VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

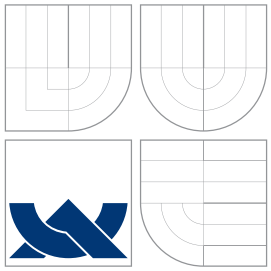
ASPEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

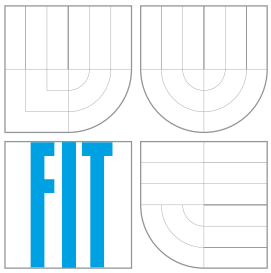
AUTOR PRÁCE
AUTHOR

MARTIN JONÁŠ

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ASPEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

ASPECT ORIENTED PROGRAMMING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN JONÁŠ
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK KOČÍ, Ph.D.
SUPERVISOR

BRNO 2008

Licenční smlouva je uvedena v archivním výtisku uloženém v knihovně FIT VUT v Brně.

Abstrakt

Cílem práce je prozkoumat techniku aspektově orientovaného programování (Aspect Oriented programming - AOP). Práce uvádí základní informace o filozofiia principu práce AOP a stručný přehled nejpoužívanějších nástrojů. Dále obsahuje praktické ukázky použití AOP na jednoduchých příkladech i ve vývoji komplexnější aplikace pomocí nástroje AspectJ.

Klíčová slova

aspekt, koncern, aspektově orientované programování, AspectJ, AspectWerkz, AOP, proplétání

Abstract

This thesis discuss technique of Aspect Oriented Programming (AOP). Thesis contains basic informations about AOP philosophy, principles and short overview of AOP tools. Thesis also contains practic demonstrations of using AOP for simple examples and complex aplication using AspectJ tool.

Keywords

aspect, concern, aspect oriented programming, AspectJ, AspectWerkz, AOP, weaving

Citace

Martin Jonáš: Aspektově orientované programování, bakalářská práce, Brno, FIT VUT v Brně, 2008

Aspektově orientované programování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Kočího, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Jonáš
12. května 2008

Obsah

Kapitola 1

Úvod

Vývoj programování a programovacích technik zůstával v posledních letech poměrně ustáleným prostředím bez výrazných změn. Posledním velkým pokrokem v této oblasti byl nástup objektově orientovaného programování, které přineslo zcela nový pohled na abstrakci řešených problémů. Veškerý následující vývoj se týkal převážně zdokonalování existujících technik.

V poslední době se ale opět začaly objevovat nové techniky. Mezi nejzajímavější a nejperspektivnější z nich v současnosti patří aspektově orientované programování (Aspect Oriented Programming - zkráceně AOP). To, podobně jako objektově orientované programování, přináší nový pohled na abstrakci problémů. Rozděluje program nejen z pohledu dekompozice na jednotlivé strukturální části, ale i z pohledu jednotlivých oblastí funkčnosti.

V této práci představím základní principy aspektově orientovaného programování, jeho filozofii, výhody a nevýhody. Mým cílem je poskytnout stručný a ucelený přehled o této technice a poskytnout čtenářům možnost se na základě předložených informací rozhodnout o vhodnosti použití aspektově orientovaného programování pro jejich programy.

Kapitola 2 popisuje aspektově orientované programování. Nejprve uvedu důvody, které vedly ke vzniku AOP, jeho historii a popíšu princip jeho činnosti. Kapitola 3 přináší stručný přehled nejpoužívanějších nástrojů pro AOP a jejich základní porovnání. V kapitole 4 uvedu několik typických příkladů použití AOP. Kapitola 5 popisuje program, který jsem vytvořil v rámci své bakalářské práce jako ukázku programu vytvořeného s využitím aspektově orientovaného programování. Kapitola 6 shrnuje výhody a nevýhody, které použití AOP přináší a uvádí celkové zhodnocení této techniky.

V příloze A se nachází překladová tabulka, kterou jsem použil pro překlad některých anglických termínů do češtiny. Příloha B popisuje postup instalace nástroje AspectJ. Příloha C obsahuje ukázku zdrojového kódu programu z kapitoly 5. K práci je přiloženo CD, jehož obsah je popsán v příloze D.

Kapitola 2

Aspektově orientované programování

Při psaní této kapitoly jsem vycházel především z informací uvedených v [1], [13], [15], [16], [17], [22] a [23].

2.1 Důvody vzniku AOP

2.1.1 Problémy současných technik

Návrh aplikací

Všechny současné metody vývoje programů, jako je procedurální programování a objektově orientované programování, se při návrhu snaží o vzájemnou separaci jednotlivých částí funkčnosti programu do logicky oddělených celků. Tyto oddělené oblasti funkčnosti se obvykle v literatuře označují jako *koncerny* (viz například [23]).

Při návrhu aplikací musí návrhář uvažovat různé koncerny, ze kterých funkčnost programu sestává. Obvykle existuje jeden nebo několik základních koncernů, které tvoří vlastní funkčnost aplikace. Tyto základní koncerny je ale ve většině případů nutné doplnit dalšími podpůrnými koncerny, které slouží k zajištění podmínek pro práci základních koncernů nebo jejich funkčnost nějakým způsobem rozšiřují.

Využití současných technik dokáže oddělit a zapouzdřit většinu koncernů, které se v programech vyskytují. Některé koncerny ale s pomocí současných technik od ostatních oddělit nelze. Tyto koncerny se obvykle nazývají *průřezové (cross-cutting) koncerny*, protože svojí funkcí protínají oblast funkčnosti ostatních koncernů a různých částí programu. Návrhář je proto nucen do návrhu zahrnout i součásti nutné pro správné provádění průřezových koncernů v rámci všech koncernů, do kterých zasahují.

Příkladem průřezového koncernu může být například zaznamenávání činnosti programu (logování). Kód, který provádí zaznamenávání, je rozptýlen do všech částí programu. Při změně formátu zápisů je pak nutná úprava ve všech částech programu. Dalším příkladem může být transakční zpracování v databázích. Kód starající se o správné zapouzdření operací do transakcí musí být součástí všech částí programu, které pracují s databází.

Zmíním se ještě o problému tzv. *návrhářského dilema*. Návrhář je většinou nucen volit kompromis mezi jednoduchostí a rozsahem programu a její rozšiřitelností. Často to vyžaduje zahrnout do programu některé v současnosti nepotřebné součásti, které jsou ale nezbytné pro možnost dalšího rozšiřování. Návrhář proto musí zvolit správnou míru vyváženosti

těchto požadavků. Pokud jsou součásti pro rozšiřování zanedbány, může být v budoucnosti velmi obtížné program modifikovat. Na druhou stranu pokud je těchto součástí příliš mnoho, program se stává těžkopádným a nepřehledným.

