tak i přesto tento nástroj na poli generování zdrojových kódů a reverzního inženýrství znatelně pokulhává za ostatními komerčními CASE nástroji. Bohužel neumí ani generování diagramů tříd.	
Generování kódu	●○○○○
Reverzní inženýrství	●◐○○○
Práce s nástrojem	●●●◐○

Tabulka 6.5 – Hodnoceni IBM Rational Software Architect

6.6 Smart Draw

6.6.1 Základní informace

Název: SmartDraw 2009

Verze: 2009.55 Mar 11 2009

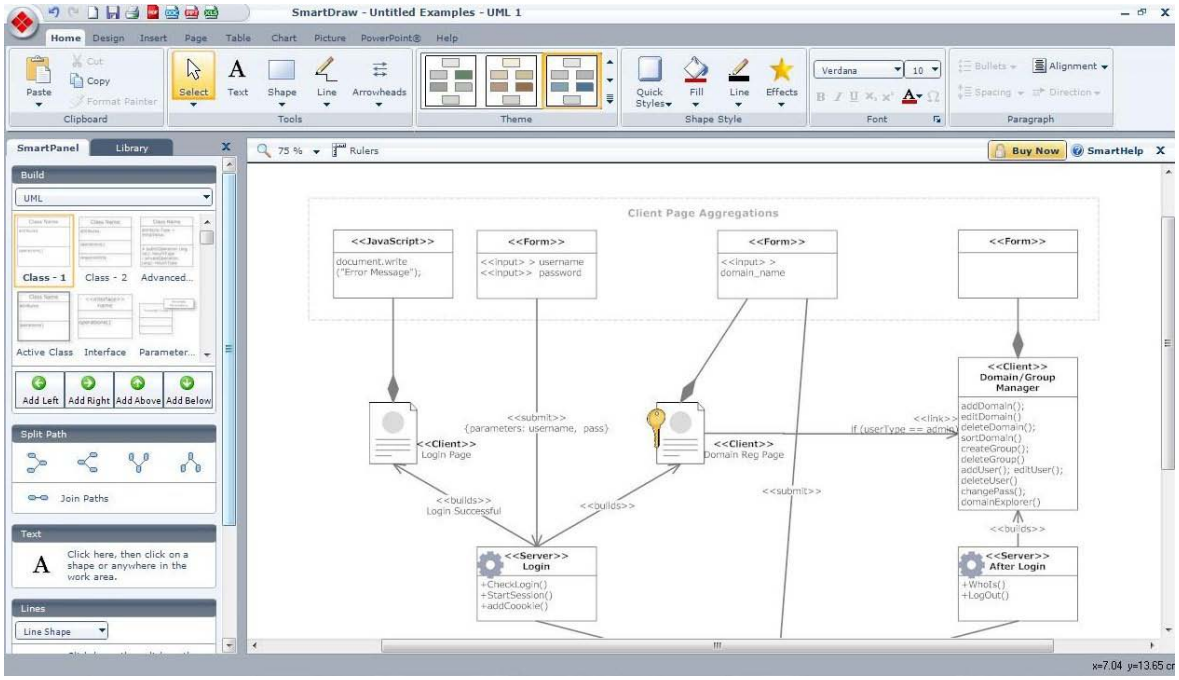
Autor: SmartDraw.com

Licence: Trial (7 dní)

Stažení: http://www.smartdraw.com/downloads/

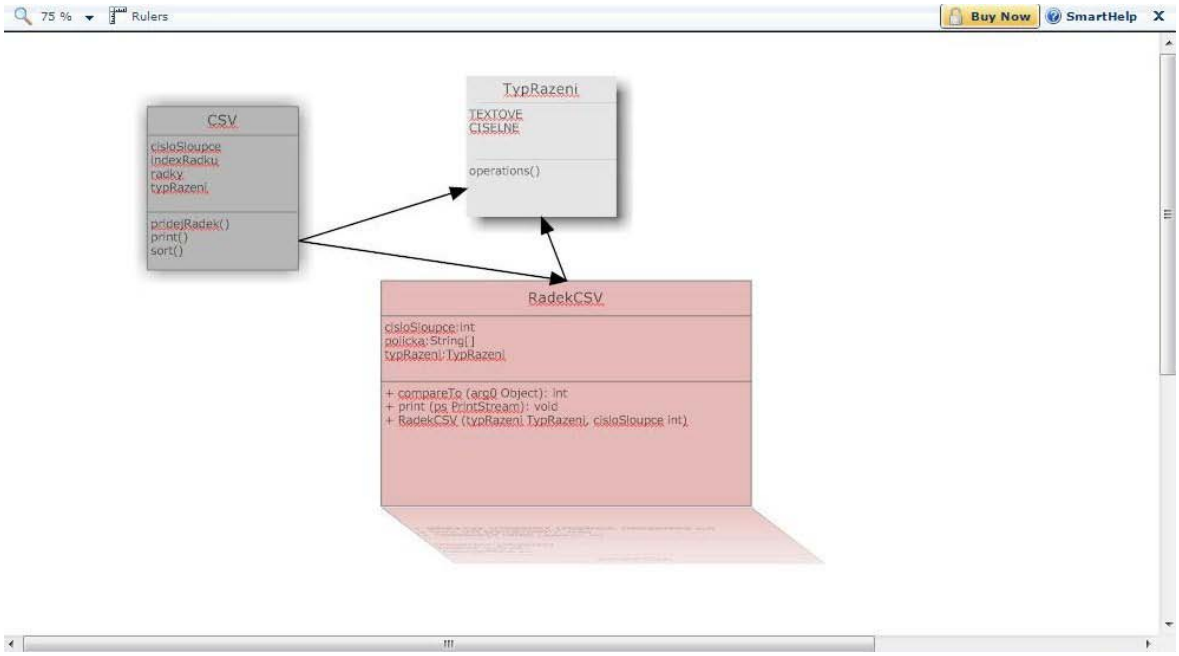
SmartDraw pochází od společnosti SmartDraw.com, která byla založena roku 1994. Tento program je jedním z mnoha nástrojů pro modelování všech možných diagramů a grafů. Bohužel tento nástroj nepodporuje jakékoli generování zdrojového kódu ani možnost reverzního inženýrství.[15]

Prostředí SmartDraw 2009 je hodně podobné prostředí Microsoft Office 2007 – tedy GUI uspořádáno do jednotlivých karet, které obsahují příslušné nástroje pro manipulaci s grafikou.



Obrázek 6.21 – Prostředí SmartDraw 2009

Tento nástroj slouží hlavně pro grafické znázornění různých postupů na business úrovni. Rozhodně se nedá využít například pro generování kódu či programové dokumentace. Jeho přednost je v grafické rozmanitosti. Tedy možnosti udělat z jinak „nudného“ diagramu efekty hýřící model. Některé z mnoha efektů znázorňuje obrázek (Obrázek 6.22).



Obrázek 6.22 – Ukázka efektů od SmartDraw 2009

6.6.2 Hodnocení

SmartDraw	
Tento CASE nástroj slouží pouze pro grafické znázornění nejrůznějších diagramů či jiných postupů. Jakákoli práce s daty (generování kódu, zpětné inženrství, ...) je mu však zapovězena	
Generování kódu	○○○○○
Reverzní inženýrství	○○○○○
Práce s nástrojem	●●●●○

Tabulka 6.6 – Hodnocení SmartDraw

7 VOLNĚ DOSTUPNÉ CASE NÁSTROJE

Testy vybraných volně dostupných nástrojů.

7.1 ArgoUML

7.1.1 Základní informace

Název: ArgoUML
Verze: v0.28
Autor: University of California, Project leader: Linus Tolke
Stažení: http://argouml.tigris.org/

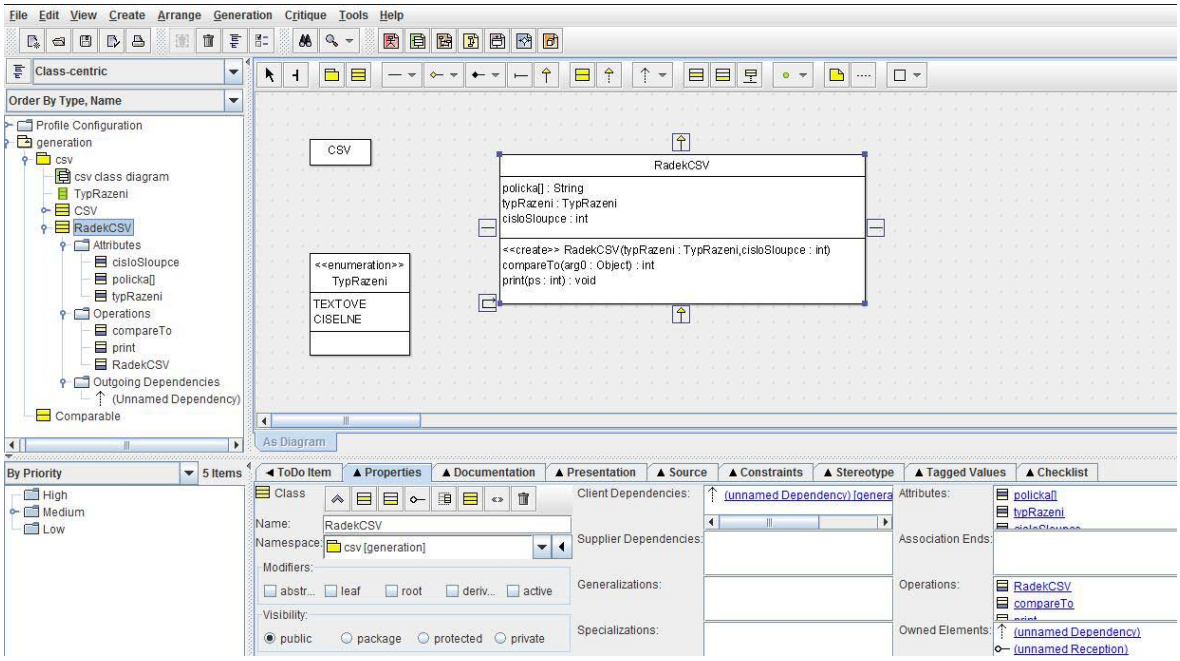
ArgoUML je „freewarovým" bratříčkem jiného komerčního Poseidonu for UML. ArgoUML je k dispozici všem uživatelům už přes 10 let. I přes tuto dobu však domovské stránky tohoto nástroje stále postrádají úplnou dokumentaci, a proto následující testování probíhalo převážně způsobem pokus-omyl.[8]

7.1.2 Schopnost generování zdrojového kódu

Při spuštění ArgoUML nás uvítá typické GUI Java aplikace. Aplikace působí stroze, za to ale velmi přehledně. Pro modelování stačí vytvořit nový projekt, zvolit typ diagramu a můžete začít (Obrázek 7.1). Doplňující informace ke všem prvkům se vkládají do karet pod diagramem.

