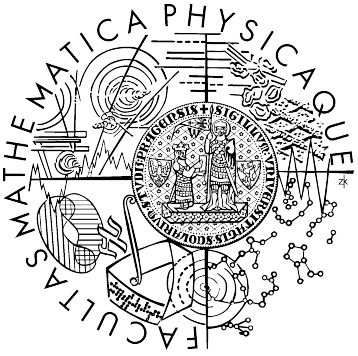
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martin Hlavatý

Softwarové řešení pro firmy zabývající se nákupem a prodejem

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Ing. Lubomír Bulej

Studijní program: Informatika, Programování

2006

Rád bych poděkoval svému vedoucímu, Ing. Lubomíru Bulejovi, za cenné rady a připomínky k této práci.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 22.5.2006 Martin Hlavatý

Obsah

Název práce: Softwarové řešení pro firmy zabývající se nákupem a prodejem
Autor: Martin Hlavatý
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: Ing. Lubomír Bulej
e-mail vedoucího: Lubomir.Bulej@mff.cuni.cz

Abstrakt: Tato bakalářská práce pojednává o skladovém systému a internetovém obchodu pro malou až středně velkou firmu podnikající v oblasti nákupu a prodeje zboží. Zabývá se potřebami těchto firem, popisuje nejčastěji prováděné operace a jejich aktéry. Z analýzy potřeb, operací a aktérů poté vychází návrh softwarového řešení a jeho implementace. Navržené řešení je postaveno na třívrstvé architektuře s využitím webových služeb. Implementace je vytvořena převážně v jazycích Java a PHP, je nezávislá na platformě, snadno spravovatelná (díky přidělování oprávnění prostřednictvím systému skupin a rolí) a rozšiřitelná.

Klíčová slova: skladový systém, internetový obchod, webové služby

Title: Solution for Small and Medium-Sized Retail Companies

Author: Martin Hlavatý
Department: Department of Software Engineering
Supervisor: Ing. Lubomír Bulej
Supervisor's e-mail address: Lubomir.Bulej@mff.cuni.cz

Abstract: The aim of this thesis is to describe a warehouse management system and an Internet shopping website suitable for a small to medium-sized retailers. The thesis analyzes the needs of such companies, the most commonly performed operations and the persons required to carry them out. Based the results of the analysis, a software solution and its implementation is proposed. The suggested solution is built on a three-tier architecture using web services and implemented mostly in the Java and PHP languages. It is platform-independent, open-ended, and easily manageable (due to a group- and role-based authorization system)

Keywords: warehouse management system, e-shop, web services

1. Úvod

Není to tak dávno, kdy symbolem inženýra bylo logaritmické pravítko. Dnes již tento nástroj upadl v zapomnění. Stejně jako mnoho jiných nástrojů a zařízení nepřežil nástup osobních počítačů (a příručních kalkulaček). Počítače se staly nedílnou součástí našich životů a nadobro je změnily. Ovšem aby to byly změny k lepšímu, musíme umět využít jejich potenciál a nebát se jich. Velké firmy si to uvědomily již dávno a pořídily si rozsáhlé informační systémy, jejichž služeb umějí naplno využít. I středně velké podniky (v naprosté většině) již možností počítačů využívají. Ovšem u malých firem a zejména drobných podnikatelů a živnostníků je situace spíše opačná. Buď počítače využívají značně neefektivně, či dokonce vůbec.

Tato práce je věnována právě malým firmám, analyzuje jejich potřeby (z hlediska informačních systémů) a snaží se najít řešení jejich specifických problémů. Protože existuje obrovské množství takovýchto firem podnikajících v různých oborech a protože jejich potřeby a požadavky jsou velmi různorodé a občas dokonce protichůdné, zaměřil jsem se výhradně na firmy zabývající se nákupem a prodejem. Pod tímto formálním názvem si můžete představit například vaše oblíbené železářství, počítačovou prodejnu či obchod se starožitným nábytkem.

Na závěr je ještě nutné poněkud upřesnit termín malá firma. Pro účely této práce můžeme firmy rozdělit do následujících kategorií:

1) Živnostníci a drobní podnikatelé – 0 až 1 zaměstnanec
2) Malé firmy – 2 až 20 zaměstnanců
3) Středně velké firmy – 21 až 500 zaměstnanců
4) Velké firmy – více než 500 zaměstnanců

Dále se budeme zabývat hlavně druhou kategorií, ovšem občas padne zmínka i o první či třetí.

1.1. Problematika malých firem

Nejdůležitějším aspektem, který je třeba brát v úvahu, je množství finančních prostředků, které si firma může dovolit investovat do IT. Tyto prostředky jsou u malých (a většiny středních) firem velice omezené, a proto je bezpodmínečně nutné minimalizovat náklady na pořízení a provoz informačního systému. Tyto náklady zahrnují: cenu hardware a potřebného software (operační systém, databáze a další nezbytné programy), školení zaměstnanců, mzdy administrátorů, náklady na snížení bezpečnostních rizik atd.

Druhým problémem je nízká počítačová gramotnost zaměstnanců. Vzhledem k omezenému rozpočtu jsou mzdy v malých firmách zpravidla nižší než ve větších firmách podnikajících ve stejném oboru, a proto i kvalifikace zaměstnanců bývá nižší. Je tedy nutné aby software byl uživatelsky přívětivý a intuitivní, což souvisí i s výše zmíněnými náklady na školení zaměstnanců.

Dále je zapotřebí si uvědomit, že vyhrazení byť i jediného zaměstnance pouze pro administraci skladového systému může být pro řadu firem značně problematické. Proto je nutné správu maximálně zjednodušit, aby bylo možné ji provádět jako vedlejší činnost.

Neméně důležitá je rovněž otázka úprav systému. Malá firma má samozřejmě mnohem větší prostor k růstu než firma velká a její informační systém se tomuto faktu musí přizpůsobit. Měl by tedy být snadno rozšiřitelný a modifikovatelný. Malé firmy rovněž musejí mnohem pružněji reagovat na změny v požadavcích trhu a jsou více ovlivněny změnami legislativy (viz. například schválení zákona č. 215/2005 Sb., o registračních pokladnách), s čímž souvisí požadavek na možnost program jednoduše upravovat.

Zamysleme se ještě nad otázkou uskladnění zboží, kterou musí řešit každá firma prodávající nějaké zboží. Malé firmy typicky nedisponují velkými skladovými prostorami, ovšem nezřídka mají takovýchto prostor vícero na různých místech. To znamená, že skladový systém musí počítat s existencí více než jednoho skladu (což některé operace výrazně komplikuje).

Rovněž nelze opomenout průměrné stáří počítačů a jejich výkon. Zatímco doporučená doba obnovy PC jsou 3 roky, kvůli úspoře nákladů je tento interval v malých firmách více než dvojnásobný. Z toho nám vyplývá další požadavek a sice rozumná hardwarová náročnost (schopnost běžet i na starších strojích).

Nakonec si shrneme požadavky, které byly vysloveny v předchozích odstavcích:

- Nízké náklady na pořízení a údržbu
- Hardwarová nenáročnost (souvisí s prvním bodem)
- Jednoduchá administrace
- Uživatelská přívětivost a intuitivní ovládání
- Podpora vícero skladů
- Možnost jednoduše provádět úpravy

1.2. Cíle práce

1. Analyzovat softwarové potřeby malé až středně velké firmy
2. Navrhnout řešení, které by tyto potřeby v maximální možné míře uspokojilo
3. Toto řešení implementovat
4. Srovnat vytvořenou implementaci s již dostupnými produkty

1.3. Struktura práce

Kapitola číslo 2 (*Analýza problému)* obsahuje popis aktérů, operací a procesů ve firmě. 3. kapitola (*Návrh řešení)* se zabývá rozdělením programu do částí, návrhem jeho architektury a datového modelu. Kapitola číslo 4 (*Implementace prostředí*) popisuje programy třetích stran, které implementace potřebuje ke svému fungování, a jejich výhody a nevýhody ve srovnání s konkurenčními produkty. V 5. kapitole (*Implementace systému)* je popsána implementace jednotlivých částí navrženého řešení a jsou rozebrány možnosti upravování vytvořeného programu. Kapitola číslo 6 (*Srovnání s konkurencí*) obsahuje srovnání navrženého řešení a jeho implementace s konkurencí. 7. kapitola (*Závěr*) zkoumá naplnění cílů práce.

2. Analýza problému

Tato kapitola se zabývá prostředím malých firem a popisuje prováděné operace a jejich aktéry. Nejdůležitější operace jsou popsány ve formě takzvaných „případů užití“.

2.1. Prostředí firem zabývajících se nákupem a prodejem

Na úvod je nutné zdůraznit, že zde uvedený výčet operací a procesů nelze v žádném případě považovat za kompletní. Spíše jde o jakýsi úvod do problematiky.

Nejdůležitější činností firem zabývajících se nákupem a prodejem je, jak už název napovídá, prodej zboží. Tento proces se dá rozdělit do několika fází:

1. Zákazník vytvoří objednávku, čímž firmě oznámí, které zboží chce koupit.
2. Firma zajistí potřebné zboží (ze skladu či od dodavatelů).
3. Zákazník zaplatí objednané zboží.
4. Zákazník převezme zboží.

U některých firem se výše uvedené fáze mohou mírně lišit. Odlišné může být i jejich pořadí (například při nákupu na fakturu může zákazník zboží nejprve odebrat a až poté uhradit).

Vytvoření objednávky lze provést buď za pomoci zaměstnance firmy (osobně, telefonicky, poštou, emailem) nebo pomocí programu napojeného na firemní informační systém (internetový obchod, objednávkový terminál).

Po přijetí objednávky firma zkontroluje, zda je požadované zboží v dostatečném množství k dispozici v některém jejím skladu. Pokud tomu tak není, musí zboží nakoupit od některého ze svých dodavatelů (výběr dodavatele je netriviální záležitost, protože se musí brát v úvahu cena samotného zboží, termín dodání a náklady na dopravu.). Zpravidla platí, že nejvíce prodávaného zboží je skladem dostatečný počet kusů, zatímco ostatní zboží je k dispozici jen v několika kusech či vůbec. Skladové zásoby je samozřejmě nutné pravidelně doplňovat.

Zboží je možné zaplatit různými způsoby. Nejčastěji to bývá platba v hotovosti, dobírkou, pomocí kreditní či debetní karty, bankovním převodem či dárkovou poukázkou.

Doručit zboží zákazníkovi lze opět několika způsoby. V současnosti nejpoužívanějším je takzvaný osobní odběr, kdy si zákazník zboží vyzvedne v provozovně. Dále je možné zaslat zboží zákazníkovi poštou či pomocí kurýrní služby. V případě zboží, které lze distribuovat elektronicky (knihy, hudba, filmy, software) je možné rovněž umožnit zákazníkovi stažení daného produktu z firemního serveru.

Kromě prodeje a s ním souvisejícího nákupu zboží musí firma provádět celou řadu dalších operací. Některé ukládá přímo zákon - například všechny právnické osoby (firmy), které mají sídlo na území České republiky, a podnikající fyzické osoby (definované v §1 odst. 2 písm. d) až h) zák. č. 563/1991 Sb.) musí podle zákona č. 563/1991 Sb. vést záznamy o „stavu a pohybu majetku a jiných aktiv, závazků a jiných pasiv, dále o nákladech a výnosech a o výsledku hospodaření“ neboli *účetnictví*. Dalším příkladem je povinnost přijmout zboží k reklamaci (Práva a povinnosti spotřebitele v případě reklamace stanovuje Občanský zákoník (40/1964 Sb.) §619 až §627 a Zákon o ochraně spotřebitele (634/1992 Sb.) §19 [12]).

Dále je nutné řešit problémy vzniklé při prodeji – například dodání jiného zboží, než si zákazník objednal, neuhrazení zboží v daném časovém limitu či dodávka poškozeného zboží.

S prodejem souvisí i přidávání a odstraňování zboží z nabídky, oslovování nových zákazníků (reklama) a vytváření akčních nabídek zboží.

Při přijímání a vydávání zboží může docházet k omylům, a proto je nutné v pravidelných intervalech kontrolovat, zda zboží fyzicky přítomné ve skladu souhlasí s údaji uvedenými v informačním systému. Tomuto procesu se říká inventura a provádí se minimálně jednou ročně. Při inventuře je značně omezeno fungování skladu a potažmo celé firmy, proto je snaha provádět ji méně často. Na druhou stranu při velkých intervalech provádění inventury lze příčiny případných nesrovnalostí jen velmi těžko dohledat a napravit.

Některé operace jsou méně časté, ale o to důležitější – například přijímání a propouštění zaměstnanců či akvizice firem a s ní související slučování informačních systémů.

V dalších částech této práce se zaměříme pouze na operace bezprostředně související s prodejem zboží (vytváření, úpravy a zpracování objednávek, naskladňování a expedice zboží, přidávání a odstraňování zboží z nabídky). V rámci informačního systému tyto operace zajišťuje skladový systém (skladová evidence) a internetový obchod.

2.2. Operace související se skladovým systémem a internetovým obchodem

Hlavním účelem internetového obchodu je přijímání objednávek od zákazníků a poskytování informací o firmě. Proto je nutné, aby podporoval následující operace:

Zobrazení informací o firmě

Obchod musí umět v přehledné podobě zobrazit minimálně následující údaje: název firmy, IČO, DIČ, adresa, kontakt (email, telefon), otevírací doba. Dále je vhodné uvést seznam všech prodejen s mapou nebo popisem spojení.

Zobrazení seznamu zboží

Vzhledem k tomu, že obchod obsahuje zpravidla velké množství druhů zboží, je vhodné namísto jednoho velkého a nepřehledného seznamu vytvořit několik seznamů obsahujících zboží, které k sobě logicky patří. Dané seznamy poté budou přístupné prostřednictvím stromového navigačního menu, jehož položkám budeme říkat kategorie zboží.

Vyhledávání zboží

Pokud chce zákazník koupit konkrétní druh zboží, může pro něj být procházení navigačním menu příliš zdlouhavé. V takovém případě je vhodné umožnit mu zadat identifikátor nebo název (či alespoň část názvu) zboží a podle něj vyhledat odpovídající zboží. Je možné, že zákazník nezná název ani identifikátor, ale má představu o vlastnostech zboží, a proto by vyhledávání mělo mít druhý režim, ve kterém by tyto vlastnosti (například cenu či záruku) mohl specifikovat.

Zobrazení detailů zboží

Výše uvedené seznamy zboží mohou obsahovat jen několik údajů o zboží (hlavně identifikátor, název, cenu a případně stručný popis). Ostatní informace se zpravidla

umisťují na zvláštní stránku, přístupnou výběrem zboží ze seznamu. Na této stránce jsou zpravidla uvedeny následující údaje:

- Identifikátor zboží
- Název zboží
- Výrobce
- Cena za nejmenší prodejnou jednotku (kus, balení).
- Cena včetně všech poplatků a daní
- Záruční doba
- Podrobný popis
- Obrázek
- Orientační dodací doba

Tyto údaje se mohou lišit v závislosti na nabízených produktech (například záruční doba může být u potravin nahrazena minimální dobou trvanlivosti).

