Excel@FIT 2015
http://excel.fit.vutbr.cz

Automatické generování UML diagramu tříd

Martin Brázdil*

Abstrakt
K optimalizaci procesu vývoje produktu softwarové aplikace, tedy produktu v IT oblasti, na kterou se práce zaměřuje, je třeba mít přehled o aktuálním postupu. Tato práce cílí na oblast architektury vyvíjeného programu – UML diagramy tříd. Aplikace je koncipována jako webová služba psaná pro platformu C# .NET s rozšířenou podporou platformy Java. Po propojení služby s projektem (přeloženými zdrojovými kódy aplikace), se pak lze dotazovat dané služby na diagramy tříd potřebných částí architektury. Jelikož jsou vygenerované diagramy dostupné ve tvaru obrázků z webového prostoru, je možno je integrovat i do online dokumentace (např. Wiki stránky). Po úpravě zdrojového projektu a jeho analýze, se vložený diagram sám aktualizuje, čímž se udržuje neustálá konzistence mezi reálnou implementací a dokumentací. Výsledná aplikace poskytuje vývojovému týmu informace o aktuálním stavu architektury jeho produktu, čímž dopomáhá ke sdílení informací uvnitř týmu, prezentaci a případné kontrole doposud odvedené práce.

Klíčová slova: Diagram tříd — Webová služba — Reflexe — C# .NET — Java

Přiložené materiály: Ukázkové video

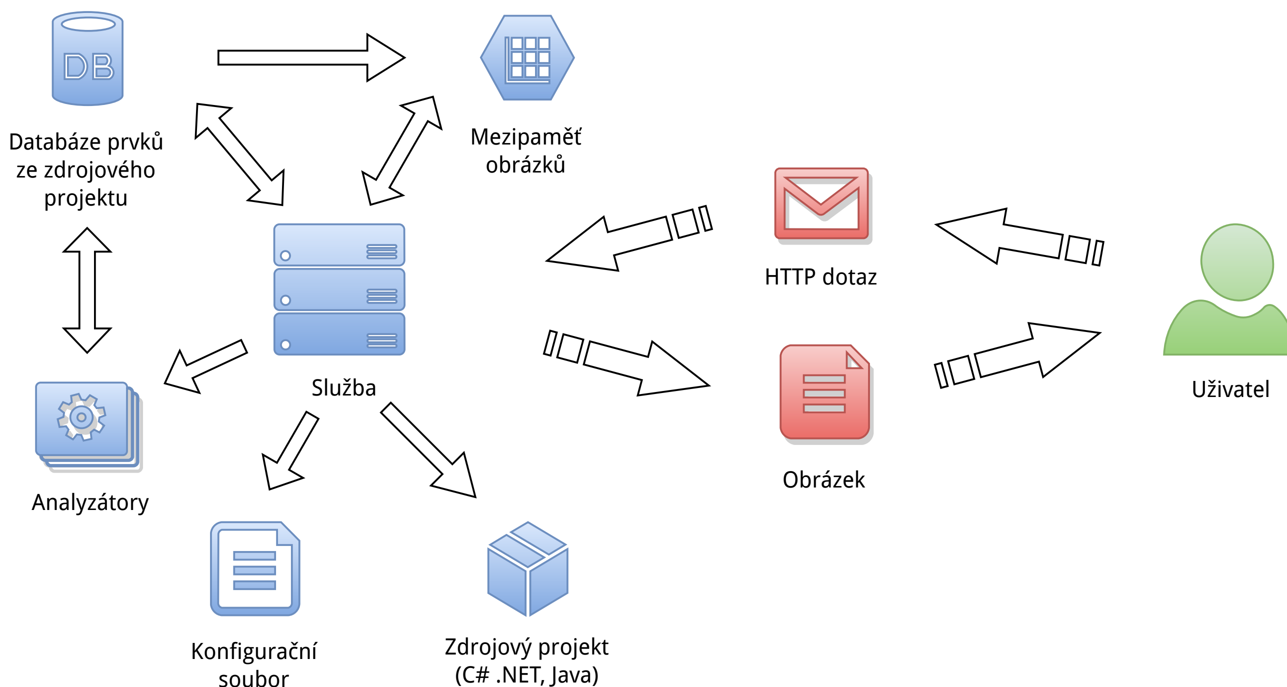
*xbrazd14@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