Implementace aplikací

Při implementaci průřezových koncernů pomocí současných technik narážíme na několik problémů. Programátoři jednotlivých koncernů jsou nuceni implementovat nejen funkčnost koncernu, na kterém pracují, ale i části všech průřezových koncernů, které do jeho funkčnosti zasahují. To programátora samozřejmě nežádoucím způsobem rozptyluje. Programátor se při práci nesoustředí pouze na jeden problém, ale zpracovává jich současně hned několik. To výrazně zvyšuje riziko vzniku chyb.

Dalším problémem je tzv. *zašmodrchání (tangling)* kódu. Vzniká v případě, že na jednom místě programu je vzájemně propleten kód plnící funkčnost několika různých koncernů. V takovém kódu se pak špatně orientuje a může být velmi obtížné identifikovat a oddělit části patřící k jednotlivým koncernům.

Posledním problémem při implementaci je tzv. *rozptýlení (scattering)* kódu. Kód plnící funkci průřezového koncernu je rozptýlen v různých částech kódu programu a často se mnohokrát opakuje. Tím se zvyšuje rozsah kódu a vznikají problémy při jeho úpravách, protože stejné úpravy je většinou nutné provést vždy na několika místech současně. Při opomenutí úpravy na některém místě, kam je kód koncernu rozptýlen, může dojít ke vzniku nepříjemných chyb.

2.1.2 Možná řešení problémů

O řešení výše uvedených problémů se pokouší celá řada technik. Uvedu zde některé z nich.

Návrhové vzory

Návrhové vzory jako *Template method* nebo *Proxy* částečně řeší některé výše uvedené problémy. Jejich použití ale vyžaduje velmi schopného a předvídavého návrháře, protože musí být součástí návrhu programu už od začátku.

Více o uvedených návrhových vzorech viz [12] a [5].

Frameworky

Frameworky jsou knihovny zaměřené na podporu a zjednodušení vývoje programů pro určitou aplikační oblast. Obsahují většinu potřebných podpůrných koncernů, které programátor může potřebovat při vytváření programu právě pro danou oblast. Programátor poté při vývoji pouze doplňuje vlastní aplikační logiku a o veškerou podpůrnou funkčnost se stará framework. Problémem frameworků je jejich zaměření pouze na určitou aplikační oblast. Programátoři jsou také nuceni se naučit využívat možnosti frameworku při vytváření svých programů.

Aspektově orientované programování

Další možností řešení výše uvedených problémů je využití aspektově orientovaného programování, které je předmětem této bakalářské práce. Tato technika je považována za další krok v evoluci programovacích technik.

Základním prvkem aspektově orientovaného programování jsou tzv. *aspekty*. Aspekt je část kódu programu, která není součástí klasické struktury. V určitém slova smyslu aspekty klasickou strukturu programu protínají. Tím doplňují klasické programování o možnost uvažovat o struktuře programu z dalšího pohledu. Aspekty tvoří možnost, jak dynamicky zasáhnout do statické struktury klasického programu a vhodným způsobem upravit nebo rozšířit jeho chování.

Aspektově orientované programování není nástupcem procedurálního nebo objektově orientovaného programování. Jde o techniku, která současné techniky vývoje nenahrazuje, ale doplňuje.

2.2 Historie AOP

Myšlenka aspektově orientovaného programování se poprvé objevila v laboratořích XEROX PARC. Jejími autory jsou členové skupiny, kterou vedl Gregor Kiczales. Výsledkem jejich práce byla první verze nástroje AspectJ, který zůstává nejpoužívanějším nástrojem pro aspektově orientované programování dodnes.

Při prvním pokusném nasazení aspektově orientovaného programování při vývoji software se dosáhlo velmi zajímavých výsledků. V [16] je uveden příklad vývoje programu, při kterém došlo ke zkrácení zdrojového kódu z 35 213 řádek bez použití AOP na 1039 řádek s využitím AOP. Do vývoje aspektově orientovaných nástrojů se poté pustila i firma IBM a brzy následovaly další. Tím začal vývoj v této oblasti. Stávající nástroje se postupně zdokonalují a rozšiřují se jejich možnosti. Objevuje se také celá řada nových nástrojů.

Aspektově orientované programování vzniklo pro jazyk Java a dodnes Java zůstává jazykem, ve kterém je aspektově orientované programování nejčastěji využíváno. Vznikla ale celá řada nástrojů pro další jazyky jako jsou C, C++, PHP, Python a další. Více o nástrojích pro AOP je uvedeno v kapitole 3.

2.3 Základní pojmy

Nejprve vysvětlím několik základních pojmů, na kterých je aspektově orientované programování založeno.

Model přípojných bodů (Join point model)

Základním kamenem aspektově orientovaného programování jsou tzv. *modely přípojných bodů (Join point model)*. Ty určují, jakým způsobem se aspekty připojují k základnímu programu. Různé nástroje používají různé modely přípojných bodů. Každý model přípojných bodů definuje několik základních mechanismů.

Přípojný bod (Join point)

Přípojný bod je místo v programu, na kterém lze jeho chování nějakým způsobem upravit. Může jít například o volání metody, vznik výjimky, instancování objektu a podobně.

Pointcut

Pointcut je způsob, jakým lze z množiny všech přípojných bodů v programu vybrat tu část, ve které chceme provést danou změnu chování programu. Různé nástroje se nejčastěji liší

právě ve způsobu definice pointcutů. Tyto mechanizmy se označují jako *jazyky pro definici pointcutů (pointcut specification languages)*.

Advice

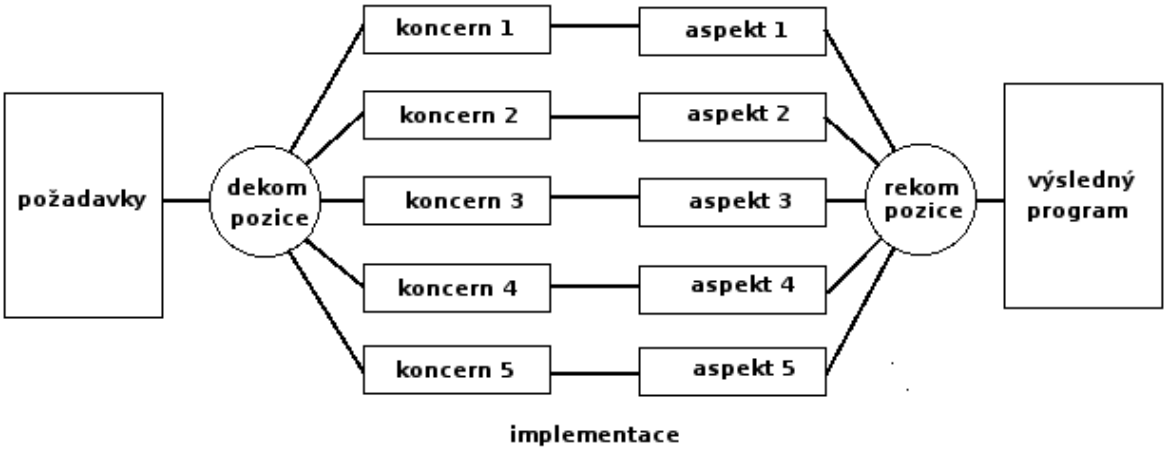
Advice je prostředek pro ovlivnění chování programu ve zvolených přípojných bodech. Jde v případě nejčastějšího modelu přípojných bodů o vkládání výkonného kódu, který se provede v místě přípojného bodu. Většinou jde o speciální metodu, k jejímuž volání v daném místě kódu dojde.