Vložení zboží do virtuálního nákupního košíku

Protože zákazníci jsou z reálného světa zvyklí na určité stereotypy nakupování, je vhodné se jim přizpůsobit a umožnit jim ukládat vybrané zboží do nákupního košíku. Možnost přidat zboží do košíku by měla být nabídnuta na stránce s detailem zboží či se seznamem zboží. Při přidávání musí být možné specifikovat počet kusů zboží.

Zobrazení obsahu virtuálního nákupního košíku

Po přidání zboží do košíku je vhodné zobrazit zákazníkovi seznam zboží, které se již v košíku nachází, a celkovou cenu nákupu. Musí být možné měnit počet kusů a případně zboží z košíku odebrat (či ho vyprázdnit celý najednou). Při každé změně se samozřejmě přepočítá celková cena.

Vytvoření objednávky z obsahu košíku

Při zobrazení obsahu košíku musí být zákazníkovi nabídnuta možnost vytvořit na základě obsahu košíku novou objednávku. Při vytváření objednávky zákazník vyplní informace o způsobu dodání zboží a platby. Po finálním potvrzení objednávky zákazníkem se vyprázdní obsah košíku a objednávka je vložena do informačního systému.

Registrace nového zákazníka

Při registraci se vytvoří nový zákaznický účet. Díky němu má zákazník při nakupování větší komfort (nemusí při každé objednávce znovu zadávat své údaje). Registrační údaje by měly minimálně obsahovat:

- Unikátní přihlašovací jméno (login) sloužící k identifikaci zákazníka
- Heslo (nebo jeho adekvátní náhradu při použití jiného způsobu autentizace)
- Jméno zákazníka
- Email (pro zasílání potvrzení objednávky a případných reklamních sdělení)
- Výchozí dodací adresu

Zákazník musí mít možnost kdykoliv své registrační údaje (včetně hesla) změnit.

Skladový systém má za úkol udržovat informace o stavu skladu. Pro účely této práce mu přidělíme další úkol - správu objednávek. Ke splnění daných úkolů musí systém podporovat alespoň následující operace:

Příjem zboží na sklad

Pří naskladňování zboží je nutné převést dodavatelův identifikátor zboží na vnitrofiremní. Je vhodné vnitrofiremní identifikátory volit tak, aby se shodovaly s těmi dodavatelskými v maximálním počtu případů (zmenší se tak režie systému i náklady na případné přelepení čárových kódů na zboží). Dalším úskalím je nalezení vhodného místa pro fyzické umístění zboží tak, aby bylo snadno dostupné a bezpečně uložené. To je velký problém zejména u menších skladů. U velkých skladů zase hraje významnou roli čas potřebný na transport zboží a na přesun vysokozdvižných vozíků. V takovém případě je vhodné zvážit použití systému pro kompletní management skladu (a nikoliv jen skladové evidence, jako v našem případě).

Výdej zboží ze skladu (expedice)

Při výdeji je zapotřebí zkontrolovat, zda je vydáváno požadované zboží a zda souhlasí počet kusů. Nejčastěji se tak děje za pomoci čtečky čárových kódů, kdy je kód každého kusu nasnímán a porovnán s údaji v systému. Výhodou je přesná evidence o pohybu konkrétních kusů zboží (při inventuře tak lze zjistit, které zboží chybí nebo naopak přebývá).

Přesun zboží mezi sklady

Tato operace se dá realizovat jako vyskladnění zboží v jednom skladu a naskladnění ve druhém. Tímto způsobem lze odhalit případné krádeže při transportu.

Zjištění aktuálního počtu kusů zboží v daném skladu

Musí být možné zjistit aktuální počet kusů, které by měly být fyzicky přítomny v jednotlivých skladech, a počet kusů, které mají odběratelé rezervovány. Pokud sečteme kusy ze všech skladů a odečteme od tohoto čísla počet rezervovaných kusů získáme počet kusů, které jsou zákazníkům k dispozici (toto číslo by měl systém umět na požádání sdělit). V pravidelných intervalech je vhodné tento počet kontrolovat a případně zboží přiobjednat.

Vytvoření nové objednávky

Tato operace je velmi častá díky osobnímu prodeji. Systém musí umožňovat vytvořit objednávku pro anonymního zákazníka (protože lze jen těžko očekávat, že každý zákazník bude prodavači sdělovat své osobní údaje).

Úprava objednávky

Úprava objednávky znamená přidání, odebrání zboží nebo změnu počtu kusů. Dále musí být možné měnit dodací adresu a způsob platby a doručení. Jakákoliv změna v objednávce spolu s jejím iniciátorem by měla být systémem zaznamenávána. Otevřenou otázkou je, zda změny mohou provádět sami zákazníci (prostřednictvím internetového obchodu) nebo jen oprávnění zaměstnanci firmy.

Smazání objednávky

Při mazání objednávky je nutné zrušit rezervace zboží, které objednávka obsahuje. Odstranění objednávky ze systému nesmí být definitivní (musí se uchovat kopie), aby bylo možné v případě budoucích nesrovnalostí tuto operaci dohledat. Rovněž je vhodné vést záznamy o tom, který zaměstnanec dal příkaz k jejímu odstranění (aby bylo možné ho postihnout v případě, že tento příkaz byl vydán neoprávněně).

Zpracování objednávky

Zpracovanou objednávku nesmí být možné měnit (standardním způsobem). Změny obsahu objednávky je nutné realizovat prostřednictvím storno faktur (dobropisů). Při zpracování systém provede vyskladnění zboží (aktualizuje se

skladová evidence) a skladníci připraví zboží k expedici. Pokud na skladu není dostatek objednaného zboží nelze objednávku zpracovat. V takovém případě je nutné zboží přiobjednat nebo dohodnout se zákazníkem úpravu objednávky.

2.3. Aktéři

Ve firmě (a jejím okolí) se vyskytují uživatelé s různými požadavky na informační systém. V rámci této části jim budeme říkat aktéři, protože účinkují ve výše uvedených operacích.

Administrátor
Instaluje a konfiguruje klientské aplikace, aplikační, databázový a webový server. Kontroluje logy. Provádí zálohu databáze a konfiguračních souborů, pravidelně testuje a instaluje aktualizace všech částí systému. Vytváří, upravuje a ruší zaměstnanecké účty, přiděluje a odebírá oprávnění.

Prodavač
Vytváří, upravuje, přijímá, ruší a zpracovává objednávky. Přijímá platby a vydává na ně potvrzení. Přijímá reklamace.

Obchodník
Stejné akce jako prodavač a navíc: vytváří a upravuje účty dealerů, vytváří dobropisy. Přidává a odebírá zboží z/do nabídky, přidává, maže a upravuje kategorie zboží, upravuje ceny a detaily zboží (na příkaz vedoucího). Objednává zboží u dodavatelů.

Skladník
Přijímá a vydává zboží. Provádí inventuru skladu. Připravuje zboží pro odeslání poštou či jiným dopravcem.

Vedoucí
Vytváří a upravuje zaměstnanecké účty, přiděluje základní oprávnění. Určuje ceny zboží, rozhoduje o dealerských slevách a komunikuje s dodavateli. Má přístup ke statistikám prodejů a zisku.

Odběratel
Vytváří objednávky, odebírá a platí zboží, registruje se do internetového obchodu, mění své registrační údaje, reklamuje zboží.

Dodavatel
Přijímá objednávky na dodávku zboží.

2.4. Případy užití (Use Case)

Jednotlivé operace jsou popsány pomocí *Případů užití* (*Use Cases*) ve formátu prosazovaném C. Larmanem [16].

UC1: Prodej zboží

Aktéři
Zákazník, prodavač, skladník.

Základní tok události

1. Zákazník přichází do provozovny se seznamem zboží, které hodlá nakoupit.
2. Prodavač vytvoří novou objednávku.
3. Prodavač zadá identifikátor zboží a počet kusů.
4. Zboží je přidáno do objednávky a je zobrazen název zboží, kusová a celková cena.
5. Prodavač opakuje kroky 3 a 4 pro každé zboží ze zákazníkova seznamu
6. Prodavač zkontroluje objednávku a potvrdí ji.
7. Sytém zobrazí celkovou cenu včetně všech daní a poplatků
8. Prodavač oznámí cenu zákazníkovi a inkasuje platbu
9. Prodavač zadá platbu do systému.
10. Sytém zaúčtuje platbu a vyskladní požadované zboží
11. Je vytištěna účtenka, kterou prodavač zákazníkovi předá.
12. Zákazník přechází k výdeji zboží, kde předá skladníkovi účtenku.
13. Skladník zadá číslo účtenky.
14. Systém zobrazí seznam zboží, které má být zákazníkovi vydáno.
15. Skladník najde zboží ve skladu a přemístí ho k výdeji.
16. Skladník porovná připravené zboží se seznamem a potvrdí výdej.
17. Systém aktualizuje stavy zboží na skladě a vytiskne dodací list.
18. Zákazník podepisuje dodací list, přebírá zboží a odchází.

Alternativní tok události

1-6a: Zákazník zboží objednal prostřednictvím internetového obchodu

1. <<Include>> UC2 – Vytvoření objednávky prostřednictvím internetového obchodu
2. Zákazník sdělí prodavači číslo objednávky.
3. Prodavač zadá číslo do systému.
4. Systém vyhledá objednávku

3a: Identifikátor nenalezen

1. Prodavač ověří správnost zadání:

1a: Identifikátor je zadán chybně

1. Prodavač opraví chybu

1b: Identifikátor je zadán korektně

1. Prodavač oznámí chybu zodpovědnému pracovníkovi (nadřízenému, administrátorovi)
2. Prodavač nabídne zákazníkovi jiné zboží stejného typu

4a: Zboží není skladem v dostatečném množství

1. Prodavač na tuto skutečnost upozorní zákazníka.
2. Zboží je přidáno s počtem kusů, který je aktuálně skladem

2a: Zákazník chce zrušit objednávku

1. Prodavač zruší objednávku
2. Systém přesune objednávku mezi zrušené
3. Zákazník odchází

2b: Zákazník chce jiné zboží

1. Prodavač nabídne zákazníkovi zboží stejného typu.
2. Zákazník změnu odsouhlasí.
3. Prodavač zadá identifikátor zboží a počet kusů.
4. Zboží je přidáno do objednávky a je zobrazen název zboží, kusová a celková cena.

2c: Zákazník dané zboží nechce
1. Prodavač odstraní zboží z objednávky.

8a: Zákazník nárokuje prodej za dealerské ceny
1. Prodavač zadá do systému identifikátor zákazníka
2. Systém podle identifikátoru vyhledá výši slevy a přepočítá ceny
3. Sytém zobrazí novou celkovou cenu (včetně všech daní a poplatků).
4. Prodavač oznámí cenu zákazníkovi a inkasuje platbu.

8-9a: Platba již byla provedena (bankovní převod)
(Má smysl pouze pokud objednávka byla vytvořena předem)
1. Zákazník na tuto skutečnost upozorní prodavače
2. Prodavač ověří přijetí platby

8-9b: Platba kreditní kartou
1. Zákazník vloží kartu do snímače
2. Systém načte údaje z karty a čeká na potvrzení transakce
3. Prodavač potvrdí platbu
4. Systém vyšle žádost o autorizaci platby externí bankovní aplikaci
5. Systém obdrží kladnou odpověď na žádost, zaúčtuje platbu a vytiskne účtenku

5a: Žádost je zamítnuta
1. Prodavač pořádá zákazníka o platbu v hotovosti (kroky 8-9)

5b: Ve stanoveném časovém limitu nedostane systém odpověď
1. Prodavač zadá pokyn k opětovnému vyslání žádosti.
2. Pokud systém nedostane ve stanoveném časovém limitu odpověď postupuje se stejně, jako kdyby žádost byla zamítnuta

6. Zákazník podepíše účtenku.
7. Prodavač zkontroluje podpis a uloží podepsanou účtenku

Stav po ukončení
- Prodej je zaúčtován
- Skladová evidence je aktualizována
- Objednávka je označena jako vyřízená

Speciální požadavky
- Čtečka čárových kódů ve skladu.

Otevřené problémy
- Zboží nutné k uspokojení objednávky se nachází v jiném skladu (jiných skladech)
- Současné zpracovávání (vícero prodavači) objednávek obsahujících zboží, které není skladem v dostatečném množství pro uspokojení všech zpracovávaných objednávek.

UC2: Vytvoření objednávky prostřednictvím internetového obchodu

Aktéři
Zákazník

Základní tok události
1. Zákazník otevírá ve svém prohlížeči úvodní stránku internetového obchodu.

2. Zákazník se přihlašuje.
3. Zákazník prochází nabídku zboží.
4. Zákazník vybírá jeden druh zboží, který ho zajímá.
5. Systém zobrazí detaily daného zboží.
6. Zákazník vkládá zboží do virtuálního nákupního košíku.
7. Systém zobrazí stránku s obsahem košíku a možností upravit počty kusů či zboží odebrat.
8. Zákazník opakuje kroky 3-7 dokud není spokojen s obsahem košíku.
9. Systém nabídne vytvoření nové objednávky z obsahu košíku.
10. Zákazník vyplní způsob doručení a platby a případně dodací adresu.
11. Zákazník potvrdí vytvoření objednávky.
12. Systém objednávku uloží, rezervuje objednané zboží a odešle na adresu zákazníka potvrzující email s detaily objednávky.
13. Zákazník se odhlašuje.

Alternativní tok události

2a: Zákazník nemá zřízen účet (není registrován).
1. Systém zobrazí registrační formulář.
2. Zákazník vyplní požadovaná data včetně přihlašovacího jména a hesla. Heslo zadává pro kontrolu dvakrát.
3. Systém ověří korektnost dat (například správný formát emailu) a unikátnost přihlašovacího jména.
3a: Formulář obsahuje nekorektní data.
1. Systém upozorní zákazníka a nabídne mu možnost opravy.
2. Zákazník opraví nekorektní data.
4. Systém vytvoří zákazníkovi účet a nabídne mu přihlašovací obrazovku.
5. Zákazník se přihlašuje.

Stav po ukončení

- Objednávka je uložena v systému
- Ve skladu je rezervován příslušný počet kusů objednaného zboží

UC3: Příjem zboží

Aktéři

Skladník, obchodník, (dodavatel, dopravce, vedoucí, administrátor)

Základní tok události

1. Obchodník zadá pokyn k vyhledání zboží, které není skladem v definovaném minimálním počtu kusů.
2. Systém vyhledá zboží a zobrazí seznam dodavatelů i s cenami a dodacími lhůtami.
3. Obchodník vybere dodavatele a vytvoří novou objednávku.
4. Obchodník zadá zboží a počty kusů.
5. Obchodník zadá sklad, do kterého se má zboží doručit, a potvrdí objednávku.
6. Systém objednávku uloží, kontaktuje informační server dodavatele a objednávku mu předá.
7. Dodavatel zpracuje objednávku a zboží odešle (v domluveném časovém limitu).
8. Dopravce doručí zboží do zadaného skladu.

9. Skladník zkontroluje zda zboží v dodávce souhlasí se zbožím uvedeným na dodacím listě.
10. Skladník zboží převezme a podepíše dodací list.
11. Skladník vyhledá v systému příslušnou objednávku.
12. Skladník zadá čárové kódy přijatého zboží.
13. Systém zkontroluje zda počty kusů a typ zboží souhlasí s objednávkou.
14. Systém aktualizuje skladovou evidenci a označí objednávku za vyřízenou.
15. Skladník uloží zboží do skladu.