Množství vložitelných informací je průměrné. Na jednu stranu lze k jednotlivým metodám dopsat zdrojový kód. Na stranu druhou si ArgoUML moc nerozumí se standardními knihovnami JDK - obsahuje pouze základní datové typy. Ostatní si uživatel musí vymodelovat sám či počkat na nějakou aktualizaci ze strany vývojářů ArgoUML.

Nebylo tedy možné importovat například typ PrintStream či inteface Comparable. Dalším problémem jsou importy již vytvořených typů. Ačkoli je třída TypRazeni vytvořena a v třídě RadekCSV využita, tak ve výsledném kódu její import chybí (Obrázek 7.2). Dále také chybí určení datového typu u pole „policka" ačkoli v původním diagramu tento typ uveden je. Ve zdrojovém kódu je uveden konstruktor včetně zdrojového kódu. Ten jsme cvičně uvedli už v diagramu tříd.



Obrázek 7.1 – GUI včetně diagramu tříd v ArgoUML

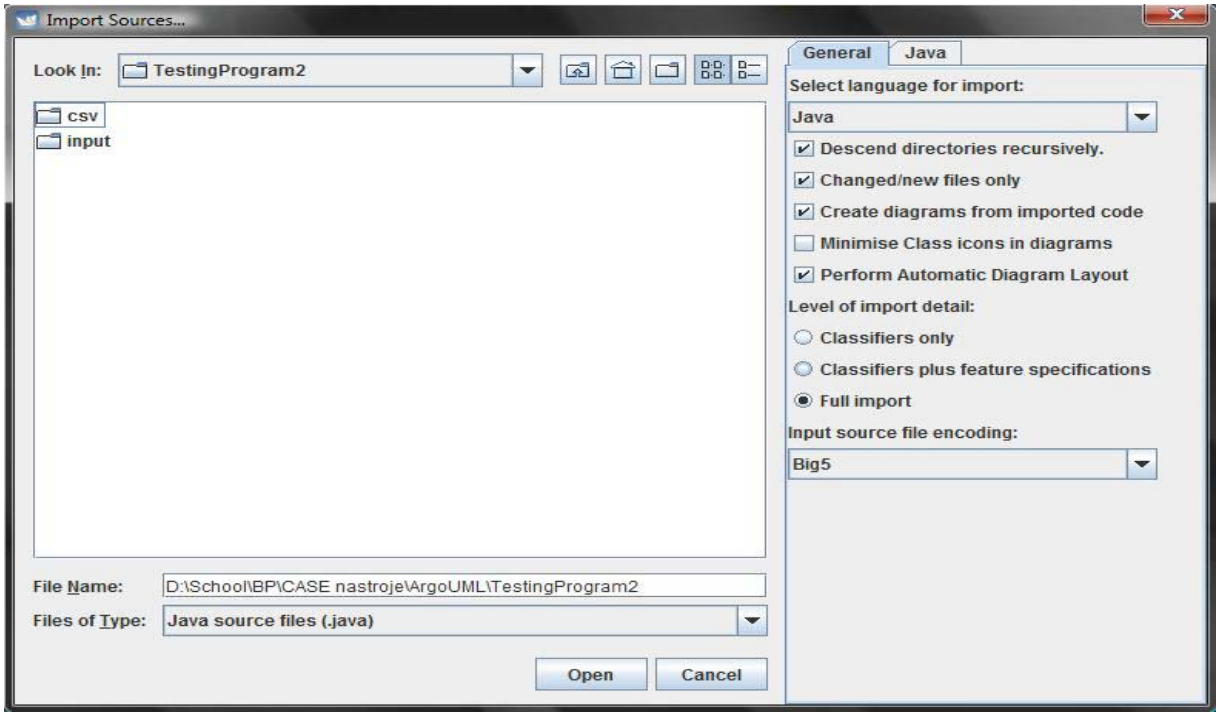
Vzhledem ke špatné spolupráci tohoto nástroje s JDK, není generování Java kódu jeho nejsilnější stránkou. Dalším záporem je nemožnost generování kódu z dynamických diagramů (stavové diagramy, sekvenční diagramy, …).

```
package csv;

/**
 *  Trida reprezentujici jeden radek v CSV objektu
 */
public class RadekCSV {

  public policka[];

  private TypRazeni typRazeni;

  private int cisloSloupce;

  /**
   *  Kontruktor - nastavi zpusob razeni a cislo sloupce, podle ktereho se bude tridit
   *
   *  @param typRazeni
   *  @param cisloSloupce
   */
  public RadekCSV(TypRazeni typRazeni, int cisloSloupce) {
this.typRazeni = typRazeni;
this.cisloSloupce = cisloSloupce;
  }

  /**
   *  Porovnani 2 radku
   */
  public int compareTo(Object arg0) {
  return 0;
  }

  /**
   *  Vypis radku na vystup
   *
   *  @param ps
   */
  public void print(int ps) {

  }
}
```

Obrázek 7.2 – Výsledný zdrojový kód generovaný ArgoUML

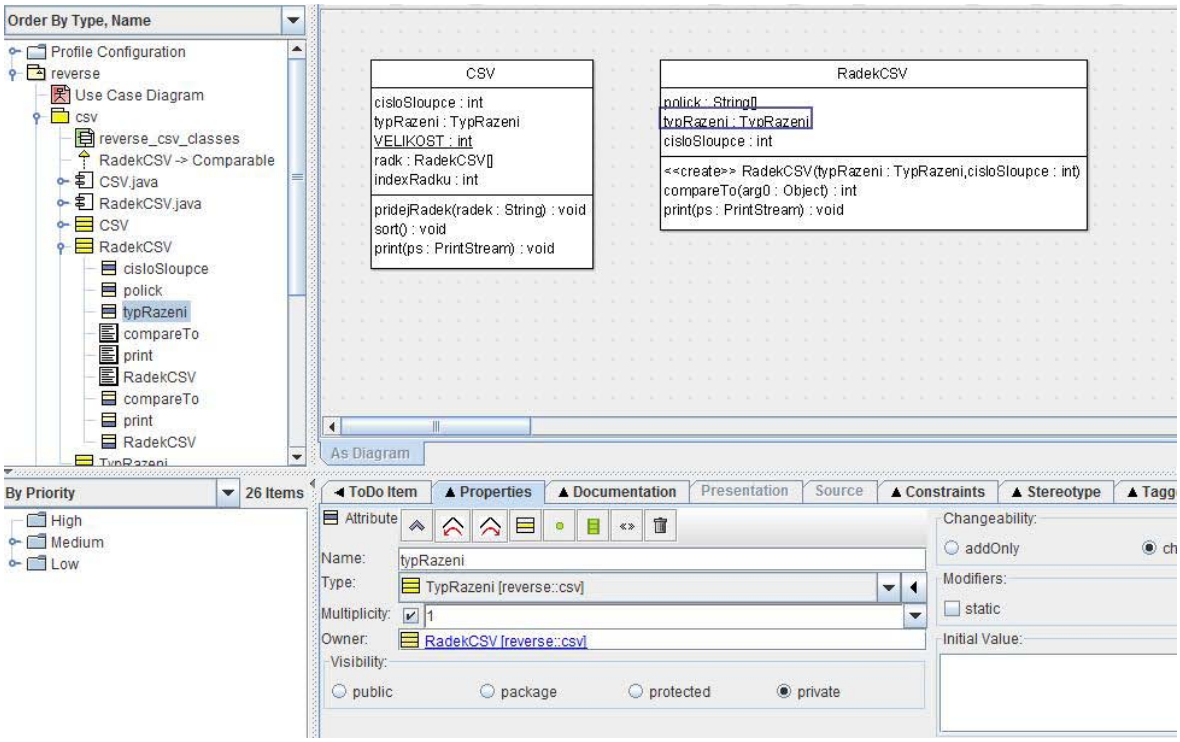
7.1.3 Schopnost reverzního inženýrství

Reverzní inženýrství je v tomto nástroji velmi jednoduchou záležitostí. V hlavním menu nástroje zvolíte Import Sources. Následně se Vám zobrazí okno s veškerým možným nastavením (Obrázek 7.3). Zde nastavíte cestu k adresáři s kódem, vyberete typ kódu (C++, Java, C#, …) a případně nastavíte další parametry importu.



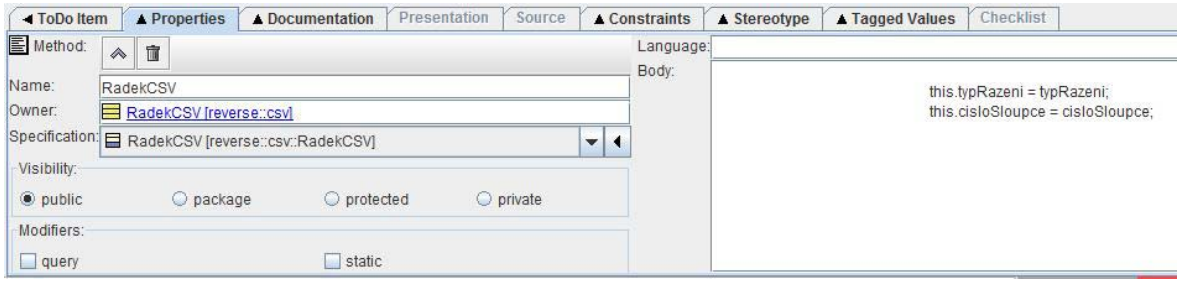
Obrázek 7.3 – Nastavení parametrů pro import zdrojových kódů

Po dokončení importu se vlevo vytvoří základní hierarchie kódu (packages, třídy, …). Pokud jsme při nastavení importu zaškrtli i požadavek na vytvoření diagramů, pak se ke každému package vytvoří i příslušný diagram tříd. Bohužel jiný typ diagramů ArgoUML neumí generovat. Diagramy tříd obsahují všechny atributy i metody. Pouze v grafickém znázornění chybí označení propojení mezi jednotlivými třídami. ArgoUML dále správně identifikoval třídu TypRazeni, bohužel však nepoznal, že jde o typ enumeration a zařadil ji mezi klasické třídy. Další chybou je špatné označení názvů polí – chybí poslední písmeno (místo „radky“ je atribut označen jako „radk“).