Aspekt

Aspekt je modulem programu, který v sobě zapouzdřuje advice a jejich pointcuty. Jde o implementaci konkrétního koncernu.

2.4 Vývoj programů s využitím AOP

Samotné vytváření programů s využitím AOP se skládá ze tří základních kroků. Prvním krokem je návrh aplikace, při kterém dojde k oddělení jednotlivých koncernů. Následuje oddělená implementace těchto koncernů (aspektů) a posledním krokem je opětovné spojení již hotových aspektů do jednoho celku. Spojování jednotlivých koncernů se nazývá *proplétání(weaving)*.



Obrázek 2.1: Vývoj programů s využitím AOP

2.4.1 Návrh aplikací

Tato metodika návrhu se obvykle označuje jako *AOSD ("Aspect Oriented Software Development")*. Základní činností je tzv. *aspektové dekompozice*, jejímž cílem je jednoznačně identifikovat jednotlivé koncerny. Koncerny by na sobě měly být co nejméně závislé. V ideálním případě by o sobě vůbec neměly vědět. Dále musí návrhář zvolit základní koncerny, které budou tvořit jádro činnosti programu, které bude poté rozšiřováno pomocí aspektů.

Protože jsou jednotlivé koncerny na sobě nezávislé, je také velmi zjednodušeno přidávání nových koncernů v budoucnosti. Při přidání nového koncernu není většinou vůbec nutné zasahovat do kódu původních koncernů.

Častým problémem je otázka, zda určitou funkčnost implementovat jako aspekt, nebo ji zahrnout přímo do základního kódu. Nejzákladnějším doporučením pro toto rozhodnutí je orientovat se podle granularity daného koncernu. Pokud má koncern stejnou granularitu jako základní kód (tj. jeho činnosti si 1:1 odpovídají s činnostmi základního kódu), je možné ho zahrnout přímo do základního kódu. Pokud má ale koncern granularitu menší a bylo by nutné stejný kód opakovat, je jednoznačnou volbou použití aspektu. Toto doporučení vychází z [18].

Jako příklad můžeme uvést transakční zpracování v programech pracujících s databázemi. V základním kódu programu se vyskytuje velké množství funkcí s operacemi, které mají být provedeny v rámci jedné transakce. Samotný kód pro započetí, ukončení a zrušení transakce bude ale s největší pravděpodobností pro všechny tyto funkce shodný. Tento kód má tudíž menší granularitu než samotný základní kód a měl by proto být součástí aspektu. Pokud by ale základní program obsahoval několik funkcí, jež by bylo každou potřeba rozšířit zcela specifickým způsobem, mohl by být tento rozšiřující kód zahrnut i přímo do základního kódu.

2.4.2 Implementace aplikací

Aspektově orientované programování přináší značné zjednodušení pro programátory implementující jednotlivé koncerny. Odstraňuje problémy, které vznikají při použití současných technik, které byly zmíněny výše v kapitole 2.1.1.

Programátor není nucen se při programování koncernu, na kterém pracuje, zabývat průřezovými koncerny, které do něj zasahují. Může se proto soustředit jen na svoji práci.

Jednotlivé koncerny jsou vzájemně izolované, a proto jejich implementace může být svěřena vývojářům specializujícím se na danou oblast. A to včetně průřezových koncernů.

2.4.3 Proplétání

Poté, co jsou všechny koncerny implemetovány, je nutné jednotlivé kódy spojit do jednotného celku (*aspektová rekompozice*). Proces spojování aspektů je základem aspektově orientovaného programování a nazývá se *proplétání (weaving)*. Výsledkem proplétání je hotový program, který obsahuje všechny koncerny implementované jednotlivými aspekty.

Proplétání je proces, při kterém dojde ke spojení základního kódu programu a kódu aspektů do společného výsledného kódu.

Systém práce proplétání si můžeme jednoduše představit zhruba takto:

Proplétací nástroj (weaver) vezme původní kód programu a kódy aspektů a rozdělí je na jednotlivé součásti. Poté tyto součásti kódu zkombinuje podle předpisů uvedených v kódu aspektů. Ze zkombinovaných částí kódu je poté vytvořen nový kód, který v sobě zahrnuje spojenou funkčnost všech aspektů.

Proplétání se rozděluje na několik druhů v závislosti na tom, kdy v průběhu vývoje programu k němu dochází. Některé způsoby proplétání jsou použitelné pouze pro jazyk Java.

Při překladu programu

Proplétání při překladu (compile-time weaving) je nejčastějším způsobem proplétání. Jde v zásadě o určitý druh preprocesoru kódu. Tento způsob je použitelný pro všechny jazyky.

Proplétací nástroj pracuje přímo se zdrojovým kódem programu, do kterého vkládá kód aspektů. Pro jeho funkčnost je nutné, aby měl přístup ke zdrojovým kódům programu i aspektů. Tento způsob proplétání proto nejde použít na knihovny třetích stran, které jsou dodávány bez zdrojových kódů.

Výsledný přeložený kód nepotřebuje žádné speciální prostředí pro spuštění. V případě jazyka Java je možné využít standardní JVM i classloader.

Při použití tohoto způsobu proplétání hrozí pouze jediný problém, a to, že výsledný kód nebude odpovídat kódu, který by bylo možné dostat jako výsledek při použití standardního překladače daného jazyka. Kód proto nemusí být čitelný pro další nástroje jako jsou například debuggery, které očekávají určitý standardizovaný formát přeloženého kódu.

Tento typ proplétání může být buď přímo součástí speciálního překladače (jak je tomu u nástroje AspectJ) nebo může být použit jako samostatný nástroj (jak u nástroje XWeaver), který provede předzpracování kódu, který je poté přeložen standardním překladačem.

Po překladu programu

Proplétání po překladu (byte-code weaving, post-compile weaving) je dalším způsobem proplétání. Ke spojení kódu programu s kódy aspektů dochází až po překladu. Přeložené kódy jednotlivých částí se v proplétacím nástroji vzájemně spojí a výsledkem je nový program, který již obsahuje spojené kódy všech částí.

Pracuje se s již přeloženým kódem programu. Pro proplétání proto není nutné mít přístup ke zdrojovým kódům a je proto použitelné i na knihovny třetích stran.

Výsledný kód nepotřebuje žádné speciální prostředí pro spuštění. V případě jazyka Java je možné využít standardní JVM i classloader. Jediným omezením může být určitá minimální požadovaná verze JVM.

