JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Ekonomická fakulta

Katedra aplikované matematiky a informatiky

Studijní program: B6222 Matematické metody v ekonomii

Studijní obor: Matematické modelování v ekonomii

Využití UML modelů při návrhu objednávkového systému

Vedoucí bakalářské práce

RNDr. Josef Milota

Autor

Ladislav Cibulka

2010

Prohlášení

Prohlašuji, že jsem bakalářskou práci na téma: Využití UML modelů při návrhu objednávkového systému vypracoval na základě vlastních zjištění a pomocí materiálů, které uvádím v seznamu použité literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích 10. 5. 2010

Ladislav Cibulka

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Ekonomická fakulta
Katedra aplikované matematiky a informatiky
Akademický rok: **2008/2009**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Ladislav CIBULKA**
Studijní program:	**B6222 Matematické metody v ekonomii**
Studijní obor:	**Matematické modelování v ekonomii**
Název tématu:	**Využití UML modelů při návrhu objednávkového systému**

Zásady pro vypracování:

Cíl práce: popsat vybrané UML diagramy a ukázat jejich využití při vývoji objednávkového systému.

Metodický postup:

1. UML (Unified Modeling Language) - úvod a popis základních prvků a součástí.
2. Výběr vhodných UML diagramů pro praktické použití a jejich podrobnější popis (diagramů tříd, stavových diagramů a diagramů užití).
3. Výběr a popis vhodného nástroje pro tvorbu UML diagramů.
4. Aplikace vybraných UML diagramů při návrhu objednávkového systému.
5. Prezentace výsledků a formulování závěrů.

Rozsah grafických prací:	**6-10 diagramů**
Rozsah pracovní zprávy:	**40 - 50 stran**
Forma zpracování bakalářské práce:	**tištěná**

Seznam odborné literatury:

ArgoUML Project [online]. [cit. 20.3.2009]. Dostupné na WWW: http://argouml.tigris.org.

Miller, R. Practical UML: A Hands-On Introduction for Developers [online]. [cit. 26.3.2009]. Dostupné na WWW: http://dn.codegear.com/article/31863.

Page-Jones, M. Základy objektově orientovaného návrhu v UML. 1. vyd. Grada Publishing. Praha: 2001. ISBN 80-247-0210-X.

Schmuller, J. Myslíme v jazyku UML: knihovna programátora. 1. vyd. Grada Publishing. Praha: 2001. ISBN 80-247-0029-8.

UML Resource Page [online]. [cit. 20.3.2009]. Dostupné na WWW: http://www.uml.org/.

Vedoucí bakalářské práce:	**RNDr. Josef Milota** Katedra aplikované matematiky a informatiky
Datum zadání bakalářské práce:	**20. února 2009**
Termín odevzdání bakalářské práce:	**15. dubna 2010**

prof. Ing. Magdalena Hrabánková, CSc.
děkanka

prof. RNDr. Pavel Tlustý, CSc.
vedoucí katedry

V Českých Budějovicích dne 30. března 2009

Obsah

1. Úvod a cíl práce

Vývoj informačních systémů a technologií již dávno není jednoduchou záležitostí. Do vývojového procesu jsou zapojeny velké skupiny lidí, které musí úzce spolupracovat. Jako v každé jiné lidské činnosti, tak i v procesu vývoje aplikací, je pro usnadnění komunikace a spolupráce potřebné mít dostatek dorozumívacích prostředků, které jsou jednoznačné a všem srozumitelné. Jednou z variant komunikačních prostředků, které jsou v současné době k dispozici, je sjednocený modelovací jazyk, který umožňuje vytvářet UML modely.

Cílem této práce je podat základní informace o sjednoceném modelovacím jazyku UML a jeho využití při návrhu konkrétního systému. Součástí práce budou vybrané UML diagramy a jejich aplikace na objednávkový systém vyvíjený pro konkrétní firmu.

První část této práce bude rozdělena do třech kapitol zaměřených na teoretický obsah problému. Kapitoly budou orientovány na seznámení s UML modely a budou popisovat vlastní vývojový proces. Stěžejní částí práce budou kapitoly o UML diagramech, ve kterých budou popsány hlavní součásti UML.

Další část bude zaměřena na praktické využití UML při vývoji internetového objednávkového systému pro firmu, jejímž hlavním předmětem činnosti je produkce rostlin. Zde bude proveden návrh aplikace, popsány konkrétní diagramy a hlavní funkcionality. Při navrhování diagramů bude použit program ArgoUML verze 0.26.2.

Při zpracování tématu budou použity metody modelování a abstrakce. Pomocí modelování budou názorně zobrazeny jednotlivé komponenty.

2. Sjednocený modelovací jazyk (UML)

2.1. Představení UML

Sjednocený modelovací jazyk – UML (Unified Modeling Language) je grafická notace podporovaná nezávislým meta-modelem. Umožňuje popisovat a navrhovat softwarové systémy založené na základě objektově orientované architektury. UML lze také využít při procesu softwarového inženýrství pro všechny druhy aplikací. [Fowler, s. 23] UML je složeno ze tří hlavních komponent [Booch, s.18]

2.1.1. Stavební bloky

UML je sestaven ze tří bloků.

- Předměty (things)
  Předměty jsou abstrakce v modelu, které se dají dále dělit.
  - Strukturní abstrakce (structural things)
    Jsou podstatná jména v modelu (třídy, rozhraní, komponenta, uzel, atd.)
  - Chování (behavioural things)
    Slovesa v modelu (interakce, stav)
  - Seskupení (grouping things)
    Balíčky používané k seskupování významově souvisejících prvků modelu do soudržných jednotek.
  - Poznámky (annotional things)
    Připojení zvýrazněných poznámek k modelu. [Booch, s. 19 – 24]
- Vztahy (relationships)
  Relace umožňuje ukázat na modelu, jaký je významový vztah mezi dvěma předměty. [Booch, s. 24]
- Diagramy (diagrams)

Diagramy graficky popisují požadované chování softwarového systému – model. Diagramy jsou okna nebo pohledy na model. [Booch, s. 92]

2.1.2. Společné mechanizmy

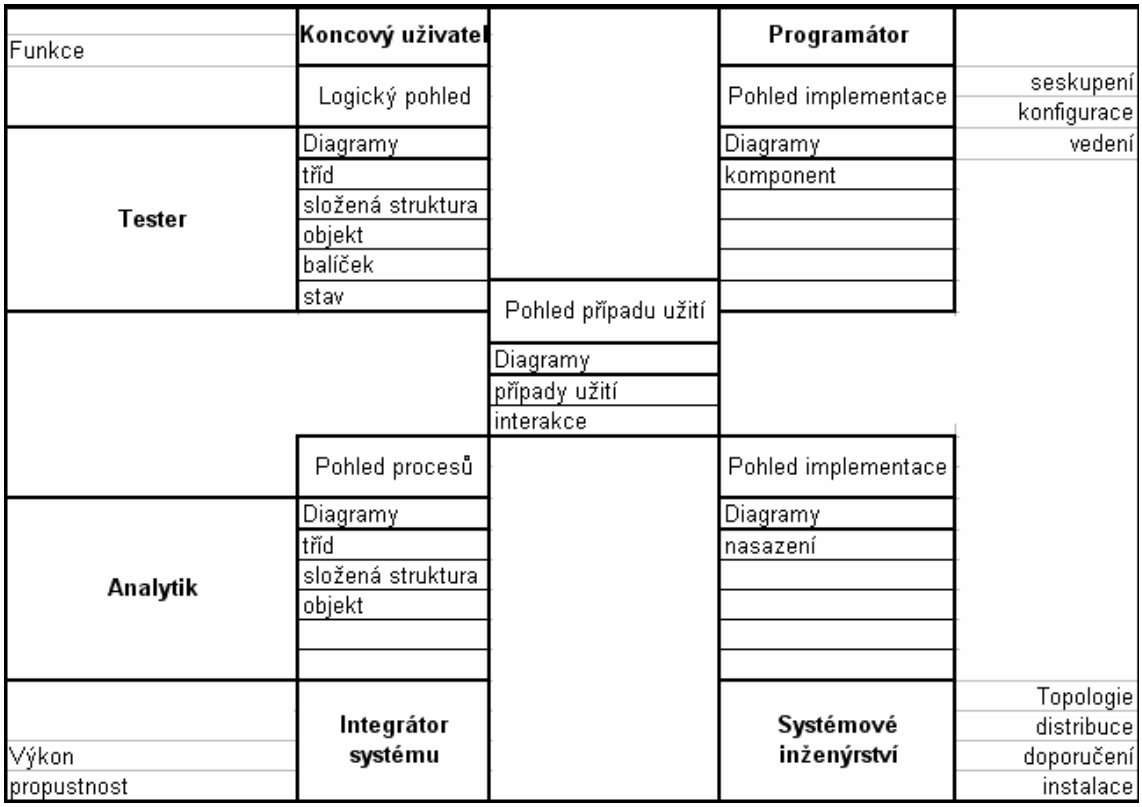
UML obsahuje čtyři společné mechanizmy používané v celém jazyku. Popisují cesty k modelování objektů. [Arlow, s 40 – 42]

- Specifikace
  Specifikace je jádro a podstata modelu UML, sémantický základ modelu.
- Ornamenty
  Zvýrazňují či zdůrazňují důležité detaily.
- Podskupiny
  Popisují různé způsoby náhledu na problém.
- Mechanizmy rozšiřitelnosti
  Navrhují způsoby pro možnost rozšíření mezi všechny uživatele UML. Využívají omezení, stereotypy, označené hodnoty a profily UML.

2.1.3. Architektura

Architektura UML je organizační struktura systému, včetně rozkladu na součásti, propojitelnost, interakci, mechanizmy a směrné zásady, které pronikají do návrhu systému. Zachycuje aspekty vyšší struktury systému, využívá k tomu čtyř pohledů na systém. [Arlow, s 43 – 47]

Obrázek č. 1: Pohledy na architekturu a proces vývoje



2.2. Historie UML

V osmdesátých letech 20. století se zrodilo objektově orientované programování. Objekty byly postupně propouštěny z výzkumných laboratoří do programátorského světa. Průlomem bylo vytvoření objektově orientovaného programovacího jazyka (C++). V té době se začalo s úvahami nad objektově orientovaným grafickým jazykem pro návrh systémů.

Až do roku 1995 nebyl stanoven žádný standard. Existovalo několik konkurenčních metodik. Nejvýznamnější byly metody Booch a OMT (Object Modeling Technique, Rumbaugh). [Arlow s. 30] V roce 1995 uvedli Booch a Rumbaugh první veřejnou prezentaci jejich sloučené metody; verze 0.8 dokumentace Sjednocené metody (Unified Metod). [Fowler, s. 29]

V lednu 1997 byl podpořen návrh standardních metod, které by usnadnily výměnu modelů a slučitelnost CASE nástrojů. **CASE nástroj** je označení pro Computer Aided Software Engineering (počítačem podporované softwarové inženýrství). Umožňují modelování systémů pomocí diagramů, generování kódu z modelu, zpětné vytvoření modelu, synchronizaci modelu se zdrojovým kódem a vytvoření dokumentace z modelu. [CASE nástroje]

Společnost Rational Software vydala dokumentaci ke standardu UML 1.0. Tím se zrodilo UML. Následovala intenzivní práce na sjednocování návrhů modelů. OMG (Object Management Group) přijala za standard UML 1.1, který vznikl z této činnosti. Další revize následovaly až do verze UML 1.6. V roce 2005 byly dokončeny práce na specifikaci UML 2.0. V současné době je UML 2.x běžný standard a podle údajů OMG je vytvořena beta verze UML 2.3. [OMG Modeling and metadata specifications]

Budoucnost jazyka UML je iniciativa nazvaná MDA (Model Driven Architecture). [MDA] Jedná se o vývoj software na základě modelů. MDA oproti dnešním CASE nástrojům nabízí daleko vyšší stupeň automatizace procesu vývoje a nezávislost na platformě a počítači.

2.3. Použití UML

Existuje několik způsobů použití UML. Záleží na jednotlivci k čemu UML využije. A protože se vedou neustálé polemiky, jak by se UML mělo používat, je zřejmé, že i v této oblasti dojde a bude docházet k bouřlivému vývoji. V současnosti jsou uváděny tři způsoby použití. [Fowler, s. 31]

2.3.1. Tvorba náčrtku

Náčrtek (UML as sketch) je zdaleka nejčastější použití UML. Slouží k usnadnění komunikace mezi vývojáři, zadavatelem a analytikem aplikace. Dále je používán v rámci dopředného inženýrství (forward engineering) – vytváření kódu na základě UML diagramu. Náčrtek se využívá i ke zpětnému inženýrství (reverse engineering) – UML diagram na základě existujícího kódu – slouží k interpretaci a pochopení zdrojového kódu či části systému.

2.3.2. Podrobný UML návrh

Hlavní význam podrobného návrhu (UML as blueprint) spočívá v kompletnosti. V rámci dopředného programování připraví návrhář detailní návrh systému pro programátory, kteří jej zakódují. Takový návrh musí být kompletní a musí být již učiněna všechna zásadní rozhodnutí ohledně chování celého systému. Podle podrobného návrhu musí být programátor schopen plynule pokračovat v procesu kódování bez vymýšlení funkcionalit systému. Díky kompletnosti návrhu lze ke kódování použít i CASE (computer-aided software engineering) nástroje (nástroje pro generování kódu). [CASE nástroje]

2.3.3. Programovací jazyk

UML jako programovací jazyk je použit, když je celý systém specifikován prostřednictvím UML a k jeho kódování jsou použity CASE nástroje do podoby spustitelného kódu. Tento způsob použití vyžaduje výkonné CASE nástroje. Díky notaci meta-modelu v UML2, kde je zachycena logika chování, lze vygenerovaný kód použít bez větších programátorských zásahů. Pro vývojáře v UML je nejpodstatnější součást meta-model, diagramy jsou jen jeho prezentací. Při užívání UML jako

programovacího jazyka je programátor omezen grafickými normami jednotlivých diagramů, takže některé specifické situace musí popsat textově ve zdrojovém kódu.

2.4. Proces vývoje

Každá aplikace projde svým vývojovým procesem. Při výběru typu vývoje záleží na rozsahu aplikace a vývojovém týmu. Lze vybírat z několika typů vývojových procesů. [Fowler, s. 37]

2.4.1. Iterativní a vodopádový proces

Tyto procesy mají společné, že projekt rozkládají. Vodopádový proces jej rozkládá podle činností a iterativní podle funkcionalit.

- **Vodopádový styl**

  K vybudování procesu jsou potřeba činnosti:

  - Analýzy požadavků
  - Návrh systému
  - Kódování
  - Testování

  Každá z činností má pevný časový harmonogram a jednotlivé části na sebe navazují. Ačkoli je proces pojmenován vodopádový i v něm může ve výjimečných případech docházet ke zpětným krokům až po návrat k analýze a návrhu.

- **Iterativní styl**

  Provádí se několik opakování stejných kroků s menšími částmi systému. Před začátkem iterace se provede pouze hrubá analýza, která poslouží k rozdělení do jednotlivých iteračních kroků. Každá iterace by měla vyprodukovat integrovaný

software připravený k nasazení do provozu. S iteracemi se používá technika časových oken (time boxing), která pevně vymezuje časový rámec iterace. [Fowler s. 38 – 40]

2.4.2. Prediktivní a adaptivní plánování

Tento druh procesu se využívá, když není jasná představa o době vývoje projektu a jeho nákladech.

- **Prediktivní plánování**

  Projekt probíhá ve dvou fázích. První fáze zajistí plány dalšího postupu a předpokládané náročnosti (jak časové, tak finanční). Výstupem druhé fáze je hotový funkční software. Problémem je, dojde-li k dodatečné změně požadavků v pozdější fázi vývoje projektu. Dojde ke zhroucení celého konceptu a je třeba začít plánovat od začátku.

