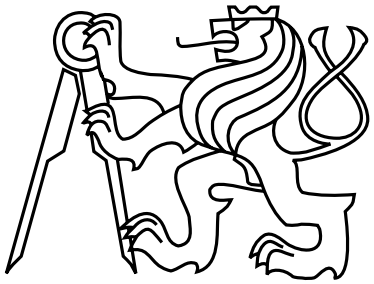
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce



Diplomová práce

Aspektově orientovaný vývoj uživatelských rozhraní pro Java SE aplikace

Bc. Martin Tomášek

Vedoucí práce: Ing. Tomáš Černý M.S.C.S.

Studijní program: Otevřená informatika, Magisterský

Obor: Softwarové inženýrství

1. prosince 2014

Poděkování

Tímto bych rád poděkoval celé svojí rodině za podporu během studia. Dále bych rád poděkoval vedoucíme mé diplomové práce panu Ing. Tomáši Černému za ochotu, pomoc, čas, zkušenosti a příležitosti, které mi poskytoval během celého mého studia.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

Ve Strakonicích 1. 12. 2014 .....

Abstract

Abstrakt

Dnešní aplikace využívají grafické uživatelské rozhraní k interakci s uživatelem. V každé aplikaci můžeme najít vstupní pole, výstupní pole a další komponenty, které slouží k ovládání aplikace. Generování formulářů, která obsahují data, jenž chceme vkládat, editovat nebo prohlížet je časově náročné. Kromě generování správných vstupních polí je také potřeba nahlížet na tvorbu uživatelského rozhraní z pohledu validací, rozložení komponent a bezpečnosti. V těchto případech by bylo vhodné, aby se části rozhraní generovala na základě modelu. V případě aplikací typu klient server může model a data poskytovat server. Toto řešení zajistí centralizovanou správu dat a jejich definicí a umožní klientovi pružně reagovat na změnu datového modelu.

Obsah

Seznam obrázků

Seznam tabulek

Seznam částí zdrojových kódů

Kapitola 1

Úvod

Kapitola 2

Popis problému a specifikace cíle

2.1 Popis problematiky

Softwarové systémy jsou určeny k tomu, aby méně či více úspěšně poskytovali uživateli nástroj, který mu pomůže s řešením problémů. Systém tedy musí komunikovat s uživatelem. K tomuto účelu se využívá uživatelské rozhraní. Vývoj uživatelského rozhraní zabere přibližně 50 % času [2], který je určen na vývoj konkrétního systému. Tento údaj se samozřejmě může lišit v závislosti na účelu a velikosti systému. Při tvorbě uživatelského rozhraní se obvykle zaměřujeme na použitelnost. V tomto případě provádíme testy použitelnosti na cílové skupině, na jejichž základě jsme schopni určit, zdali je návrh použitelný či nikoliv. Důvodem tohoto testování je fakt, že obvykle systém vyvtváříme pro uživatele a ne obráceně. Z výše uvedených skutečností vyplívá, že je potřeba uživatelské rozhraní důkladně testovat, aby bylo pro cílovou skupinu správně použitelné. Bohužel, když se hovoří o uživatelském rozhraní, tak se často zapomíná na to, že toto rozhraní se musí nejen vytvořit, ale také udržovat. Softwarový systém tráví většinu svého života v udržovacím režimu, kterému se říká support nebo li podpora. V této fázi přichází na systém mnoho požadavků, které musí být proveditelné a to za přijatelné náklady. Nedílnou součástí jsou změny, které se týkají databázového modelu a obvykle tyto změny musí reflektovat UI. Podívejme se proto na systém z pohledu vývojáře. Systém pro něj musí být snadno udržovatelný, změny lehce proveditelné a bez větších dopadů na systém. V tomto případě by bylo vhodné reflektovat tyto změny v UI. Nedílnou součástí každého uživatelského rozhraní jsou validace, které by měli reflektovat například změny v databázovém modelu ale i změny v business modelu.

2.1.1 Typy uživatelských rozhraní

Jak již bylo zmíněno, tak uživatelské rozhraní se testuje na základě typu aplikace a jejím použití. Je také důležité vzít v potaz zařízení, na kterém je aplikace provozována. Může se jednat o desktopovou, mobilní či serverou aplikaci. V každém z výše uvedených případů bude návrh uživatelského rohraní podíměn jinými faktory, které jsou specifické pro dané zařízení. Těmito faktory jsou způsoby, jakými se aplikace ovládá, prostředí, v kterém se uživatel právě nachází a účel, ke kterému je aplikace určena. Například aplikace na mobilních zařízení nemusí podporovat klávesové zkratky, ale mohla by podporovat gesta. Obdobně aplikace použitá na desktopu může počítat s použití myše, touchpadu, klávesnice - jak standardní

tak dotykové, či jiného externího zařízení. Je tedy zřejmé, že uživatelské rozhraní je kromě jeho účelu podmíněno i zařízením, na kterém je používáno.