Alternativní tok události

6a: Dodavatel nemá informační systém schopný spolupráce.
1. Obchodník zvolí alternativní způsob doručení objednávky (telefonicky, poštou, emailem).

6b: Server dodavatele nepotvrdí převzetí objednávky ve stanoveném časovém limitu.
1. Systém oznámí obchodníkovi vypršení limitu a nabídne opakování akce.
2. Obchodník potvrdí opakování akce nebo postupuje jako v případě 6a.

9a: Zboží nesouhlasí s dodacím listem
1. Skladník toto ohlásí dopravci a nepodepíše dodací list.
2. Skladník vyhledá v systému související objednávku a přiloží k ní poznámku o nepřevzetí zboží.

12a: Došlo k zadání neznámého čárového kódu
1. Skladník zkontroluje správnost zadání kódu.
2. Pokud je kód správný, vyhledá skladník čárový kód uložený v systému.
   3a: Čárový kód je přítomen, ale nesouhlasí s kódem na zboží.
      1. Skladník nalepí správný kód na zboží
      2. Skladník provede příjem zboží
   3b: Čárový kód je přítomen a souhlasí s kódem na zboží.
      1. Skladník oznámí administrátorovi chybu informačního systému.
      2. Skladník přeruší přijímání zboží dokud nebude chyba odstraněna.
3. Není-li zboží žádný kód přiřazen kontaktuje skladník vedoucího.
4. Vedoucí vloží čárový kód do systému.

13a: Dodané zboží nesouhlasí s objednávkou
1. Systém zobrazí seznam zboží, které nesouhlasí (přebývá či chybí).
2. Skladník kontaktuje vedoucího, který rozhodne, zda je možné pokračovat v naskladňování.
   2a: V naskladňování není možné pokračovat.
      1. Skladník vyhledá v systému související objednávku a přiloží k ní poznámku o chybné dodávce zboží.
      2. Skladník fyzicky přesune všechno dodané zboží do meziskladu.
   2a: V naskladňování je možné pokračovat.
      1. Skladník naskladní objednané zboží.
      2. Neobjednané zboží je přesunuto do meziskladu.
3. Vedoucí kontaktuje dodavatele a dohodne s ním nápravu.

Stav po ukončení

- Objednávka je označena jako vyřízená.
- Zboží je fyzicky přítomno na skladě a skladová evidence je aktualizována.

Speciální požadavky

- Čtečka čárových kódů ve skladu.

UC4: Odstranění zboží z nabídky

Aktéři

Obchodník, (odběratel)

Základní tok události

1. Obchodník zadá identifikátor zboží, které chce odstranit z nabídky.
2. Systém zkontroluje, zda je počet kusů skladem roven nule.
3. Systém ověří, že zboží není objednáno u žádného dodavatele.
4. Systém ověří, že zboží není objednáno žádným odběratelem.
5. Systém požádá o potvrzení.
6. Obchodník potvrdí odstranění.
7. Systém označí zboží jako odebrané a aktualizuje všechny související záznamy.

Alternativní tok události

2a: Zboží je skladem:

1. Systém zkontroluje zda počet kusů objednaných odběrateli je nižší než počet kusů skladem.
    1a: Zboží je skladem v nedostatečném množství.
        1. Obchodník vybere objednávky, které nelze uspokojit.
        2. Obchodník kontaktuje odběratele, kteří vytvořili dané objednávky a informuje je o nemožnosti dodání daného zboží.
        3. Odběratel požaduje odstranění daného zboží z objednávky.
            3a: Odběratel požaduje zrušení objednávky:
                1. Obchodník stornuje objednávku.
            3b: Odběratele nelze zastihnout:
                1. Obchodník stornuje objednávku.
                2. Obchodník zašle odběrateli email informující ho o zrušení objednávky.
        4. Obchodník upraví objednávku.
2. Systém převede zboží do režimu výprodeje – zboží lze prodávat, ale již ne nakupovat.
3. Obchodník nastaví výši výprodejové slevy.
4. Po prodání posledního kusu zboží se odstraní z nabídky.

3a: Zboží je objednáno u alespoň jednoho dodavatele:

1. Obchodník kontaktuje dodavatele a dohodne s ním odstranění zboží s objednávky.
    1a: Zboží nelze z objednávky odstranit.
        1. Odstranění zboží z nabídky se odloží do doby, než dorazí poslední dodávka zboží.
        2. Zboží se naskladní.
        3. Stejný postup jako v situaci 2a – zboží je skladem.

4a: Zboží je objednáno alespoň jedním odběratelem:

1. Stejný postup jako v situaci 2a – zboží je skladem.

Stav po ukončení

- Zboží je označeno jako odebrané a skladová evidence je aktualizována
- Všechny objednávky obsahující dané zboží jsou vyřízené.

Kvůli větší přehlednosti budou další případy užití uvedeny ve stručnějším formátu:

UC5: Přidání nového zboží do nabídky

Aktéři

Vedoucí, obchodník

Základní tok události

Zajistí se alespoň jeden dodavatel nového druhu zboží. Vedoucí určí ceny pro koncové zákazníky a jednotlivé dealerské kategorie. Poté se zboží přiřadí kód, který ho jednoznačně identifikuje v rámci firmy, a určí se kategorie (jedna či více), ve kterých bude zobrazeno v rámci internetového obchodu. Všechny tyto informace a některé další (název, výrobce, popis, obrázek, velikost balení, DPH, záruka, …) jsou vedoucím (či pověřeným obchodníkem) vloženy do systému. Systém ověří unikátnost identifikátoru a korektnost všech vkládaných dat (nezápornost cen …). Po úspěšném vložení je možné přidat převody mezi dodavatelskými a vnitrofiremními identifikátory.

Alternativní tok událostí

Není-li identifikátor unikátní systém to ohlásí a vedoucí zvolí jiný identifikátor nebo odebere kolidující zboží z nabídky. Poté obchodník zkusí zboží vložit znovu.

Při vložení nekorektních dat ohlásí systém chybu. Obchodník data opraví a zkusí zboží vložit znovu.

UC6: Reklamace

Aktéři

Prodavač, odběratel, skladník, vedoucí

Základní tok události

Odběratel přijde do provozovny a oznámí prodavači svůj úmysl reklamovat zboží. Prodavač ověří, že zákazník má právo dané zboží reklamovat (zboží bylo koupeno od jeho firmy a od prodeje uplynul čas kratší než je záruční doba). Prodavač sepíše se zákazníkem reklamační protokol, vytiskne ho, předá ho zákazníkovi a převezme od něj zboží. Zákazník odchází. Prodavač předá zboží skladníkovi, který ho připraví k odeslání dodavateli, výrobci nebo přímo do autorizovaného servisu. Po vyřízení reklamace je zákazník kontaktován a zboží je mu předáno. Zákazník podepíše převzetí a prodavač potvrdí v systému vyřízení reklamace.

Alternativní tok událostí

Zákazník má nárok na vrácení peněz nebo výměnu výrobku. Požaduje-li zákazník vrácení peněz zadá prodavač tento požadavek do systému. Poté vyplatí zákazníkovi požadovanou sumu a nechá zákazníka podepsat příjmový doklad. V případě výměny výrobku vytvoří prodavač novou objednávku obsahující pouze daný výrobek a označí ji jako uhrazenou. Poté se postupuje stejně jako v případě obyčejného prodeje.

UC7: Inventura

Aktéři

Skladník, administrátor

Základní tok události

Příjem a výdej zboží je po dobu inventury zastaven. Pro každý druh zboží spočítá skladník počet kusů fyzicky přítomných ve skladu. Zadá identifikátor zboží do systému a nechá si zobrazit počet kusů, který má být skladem. Obě čísla se musí rovnat.

Alternativní tok událostí

Počet kusů skladem nesouhlasí s údaji v systému. Skladník znovu přepočítá zboží ve skladu. Je-li počet kusů v systému větší než ve skladu je údaj v systému ručně změněn administrátorem (tato operace se zaznamená do logů) a vzniklá škoda je rozpočítána mezi zaměstnance mající hmotnou odpovědnost za sklad. Je-li počet kusů ve skladu vyšší než počet kusů v systému provede se naskladnění přebývajících kusů zboží.

3. Návrh řešení

V této kapitole se budeme zabývat návrhem řešení daných problémů a cílů z hlediska softwarové architektury. Obsahem je výběr platformy, rozdělení programu do částí v rámci třívrstvé architektury a návrh databázového schématu.

3.1. Platforma

Jednou z nejdůležitějších otázek, kterou si musí položit většina vývojářů, je, pod jakým operačním systémem (či operačními systémy) má aplikace fungovat. Toto rozhodnutí výrazně ovlivní celý další návrh.

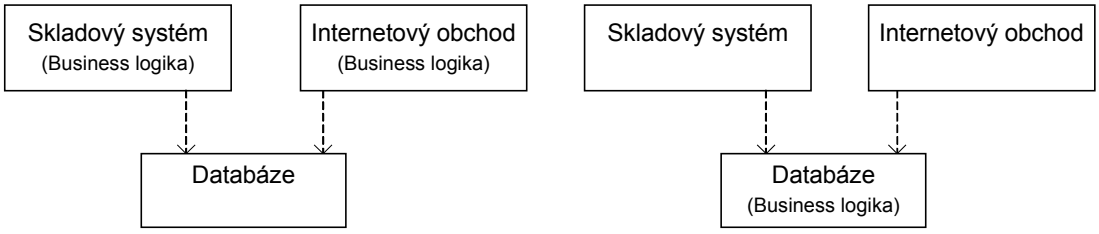
V našem případě můžeme uvažovat pouze operační systémy běžící na procesorech s architekturou x86, tedy na běžných PC a menších serverech. V České Republice jsou zdaleka nejrozšířenějším systémem Windows[1] od společnosti Microsoft. Jejich výhodou je právě velká rozšířenost, uživatelská přívětivost a dostupnost ovladačů hardware. Mezi nevýhody patří relativně vysoká pořizovací cena (v porovnání například s Linuxem), nižší konfigurovatelnost, nedostupnost zdrojových kódů a pomalejší vydávání oprav bezpečnostních chyb. Další operační systém, který je vhodné vzít v úvahu, je Linux. Jeho výhodou je otevřený zdrojový kód, nízké náklady na pořízení, schopnost běhu i na slabším hardware, rychlé opravy nalezených chyb a široké možnosti konfigurace. Mezi nevýhody patří menší rozšířenost, horší dostupnost ovladačů a existence mnoha navzájem ne zcela kompatibilních distribucí (což někteří lidé považují spíše za výhodu – možnost svobodně si zvolit takovou, která jim vyhovuje). Ostatními systémy (Solaris, FreeBSD, BeOS, DOS) se nebudeme kvůli jejich malému podílu na trhu zabývat.

Protože naprostá většina malých firem (a všechny střední a velké) už nějaké počítače, a tedy i nějaký operační systém, vlastní, je v zájmu snížení počátečních nákladů nutné umožnit běh programu na jejich stávajících systémech, což v naprosté většině znamená Windows.

Primární platformou tedy budou Windows, ale program by se měl dát jednoduše upravit tak, aby běžel i na unixových systémech (Linuxu).

3.2. Části programu

Zadání práce požaduje, aby architektura programu umožňovala přístup k systému prostřednictvím různých klientských aplikací. Pokud bychom chtěli použít klasický dvouvrstvý model klient-server, znamenalo by to, že je nutné aplikační logiku buď

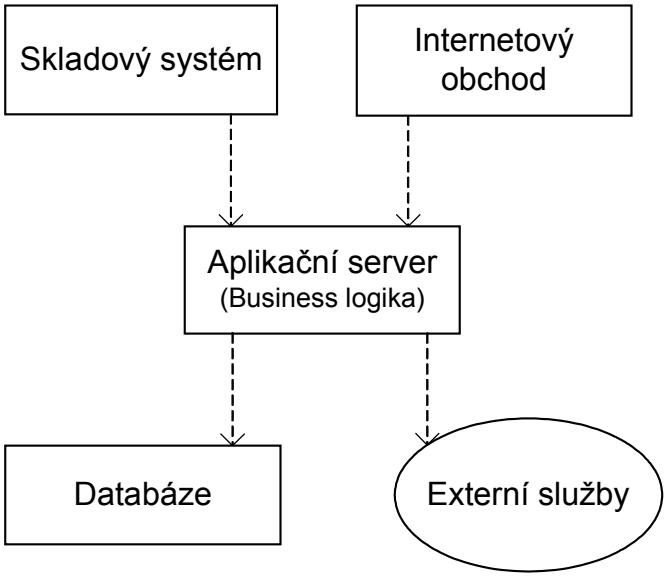


Obrázek 3.1.: Umístění aplikační logiky

1 Dle statistik společnosti Internet Info, s.r.o. tvoří podíl windows 98,32%. Verze XP má podíl 78,94% a verze 2000 7,52%. Tyto statistiky ukazují podíl jednotlivých operačních systémů u návštěvníku webů, které využívají služby NAVRCHOLU.cz, a tedy skutečné zastoupení operačních systémů může být mírně odlišné (menší podíl systému Windows). [9]

přidat do databáze (server), nebo že musí být přítomna v každé klientské aplikaci (a to by značně zkomplikovalo možnost úprav – každá by se musela provést redundantně ve všech klientech, což je v rozporu s naší snahou o co nejjednodušší upravovatelnost) jak je vidět na obrázku 3.1. V případě umístění logiky do databáze je problém s přenositelností kódu při změně databázového software (jednotlivé databázové platformy nejsou v tomto směru kompatibilní). Tyto problémy se samozřejmě dají vyřešit, nicméně ukazují, že dvojvrstvá architektura není v našem případě nejvhodnější.

Mnohem větší flexibilitu nám poskytne architektura třívrstvá, kde bude kromě databázového serveru a klientských aplikací ještě aplikační server, jenž bude obsahovat aplikační logiku[2]. Díky tomu, že bude logika na jednom místě, budou případné úpravy, včetně změny databázové platformy, značně jednodušší, což se samozřejmě odrazí ve snížených nákladech firmy. Toto řešení navíc umožňuje, pro klienty zcela transparentní, použití více databází na různých serverech, či integraci externích služeb (příkladem může být webová služba pro ověřování správnosti DIČ, fungující v rámci Portálu Evropské unie [10]). Další podstatnou výhodou je možnost napojení aplikačního serveru na aplikace (servery) dodavatelů (například automatické objednávání docházejícího zboží). Třívrstvá architektura má samozřejmě i své nevýhody - například klade větší nároky na kvalitu síťového propojení, je o něco náročnější na hardwarové prostředky (kvůli aplikačnímu serveru) a rovněž naprogramování a otestování třívrstvých aplikací je obtížnější.