Obrázek 7.4 – Výsledek reverzního inženýrství v ArgoUML (diagram tříd)

ArgoUML importuje také zdrojový kód k jednotlivým metodám. Uživatel má tedy k dispozici zdrojový kód ke každé metodě (Obrázek 7.5). Zde už jsou názvy veškerých proměnných správné. Bohužel však z tohoto kódu nelze žádným způsobem vygenerovat například sekvenční diagram.



Obrázek 7.5 – Zdrojový kód k jedné z metod třídy RadekCSV

ArgoUML nalezl všechny prvky, které obsahoval zdrojový kód, měl problém ale s identifikací některých typů a hlavně s chybným označením některých atributů a celkového označení tříd.

7.1.4 Hodnocení

ArgoUML	
Špatná komunikace s JDK má za následek chyby jak v generování zdrojového kódu, tak chyby při reverzním inženýrství. Navíc je tento nástroj i přes svou dlouhou dobu vývoje stále minimálně zdokumentovaný.	
Generování kódu	●○○○○
Reverzní inženýrství	●●○○○
Práce s nástrojem	●●●○○

Tabulka 7.1 – Hodnocení ArgoUML

7.2 BOUML

7.2.1 Základní informace

Název: BOUML

Verze: 4.12.2

Autor: Bruno Pagés

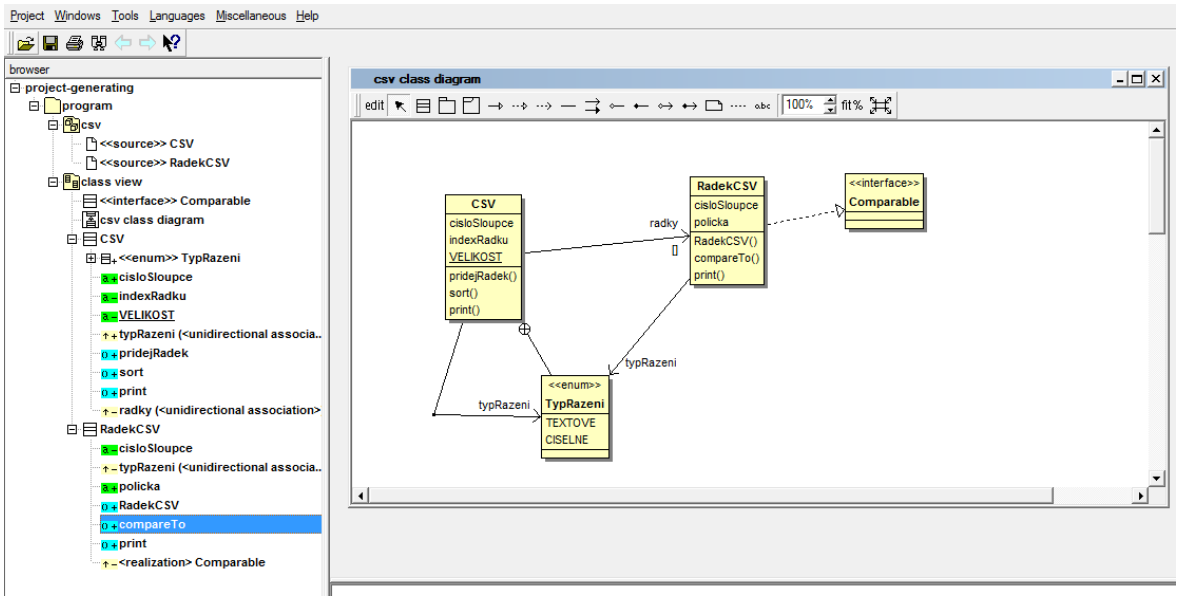
Stažení: http://downloads.sourceforge.net/bouml/Bouml_4.12.2_doc_4.9_setup.exe

Bouml je volně dostupným nástrojem, který vyvíjí pouze jeden člověk a to Francouz Bruno Pagés. Bouml běží pod mnoha platformami (Linus, Solaris, Windows, …). Umožňuje také simultánní práci s více jazyky najednou (C++, Java, Python, …). Spolupracuje s externími doplňky, tzv. plug-outs, které umožňují například generování zdrojového kódu.[9]

7.2.2 Schopnost generování zdrojového kódu

Modelování v Bouml je stejně jednoduché a přehledné jako celé GUI tohoto nástroje. V pracovním projektu uživatel vytvoří tzv. class view a v něm pak diagram tříd. V tomto diagramu následně uživatel modeluje. Během toho se v levé části automaticky zobrazují vymodelované třídy včetně všech prvků (atributy, metody, asociace, …). Výsledný diagram package CSV znázorňuje obrázek (Obrázek 7.6).

Bohužel BOUML nijak nekomunikuje s JDK, a proto je nutné všechny potřebné třídy a rozhraní dopsat či vymodelovat ručně. Taktéž tento nástroj neumí generovat zdrojový kód z diagramů, ale pouze z prvků, které jsou zobrazeny vlevo v projektovém stromu (Obrázek 7.6). Nakonec lze ke každé metodě v jednoduchém editoru dopsat i vlastní zdrojový kód.



Obrázek 7.6 – GUI nástroje Bouml včetně vymodelovaného diagramu tříd

Pokud chceme nadefinovat i hlavičky k různým třídám (licence, importy tříd – nejsou automatické, …), nadefinujeme je v tzv. deployment view. Ty pak napojíme na jednotlivé třídy.

Pro projekt stačí následně nadefinovat cílový adresář, kam se má vygenerovaný zdrojový kód umístit a následně spustit samotné generování. To proběhne velmi rychle a výsledek je díky kvalitním Java šablonám, které jsou součásti Bouml, velice dobrý (Obrázek 7.7).

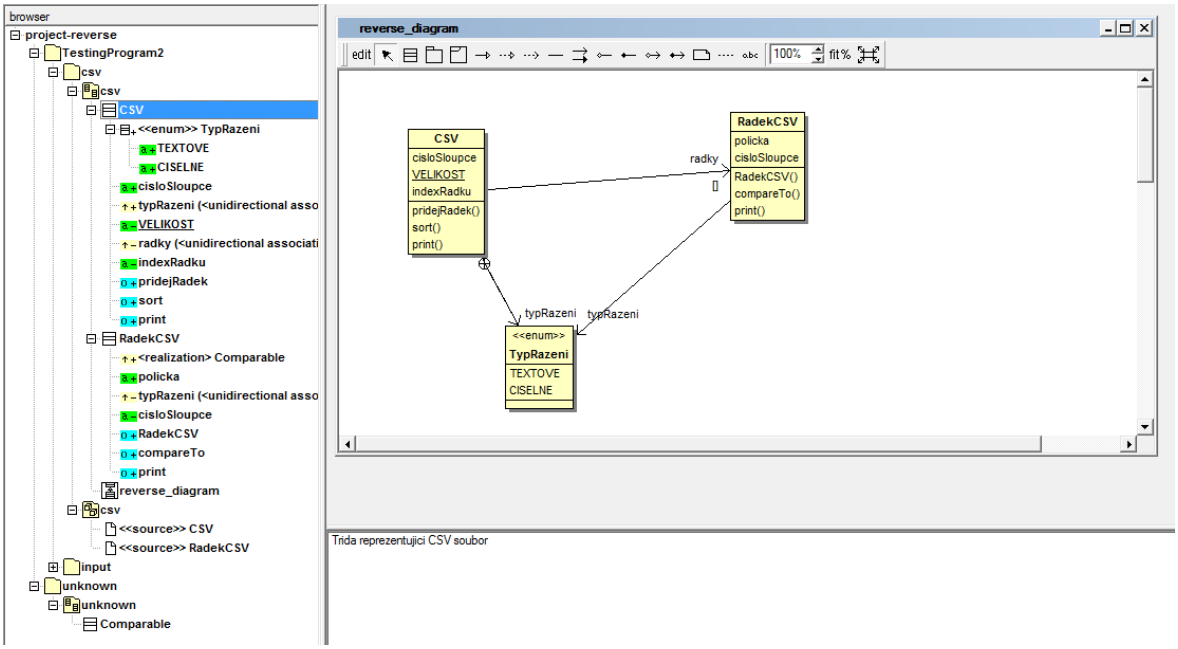
Pokud bychom zdrojový kód dopsali ke všem metodám a ne jen cvičně k jednomu konstruktoru (Obrázek 7.7), nebyl by výstupní kód vůbec odlišný od svého originálu, dle kterého byl vytvářen. Cenou za tuto kvalitu je však více práce při modelování (starost o importy knihoven, hlavičky tříd, implementace rozhraní, …).

```
/**
 * 1   This program is free software; you can redistribute it and/or modify
 * 2   it under the terms of the GNU General Public License as published by
 * 3   the Free Software Foundation; version 2 of the License.
 * 4
 * 5   This program is distributed in the hope that it will be useful,
 * 6   but WITHOUT ANY WARRANTY; without even the implied warranty of
 * 7   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * 8   GNU General Public License for more details.
 */
package csv;

import java.io.PrintStream;
import csv.CSV.TypRazeni;
/**
 * Trida reprezentujici jeden radek v CSV objektu
 */
public class RadekCSV implements Comparable {
  private int cisloSloupce;

  private TypRazeni typRazeni;

  public String[] policka;

  /**
   * Kontruktor - nastavi zpusob razeni a cislo sloupce, podle ktereho se bude
   * tridit
   *
   * @param typRazeni
   * @param cisloSloupce
   */
  public RadekCSV(TypRazeni typRazeni, int cisloSloupce) {

        this.typRazeni = typRazeni;
        this.cisloSloupce = cisloSloupce;
  }

  /**
   * Porovnani 2 radku
   */
  public int compareTo(Object arg0) {
  }

  /**
   * Vypis radku na vystup
   *
   * @param ps
   */
  public void print(PrintStream ps) {
  }

}
```