Základnímy ovládacími a vizuálními prvky téměř každé aplikace prvky jsou tlačítka, vstupní pole, přepínače, tabulky, menu a statické texty. Vstupní pole můžeme shrnout do jedné kategorie, která se nazývá formulář. Formulář obvykle osahuje 1 až N prvků, s tím, že každý prvek má zde svojí funkčnost a účel. Účelem je poskytnout uživateli možnost vložení dat, či možnost volby chování aplikace. Funkčností je tato data správně interpretovat a na jejich základě provést specifické akce. Ve formuláři také mohou být pouze statická data, která slouží k reprezentaci aktuálního stavu, který slouží uživateli k tomu, aby pochopil aktuální stav ve kterém se aplikace či jeho část nachází a na základě tohoto stavu mohl rozhodnout o další akci, pokud je toto rozhodnutí vyžadování a umožněno.

2.1.2 Získávání a vkládání dat

Aplikace použí vizuální prvky k tomu, aby uživateli reprezentovala data či umožňila uživateli tato data vytvořit. Moderním způsobem je dnes využívat k získávání a vkládání dat webové služby. Výhodou je, že se klient může připojit na různé zdroje a z těchto dat si vytvářet tzv: mashup. Obvyklé použití je takové, že server získá data z více zdrojů a ty pak interpretuje klientům. Klient tedy nezná orignální původ informací. Jedním z dalších způsobů je vlastní databáze na klientovi. V tomto případě již ale nemůžeme hovořit o klientovi, neboť se jedná o soběstačnou aplikaci, v případě, že nezískává data z jiných dalších zdrojů. Samozřejmě existují i kombinace těchto možností. Volba závisí vždy na konkrétním zadání a účelu, pro který je aplikace navržena.

V případě reprezentace je potřeba data získat. Jak již zbylo zmíněno výše existuje mnoho způsobů kde a jak data získat. Zaměříme se nyní na získání dat z jiných zdrojů. Pokud žádáme o data jiný zdroj, tak jsme obvykle schopni zjistit formát a způsob, jakým o data požádat, ale formát dat neznáme. Nejčastěji jsou data přenásena jako JSON [1] nebo XML. To nám umožní data serializovat do objektu, pokud známe definici objektu. Definice objektu lze získat obvykle v dokumentaci k dané službě, takže námi navržená aplikace očekává data specifického typu. Uvažujme nyní příklad, kdy jsme vytvořili klienta, který zobrazuje jména a příjmení uživatelů v systému. Po určité době, je však potřeba kromě jména i zobrazovat jejich uživatelské jméno. Do dat, která získáváme od služby tedy přibude sloupeček s uživatelským jménem. Nyní musíme naší klientskou aplikaci upravit, tak aby byla schopná zobrazovat i tyto informace. Provedli jsme tady poměrně triviální úpravu. Přidali jsme pole k zobrazení uživatelského jména, upravili jsme objekt, do kterého se data serializovala a v další verzi vydání aplikace se tato změna projeví. Změna tedy není klientům dostupná ihned. Po uričté době se rozhodlo, že se kolonka uživatelského jména odstraní. Tento případ je tedy mnohem horší. Neboť po serializace bude v poli reprezentující uživatelské jméno hodnota null. Pokud jsme jméno pouze vypisovali tak je vše v pořádku, avašak pokud jsme nad ním volali nějaké operace můžeme obdržet vyjímku a aplikace nebude schopna pokračovat v běhu.

V případě vkládání dat do jiného zdroje platí chování a nastavení z odstavce uvedeného výše. Je tedy potřeba znát zdroj, na který data odeslat, formát dat, metodu a popřípadě další dodatečná nastavení služby. Budeme uvažovat stejný příklad jako byl již rozebrán v předchozím odstavci s tím rozdílem, že nyní data vkládáme. Na server tedy nejprve odesíláme pouze jméno a přijímení. Předpokládejme, že tyto dvě hodny jsou serverem vyžadovány. Je

tedy potřeba mít validaci, která hlídá zdali jsou data před odesláním v pořádku, či správně interpretovat odpověď serveru, který sdělí, že data nejsou validní, pokud touto validací disponuje. Při přidání nového pole, u kterého server vyžaduje jeho vyplnění nyní klient přestane správně fungovat a server by měl data odmítat. Musíme tedy udělat změnu na klientovi. Přidat vstupní pole, přidat proměnou, která bude zastupovat uživatelské jméno a novou verzi nasadit a distribuovat. Po určité době, stejně jako v prvním případě se definice dat změní a uživatelské jméno již nebude klientům zasíláno a nebude ani možnost ho vyplňovat. Klient se opět stane nevalidním, neboť při serializaci na straně serveru dojde k chybě, protože klient posílá proměnou, kterou již nyní server neznám. Formulář se tedy opět stane nefunkčním.

2.1.3 Aspektový přístup

Z dosavadního testu je zřejmé, že aplikace obsahuje vizuální prvky, na které lze nahlížet z několika aspektů. Jedním z důležitých aspektů je bezpečnost. Funkce, které může konkrétní uživatel využívat se přidělují na základě uživatelských rolí. V této souvislosti je mnoho přístupů jak role přidělovat a spravovat, ale všechny způsoby mají jedno společné. Ověřeřují, zdali má uživatel právo akci provádět. Souvislost mezi rolí a uživatelským rozhraním je patrá. Mějmě uživatele v role administrátora. Tato role bude mít práva na úpravu uživatelských dat jiných uživatelů. Dále mějme roli hosta, která si může data uživatelů pouze zobrazit. Při detailnějším prozkoumání zjistíme, že existuje množina zobrazovaných dat, která je pro obě role stejná, nicméně pro roli hosta by měla být všechna tato data needitovatelná. Dosáhnout této možnosti lze několika způsoby a záleží na platformě a volbě řešení. Nicméně v Java SE aplikaci, která využívá SWING to znamená, že buď musí být data zobrazována jakou label nebo musí být komponenty vypnuty. V obou případech musí vývojář na základě uživatelské role zvolit jeden z přístupů a nastavovat data na konkrétní komponentě. Pokud zvolí způsob labelů, tak vytváří duplicitní formulář pouze s jiným aktivním prvkem. Pokud jsou data získávána ze serveru, tak na něm jsou již bezpečnostní politiky implementované a postačilo by jejich výsledky propagovat do klientské aplikace. Server by tedy sám rozhodl, jaká data zobrazit.