- **Adaptivní plánování**

  Jedná se o přístup, kdy je se změnou požadavků během realizace projektu počítáno. Projekt je po celou dobu vývoje hodnocen jako nepředvídatelný, ale řiditelný. Lze pevně stanovit jen některé cíle a předpokladem je aktivní účast zadavatele při vývoji produktu. .[Fowler, s. 43]

2.4.3. Agilní metodiky

Agilní (agile) jsou silně adaptivní metodiky orientované na osoby účastnící se vývoje. Používají krátkých iterací a nekladou důraz na dokumenty – nízký stupeň formálnosti. Pro použití s UML nejsou příliš vhodné. [Fowler, s. 42]

2.4.4. Unified Process

Unified Process (UP) je metodika vývoje aplikace založená na iterativním a přírůstkovém procesu. Metodika byla vyvinuta v roce 1967 jako otevřený standard. Rozkládá projekt do celku více menších částí z důvodu snadné správy a úspěšného dokončení. Proces se skládá z pěti kroků – iterací.

- Plánování
- Analýza a návrh
- Tvorba
- Integrace a testování
- Uvedení, nasazení

Každá iterace generuje vlastní linii (baseline), která se skládá z částečně kompletní verze systému a dokumentace. Rozdíl mezi dvěma základními liniemi je označován za přírůstek. [Arlow, s. 53 – 55]

2.4.5. Rational Unified Process

Rational Unified Process (RUP) je komerční verzí metodiky Unified Process, která se od roku 2003 využívá ve společnosti Rational Sotware. Je rozdělen na čtyři fáze, mezi nimiž nejsou ostré hranice.

- Zahájení
- Rozpracování
- Konstrukce
- Předávání

V každé fázi vývoje lze s tímto procesem přejít do prediktivního režimu. [Fowler, s. 42]

2.4.6. Přizpůsobení procesu pro projekt

Díky odlišnostem softwarových projektů nelze na každý z nich aplikovat právě jeden vývojový proces. Proces se proto musí přizpůsobit tak, aby byl vhodný pro specifický projekt i vývojové prostředí. Důležitou součástí procesu je pozorování růstu a vývoje aplikace a jejich interpretace (retrospektiva iterace). Na základě těchto pozorování lze proces průběžně přizpůsobovat.

2.4.7. Zabudování UML do procesu

Používat UML se jeví jako prospěšné nejen při návrhu, ale i při realizaci softwarového projektu. UML lze využít nejen ke generování kódu CASE nástroji, ale pomáhá při dalších aktivitách. [Fowler, s. 45]

- **Analýza požadavků**

  Pochopení a interpretace očekávání zadavatele projektu. Grafické rozhraní odbourá problémy při komunikaci s netechnicky založeným zákazníkem. Je efektivnější používat jednodušší diagramy a v případu nutnosti lze postupovat i mimo rámec UML standardu.
- **Návrh**

  Při práci na návrhu se pracuje s přesnější a odbornější notací diagramů. U nich je očekávána implementace kódu. Každá odchylka kódu od návrhu je lehce odhalitelná. Použití UML je pružná metoda schopná okamžitě reagovat na změny.
- **Dokumentace**

  Pokud je software již hotov, je UML rychlý nástroj k vytvoření dokumentace (pokud není automaticky generována). Diagramy jsou schopny vytvořit logickou strukturu, fyzický popis, interakce systému i jeho jednotlivé stavy.
- **Pochopení významu kódu**

  Díky UML lze snáze pochopit kód, který je potřeba nastudovat.

2.5. UML Diagramy

Součástí UML je 12 různých diagramů, které jsou rozdělené do 3 skupin.

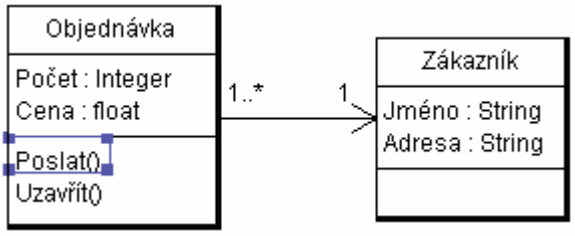
Tabulka č. 1: Rozdělení UML diagramů do skupin [BENEŠ]

Statická struktura	**Dynamické chování**	**Správa modulů**
Diagram tříd (Class Diagram)	Diagram případů užití (Use Case Diagram)	Balíčky (Packages)
Objektový diagram (Object Diagram)	Sekvenční diagram (Sequence Digram)	Subsystémy (Subsystems)
Komponentový diagram (Component Diagram)	Diagram činností (aktivit) (Activity Diagram)	Modely (Models)
Diagram nasazení (Deployment Diagram)	Diagram spolupráce (Collaboration Diagram)	
	Stavový diagram (Statechart Diagram)	

2.5.1. Diagram tříd

Diagram tříd (class diagram) je nejběžnější UML diagram. Popisuje objekty v systému a statické vztahy mezi nimi. Obdélníky v diagramu představují třídy. Každá třída je rozdělena do tří oddílů (comparments). První oddíl je název třídy, dále její atributy a operace. V diagramu tříd se také zobrazují rozhraní (třída bez implementace) a abstraktní třídy (třída od níž nemůžeme vytvořit instanci).

Obrázek č. 2 Jednoduchý diagram tříd



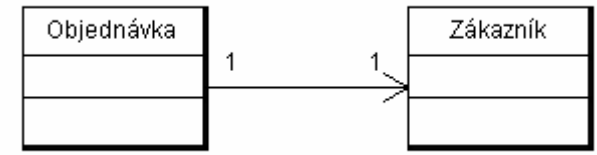
Vztahy mezi třídami

Vztahy mezi třídami jsou reprezentovány několika typy relací. Relace je významová vazba mezi jednotlivými modelovými prvky.

- **Asociace**

  Asociace je způsob vyjádření vlastností jednotlivých atributů i jejich násobnost. Je znázorněna plnou čarou případně se šipkou mezi dvěma třídami. Orientace šipky je ze zdrojové do cílové třídy – tím je definován datový typ dané vlastnosti. K cílovému konci se může napsat název vlastnosti a její násobnost. Asociace může být i obousměrná, v tom případě se šipky nezakreslují.

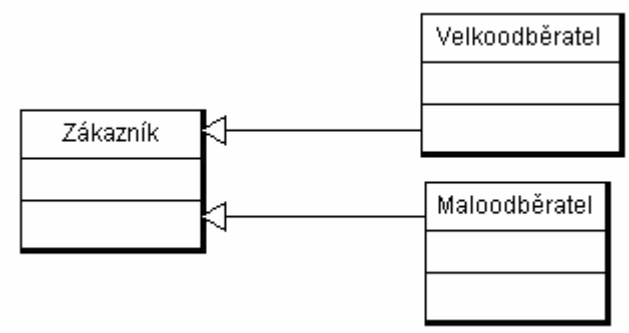
  Obrázek č. 3: Asociace objednávky k zákazníkovi



- **Zobecnění**

  Je grafické vyjádření dědičnosti objektů, kdy podtřída dědí vlastnosti nadtřídy. Zobecnění (generalizace) je znázorněno plnou čarou s trojúhelníkovou šipkou směřující k nadtřídě.

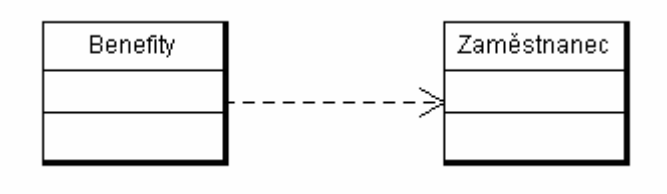
Obrázek č. 4: Generalizace velkoodběratele a maloodběratele



- **Závislost**

  Závislost (dependency) mezi elementy je, když změna jednoho elementu zapříčiní nutnost změny druhého. Pro správný běh programu je řízení závislostí důležité, při ztrátě kontroly hrozí dominový efekt a ztráta orientace v programu. Závislost je znázorněna čárkovanou čárou s šipkou směřující od závislé třídy.

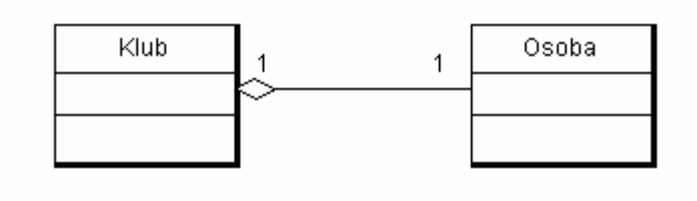
Obrázek č. 5: Závislost benefitů na zaměstnanci



- **Agregace**

  Agregace je relace typu celek/součást. Objekt (celek) používá služby dalšího objektu (součásti). Celek je v relaci dominantní, řídí její chod. Agregace je tranzitivní a asymetrická. Je zobrazena jako plná čára s šipkou směřující k součásti. U začátku agregace (celku) je prázdný kosočtverec.

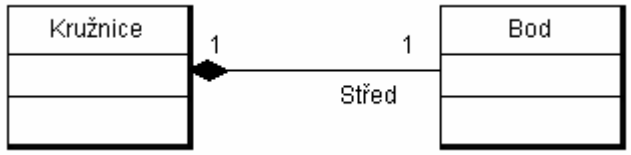
Obrázek č. 6: Agregace jednotlivých osob do klubů



- **Kompozice**

  Kompozice je přesněji definovaná forma agregace. V kompozici na rozdíl od agregace nemohou její součásti existovat mimo celek a každá z nich patří jen jednomu celku. Zobrazení je analogické jako u agregace, jen kosočtverec je plný.

Obrázek č. 7: Kompozice kružnice a jejího středu



2.5.2. Diagram aktivit

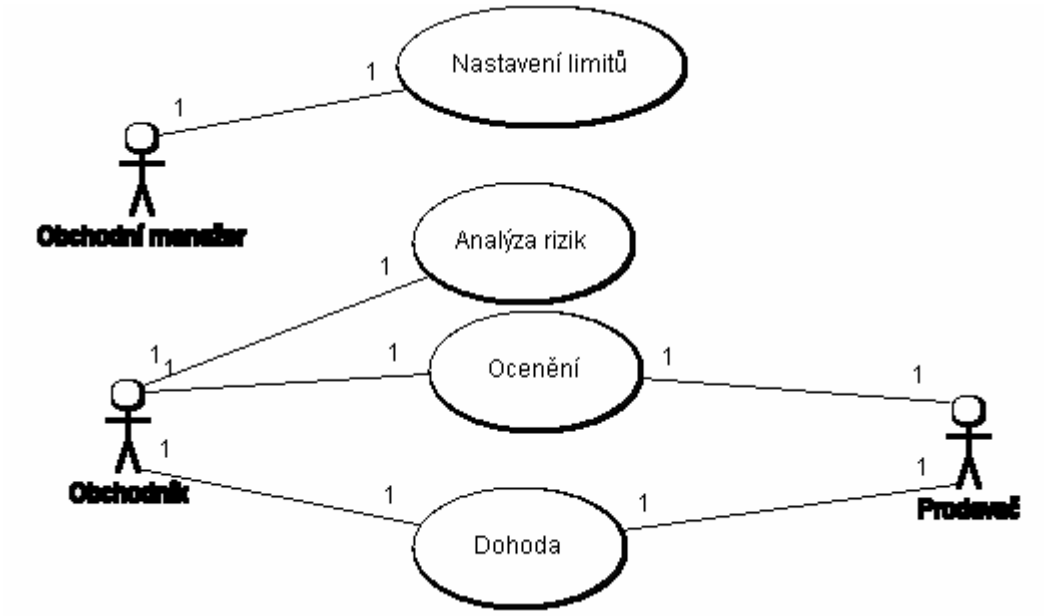
Diagram aktivit je technika určená k popisu procedurální logiky, business procesů a toku práce. Aktivita začíná v počátečním uzlu (initial node) až posléze se provádějí akce. Může dojít i k testování a následnému rozvětvení (fork) činností do paralelních toků, celá aktivita končí ve výstupním bodě ukončení aktivity. Diagram aktivit je vhodný pro modelování paralelního chování. Je rozšířen především mezi programátory v UML.

Obrázek č. 8: Diagram aktivit pro jízdu v autobuse

2.5.3. Diagram případu užití

Případy užití (use cases) jsou metodou zachycení funkčních požadavků na systém. Popisují interakce mezi uživateli systému a systémem a předkládají zprávy o tom, jak je systém používán. Případem užití je sada scénářů dohromady svázaná společným cílem uživatele. Uživatel je nazýván účastníkem nebo aktérem (actor též role). Diagram případu užití popisuje jen jak spolu jednotlivé role souvisí – nezachycuje jejich obsah, tím je funkčnost diagramu omezená. Struktura případu užití poslouží k promyšlení alternativ hlavního úspěšného scénáře.

Obrázek č. 9: Diagram případu užití

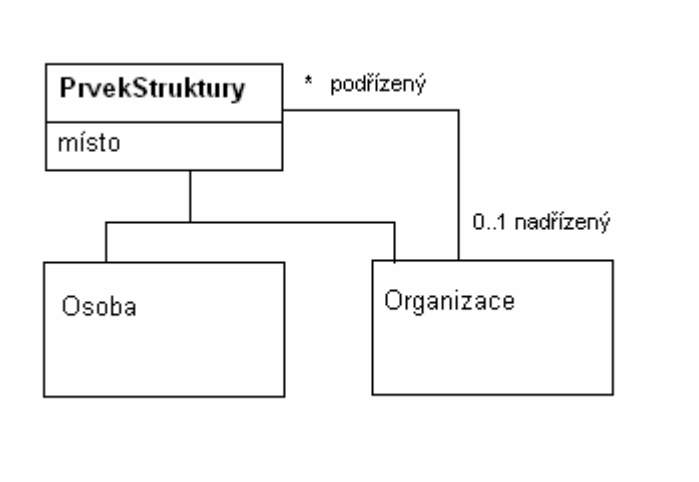


2.5.4. Diagram objektů

Diagram objektů (object diagram) je snímkem objektů systému v daném okamžiku. Místo tříd jsou v diagramu zachyceny jejich instance, specifikace instancí (objekty), a proto je někdy nazýván také jako diagram instancí. Diagram objektů je užitečný pro zobrazování příkladů objektů a jejich spojení. Často může být i přes podrobně

připravenou strukturu diagramu tříd problém jí porozumět, proto se vyplatí obtížnější části popsat diagramem objektů.

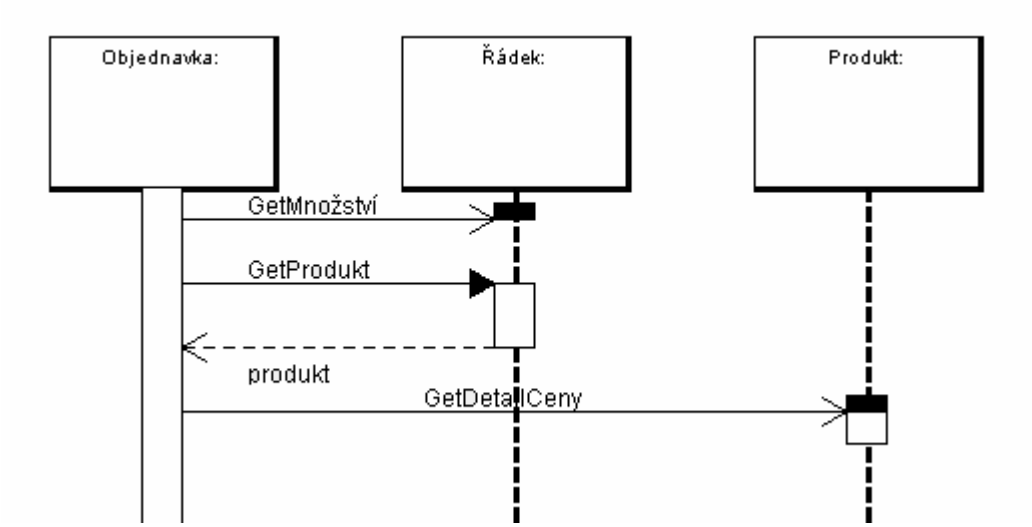
Obrázek č. 10: Diagram objektů k zobrazení konfigurace



2.5.5. Diagram sekvencí

Diagram sekvencí (sequence diagram) je nejběžnější z diagramů interakcí (interaction diagram). Zachycuje chování jednoho scénáře a ukazuje několik vzorových zpráv a objektů, které jsou předávány v rámci daného případu užití. Sekvenční diagram se znázorňuje následovně. Objekty jsou umístěny v horní části diagramu vedle sebe zleva doprava. Směrem dolů od každého objektu směřuje přerušovaná čára, která se nazývá životočára. [BENEŠ] Vykonání operace, kterou má objekt za úkol provádět, se označuje jako aktivace a je znázorněna jako úzký obdélníček podél životočáry. Délka obdélníku naznačuje, jak dlouho aktivace trvá. Podél životočáry vertikálně dolů ubíhá čas. Sekvenční diagramy obsahují také konstrukty. Podmínky se zapisují do hranatých závorek nad příslušnou šipku. Podmínky pak způsobují selekci (rozdvojení) toku řízení. Každá větev směřuje ke stejnému objektu, avšak tento objekt bude na každou z nich reagovat odlišně. To se znázorní rozdvojením jeho životočáry. Rozdvojené životočáry se později někde spojí. Sekvenční diagram se používá při pohledu na chování více objektů v rámci jednoho případu užití.

