Otvorený softvér vo vzdelávaní, výskume a v IT riešeniach
Žilina 2.-5. júla 2009

# UML/DSM nástroj UML .FRI

**JANECH, Ján, (SK)**

## 1 Úvod

Pri analýze softvérových produktov sa v stále väčšej miere využívajú rôzne CASE nástroje. Tie pokrývajú rôzne fázy vývoja softvéru. Niektoré sú vhodné na biznis analýzu, niektoré na návrh objektovej štruktúry, či návrh databázy. Niektoré, ako UML sa snažia pokryť celý vývojový proces. Mnohé z nich sú dokonca zamerané na konkrétne typy projektov, ako simulácie alebo multimediálne informačné systémy.

V priebehu štúdia informatiky na vysokej škole príde študent do styku s mnohými z týchto typov nástrojov. Vizualizácia štruktúr a algoritmov, ktorú prinášajú CASE nástroje je dobrou pomôckou pri vyučovaní a umožňuje jednoduchšie pochopenie preberanej látky. Okrem toho sa študent stretne s podobnými nástrojmi ako sa v danej oblasti používajú aj v praxi. Analýza problémov v CASE dokonca podporuje abstraktné myslenie a učí študentov nepozerať hneď na implementáciu v konkrétnom programovacom jazyku. Znamená to však používať niekoľko rôznych nástrojov na rôznych predmetoch, čo pre školu automaticky znamená veľké výdavky na ich zakúpenie. Okrem toho sa treba starať o pravidelnú aktualizáciu týchto nástrojov na nové verzie, čo tiež stojí peniaze.

Napriek širokému rozsahu zameraní, majú tieto nástroje veľa spoločného. Väčšinou sa jedná o zápis a prezentáciu informácii pomocou diagramov rôznych druhov a následné automatické spracovanie týchto diagramov. Líšia sa len typom podporovaných diagramov a spôsobom ich automatického spracovania. Špeciálnou výnimkou sú nástroje DSM[1]. Tie nemajú špeciálne zameranie (ako napr. E-R, BPMN,. . . ) a ani sa nesnažia pokryť všetky potreby jedným univerzálnym modelovacím jazykom (ako napr. UML). Namiesto toho definujú jazyk na definíciu modelov (metamodel) a výber, alebo návrh, konkrétneho modelovacieho jazyka zostáva na užívateľovi. Nechávajú mu teda slobodu vybrať si modelovací jazyk, ktorý sa mu najviac hodí na použitie v konkrétnej situácii.

[1] Domain Specific Modeling

Tento princíp sa dá s výhodou využiť pri vyučovaní. Stačí mať jeden nástroj a len zadefinovať metamodely podľa požiadaviek konkrétnych predmetov. Okrem toho tým vzniká možnosť využiť nástroj aj na predmetoch, kde žiadny nemali. Na Fakulte riadenia a informatiky Žilinskej univerzity v Žiline sa teda v roku 2005 začal vývoj vlastného DSM nástroja, ktorý by plne zodpovedal požiadavkám na nástroj používaný pri výučbe.

Nástroj je označovaný ako UML/DSM, čím je označené jeho hlavné zameranie na jazyk UML a technológie s ním spojené. Jeho názov je UML .FRI a stiahnuť sa dá pod licenciou GPL zo stránky `umlfri.org`.

## 2 Nástroj vyvíjaný študentmi

Nástroj UML .FRI je vyvíjaný študentmi Fakulty riadenia a informatiky v rámci projektovej výučby počas inžinierskeho štúdia. Študenti sa teda na projekte striedajú a každý rok nastúpi nová skupina. Na výmenu skúsenosti majú vždy dve skupiny vyhradený jeden semester. Tento spôsob vývoja má ako výhody, tak aj nevýhody.