Obrázek 7.7 – Zdrojový kód vygenerovaný nástrojem Bouml

7.2.3 Schopnost reverzního inženýrství

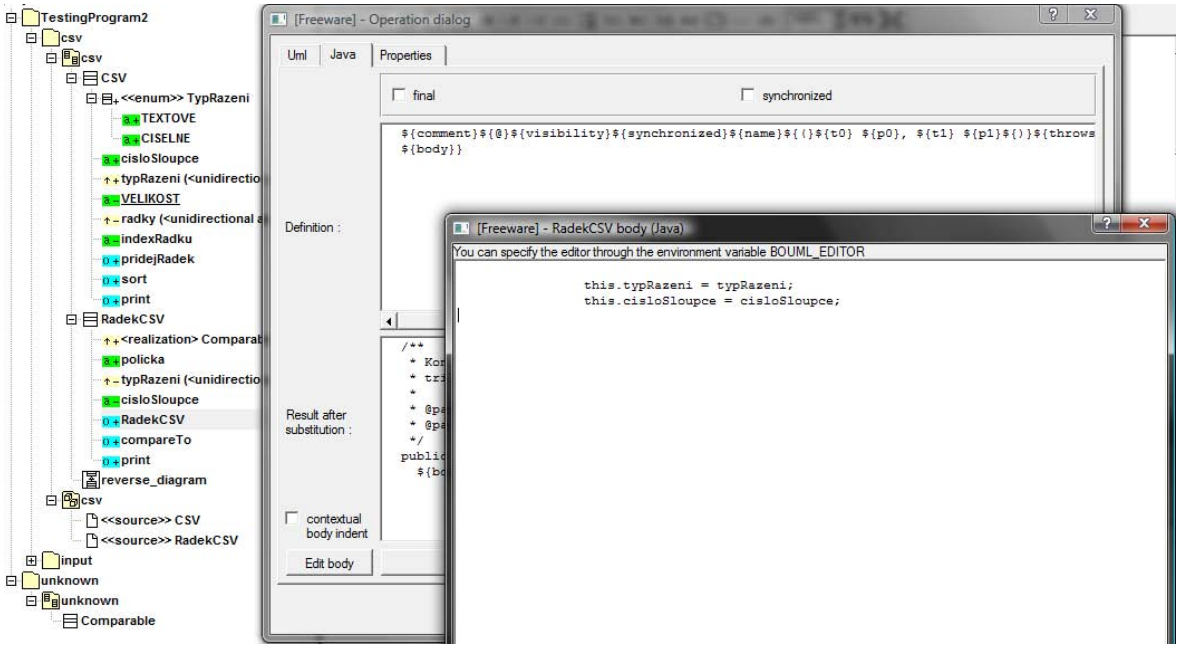
V této záležitosti dokázal Bouml skutečně překvapit. Import testovacího programu proběhl takřka okamžitě a jediná věc, se kterou si Bouml nedokázal poradit, byla implementace rozhraní Comparable ve třídě RadekCSV. Všechny ostatní prvky zdrojového kódu dokázal naprosto přesně rozeznat a identifikovat. V levé části nástroje tedy vytvořil přehlednou hierarchii všech importovaných částí programu Obrázek 7.8). Nechybí zde ani hlavička tříd, která ve formě komentáře má informovat o licenci programu.



Obrázek 7.8 – Výsledek reverzního inženýrství a následně vytvořený diagram tříd

Jediným mínusem v tomto pohledu je nemožnost automatického generování jakéhokoli diagramu ze zdrojového kódu. Bouml si poradí pouze s diagramem tříd, který si však musí uživatel sám vytvořit z importovaných dat. Práci mu však Bouml dokáže ušetřit v podobě již namapovaných propojení mezi třídami – jednotlivé třídy stačí do diagramu pouze přetáhnout a jejich propojení se následně zobrazí automaticky (Obrázek 7.8).

K tomu všemu importuje Bouml i kompletní zdrojový kód ke všem metodám, který je pak dostupný z vlastností vybrané metody (Obrázek 7.9).



Obrázek 7.9 – Importovaný zdrojový kód k jedné z metod

7.2.4 Hodnocení

BOUML	
Ačkoli tento nástroj působí na první pohled velmi lacině, jeho skutečná síla se projeví až při delší práci. Velmi kvalitní generování zdrojového kódu i funkce reverzního inženýrství. Bohužel však chybí komunikace s JDK a automatická tvorba diagramů - což velmi ubírá i na závěrečném hodnocení.	
Generování kódu	●●○○○
Reverzní inženýrství	●●◎○○
Práce s nástrojem	●●●●○

Tabulka 7.2 – Hodnocení BOUML

7.3 StarUML

7.3.1 Základní informace

Název: StarUML™

Verze: 5.0.2.1570

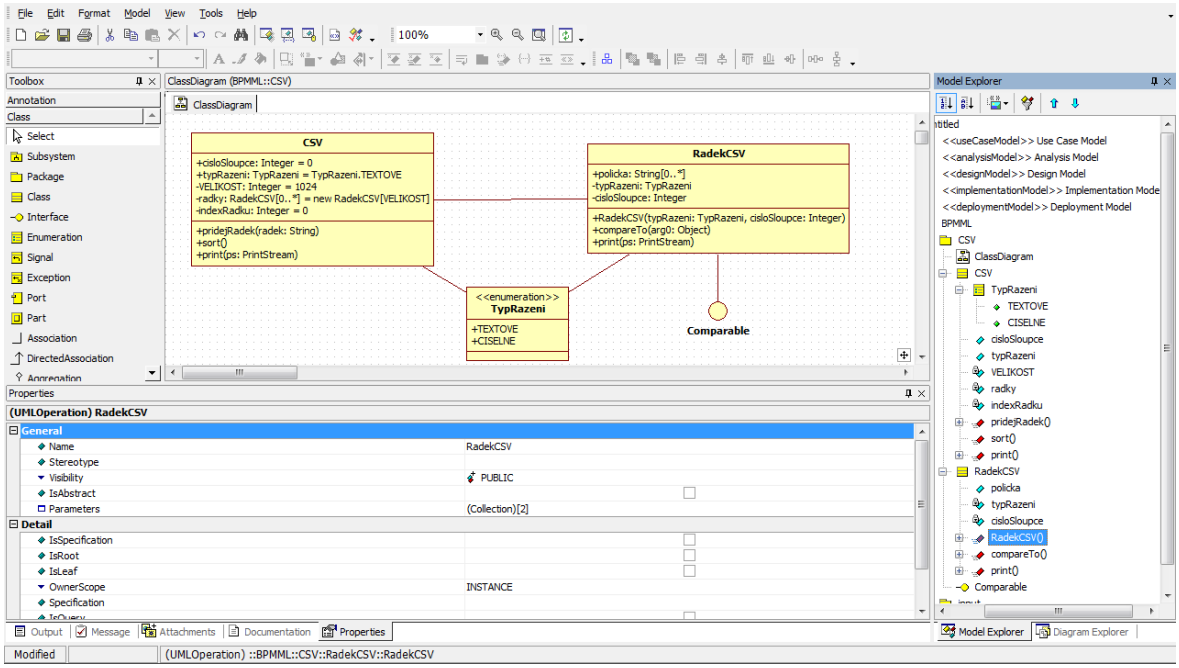
Autor: open source project

Stažení: http://staruml.sourceforge.net/en/download.php

StarUML je open source projekt, který je vyvíjen jako případná volně dostupná náhrada pro komerční UML nástroje. Je to nástroj, který podporuje standard UML 2.0 a jeho základní strukturu lze rozšiřovat o další zásuvné moduly (tzv. Plug-ins). Bohužel generovat zdrojový kód lze opět pouze z diagramů tříd.[17]

7.3.2 Schopnost generování zdrojového kódu

Modelování v tomto nástroji je jednoduché a člověk si na prostředí StarUML zvyká velmi rychle. Modelování probíhá standardním způsobem – umisťování jednotlivých prvků na pracovní plochu, propojovaní relacemi atd. Ke každému prvku můžeme přidat další informace přímo do karet umístěných pod pracovní plochou.



Obrázek 7.10 – GUI nástroje StarUML včetně diagramu tříd

Bohužel zde StarUML značně pokulhává, protože množství vložitelných informaci není mnoho. Navíc tento nástroj nespolupracuje s JDK, proto zná jen primitivní datové typy + typy vytvořené uživatelem. Dědičnost sice namodelovat lze, bohužel však implementace rozhraní možná není.

Generování kódu podporuje StarUML ve svém základu do jazyků C++, C# a Java. Vygenerovaný zdrojový kód znázorňuje obrázek (Obrázek 7.11). Zde je vidět základní kostra třídy (bohužel bez zmíněné implementace rozhraní). Chybí také import použitých knihoven JDK a nastavení typu jednotlivých metod. Kupodivu chybí i označení pole (které však v modelu nadefinované je). Použité vymodelované datové typy jsou uvedeny přímo v deklaraci proměnné a nejsou tedy importovány v podobě balíčků v hlavičce třídy. Generování poznámek StarUML umí na výbornou (je zde možnost generováni i Java dokumentace - JavaDoc).

```
//
//
//  Generated by StarUML(tm) Java Add-In
//
//  @ Project : Untitled
//  @ File Name : RadekCSV.java
//  @ Date : 8.5.2009
//  @ Author :
//
//



package CSV;


/**
 * Trida reprezentujici jeden radek v CSV objektu
**/
public class RadekCSV {
    public String policka;
    private CSV.CSV.TypRazeni typRazeni;
    private Integer cisloSloupce;
    /**
     * Kontruktor - nastavi zpusob razeni a cislo sloupce, podle ktereho se bude tridit
     *
     * @param    typRazeni
     * @param    cisloSloupce
    **/
    public void RadekCSV(CSV.CSV.TypRazeni typRazeni, Integer cisloSloupce) {

    }

    /**
     * Porovnani 2 radku
     *
     * @param    arg0
    **/
    public void compareTo(Object arg0) {

    }

    /**
     * Vypis radku na vystup
     *
     * @param    ps
    **/
    public void print(PrintStream ps) {

    }
}
```