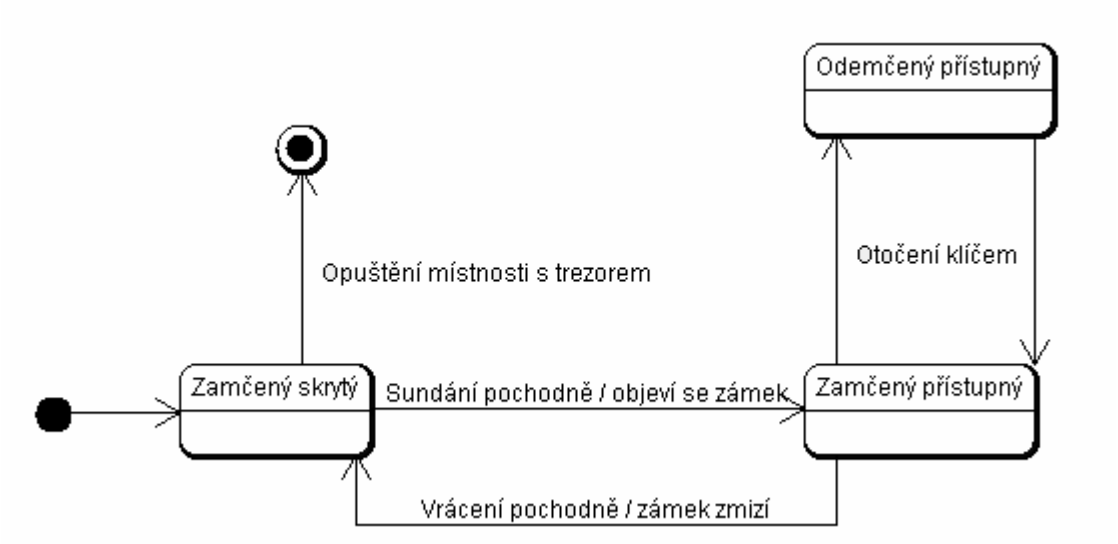
Obrázek č. 11: Diagram sekvencí



2.5.6. Diagram stavů

Diagram stavů (statechart diagram) je používán pro náhled na chování jednoho objektu v rámci více případů užití. Zachycuje stavy objektu, kterými může projít v průběhu svého životního cyklu v systému. Ze změny stavu pak plynou určité akce. Mohou být specifikovány i vstupní a výstupní akce. Každý stavový diagram musí mít počáteční stav, koncový stav není nutný. Stavy se mohou rozpadat do nadstavů, paralelních stavových diagramů. Stavové diagramy jsou vhodné k popisu chování objektu napříč několika případy užití. Není nutno kreslit stavový diagram pro každou třídu systému, vhodnější je vybrat jen několik tříd se zajímavým chováním.

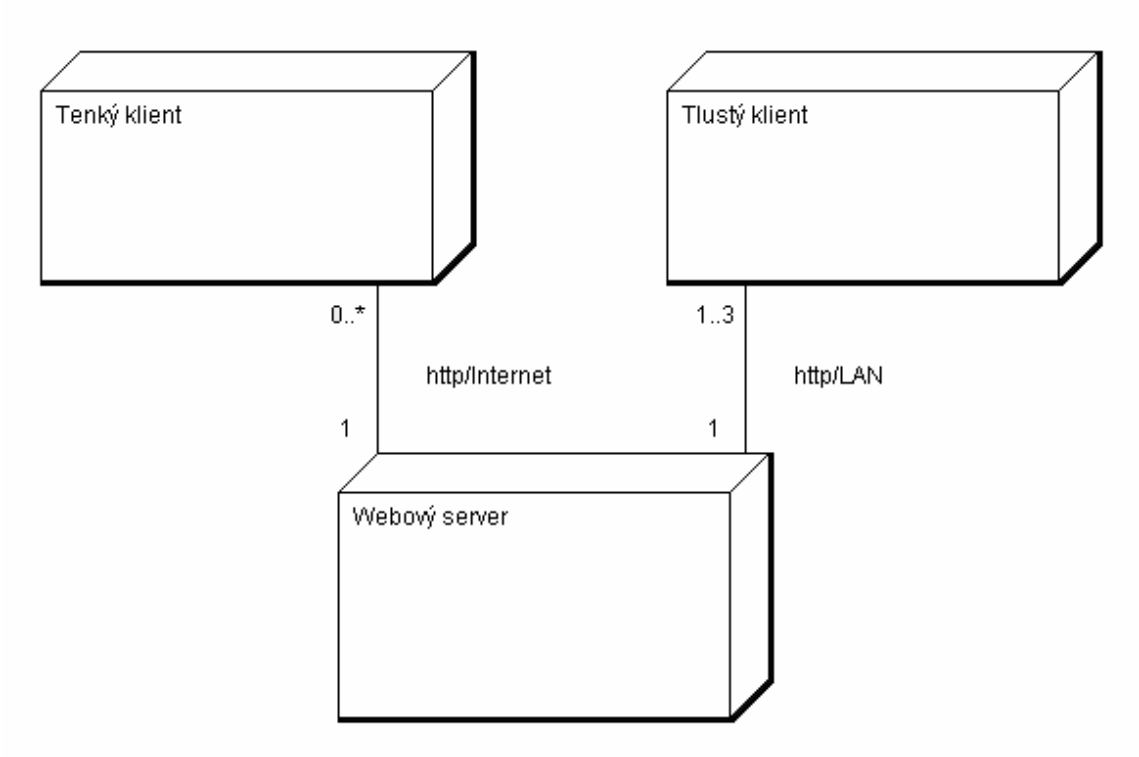
Obrázek č. 12: Stavový diagram – otevření skrytého trezoru



2.5.7. Diagram nasazení

Diagram nasazení (deployment diagram) ukazuje fyzické rozvržení systému a ukazuje jaký software běží na kterém hardwaru. Skládá se z uzlů (node), ty reprezentují software, dále ze zařízení (device), to je reprezentace hardware. Poslední složka je prostředí pro běh (execution enviroment), což je operační systém nebo kontejner. Mezi uzly jsou komunikační cesty, které ukazují jakým způsobem software mezi sebou komunikuje.

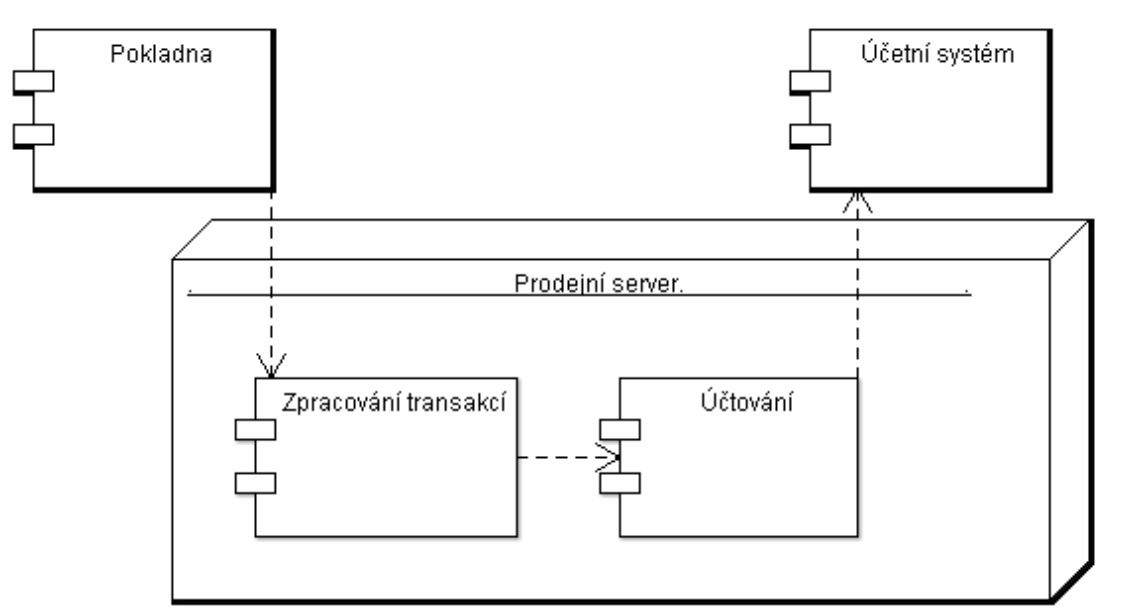
Obrázek č. 13: Diagram nasazení aplikace na tenkém a tlustém klientu



2.5.8. Diagram komponent

Diagram komponent je vhodné znázornění situace, pokud se systém dělí do více nezávislých částí. Diagram zobrazuje jejich vzájemné vztahy prostřednictvím rozhraní nebo rozkladu komponent do struktur nižší úrovně. Komponenta je modulární část systému, která zapouzdřuje svůj obsah a její projev je nahraditelný. Komponenta je samostatně fungující, prodejná a aktualizovatelná.

Obrázek č. 14: Diagram komponent pokladního prodeje



2.5.9. Diagram spolupráce

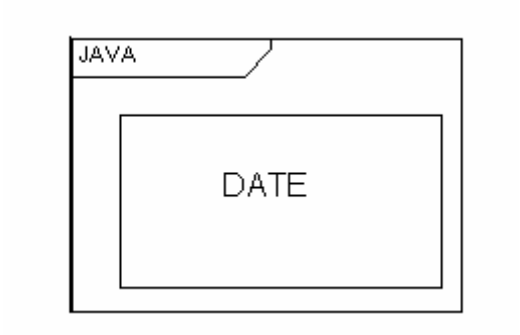
Diagram spolupráce (collaboration) neodpovídá žádnému oficiálnímu diagramu UML 2. [Fowler, s. 137] Standard pojednává o spolupráci jako o součásti složených struktur. Spolupráce poskytuje způsob jak seskupit kusy interakčního chování, kdy role jsou hrané různými třídami. Znázornění spolupráce se provádí v diagramu tříd pomocí elementu výskyt spolupráce (collaboration occurrence) – čárkovaný ovál ohraničující spolupracující třídy.

2.5.10. Diagram balíčků

Balíček (package) je sdružující konstrukce, která umožňuje seskupit libovolné UML elementy do společných jednotek vyšší úrovně. Nejčastěji je používán pro seskupování tříd. Každý balíček představuje jmenný prostor (namespace), ve kterém musí mít každý objekt jedinečný název. V diagramech jsou balíčky znázorněny jako pořadače (obdélník s ouškem). [Fowler, s. 96] Závislosti mezi jednotlivými balíčky jsou znázorněny

čárkovanou čárou se šipkou směřující k nadřazenému balíčku. Diagramy balíčků jsou užitečné pro zjednodušení představy o rozsáhlých systémech a závislostech mezi jejich hlavními prvky. Zobrazují mechanizmus seskupování objektů v okamžiku překladu aplikace. Grafické zobrazení balíčků a závislostí pomáhá udržet aplikaci při vývoji pod kontrolou.

Obrázek č. 15: Diagram balíčků, zobrazení obsahu balíčku



3. UML nástroje pro praktické použití

K využití UML v praxi je potřeba pracovat s nástroji, které jsou schopné modelovat UML diagramy. Nástrojů je celá řada a dají se rozdělit podle několika hledisek.

- **Cena pořízení**

  Z hlediska ceny pořízení je můžeme rozdělit na komerční a nekomerční šířené pod různými licencemi. Toto hledisko je asi nejzajímavější, proto s ním bude dále pracováno. Komerční nástroje jsou poměrně nákladné.

- **Možnost provozování**

  Z hlediska provozování nástrojů můžou být nástroje integrované do jiného programovacího nástroje, samostatně pracující nebo kombinace předchozích možností. Záleží na každém programátorovi, jaký nástroj preferuje či jak je zvyklý na své vývojové prostředí.

- **Hledisko otevřenosti kódu**

  Toto hledisko se týká nástrojů nekomerčních, je-li jejich zdrojový kód otevřen či nikoli. Dále pod jakou licencí je software šířen.

- **Obsáhlost nástroje**

  Každý nástroj nabízí jisté možnosti ulehčení vývoje, či přímo generování kódu. Záleží na druhu projektu a potřebách programátora, jak obsáhlý nástroj zvolí. Čím robustnější nástroj, tím obtížnější jeho obsluha.

Pro větší názornost jsou v příloze číslo 1 uvedeny screenshoty vybraných nástrojů určených pro tvorbu UML diagramů.

3.1. Komerční nástroje

Komerční UML nástroje jsou většinou velmi rozsáhlé profesionální produkty, pro jejichž úspěšné a efektivní používání je zapotřebí absolvovat školení či poměrně dlouhé

studium. Výsledkem nasazení těchto systémů však bývá výrazné urychlení práce, zejména prvotního návrhu systému.

3.1.1. Rational Rose

Software **Rational Rose**, původně vyvíjen společností Rational (nyní součást IBM), je nejznámější a pravděpodobně nejkvalitnější produkt pracující s UML. Systém je možno provozovat jako samostatnou aplikaci i jako integrální součást různých vývojových prostředí. V současnosti jsou podporovanými jazyky C, C++ a Java. Dále je k dispozici CASE nástroj, který převádí diagramy do kódu podporovaných jazyků (pro převod do jazyka C, který není čistě objektový, je použit jiný přístup). Největší nevýhodou je cena. Cena uživatelské licence je 5 020 USD. [IBM - Rational Rose Enterprise]

3.1.2. ARTiSAN Studio

Komerční systém **ARTiSAN Studio** [Artisan Studio] nabízí uživatelům podporu pro modelování a vývoj aplikací, zejména se zaměřením na systémy běžící v reálném čase. Mezi významné vlastnosti tohoto rozsáhlého aplikačního balíku patří podpora týmové práce, rozšíření jazyka UML o složky umožňující definovat real-time procesy a technologie pro získání UML diagramu ze stávajících zdrojových kódů aplikací. V současnosti jsou podporovány programovací jazyky C (neobjektové rozhraní), C++ a Java. K dispozici je také filtr, který z UML diagramu vytvoří programový stavový stroj s popisem jednotlivých stavů a závislostí mezi těmito stavy. Jeho nevýhodou je opět cena, která je v podobné výší jako u konkurenčního Rational Rose.

3.1.3. I-Logix Rhapsody

Systém **Rhapsody** byl vyvíjen americkou firmou I-Logix pro řešení problémů řízení a vývoje software s využitím UML diagramů. V roce 2008 byla společnost koupena firmou IBM a byla zařazena do korporace společně s firmou Rational Software. Z modelů lze automatizovaně vytvořit zdrojový kód s hlavičkami veškerých tříd spolu s jejich rozhraními. Tento systém také umožňuje správu poznámek a ručně kreslených grafů i schémat. Na tomto produktu je zajímavý také fakt, že je kromě podpory programovacích jazyků C, C++ a Java podporován i programovací jazyk Ada, který se používá například v některých projektech NASA a ve vojenských institucích armády USA. Předností tohoto produktu je jeho rozsáhlá funkcionalita, zejména při týmové práci většího množství vývojářů a vývojových skupin.