Medzi nevýhody určite patrí pomalý vývoj. Vzhľadom na to, že sú študenti vyťažení štúdiom, práca na projekte prebieha pomaly. Okrem toho, ako už bolo spomenuté, jeden tím pracuje na projekte dva semestre, z ktorých jeden je vyhradený na výmenu skúseností a začlenenie tímu do projektu. Aby dohnali stratený čas, sú často nútení na projekte pracovať počas víkendov, sviatkov, alebo prázdnin. Napriek tomu, keby existuje vývojový tím, ktorý by sa plne venoval vývoju nástroja, určite by práce napredovali omnoho rýchlejšie.

Ďalšou nevýhodou je tiež určite nižšia kvalita kódu. Na projekt prichádza celá škála študentov, od tých najlepších až po tých najslabších. Napriek snahe vedúcich projektu, nie je možné skontrolovať zdrojové kódy vytvorené všetkými študentmi. Tým sa do projektu dostáva isté množstvo zle navrhnutého, resp. zle implementovaného kódu.

Medzi nesporné výhody určite patrí cena riešenia. Vývoj študentmi ako F/OSS riešenie minimalizuje náklady na vývoj. Zaplatiť stále treba potrebnú techniku, učebnice a softvér. Stále to je však lacnejšie, ako nechať softvér vyhotoviť externou firmou, alebo zamestnať na vývoj zamestnancov na plný úväzok.

Okrem toho, študenti sú schopní, v spolupráci s vedúcimi spomedzi učiteľov, lepšie špecifikovať požiadavky na daný softvér. Práve študenti budú tí, čo budú výsledný nástroj používať. Majú skúsenosti z rôznych nástrojov používaných na škole a dokážu povedať, aké chyby mali, ktorým sa treba vyhnúť.

Za výhodu je možné považovať aj to, že sú študenti ešte počas štúdia zaradení do reálneho projektu a nútení pracovať na ňom v tíme. Majú vďaka tomu možnosť zistiť, aké výhody plynú z dobrého návrhu. Vyskúšajú si rôzne nástroje na podporu prácu v tíme, ako verzionovací systém, alebo nástroj na správu chybových hlásení. Tak isto sa naučia technológie, s ktorými sa počas štúdia vôbec nestretli, alebo im bolo venované málo priestoru (Python, GTK+, XML, XML Schema, metaprogramovanie. . . ), čím si zlepšujú rozhľad.

## 2.1 Projektová dokumentácia

Ako bolo spomenuté vyššie, projekt je vyvíjaný generačne, pričom dve po sebe idúce generácie majú iba jeden semester na výmenu skúseností. Projekt však stále naberá na zložitosti. V dnešnej dobe už neexistuje človek, ktorý mal prehľad v celom projekte do podrobností, čo komplikuje plánovanie prác a komplikuje vývoj nových vlastností. To prispelo k tomu, že vznikla nutnosť vypracovať kvalitnú programátorskú dokumentáciu.

Tá sa začala vytvárať na dvoch úrovniach:

- *API dokumentácia* – je realizovaná ako generovaná dokumentácia vnútorného API aplikácie. Ako formát dokumentačných komentárov bol použitý pravidlami obmedzený Epytext[2]. Vďaka tomu sa dá jednoducho pomocou nástroja Epydoc vygenerovať prehľadná dokumentácia vo formáte HTML, alebo PDF. Táto dokumentácia je pravidelne generovaná skriptom každý deň o polnoci a vystavená na projektovej stránke. Vďaka tomu je dostupná aj tým vývojárom, ktorí nemajú Epydoc nainštalovaný.

- *Projektová dokumentácia* – opisuje princípy fungovania aplikácie, jej architektúru a technológie, na ktorých je založená. Dokumentácia je určená hlavne na zmiernenie problémov so zaúčaním novej generácie študentov. Písaná je preto ako ucelený dokument. Aby bola zabezpečená jednoduchosť editácie viacerými študentmi súčasne, bol za formát zvolený LaTeX zdielaný pomocou systému Subversion. Dokumentácia je síce verejná, ale jej editácia je momentálne prístupná len interným členom tímu UML .FRI. Dokumentácia je momentálne v štádiu riešenia.

## 2.2 Diplomové práce

Po skončení prác v rámci projektovej výučby majú študenti možnosť pokračovať na projekte vo forme diplomovej práce. V takom prípade je vybraná jedna konkrétna problematika, ktorú potom po teoretickej aj praktickej stránke spracuje študent ako svoju diplomovú prácu.