Pokud používáme tento způsob proplétání na přeložený kód, ke kterému nemáme k dispozici zdrojové kódy a neznáme tedy jeho vnitřní strukturu, hrozí nebezpečí, že po aplikaci aspektů dojde ke vzniku nezamýšlených vedlejších efektů.

Při nasazení programu

Proplétání při nasazení (deploy-time weaving) je způsobem proplétání, při kterém k aplikaci aspektů dojde až v okamžiku závádění programu.

Pokud chceme využít tento způsob proplétání, potřebujeme speciální nástroje pro zavedení programu. V případě jazyka Java musíme využít speciální classloader, který při načítání tříd každou třídu doplní o kód aspektů. V principu pracuje stejně jako proplétání po překladu, ale k proplétání dochází až v okamžiku, kdy je aplikace zaváděna.

Za běhu programu

Proplétání za běhu (run time weaving) je způsob proplétaní, kdy k aplikaci aspektů dochází až v průběhu běhu programu. Tento způsob proplétání se také nazývá „online“ jako protiklad k ostatním „offline“ druhům proplétání.

V principu jde o změnu způsobu, kterým program za běhu vytváří nové objekty a přistupuje k nim. Pro jeho práci je nutné využít speciální prostředí, ve kterém bude program pracovat. V případě jazyka Java jde o speciální implementaci JVM.

V jazyce Java existují dva přístupy k implementaci tohoto druhu proplétání. Prvním z nich je *„bytecode enhancement“* – kód jednotlivých tříd je doplněn o kód aspektů. Druhým je tzv. *„dynamic proxy“* – při využití tohoto způsobu je při vytváření instance každého objektu současně vytvořen proxy objekt, který se používá při každém přístupu k původnímu objektu. Tento proxy objekt nejprve zajistí provedení funkčnosti vyžadované aspekty a teprve poté spustí výkonný kód původního objektu.

2.5 Nástroje

V zásadě existují dva přístupy k implementaci nástrojů pro aspektově orientované programování.

Prvním z nich je upravení existujících nástrojů a jejich rozšíření o podporu aspektově orientovaného programování. Tento přístup používá například AspectJ, který rozšiřuje překladač jazyka Java o definici několika nových klíčových slov. Více o nástroji AspectJ naleznete v kapitole 3.1.

Druhým přístupem je využití existujících nástrojů nezměněných, které jsou pouze doplněny o nástroje pro AOP. Nejznámějším zástupcem tohoto přístupu je nástroj AspectWerkz. Více o nástroji AspectWerkz naleznete v kapitole **??**.

Nejčastějšími rozdíly mezi různými nástroji jsou různé modely přípojných bodů a způsob, jaký používají pro definici pointcutů. Využívá se nejčastěji regulárních výrazů, zástupných znaků (wildcard), XML a konfiguračních souborů.

Nejlepší podporu pro aspektově orientované programování má jazyk Java, pro který bylo AOP původně vyvinuto. Existuje ale množství nástrojů pro další jazyky jako je C, C++, JavaScript, Perl, PHP, Python, Ruby, a mnoho dalších.

Více viz následující kapitola 3.

2.6 Další funkce

Přidávání spustitelného kódu do přípojných bodů je sice nejčastějším typem modelu přípojných bodů, ale rozhodně ne jediným.

Mezi-typové deklarace

Mezi další typy JPM patří *mezi-typové deklarace (inter-type declarations)*. Tento JPM umožňuje přidat do existujících tříd doplňující deklarace. V tomto JPM jsou přípojnými body jednotlivé třídy v základním kódu programu a advice jsou doplňující deklarace členských proměnných a metod. Třídy tak lze pomocí aspektů rozšířit. Umožňuje také upravovat hierarchii tříd, změnit rodičovskou třídu nebo přidat implementaci dalších rozhraní (interface).

Kapitola 3

Nástroje pro AOP

Nyní představím několik nejčastějších nástrojů pro aspektově orientované programování a shrnu jejich hlavní rozdíly. Při psaní této kapitoly jsem vycházel z [11], [13] a [23]. Podrobně zpracované porovnání nástrojů pro AOP lze nalézt v [14].

3.1 AspectJ

AspectJ je nejstarším a nejpopulárnějším nástrojem pro aspektově orientované programování. Byl vyvinut v laboratořích XEROX PARC v roce 1997. Jeho vývoj v současnosti probíhá pod záštitou Eclipse Foundation.

Má nejširší možnosti definice pointcutů ze všech existujících nástrojů. Podporuje všechny typy proplétání kromě proplétání v době běhu programu. Umožňuje nejen klasický model přípojných bodů pro vkládání výkonného kódu, ale i mezi-typové deklarace.

AspectJ patří k nástrojům, které upravují existující nástroje. Upravuje kompilátor jazyka Java a rozšiřuje ho o nová klíčová slova. To s sebou nese pro programátory nutnost naučit se novou syntaxi. Aspekty jsou reprezentovány speciálním typem tříd, které obsahují advice a definice pointcutů. Pro definování pointcutů se používají speciální klíčová slova a regulární výrazy. Obecně AspectJ vyžaduje mnohem kratší zdrojový kód než ostatní nástroje.

Pro překlad programů se používá speciální překladač – `ajc`.

Tento nástroj jsem také použil při vytváření ukázkových příkladů v kapitole 4 a demonstračního programu popsaného v kapitole 5.

Jako podpůrný nástroj pro vývojáře používající AspectJ existuje plugin pro vývojové prostředí Eclipse - AJDT(„AspectJ Development Tools“).

AspectJ později začalo spolupracovat s nástrojem AspectWerkz na vývoji společného nástroje – viz 3.2. Bližší informace o AspectJ lze nalézt například v [2].

3.2 AspectWerkz

Tento nástroj je v současnosti největším konkurentem AspectJ. Nemá všechny jeho široké možnosti, ale pro naprostou většinu situací plně dostačuje. Jeho vývoj započali dva zaměstnanci firmy BEA – Jonas Boner a Alex Vasseur.

AspectWerkz je představitelem nástrojů, které se snaží pouze doplňovat existující nástroje. Aspekty jsou obyčejné třídy. Pro definici pointcutů a začlenění aspektu do kódu programu

se používají buď anotace nebo XML konfigurační soubory. AspectWerkz také nepodporuje mezi-typové deklarace a vyžaduje delší zdrojový kód než AspectJ.

Využívá dva způsoby proplétání:

- První z nich je „online“. Využívá se v něm speciální JVM s upraveným classloaderem. Programy jsou spouštěny speciálním příkazem.
- Druhý z nich je „offline“. Aspekty se aplikují přímo do přeloženého kódu a pro spuštění pak již není potřeba speciální prostředí. Výsledný kód vyžaduje JVM minimálně verze 1.3.

AspectWerkz a AspectJ se v roce 2005 rozhodly spolupracovat na vývoji jednotného nástroje pro AOP – více viz [3]. Nástroj AspectJ 5 již obsahuje funkčnost převzatou z nástroje AspectWerkz.