3.1.4. MetaMatrix MetaBase Modeler

Systém **MetaBase Modeler** [redhat] je tvořen soustavou několika relativně nezávislých aplikací, které mezi sebou navzájem komunikují. Účelem těchto aplikací je tvorba, sdílení a prezentace různých typů informací v podnikových informačních systémech. Při návrhu těchto systémů je využit modelovací a návrhový program založený na jazyku UML a UML diagramech. Tento systém také zajišťuje propojení s dalšími podnikovými informačními systémy, k tomuto účelu používá technologie založené na jazyku SQL. V roce 2007 byla společnost MetaMatrix převzata korporací RedHat a tento produkt je nabízen v rámci RedHat řešení. [redhat.com]

3.1.5. Borland Together Designer

Mezi základní vlastnosti tohoto systému patří podpora jednoduché tvorby UML diagramů, návaznost na další vývojové nástroje této firmy (JBuilder, C++ Builder) a také možnost navázání na softwarové nástroje jiných firem (Elipse). Poslední verze (z roku 2008) podporuje UML 2 a OCL 2.0. [Software architecture design from Borland]

3.1.6. Gentleware Poseidon

Poseidon je sofistikovaný modelovací program. Má příjemné, intuitivní uživatelské rozhraní, které usnadní práci programátora. Poseidon je možné stáhnout v omezené nekomerční verzi zdarma. Pro komerční využití je nutno vybrat z komerčních verzí. Cena jedné licence Poseidon Professional Edition je 875 USD. [Poseidon for UML]

3.1.7. MagicDraw

MagicDraw je modelovací software s podporou týmového vývoje. V programu lze vyvíjet pro jazyky Java, C++, C# a XML, dále podporuje modelování databázových schémat. Výhodou programu je rychlá navigace napříč návrhem a možnost propojování jednotlivých návrhů. Dalším kladem je automatické generování statických struktur, závislostí balíčků a hierarchie modelu. [UML Modeling Tool]

3.1.8. Enterprise Architect

Enterprise Architect je pokročilá modelovací platforma od společnosti Sparx Systems. Kromě nástroje pro tvorbu UML 2.1 nabízí možnost výstupu v RTF a HTML formátu. Další výhodou je CASE nástroj pro vytváření zdrojového kódu v 10 programovacích jazycích. Velkým kladem je možnost importu databázového schématu a .NET a Java

struktur. Přijatelná je i cena, která se dle typu licence pohybuje okolo 200 USD. [Sparx Systems]

3.2. Nekomerční nástroje

Nekomerční UML nástroje jsou šířeny především pod licencí GPL. Předností a současně i záporem těchto produktů je menší komplexnost, což znamená i nižší čas na zaučení.

3.2.1. Violet

Violet je jednoduchá aplikace šířená pod licencí GPL. Slouží k tvorbě UML diagramů, ale bez návaznosti na zdrojový kód. Je možné vytvářet diagramy tříd, sekvencí, stavů a objektů. Díky tomuto omezení a nemožnosti generovat kód je tento produkt vhodný pro výuku nebo první návrh aplikace bez další nutnosti kódování. Uložené diagramy lze exportovat do formátu XML, PNG, JPG a PS. Výhodou aplikace je přenositelnost mezi platformami. [Violet UML editor]

3.2.2. Umbrello UML Modeller

Softwarový projekt nazvaný **Umbrello UML Modeller** je určen především pro operační systémy typu Unix a samozřejmě také pro Linux. V roce 2008 byla uvolněna verze pro Windows. Velkou předností tohoto projektu je rozsáhlá podpora pro mnoho programovacích jazyků, včetně PHP 5, SQL, Pythonu i Perlu. Pro některé vytvořené jazykové prvky lze použít i refaktoring, který není mnohdy dostupný ani v drahých

komerčních aplikacích. Vzhledem k tomu, že se jedná o aplikaci napsanou v programovacím jazyce C++, je rychlost odezvy programu vyšší než aplikace Violet programu ArgoUML (tyto aplikace jsou napsány v Javě). [Umbrello UML Modeller]

3.2.3. ArgoUML

ArgoUML je open source projekt podporující tvorbu UML diagramů a je přenosný na libovolnou platformu, vyžaduje však nainstalovanou Javu. Je přeložen do deseti světových jazyků. Práce v ArgoUML s jednotlivými typy UML diagramů je velmi intuitivní, u některých diagramů lze dokonce vytvářet hierarchicky organizované poddiagramy. Diagramy lze uložit do grafického souboru nebo vygenerovat kód v jazycích C++, C# a PHP. Generování kódu jiných programovacích jazyků je možné pomocí přídavných modulů. Nevýhodou je podpora metamodelu UML 1.4 a výběr jen ze sedmi druhů diagramů (tříd, stavů, aktivity, případu užití, nasazení, sekvencí a spolupráce). [Argouml.tigris.org]

3.2.4. DIA

Program **DIA** je šířen pod licencí GPL, je inspirován komerčním programem MS Visio pro zobrazování schémat. Nakreslené diagramy je možné převést do různých grafických formátů, XML nebo vygenerovat zdrojový kód jazyku Python. [DIA a drawing program]

4. Použití UML při návrhu objednávkového systému vybrané firmy

4.1. Představení systému

4.1.1. Představení společnosti

Obchodní firma Zahradnictví Kolář, s. r. o., vznikla v roce 1991 v jihočeském městě Milevsku. Jednalo se o rodinnou firmu, jejímž hlavním předmětem podnikání byla produkce řezaných květin. Tato firma vznikla privatizací městského zahradnictví a společníky se stali čtyři sourozenci. V areálu se nacházely na rozloze 2 500 m^2 skleníky, ve kterých se celoročně pěstovaly řezané květiny. S ohledem na známé komparativní výhody nizozemské produkce květin i tato firma nestačila cenové konkurenci. Od roku 2004 byly podniknuty kroky vedoucí k její záchraně. Jako konečné řešení vyplynul prodej, ke kterému došlo v roce 2007, a to jedinému společníku Ing. Martinu Trávníčkovi. Zároveň byla provedena restrukturalizace a začal se realizovat nový podnikatelský záměr. Přeměna produkce řezaných květin na produkci a prodej okrasných dřevin a trvalek.

Současný název firmy je Zahradnictví Trávníček, spol. s r. o., Milevsko. Základní kapitál činí Kč 140 000,--. Počet stálých zaměstnanců je 10 a firma dále hospodaří v areálu o rozloze 2 hektary. V současné době je sice mírně zisková, ale s ohledem na vysoké náklady na restrukturalizaci a nutnost nemalých investic do velmi zastaralého areálu nemá dostatek hotových finančních prostředků.

Svou roli hraje i hodnota skladovaného materiálu potřebného k výrobě. Řádově se jedná o tisíc druhů rostlin a značné množství podnoží nutných k další produkci.

4.1.2. Motivace k vývoji

Pro každého obchodníka je důležitá spokojenost jeho zákazníků. Pro zvýšení komfortu nakupujících a zlepšení efektivity práce je důležité do nabídky služeb implementovat i objednávkový systém.

V současné době lze na internetových stránkách zmiňované firmy nalézt jen nabídkový katalog, který není skutečným obrazem všech skladových položek. Obsahuje pouze seznam rostlin. Údaje o jejich velikosti a naskladněném množství jsou nedostupné, jelikož není nastavena provázanost mezi skladovou databází a nabídkou rostlin. Pro objednání zboží musí zákazník zatelefonovat, napsat e-mail nebo osobně přijet.

Zavedením objednávkového systému se předejde těmto neefektivním úkonům. Zákazník není omezen otevírací dobou zahradnictví a ušetří i prostředky za cestovní náklady či telefon. Další motivací k vývoji byla možnost spolupráce s již funkční skladovou evidencí a fakturací.

4.1.3. Proces vývoje

Během postupu při plánování realizace objednávkového systému nebyl systematicky sledován žádný specifický proces vývoje. Vzhledem k relativní jednoduchosti a jednoznačnosti problému lze vývoj rozložit dle sjednoceného procesu do několika činností. Každá činnost má navržený časový rámec. Časové hodnoty jsou udávány v tzv. člověkodnech (čd), kdy 1 člověkoden odpovídá 8 hodinám práce.

- Analýza požadavků – 14 čd
- Návrh systému – 20 čd
- Kódování systému – 30 čd
- Testování systému – 14 čd
- Nasazení systému – 7 čd

Celkem vývoj aplikace bude trvat 85 člověkodnů, což při jednom vývojáři je 85 dní, po její ostré nasazení.

4.1.4. Navazující aplikace

Objednávkový systém je nadstavba již fungující skladové evidence a fakturace. Tato evidence umožňuje provádět příjem a výdej zboží, tisk dokladů a zobrazuje přehled o pohybech za určité období. Je nastavena jen na jeden sklad a je možná práce v síti. Datové úložiště skladové evidence je databáze PostgreSQL 8.4 umístěná na stejném počítači jako obslužný program.

4.2. Popis systému

Navrhovaný systém umožní zákazníkovi nejen vybírat z aktuálních skladových položek (propojení se skladovou databází), ale i vytvořit z nich objednávku. Po odeslání objednávky dostane potvrzení i s termínem vyzvednutí zboží. Aplikace bude zprovozněna na webových stránkách firmy.

Uživatel si pro přihlášení do systému musí vytvořit své přihlašovací jméno, heslo a vyplnit údaje v kontaktním formuláři. Tato data budou použita při pozdější komunikaci a tisku dokladů.

Po přihlášení do systému (jméno, heslo) se zobrazí aktuální nabídka zboží. Zboží je možné přiřazovat jednoduchým způsobem do objednávky. Po dokončení výběru zboží se objednávka uzavře a odešle k potvrzení. Poté je nutné vyčkat na potvrzení objednávky s termínem možného vyzvednutí objednaného zboží. Způsob odebírání zboží není záměrně řešen, protože nebyl zadán v požadavcích na systém.

Administrátorovi systému, prodejci, přijde upozornění na podanou objednávku. Přijaté objednávky potvrdí, určí datum expedice a převede objednávku do stavu potvrzená.

Komunikace mezi uživateli a systémem má svůj odraz v databázi (viz příloha č. 4). Veškeré evidence a realizace jsou archivovány v příslušný databázových tabulkách.

4.3. Použité UML diagramy

4.3.1. Diagram tříd

Diagram tříd popisuje použité objekty k implementaci objednávkového systému. Třídy a atributy jsou pro přehlednost popsány i pomocí písmen s diakritikou, pro potřebu generování kódu je nutné diakritiku odstranit.

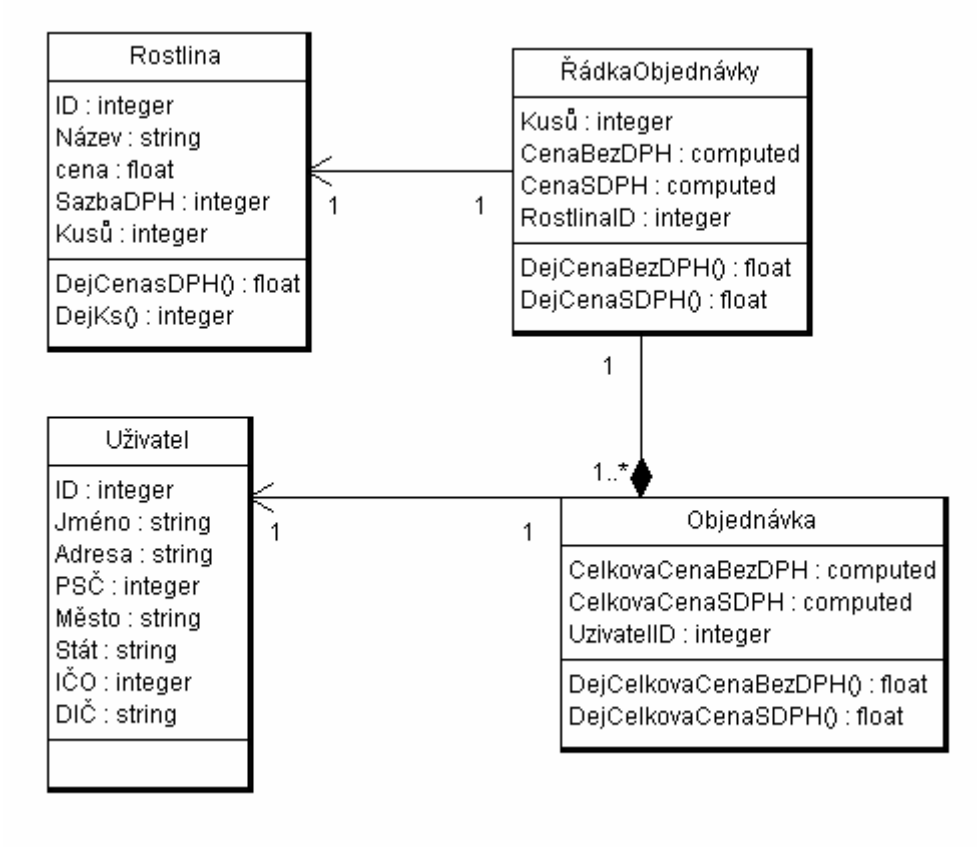
Třída Rostlina svými atributy odpovídá záznamům v databázi o jednotlivých rostlinách. Metody třídy vrací cenu za jeden kus včetně DPH a počet kusů na skladě. Pro každou rostlinu využitou v objednávce se vytvoří nová instance třídy.

Třída ŘádkaObjednávky reprezentuje jednu rostlinu, kterou uživatel vloží do objednávky. Atribut Kusů bude vyplněn zákazníkem, atributy CenaBezDPH a CenaSDPH se dopočtou. K dopočtení těchto atributů slouží metody DejCenaBezDPH a DejCenaSDPH.

Řádky jsou dále agregovány do třídy Objednávka. Z naznačené násobnosti vyplývá, že objednávka se skládá z jednoho a více řádků. Atributy jsou obdobné jako u jednotlivých řádků objednávky, navíc je atribut sloužící k identifikaci odběratele. Metody slouží k dopočítání atributů, které reprezentují cenu za celou objednávku.

Třída uživatel je reprezentací záznamu v databázi. Všechny atributy, kromě ID, musí zákazník vyplnit při prvním přihlášení do systému.

Obrázek č. 16: Diagram tříd objednávkového systému

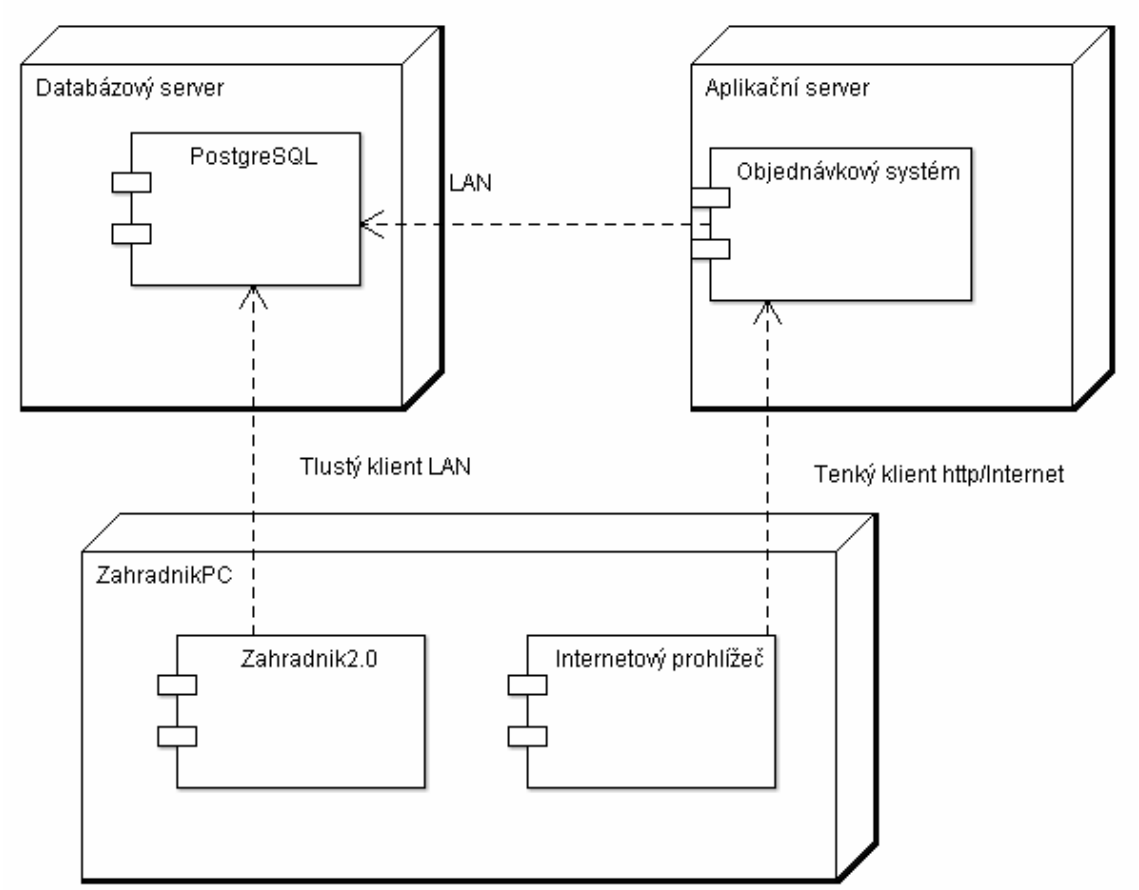


4.3.2. Diagram nasazení

Z diagramu nasazení je patrné, že objednávkový systém navazuje a spolupracuje s dalšími aplikacemi. Nasazení systému proběhne na aplikačním serveru, který zajistí komunikaci systému a databáze s internetovým prohlížečem (tenký klient).