2.2 Existující řešení

V současné době existuje několik řešení, které se snaží zjednodušit tvorbu uživatelského rozrahní. Níže zmíněné technologie se zaměřují pouze na konkrétní framework. Například JSF, JSP, Swing, Android, Struts, Vaadin. Tyto řešení se zaměřují na inspekci objektů, která již nesou informace o daném objektu. Výhodou inspekce za běhu programu je to, že dokáží na základě typu dat postavit přesné uživatelské rozhraní. Mezi hlavní nevýhody patří samotné generování uživatelského rozhraní, které se dále velmi špatně donastavuje a samozřejmě také fakt, že pro inspekci je obvykle využita reflekce. Níže jsou uvedeny současné frameworky, které umožňují vytvářet generovaná uživatelská rozhraní.

2.2.1 SwinXml

Tento framework se zaměřuje pouze na generování uživatelského rozhraní ve Swingu. Základem je specifikace rozhraní pomocí XML, což je velmi velkou výhodou. neboť specifikace v

tomot formátu má jasně dané možnosti a je na první pohled zřejmé jak bude dáná komponenta fungovat. Knihovna implementuje téměř všechny možnosti, které lze nastavit standardní cestou vývoje Swing aplikace. ActionListenery a dodatečné nastavení si vývojář může specifkovat po vygenerování komponent. Hlavní nevýhodou je, že všechny komponenty, které se generují je potřeba mít specifikované i ve výsledné aplikaci. Není zde žádné zapouzdření komponent a při detailnější specifikaci se stává XML definice až příliš rozsáhlá.

2.2.2 Metawidget

Projekt Metawidget se zaměřuje na vytváření uživateského rozhraní na základě inspekce tříd. Použít ji lze z mnoha populárními frameworky jako jsou Spring, Struts, JSF, JSP a další. Generování uživatelského rozhraní probíhá na základě inspekce již existující třídy a konfigurace konkrétní aplikace. Aktuální verze je 4.0 a vyšla 1.listopadu 2014. Jsou k dispozici pod licencí LGPL/EPL. Co se týče použitelnosti frameworku tak je Mezi hlavní výhody této knihovny patří zejména široká škála podporovaných frameworků a hlavně velká škála podporovaných validátorů. Data lze získat i pomocí REST, nicméně nativní a jednoduchá podpora zde chybí. Framework bohužel neumožňuje generování tabulek.

2.2.3 AspectFaces

Jedná se o framework, který umožňuje inspekci na základě tříd. Framework umožňuje použití různých layoutů a inspekčních pravidel. Výsledné vygenerované uživatelské rozhraní se může pro stejné obejkty lišit na základě specifického nastavení. V současné době je stabilní verze 1.4.0 a na verzi 1.5.0 se pracuje. Vývojář si může své vlastní nastavení generování tříd upravit. Tato nastavení jsou v XML formátu a lze je tedy snadno modifikovat. Framework podporuje velkou škálu anotací z JPA, Hibernate a uživatel si v případě nutnosti může vytvořit vlastní antoaci, která se promítne do inspekce. Framework je prozatím bohužel jednostranně orientovaný na JSF. Tomu odpovídá i způsob generování dat a způsob, jakým je prováděna složitější inspekce.

2.3 Cíle projektu

Existující řešení poskytují mnoho různorodých funkcí. Jejich hlavními výhody jsou zkrácení času, který je potřeba k vývoji a úpravě uživatelského rozhraní. V trendem současné doby jsou webové služby, proto se i v této práci bude soustředit na získávání definice dat z webových služeb a jejích interpretaci na klienta stejně tak jako na plnění této reprezentace skutečnými daty. Inspekce tedy bude prováděna na straně serveru, který zná objekty, s kterými pracuje a klient pouze obdrží jejich definici. Tento přístup umožní klientovi pružně a ihned reagovat na změny v datovém formátu, který diktuje server. Dalším pozitivním vlivem, bude to, že server bude klientovi poskytovat i seznam validací, jimiž musí jednotlivá komponenta vyhovět, aby bylo možné data odeslat zpět na server a ten je správně zpracoval. Celý tento proces by měl být pro kleinta zapouzdřen, aby aplikace nevyžadovala od klienta více informací než je nutné. Mezi nutnou informaci patří specifikace připojení a formát dat. Například JSON, XML. Použití frameworku by mělo být velmi jednoduché a v případě, že bude chtít klient postavit

formulář, tak by mu mělo stačit pouze několik řádků kódu. Dalším důležitým aspektem jsou již existující zdroje na serveru, které by měli po přidání frameworku zůstat stejné.