1. Úvod

[Motivace] V dnešní době agilních přístupů k vývoji softwarových produktů, se může stát, že tým, jakožto celek, ztratí přehled nad aktuálním postupem vývoje z pohledu architektury aplikace. A vzhledem k jednomu z bodů agilního manifestu – „Fungující software před vyčerpávající dokumentací" [1] – není na dokumentaci produktu ve fázi vývoje kladen příslušný důraz, protože by mohla zpomalovat vývoj samotný. Projektový vedoucí však musí mít přehled a prezentovat dosažené výsledky svému zadavateli. Z výše uvedeného plyne motivace této práce, a sice vytvořit nástroj, který dopomůže k optimalizaci procesu udržení konzistentních informací o aktuální architektuře.

[Definice problému] Jádrem problému je přístup k informacím o architektuře vyvíjené aplikace. Tyto informace musí být vždy aktuální, přehledné a přístupné. Proto výsledné řešení musí obsahovat jednak možnost jednoduše specifikovat zdrojový projekt, respektive přeložené zdrojové kódy do spustitelné podoby včetně periody automatické aktualizace analyzovaných dat, a jednak snadný přístup k získání diagramů tříd unifikovaného modelovacího jazyku (dále jen UML) dle specifických potřeb uživatele.

[Existující řešení] Na poli vývoje softwarových produktů existují nástroje, které usnadňují fázi návrhu, potažmo implementaci nebo testování. Obecně se nazývají CASE nástroje (angl. *Computer Aided Software Engineering*). Pro vizualizaci se standardně využívají diagramy z rodiny UML. Jedná se o rozsáhlé desktopové aplikace pokrývající širokou oblast modelovacích a simulačních technik pro vývoj softwarových produktů. Jejich grafická uživatelská rozhraní jsou



Obrázek 1. Architektura aplikace a komunikace mezi uživatelem a webovou službou

založena na neustálé interakci s uživatelem. Výsledné modely nelze aktualizovat automaticky, je nutno manuálně buď upravit diagramy, nebo importovat podporované spustitelné soubory do nástroje (z diagramů se však vytratí editované rozložení prvků). CASE dále umožnují i využít reverzního inženýrství[1] nad již spustitelným programy navrženými pro platformy C# .NET nebo Java. Popsané vlastnosti dokládají nevhodnost pro využití ve stanovených vstupních podmínkách. Tato práce, potažmo aplikace, určitě nepředstavuje náhradu CASE nástrojů, nýbrž rozšíření hlavní ideji jejich použití – usnadnění práce vývojářům.

[Řešení] Aplikace je rozdělena do tří částí (blíže popsané v následujících kapitolách):

1. analýza přeložených zdrojových kódů v platformách C# .NET a Java,
2. tvorba diagramů tříd ve formě obrázků na základě analyzovaných dat
3. a přístup k aplikaci skrze webové rozhraní.

Pro každý softwarový produkt se spustí webová služba s potřebným konfiguračním souborem, který definuje umístění analyzovaných souborů, periodu aktualizace dat apod. Koncovému uživateli pak služba poskytuje diagramy tříd ve formě obrázků, ať už rastrové, nebo vektorové, ná základě jeho potřeb definovaných v dotazovacím jazyce skrze HTTP dotaz.

[Přínos] Pokud jste členem vývojového týmu nebo projektovým vedoucím softwarového produktu v podporovaných platformách a potřebujete mít aktuální a konzistetní přístup k vyvíjené architektuře (diagramům tříd), pak je tento produkt určen právě vám. Dalším z hlavním přínosů je jeho jednoduchost při uplatnění díky využití automatizace.

[1] Elliot Chikovsky definoval reverzní inženýrství jako proces analýzy nějakého systému, jehož cílem je odhalení komponent, provázanosti a popsání daných skutečností odlišnou formou. [2]

2. Architektura řešení

Aplikace je koncipována jako webová služba (angl. *Web Service*) bez uživatelského rozhraní rozdělena do dvou vrstev – vrstva pro analýzu vstupního zdrojového projektu přeloženého z platformy C# .NET nebo Java (kapitola 3) a vrstva pro generování diagramu tříd do obrázku na základě dotazu od uživatele (kapitola 4). Obě vrstvy však budou spolupracovat s jednou databází a konfiguračním souborem.