Obrázek 7.11 – Zdrojový kód vygenerovaný pomocí StarUML

7.3.3 Schopnost reverzního inženýrství

StarUML podporuje reverzní inženýrství z jazyků C++, C# a Java. Bohužel při parsování testovacího zdrojového kódu vždy narazil na nějakou chybu (Obrázek 7.12). Při bližším přezkoumání zdrojového kódu se ukázalo, že označené chyby jsou jen falešným poplachem. Chyba tedy bude nejspíše v zastaralosti použitého Java parseru, který StarUML používá.

```
[12:55:55]  Starting Java reverse engineering. (3 files)
[12:55:55]  Syntax error ( D:\School\BP\CASE nastroje\StarUML\TestingProgram2\csv\CSV.java )
Description : Unrecoverable Parse Error
Location : 86Line -1Column
[12:55:55]  Syntax error ( D:\School\BP\CASE nastroje\StarUML\TestingProgram2\csv\RadekCSV.java )
Description : Unrecoverable Parse Error
Location : 61Line 16Column
[12:55:55]  Syntax error ( D:\School\BP\CASE nastroje\StarUML\TestingProgram2\input\SortCSV.java )
Description : Unrecoverable Parse Error
Location : 71Line 16Column
[12:55:57]  Java reverse engineering has been completed successfully.
```

Obrázek 7.12 – Část logu při reverzním inženýrství

7.3.4 Hodnocení

StarUML	
Tento nástroj podporuje jak reverzní inženýrství, tak generování zdrojového kódu. Bohužel však v obou případech obsahuje velké množství chyb, a proto jednotlivé výsledky nejsou příliš kvalitní.	
Generování kódu	●○○○○
Reverzní inženýrství	○○○○○
Práce s nástrojem	●●●○○

Tabulka 7.3 – Hodnocení StarUML

7.4 Eclipse UML2Tools

7.4.1 Základní informace

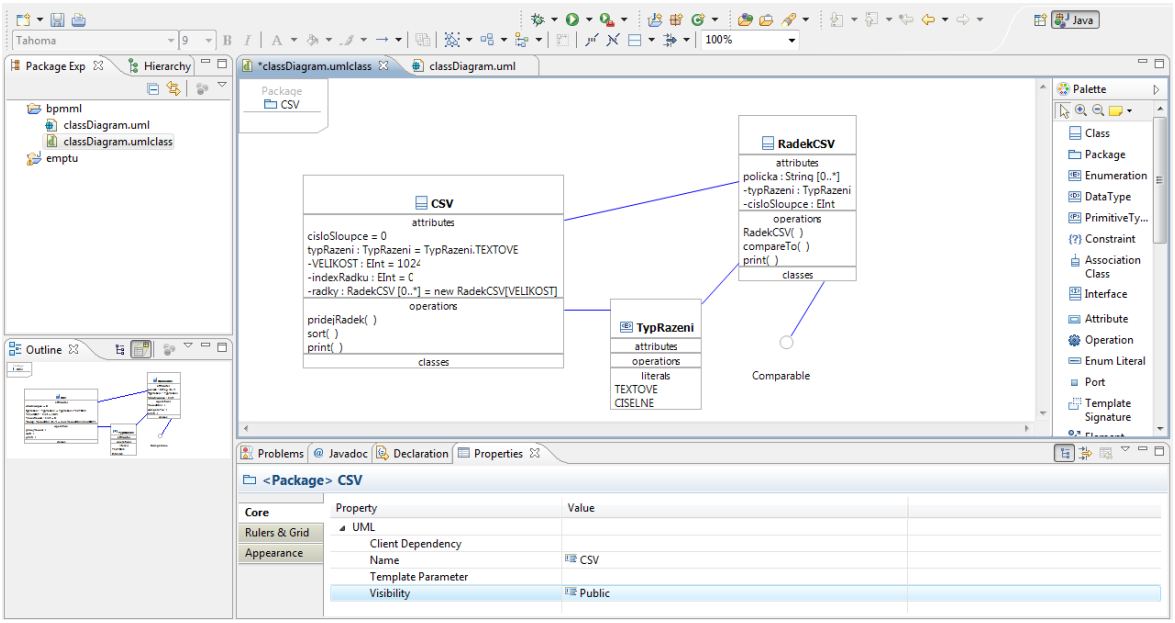
Název: Eclipse + Plug-in MDT-UML2Tools

Verze: Eclipse Platform 3.4.2 + UML2Tools 0.8.1.v200809231457

Autor: Borland Software Corporation and others

Stažení: http://www.eclipse.org/downloads/ - Eclipse Modeling Tools

UML2Tools není nic jiného než pouze plug-in (zásuvný modul) pro Eclipse. Tento plug-in umožňuje jediné, a to grafické modelování UML diagramů.[14]

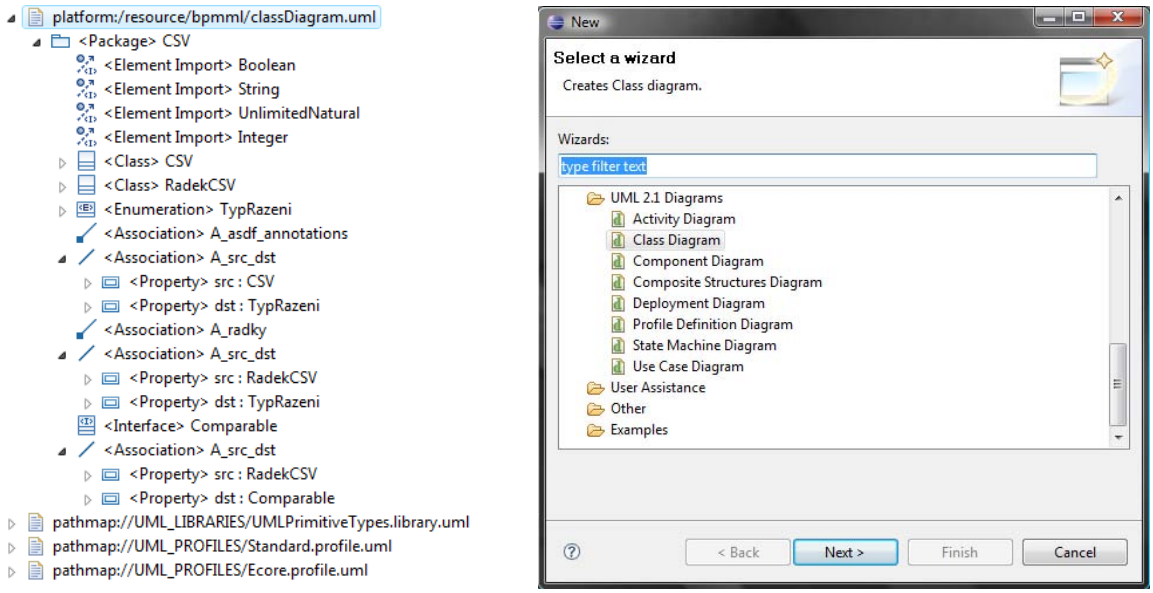


Obrázek 7.13 – GUI Eclipse UML2Tools včetně diagramu tříd (package CSV)

Práce v UML2Tools je sice jednoduchá, ale plug-in se potýká s řadou nedodělků (např. někdy je problém změnit hodnotu atributu, pokud neklikneme přesně do jednoho místa v jejím označení). GUI je velmi srozumitelné a přehledné. Informace o jednotlivých prvcích modelu se vkládají tradičně do karet ve spodní části pracovní plochy.

Co se týče zmíněných nedodělků, tak se tyto týkají například nastavování asociací a typů atributů – UML2Tools nabízí obrovské množství voleb (jen pro primitivní typ integer jsou k dispozici asi 4 možnosti) a uživatel se tak jen horko těžko orientuje. Dalším, velmi podstatným, problémem bylo modelování metod, u kterých se nepodařilo zjistit ani jakým

způsobem se přidávají jednotlivé vstupní parametry. Nepřehlednost zvyšovaly i některé již smazané prvky, které UML2Tools i nadále nabízel v některých volbách.



Obrázek 7.14 – Hierarchie výsledného modelu + typy podporovaných diagramů

Bohužel, jediným přínosem tohoto plug-inu je tedy pouze vytváření grafických UML modelů. Součástí není ani podpora generování programového zdrojového kódu, ani možnost reverzního inženýrství. Nástroj tedy poslouží pouze k vytvoření modelu, který může být následně přenesen do jiného nástroje (či využit jiným plug-inem), kde s ním mohou být provedeny další operace jako je například již zmíněné generování zdrojového kódu.

7.4.2 Hodnocení

Eclipse UML2Tools	
Plug-in pro Eclipse, který umožňuje grafické modelování UML diagramů. Bohužel nepodporuje generování programového kódu ani možnost reverzního inženýrství.	
Generování kódu	○○○○○
Reverzní inženýrství	○○○○○
Práce s nástrojem	●●●○○

Tabulka 7.4 – Hodnocení Eclipse UML2Tools

7.5 Eclipse PyUML

7.5.1 Základní informace

Název: Eclipse + Plug-in PyUML

Verze: Eclipse Platform 3.3.1.1 + PyUML 0.9

Autor: Freie University Berlin, Germany, Martin Dittmar

Stažení: http://sourceforge.net/projects/eclipse-pyuml

Tento nástroj je pouze další úpravou v podobě zásuvných modulů pro Eclipse. PyUML je postaven na zásuvných modulech UML2Tools a PyDev. Podporuje tedy i vizuální tvorbu UML diagramů.[10]

GUI pro tvorbu diagramů je tvořeno plug-inem UML2Tools který je popsán výše. PyUML podporuje generování kódu včetně reverzního inženýrství. Pomocí pár kliknutí můžete tedy buď synchronizovat zdrojový kód s UML diagramem tříd nebo naopak.

Bohužel tento nástroj podporuje pouze programovací jazyk Python. Proto jej nelze nijak srovnávat s ostatními nástroji v rámci našeho testovacího příkladu.