Obrázek 3.2: Třívrstvá architektura

Protože jedním z požadavků je možnost jednoduchého provádění úprav, je více než rozumné jednotlivé části programu ještě dále rozdělit do podvrstev[3] tak, aby úpravy v jedné podvrstvě ovlivnily ostatní co možná nejméně. Například u aplikačního serveru je vhodné oddělit přístup k databázi a externím službám, aplikační logiku a prezentaci dat vyšší vrstvě. V rámci skladového systému jsou zapotřebí alespoň dvě podvrstvy – jedna pro komunikaci s aplikačním serverem a

2 Toto je trochu upravené pojetí třívrstvé architektury, vyskytující se hlavně v souvislosti s webovými službami. Obecně se hovoří o presentační vrstvě, vrstvě logiky a datové vrstvě. [15]

3 V originále *layers*. Čeština má bohužel pro výrazy *tier* (Three tier architecture) a *layer* stejný ekvivalent a sice slovo vrstva. V rámci této práce tedy bude používán výraz vrstva ve významu odpovídajícím anglickému *tier* a podvrstva bude odpovídat anglickému *layer.*

druhá pro komunikaci s uživatelem. Stejně tak u internetového obchodu. Databáze může být monolitická, tvořená jedinou podvrstvou starající se o integritu dat (triggery a integritní omezení).

Pro urychlení vývoje aplikačního serveru je vhodné použít nějakého mechanismu vzdáleného volání procedur (zjednoduší komunikaci s klienty – odpadne tak nutnost vymýšlet a implementovat vlastní protokol). Tyto mechanismy můžeme rozdělit na binární a textové. Mezi binární patří například CORBA[4], DCOM, .NET Remoting či Java RMI, z textových jmenujme protokol XML-RPC a jeho nástupce SOAP[5]. Z daných možností se pro naše účely jeví jako nejvhodnější CORBA a SOAP. DCOM je již zastaralý a byl nahrazen .NET Remotingem. Ten je určen jen pro platformu .NET, což výrazně omezuje skupinu použitelných programovacích jazyků, na rozdíl třeba od mechanismu CORBA, který je univerzální. Java RMI lze vyloučit z podobných důvodů – celou aplikaci by bylo nutno naprogramovat v jazyce Java. A XML-RPC byl již nahrazen modernějším protokolem SOAP (ačkoli se stále pro některé jednodušší úlohy používá).

Výhodou architektury CORBA je větší rychlost a menší objem zpráv, nevýhodou naopak problematické fungování v sítích s firewally (kvůli protokolu IIOP). Oproti tomu webové služby nemají s firewally problém (díky protokolu http, nad kterým funguje SOAP). Webové služby jsou mladší než CORBA a v dnešní době mnohem rozšířenější. Jejich hlavní nevýhodou je relativní pomalost, daná textovým charakterem (zprávy jsou posílány ve formátu XML). Obecně se dá říci, že webové služby se více hodí v případě značně heterogenního prostředí, jakým je například Internet. CORBA je naopak vhodnější v případě většího objemu zasílaných zpráv, kdy se naplno projevují její větší rychlost. Mírným favoritem jsou, díky jejich větší rozšířenosti, webové služby, nicméně výběr vhodného mechanismu RPC záleží hlavně na konkrétní situaci. Tuto otázku tedy necháme otevřenou.

3.3. Autentizace a autorizace

Protože program budou používat uživatelé vystupující v různých rolích (jak vyplývá přímo ze zadání), je nutné zajistit, aby mohli provádět jen ty operace, které bezprostředně souvisí s jejich rolí. Tím se značně zmenší pravděpodobnost výskytu omylem provedených operací a tedy i náklady na odstranění vzniklých chyb. Z toho vyplývá nutnost vytvoření nějakého autorizačního (a autentizačního) mechanismu, který by měl splňovat následující požadavky:

- Rychlost – mechanismus nesmí znatelně zpomalovat odezvu systému
- Jednoduchost – musí být snadno spravovatelný
- Bezpečnost – nesmí existovat jednoduchý způsob, jak ho obejít
- Granularita – autorizace alespoň na úrovni jednotlivých operací.

Autentizace

Nejjednodušším a nejrozšířenějším způsobem je přihlašování pomocí uživatelského jména a hesla. Heslo musí být dostatečně dlouhé a složité, tak aby nešlo zjistit za pomoci tzv. slovníkového útoku[6] a útok brutální silou[7] trval bezpečně dlouho, a zároveň musí být zapamatovatelné (jinak by si jej uživatelé někam

4 CORBA stejně jako .NET Remoting je spíše architektura, než jen pouhý mechanismus RPC.

5 Možná by bylo správnější použít termín webové služby (lépe odpovídá ostatním uvedeným technologiím – například CORBA). SOAP je jen protokolem stojícím na druhé nejnižší úrovni v této architektuře ([1] str.12).

poznamenali a případný útočník by si jej odtamtud mohl přečíst). Takové heslo lze získat například vhodnou transformací tzv. *passphrase* (věta, citát, básnička).

Další možností je identifikace uživatele pomocí nějakého tokenu, typicky magnetické nebo čipové karty. Nevýhodou je potřeba speciálního zařízení, které tokeny zpracovává, a s tím související zvýšení nákladů. Další nevýhodou jsou komplikace spojené se ztrátou tokenu a s přihlášením uživatele, který si token někde zapomněl. Výhodou je naopak obtížnější kopírování tokenů a celkově větší bezpečnost.

Použití biometrik se velmi podobá tokenům. Jedná se o techniku identifikace uživatele na základě jeho osobních charakteristik (otisky prstů, tvář, analýza hlasu, …). Na rozdíl od tokenů je nelze zapomenout a rovněž kopírování je téměř nemožné. Na druhou stranu občas dochází k falešným odmítnutím legálních uživatelů či naopak k přijetí útočníků.

Z daných mechanismů se pro naše účely jeví jako nejvhodnější použití hesel. Uživatelé jsou na tento způsob autentizace zvyklí, nepřináší žádné další náklady na hardware a je dostatečně bezpečný. V případě větších nároků na bezpečnost je možné jej vyměnit za nějaký sofistikovanější.

Autorizace

Nejjednodušším řešením je přidělit každému uživateli a každé operaci nějakou číselnou konstantu a při každém přístupu je porovnat (pokud je konstanta u operace menší nebo rovna uživatelově konstantě, tak se mu operace povolí). Výhodou je snadná implementovatelnost a rychlost, nevýhodou je obtížnější administrace (pokud má uživatel A povolenu operaci 1 a nepovolenu operaci 2, pak uživatel B mající povolenu operaci 2 z principu nemůže mít nepovolenu operaci 1).

Dalším řešením je použití *Seznamu oprávnění* (Access Control List [16]) – každá operace bude obsahovat seznam uživatelů, kteří ji mají právo provádět. Výhodou je opět rychlost a snadná implementovatelnost. Nevýhodou je komplikovanější zjišťování, které operace může uživatel provádět (musí se prohledat seznamy oprávnění u všech operací), a celkově obtížnější administrace.

Nevýhody předchozího řešení částečně odstraňuje *Přístupová matice* (Access Control Matrix [16]). Řádky matice odpovídají jednotlivým uživatelům a sloupce operacím. Nevýhodou je větší paměťová náročnost, obtížnější implementovatelnost a o něco pomalejší fungování. Výhodou je naopak možnost rychle zjistit, které operace smí daný uživatel provádět a kteří uživatelé smějí provádět danou operaci.

Posledním způsobem, o kterém bude řeč, je *Systém rolí a skupin*. Role je množina oprávnění (v našem případě tedy množina operací, které smí její majitel provádět). Skupina je množina uživatelů majících stejné oprávnění ([16]). Toto řešení výrazně usnadňuje administraci větších systému, kde je mnoho uživatelů a operací, nicméně jeho přínos je znatelný i u menších programů. Z výše uvedených způsobů nejlépe splňuje naše požadavky (hlavně na rychlost a jednoduchost).

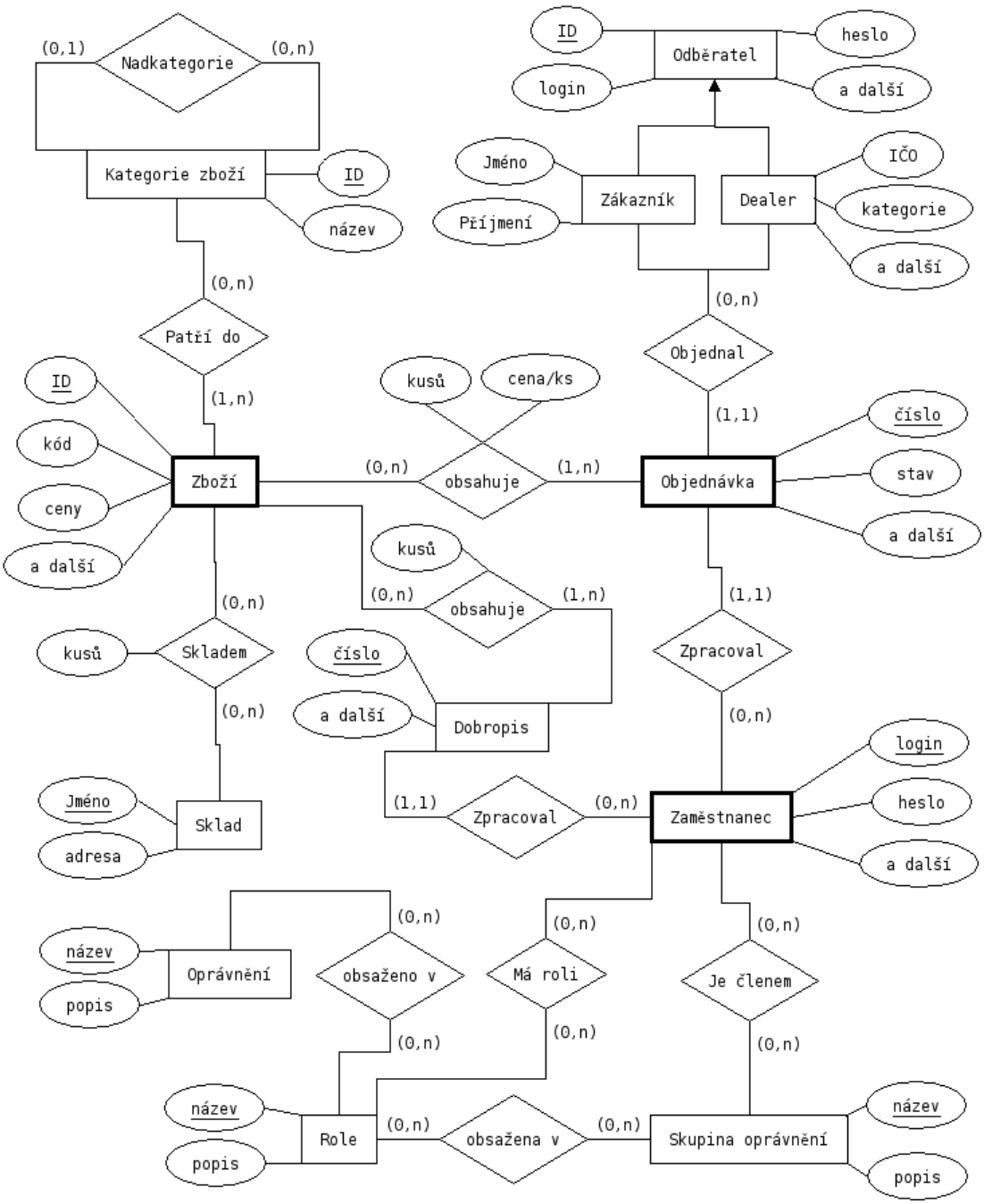
---

6 Slovníkový útok znamená automatické či ruční zadávání smysluplných kombinací písmen dle nějakého slovníku (soubor obsahující slova).

7 Při útoku brutální silou se zkouší všechny kombinace přípustných znaků v pořadí daném algoritmem

3.4. Databázové schéma

Databázové schéma můžeme rozdělit na dvě části – firemní a systémovou. Firemní část obsahuje data důležitá pro chod firmy (objednávky, zboží, odběratele, zaměstnance, …), systémová obsahuje data potřebná pro fungování aplikačního serveru a klientských aplikací (například uživatelská nastavení, šablony, logy, …). Na obrázku 3.3 je vidět ER model firemní části databázového schématu (včetně databázové části autorizačního mechanismu, která však logicky patří pod systémovou část schématu).



Obrázek 3.3.: ER schéma databáze

Kvůli přehlednosti byly na obrázku 3.3 u každé entity sdruženy méně zajímavé atributy do jednoho atributu s názvem „a další“. V následujících odstavcích jsou popsány některé entity z obrázku 3.3.

Pro reprezentaci *kategorií zboží* se předpokládá použití sebereferenční tabulky. Výhodou tohoto řešení oproti reprezentaci pomocí genealogického[8] či DFS[9] stromu je jednoduchost a snadné provádění aktualizací, nevýhodou nutnost použití rekurze při prohledávání do hloubky (u genealogického stromu stačí kategorie setřídit podle genealogického identifikátoru, u DFS stromů setřídíme kategorie podle atributu obsahujícího čas vstupu do uzlu). Každý druh zboží může být v jedné až n kategoriích. Otevřenou otázkou je, zda má být zboží nacházející se v nějaké kategorii zařazeno rovněž do každé nadřazené kategorie.

V rámci schématu rozlišujeme dva typy *odběratelů* – běžné *zákazníky* a *dealery*. Dealer je podnikající fyzická osoba nebo právnická osoba a má možnost odběru zboží za zvýhodněné ceny. Každý *odběratel* musí mít unikátní *přihlašovací jméno*, nicméně jeho identifikátorem je uměle vytvořené *identifikační číslo*. Za předpokladu že dealerům budou přidělovány identifikátory menší než nějaká konstanta a zákazníkům větší, půjde u každé objednávky snadno zjistit zda je odběratel *dealer* či běžný *zákazník*.

Zboží je identifikováno umělým identifikátorem, přestože má atribut *kód*, který by měl být unikátní. Jeho unikátnost se však vztahuje jen na zboží v nabídce, zatímco objednávka může obsahovat zboží, které již v nabídce není, a proto je zapotřebí umělé identifikační číslo. *Kódem* zboží je myšlen čárový kód nebo nějaký textový řetězec (například FOT-20-89), který může obsahovat informace o výrobci a typu zboží.

Nejzajímavějším atributem *objednávky* je *stav*. Jak už název napovídá, tento atribut zachycuje stav objednávky (a odráží její putování informačním systémem). Nejprve je objednávka *nová*. Pokud jí lze uspokojit (zboží je k dispozici) změní se stav na *zpracovaná*. V ideálním případě je poté *zaplacená* a po předání zboží odběrateli *vyřízená*. V konkrétní implementaci mohou být přidány další stavy v závislosti na požadavcích firmy. Způsob reprezentace *stavu* (číslo, textový řetězec) závisí na implementaci a pro nás není v tuto chvíli důležitý.

Pokud byla již objednávka zaúčtována nelze její obsah již měnit. Pokud je změna nezbytná musí se provést prostřednictvím takzvaného *dobropisu* (neboli storno faktury). *Dobropis* si lze představit jako objednávku obsahující zboží se zápornou cenou.

Poslední zajímavou entitou je *Zaměstnanec*. *Zaměstnanec* reprezentuje osobu mající přístup k informačnímu systému. *Zaměstnanec* může mít libovolné množství *rolí* (a s nimi souvisejících *oprávnění*) a může být členem nějaké *skupiny*. Členstvím ve *skupině* získá všechny *role*, které daná *skupina* obsahuje. Tyto entity reprezentují systém rolí a skupin popsaný v části 3.3.