Objednávkový a fakturační systém budou umístěny na stejném počítači, ale komunikace mezi nimi bude probíhat jen prostřednictvím databáze resp. jejích tabulek. Aplikační server poskytuje přístupy do objednávkového systému všem uživatelům.

Obrázek č. 17: Diagram nasazení objednávkového systému

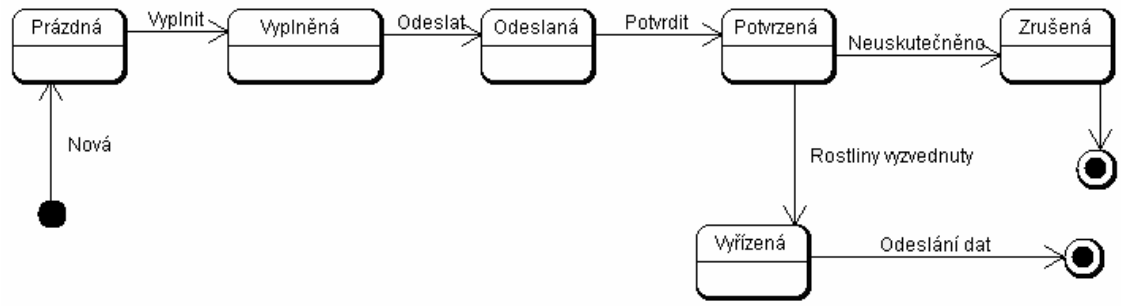


4.3.3. Diagram stavů

Diagram stavů popisuje chování objektu Objednávka v rámci objednávkového systému. Po počátečním pseudostavu je objednávka prázdná. Jakmile ji zákazník začne vyplňovat, přejde do stavu vyplněná. V tomto stavu zůstane, dokud zákazník nedokončí výběr rostlin a neodešle ji k potvrzení prodejci. Prodejce objednávku přijme, tím se ale její stav nezmění. Ke změně dojde až v okamžiku potvrzení. Po potvrzení objednávky dojede k rozvětvení diagramu. Objednávka se buď uskuteční, nebo se z nějakého důvodu nezrealizuje. Při uskutečnění objednávky, když dojde k vyzvednutí rostlin, přejde do stavu vyřízená. Odesláním dat o uskutečněné transakci dojde ke zrušení

objektu. Pokud se objednávka neuskuteční, tak se zruší. Dojde také k zrušení řídícího objektu, ale bez posílání dat databázi.

Obrázek č. 18: Stavový diagram objednávky



4.3.4. Diagram případu užití

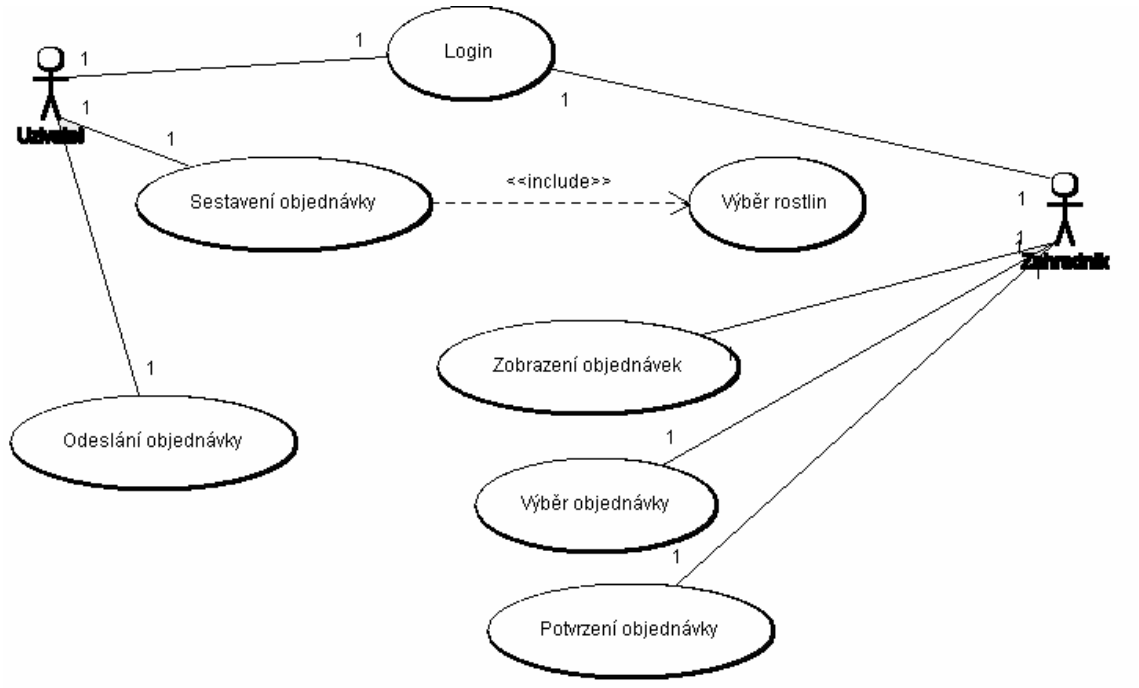
Případy užití objednávkového systému zachycují funkční požadavky na jednotlivé aktéry přítomné v systému. Aktéři či role uživatelů jsou dvě.

- Role uživatel
- Role zahradník

Uživatel je zákazník, který pro vstup do systému musí provést přihlášení (login). Po přihlášení má možnost sestavení objednávky pomocí výběru rostlin. Po sestavení objednávky musí dojít k jejímu odeslání. Tím jsou jeho aktivity vyčerpány.

Zahradník je prodejce a správce systému. Po přihlášení má právo na prohlížení a potvrzení přijatých objednávek. Objednávku potvrdí o jejím výběru a zobrazení. Potvrzení objednávky je sestavení výpisu pro zákazníka s informací o možnosti vyzvednutí objednaného zboží. Tvorba výpisu a jeho odeslání je generováno fakturačním systémem, který není popisován.

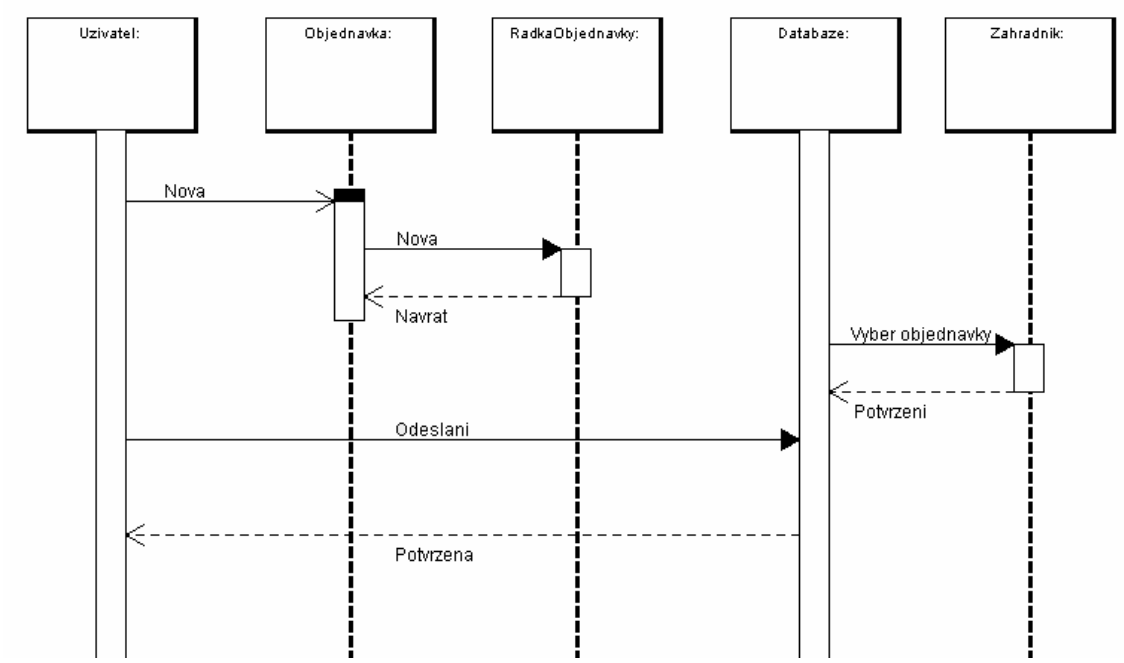
Obrázek č. 19: Případy užití



4.3.5. Diagram sekvencí

Diagram sekvencí znázorňuje chování objednávky při jejím vzniku, vyplňování a odeslání. Zákazník si založí novou objednávku. Do ní vkládá jednotlivé řádky. Je-li objednávka kompletní odešle se na databázi k uložení. Zahradník si ji vyžádá z databáze a potvrdí ji. Uživatel následně obdrží zprávu o termínu dodání. Svislé čáry života naznačují od a do jaké doby se objekt v sekvenci vyskytuje. Sekvence mezi Objednávkou a jednotlivými řádkami se opakuje až do vyplnění Objednávky.

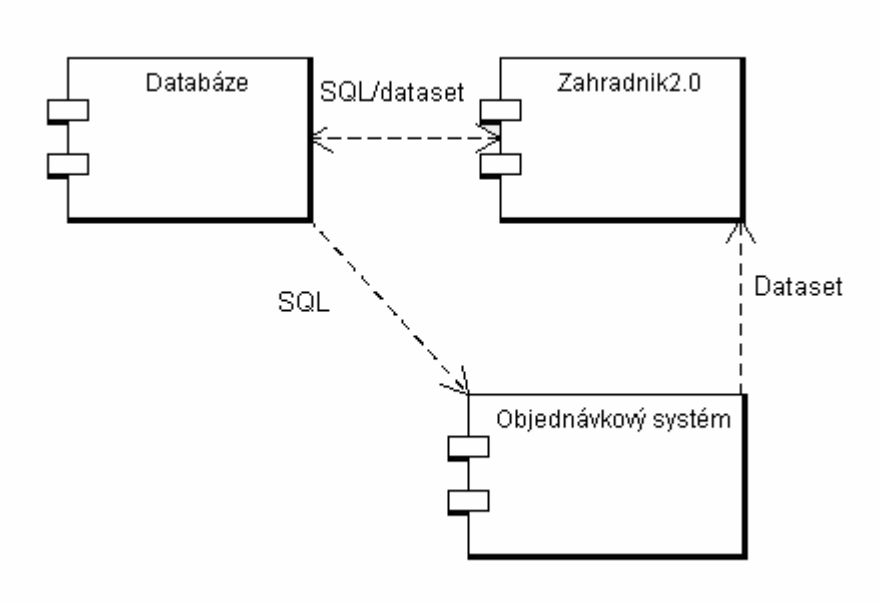
Obrázek č. 20: Diagram sekvencí pro objednávku



4.3.6. Diagram komponent

Diagram komponent znázorňuje jednotlivé části systému a jejich vztahy. Hlavní komponenta je datové úložiště – databáze. S ní komunikuje skladová evidence a fakturace z aplikace Zahradnik 2.0. Objednávkový systém využívá jak dat z databáze, tak i některé funkcionality z fakturační i skladové části nadstavbové aplikace. V diagramu jsou zachyceny i způsoby komunikace mezi jednotlivými komponentami. Jedná se o tok dat ve formě typizovaného objektu dataset a komunikaci v jazyce SQL.

Obrázek č. 21: Diagram komponent v systému pro zahradnictví



5. Volba a popis nástrojů pro tvorbu aplikace

5.1. UML

5.1.1. ArgoUML

Pro vývoj návrhu objednávkového systému byl zvolen program ArgoUML, verze 0.26.2. Tento program poskytuje možnost nakreslení základních UML diagramů. Jeho výhodou je přenositelnost mezi platformami a jednoduché, intuitivní ovládání. Podporuje notaci UML 1.4. Jazyky Java a C++, nejsou pro další vývoj objednávkového systému dostačující, proto je nutné najít externí CASE nástroj pro generování kódu v jiném jazyku.

5.2. CASE nástroj

5.2.1. ArchGenXML

Pro další práci s návrhem je proto vhodné z návrhu nechat vygenerovat zdrojový kód. Tento kód bude použit při dalším kódování systému. Pro generování kódu v jazyce Python slouží externí aplikace ArchGenXML. [ArchGenXML — Plone CMS: Open Source Content Management] Vygenerovaný kód lze využít při dalším kódování aplikace nebo připravit ke spuštění na CMF/Plone serveru.

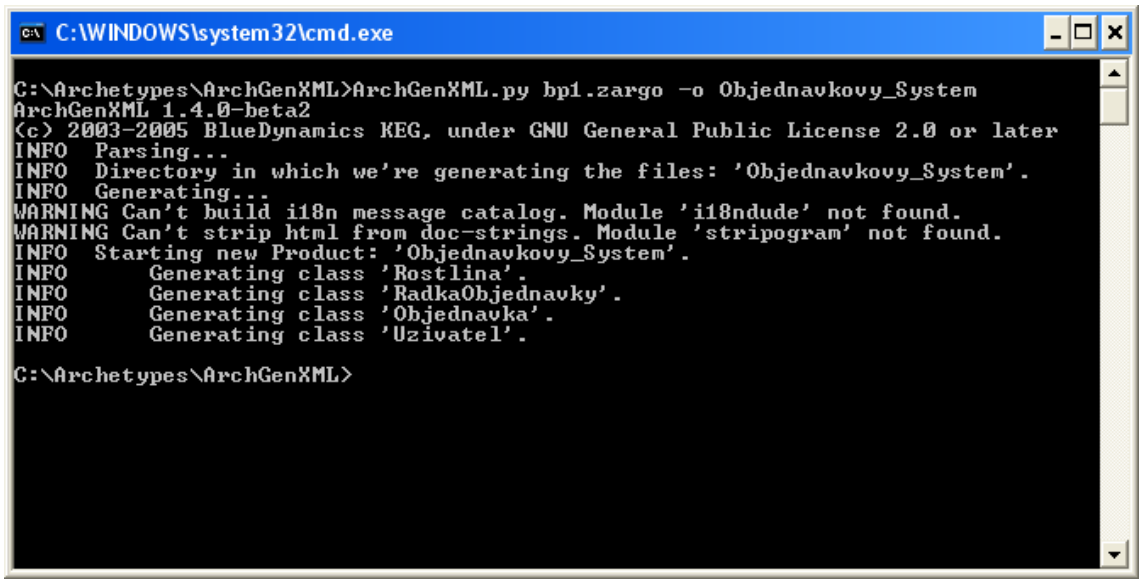
5.2.2. Generování kódu pomocí ArchGenXML

Pro generování kódu je potřeba mít stažen produkt ArchGenXML. Pro jeho obsluhu slouží příkazová řádka. Pro snadnou obsluhu a spuštění CASE nástroje je potřeba mít umístěn ArgoUML produkt v adresáři aplikace ArchGenXML a v příkazovém řádku být na též lokaci. Pro zahájení generování slouží soubor ArchGenXML.py, který se spouští

s parametrem jméno ArgoUML produktu. Další parametr „–o“ slouží k určení názvu adresáře do kterého bude generováno, jak je patrné z obrázku č. 22.