AspectWerkz je free software šířený pod licencí vycházející z GNU LGPL licence. Bližší informace o tomto nástroji lze nalézt v [4].

3.3 Spring Framework

Spring Framework je v poslední době velmi oblíbeným nástrojem pro vývoj serverových aplikací v Javě. Tento framework hojně používá aspektově orientované programování interně ve vlastním kódu. Současně přináší jeho podporu i pro vyvíjené programy. Používá vlastní implementaci AOP, která využívá XML konfiguraci a proplétání za běhu s využitím dynamic proxy. XML konfigurace obsahuje předpisy určující kdy a kde se má který aspekt použít. Aspekty jsou implementovány jako obyčejné Java třídy.

Tento framework také současně poskytuje aplikacím podporu pro využití nástroje AspectJ. Vývojáři proto mohou využít ten nástroj, který je pro ně jednodušší, nebo oba nástroje vhodným způsobem kombinovat. Bližší informace lze nalézt v oficiální dokumentaci [9] nebo česky v [19].

3.4 JBoss AOP

Aplikační server JBoss je nástrojem pro provozování serverových aplikací v Javě vyvíjený firmou RedHat, který obsahuje podporu pro aspektově orientované programování. Prostředí pro AOP je přímo součástí aplikačního serveru, který obsahuje speciální classloader. Využívá proplétání za běhu.

Tento nástroje je šířen pod licencí GNU LGPL. Více informací o tomto nástroji naleznete v [7].

3.5 XWeaver

Tento nástroj poskytuje podporu pro aspektově orientované programování pro jazyky C/C++ a Java. Jeho vývoj začal roku 2003 v laboratořích Automatic Control Laboratory na ETH-Zurich. Tento nástroj pracuje jako preprocesor zdrojových kódů a využívá XML konfiguračních souborů. Výsledkem jeho práce jsou zdrojové kódy, které lze poté přeložit standardním překladačem. Hlavní motivací pro vývoj tohoto nástroje bylo umožnit vývojářům kontrolovat kvalitu kódů po proplétání.

Více informací o tomto nástroji naleznete v [10].

3.6 phpAspect

Tento nástroj vznikl v rámci projektu *Google Summer of Code 2006*. Jde o podporu aspektově orientovaného programování pro populární jazyk pro vývoj webových aplikaci PHP (Php Hypertext Preprocessor). Samotný nástroj je napsán v PHP a využívá syntaxi podobnou AspectJ.

Více informací o tomto nástroji naleznete v [8].

3.7 AspectC++

Jde o obdobu AspectJ pro jazyk C++. Podporuje ale pouze proplétání při kompilaci. Bližší informace o tomto nástroji lze nalézt v [6].

3.8 Další nástroje

Existuje velké množství dalších nástrojů pro aspektově orientované programování. Jejich přehled můžete nalézt například v [23].

Kapitola 4

Příklady použití AOP

4.1 Použité nástroje

Pro demonstrační příklady budu používat syntaxi používanou nástrojem AspectJ, která je většinou snadno pochopitelná. Popíšu zde některé základní konstrukce. Podrobnější informace o syntaxi používané AspectJ lze nalézt například v [2].

Aspekty

Aspekty jsou tvořeny jako speciální druh tříd. Místo klíčového slova `class` se pouze použije klíčové slovo `aspect`.

```
aspect aspectName {
  ...
}
```

Pointcuty

Pointcuty se uvnitř aspektů definuji následujícím stylem:

```
pointcut pointcutName() : PODMINKY;
```

Jako PODMINKY lze uvést celou řadu různých výrazů:

- `within(className)` určuje, že daný pointcut bude platný uvnitř kódu třídy `className`.
- `execution(MODIFIKATOR NAVRATOVY_TYP NAZEV(PARAMETRY) )`, určuje volání konkrétní metody v závislosti na udaných parametrech.
  - `MODIFIKATOR` určuje, zda jde o `public` nebo `private` metodu.
  - `NAVRATOVY_TYP` vybírá metody vracející konkrétní typ.
  - `NAZEV` určuje název metody.
  - `PARAMETRY` určuje počet a typy parametrů metody.

  Všechny parametry lze reprezentovat pomocí zástupných znaků. Například `*` znamená libovolnou hodnotu. Pro volbu libovolného počtu parametrů metody lze použít výraz `NAZEV(...)`.

Advice

Advice se rozlišují na tři druhy. Advice provedené před přípojným bodem (*before*), po přípojném bodu(*after*) a advice obalující přípojný bod (*around*). Advice obalující přípojný bod mohou spustit daný přípojný bod voláním funkce `proceed()`, k volání přípojného bodu ale vůbec nemusí dojít.

Styl zápisu je následující:

```
before() : pointcutName() {
  ...
}
```

4.2 Logování

Jedním z nejtypičtějších příkladů použití aspektově orientovaného programování, který je uváděn ve většině publikací, je doplnění programu o zaznamenávání informací o prováděných činnostech (logování).

Mějme třídu `BusinessClass`:

```
class BusinessClass {

  public BusinessClass() {
    //inicializace
  }

  public void doSomething() {
    //výkonný kód
  }

}
```

Potřebujeme, aby program zaznamenával na standardní výstup každé volání metody `doSomething`.

Nejprve vytvoříme nový aspekt `Loging`, který bude obsahovat pointcut `doSomethingExecution` jednoznačně identifikující spuštění metody `doSomething` ve třídě `BusinessClass`:

```
aspect Loging {

  pointcut doSomethingExecution() :
    within(BusinessClass) &&
    execution (public void doSomething());

}
```

Nyní vytvoříme novou advice, která se provede před všemi přípojnými body definovanými pointcutem `doSomethingExecution` a zaznamená tuto událost:

```
before() : doSomethingExecution() {
  System.out.println("Do Sometning execution");
}
```

4.3 Cache

Dalším častým a vhodným použitím aspektově orientovaného programování je přidání vyrovnávací paměti (cache) k programu.

Mějme třídu implementující rozhraní pro přístup k datům v databázi nebo na vzdáleném serveru `DataClass`:

```
class DataClass {

  DataStore data;

  public DataClass() {
    //inicializace přístupu k datům
    data = initializeDataStore();
  }

  public String getData(String key) {
    //načtení dat
    return data.getData(key);
  }

  public void setData(String key, String value) {
    //zápis dat
    data.setData(key, value);
  }

}
```

Předpokládáme, že operace zápisu a čtení dat jsou časově náročné. Pokud program pracuje často se stejnými daty, bývá výhodné použít vyrovnávací paměť, která k nim poskytne rychlejší přístup. V případě klasického přístupu by se implementace vyrovnávací paměti musela provést zásahem do kódu třídy `DataClass`. Aspektově orientované programování ale umožňuje vytvořit mnohem elegantnější řešení.