8 Genealogický strom (jeho tabulková reprezentace) má kromě reference na předka i takzvaný genealogický atribut. Genealogický atribut se skládá z prefixu, který tvoří genealogický atribut předka, a z, zpravidla jednopísmenného, identifikátoru uzlu. [17]

9 DFS (Depth First Search) reprezentace stromu obsahuje kromě reference na předka i další dva atributy – čas vstupu do vrcholu a čas „uzavření“ vrcholu. Tyto atributy vycházejí z algoritmu prohledávání stromu do hloubky (odtud tedy DFS strom).

4. Implementace prostředí

V této kapitole jsou popsány programy, které implementace potřebuje pro své fungování. Jedná se hlavně o webový a databázový server. U každého programu jsou uvedeny důvody, které k jeho výběru vedly, a možné alternativy. Stejným způsobem je popsán výběr programovacích jazyků, ve kterých jsou jednotlivé části programu vytvořeny.

V části 1.1 jsme definovali některé požadavky na informační systém. Tyto požadavky by měly splňovat i všechny programy, které naše implementace využívá. Budeme od nich požadovat:

- Nízkou cenu (nejlépe nulovou)
- Nízké hardwarové nároky (relativně – vzhledem ke konkurenci)
- Jednoduchou administraci (v porovnání s konkurencí)
- Fungování pod operačními systémy Windows a Linux (nejlépe nezávislost na platformě)

Největší důraz bude kladen na cenu.

4.1. Databáze

Jako primární databázová platforma bylo zvoleno PostgreSQL [3]. Hlavními důvody byla nízká pořizovací cena (je k dispozici zdarma), snadná administrace, bohatá dokumentace, existence verzí pro Windows i Linux, otevřený zdrojový kód a hlavně funkcionalita srovnatelná s komerčními databázovými systémy.

Druhou databázovou platformou, pro kterou byl program optimalizován, je Oracle. Důvody jsou velmi podobné těm, které byly popsány v předchozím odstavci. Nevýhodou je nedostupnost zdrojových kódů, trochu obtížnější administrace (což je daň za větší množství funkcí) a u vyšších verzí ne zrovna nízká cena (ale existuje odlehčená verze XE, která je k dispozici zdarma a pro použití v malých firmách je dostačující). Tyto dvě platformy by měly uspokojit požadavky naprosté většiny malých i středně velkých firem.

Dále byly zvažováno použití serverů MSSQL a MySQL. Prvně jmenovaný je určen výhradně pro platformu Windows a jeho cena je poměrně vysoká, což je v rozporu s našim požadavkem na minimální pořizovací náklady. Na rozdíl od něj je MySQL k dispozici zdarma. Bohužel až do verze 5, která byla uvolněna teprve nedávno, neměl MySQL podporu triggerů a uložených procedur a byl tedy pro naše účely nevhodný. Nyní sice tyto funkce podporuje, ale vzhledem k akvizici firmy InnoBase, hlavního tvůrce jádra databáze, společností Oracle může být tato podpora do budoucna ohrožena.

S databázovou platformou souvisí i výběr jazyka, ve kterém budou naprogramovány triggery, procedury a funkce (databázové). PostgreSQL umožňuje použít k tomuto účelu velké množství jazyků, za všechny jmenujme například C, Python, Javu a PLpg/SQL. A právě posledně jmenovaný je v programu RP použit. Důvodem je rychlost a rozumná přenositelnost – při použití jiné platformy (Oracle) je zapotřebí upravit pouze malou část kódu (konkrétně ošetření výjimek a volání specifických funkcí a procedur).

4.2. Aplikační server

Aplikační server je v této implementaci vlastně jen množina webových služeb. Tyto služby jsou tvořeny Java třídami umístěnými v servlet kontejneru Apache Tomcat [5] a dostupnými prostřednictvím frameworku Apache Axis [4](což pro je pro konzumenty těchto služeb naprosto nepodstatné). Toto řešení je možné provozovat jak na platformě Windows, tak i na Linuxu (+ několika dalších platformách) a tedy maximálně odpovídá našim cílům.

Webové služby dostaly přednost před architekturou CORBA hlavně díky své větší popularitě v rámci Internetu, což zvyšuje pravděpodobnost, že programy (informační systémy) zákazníků s nimi budou umět spolupracovat.

Apache Axis dostal přednost před Apache WS-SOAP díky lepší podpoře uživatelských objektů, vyšší rychlosti, možnosti generování WSDL souborů přímo z jednotlivých Java tříd a naopak vytváření Java tříd z WSDL souborů. Naopak výhodou WS-SOAP je lepší dokumentace (například [1]).

Kromě Tomcatu byly zvažovány aplikační servery JBoss a BEA WebLogic. Prvně jmenovaný je open source implementací J2EE aplikačního serveru. Nicméně podporu servletů[10] (a tedy i Apache Axis) obstarává integrovaný Apache Tomcat, tudíž jestliže nepotřebujeme využít některou z dalších J2EE technologií je lepší použít samostatný Tomcat. BEA WebLogic je zástupcem komerčních produktů a kvůli své pořizovací ceně je pro nás nevhodný.

Při výběru programovacího jazyka hrála roli jeho rozšířenost, dostupnost knihoven a dokumentace, rychlost a přenositelnost. Nejlépe dané požadavky splňovaly jazyky C++ a Java. Díky JIT (Just In Time) kompilaci do nativního kódu není Java o mnoho pomalejší než C++ a v její prospěch hovoří typová bezpečnost, lepší přenositelnost (kód je platformou ovlivněn minimálně) a celkově snazší a rychlejší vývoj aplikací. Proto byla jako programovací jazyk zvolena Java.

4.3. Internetový obchod

Internetový obchod je napsán v jazyce PHP (verze 4) a využívá webového serveru Apache [6]. Výhodou tohoto řešení je funkčnost na mnoha platformách, včetně Windows a Linuxu. Jazyk PHP byl zvolen kvůli jeho velké popularitě mezi webovými vývojáři a s tím souvisejícími nižšími náklady na programátorskou práci při úpravách (které na počátku budou jistě zapotřebí, protože současná verze poskytuje pouze ty nejzákladnější funkce).

Kromě zvolené kombinace Apache + PHP byly zvažovány ještě Microsoft IIS + ASP.NET a Apache Tomcat + JSP. Stejně jako databázový server MSSQL je i IIS k dispozici pouze pro operační systém Windows, což je, vzhledem k platformní nezávislosti ostatních částí našeho programu, pádný argument hovořící v jeho neprospěch.

Na rozdíl od IIS je Apache Tomcat k dispozici pro širokou paletu operačních systémů. Nejedná se však o klasický webový server (i když se tak dá rovněž použít), spíše se o něm hovoří jako o servlet kontejneru. Jazyk JSP (Java Server Pages) slouží ke snazší tvorbě servletů, na rozdíl od většiny ostatních jazyků používaných k dynamickému generování stránek není JSP kód přímo interpretován, nýbrž je pomocí JSP kompilátoru přeložen do binárního kódu. Výhodou je větší rychlost a menší

10 Servlet je speciální Java objekt, který dostává žádosti (requests) a na jejich základě generuje odpovědi (responses). Nejčastěji bývá využíván pro dynamickou tvorbu html stránek (podtřída HttpServlet).

zátěž serveru. Dalším argumentem hovořícím pro použití Apache Tomcat je jeho podpora podpora webových služeb prostřednictvím Apache Axis. Protože aplikační server v naší implementaci využívá služeb Tomcatu s Apache Axis, může se zdát výhodné použít ho i pro internetový obchod. Nicméně umístění obou částí na jeden server s sebou nese řadu nevýhod. Jednou z nich je omezení dostupnosti aplikačního serveru při velké zátěži webového serveru. Během restartu jednoho serveru jsou nedostupné oba, stejně tak při pádu Tomcatu. Výše uvedené argumenty, spolu s menší rozšířeností jazyka JSP mezi webovými vývojáři a s tím souvisejícími náklady na úpravu internetového obchodu, vedly k rozhodnutí nepoužít Apache Tomcat jako webový server (respektive nepoužít ho jako webový a zároveň aplikační server).

4.4. Skladový systém

Skladový systém[11] (v této práci rovněž označován jako klientská aplikace) je napsán v jazyce Java a ke komunikaci s aplikačním serverem využívá frameworku Apache AXIS. Důvody pro výběr jazyka Java jsou podobné důvodům uváděným v části věnované aplikačnímu serveru – přenositelnost, typová bezpečnost a rychlost vývoje.

Uživatelské rozhraní využívá knihovny *Swing*, která je součástí téměř každého běhového prostředí jazyka Java (Java Virtual Machine), a proto ji není nutné distribuovat spolu s aplikací.

Pro zobrazování grafů je použita knihovna *JFreeChart* [2], která je šířena pod licencí LGPL. Tato knihovna umožňuje snadno vytvářet všechny základní typy grafů, v programu je však použita pouze k tvorbě jednoduchých sloupcových grafů.

11 Někdy se jako skladový systém označuje kompletní řešení pro kompletní správu skladu (Warehouse Management System), který uchovává informace o fyzickém umístění zboží a proporcích skladu a zefektivňuje práci skladníků. V takovém případě lze pro náš program použít označení *Skladová evidence*

5. Implementace systému

Tato kapitola popisuje implementaci navrženého řešení (naleznete ji na přiloženém médiu). Zabývá se všemi částmi programu, nicméně důraz je kladem hlavně na aplikační server a klientskou aplikaci. Závěr je věnován možnostem rozšíření a úprav.

5.1. Databáze

Podrobný popis celého schématu lze nalézt v programátorské dokumentaci programu RP [13]. Dále se proto budeme zabývat jen jeho vybranými částmi.

Jedním z hlavních problému, které bylo zapotřebí vyřešit, je odstraňování zboží z nabídky. Jelikož se může stát (a to poměrně často), že odstraňované zboží bude v nějakém množství skladem či si ho někdo objednal a dosud nevyzvedl, nelze zboží jednoduše z dané tabulky smazat. V této implementaci je problém vyřešen přidáním tabulky *Vyprodej*, do které se takové zboží přesouvá. Dále je definován trigger, který při poklesu počtu kusů daného zboží skladem na nulu toto zboží odstraní z nabídky i z tabulky *Vyprodej*.

Další problémy přináší podpora více skladů. Je nutné zajistit, aby bylo možné uspokojit objednávku obsahující zboží, které se nachází v různých skladech. Každá firma má jiný názor na řešení tohoto problému (některé preferují jeden centrální sklad, ze kterého se zboží distribuuje do poboček, jiné potřebují vícero rovnocenných skladů, mezi kterými se zboží bude pravidelně přesouvat), nicméně je zapotřebí, aby systém obsahoval nějaké základní, ale alespoň částečně vyhovující, řešení. V naší implementaci je problém řešen pomocí určení centrálního skladu, do kterého se veškeré potřebné zboží přesune, nelze-li objednávku uspokojit z jednoho skladu (a nejedná-li se o osobní odběr, v kterémžto případě se samozřejmě zboží přesunuje do skladu v místě odběru). Nevýhodou je častější transfer zboží v případě několika rovnocenných skladů, výhodou naopak vysoká efektivita v případě pouze jednoho skladu (který se automaticky stává centrálním).

Se sklady souvisí i další otázka a sice co se zbožím, které již bylo vyskladněno, ale zákazník si ho dosud fyzicky nepřevzal. Řešením je zavedení virtuálního meziskladu, kam se takové zboží bude ukládat – v rámci databázového schématu k tomuto účelu slouží tabulka s příléhavým názvem *Mezisklad.* Tato tabulka je využita i v případě vytvoření dobropisu k nějaké faktuře – ještě než se zboží fyzicky vrátí do skladu je umístěno do meziskladu (aby souhlasily počty kusů).

Výše popsané problémy souvisí s firemní částí databázového schématu, která byla popsána v části 3.4. Kromě ní má schéma ještě systémovou část, která je popsána v následujících odstavcích.

Pro uložení informací o firmě (název, IČO, DIČ, adresa, …) slouží tabulka *Konstanty.* Obsahuje záznamy typu klíč-hodnota, které jsou doplněny o atribut udávající typ záznamu (například *firemní data*). Aplikační server si při své inicializaci tyto data načte do paměti a s tabulkou pracuje jen při jejich aktualizaci. Důvodem je malá velikost těchto dat a jejich časté využívání (například při generování faktur, potvrzovacích emailů, dobropisů, …).

V části 2.2 jsme požadovali, aby při úpravě a zrušení objednávky bylo možné dohledat iniciátora této akce. K tomu slouží tabulka *Log*, která obsahuje mimo jiné i záznamy o smazání a úpravách objednávek (ukládá se do ní kopie původní verze

objednávky ve formátu XML). U každého záznamu je uveden čas vytvoření, typ akce a identifikátor zaměstnance, který akci inicioval.

V tabulce *Settings* se ukládají uživatelská nastavení klientské aplikace (klientských aplikací). Díky tomu je chování aplikace závislé na přihlášeném uživateli a nikoliv na stroji, kde je nainstalována. Tabulka je navržená tak, aby ji mohla využít libovolná aplikace, protože záznamy jsou tvořeny trojicí uživatel, klíč, textová hodnota (například XML). V naší implementaci se jako hodnota ukládají serializované objekty zakódované do Base64. Výhodou je jednoduchost implementace (daný objekt musí implementovat prázdné rozhraní *Serializable*), nevýhodou jsou větší nároky na prostor.

Poslední důležitá tabulka patřící do systémové části schématu má název *Templates*. Obsahuje textové šablony pro generování faktur, potvrzovacích emailů, dobropisů a dalších dokumentů. O způsobu generování těchto dokumentů se podrobněji zmíníme v části 5.5.

5.2. Aplikační server

Na rozdíl od řešení navrženého v předchozí kapitole je aplikační server rozdělen pouze do dvou podvrstev – aplikační logika a prezentace dat klientům jsou sloučeny dohromady. Je to dáno tím, že aplikační logiky obsahuje server zatím jen minimum, protože, jak už název napovídá, aplikační logika je silně závislá na konkrétní firmě, která bude program používat. Spodní vrstvu sloužící k přístupu k databázi tvoří třída *DBConnect*, která „obaluje“ JDBC ovladač a vytváří spojení s databází.

Další rozdíl je v autorizačním mechanismu. Zatímco v předchozí kapitole jsme definovali *skupinu oprávnění* jako množinu uživatelů se stejnými právy, zde se chápe jako množina rolí. Původně navržené řešení (každý uživatel je členem nejvýše jedné skupiny) je tedy speciálním případem toho implementovaného. Hlavním důvodem pro tuto změnu je usnadnění vývoje a administrace klientských aplikací.

Minimální jednotkou oprávnění je metoda webové služby, role jsou tedy množinou metod, které smí aplikace daného uživatele volat. Skupiny nám umožňují role skládat a přidělovat je uživatelům po větších kvantech. To je důležité pro usnadněni administrace klientských aplikací, které nezřídka potřebují pro provedení jedné operace volat vícero metod – tedy jedna operace odpovídá roli (jako příklad nám může posloužit úprava objednávky – je nutné mít oprávnění k získání detailů objednávky, oprávnění zamknout jí proti úpravám a oprávnění změnit její detaily).