Schéma komunikace se službou, znázorněné na obrázku 1, obsahuje dva hlavní aktéry – službu samotnou a koncového uživatele. Uživatelem služby je vzdálený klient, konkrétně pak klientský systém fyzického uživatele, vzdálený server nebo další služba. Uživatel na svůj dotaz, v případě korektního zadání doménového jména (identifikátoru) vedoucího ke spuštěné službě, obdrží obrázek s požadovaným diagramem tříd.

Základní komponenty komunikace mezi uživatelem a službou popisuje následující výčet:

Relační databáze slouží k uložení informací o analyzovaném projektu tak, aby nebylo nutné pro každý dotaz neustále analyzovat celý zdrojový projekt a postačovalo použít jen výsledky analýzy pro sestavení požadovaného diagramu tříd. Pro

implementaci konektoru databáze k aplikaci je využito knihovny `NHibernate 4` a díky tomu se podpora databázových systémů značně rozrostla. [3]

Analyzátory jsou spustitelné soubory, které provedou analýzu zdrojového projektu na základě jeho platformy (C# .NET nebo Java). Analyzátorům se blíže věnuje kapitola 3.

Mezipaměť obrázků poskytuje posledních *n* výsledků dotazů nebo výsledky často opakovaných dotazů. Definici hodnoty *n* nebo nutného počtu opakování obsahuje konfigurační soubor služby.

Konfigurační soubor poskytuje určitou míru přizpůsobitelnosti použití navrhované služby. Jsou zde uloženy informace o umístění zdrojového projektu, umístění nutných nástrojů podpory služby, konfiguraci mezipaměti obrázků, konfiguraci databáze, čas spuštění analýzy, perioda aktualizace analyzovaných dat apod. Každá instance služby musí disponovat vlastním konfiguračním souborem, který se předá skrze parameter při spuštění.

Zdrojový projekt je umístěn, stejně jako konfigurační soubor, na službě přístupném místě, ať už se jedná o lokální nebo vzdálené umístění. Služba podporuje analýzu pro projekty vytvořené v platformách C# .NET a Java.

HTTP dotaz zasílá koncový uživatel, ve kterém specifikuje informace o chtěném diagramu tříd. Má podobu URI s přenosovým protokolem HTTP, tedy `GET` dotaz. Uživatel tak jednoduchým způsobem vytvoří obyčejný odkaz, který vrací obrázek.

Služba na dotaz vždy odpoví **obrázkem** buď sestaveného diagramu tříd, nebo chybovým obrázkem obsahující chybovou zprávu. Tímto je zajištěna její jednotnost při použití. Obrázek může mít jak rastrovou, tak i vektorovou podobu, záleží na preferencích uživatele uvedených v parametrech dotazu.

Služba je postavena na platformě .NET a psaná v jazyce C#. Tento fakt umožňuje využít základní vlastnosti zvolené platformy a sice *reflexe*, která je přehlednější a také snazší pro realizaci analýzy vstupního přeloženého projektu na stejné platformě, než jakou by poskytovala statická analýza srkze .NET IL Assembler (mezijazyk platformy C# .NET).

Vstupní projekty mohou být buď typu spustitelného `*.exe` souboru, případně sdílené knihovny `*.dll`, v platformě C# .NET, nebo kompresní soubory platformy Java typu `*.jar`, `*.war` (soubor webového modulu) nebo `*.ear` (soubor obsahující předešlé typy). Kompresní formáty typu `*.war` a `*.ear` jsou rozbaleny do dočasného pracovního adresáře (ve Windows se jedná o známý `Temp` adresář). Odtud se poté analyzují a konvertují `*.jar` soubory do souborů platformy C# .NET, čemuž se věnuje kapitola 3.1.

3. Analyzování spustitelných aplikací

Analyzované spustitelné soubory platformy C# .NET lze za běhu načíst do analyzátoru a využít přístupu zvaného reflexe. To vše za předpokladu, že implementace analyzátoru je realizována na stejné platformě. V případě platformy Java se využívá nástroje IKVM, tento postup je nastíněn v kapitole 3.1.