Vytvoříme aspekt `DataCache` obsahující pointcut pro spuštění metody `getData` a pointcut pro spuštění metody `setData`:

```
aspect DataCache {

  pointcut getDataExecution() :
    within(DataClass) &&
    execution (public String getData(...));

  pointcut setDataExecution() :
    within(DataClass) &&
    execution (public void setData(...));

}
```

Nyní vytvoříme třídu představující paměť cache:

```
class DataCacheClass {

  Map<String, String> data;

  public DataCacheClass() {
    data = new HashMap<String, String>();
  }

  public String getData(String key) {
    return (String)data.get(key);
  }

  public void setData(String key, String value) {
    data.put(key, value);
  }

  public boolean containsData(String key) {
    return data.containsKey(key);
  }

}
```

Do aspektu `DataCache` doplníme advice, které přidají použití vyrovnávací paměti do třídy `DataClass`:

```
aspect DataCache {

  //vytvoříme objekt cache
  DataCacheClass cache = new DataCacheClass();

  pointcut getDataExecution() :
    within(DataClass) &&
    execution (public String getData(...));

  pointcut setDataExecution() :
    within(DataClass) &&
    execution (public void setData(...));

  String around(): getDataExecution() {

    //zjistíme s jakými parametry byla metoda volána
    Object[] args = thisJoinPoint.getArgs();
    String key = (String) args[0];

    //pokud se data v cache nacházejí načteme je
    if(!cache.containsData(key) {
```

```
      //spustíme metodu, kolem které je advice prováděna (getData)
      String value = proceed();
      cache.setData(key, value);
    }

    //vrátíme data z cache
    return cache.getData(key);
  }

  after(): setDataExecution() {
    //zjistíme s jakými parametry byla metoda volána
    Object[] args = thisJoinPoint.getArgs();
    String key = (String) args[0];
    String value = (String) args[1];

    //uložíme data i do cache
    cache.setData(key, value);
  }

}
```

Příklad na použití AOP pro cache lze nalézt v [20].

4.4 Profilování

Dalším možným použitím aspektově orientovaného programování je profilování kódu. Aspekt pro profilování může velice snadno profilovat celý kód programu. Pro nasazení aplikace do provozu pak stačí aspekt pro profilování vynechat z překladu aniž by byl nutný jakýkoli zásah do kódu.

Mějme třídu `CriticalClass`:

```
aspect CriticalClass {

  public void taskA() {
    //výkonný kód
  }

  public void taskB(int par) {
    //výkonný kód
  }

  public void taskC(String par) {
    //výkonný kód
  }


}
```

Uvedu příklad jednoduchého aspektu, který se naváže na spuštění všech metod ve třídě `CriticalClass` a vypíše dobu jejich běhu.

```
aspect CriticalClassProfiler {

  pointcut taskExecution() :
    within(CriticalClass) &&
    execution (* * *(...)); //spuštění libovolné metody

  void around(): taskExecution() {
    long timeStart = System.nanoTime();
    thisJoinPoint.proceed();
    long timeEnd = System.nanoTime();
    long time = (timeStart - timeEnd) / 1000000;
    System.out.println("Method " + thisJoinPointStaticPart.getSignature() +
      " execution time " + time + "ms";
  }


}
```

Velmi pěkný příklad na použití aspektově orientovaného programování pro profilování lze nalézt například v [21].

Kapitola 5

Program s využitím AOP

Součástí mojí bakalářské práce je i komplexní program vyvinutý s využitím aspektově orientovaného programování. Jedná se o jednoduchý demonstrační program představující sklad souborů na serveru, z nějž si mohou klienti soubory stahovat. Program byl vyvinut s pomocí nástroje AspectJ (viz kapitola 3.1). Postup instalace tohoto nástroje je popsán v příloze B.

5.1 Analýza

Program byl nejprve vyvinut bez využití aspektově orientovaného programování a následně byl rozšířen o některé koncerny pomocí aspektů. Seznam aspektů lze nalézt v kapitole 5.4.

Program sestává ze dvou oddělených částí:

- *Server* obsahuje sklad souborů a poskytuje rozhraní pro přístup k nim.
- *Klient* obsahuje rozhraní pro uživatele, s jehož pomocí může k souborům na serveru přistupovat. Komunikace probíhá pomocí síťového rozhraní socketů.

5.2 Návrh

Prvním úkolem bylo rozhodnout, jaká funkčnost programu bude součástí základního kódu a kterou oddělíme do aspektů. Základní program jsem navrhl tak, aby byl schopen pracovat i bez použití aspektů. Aspekty se používají pouze k rozšíření jeho funkčnosti.

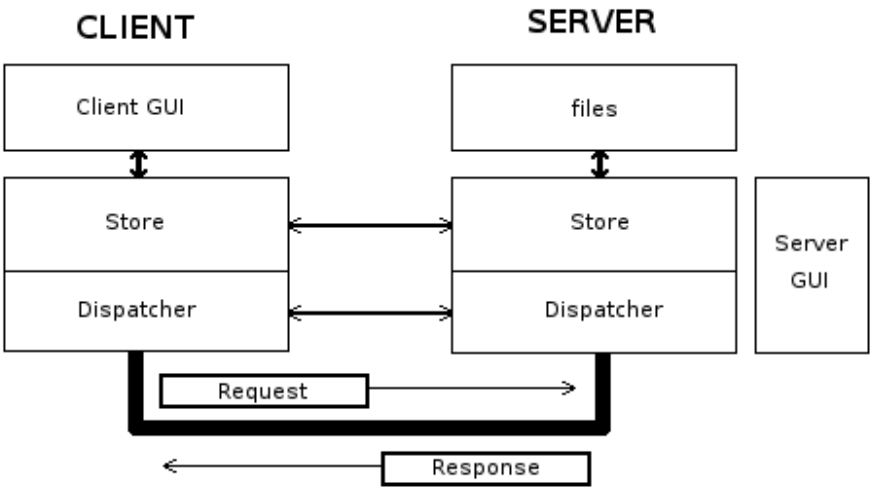
Obě součásti programu (klient i server) se skládají z několika částí – viz obrázek 5.1.

`Client GUI`

Tato součást obsahuje implementaci grafického uživatelského rozhraní (GUI) klientské části programu. Stará se o vytvoření ovládacího okna a předává akce uživatele ke zpracování nižším vrstvám programu.

`Store`

Tato součást představuje samotný sklad souborů. Přistupuje k souborům na pevném disku serveru a poskytuje programové rozhraní pro přístup k nim. Umožňuje čtení seznamu souborů, informací o souborech (velikost a čas poslední změny) a jejich načítání. Tato součást



Obrázek 5.1: Architektura programu

se vyskytuje v obou částech programu (klientu i serveru) a tvoří společně společnou vrstvu rozhraní.

`Dispatcher`

Tato součást představuje rozhraní pro komunikaci mezi klientem a serverem přes síťové rozhraní. Komunikace sestává z vyslání požadavku klientem, jeho zpracování serverem a odeslání odpovědi zpět klientovi. Tato součást se vyskytuje v obou částech programu (klientu i serveru) a tvoří společně společnou vrstvu rozhraní.