7.5.2 Hodnocení

Eclipse PyUML	
Svižný nástroj, který podporuje jak generování zdrojového kódu z diagramů tříd, tak reverzní inženýrství. Podporuje jen programovací jazyk Python.	
Generování kódu	○○○○○
Reverzní inženýrství	○○○○○
Práce s nástrojem	●●◐○○

Tabulka 7.5 – Hodnocení Eclipse PyUML

7.6 UML Designer

7.6.1 Základní informace

Název: UmlDesigner

Verze: 1.2.3144.37056

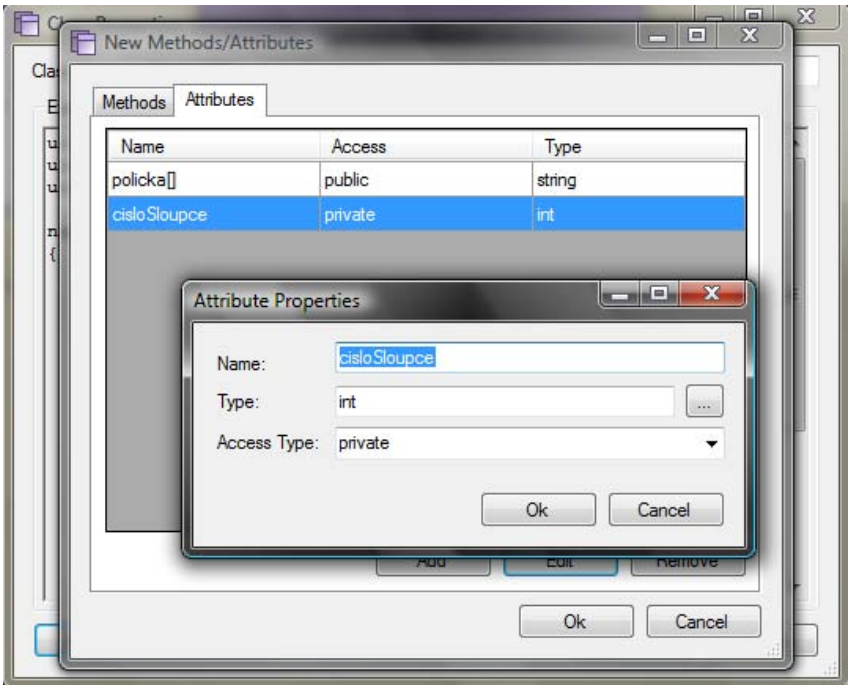
Autor: Green Bird Software

Stažení: http://www.greenbirdsoftware.com/Download.html

Tento nástroj od společnosti Green Bird Software slouží pouze k základnímu modelování diagramů tříd. Ostatní typy diagramů jsou mu zapovězeny. Výhodou tohoto nástroje tak zůstává jen rychlost a jednoduchost použití.[18]

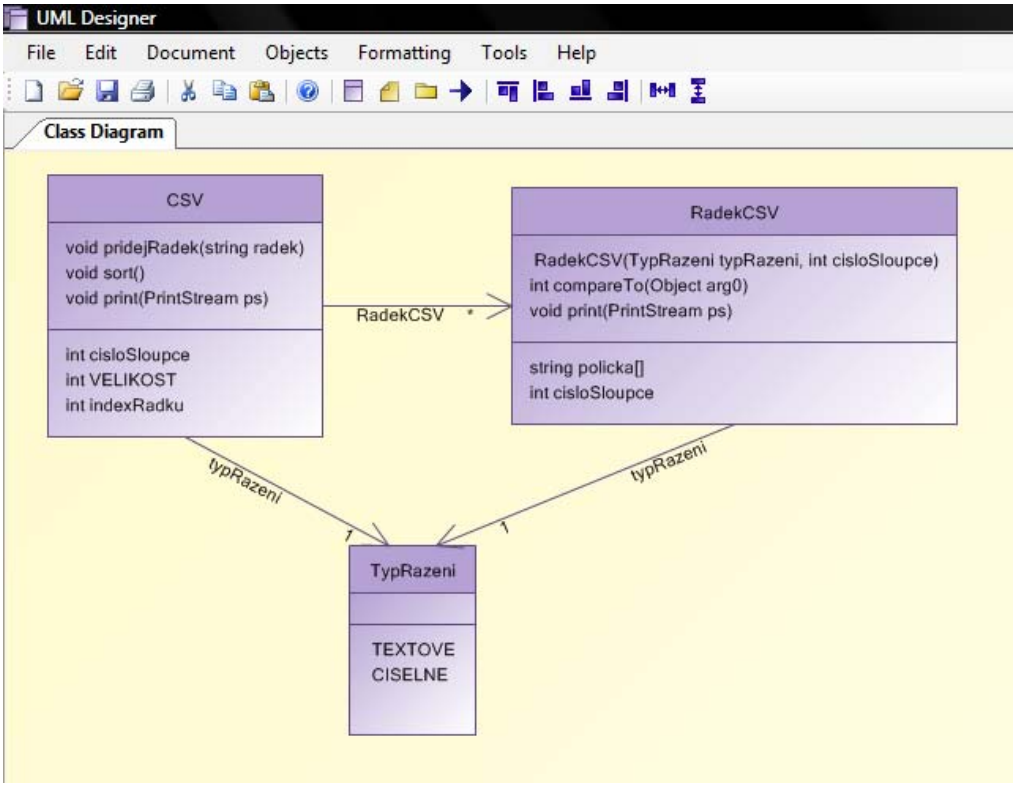
7.6.2 Schopnost generování zdrojového kódu

Tento nástroj podporuje pouze jednoduché modelování tříd a jejich prvků. Samozřejmostí je i modelování relací mezi třídami. Bohužel přidávání metod a atributů je základní, a proto jsou k dispozici pouze primitivní datové typy a typy už vymodelovaných tříd (Obrázek 7.15).



Obrázek 7.15 – Přidávání atributů k třídě

UML Designer podporuje při generování, a vůbec při celkovém modelování, pouze programovací jazyk C#. Není tedy možné namodelovat diagram přesně dle testovacího příkladu. Přibližný model package CSV znázorňuje obrázek (Obrázek 7.16).



Obrázek 7.16 – Vymodelovaný diagram tříd

Výsledný zdrojový kód (Obrázek 7.17) obsahuje pouze kostry daných tříd. Jelikož je v jiném programovacím jazyce, nemůžeme jej srovnávat s testovacím příkladem.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace UmlDesignerGeneratedCode
{
    class RadekCSV
    {
        private TypRazeni typRazeni;

        public  RadekCSV(TypRazeni typRazeni, int cisloSloupce)
        {
            throw new NotImplementedException();
        }

        private int compareTo(Object arg0)
        {
            throw new NotImplementedException();
        }

        private void print(PrintStream ps)
        {
            throw new NotImplementedException();
        }

        public string policka[];

        private int cisloSloupce;