Při překladu se zdrojové kódy v C# .NET transformují do spustitelného mezijazyka zvaného `.NET IL Assembler` z rodiny `CIL` (Common Intermediate Language). [4] Tato technologie pak umožňuje za běhu analyzátoru načíst přeložené soubory a využít přístupu k metadatům (data o přeloženém kódu) za pomocí *reflexe*. Proto přístup k informacím typu název třídy, parametry metod (název a datový typ), parametrizované generické třídy a metody apod., je poměrně snadný v porovnání s čistou statickou analýzou mezikódu. To je ovšem vykoupeno pomalejším přístupem k datům. Určitou optimalizaci v rychlosti přístupu a bezpečnosti představuje načtení analyzovaného projektu do jedné, separátní aplikační domény[2]. Tím dojde k ušetření času potřebného ke komunikaci mezi více doménami, což využívá obecný přístup načtení každé sestavené jednotky (zvané `assembly`) přeloženého kódu do vlastní domény.

Jakmile je celý analyzovaný projekt, včetně jeho patřičných knihoven, načten, proběhne samotná analýza prvků platformy C# .NET (tyto prvky specifikuje standard ECMA-334 [5]) ve dvou krocích:

1. Analýza *struktury* jednotlivých komponent (jmenných prostorů, tříd, výčtů, rozhraní, struktur a delegátů) v celém projektu.
2. Analýza *vazeb* mezi komponentami (závislosti, zobecnění, realizace, vnořování a asociace). Upřesňující vztahy asociace typu agregace a kompozice nelze z dat získat, neboť souvisí se sémantikou použití.

Analyzátor informuje uživatele o svém postupu prostřednictvím textového záznamu uloženého v souboru specifikovaném v konfiguračním vstupu navržené aplikace. Jakákoliv chyba je uvedena v záznamu a přeskočena, tak aby nedošlo po aktualizaci projektu

[2] Aplikační doména představuje izolovaný prostor, podobně jako procesy, ovšem na rozdíl od nich mohou bez problému aplikační domény vzájemně komunikovat. [6]

	základ	atribut	vlastnost	událost	indexer	operace	getter	setter
`public`	+	▲	■	◎	⊞	●	<	>
`protected`	#	▲	■	◎	⊞	●	<	>
`internal`	∼	▲	■	◎	⊞	●	<	>
`private`	–	▲	■	◎	⊞	●	<	>

Tabulka 1. Vizualizace modifikátorů viditelnosti jednotlivých složek třídy

k potencionálnímu znepřístupnění již vytvořených diagramů tříd.

Výsledek analýzy je uložen v databázi, kterou následně používá generátor diagramu tříd. Případný uživatel se tak může rozhodnout pro implementaci vlastního analyzátoru, ať už pro C# .NET, Java nebo jakýkoliv jiný objektově orientovaný jazyk, právě díky použití databáze. Musí ovšem zachovat její strukturu, především uchování záznamu s kořenovým jmenným prostorem, který má prázdný název a je přístupovým bodem k datům.

3.1 Java

Analyzátor disponuje také podporou platformy Java. Ta je samozřejmě odlišná od C# .NET, neboť používá svoji vlastní syntaxi zdrojových kódů, mezikód a obecně i prostředí pro běh aplikací zapsaných na této platformě. Odlišnost lze částěčně eliminovat použitím nástroje IKVM [7], který převede Java mezikód do CIL platformy .NET.

Následně je třeba do programu zavést knihovny nástroje IKVM, tak aby bylo možné provést patřičnou analýzu. Konvertovaný Java mezikód, také nazývaný `bytecode`, se však liší od CIL čistě přeloženého kódu z platformy C# .NET. Největší rozdíly představují:

- *Výčty* – v Java se jedná o speciální typ třídy, což znamená, že na rozdíl od C# může obsahovat mimo jiné i operace.
- *Implementace v rozhraní* – v nejnovější verzi Java 8 lze implementovat metody přímo v rozhraní, nazývají se výchozí (angl. *default*) metody. Mimo to Java 8 podporuje i statické metody v rozhraní. [8] Analyzátor pokládá výchozí metody za abstraktní tak, aby je bylo možno graficky odlišit od ostatních metod rozhraní.
- *Parametrizovatelná generika* – informace o generických třídách a metodách, respektive generických parametrech, překladač Java neukládá do mezikódů v podobě datových typů, ale přímo je nahrazuje za navázané reálné datové typy. Původní informace je uložena v anotaci daného prvku, která osahuje *typovou signaturu* v podobě řetězce.