`Server GUI`

Tato součást obsahuje implementaci grafického uživatelského rozhraní (GUI) serverové části programu. Stará se o vytvoření ovládacího okna, které slouží k ovládání serveru. Propojení s ostatními součástmi serveru je zajištěno pomocí aspektově orientovaného programování.

`Request`

Tato součást zapouzdřuje práci s požadavky odesílanými klientem serveru.

`Response`

Tato součást zapouzdřuje práci s odpovědmi odesílanými serverem klientovi jako reakce na požadavky.

5.3 Implementace

Implementace programu je rozdělena do několika balíčků.

`cz.vubr.fit-stud.xjonas03.ibp`

Tento balíček obsahuje definici obecných rozhraní pro jednotlivé vrstvy programu. Implementace pro klienta a server pak vychází z těchto rozhraní. Tím je zajištěna transparentnost jednotlivých vrstev programu na obou stranách. Tento balíček také obsahuje kód pro třídy `Request` a `Response`, která slouží jako transportní objekty pro přenos požadavků a odpovědí po síti.

`cz.vubr.fit-stud.xjonas03.ibp.client`

Tento balíček obsahuje implementaci klientského uživatelského rozhraní a implementaci jednotlivých vrstev aplikace pro komunikaci se serverem.

`cz.vubr.fit-stud.xjonas03.ibp.server`

Tento balíček obsahuje kódy pro server. Server je v základní podobě implementován jako konzolová aplikace.

`cz.vubr.fit-stud.xjonas03.ibp.server.gui`

Tento balíček rozšiřuje server o grafické uživatelské rozhraní.

5.4 Aspekty

Logování práce serveru

Server zaznamenává svoji činnost na standardní výstup pomocí aspektu definovaného v souboru `server/Loging.aj`.

Tento aspekt je navázán na spuštění některých klíčových metod. Například příchod nového požadavku, dokončení zpracování požadavku a podobně. Každá takováto událost je zaznamenána na standardní výstup.

Logování práce klienta

Klient také zaznamenává svoji činnost na standardní výstup pomocí aspektu definovaného v souboru `client/Loging.aj`.

Tento aspekt je napsán tak, aby zaznamenával na standardní výstup spuštění všech metod v programu. V podstatě se jedná o trasování běhu programu.

Cache souborů na klientu

Aby se omezilo množství dat přenášených po síti, je klient rozšířen o lokální chache souborů pomocí aspektu v souboru `client/Cache.aj`.

Při požadavku na stažení souboru se nejprve porovná velikost a čas poslední změny souboru uloženého v cache a souboru aktuálně se vyskytujícího na serveru. Pokud se na serveru nachází odlišná aktuálnější verze je stažena a uložena do cache.

Zdrojový kód tohoto aspektu naleznete v příloze C.

Grafické uživatelské rozhraní serveru

Server je v balíčku `cz.vubr.fit-stud.xjonas03.ibp.server.gui` rozšířen o grafické uživatelské rozhraní. Aby toto rozhraní mohlo zobrazovat informace o běhu serveru, je v souboru `server/gui/GUILoging.aj` vytvořen aspekt, který provede navázání uživatelského rozhraní na události v běhu serveru.

Kapitola 6

Závěr

Tato kapitola shrnuje hlavní výhody a nevýhody, které přináší využití aspektově orientovaného programování oproti jiným programovacím technikám.

6.1 Výhody

- Hlavní výhodou aspektově orientovaného programování je izolace průřezových koncernů do samostatných modulů (aspektů). Veškerý související kód koncernu je díky tomu na jednom místě. To umožňuje snadněji provádět změny chování programu.
- Pomocí aspektů je možné snadno upravit chování celého programu.
- Odstraňuje rozptýlení a zašmodrchání kódu.
- Aspektově orientované programování také umožňuje rozšířit chování programu o další koncerny dodatečně. Při přidávání koncernů není nutný zásah do původního kódu.
- Aspekty jsou vzájemně nezávislé a proto může být jejich implementace svěřena odborníkům na danou oblast.
- Kód aspektů je izolovaný do samostatných modulů, což umožňuje jeho snadnější znovupoužitelnost v dalších programech.
- Aspektově orientované programování produkuje obecně kratší, srozumitelnější kód a zrychluje vývoj.
- Umožňuje velmi elegantní řešení některých problémů.

6.2 Nevýhody

- První ze dvou hlavních nevýhod AOP je možnost neúmyslných aplikací advice. Pokud při definici pointcutu udělá programátor chybu, může dojít k aplikaci advice i v přípojných bodech, kde k němu dojít nemělo. Tím může být vážně poškozeno chování celého programu. Jediná chyba tak může snadno způsobit nefunkčnost celého systému.
- Druhou hlavní nevýhodou je, že programátor při čtení zdrojového kódu programu neví, co všechno se v daném místě kódu provede, protože chování je dodatečně

rozšiřováno aspekty. Obě tyto nevýhody odstraňují podpůrné nástroje pro vývojáře, které dokáží zobrazit výslednou podobu kódu po propletení.

- Aspektově orientované programování je zatím poměrně nová a časem neprověřená technologie.
- Programátoři se musí naučit aspektově orientované programování používat. To může znamenat pro vývojářské firmy zvýšení nákladů na zaškolení pracovníků.
- Aspektově orientované programování může být obtížné použít pro dodatečné rozšíření špatně navržených systémů, které neobsahují dostatek vhodných pointcutů.

6.3 Zhodnocení

Aspektově orientované programování je velmi silný nástroj pro řešení problém spojených s průřezovými koncerny. Oproti jiným řešením je velmi elegantní a univerzální. Stále jde ale o poměrně mladou technologii, která se neustále vyvíjí. Postupně se ale čím dál tím více rozšiřují její možnosti a zjednodušuje její používání.

Aspektově orientované programování rozhodně není lékem na všechny problémy spojené s vývojem složitějších programů a implementaci průřezových koncernů. V mnoha případech je ale jeho použití velice praktické a přináší nesporné výhody v podobě kratšího a lépe organizovaného zdrojového kódu, který neobsahuje rozptýlený a zašmodrchaný kód různých koncernů.

Nejde ale o univerzální řešení a jeho použití není vhodné ve všech oblastech. Použití aspektově orientovaného programování sebou přináší zvýšené náklady na zaškolení programátorů a vyšší nebezpečí vzniku skrytých chyb.

Aspektově orientované programování je velice mocným nástrojem, který mé jistě slibnou budoucnost.