Ako prvé sa v rámci diplomových prác riešili transformácie M2T[3] a T2M[4]. Študenti navrhli kompletný transformačný jazyk, vďaka ktorému je možné jedným zápisom definovať obe transformácie. Okrem toho byli vytvorené transformácie na generovanie zdrojových kódov (C++, Delphi Pascal a Python) a reverzné inžinierstvo pre diagram tried a generovanie textovej dokumentácie z modelu UML vo formáte HTML. Tieto transformácie môžu uľahčiť prácu s nástrojom na predmetoch, na ktorých sa vyučuje algoritmizácia, objektové programovanie, údajové štruktúry a podobne, lebo z modelu umožňujú vygenerovať (po zadefinovaní transformácie) kompletný zdrojový kód.

Ďalšou diplomovou prácou je systém zásuvných modulov[5]. Ten umožňuje rozšírenie nástroja o automatické spracovávanie diagramov. Napríklad môžu vykonávať rôzne algoritmy

[2] Formátované dokumentačné reťazce podobné formátu javadoc

[3] Model-to-Text – transformácia modelu do textovej podoby

[4] Text-to-Model – reverzné inžinierstvo zdrojových kódov do modelu

[5] anglicky plugin

z teórie grafov, alebo asistovať študentovi pri tvorbe modelu. Zásuvné moduly môžu byť vytvorené v takmer ľubovoľnom jazyku, vďaka čomu ich môže vytvárať každý so základnými znalosťami objektového programovania.

Zatiaľ posledná diplomová práca rieši implementáciu systému UNDO/REDO[6]. Na jeho implementáciu existuje niekoľko algoritmov, cieľom študenta bolo nájsť najvhodnejší a upraviť ho pre potreby nástroja UML .FRI. Okrem toho musel identifikovať miesta v nástroji, ktoré bránia implementovať systém UNDO/REDO a nájsť ich riešenie.

V budúcnosti sa plánuje niekoľko ďalších diplomových prác (transformácie M2M[7], podpora pre elektronické tabule. . . ), ale aj bakalárske práce na jednoduchšie časti nástroja (napr. import projektov uložených v iných nástrojoch). Bakalárske práce môžu slúžiť ako úvod do projektu, čím by sa mohol zjednodušiť nástup novej generácie na projekt v inžinierskom štúdiu. Všetky diplomové a bakalárske práce by mali zjednodušiť nasadenie nástroja vo vyučovacom procese, alebo priniesť do neho nové možnosti.

# 3 Použité technológie

Celý systém je naprogramovaný v jazyku Python. Implementácia mohla byť vďaka tomu na niektorých miestach zjednodušená dynamickým typovaním a využitím metaprogramovania. Zdrojové kódy sú vďaka tomu kratšie a prehľadnejšie, ako v iných programovacích jazykoch. Jazyk Python však umožňuje niektoré konštrukcie, ktoré odporujú praktikám zaužívaným v objektovom programovaní (hlavne neobsahuje nástroje na vynútenie zapuzdrenia), čo si vynútilo vydanie príručky „Coding style[8]", ktorá popisuje, čo môžu študenti využívať a čo nie. Tá je súčasťou projektovej dokumentácie. Z istého pohľadu je to aj výhoda, lebo sa tým učia určitej programátorskej disciplíne.

Vzhľadom na to, že programovací jazyk Python neobsahuje žiadne východzie knižnice pre tvorbu GUI rozhrania, bolo treba vybrať externú knižnicu. Zvolená bola GTK+. Je to multiplatformová knižnica na tvorbu GUI. Vďaka kombinácii Python a GTK+ je možné portovať aplikáciu na takmer ľubovoľnú platformu, od OS GNU/Linux, MS Windows, až po mobilné telefóny, alebo rôzne zabudované systémy. Knižnica je napísaná v jazyku C, čo čiastočne obmedzuje možnosti objektového programovania. Preto bola do aplikácie pridaná podpora pre objektové zapuzdrenie prezentačnej vrstvy.