Kapitola 3

Analýza

Kapitola 4

Iimplementace

Kapitola 5

Testování

Kapitola 6

Závěr

Literatura

[1] *Java EE at a Glance* [online]. [cit. 8. 4. 2012]. Dostupné z: <http://www.oracle.com/technetwork/java/javaee/overview/index.html>.

[2] CERNY, T. – CHALUPA, V. – DONAHOO, M. Towards Smart User Interface Design. In *Information Science and Applications (ICISA), 2012 International Conference on*, s. 1–6, May 2012. doi: 10.1109/ICISA.2012.6220929.

Příloha A

Seznam použitých zkratek

A7B36ASS Architektury softwarových systémů

GNU GPL GNU General Public License

OCL Object Constraint Language

UML Unified Modeling Language

MVC Model-view-controller

AJAX Asynchronous JavaScript and XML

JAAS Java Authentication and Authorization Service

SSH Secure Shell

REST Representational State Transfer

JDBC Java Database Connectivity

OSGI Open Services Gateway Initiative

HTTP Hypertext Transfer Protocol

ORM Object relational mapping

EJB Enterprise JavaBean

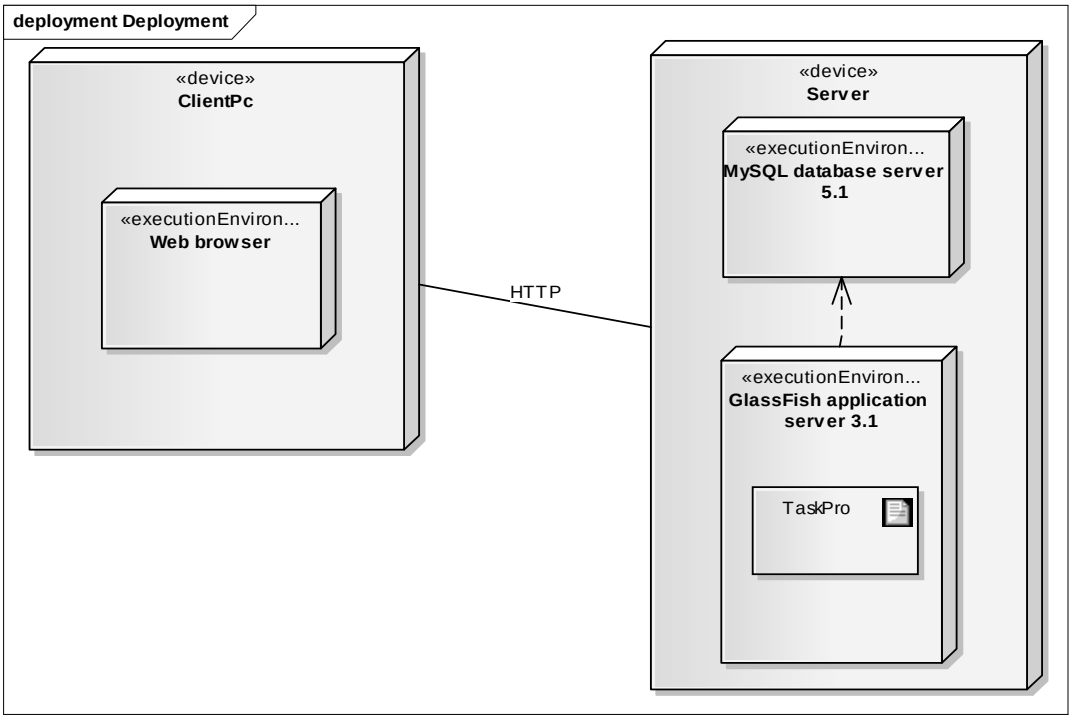
JPA Java Persistence API

API Application programming interface

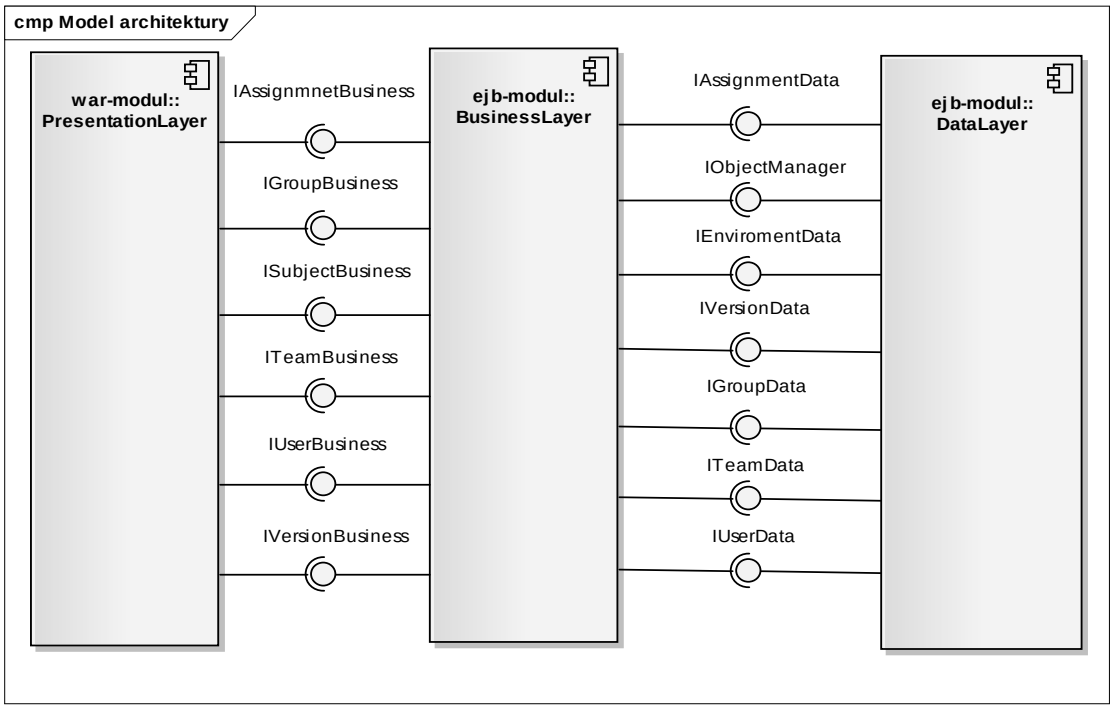
Příloha B

UML diagramy

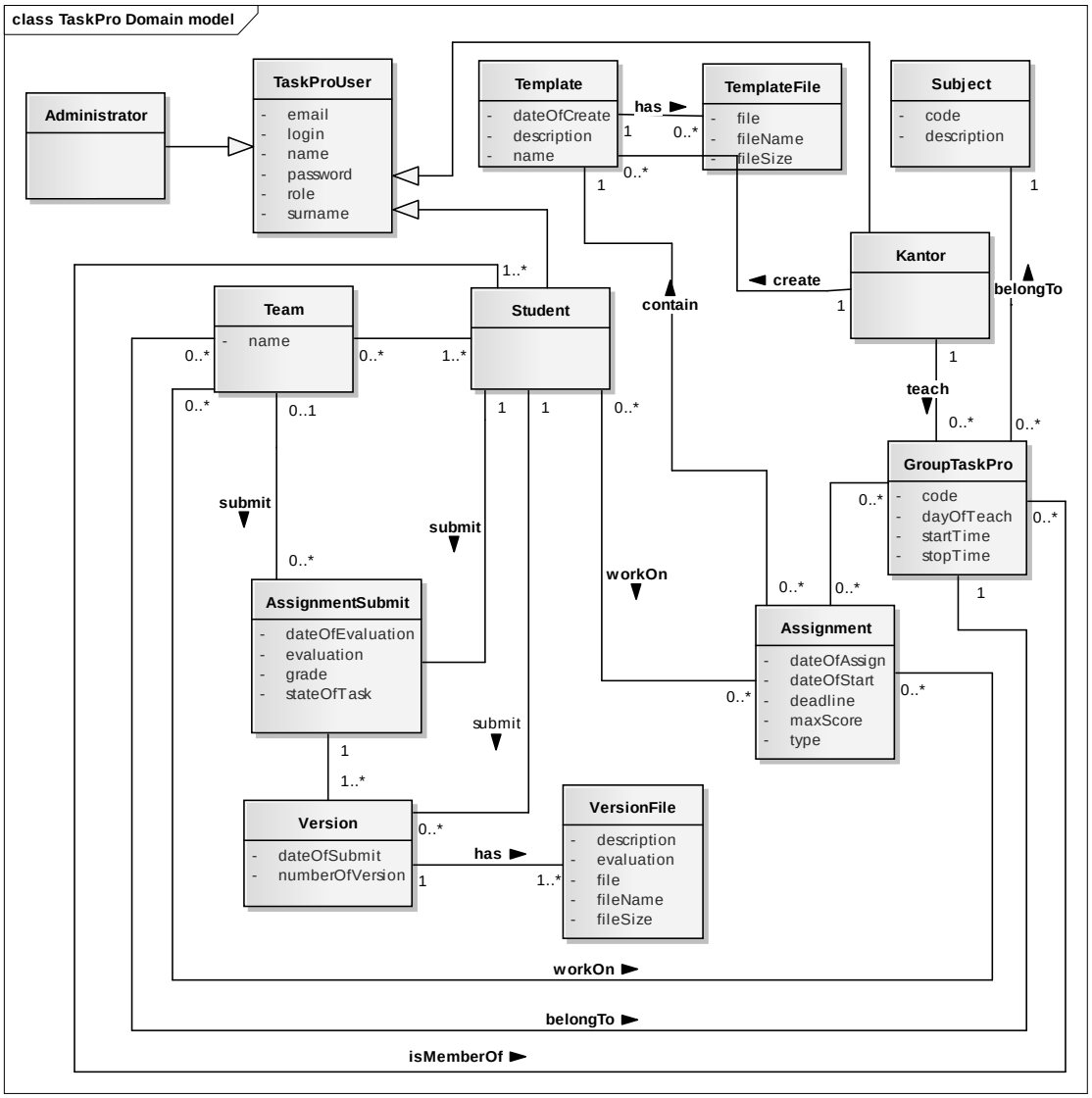
V této sekci naleznete použité UML diagramy, na které bylo v textu odkazováno.