Literatura

[1] aosd.net. [online].
URL http://aosd.net/

[2] AspectJ. [online].
URL http://www.eclipse.org/aspectj/

[3] AspectJ and AspectWerkz to Join Forces. [online].
URL http://aspectwerkz.codehaus.org/index-merge.html

[4] AspectWerkz - Plain Java AOP. [online].
URL http://aspectwerkz.codehaus.org/

[5] Design pattern (computer science). [online].
URL http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

[6] The home of AspectC++. [online].
URL http://www.aspectc.org/

[7] JBoss AOP: Framework for Organizing Cross Cutting Concerns. [online].
URL http://www.jboss.org/jbossaop/

[8] phpAspect: Aspect-Oriented Programming for PHP. [online].
URL http://phpaspect.org/

[9] The Spring Framework - Reference Documentation: Aspect Oriented Programming with Spring. [online].
URL
http://static.springframework.org/spring/docs/2.5.x/reference/aop.html

[10] The XWeaver Project. [online].
URL http://www.xweaver.org/

[11] AspectJ and AspectWerkz compared... sort of... [online], 2004.
URL http://radio.javaranch.com/val/2004/06/01/1086086742000.html

[12] Dvořák, M.: Návrhové vzory (design paterns). [online].
URL http://objekty.vse.cz/Objekty/Vzory

[13] Kay, R.: Aspektově orientované programování. [online], 2004.
URL
http://www.computerworld.cz/cw.nsf/id/FF5965BA7580523BC1256E9400332DCB

[14] Kersten, M.: AOP@Work: AOP tools comparison. [online], 2005.
URL http://www.ibm.com/developerworks/library/j-aopwork1/

[15] Kinczales, G.; aj.: Aspect-Oriented Programming. [online].
URL
http://www.cs.wm.edu/~coppit/csci780-fall2004/presentations/15-aop.pdf

[16] Kinczales, G.; Lamping, J.; aj.: Aspect-Oriented Programming. In *European Conference on Object-Oriented Programming (ECOOP)*, červen 1997.
URL http://www.cs.ubc.ca/~gregor/papers/kiczales-ECOOP1997-AOP.pdf

[17] Laddad, R.: I want my AOP! [online], 2002.
URL http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html

[18] Lamping, J.: The role of the base in aspect oriented programming. [online].
URL http://trese.cs.utwente.nl/aop-ecoop99/papers/lamping.pdf

[19] Matulík, P.; Páral, T.: Moderní JEE technologie a nástroje: Rámec Spring - AOP. [online].
URL http://morosystems.cz/java/spring/ch04.php

[20] Pichlík, R.: Transparentní cache pomocí Aspektově Orientovaného Programování. [online], [cit. 2008-5-11].
URL http:
//www.sweb.cz/pichlik/archive/2006_11_05_archive.html#116276362881930203

[21] Schenk, C.: Caching with AspectJ. [online].
URL http://www.christianschenk.org/blog/caching-with-aspectj/

[22] Sharwood, S.: A new aspect to programming? [online].
URL http://www.builderau.com.au/strategy/developmentprocess/soa/
A-new-aspect-to-programming-/0,339028278,339183763-2,00.htm

[23] Wikipedie: Aspektově orientované programování. [online], [cit. 2008-5-11].
URL http://cs.wikipedia.org/wiki/Aspektov%C4%9B_orientovan%C3%
A9_programov%C3%A1n%C3%AD

Dodatek A

Překlad anglických termínů

Jako základ překladové tabulky jsem použil překlady použité v [23].

cross-cuting	průřezový
concern	koncern
aspect	aspekt
joinpoint	přípojný bod
weaving	proplétání
weaver	proplétací nástroj
inter-type	mezi-typový
scatering	rozptýlení
tangling	zašmodrchání

Dodatek B

Postup instalace AspectJ

Tato příloha popisuje postup, kterým lze nainstalovat knihovnu AspectJ.

1. Získat instalátor prostředí AspectJ. Instalátor naleznete na přiloženém CD nebo jej lze stáhnout ze stránek Eclipse Foundation (`http://www.eclipse.org/downloads/download.php?file=/tools/aspectj/downloads/aspectj-1.5.4.jar`)
2. Spustit instalátor a nainstalovat knihovnu.
3. Nastavit proměnné prostředí. Proměnná CLASSPATH musí obsahovat cestu k adresáři `/lib` v instalačním adresíři AspectJ. Proměnná PATH musí obsahovat cestu k adresáři `/bin` v instalačním adresíři AspectJ.

Dodatek C

Ukázka zdrojového kódu aspektu

```
/*  ********************************************************************
 **
 **  Projekt :  BP - Aspektově orientované programování
 **
 **  Autor   :  Martin Jonáš      xjonas03@stud.fit.vutbr.cz
 **
 **  Popis   :  Aspekt přidávající cachovaní souborů na lokálním disku
 ** ******************************************************************** */

package cz.vutbr.fit.stud.xjonas03.ibp.client;

import java.io.*;

import cz.vutbr.fit.stud.xjonas03.ibp.*;

/**
 * Ukládá stahované soubory na lokálním disku.
 * při požadavku na stažení zkontroluje aktuální verzi v cache.
 * Pokud je verze aktuální načítá se soubor z cache.
 */
aspect Cache {

  /** Složka do které se ukládají cachované soubory */
  String folder = "cache";

  /**
   * Pokus o stažení souboru
   */
  pointcut getFile() :
    target(Store) &&
    execution (public InputStream getFile(String));
```

```
  /**
   * Kontrola obsahu cache při pokusu o stažení souboru
   */
  InputStream around(): getFile() {

    //odkaz na sklad souborů
    Store store = (Store) thisJoinPoint.getTarget();

    //název požadovaného souboru
    Object[] args = thisJoinPoint.getArgs();
    String filename = (String) args[0];

    //načteme data o aktuální verzi na serveru
    FileInfo remoteFileInfo = store.getFileInfo(filename);

    //načteme data o aktuální verzi v cache
    File localFile = new File(folder, filename);
    FileInfo localFileInfo = new FileInfo();
    localFileInfo.setSize(localFile.length());
    localFileInfo.setLastModified(localFile.lastModified());

    //zjistíme zda je verze v cache aktuální
    if(localFileInfo.getSize() != remoteFileInfo.getSize() &&
       localFileInfo.getLastModified() < remoteFileInfo.getLastModified())
    {
      //pokud není stáhneme novou verzi
      InputStream remoteFile = proceed();

      //uložíme kopii do cache (aktualizujeme)
      try {
        localFile.createNewFile();
        FileOutputStream localFileHandler = new FileOutputStream(localFile);

        int c;
        while ((c = remoteFile.read()) != -1) {
          localFileHandler.write(c);
        }

        localFileHandler.close();
      } catch (IOException ex) {}
    }

    //vrátíme soubor z cache
    try {
      return new DataInputStream(new FileInputStream(localFile));
    } catch(FileNotFoundException e) {
      return null;
    }
  }
}
```

Dodatek D

Obsah přiloženého CD

Přiložené CD obsahuje následující adresáře:

`/tex`	– zdrojové kódy pro vygenerování technické zprávy pomocí systému LaTeX.
`/pdf`	– technická zpráva ve formátu PDF.
`/src`	– zdrojové kódy ukázkového programu (včetně prostředí AspectJ).
`/AspectJ`	– instalační balíček prostředí AspectJ.