Na vykresľovanie diagramov je použitá knižnica Cairo. Tá umožňuje jednoduchý spôsob práce s vektorovou aj bitmapovou grafikou. Výsledné obrázky dokáže prezentovať priamo užívateľovi, ale aj exportovať do rôznych vektorových a bitmapových formátov. Okrem toho prináša výhody hardvérovej akcelerácie, podporuje anti-aliasing, transformačné matice a podobne.

Na ukladanie štruktúrovaných dát bol zvolený formát XML. Jeho hlavnými výhodami

[6] vrátiť poslednú činnosť/zopakovať činnosť

[7] Model-to-Model – transformácia modelu na iný model

[8] Štýl programovania

sú jednoduché spracovanie pomocou technológií DOM alebo Element Tree a možnosť jednoduchej validácie voči definícii vo formáte XML Schema. Uložené modely sú navyše komprimované vo formáte ZIP. Je to v súčastnosti pomerne často používaná kombinácia. Študenti sa teda zoznámia s možnosťami spracovania tohoto formátu, jeho výhodami i nevýhodami.

Kvôli rozsahu projektu sa študenti nezoznámia do podrobností s každou uvedenou technológiou. Na tom sa prejaví aj ich schopnosť tímovej práce. Každý študent musí pracovať na inej časti a skrývať detaily použitej technológie za vopred dohodnuté rozhrania vnútornej objektovej štruktúry.

# 4 Nástroj používaný študentmi

Ako bolo uvedené vyššie, nástroj UML .FRI je určený hlavne pre podporu vyučovania. Cieľovou skupinou sú teda študenti. Tomu je teda prispôsobený aj systém vývoja. Študenti na niektorých predmetoch sú motivovaní, aby nástroj testovali a hlásili chyby.

Plánu rozšíriť nástroj na začiatku na Fakulte riadenia a informatiky boli podriadené aj používané nástroje. Najdôležitejší spôsob spätnej väzby – hlásenie chýb je implementovaný dvoma rôznymi spôsobmi.

- *Hlásenie chýb priamo z aplikácie* – V prípade, že aplikácia spôsobí automaticky detekovateľnú chybu, užívateľovi sa zobrazí okno s chybovou správou. To priamo obsahuje možnosť automatického odoslania informácie o chybe na server. Hlásenie okrem samotnej chybovej správy voliteľne obsahuje užívateľov projekt, aby mali vývojári na čom testovať, a krátky popis chyby od užívateľa.

- *Hlásenie chýb pomocou systému na správu chybových hlásení* – Na internetovej stránke projektu sa nachádza odkaz na systém na správu chybových hlásení[9]. Užívateľ je síce nútený sa zaregistrovať, ale registrácia je voľná a nie je ani viazaná na štúdium na Fakulte riadenia a informatiky. Vzhľadom na cieľovú skupinu je celé rozhranie v slovenčine a tak isto chybové hlásenia sa píšu po slovensky.

Nasadenie aplikácie na vyučovaní prebieha postupne podľa toho, ako sa do aplikácie implementujú nové vlastnosti. Už pred zaradením nástroja do výučby niektorého predmetu ho používalo niekoľko študentov na tvorbu modelov do semestrálnych prác. Bolo ich však málo a tak spätná väzba od študentov bola malá.

Ako prvé prebehlo nasadenie na predmet Objektové programovanie v druhom ročníku, kde boli študenti bodmi motivovaní hlásiť chyby v aplikácii. Študenti začali nástroj používať aj na tvorbu semestrálnych prác. Spätná väzba bola už väčšia, študenti boli s nástrojom spokojní, dokázal čo potrebovali (jednoduché UML diagramy tried). Napriek tomu sa našlo dosť veľké množstvo chýb, ktoré boli členmi tímu odstraňované priebežne.

[9] Bug tracking system

Ďalší predmet, na ktorom sa nástroj používal, bol predmet Základy informatiky 2 v prvom ročníku. Na tomto predmete sa využíval na modelovanie vývojových diagramov. Bolo to však tento semester a na spätnú väzbu od študentov v dobe písania tohoto článku sa ešte len čaká.