    }
}
```

Obrázek 7.17 – Výsledný zdrojový kód vygenerovaným nástrojem UML Designer

7.6.3 Hodnocení

UML Designer	
Nástroj slouží pouze pro základní modelování tříd a jejich prvků. Nepodporuje reverzní inženýrství. Generování kódu zvládá pouze do jazyka C#. Nespolupracuje s žádným jiným programovacím jazykem.	
Generování kódu	○○○○○
Reverzní inženýrství	○○○○○
Práce s nástrojem	●●●○○

Tabulka 7.6 – Hodnocení UML Designer

8 ZÁVĚREČNÉ VÝSLEDKY TESTOVÁNÍ

8.1 Výsledky testování komerčních nástrojů

KOMERČNÍ CASE NÁSTROJE						
	Sparx Enterprise Architect	**Visual Paradigm for UML**	**MagicDraw UML**	**Poseidon for UML**	**IBM Rational Software Architect**	**SmartDraw**
Generování kódu						
diagramy tříd	●●●●○	●●●●○	●●●●○	●●●●○	●●○○○	-
stavové diagramy	-	-	-	-	-	-
diagramy aktivit	-	-	-	-	-	-
sekvenční diagramy	-	-	-	-	-	-
generování poznámek	●●●●○	●●●●○	●●●●○	●●●●○	●●●○○	-
import knihoven	●●○○○	●●●●○	●●●●○	●●●●○	●●●●○	-
Průměr	●●○○○	●●○○○	●●○○○	●●○○○	●○○○○	○○○○○
Reverzní inženýrství						
deklarace třídy	●●●●●	●●●●○	●●●●○	●●●●○	●●●○○	-
deklarace datových členů	●●●●●	●●●●●	●●●●●	●●○○○	●●●●○	-
implementace členských funkcí	-	-	-	●●●●●	-	-
gen. stavových diagramů	-	-	-	-	-	-
gen. diagramů aktivit	-	-	-	-	-	-
gen. sekvenčních diagramů	-	-	●●●●●	-	-	-
zpracování poznámek z kódu	●●●●○	●●●●○	●●●●○	●●●●○	●●●●○	-
Průměr	●●○○○	●●○○○	●●●○○	●●○○○	●○○○○	○○○○○
Práce s nástrojem						
komfort modelování	●●●●○	●●●○○	●●●●○	●●●●○	●●●○○	●●●●○
dokumentace	●●●●○	●●●●○	●●●○○	●●●●●	●●●●○	●●●●○
srozumitelnost GUI	●●●●○	●●○○○	●●●●○	●●●○○	●●●●○	●●●●○
svižnost nástroje	●●●●○	●●○○○	●●●●○	●●●●○	●●●●○	●●●●○
Průměr	●●●●○	●●○○○	●●●○○	●●●●○	●●●○○	●●●●○
CELKOVÉ HODNOCENÍ	●●○○○	●●○○○	●●○○○	●●○○○	●○○○○	○○○○○
	Sparx Enterprise Architect	**Visual Paradigm for UML**	**MagicDraw UML**	**Poseidon for UML**	**IBM Rational Software Architect**	**SmartDraw**

Tabulka 8.1 – Podrobné hodnocení komerčních CASE nástrojů

8.2 Výsledky testování volně dostupných nástrojů

VOLNĚ DOSTUPNÉ CASE NÁSTROJE						
	ArgoUML	**BOUML**	**StarUML**	**Eclipse UML2Tools**	**Eclipse PyUML**	**UML Designer**
Generování kódu						
diagramy tříd	●●●●○	●●●●●	●●●○○	-	-	●●○○○
stavové diagramy	-	-	-	-	-	-
diagramy aktivit	-	-	-	-	-	-
sekvenční diagramy	-	-	-	-	-	-
generování poznámek	●●●●○	●●●●●	●●●●●	-	-	-
import knihoven	○○○○○	●●○○○	●○○○○	-	-	-
Průměr	●○○○○	●●○○○	●○○○○	○○○○○	○○○○○	○○○○○
Reverzní inženýrství						
deklarace třídy	●●●○○	●●●○○	-	-	-	-
deklarace datových členů	●●●●○	●●●●●	-	-	-	-
implementace členských funkcí	●●●●●	●●●●●	-	-	-	-
gen. stavových diagramů	-	-	-	-	-	-
gen. diagramů aktivit	-	-	-	-	-	-
gen. sekvenčních diagramů	-	-	-	-	-	-
zpracování poznámek z kódu	●●●●○	●●●●●	-	-	-	-
Průměr	●●○○○	●●○○○	○○○○○	○○○○○	○○○○○	○○○○○
Práce s nástrojem						
komfort modelování	●●●●○	●●●●●	●●●●○	●●○○○	●●○○○	●●●●○
dokumentace	●●○○○	●●●○○	●●●○○	●●●○○	●○○○○	●○○○○
srozumitelnost GUI	●●●●○	●●●●○	●●●●○	●●●●○	●●●●○	●●●●○
svižnost nástroje	●●●●○	●●●●●	●●●●○	●●●●○	●●●●○	●●●●●
Průměr	●●●○○	●●●●○	●●●○○	●●●○○	●●○○○	●●●○○
CELKOVÉ HODNOCENÍ	●○○○○	●●○○○	○○○○○	○○○○○	○○○○○	○○○○○
	ArgoUML	**BOUML**	**StarUML**	**Eclipse UML2Tools**	**Eclipse PyUML**	**UML Designer**

Tabulka 8.2 – Podrobné hodnocení volně dostupných CASE nástrojů

8.3 Vysvětlivky k závěrečným tabulkám

8.3.1 Bodové hodnocení

Bodové hodnocení znázorňuje tabulka (Tabulka 8.3). Platí, že čím víc bodů, tím lepší je daná vlastnost nástroje (maximum je 5 bodů, minimum je 0 bodů). Neohodnocení znamená nemožnost danou vlastnost otestovat (např. z důvodu nepodporování).

Označení	Počet bodů
●	1
◘	0,5
○	0
-	nehodnoceno (=0)

Tabulka 8.3 – Bodování

8.3.2 Průměry a celkové hodnocení

Jednotlivé průměry jsou počítány jako vážené průměry dle vztahu (1). Váhy k jednotlivým vlastnostem jsou uvedeny níže.

$$\overline{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} \quad \text{, kde} \tag{1}$$

$$X = \{x_1, ..., x_n\} \rightarrow \text{soubor } n \text{ hodnot}$$

$$W = \{w_1, ..., w_n\} \rightarrow \text{k nim odpovídající váhy}$$

Celkové hodnocení je pak opět spočítáno pomocí váženého průměru. Váhy jsou v tomto případu určeny takto:

- Průměr z generování kódu (w=1)
- Průměr z reverzního inženýrství (w=1)
- Průměr z práce s nástrojem (w=0,3)

8.3.3 Vysvětlení hodnocených vlastností

Jméno vlastnosti + (váha dané vlastnosti pro výpočet váženého průměru) + její vysvětlení.

Generování zdrojového kódu:

- **diagramy tříd** (w=1), **stavové diagramy** (w=0,75), **diagramy aktivit** (w=0,75), **sekvenční diagramy** (w=0,75) – schopnost CASE nástroje generovat zdrojový kód z vymodelovaných diagramů
- **generování poznámek** (w=0,5) – importování poznámek z modelu do generovaného zdrojového kódu
- **import knihoven** (w=0,75) – importování všech potřebných knihoven do generovaného zdrojového kódu

Reverzní inženýrství:

- **deklarace třídy** (w=1) – generování diagramů tříd a jejich celková kvalita
- **deklarace datových členů** (w=1) – správnost deklarace všech datových členů
- **implementace členských funkcí** (w=0,75) – schopnost importu členských funkcí včetně jejich obsahu
- **generování stavových diagramů** (w=0,75), **diagramů aktivit** (w=0,75), **sekvenčních diagramů** (w=0,75) – schopnost CASE nástroje generovat jednotlivé diagramy ze zdrojových kódů
- **zpracování poznámek z kódu** (w=0,5) – schopnost a kvalita zpracování poznámek ze zdrojového kódu

Práce s nástrojem:

- **komfort modelování** (w=1) – celkový komfort při práci s CASE nástrojem (klávesové zkratky, nabídka nástrojů, …)
- **dokumentace** (w=1) – dosažitelnost a kvalita dokumentace k danému nástroji
- **srozumitelnost GUI** (w=1) – celková srozumitelnost a jednoduchost uživatelského prostředí
- **svižnost nástroje** (w=1) – rychlost spouštění, doba reakce na uživatelskou práci, rychlost importu (reverzní inženýrství) a exportu (generování zdrojového kódu)

ZÁVĚR

Tato bakalářská práce měla za cíl srovnat současné CASE nástroje z hlediska generování zdrojového kódu a reverzního softwarového inženýrství.

Generování zdrojového kódu bylo prováděno do objektově-orientovaného programovacího jazyka JavaSE. Testovala se schopnost CASE nástroje generovat zdrojový kód z diagramů tříd, stavových diagramů, diagramů aktivit a sekvenčních diagramů.

V tomto směru drtivá většina CASE nástrojů (jak komerčních, tak volně dostupných) dokázala spolupracovat pouze se statickými diagramy (diagramy tříd). Generování zdrojových kódů z dynamických diagramů nedokázal žádný z nástrojů. Některé však tento nedostatek nahrazovaly možností připojení zdrojového kódu přímo k metodám v diagramech tříd (např. Poseidon for UML nebo BOUML).

Reverzní softwarové inženýrství zvládala většina komerčních nástrojů a dva volně dostupné nástroje (ArgoUML a BOUML). Opět jsou podporovány pouze statické diagramy. Výjimku v tomto tvoří pouze nástroj MagicDraw UML, který dokáže vytvářet dynamické sekvenční diagramy z importovaných metod.

Celkově vybrané CASE nástroje v těchto testech moc dobře neobstály. Zajímavé však může být srovnání volně dostupného nástroje BOUML, který se svým velmi dobrým hodnocením může rovnat řadě komerčních (a ne zrovna levných) nástrojů.

Na přiloženém CD se nachází text bakalářské práce (v PDF formátu) a zdrojové kódy testovacího programu.

ZÁVĚR V ANGLIČTINĚ

The aim of this bachelor project was to compare actual development tools for software engineering from code generating and reverse software engineering point of view where object-oriented programming language JavaSE was used.

The CASE tools were tested for the capability of source code generation from class diagrams, state transition diagrams, activity diagrams and sequence diagrams.

From this point of view the most of nowaday versions of CASE tools (commercial and freeware) can cooperate only with static diagrams (class diagram) and code generation from dynamic diagrams is not supported. However, some of them allowed connecting source code straight to methods in class diagrams (e.g. Poseidon for UML or BOUML).

Reverse software engineering was supported in the most of the tested commercial tools and two freeware tools (ArgoUML and BOUML). Unfortunately they manage to create only static diagrams as well. The only exception in this case is MagicDraw UML which supports generating of dynamic sequence diagrams from imported source code.

Generally, the rating of chosen CASE tools in these tests was not very good. But it is interesting to compare a very well rated freeware tool BOUML with rating of other commercial (and not so cheap) tools which BOUML could be equal to.

The enclosed CD contains the text of bachelor project (in PDF format) and source codes of the testing program.

SEZNAM POUŽITÉ LITERATURY

Monografie:

[1] ARLOW, J., NEUSTADT, I. UML a unifikovaný proces vývoje aplikací. Brno: Computer Press, 2003. 387 s. ISBN 80-7226-947-X.

[2] KANISOVÁ, H., MÜLLER, M. UML Srozumitelně. 2. akt. vyd. Brno: Computer Press, 2006. 176 s. ISBN 80-251-1083-4.

Internetové zdroje (monografie):

[3] NOUZA, Oldřich. *Principy generování zdrojového kódu z UML*. [s.l.], 2004. 8 s. ČVUT v Praze, FJFI, katedra matematiky. Článek. Dostupný z WWW: <http://objekty.pef.czu.cz/2004/sbornik/05_Nouza.pdf>.

[4] NOVÁČEK, Zdeněk. *Reverzní inženýrství v prostředí CASE Studio 2*. [s.l.], 2006. 28 s. Masarykova univerzita, fakulta informatiky. Bakalářská práce. Dostupný z WWW: <http://is.muni.cz/th/72725/fi_b/bp.pdf>.

[5] PROCHÁZKA, Jaroslav. *Automatic Code Generation*. [s.l.], 2004. 4 s. Ostravská univerzita v Ostravě, Přírodovědecká fakulta, katedra informatiky a počítačů. Článek. Dostupný z WWW: <http://dar.site.cas.cz/download.php?bd=313>.

Internetové zdroje (seriály):

[6] Nástroje pro tvorbu UML diagramů. *Root.cz* [online]. 4.7.2005 [cit. 2009-03-08]. Dostupný z WWW: <http://www.root.cz/clanky/nastroje-pro-tvorbu-uml-diagramu/>.

[7] Návrh aplikací v jazyce UML. *Interval.cz* [online]. 2003-2005 [cit. 2009-03-09]. Dostupný z WWW: <http://interval.cz/serialy/navrh-aplikaci-v-jazyce-uml/>.

Internetové zdroje (CASE nástroje):

[8] *ArgoUML : Tygris.org* [online]. ©2001-2008 [cit. 2009-04-23]. Dostupný z WWW:<http://argouml.tigris.org/>.

[9] *BOUML : a free UML tool box* [online]. ©2004-2009 [cit. 2009-04-28]. Dostupný z WWW: <http://bouml.free.fr/>.

[10] *Eclipse-PyUML : SourceForge.net* [online]. 2008-01-30 [cit. 2009-05-08]. Dostupný z WWW: <http://sourceforge.net/projects/eclipse-pyuml>.

[11] *Gentleware : model to business* [online]. ©2000-2009 [cit. 2009-04-05]. Dostupný z WWW: <http://www.gentleware.com/>.

[12] *IBM : Rational Software Architect* [online]. [2002] [cit. 2009-04-15]. Dostupný z WWW: <http://www-01.ibm.com/software/awdtools/swarchitect/websphere/>.

[13] *MagicDraw* [online]. ©2000-2009 [cit. 2009-04-02]. Dostupný z WWW: <http://www.magicdraw.com/>.

[14] *MDT-UML2Tools FAQ : Eclipsepedia* [online]. ©2009 , 5 March 2009 [cit. 2009-05-05]. Dostupný z WWW: <http://wiki.eclipse.org/MDT-UML2Tools_FAQ>.

[15] *SmartDraw : Communicate Visually* [online]. ©2009 [cit. 2009-04-18]. Dostupný z WWW: <http://www.smartdraw.com/>.

[16] *Sparx Systems* [online]. ©2000-2009 [cit. 2009-03-25]. Dostupný z WWW: <http://www.sparxsystems.com/>.

[17] *StarUML : The Open Source UML/MDA Platform* [online]. [1996] [cit. 2009-05-08]. Dostupný z WWW: <http://staruml.sourceforge.net/en/>.

[18] *UML Designer : Green Bird Software\'s .NET UML Modeling Tool* [online]. ©2008 [cit. 2009-05-03]. Dostupný z WWW: <http://www.greenbirdsoftware.com/>.

[19] *Visual Paradigm* [online]. ©1999-2008 [cit. 2009-03-30]. Dostupný z WWW: <http://www.visual-paradigm.com/>.

Internetové zdroje (ostatní):

[20] *Introduction to the Diagrams of UML 2.0* [online]. ©2003-2007 [cit. 2009-03-12]. Dostupný z WWW: <http://www.agilemodeling.com/essays/umlDiagrams.htm>.

[21] *Object Management Group : UML* [online]. ©1997-2009 , Thursday, January 08, 2009 [cit. 2009-03-10]. Dostupný z WWW: <http://www.uml.org/>.

[22] *Softwarové inženýrství : UML* [online]. 23.6.2007 [cit. 2009-03-08]. Dostupný z WWW: <http://srakyi.modry.cz/blog/category/uml>.

[23] *Úvod do jazyka UML : Komix s.r.o.* [online]. 2009 [cit. 2009-03-10]. Dostupný z WWW: <http://www.komix.cz/Tisk/Clanky/Historie/Uvod_UML.aspx>.

[24] *Wikipedia : Computer-aided software engineering* [online]. [2009] [cit. 2009-03-21]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Computer-aided_software_engineering>.

[25] *Wikipedia : Reverse engineering* [online]. [2009] [cit. 2009-03-20]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Reverse_engineering>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

EA	Sparx Enterprise Architect
CASE	Computer-aided Software Engineering
CBD	Computer-based Development
CSV	Comma Separated Value
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
JAR	Java Archive
JDK	Java Development Kit
JRE	Java Runtime Environment
OMG	Object Management Group
OMT	Object Modeling Technique
RSA	Rational Software Architect
UML	Unified Modeling Language
VP	Visual Paradigm for UML
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

ZDROJOVÝ KÓD ZK I: TESTOVACÍ PROGRAM, ČÁST PRVNÍ

- třída RadekCSV (package csv) – třída zastupující jeden řádek virtuálního CSV souboru