S autorizací souvisí jeden bezpečnostní problém – zaměstnanec přidělující oprávnění (například vedoucí oddělení) by při špatné implementaci mohl sám sobě libovolně zvyšovat práva. Nejjednodušším řešením by bylo umožnit přidělování oprávnění pouze administrátorovi, který by směl provádět libovolné operace (maje maximální oprávnění, nemohl by si ho samozřejmě zvyšovat). Problém však nastává v případě jeho nedostupnosti (pokud by byl v pracovní neschopnosti či na dovolené) a dá se vyřešit jen přidáním dalšího administrátora se stejným oprávněním. A co v případě, že není k dispozici ani jeden z nich? Toto řešení není tedy pro malé firmy vhodné. V naší implementaci je přidělování oprávnění možné jen za následujících podmínek:

a) Administrátor, mající množinu oprávnění M, smí zaměstnanci přidělit roli

A pokud:

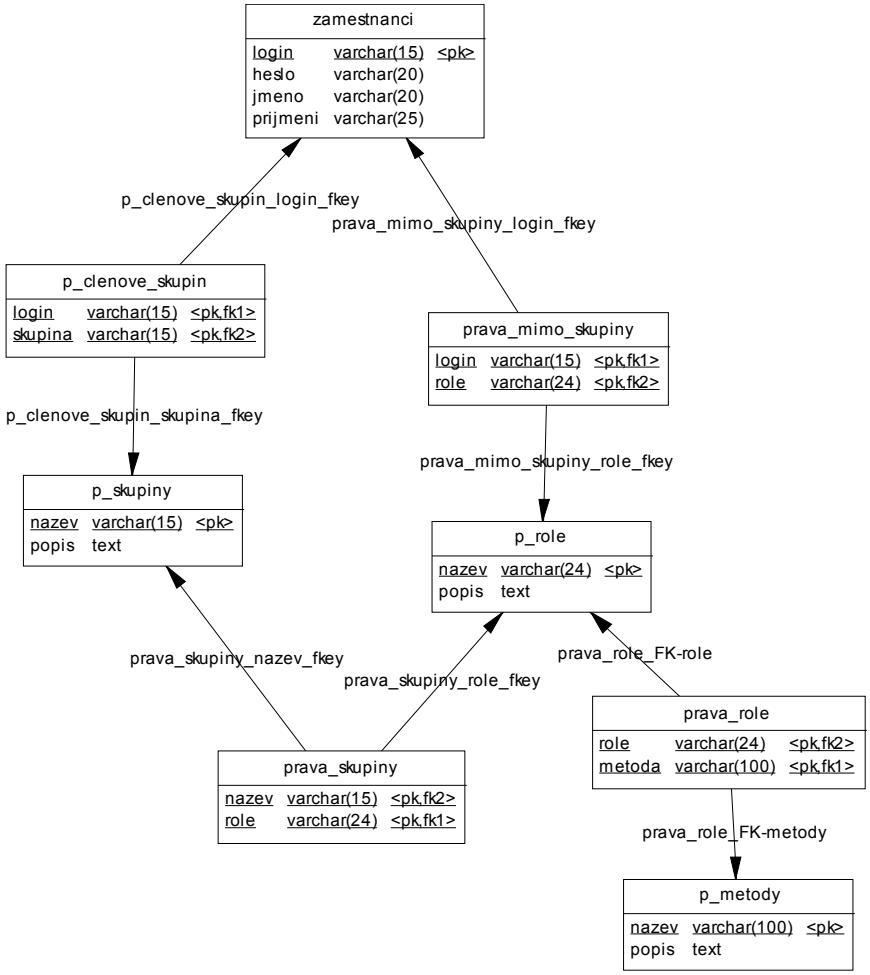
1) Má právo přidělovat role uživatelům
2) $\forall o \in A : o \in M$

b) Administrátor, mající množinu oprávnění *M*, smí zaměstnanci přidělit členství ve skupině Φ pokud:
   1) Má právo přidělovat uživatelům členství ve skupinách
   2) $\forall A \in \Phi : A \subseteq M$

c) Administrátor, mající množinu oprávnění *M*, smí do role *A* přidat oprávnění *o* jestliže:
   1) Má právo upravovat role
   2) Má oprávnění *o* ($o \in M$)

d) Administrátor, mající množinu oprávnění *M*, smí do skupiny Φ přidat roli *A* jestliže:
   1) Má právo upravovat skupiny
   2) $\forall o \in A : o \in M$

Z těchto pravidel jasně vyplývá, že si nikdo nemůže sám přidělit oprávnění, které dosud neměl. Pro odejmutí oprávnění platí obdobná pravidla. Na obrázku 5.3 je vidět schéma databázové části implementace tohoto řešení.



Obrázek 5.1.: Schéma části databáze obsahující zaměstnance, role a skupiny

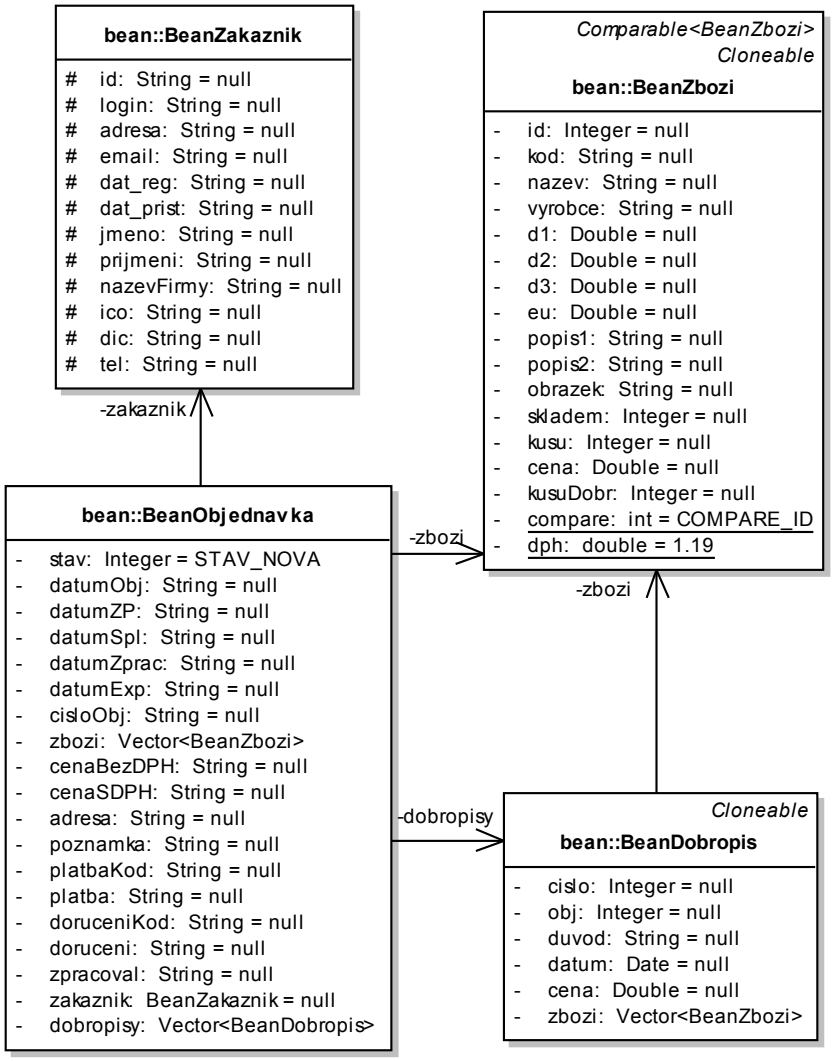
Aplikační server je tvořen třemi základními Java třídami – *app.Client*, *web.WebClient* a *web.Kategorie*. Název příslušné webové služby odpovídá malými písmeny napsanému plnému názvu třídy s prefixem *urn:* a s dvojtečkami namísto teček (tedy například *urn:app:client*).

Třída *Kategorie* existuje v rámci aplikačního serveru pouze v jedné instanci a jejím hlavním úkolem je udržování stromu kategorií zboží. Nejčastější operací je

získání linearizované verze (ekvivalentní průchodu do hloubky) tohoto stromu, konkrétně pomocí metody *getKategorie*. Tato metoda vyžaduje pouze čtení jednoho atributu, a proto je bezpečná vůči vícenásobnému přístupu z různých vláken.

WebClient je třída určená pro komunikaci s internetovým obchodem. Každý uživatel má k dispozici jednu její instanci.

Pro komunikaci s klientskou aplikací (program RP) slouží třída *Client*. Stejně jako v předchozím případě má každý uživatel (klientská aplikace) k dispozici jednu instanci této třídy. Po přihlášení uživatele dojde mimo jiné k načtení seznamu jeho oprávnění a poté se při volání libovolné metody kontroluje, zda je k jejímu volání uživatel oprávněn. Jednou z nejdůležitějších funkcí třídy *Client* je správa objednávek a jejich zámků. Pokud má uživatel dostatečné oprávnění, může si zamknout libovolnou objednávku. Poté ji až do odemčení (nebo do vypršení stanoveného časového limitu, který je standardně jedna hodina) nemůže nikdo jiný upravovat, smazat a dokonce ani prohlížet. Díky tomu se nemůže stát, že by jedna objednávka byla zpracována dvakrát nebo že by byla změněna dvěma uživateli najednou.



Obrázek 5.2.: Diagram tříd zobrazující část datového modelu

Pokud potřebuje aplikační server (třída *Client*) předat klientské aplikaci nějaká složitější data (nebo je od ní naopak získat), může to provést dvěma základními způsoby. Buď využije pole (případně vícerozměrné pole) objektů odpovídajících

primitivním datovým typům, nebo vytvoří instanci nějaké k tomuto účelu vytvořené třídy. První způsob je méně náročný na implementaci na straně serveru (není nutné upravovat deployment deskriptory webových služeb a přidávat do nich mapování uživatelských objektů), ale je mnohem náročnější na implementaci na straně klienta, protože je nutné přesně vědět, co který prvek pole vyjadřuje a jakého je typu. Mnohem výhodnější je použití speciální třídy s bezparametrickým konstruktorem a getter/setter metodami pro každý atribut. Při vytvoření takovéto třídy je sice nutné změnit deployment deskriptory služeb, které budou tuto třídu využívat, a provést jejich redeploy, ale poté si ji může programátor klientské aplikace automaticky vygenerovat z WSDL popisu příslušné webové služby. V naší implementaci jsou tyto třídy umístěny v balíčku *bean* a dodržují jmennou konvenci *BeanXXX*, kde *XXX* je název entity, kterou reprezentují. Na obrázku 5.2 je vidět třída reprezentující objednávku a její vztah k třídám reprezentujícím dobropis, zboží a odběratele. Pro přehlednost jsou zobrazeny jen atributy tříd a nikoliv jejich metody.

5.3. Internetový obchod

Pro komunikaci s aplikačním serverem (tedy pro volání webových služeb) se využívá knihovna NuSOAP, konkrétně objekt *SoapClient*, jehož instance se uchovává v rámci PHP session. Žádné další informace se v rámci session neukládají.

Uživatelské rozhraní je stejné jak pro obyčejné zákazníky tak pro dealery (samozřejmě s výjimkou formuláře pro změnu registračních údajů, kde mají dealeři možnost zadat IČO, DIČ a další podrobnosti). Vlevo se nachází navigační menu, jehož většinu tvoří seznam kategorií zboží. Tento seznam má stromovou strukturu, přičemž jednotlivé větve lze libovolně skrývat a opět zobrazovat. Toto je řešeno pomocí JavaScriptu, proto není nutné pro výběr konkrétní kategorie stránku načítat vícekrát, jak je obvyklé u většiny ostatních webových obchodů. Po nahrání celého menu se, s výjimkou větve obsahující aktuálně vybranou kategorii, skryjí všechny kategorie, které nejsou bezprostředně pod kořenem. To je opět řešeno pomocí JavaScriptu, tudíž zákazníci používající prohlížeč s vypnutým JavaScriptem uvidí menu kompletně rozbalené (a tedy použitelné).

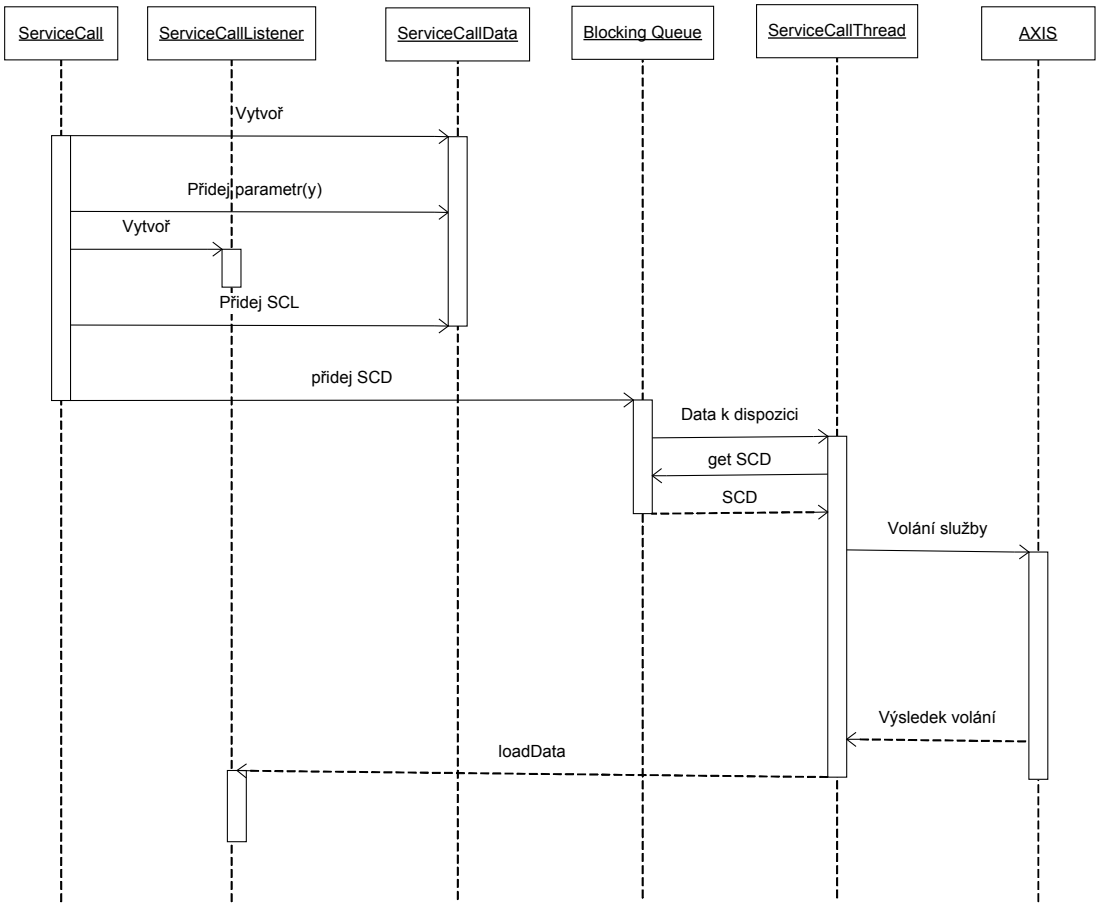
Vygenerovaný HTML kód je v souladu s normou HTML 4.01 Strict (je takzvaně validní). Použité CSS styly jsou rovněž v souladu s doporučeními W3C konsorcia.