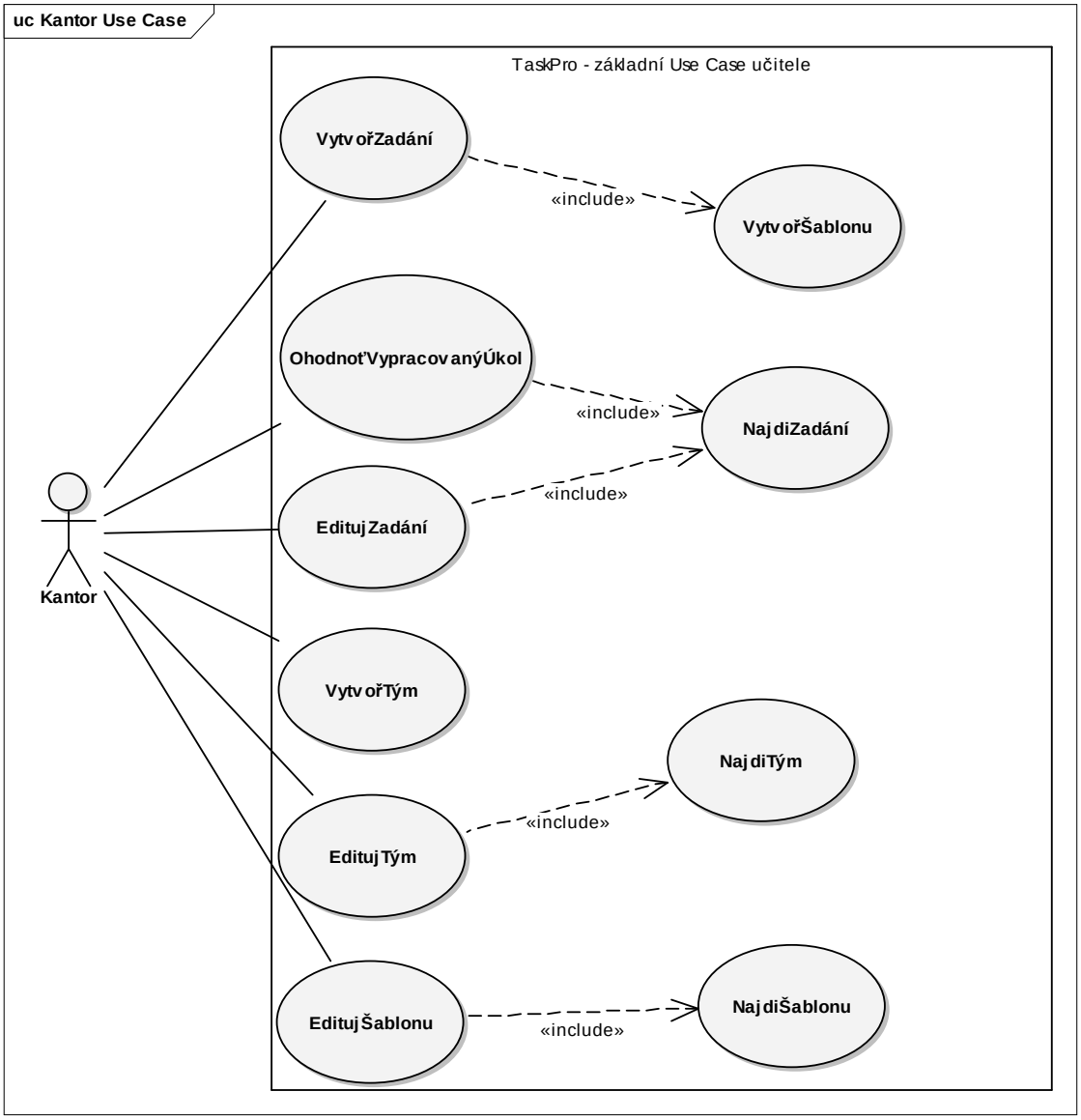
Obrázek B.1: Diagram nasazení



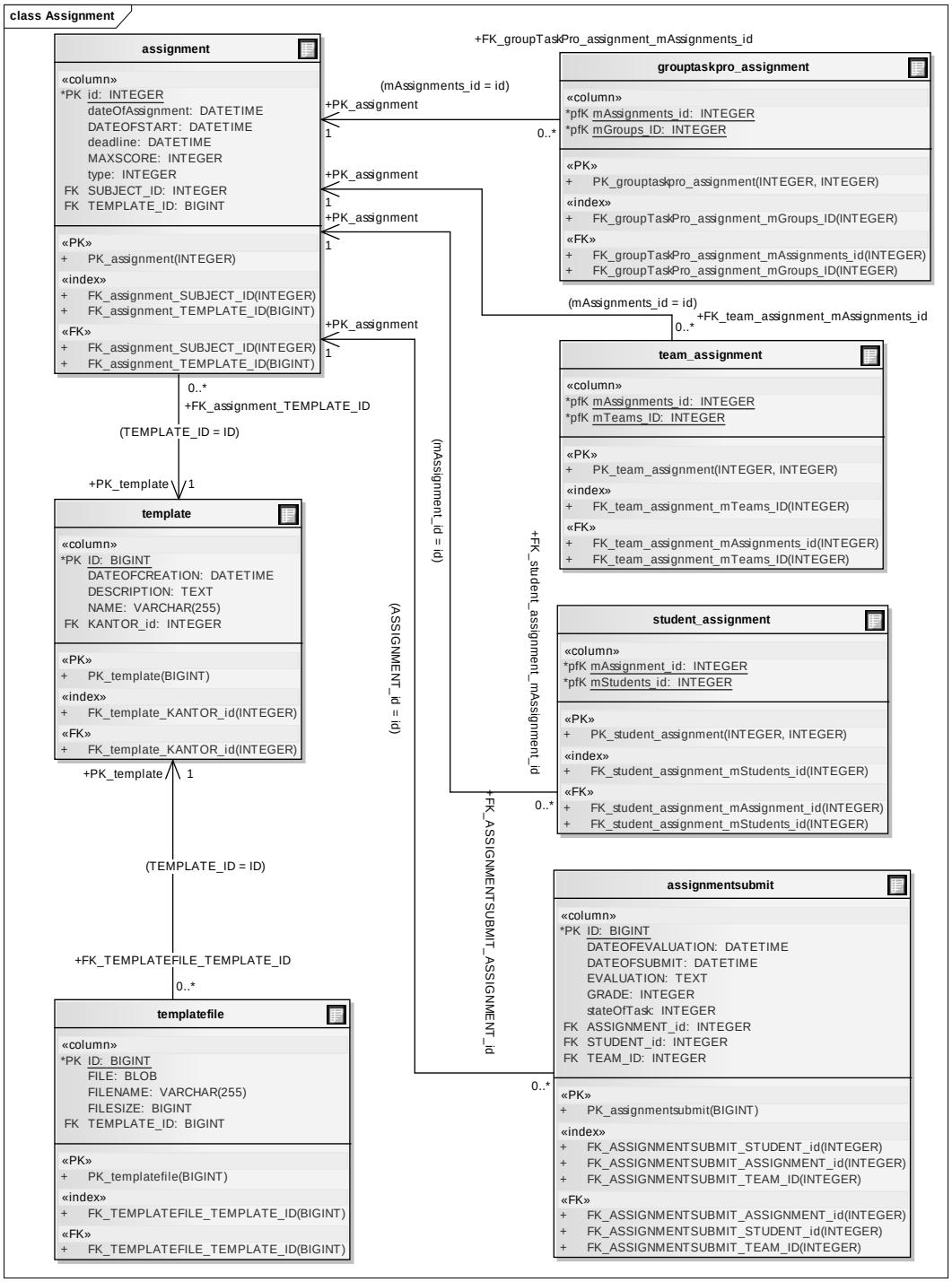