```
/**
 1   This program is free software; you can redistribute it and/or modify
 2   it under the terms of the GNU General Public License as published by
 3   the Free Software Foundation; version 2 of the License.
 4
 5   This program is distributed in the hope that it will be useful,
 6   but WITHOUT ANY WARRANTY; without even the implied warranty of
 7   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 8   GNU General Public License for more details.
 */

package csv;

import java.io.PrintStream;

import csv.CSV.TypRazeni;

/**
 * Trida reprezentujici jeden radek v CSV objektu
 *
 */
public class RadekCSV implements Comparable {
	public String policka[];

	private TypRazeni typRazeni;

	private int cisloSloupce;

	/**
	 * Kontruktor - nastavi zpusob razeni a cislo sloupce, podle ktereho se bude
	 * tridit
	 *
	 * @param typRazeni
	 * @param cisloSloupce
	 */
	public RadekCSV(TypRazeni typRazeni, int cisloSloupce) {
		this.typRazeni = typRazeni;
		this.cisloSloupce = cisloSloupce;
	}

	/**
	 * Porovnani 2 radku
	 */
	public int compareTo(Object arg0) {
		if (typRazeni == TypRazeni.TEXTOVE) {
			return policka[cisloSloupce]
					.compareTo(((RadekCSV) arg0).policka[cisloSloupce]);
		} else if (typRazeni == TypRazeni.CISELNE) {
			return Integer.decode(policka[cisloSloupce])
					- Integer.decode(((RadekCSV)
arg0).policka[cisloSloupce]);
		}
		return 0;
	}

	/**
	 * Vypis radku na vystup
	 *
	 * @param ps
	 */
	public void print(PrintStream ps) {
		for (String s : policka)
			ps.print(s + "  |  ");
	}
}
```

ZDROJOVÝ KÓD ZK II: TESTOVACÍ PROGRAM, ČÁST DRUHÁ

- třída CSV (package csv) – třída zastupující virtuální CSV soubor

```
/**
 1   This program is free software; you can redistribute it and/or modify
 2   it under the terms of the GNU General Public License as published by
 3   the Free Software Foundation; version 2 of the License.
 4
 5   This program is distributed in the hope that it will be useful,
 6   but WITHOUT ANY WARRANTY; without even the implied warranty of
 7   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 8   GNU General Public License for more details.
 */

package csv;

import java.io.*;

/**
 * Trida reprezentujici CSV soubor
 *
 */
public class CSV {

	/**
	 * Moznosti razeni CSV souboru (textove X ciselne)
	 */
	public enum TypRazeni {
		TEXTOVE, CISELNE
	};

	/**
	 * cislo sloupce v CSV souboru
	 */
	public int cisloSloupce = 0;

	/**
	 * Zpusob razeni CSV souboru
	 */
	public TypRazeni typRazeni = TypRazeni.TEXTOVE;

	private static final int VELIKOST = 1024;

	/**
	 * Pole jednotlivych radku CSV soubrou
	 */
	private RadekCSV radky[] = new RadekCSV[VELIKOST];

	/**
	 * index radku CSV souboru
	 */
	private int indexRadku = 0;

	/**
	 * Pridani radku do CSV objektu
	 *
	 * @param radek
	 */
	public void pridejRadek(String radek) {
		radky[indexRadku] = new RadekCSV(this.typRazeni, this.cisloSloupce);
		radky[indexRadku++].policka = radek.split("[;|,]");
		if (indexRadku >= radky.length) {
			RadekCSV pomPole[] = new RadekCSV[2 * radky.length];
			System.arraycopy(radky, 0, pomPole, 0, radky.length);
			radky = pomPole;
		}
	}

	/**
	 * Setrizeni radku v CSV objektu
	 *
	 */
	public void sort() {
		java.util.Arrays.sort(radky, 0, indexRadku);
	}
```

```java
	/**
	 * Vypsani obsahu CSV objektu na vystup
	 *
	 * @param ps
	 */
	public void print(PrintStream ps) {
		for (int i = 0; i < indexRadku; i++) {
			radky[i].print(ps);
			ps.println();
		}
	}
}
```

ZDROJOVÝ KÓD ZK III: TESTOVACÍ PROGRAM, ČÁST TŘETÍ

- třída SortCSV (package input) – spouštěcí třída

```
/**
 1   This program is free software; you can redistribute it and/or modify
 2   it under the terms of the GNU General Public License as published by
 3   the Free Software Foundation; version 2 of the License.
 4
 5   This program is distributed in the hope that it will be useful,
 6   but WITHOUT ANY WARRANTY; without even the implied warranty of
 7   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 8   GNU General Public License for more details.
 */

package input;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintStream;

import csv.CSV;
import csv.CSV.TypRazeni;

/**
 * Spusteci trida pro spusteni programu (Serazeni radku v souboru dle zadanych
 * parametru)
 *
 */
public class SortCSV {
	/**
	 * objekt CSV
	 */
	private static CSV csv = new CSV();

	/**
	 * vstupni proud
	 */
	private static InputStream inp = System.in;

	/**
	 * vystupni proud
	 */
	private static PrintStream outp = System.out;

	/**
	 * Inicializacni metoda
	 *
	 * @param args
	 * @throws FileNotFoundException
	 */
	public static void main(String[] args) throws FileNotFoundException {
		zpracujArgumenty(args);
		BufferedReader inb = new BufferedReader(new InputStreamReader(inp));
		String str;
		try { // nahrani jednotlivych radku do csv objektu
			while ((str = inb.readLine()) != null)
				csv.pridejRadek(str);
		} catch (IOException e) {
			e.printStackTrace();
		}
		csv.sort(); // setridi jednotlive radky
		csv.print(outp); // vypise vysledek na vystup
	}

	/**
	 * Zpracuje argumenty z prikazove radky
	 *
	 * @param args
	 */
	private static void zpracujArgumenty(String[] args) {
		for (String s : args) {
			String option = s.substring(0, 2);
```

```
                        String value = s.substring(3);
                        if (option.equals("-f")) {
                                // ------------------------------------------------------------
------------------------------
                                try { // nastaveni vstupu
                                        inp = new java.io.FileInputStream(value);
                                } catch (FileNotFoundException e1) {
                                        System.err.println("Vstupni soubor nenalezen");
                                        e1.printStackTrace();
                                }
                                // ------------------------------------------------------------
------------------------------
                        } else if (option.equals("-o")) {
                                // ------------------------------------------------------------
------------------------------
                                try { // nastaveni vystupu
                                        outp = new PrintStream(value);
                                } catch (FileNotFoundException e) {
                                        e.printStackTrace();
                                }
                                // ------------------------------------------------------------
------------------------------
                        } else if (option.equals("-c")) {
                                // ------------------------------------------------------------
------------------------------
                                try { // zpracovani vyber primarniho sloupce
                                        csv.cisloSloupce = Integer.parseInt(value) - 1;
                                } catch (NumberFormatException e2) {
                                        System.err
                                                        .println("Spatny format -> zadejte
prosim CISLO sloupce");
                                        e2.printStackTrace();
                                }
                                // ------------------------------------------------------------
------------------------------
                        } else if (option.equals("-t")) {
                                // ------------------------------------------------------------
------------------------------
                                if (value.equals("number")) // zpracovani zpusobu razeni
                                        csv.typRazeni = TypRazeni.CISELNE;
                                else if (value.equals("text"))
                                        csv.typRazeni = TypRazeni.TEXTOVE;
                                else {
                                        System.err
                                                        .println("Spatne zadan zpusob razeni -
mozne volby: '-t=text' nebo '-t=number'");
                                        System.exit(0);
                                }
                                // ------------------------------------------------------------
------------------------------
                        }
                }
        }

}
```