Nástroj vygeneruje atributy a metody každé třídy do zvláštního souboru. Do těla metody je automaticky vygenerován příkaz pass, to znamená, že těla metod je nutné ručně doprogramovat. Dále je nutné ručně nastavit výchozí hodnoty číselných a vypočítávaných proměnných na nulu. Lze to nastavit ručně nebo v programu ArgoUML přes tzv. tagovanou hodnotu. Příklad vygenerovaného kódu třídy je v příloze číslo 2.

Obrázek č. 22: Výstup po vygenerování kódu diagramu tříd



5.3. Aplikační server

5.3.1. Plone/Zope server

Pro zveřejnění této aplikace na webu je možné využít služeb Plone/Zope serveru. [Plone CMS: Open Source Content Management] Výhodou je, že vygenerovaný kód z ArchGenXML lze po dílčích úpravách použít jako standardizovaný Plone produkt. Plone server je svobodný software, který funguje jako systém pro správu obsahu. Je provozován na aplikačním serveru Zope, k administraci je využito webové rozhraní. Přidané funkcionality v Plone serveru jsou přidané jako standardizované Plone produkty (viz příloha číslo 3).

5.3.2. Přidání produktu do Plone portálu

Pro přidání produktu musí být povolen administrátorský přístup do Plone portalu. Adresář s hotovým Plone produktem je nutné umístit do adresáře Plone\Data\Products. Po tomto kroku je nezbytné Plone server restartovat, čímž se aplikace načte a připraví k instalaci. (viz příloha číslo 3) Tím je produkt připraven k použití.

6. Závěr

Cílem bakalářské práce bylo popsat vybrané diagramy sjednoceného modelovacího jazyka UML a přiblížit jejich využití při vývoji objednávkového systému na příkladu konkrétní firmy.

Protože se jedná o velice dynamický obor, bylo nutné čerpat především z aktuálních českých i zahraničních webových stránek. Hlavním důvodem bylo, že trend v oblasti navrhování a vývoje softwaru je velmi rychlý a knižní publikace nepostihují změny dostatečně pružně. S ohledem na aktuálnost byly proto preferovány zahraniční zdroje.

Teoretická část byla zaměřena na představení UML modelů a popis vývojového procesu. V této části byly popsány i možnosti využití UML nejen jako náčrtku, ale i jako programovacího jazyka. Stěžejní částí celé práce bylo představení většiny diagramů náležejících do standardu UML 2.0. Další neméně důležitou kapitolou byly charakteristiky vybraných UML nástrojů, kterým byla věnována větší pozornost.

V praktické části byl popsán vývoj objednávkového systému pro konkrétní firmu, která se zabývá produkcí okrasných dřevin a trvalek. Základní vizí byla skutečnost, že objednávkový systém usnadní a zároveň zrychlí přístup zákazníků k aktuálním skladovým položkám a zvýší tak komfort při objednávání zahradnické produkce. Při vývoji samotné aplikace bylo důležité zvolit vývojový proces a dodržet jeho rámec.

Pro praktické využití byly původně navrženy nástroje ArchGenXML a aplikační Plone/Zope server. Vzhledem k jejich relativní nedostupnosti u poskytovatelů webhostingu a zároveň s ohledem na odmítnutí výše popsané firmy provozovat vlastní webserver, byl projekt ve fázi testování systému pozastaven s ohledem na možnost změny aplikačního serveru. Zmiňované UML diagramy se dají použít i pro vývoj

v jazyce PHP pro aplikační server Apache, který je nejvíce nabízen poskytovateli webhostingu.

Vzhledem k pozastavení vývoje ze strany objednavatele nebyl systém nasazen do plného provozu. Přesto lze konstatovat, že UML modely jsou vhodným nástrojem pro tvorbu a návrh softwarových systémů.

Na závěr je vhodné podotknout, že UML modely neslouží jen jako prostředek pro vývoj aplikací, ale i pro jednoduchou komunikaci mezi programátory. Dalším významným pozitivem je ulehčení práce nasazením CASE nástrojů, které z UML diagramů automaticky vygenerují zdrojový kód a dokumentaci.

Tvorba systémů pomocí modelů je v současné době velice progresivní částí počítačových věd. Vzhledem ke značnému dynamickému vývoji a stále větší nabídce CASE nástrojů je architektura řízená modelem slibnou budoucností při navrhování a při vývoji informačních systémů.

7. Summary

The Unified Modeling Language (UML) is the most-used graphic specification. It shows not only application structure, behavior, and architecture, but also business process and data structure. UML also provides unification of every step of development and integration from business modeling, through architectural and application modeling, to development, deployment, maintenance, and evolution.

The first part of my thesis involves the description of UML and its usage in software development process. The development process contains several typical parts. It starts with system analysis, continues to encoding the program and finishes with testing and mounting the software for commercial use. In each step of the process is UML very helpful. UML helps to inscribe the problem and make the communication in development group easier. The big advantage of UML is the opportunity of using computer-aided software engineering for generating source code from UML models. It saves a lot of work with manual typing code, making documentation and deploying the application.

In practical part there is described the application of UML on example on development of order system for given company. The company is directed on production and distribution of woody and ornamentals. The order system increases the comfort of costumers by making an order. The UML diagrams shows behavior and basic structure of the system. There is also written how to use ArchGenXML CASE software for generating Python source code and deploying the application on Plone/Zope server.

Creating software systems is these days very progressive part of computer sciences. Thanks to dynamical progress of developing UML's annotation and standard is model driven architecture promising future of developing computer systems.

8. Přehled použitých zdrojů

[1] ArchGenXML — Plone CMS: Open Source Content Management : ArchGenXML [online]. 2010 [cit. 2010-05-06]. Dostupné z WWW: <http://plone.org/products/archgenxml>.

[2] ARLOW, Jim; NEUSTADT, Ila. UML2 a unifikovaný proces vývoje : Průvodce analýzou a návrhem objektově orientovaného softwaru. 2. Brno : Computer Press, 2008. Co je vlastně UML?, 563 s. ISBN 987-80-251-1503-9.

[3] Argouml.tigris.org [online]. 2009 [cit. 2010-05-06]. Dostupné z WWW: <http://argouml.tigris.org/>.

[4] Artisan Studio [online]. 2010 [cit. 2010-04-16]. Dostupné z WWW: <http://www.artisansoftwaretools.com/products/artisan-studio/>.

[5] BENEŠ, Michal. Přehled OO notací a metodik - Diagramy UML [online]. 16.6.2005 [cit. 2010-04-18]. Diagramy UML. Dostupné z WWW: <http://objekty.vse.cz/Objekty/MetodikyANotace-UMLDiagramy>.

[6] BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar . The Unified Modeling Language User Guide : Addison-Wesley Object Technology Series. 2. Reading, Massachusetts : Addison-Wesley Professional, 2005. 496 s. ISBN 0321267974.

[7] CASE nástroje [online]. 2009 [cit. 2010-05-01]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/CASE_nástroje>.

[8] DIA a drawing program [online]. 2009 [cit. 2010-05-06]. Dostupné z WWW: <http://projects.gnome.org/dia/>.

[9] DIA's screenshots [online]. 2009 [cit. 2010-04-20]. Screenshots. Dostupné z WWW: <http://projects.gnome.org/dia/scrshot.html>.

[10] FOWLER, Martin. Destilované UML. 3. Praha : Grada Publishing, 2009., 163 s. ISBN 978-80-247-2062-3

[11] CHARNEY, Reg. Linux Journal [online]. 1.6.2005 [cit. 2010-04-19]. Programming Tools: UML Tools. Dostupné z WWW: <http://www.linuxjournal.com/article/8334>.

[12] IBM - Rational Rose Enterprise [online]. 2010 [cit. 2010-04-16]. Dostupné z WWW: <http://www-01.ibm.com/software/awdtools/developer/rose/enterprise /index.html>.

[13] MDA [online]. 2009, 12.8.2009 [cit. 2010-05-01]. Dostupné z WWW: <http://www.omg.org/mda/>.

[14] OMG Modeling and metadata specifications [online]. 2010, 5.4.2010 [cit. 2010-04-17]. Catalog of OMG Modeling and Metadata Specifications. Dostupné z WWW: <http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML>.

[15] Plone CMS: Open Source Content Management : Plone [online]. 2010 [cit. 2010-05-06]. Dostupné z WWW: <http://plone.org/>.

[16] Poseidon for UML [online]. 2010 [cit. 2010-04-16]. Dostupné z WWW: <http://www.gentleware.com/uml-software-pe.html>.

[17] Redhat.com [online]. 2010 [cit. 2010-05-06]. Dostupné z WWW: <http://www.redhat.com/metamatrix/>.

[18] Redhat [online]. 2010 [cit. 2010-05-06]. Dostupné z WWW: <http://www.redhat.com>.

[19] Software architecture design from Borland : Borland Together [online]. 2010 [cit. 2010-05-06]. Dostupné z WWW: <http://www.borland.com/us/products/together/index.html>.

[20] Sparx Systems [online]. 2010 [cit. 2010-05-02]. Dostupné z WWW: <http://www.sparxsystems.com/>.

[21] Umbrello UML Modeller [online]. 2008 [cit. 2010-05-06]. Dostupné z WWW: <http://uml.sourceforge.net/>.

[22] UML 2 diagramming, OO software modeling, Source code engineering Tool MagicDraw UML from No Magic [online]. 26.3.2010 [cit. 2010-04-20]. Screenshots. Dostupné z WWW: <http://www.magicdraw.com/screen_shots>.

[23] UML Modeling Tool : MagicDraw [online]. 2010 [cit. 2010-05-02]. Dostupné z WWW: <http://www.magicdraw.com/>.

[24] UML Ressource Page [online]. 2010 [cit. 2010-05-06]. Dostupné z WWW: <http://www.uml.org/>.

[25] Violet UML editor: easy to use, completely free [online]. 2009 [cit. 2010-04-20]. Quick tour. Dostupné z WWW: <http://alexdp.free.fr /violetumleditor/page.php?id=en:tour>.