5.4. Skladový systém

Uživatelské rozhraní je tvořeno jedním hlavním oknem a 0 až N vnitřními okny (čemuž se někdy říká MDI – Multiple Document Interface). V levé části se nachází dynamicky generované navigační menu (*JTree*), nahoře je hlavní menu (*JMenu*), pod ním se nachází řádek se seznamem otevřených vnitřních oken a dole je zatím nepříliš využitý informační řádek (obsahuje jméno aktuálně přihlášeného uživatele). Většinu plochy zabírá kontejner (*JDesktopPane*) obsahující vnitřní okna (*JFrame*). Vnitřní okna mají oproti modálním dialogům, které jsou používánu u některých konkurenčních produktů, velkou výhodu v možnosti přerušit práci na jednom úkolu a věnovat se jinému (například při zadávání nového zboží zavolá zákazník s dotazem na dostupnost nějakého produktu).

Aby uživatelské rozhraní zůstalo interaktivní je komunikace s aplikačním serverem realizována ve zvláštním vlákně *ServiceCallThread*. Synchronizace s hlavním vláknem je řešena metodou producent-konzument pomocí sdílené prioritní

fronty. Pokud je fronta prázdná je vlákno *ServiceCallThread* zablokováno, pokud je fronta plná a hlavní vlákno se do ní pokouší přidat další požadavek je generována výjimka (protože to zpravidla znamená přerušení komunikace s aplikačním serverem, či jeho zahlcení). Postup volání webové služby je znázorněn na obrázku 5.3.

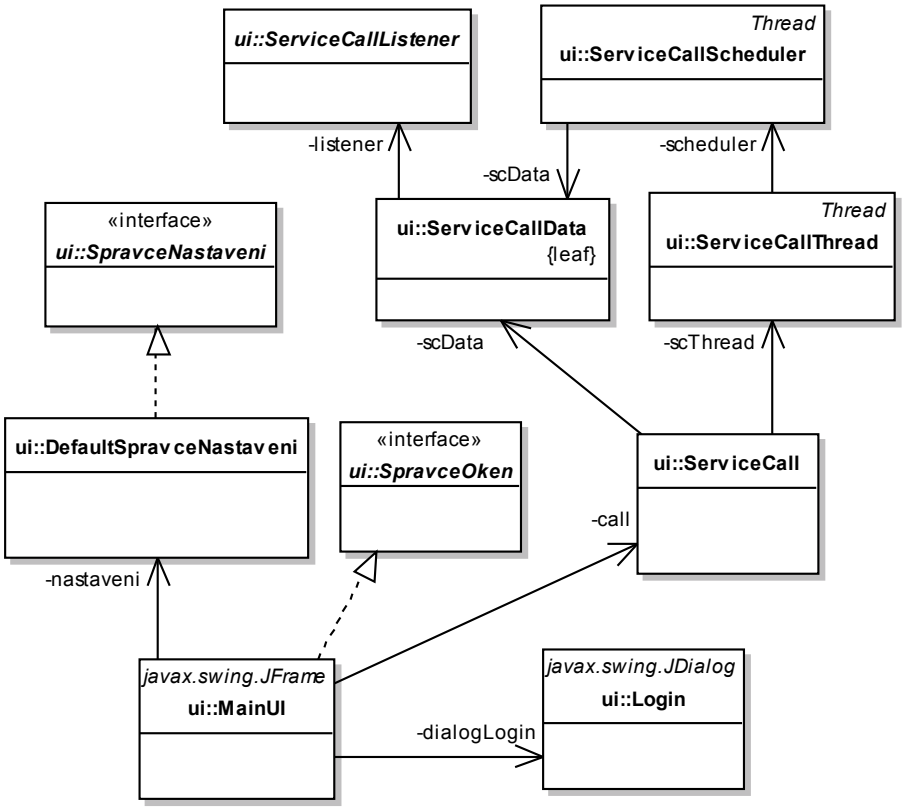


Obrázek 5.3.: Volání webové služby

Kromě hlavního vlákna generuje požadavky na komunikaci s aplikačním serverem ještě vlákno *ServiceCallScheduler*, které v zadaných intervalech přidává do fronty požadavek na volání testovací služby. To je nutné pro udržování spojení s aplikačním serverem, v případě, že uživatel dlouho nevyvolal žádnou akci, která by se serverem potřebovala komunikovat (v takovém případě by došlo k automatickému odhlášení uživatele a při dalším požadavku by byla generována výjimka *UserNotLoggedException*). Dané vlákno se dá využít i k takzvanému pollingu (pravidelnému dotazování serveru) v případě, že chceme aby klient reagoval na nějakou událost, která nastala na straně serveru. V současné implementaci vlákno přidá do fronty požadavek pouze v případě, že po zadanou dobu nedošlo k volání webové služby.

Na obrázku 5.4 jsou znázorněny všechny výše uvedené třídy související s voláním webových služeb. Pro přehlednost nejsou u tříd uvedeny jejich atributy ani metody. Dále je na obrázku 5.4 vidět hlavní třída uživatelského rozhraní – *MainUI*, která implementuje dvě důležitá rozhraní – *SpravceOken* a *SpravceNastaveni*. Prvně jmenované rozhraní slouží vnitřním oknům ke komunikaci s hlavním oknem (jedná se o povinný parametr jejich konstruktoru). Kromě správy vnitřních oken má

SpravceOken za úkol správu nápovědy (přiřazuje jednotlivým heslům nápovědy konkrétní komponenty a vnitřní okna) a obsahuje rovněž metody pro získání implementace *SpravceNastaveni* a objektu *ServiceCall*. *SpravceNastaveni* slouží, jak už název napovídá, ke správě uživatelských nastavení. Při inicializaci získá od aplikačního serveru seznam dvojic klíč-hodnota (hodnotou je serializovaný objekt zakódovaný do Base64), provede deserializaci získaných objektů, dvojice si uloží do hašovací tabulky a poté na požádání vydává objekty odpovídající zadaným klíčům.



Obrázek 5.4.: UML class diagram třídy MainUI

Důležitou funkcí je nahrávání nápovědy z webového serveru (uvedeného v konfiguračním souboru), což umožňuje pro uživatele zcela transparentní aktualizace a usnadňuje to administraci celého systému (společně s dalšími funkcemi, které budou popsány v následující části).

5.5. Rozšířitelnost

Úpravy programu rozdělíme na dvě části:

1. Customizace – počáteční úprava podle potřeb zákazníka. Vzhledem ke snaze o maximální obecnost použití programu a rozdílným požadavkům firem, bude tato úprava vždy nutná a někdy i dosti rozsáhlá.
2. Evoluce – úpravy odrážející změnu požadavků zákazníků. Tyto změny jsou většinou méně rozsáhlé a hlavně rozložené v čase (na rozdíl od customizace, která je jednorázová). Jako příklad lze uvést nedávno zavedený poplatek za elektroodpad, který některé firmy účtují odděleně od ceny zboží (a tedy musely upravit svůj informační systém, aby toto umožňoval)

Dále můžeme úpravy rozdělit podle částí systému, které postihují:

A) Úpravy databáze
   A.1. Úpravy databázového schématu
   A.2. Změna databázové platformy
B) Úpravy aplikačního serveru
   B.1. Úprava vzhledu faktur, objednávek a automaticky generovaných emailů
   B.2. Změna mechanismu vzdáleného volání procedur (RPC)
   B.3. Přidávání služeb
   B.4. Úprava stávajících služeb
C) Úpravy webového obchodu
   C.1. Změna mechanismu vzdáleného volání procedur (RPC)
   C.2. Využití hostingových služeb
   C.3. Změna skriptovacího jazyka
D) Úpravy klientské aplikace
   D.1. Změna mechanismu vzdáleného volání procedur (RPC)
   D.2. Přidávání nových oken a úprava stávajících

A) Úpravy databáze

A.1 Úpravy databázového schématu

Nejjednodušší úpravou je přidání nového sloupce (atributu) do nějaké tabulky. V tomto případě zůstane funkčnost celé aplikace zachována beze změny (pokud zároveň nedojde k úpravě některých služeb, aby nový sloupec využívaly, což je samozřejmě žádoucí), protože všechny SQL příkazy explicitně vyjmenovávají použité sloupce (nevyskytuje se zde například *SELECT * FROM tabulka*, nebo *INSERT INTO tabulka VALUES...*) a tedy se jich změna tabulky nedotkne.

Poněkud náročnější úpravou je sloučení více tabulek do jedné, či naopak rozdělení jedné tabulky do více. V tomto případě je buď zapotřebí přepsat všechny SQL příkazy využívající dané tabulky, což je velmi pracné, nebo vytvořit pohled(y) (view) simulující staré schéma. Pokud zvolíme druhou možnost je ještě nutné dodefinovat operace INSERT, UPDATE a DELETE a to buď pomocí RULES, v případě že se jedná o PostgreSQL, nebo pomocí INSTEAD OF triggerů, pokud je databázovou platformou Oracle (či jinak, pokud se nejedná ani o žádnou z výše uvedených). Postupem času mohou být SQL příkazy využívající staré schéma přepsány tak, aby odpovídali novému schématu, a odpovídající pohledy mohou být odstraněny.

A.2 Změna databázové platformy

Při tvorbě programu, bylo počítáno se dvěma databázovými platformami – PostgreSQL (primární) a Oracle. Při přechodu z PostgreSQL na Oracle je nutné udělat následující věci:

- Změnit typ sloupce *TEXT* na *CLOB* a upravit příslušné SQL příkazy, které s danými sloupci pracují
- Převést PLpg/SQL funkce a triggery na Oracle PL/SQL – nutno změnit hlavně ošetření výjimek

- Neúplné SELECT dotazy doplnit o „FROM dual“

Při přechodu na jinou platformu budou změny pravděpodobně rozsáhlejší, pokud vůbec bude přechod možný. Základní požadavky na databázovou platformu jsou:

- Kompatibilita s normou SQL-92
- Podpora triggerů
- Existující JDBC ovladač

B) Úpravy aplikačního serveru

Veškeré úpravy je vhodné provádět nejprve na testovacím systému.

B.1 Úprava vzhledu faktur, objednávek a automaticky generovaných emailů

Vzhledem k využití jazyka Jython[12] jsou tyto úpravy značně usnadněny. Stačí pouze přepsat příslušnou šablonu v tabulce *Templates*, případně přidat novou šablonu a pozměnit konfiguraci serveru tak, aby využíval tuto nově přidanou šablonu. Drobné úpravy lze provádět i bez znalosti jazyka Jython, protože převážnou část šablon tvoří HTML značky (tagy).

B.2 Změna mechanismu vzdáleného volání procedur (RPC)

Na straně serveru je tato změna velmi snadná a nevyžaduje zásah do zdrojových kódů. Stačí jen zaregistrovat stávající třídy aplikačního serveru způsobem, který je vyžadován novým mechanismem (například *deploy* pomocí nástroje *AdminClient* u *Apache AXIS*)

B.3 Přidávání služeb

V případě, že je z hlediska architektury systému výhodnější přidat novou službu, namísto úpravy stávající, je zapotřebí udělat následující kroky (předpokládáme použití Apache AXIS):

- Zkompilovat příslušnou třídu (a všechny potřebné třídy) a výsledek umístit do *$AXIS_HOME/WEB-INF/Classes* při zachování adresářové struktury balíčků (packages). Případné potřebné knihovny umístit do *$AXIS_HOME/WEB-INF/lib*
- Vytvořit deployment descriptor[13] nové služby:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
 mlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <service name="urn:app:client" provider="java:RPC">
  <parameter name="className" value="app.Client"/>
  <parameter name="allowedMethods" value="*"/>
  <parameter name="scope" value="Session"/>
      <beanMapping qname="myNS:Login" xmlns:myNS="urn:app"
      languageSpecificType="java:bean.BeanLogin"/>
 </service>
</deployment>
```

Příklad 1: Deployment descriptor webové služby

12 Skriptovací jazyk na bázi jazyka Python. www.jython.org

13 Pro detailní popis formátu deployment deskriptoru navštivte domovské stránky projektu Apache Axis: http://ws.apache.org/axis

Zajímavé jsou zejména tagy *service*, jehož atribut *name* určuje jméno služby, a *beanMapping*, který slouží k registraci uživatelských typů parametrů a návratových hodnot.

- Provést deployment služby pomocí utility *org.apache.axis.client.AdminClient*, které jako parametr předáme název souboru s výše uvedeným deployment descriptorem.
- Otestovat funkčnost služby testovacím voláním.

B.4 Úprava stávajících služeb

Tato úprava bude pravděpodobně nejčastější, proto je dobrou zprávou, že není nikterak náročná. Po kompilaci nahrajeme třídy do adresáře *$AXIS_HOME/WEB-INF/Classes*, přičemž je nutné zachovat adresářovou strukturu danou balíčky jazyka Java. Poté již jen stačí restartovat server, redeploy služby není zapotřebí.

C) Úpravy webového obchodu

Webový obchod je místem, kde se mohou firmy odlišit od konkurence, proto jsou úpravy nanejvýš pravděpodobné a žádoucí.

C.1 Změna mechanismu vzdáleného volání procedur (RPC)

Pokud chceme zachovat stávající kód bez úprav, je nutné dopsat „proxy“ třídu s metodou *call(jméno metody, parametry, jméno služby)*, která bude simulovat starý mechanismus. Touto třídou je zapotřebí inicializovat proměnnou session jménem *client* v souboru *session_soap.php* (místo třídy *soapclient*).

C.2 Využití hostingových služeb

Bohužel pro některé firmy je kvalitní připojení k Internetu zatím nedostupné (ať již finančně či geograficky), a proto nemohou využít webový obchod ve stávající podobě. Jak už to tak bývá, je několik způsobů, jak tento problém vyřešit.

První možností je problém neřešit a webový obchod nevyužívat (objednávky se musí přijímat nějakou jinou cestou – telefonicky, emailem, poštou či osobně). Toto je nejlevnější a v mnohých případech i nejefektivnější řešení (hlavně u drobných podnikatelů).

Druhý způsob je z architektonického hlediska nejzajímavější, protože předpokládá dopsání druhého aplikačního serveru, který se bude s tím primárním synchronizovat za pomoci emailů (objednávky, registrace nových zákazníků) a pomocí nějakého mechanismu vzdáleného volání procedur (ve směru od primárního, pokud bude dostupná linka). Nový aplikační server bude spolu s webovým obchodem umístěn na serveru poskytovatele hostingových služeb a bude mít přístup k vlastní databázi, oddělené od té, kterou využívá hlavní část aplikace. Primární aplikační server bude nutné upravit tak, aby všechny změny od poslední synchronizace ukládal a při dostupnosti linky je předal novému serveru.

Třetí způsob se podobá druhému jen s tím rozdílem, že webový obchod bude kompletně přepsán tak, aby vyhovoval novým požadavkům. Bude mít vlastní databázi, nikoliv však vlastní aplikační server (půjde tedy o klasickou dvojvrstvou architekturu, model klient-server). Primární (a vlastně jediný) aplikační server se při dostupnosti linky postará o synchronizaci databází – stáhne si nové objednávky a nově registrované zákazníky a uloží veškeré změny, provedené od poslední synchronizace. Tento způsob je o něco jednodušší než předchozí a navíc umožňuje

snížit počet funkcí (například zobrazování počtu kusů skladem) webového obchodu a tím celý proces zjednodušit.

C.3 Změna skriptovacího jazyka

Tato změna sebou nese (s jedinou výjimkou) nutnost přepsání celého webového obchodu a je tedy vhodná pouze pokud jsou požadovány zásadnější změny (které by vyžadovaly úpravu většiny zdrojových kódů). Jako nejvhodnější náhrada PHP se jeví JSP (Java Server Pages). Nevýhodou tohoto jazyka je nižší rozšířenost mezi webovými vývojáři a s ní související vyšší cena za práci.

Další možností je zachování jazyka PHP s tím, že generování většiny kódu bude přesunuto na stranu aplikačního serveru – konkrétně na Jython skripty. Největší výhodou tohoto řešení je možnost postupného nahrazování starého kódu při zachování funkcionality celého systému.

D) Úpravy klientské aplikace

Díky tomu, že na jednom počítači může být umístěno libovolné množství těchto aplikací (a každá může být spuštěna v několika instancích), je možné jakékoliv úpravy důkladně otestovat bez zásahu do produktivního prostředí (testovacím verzím je možné v konfiguračních souborech nastavit cestu k testovacímu aplikačnímu serveru, který pracuje s testovací databází).

D.1 Změna mechanismu vzdáleného volání procedur (RPC)

Při změně mechanismu RPC je nutné pouze přepsat třídu *ServiceCall*, konkrétně metody *addParam*, *addNullParam*, *addMapping* a *call*. Je vhodné zachovat volání procedur ve zvláštním vlákně, rozhodně by se toto volání nemělo provádět ve stejném vlákně, ve kterém běží uživatelské rozhraní (kvůli zachování rozumné odezvy).

V průběhu vývoje programu již jedna takováto změna proběhla a sice přechod z Apache WS-SOAP na Apache AXIS. Změny v kódu a otestovaní programu zabraly dohromady zhruba 4,5 člověkohodin.

D.2 Přidávání nových oken a úprava stávajících

Protože i v malé firmě může být nainstalován větší počet klientských aplikací, jejichž ruční aktualizace by zbytečně zatěžovala administrátora (či administrátory) systému, umožňuje program provádět aktualizace centrálně. Aktualizační proces je popsán v následujících odstavcích.

Ze všeho nejdříve je zapotřebí vytvořit alespoň dvě třídy – první bude potomkem *JInternalFrame* a bude obsahovat nové či upravené vnitřní okno aplikace, druhá třída bude potomkem *NavMenuItem* a bude obstarávat zobrazení okna při požadavku uživatele (například výběru z menu). Tyto dvě třídy zabalíme do jar souboru, který umístíme na nějaký, z klienta dostupný, webový server. Aby klient začal tyto třídy