Budúci rok sa plánuje využitie nástroja na predmete Informatika 1,2 v prvom ročníku. Je to nový predmet, ktorý nahrádza Základy informatiky a bude sa na ňom vyučovať metódou „Objects first"[10]. Preto bude treba nástroj na modelovanie UML diagramov tried.

Napriek tomu, že v súčastnosti sa využívajú iba dva typy modelov, v pláne je implementovať v budúcnosti aj iné (grafy, DFD, BPMN), aby bolo možné nástrojom UML .FRI nahradiť iné, doteraz využívané nástroje. Študenti tým získajú jednotné prostredie, ktoré bude plne vyhovovať požiadavkám daných predmetov.

# 5 Metamodel

Ako už bolo uvedené, medzi najväčšie výhody nástroja patrí oddelenie metamodelu od zdrojových kódov aplikácie. Tým sa získava veľká flexibilita nástroja a možnosť jeho využitia v rôznych oblastiach. Aplikácia je schopná automaticky rozpoznať typ modelu a na základe toho zvoliť príslušný metamodel. V počítačových laboratóriách na škole teda stačí mať nainštalovaný nástroj UML .FRI jeden krát, ale s niekoľkými metamodelmi. Na predmete objektové programovanie teda študent vyberie šablónu modelu pre objektové programovanie a automaticky sa zvolí metamodel pre UML. Ak si na predmete Základy informatiky vyberie šablónu pre tento predmet, zase sa zvolí príslušný metamodel.

Väčšinu metamodelov budú definovať študenti v rámci prác na projektovej výučbe. Vzhľadom na otvorený formát metamodelu definovaný pomocou štruktúr XML si dokáže vytvoriť metamodel každý s príslušnými znalosťami daného typu modelov.

Metamodel v dobe písania tohoto článku obsahuje 4 typy objektov.

- *Objekt typu doména* – obsahuje definíciu metadát pre ostatné typy objektov (element, spojenie). Popisuje, aké parametre (atribúty) môže užívateľ pre daný objekt zadať. Ku každému atribútu definuje typ a implicitnú hodnotu, ktorá bude nastavená pri vytvorení objektu. Domény, ako jediný typ objektov môžu byť vnorované, teda každý atribút môže mať ako typ uvedenú doménu, alebo zoznam položiek s definovanou doménou.

- *Objekt typu element* – reprezentuje element diagramu. Elementy môžu byť napríklad triedy, tabuľky v databáze, vrcholy grafu a podobne. Definícia elementu obsahuje rôzne informácie od názvu elementu, až po jeho vzhľad.

- *Objekt typu spojenie* – reprezentuje spojenie medzi elementmi. Spojenie je tvorené lomenou čiarou a voliteľne aj množinou popisných políčok. Tie sa zobrazia na určených miestach pri čiare spojenia a zobrazujú užívateľom definované atribúty spojenia.

[10] Začína sa s objektovo orientovaným programovaním


Obrázok 1: Príklad vzhľadu elementu

- *Objekt typu diagram* – reprezentuje diagram v modeli. Definícia diagramu obsahuje zoznam povolených elementov a spojení pre daný typ diagramu. Plánovaná je aj podpora domén pre diagramy, aby mohol užívateľ definovať aj hodnoty atribútov diagramu.

Špeciálna pozornosť pri návrhu metamodelu bola venovaná definícii vzhľadu elementov a spojení. Toto je oblasť, ktorej mnohé konkurenčné DSM nástroje nevenujú veľkú pozornosť. Ak sa však má nástroj používať na výučbu existujúcich modelovacích nástrojov, ktoré sú často štandardizované aj čo sa týka vzhľadu, je nutné aby bol grafický výstup správne sformátovaný, aby neboli študenti zmätení, keď začnú pracovať s konkurenčným nástrojom.

Na zápis definície vzhľadu bol zvolený systém kontajnerov. V zápise existujú tri typy objektov.