Konkrétně pak pro typovou signaturu je nutné vytvořit vlastní syntaktický analyzátor (ná základě Java specifikace [8]), který zpracuje řetezec a získá parameterizovaná generika, která lze uložit do databáze.

4. Generování diagramů

Generátor, respektive vizualizátor, diagramů tříd ve formě obrázků tvoří mnou napsaná knihovna, využívající další knihovnu `QuickGraph` [9] pro zaobalení grafových uzlů a hran z používaných entit. Uzly jsou dvojího druhu: uzel tříd (případně struktury, výčtu apod.) a uzel jmenného prostoru. Uzly jsou propojeny hranami, které se liší dle daného typu vztahu (dědičnost, asociace apod.). Grafická reprezentace odpovídá standardu UML 2.4.1 [10] tak, aby byla zajištěna srozumitelnost diagramu. Pro zřetelnější rozlišení modifikátorů viditelnosti a typů prvků tříd jsem navrhl grafickou notaci, která je viditelná v tabulce 1. Dále se v jednotlivých třídách obsažené prvky řadí do dvou skupin (oddělené horizontální čarou):

- *Stavy* v pořadí - prvky výčtu (pokud se jedná o výčet), atributy, vlastnosti, události a indexery.
- *Operace* v pořadí - konstruktory, destruktory, operace typu getter (pak jeho případný setter), operace typu setter (dosud nevypsané) a ostatní operace.

Jednotlivé podskupiny jsou seřazeny v abecedním pořadí vzestupně. Navíc aplikace umožňuje zobrazit existenci operací typu getter a setter (přístupové metody) pro prvky typu vlastnost přímo okolo modifikátoru viditelnosti pomocí ostrých závorek ($<$ zleva pro getter a $>$ zprava pro setter).

Navržená knihovna pracuje ve třech navazujících krocích:

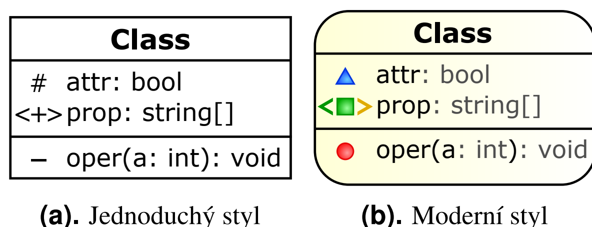
1. Tvorba grafových uzlů a hran z analyzovaných entit.
2. Vygenerování DOT[3] řetezce z grafové struktury.
3. Vykreslení DOT do obrázku, k čemuž se využívá nástroj `Graphviz`. Ten podporuje nepřeberné množství formátů jak rastrových, tak i vektorových. [11]

Zobrazení diagramu, který obsahuje všechny třídy, jejich prvky, vazby apod., může být značně nepřehledné,

[3] DOT je jednoduchý textový jazyk sloužící pro popis grafových struktur.

zvláště u rozsáhlých projektů a jak již bylo zmíněno v kapitole 2, diagramy jsou přístupné skrze webové rozhraní pomocí HTTP `GET` dotazu, který je možno parameterizovat. Lze specifikovat zdrojový jmenný prostor (případně třídu), úroveň zanoření ve struktuře jmenných prostorů (tříd) na základě hierarchie vazeb, úroveň detailu zobrazení a grafický styl diagramu.

Úrovní detailu zobrazení se specifikuje, jaké typy vztahů, operací nebo stavů tříd vizualizovat a zda zjednodušit operace typu getter a setter na zobrazení u atributu viditelnosti u stavu typu vlastnost. Aplikace disponuje dvěma základními typy grafických stylů diagramů tříd (viz obrázek 2), jedním klasickým hranatým černobílým a moderním zaobleným s přechody a rozlišením datových typů od názvů pomocí šedé barvy. Tyto styly lze roršířit použitím barevných modifikátorů viditelnosti dle uvážení uživatele.