9. Seznam obrázků

10. Seznam příloh

Příloha č. 1: Screenshoty UML nástrojů

Příloha č. 2: Vygenerovaný kód třídy s dopsaným kódem metod

Příloha č. 3: Prostředí Plone/Zope serveru

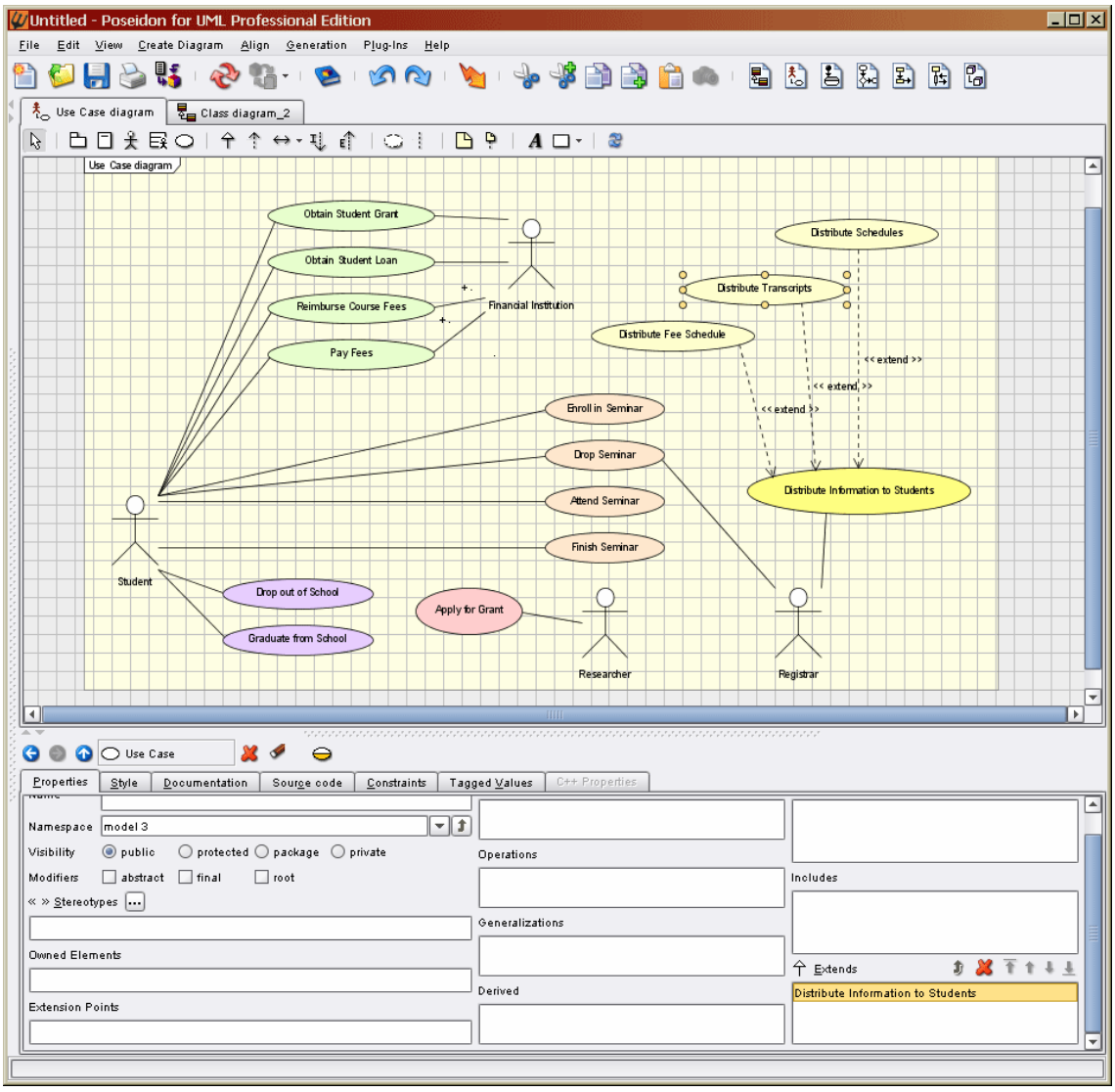
Příloha č. 4: Náhled databáze

11. Přílohy

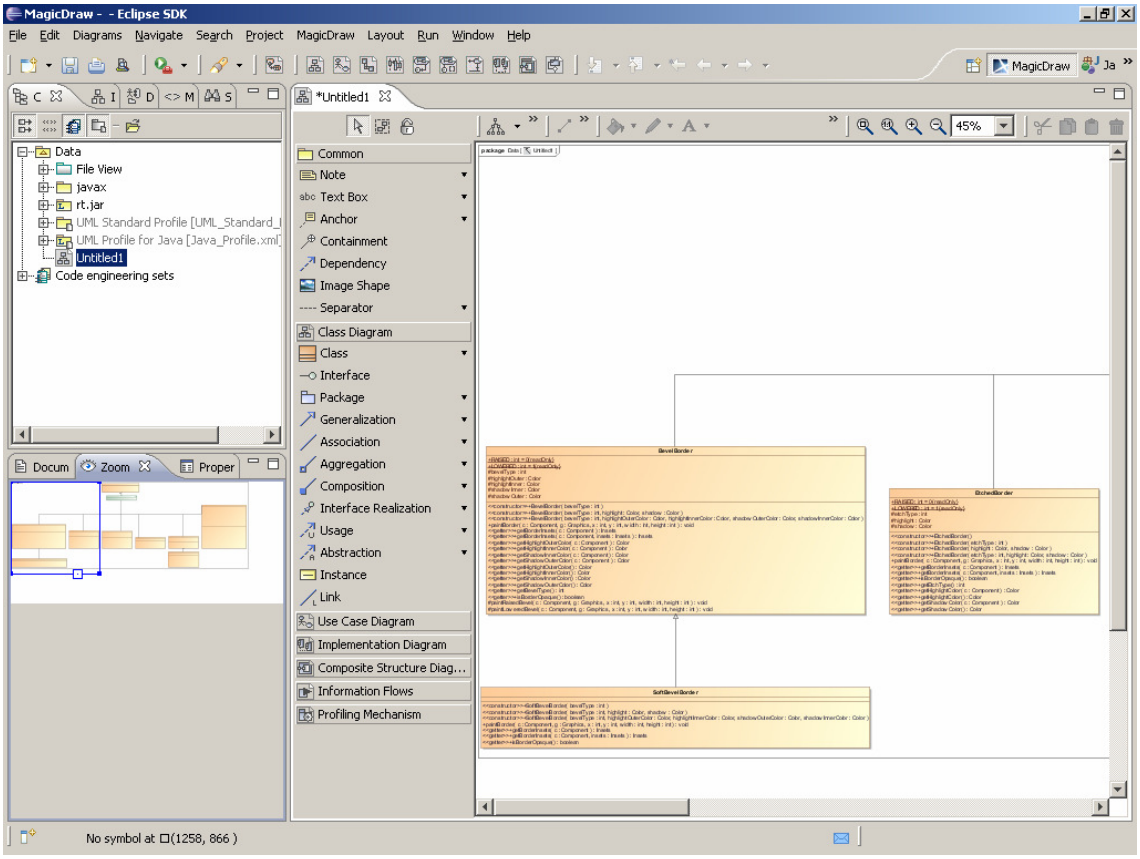
Příloha č. 1

Screenshoty UML nástrojů

- **Poseidon for UML**

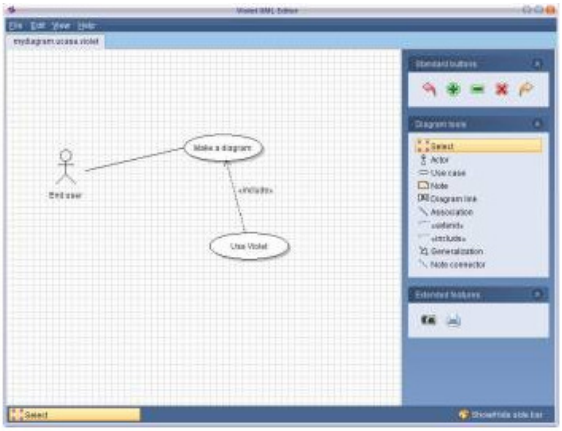


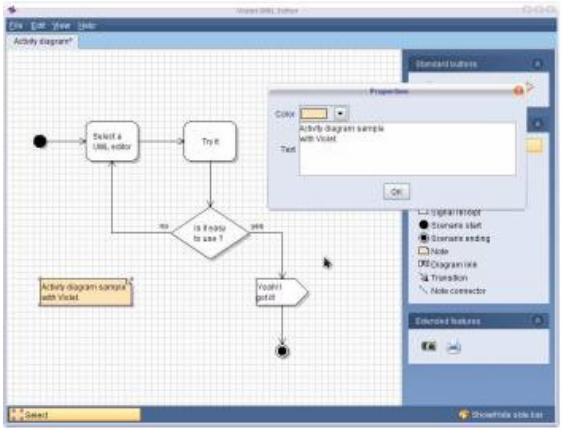
- **Magic Draw**

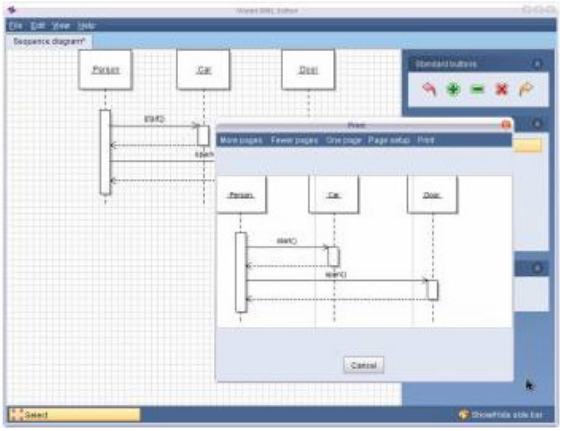

MagicDraw - - Eclipse SDK
File Edit Diagrams Navigate Search Project MagicDraw Layout Run Window Help
MagicDraw
Data
File View
javax
rt.jar
UML Standard Profile [UML_Standard_
UML Profile for Java [Java_Profile.xml]
Untitled1
Code engineering sets
Docum
Zoom
Proper
*Untitled1
45%
Common
Note
Text Box
Anchor
Containment
Dependency
Image Shape
Separator
Class Diagram
Class
Interface
Package
Generalization
Association
Aggregation
Composition
Interface Realization
Usage
Abstraction
Instance
Link
Use Case Diagram
Implementation Diagram
Composite Structure Diag...
Information Flows
Profiling Mechanism
No symbol at □(1258, 866)

- **Violet**

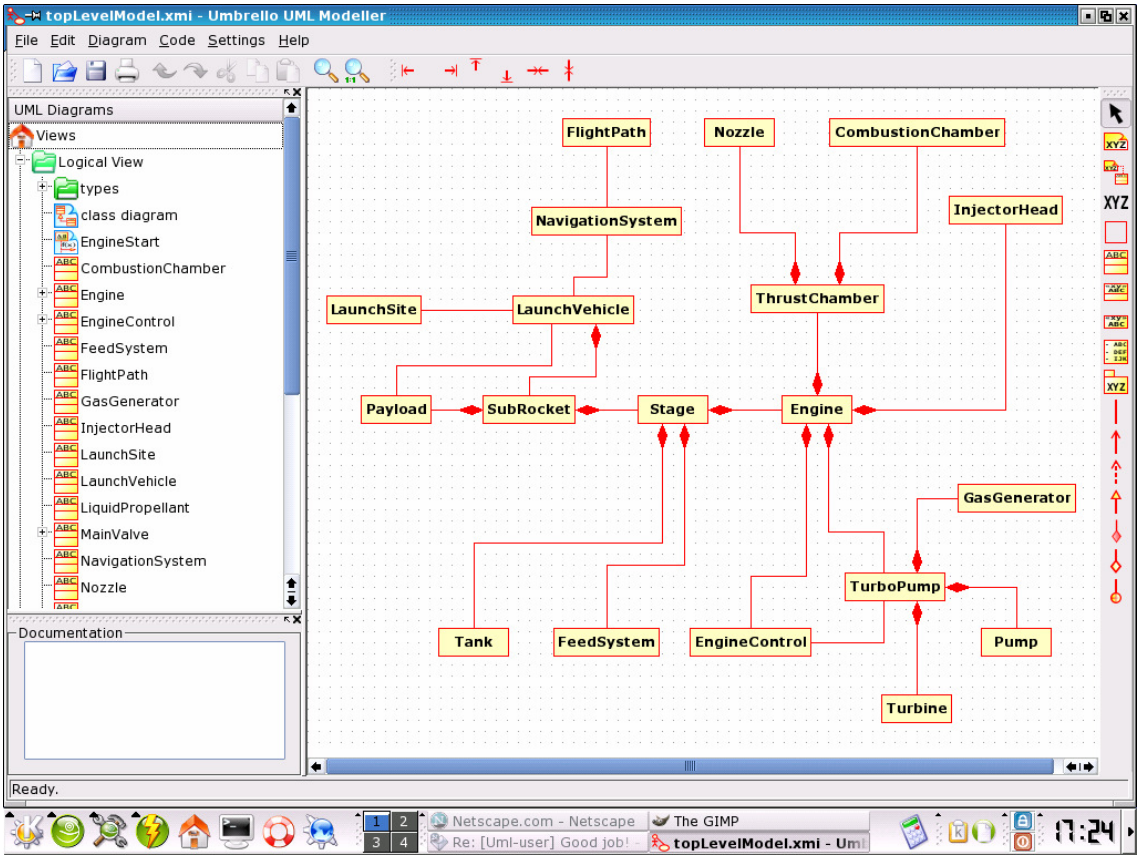

new
recent files
use case diagram
class diagram
object diagram
state diagram
activity diagram
sequence diagram
mydiagram.ucase.violet



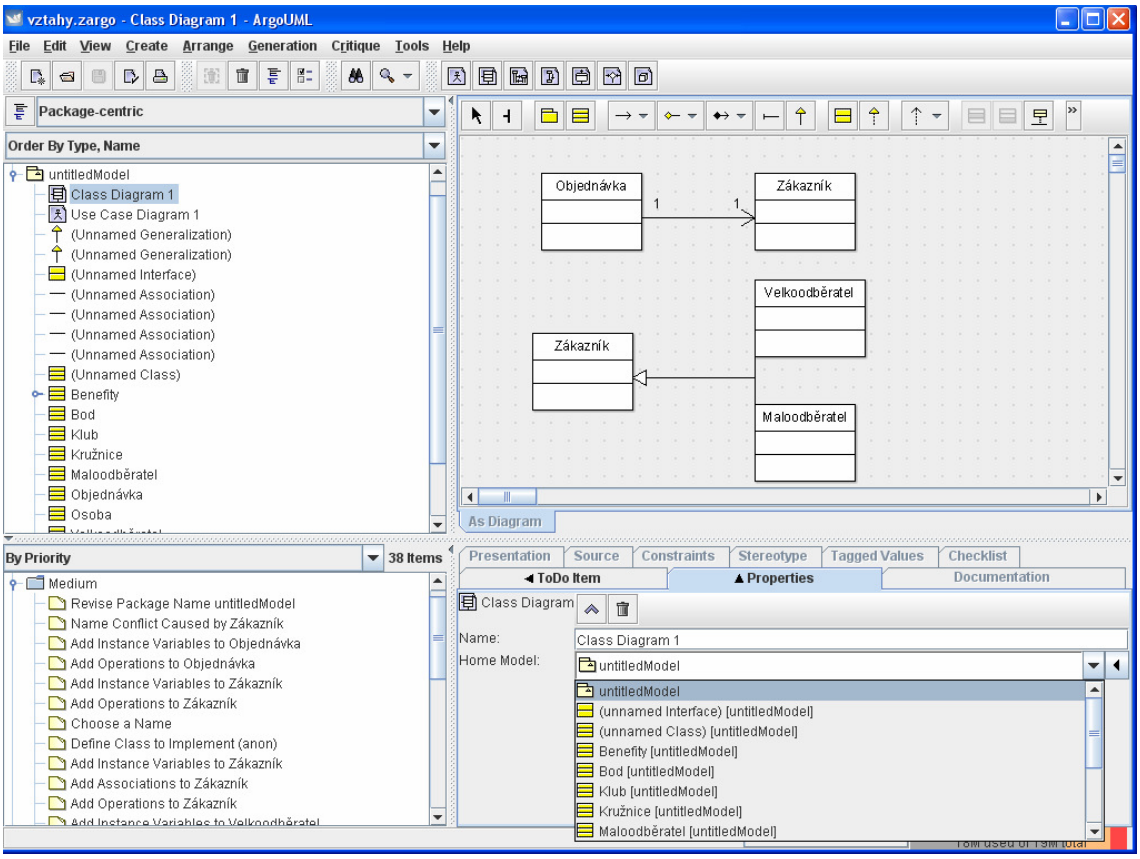




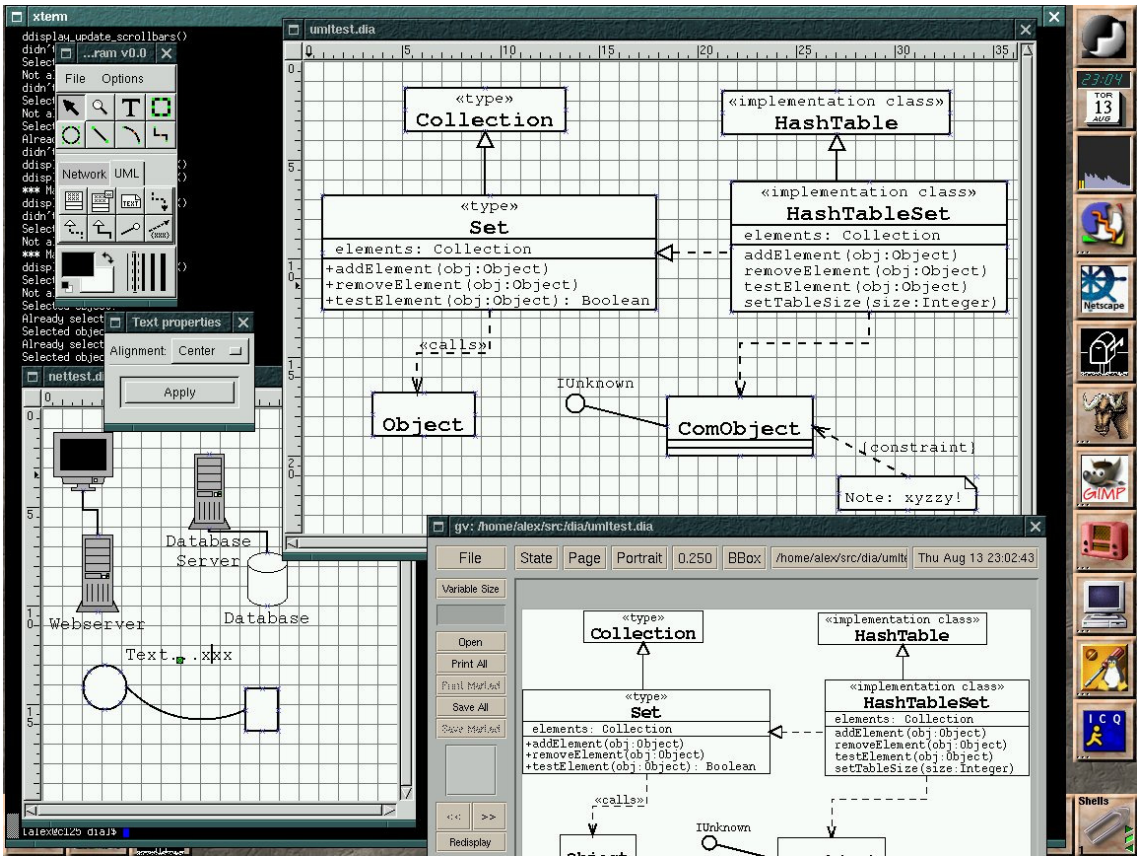
- **Umbrello UML Modeller**



- **ArgoUML**


vztahy.zargo - Class Diagram 1 - ArgoUML
File Edit View Create Arrange Generation Critique Tools Help
Package-centric
Order By Type, Name
untitledModel
Class Diagram 1
Use Case Diagram 1
(Unnamed Generalization)
(Unnamed Generalization)
(Unnamed Interface)
(Unnamed Association)
(Unnamed Association)
(Unnamed Association)
(Unnamed Association)
(Unnamed Class)
Benefity
Bod
Klub
Kružnice
Maloodběratel
Objednávka
Osoba
Objednávka
1
1
Zákazník
Velkoodběratel
Zákazník
Maloodběratel
As Diagram
By Priority
38 Items
Medium
Revise Package Name untitledModel
Name Conflict Caused by Zákazník
Add Instance Variables to Objednávka
Add Operations to Objednávka
Add Instance Variables to Zákazník
Add Operations to Zákazník
Choose a Name
Define Class to Implement (anon)
Add Instance Variables to Zákazník
Add Associations to Zákazník
Add Operations to Zákazník
Presentation
Source
Constraints
Stereotype
Tagged Values
Checklist
ToDo Item
Properties
Documentation
Class Diagram
Name:
Class Diagram 1
Home Model:
untitledModel
untitledModel
(unnamed Interface) [untitledModel]
(unnamed Class) [untitledModel]
Benefity [untitledModel]
Bod [untitledModel]
Klub [untitledModel]
Kružnice [untitledModel]
Maloodběratel [untitledModel]

- **DIA**


xterm
umltest.dia
«type»
Collection
«implementation class»
HashTable
«type»
Set
elements: Collection
+addElement(obj:Object)
+removeElement(obj:Object)
+testElement(obj:Object): Boolean
«implementation class»
HashTableSet
elements: Collection
addElement(obj:Object)
removeElement(obj:Object)
testElement(obj:Object)
setTableSize(size:Integer)
«calls»
Object
IUnknown
ComObject
{constraint}
Note: xyzzy!
Text
Database
Server
Webserver
Database
Text..xxx
Text properties
Alignment: Center
Apply
gv: /home/alex/src/dia/umltest.dia
File
State
Page
Portrait
0.250
BBox
Thu Aug 13 23:02:43
Variable Size
Open
Print All
Save All
Redisplay
Shells