- *Komplexný kontajner* v sebe môže obsahovať ľubovoľný počet iných objektov. Komplexné kontajnery existujú v systéme dva: s horizontálnym usporiadaním a s vertikálnym usporiadaním.

- *Jednoduchý kontajner* môže obsahovať len jeden objekt. Ako jednoduchý kontajner je implementovaná väčšina grafických objektov. Napríklad obdĺžnik, elipsa, tieň. . .

- *Elementárny objekt* neobsahuje ďalšie objekty. Je to text a oddeľovacia čiara.

Vždy platí, že najskôr sa vykreslí samotný kontajner a až následne na to sa vykreslia objekty vnútri v ňom. Druhý a posledný princíp je, že veľkosť v akej sa vykreslí kontajner je určená veľkosťou objektov v ňom. Celý princíp sa dá ukázať na nasledujúcom príklade. Na obrázku 1 je uvedený vzhľad elementu definovaného touto štruktúrou:

```
<Shadow padding="3" color="gray">
    <Rectangle fill="lightyellow" border="black"
                right="10 rounded" left="10 rounded">
        <Padding padding="5">
            <Align align="center center">
                <TextBox text="END" font="Arial 10" color="black" />
            </Align>
        </Padding>
    </Rectangle>
</Shadow>
```

Na najvyššej úrovni je tieň. Ten sa vykresľuje ako prvý. Tieň obsahuje obdĺžnik so zaoblenými stranami. Na úplne najnižšej úrovni je text a teda jeho veľkosti sa prispôsobuje celý vzhľad elementu.

Tento zápis umožňuje jednoducho definovať takmer ľubovoľný vzhľad elementu. Vďaka tomu je možné pokryť aj takú zložitú normu ako je UML. Okrem toho kontajnerový spôsob definície sa postará automaticky o určovanie veľkosti podľa užívateľom zadaných atribútov elementu a teda sa nestane, že by text pretekal mimo vyhradenej plochy.

# 6 Záver

Pri výučbe rôznych predmetov na vysokej škole je výhodné používať jednotné prostredie CASE. Keď sa k tomu priráta cena, je nástroj UML .FRI schopný konkurovať rôznym komerčným nástrojom. Jeho výhodou je veľká flexibilita, ktorá mu umožňuje pokryť kompletne niekoľko oblastí v rámci jednej inštalácie.

Flexibilita však nie je jediné čo musí nástroj podporovať, aby bol použiteľný vo výučbe. Do budúcnosti zostáva na nástroji ešte veľa práce, aby bol schopný pokryť čo najväčšiu časť vyučovacieho procesu. Je potrebné pridávať nové metamodely a rozširovať existujúce, podľa požiadaviek jednotlivých predmetov a pridávať nové možnosti samotného softvéru, podľa požiadaviek užívateľov.

**Poďakovanie**

Táto práca vznikla s podporou grantovej agentúry KEGA v rámci riešenia projektu 3/2158/04 „Využitie open source softvéru vo výučbe na vysokých školách“.

# Literatúra

[1] BAČA, Tomáš – JURÍČEK, Milan – ODLEVÁK, Pavol: *UML .FRI – Case tool developement*. In: *Journal of Information, Control and Management Systems*. 2008. vol.6, n°1. ISSN 1336-1716.

[2] *DSM Forum*. [online].
Dostupné online: <http://www.dsmforum.org>.

[3] JANECH, Ján – SADLOŇ, Ľubomír: CASE UML .FRI. In: *OBJEKTY 2007*. Ostrava : Fakulta elektroniky a informatiky, VŠB – Technická univerzita Ostrava, November 2007. ISBN 978-80-248-1635-7.

[4] Wikipedia: *Domain-specific modeling — Wikipedia, The Free Encyclopedia*. [online].
Dostupné online: <http://en.wikipedia.org/wiki/Domain-specific_modeling>.

**Kontaktná adresa**

**Ján JANECH,**
Katedra softvérových technológií FRI ŽU v Žiline, Univerzitná 1,
010 26 Žilina, jan.janech@kst.uniza.sk