(a). Jednoduchý styl **(b).** Moderní styl

Obrázek 2. Náhled základních grafických stylů generovaných diagramů tříd

5. Závěr

Práce představuje jeden ze způsobů optimalizace procesu vývoje softwarového produktu – aplikace – z pohledu udržení konzistentních informací mezi členy vývojého týmu a jeho prezentaci doposud odvedené práce svému okolí. Proto je práce zaměřena na architektru aplikace, k jejíž reprezentaci slouží unifikovaný modelovací jazyk, konkrétněji pak diagramy tříd a další.

Výsledná aplikace dokáže zpracovat již přeložené zdrojové kódy platforem C# .NET a Java. Takto zpracovaná data použije pro generování UML diagramu tříd ve formátu prostého obrázku na základě HTTP dotazu obdrženého webovou službou. Uživatelům je tedy poskytnuta možnost si z celé architektury zobrazit jen potřebnou část, dokonce s omezením zobrazených detailů na jejich preferované úrovní.

Pro budoucí vývoj by bylo vhodné více optimalizovat analyzování vstupních dat a případně rozšířit podporu grafického zpracování diagramů samotných. Z dostupných potřeb spolupracujících vývojových týmů nevyplynula žádost o grafické rozhraní pro konfiguraci aplikace ani pro generování diagramů, stálo by ovšem za zvážení jej vypracovat. Koncepce aplikace přímo vybízí uživatele, jejichž potřeba podpory programovacích jazyků (platforem) je vyšší, než lze nyní poskytnout, aby si napsali analyzátor sami za předpokladu, že výsledná data budou v korektním formátu pro generátor diagramů tříd.

Poděkování

Rád bych poděkoval doc. RNDr. Jitce Kreslíkové, CSc., a konzultatům ze společnosti Siemens, s. r. o., Ing. Janu Vernerovi a Ing. Zdeňku Jurkovi za cenné rady a vedení při vývoji aplikace a tvorbě této práce. Tento příspěvek byl podpořen VUT FIT grantem FIT-S-14-2299, „Výzkum pokročilých metod ICT a jejich aplikace“.

Literatura

[1] BECK, Kent, aj. *Manifest Agilního vývoje software* [online]. ©2001 [cit. 2015-03-28]. Dostupné z: http://agilemanifesto.org/iso/cs

[2] EILAM, Eldad. *Reversing: Secrets of Reverse Engineering*. Indianapolis: Wiley Publishing, 2005. ISBN 07-645-7481-7. Dostupné z: http://www.ece.ualberta.ca/~marcin/aikonsoft/reverse.pdf

[3] *NHibernate. Relational Persistence for Idiomatic .NET* [online]. ©2015, [cit. 2015-03-28]. Dostupné z: http://nhibernate.info/doc/nh/en

[4] LIDIN, Serge. *.NET IL Assembler*. Berkeley: Apress, 2014. ISBN 15-905-9646-3.

[5] ECMA-334. *C# Language Specification*. Geneva: Ecma International, 2006. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-334.pdf

[6] ECMA-335. *Common Language Infrastructure (CLI)*. Geneva: Ecma International, 2012. Dostupné z: http://www.ecma-international.org/publi\cations/files/ECMA-ST/ECMA-335.pdf

[7] FRIJTERS, Jeroen. *IKVM.NET* [online]. ©2002 - 2011, 2014-06-29 [cit. 2015-03-28]. Dostupné z: http://www.ikvm.net

[8] LINDHOLM, Tim, YELLIN, Frank, BRACHA, Gilad a BUCKLEY, Alex. *The Java Virtual Machine Specification: Java SE 8 Edition*. Redwood City: Addison-Wesley Professional, 2014. ISBN 978-013-3905-908.

[9] HALLEUX, Jonathan. *QuickGraph, Graph Data Structures And Algorithms for .NET* [online]. ©2007 - 2011, 2011-11-20 [cit. 2015-03-28]. Dostupné z: http://quickgraph.codeplex.com

[10] UML 2.4.1. *OMG Unified Modeling Language™ (OMG UML): Superstructure*. Needham: Object Management Group, 2011. Dostupné z: http://www.omg.org/spec/UML/2.4.1/Superstructure

[11] *Graphviz. Graph Visualization Software* [online]. ©2000 - 2014, 2014-04-13 [cit. 2015-03-28]. Dostupné z: http://www.graphviz.org