Obrázek B.2: Architektura aplikace



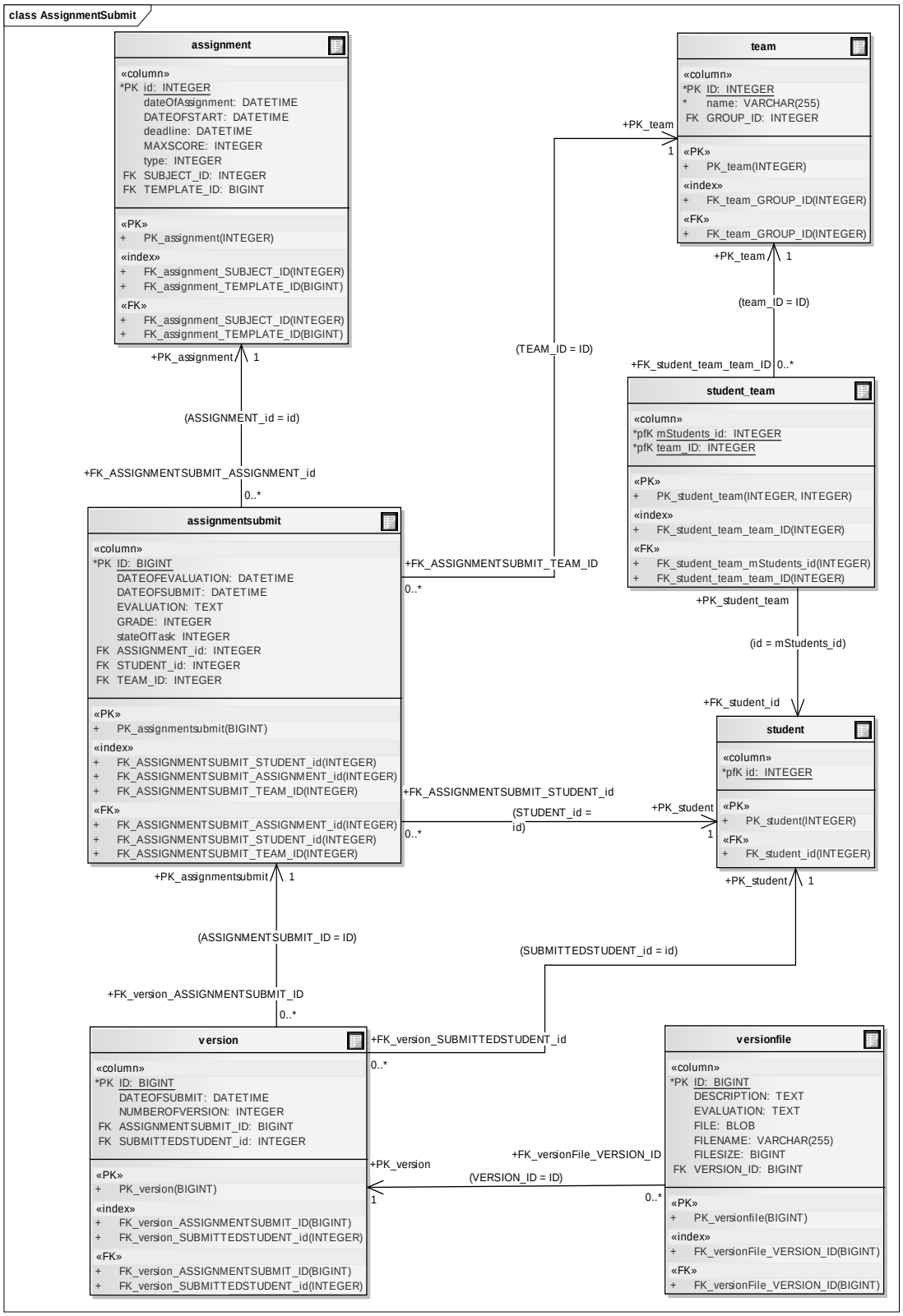
Obrázek B.3: Doménový model systému pro zadávání a odevzdávání úkolů