Příloha č. 2

Vygenerovaný kód třídy s dopsaným kódem metod

```
# File: RadkaObjednavky.py
#
# Copyright (c) 2010 by ['']
# Generator: ArchGenXML Version 1.4.0-beta2 http://sf.net/projects/archetypes/
#
# GNU General Public Licence (GPL)
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
# This program is distributed in the hope that it will be useful, but WITHOUT
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
# FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
# details.
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307  USA
#
__author__  = ''' <>'''
__docformat__ = 'plaintext'

from AccessControl import ClassSecurityInfo
from Products.Archetypes.atapi import *


from Products.BP.config import *
##code-section module-header #fill in your manual code here
##/code-section module-header

schema=Schema((
   IntegerField('Kusu',
      widget=IntegerWidget(
         label='Kusu',
         label_msgid='BP_label_Kusu',
         description_msgid='BP_help_Kusu',
         i18n_domain='BP',
      )
   ),

   ComputedField('CenaBezDPH',
      widget=ComputedWidget(
         label='Cenabezdph',
         label_msgid='BP_label_CenaBezDPH',
         description_msgid='BP_help_CenaBezDPH',
         i18n_domain='BP',
```

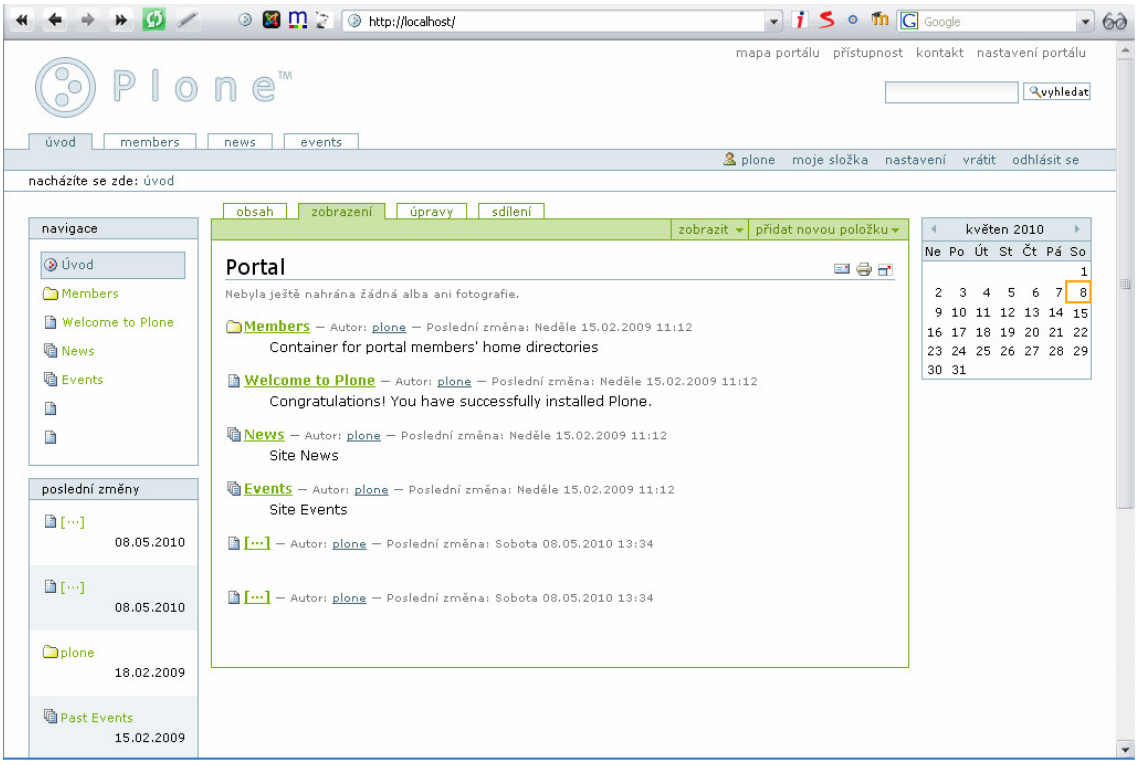
```
        )
    ),

    ComputedField('CenaSDPH',
        widget=ComputedWidget(
            label='Cenasdph',
            label_msgid='BP_label_CenaSDPH',
            description_msgid='BP_help_CenaSDPH',
            i18n_domain='BP',
        )
    ),

    IntegerField('RostlinaID',
        widget=IntegerWidget(
            label='Rostlinaid',
            label_msgid='BP_label_RostlinaID',
            description_msgid='BP_help_RostlinaID',
            i18n_domain='BP',
        )
    ),

    ReferenceField('rostlinas',
        widget=ReferenceWidget(
            label='Rostlinas',
            label_msgid='BP_label_rostlinas',
            description_msgid='BP_help_rostlinas',
            i18n_domain='BP',
        ),
        allowed_types=('Rostlina',),
        multiValued=0,
        relationship='radkaobjednavkys_rostlinas'
    ),

),
)

##code-section after-schema #fill in your manual code here
##/code-section after-schema

class RadkaObjednavky(BaseFolder):
    security = ClassSecurityInfo()
    __implements__ = (getattr(BaseFolder,'__implements__',()),)


    # This name appears in the 'add' box
    archetype_name             = 'RadkaObjednavky'

    meta_type                  = 'RadkaObjednavky'
    portal_type                = 'RadkaObjednavky'
    allowed_content_types      = ['Objednavka']
    filter_content_types       = 1
    global_allow               = 0
    allow_discussion           = 0
    #content_icon              = 'RadkaObjednavky.gif'
    immediate_view             = 'base_view'
```

```
    default_view              = 'base_view'
    typeDescription           = "RadkaObjednavky"
    typeDescMsgId             = 'description_edit_radkaobjednavky'

    schema = BaseFolderSchema + \
             schema

    ##code-section class-header #fill in your manual code here
    ##/code-section class-header

    #Methods

    security.declarePublic('DejCenaBezDPH')
    def DejCenaBezDPH(self):
        """

        """

        return (Rostlina.cena*Ks)

    security.declarePublic('DejCenaSDPH')
    def DejCenaSDPH(self):
        """

        """

        return (DejCenaBezDPH*(1+(Rostlina.SazbaDPH)/100))

registerType(RadkaObjednavky,PROJECTNAME)
# end of class RadkaObjednavky

##code-section module-footer #fill in your manual code here
##/code-section module-footer
```

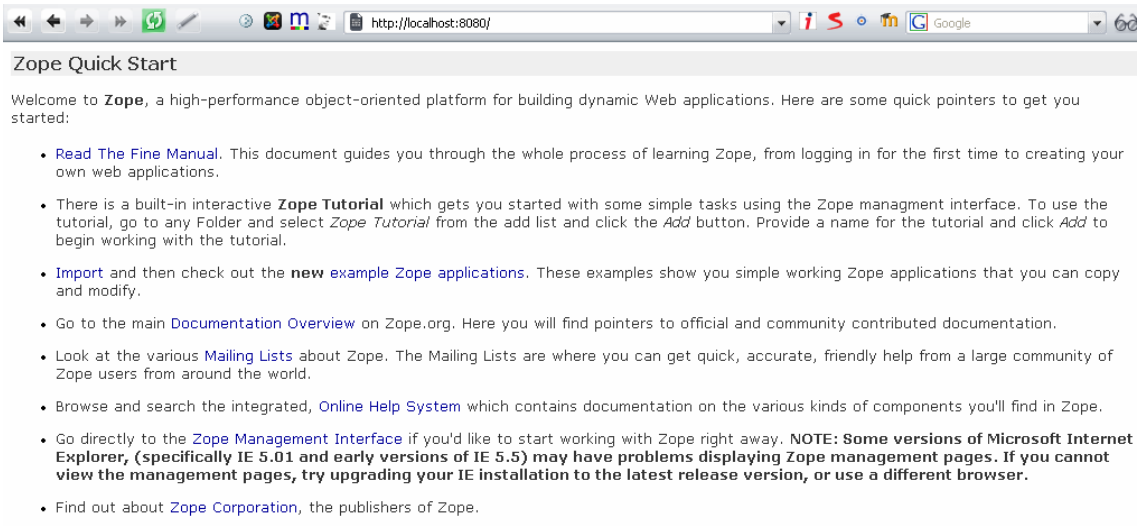
Příloha č. 3

Prostředí Plone/Zope serveru

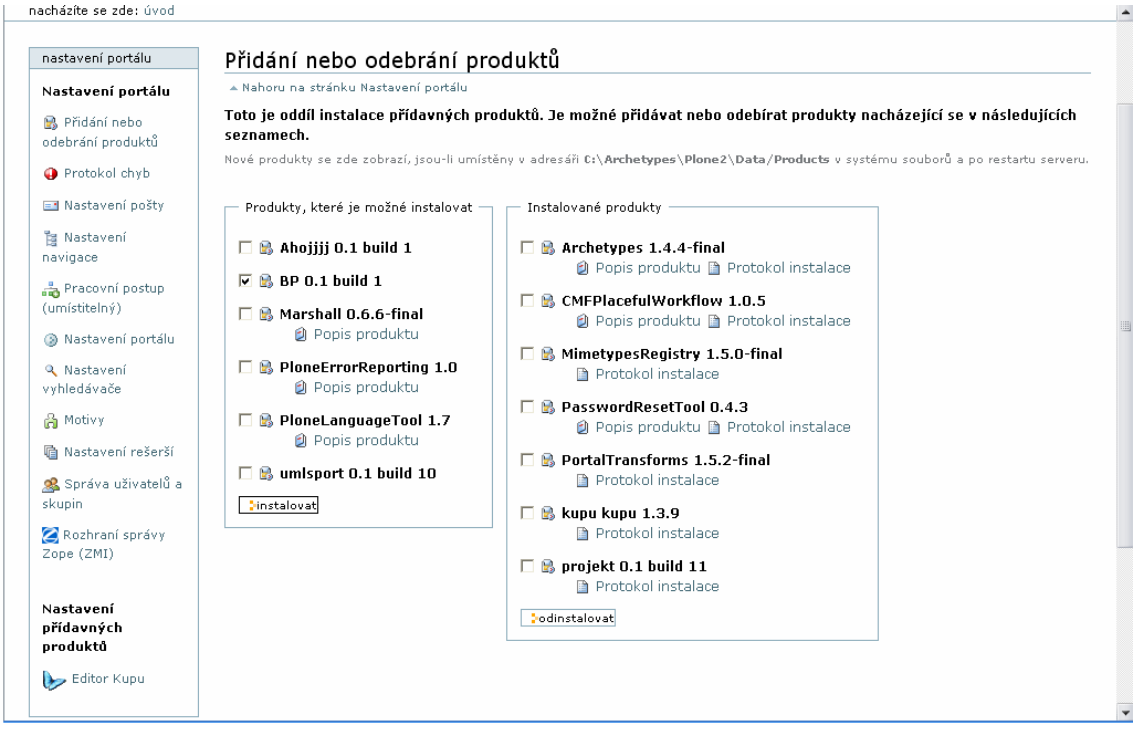
- **Prostředí Plone**



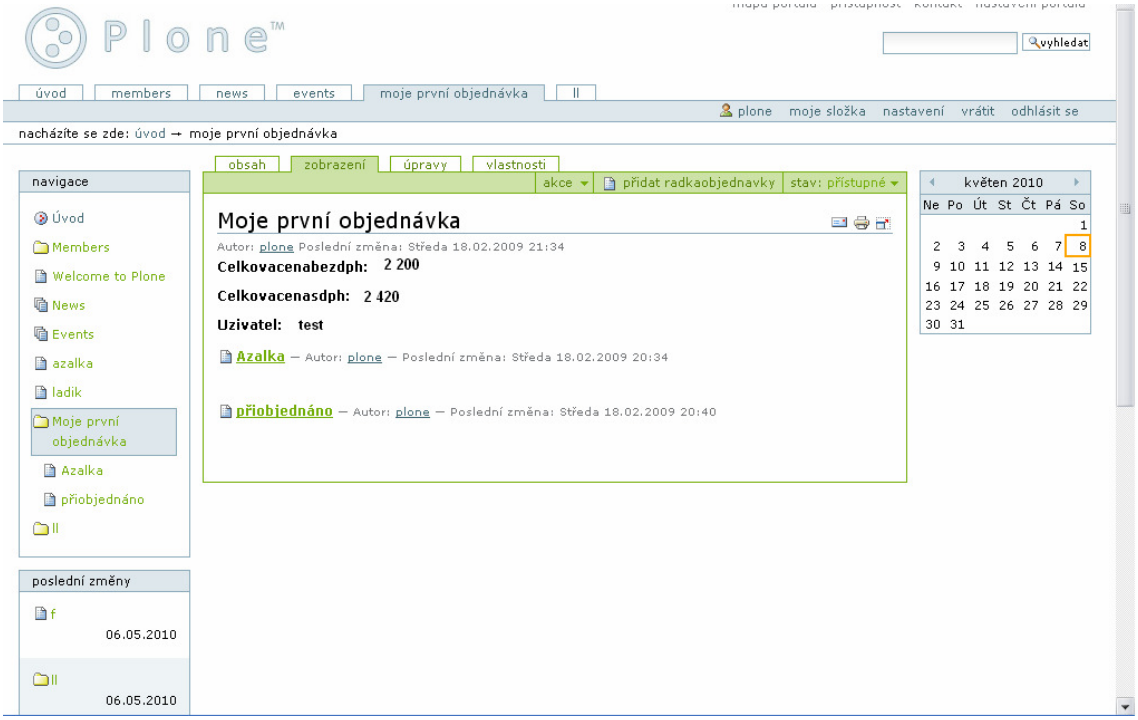
- **Prostředí Zope administrace**



- **Instalace produktu**



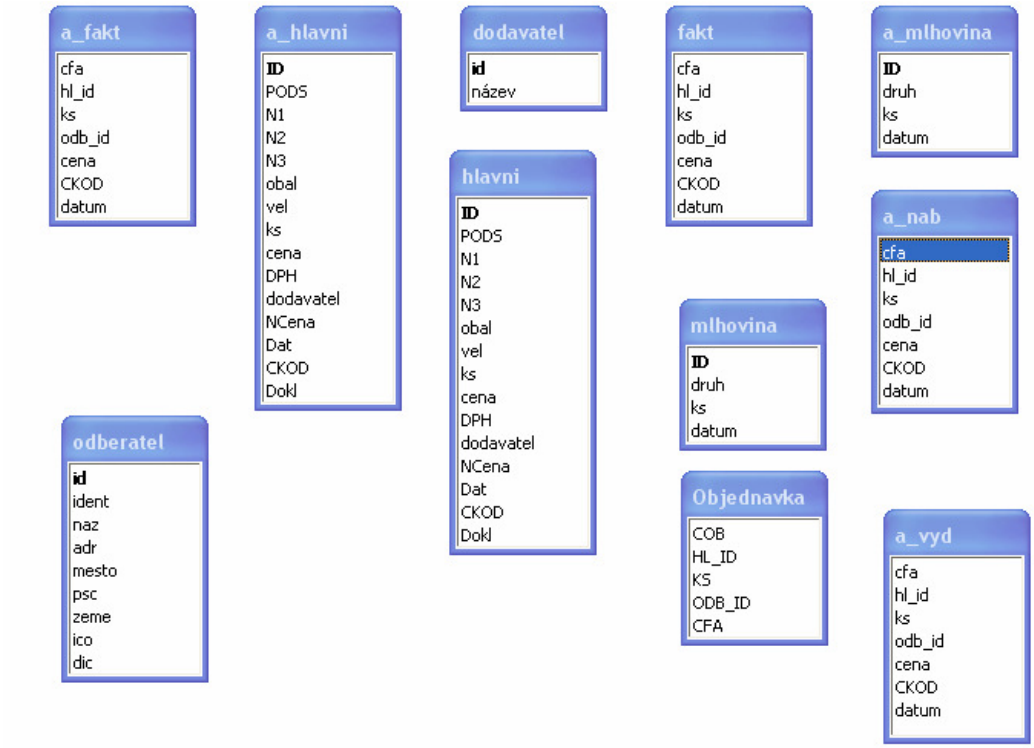
- **Náhled při testování aplikace**



Příloha č. 4

Náhled databáze

- **Zobrazení tabulek databáze**



- **SQL příkazy použité v objednávkovém systému**

Select * from hlavni

Výběr všech rostlin na skladě

Insert into Objednavka(COB,HL_ID,KS,ODB_ID,CFA) values (‘’,’’,’’,’’,’’)

Uložení objednávky

Select * from odberatel where ident=’’

Výběr odběratele