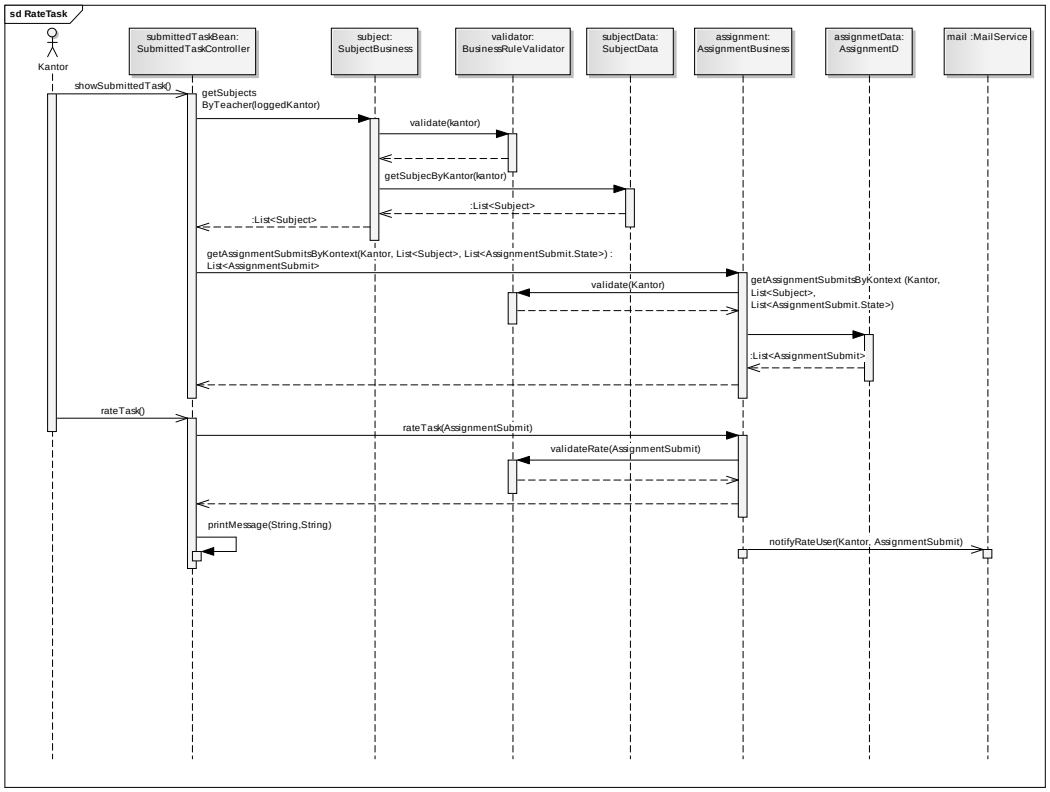
Obrázek B.4: Základní případy užití pro učitele



Obrázek B.5: Relační schéma zadání



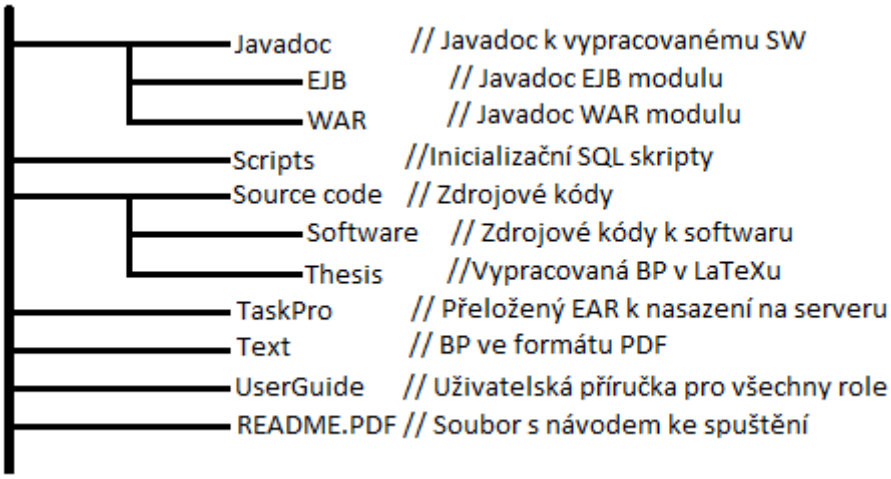
Obrázek B.6: Relační schéma odevzdání zadání



Obrázek B.7: Systémový sekvenční diagram zachycující oznámkování úkolu

Příloha C

Obsah přiloženého CD



Obrázek C.1: Obsah přiloženého CD