```
<modules>
 <module name='Konstanty' type='menu'>
  <class  nazev='NavMenuKonstanty'  name='modules.NavMenuKonstanty'
      path='http://localhost:8080/modules.jar' />
 </module>
</modules>
```

Příklad 2: Soubor modules.xml

využívat, je nutné je ještě zaregistrovat v konfiguračním souborech *modules.xml* a *menu.xml*. Prvně zmiňovaný soubor obsahuje definici takzvaných modulů, což jsou jednoduše řečeno jar soubory obsahující zkompilované Java třídy. Příklad 2 obsahuje jednoduchou verzi souboru *modules.xml*

V současné době jsou podporovány pouze moduly typu *'menu'*, které rozšiřují funkčnost aplikace přidáváním či změnou vnitřních oken. Atribut *nazev* je úzce svázán s tvorbou menu (viz. dále), atribut *name* obsahuje kvalifikovaný název třídy (tedy včetně všech názvů balíčků v klasické tečkové notaci), která je potomkem *NavMenuItem*, a konečně atribut *path* udává umístění binárních souborů všech potřebných tříd.

Souborů *modules.xml* může existovat libovolné množství (kladné) a mohou rovněž mít libovolná jména. Jedinou podmínkou je zadání názvu a adresy některého z těchto souborů do konfiguračního souboru každého z klientů (*settings.xml* - položka *MODadr*). Tímto způsobem lze snadno dosáhnout různých úprav v různých klientech (například ve skladu mohou mít okna přizpůsobena malému dotykovému displeji, zatímco v obchodním oddělení budou optimalizována na rozlišení tamních devatenácti palcových panelů).

Dalším souborem, který je zapotřebí upravit je *menu.xml*. Tento konfigurační soubor slouží ke generování hlavního (nahoře) a navigačního (vlevo) menu klientské aplikace. Detailní popis formátu souboru naleznete v programátorské dokumentaci programu RP [13]. V příkladu 3 je vidět soubor *menu.xml* obsahující jediné menu se dvěma položkami.

```
<data>
 <menu lang='cz'>
  <submenu name='Objednávky'>
   <item name='Hledat' nazevTridy='NavMenuObjHledat'
      method='findNewObj' />
   <item name='Nová' nazevTridy='NavMenuObjNova' method='newObj' />
  </submenu>
  <submenu name='Zboží'>
   <submenu name='Seznam' call='makeKatZbozi' />
  </submenu>
 </menu>
</data>
```

Příklad 3: Soubor menu.xml

Důležitý je zejména atribut *nazevTridy* (tagu *item*), který odkazuje na atribut *nazev* v souboru *modules.xml*. Atribut *method* slouží pro určení, zda se konkrétnímu uživateli daná položka zobrazí či nikoliv (musí mít práva volat metodu, která je hodnotou tohoto atributu).

Pokud chceme zaregistrovat nově vytvořené třídy, přidáme řádek (tag *item*) do příslušného submenu, případně vytvoříme nové submenu. Chceme-li nějaké okno nahradit novější verzí, upravíme pouze u příslušné položky atribut *nazevTridy* tak, aby odpovídal názvu, který jsme vyplnili v souboru *modules.xml*)

Nakonec je zapotřebí restartovat aplikační server, aby se změny v souboru *menu.xml* projevily. U klientů se změny projeví rovněž po restartu.

6. Srovnání s konkurencí

Tato kapitola se zabývá vybranými konkurenčními produkty a srovnává je s implementací popsanou v předchozích kapitolách. Závěr je věnován možnosti reálného nasazení ve firmě.

Na trhu je velké množství programů, které bychom mohli označit za konkurenční. Z této rozsáhlé množiny byly vybráni tři kandidáti, kteří poslouží pro srovnání s naším řešením. Jedná se o programy *ABC Prodeje* verze 4.1.9 Trial [15], což je komerční skladový systém určený pro velmi malé firmy, *tinyERP* [7] (kompletní manažerský informační systém) a *osCommerce* [8], jenž je zástupcem internetových obchodů.

Program *ABC Prodeje* je jediným zástupcem komerčních programů v našem srovnání, ale jeho cena není tak vysoká (999 Kč bez DPH). Na rozdíl od našeho programu se jedná o monolitickou aplikaci (uživatelské rozhraní není odděleno od dat a logiky), určenou výhradně pro platformu Windows. Jeho výhodou jsou nízké nároky na hardware (Pentium 133, 64 MB RAM), podrobná nápověda, jednoduchá instalace a velké množství funkcí. Naopak podstatným mínusem je uživatelské rozhraní založené na modálních dialozích, obtížnější administrace (pro zálohování je zapotřebí ručně ukončit všechny instance programu i v rámci sítě) a celkově méně komfortní ovládání. V testované verzi se rovněž vyskytly problémy s českou diakritikou (v programu i v datech).

Srovnáme-li náš skladový systém s programem *ABC Prodeje* zjistíme, že náš produkt je uživatelsky přívětivější, snadněji spravovatelný, má lepší architekturu a možnosti rozšiřování a je platformě nezávislý. Oproti tomu *ABC Prodeje* má více funkcí, menší hardwarové požadavky a kvalitnější nápovědu.

Stejně jako naše implementace je i *tinyERP* k dispozici zdarma. Dalšími společnými rysy jsou: otevřený zdrojový kód, nezávislost na platformě (je napsán v jazyce Python), třívrstvá architektura, dobrá rozšiřitelnost a využití databázového serveru PostgreSQL. Velkou výhodou *tinyERP* je kvalitně zpracovaná uživatelská dokumentace, která dokonce obsahuje instruktážní videa. Naopak nevýhodou je pomalá práce programu (což je dáno použitým jazykem), absence volné verze programátorské dokumentace a méně ergonomické uživatelské rozhraní (používá místo vnitřních oken panely).

Při srovnání s naším skladovým systémem zjistíme, že *tinyERP* má více funkcí, lepší uživatelskou dokumentaci a je celkově propracovanější. Na druhou stranu má horší uživatelské rozhraní a je pomalejší.

Ani jeden z výše uvedených konkurentů nenabízel možnost prodeje zboží přes vlastní internetový obchod, a proto byl do srovnání zařazen program *osCommerce*. Jedná se široce používaný open source internetový obchod vytvořený v jazyce PHP. Využívá databázi MySQL, do které přistupuje přes vlastní funkce, které jsou namapovány na ty knihovní. Díky tomu je možné změnit databázovou platformu bez větších zásahů do kódu. Bohužel *osCommerce* nepočítá s napojením na aplikační server (proto by bylo nutné buď přepsat podstatnou část zdrojových kódů tohoto obchodu nebo v rámci aplikačního serveru vytvořit službu, která by synchronizovala obsah databází). Jeho architektura je tedy pouze dvojvrstvá. Další nevýhodou je nutnost znovu načíst celou stránku při výběru kategorie.

Náš internetový obchod má oproti *osCommerce* lépe vyřešen výběr kategorií zboží a dokáže spolupracovat s aplikačním serverem. Na druhou stranu *osCommerce* má více funkcí a je vizuálně přitažlivější.

Vlastnosti výše uvedených produktů shrnuje Tabulka 1.

Tabulka 1: Srovnání programu RP s konkurencí

	RP	***ABC Prodeje***	***tinyERP***	***osCommerce***
Cena (Kč)	0	999	0	0
Architektura	třívrstvá	jednovrstvá	třívrstvá	dvojvrstvá
Platforma	Nezávislé	Windows	Nezávislé	Nezávislé
Rozšiřitelnost	Výborná	Nedostatečná	Výborná	Velmi dobrá
HW nároky	Střední	Nízké	Vyšší	Střední

Je nutné zdůraznit, že účelem této práce nebylo vytvoření komerčně úspěšné aplikace, která by konkurovala výše uvedeným produktům. Přesto je reálné nasazení vytvořeného programu možné (v malých firmách, které ještě nepoužívají žádné jiné řešení), ale až po nutných úpravách, které zahrnuji:

- Doplnění aplikační logiky podle potřeb firmy
- Integraci účetního systému
- Úpravu vzhledu internetového obchodu a přidání některých funkcí (zobrazení již odeslaných objednávek)
- Doplnění nápovědy a vytvoření materiálů pro výuku práce s programem (pro počáteční zaškolení uživatelů)

7. Závěr

Zhodnocení naplnění cílů stanovených v úvodní kapitole.

V úvodní kapitole jsme si stanovili cíle, kterých chceme v rámci této práce dosáhnout, a proto je vhodné se na závěr zamyslet, zda se je podařilo naplnit.

Ve druhé kapitole jsme se zabývali analýzou potřeb malé (až středně velké) firmy, definovali jsme aktéry a operace, které budou nejčastěji prováděny, čímž jsme splnili první cíl této práce.

Z této analýzy nám vyplynuly přesnější požadavky na informační systém. V rámci třetí kapitoly byla navržena architektura tohoto systému, rozdělení programu na jednotlivé část, byla řešena otázka autentizace a autorizace a byl popsán základní datový model. Tato kapitola tedy odpovídá našemu druhému cíli – Navrhnout řešení, které by v maximální možné míře uspokojilo potřeby cílové skupiny firem.

Třetí cíl, neboli implementace navrženého řešení, je splněn díky vytvoření programu RP, jehož jednotlivé části jsou popsány ve čtvrté a páté kapitole. Vytvořený program je k dispozici zdarma, funguje i na starších počítačích, je snadno spravovatelný a poměrně uživatelsky přívětivý, podporuje vícero skladů, lze ho relativně jednoduše upravovat, a proto splňuje požadavky definované v části 1.1.

Splnění posledního cíle, jímž je srovnání s konkurencí, jsme dosáhli v předchozí kapitole, která obsahuje popis tří konkurenčních řešení a jejich porovnání s naší implementací.

Stanovené cíle jsou tedy splněny.

Seznam použité literatury

[1] Ethan Cerami: Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI, O'Reilly 2002
[2] JFreeChart: http://www.jfree.org/jfreechart/ (24.2.2006/27.1.2006)
[3] PostgreSQL: http://www.postgresql.org/ (12.4.2006/10.4.2006)
[4] Apache Axis: http://ws.apache.org/axis/java (24.4.2006/-)
[5] Apache Tomcat: http://tomcat.apache.org/ (24.4.2006/-)
[6] Apache http server: http://httpd.apache.org/ (24.4.2006/-)
[7] TinyERP: http://tinyerp.org/ (4.5.2006/-)
[8] OSCommerce: http://www.oscommerce.info/ (4.5.2006/-)
[9] Statistika rozšířenosti operačních systémů: www.iinfo.cz/tiskova_zprava/windows_nejuzivanejsim_operacnim_systemem/ (28.3.2006/15.3.2006)
[10] Webová služba pro ověření DIČ: http://europa.eu.int/comm/taxation_customs/vies/api/checkVatPort?wsdl (21.3.2006/-)
[11] Jan Kytýr: Informační systém pro školu a obec, bak. práce na FI MUNI (2003)
[12] Portál veřejné správy České Republiky: www.portal.gov.cz (15.4.2006/-)
[13] Martin Hlavatý: Programátorská dokumentace programu RP (2006)
[14] Encyklopedie Wikipedia: www.wikipedia.org (28.4.2006/-)
[15] Program ABC Prodeje: http://www.abcprodeje.cz (27.4.2006/-)
[16] Craig Larman: Applying UML and Patterns, Addison Wesley 2004
[17] Stromy v SQL: http://www.abclinuxu.cz/clanky/navody/stromy-v-sql (3.5.2006/11.1.2006)

Příloha A: Slovníček pojmů

CORBA – Common Object Request Broker Architecture je softwarový standard umožňující spolupráci různých aplikací, napsaných v různých jazycích a běžících na různých platformách. Definuje rozhraní, komunikační protokol a informační modely. [14]

Deployment deskriptor – viz. WSDD.

Informační systém – „Informační systém je funkční celek zabezpečující cílevědomé a systematické shromažďování, zpracování, uchovávání a zpřístupňování, či přenášení údajů. Informační systém může zahrnovat výkon všech, nebo pouze některé z uvedených činností.“ [11]

RMI – viz. RPC.

RPC – Remote procedure call (vzdálené volání procedur) je protokol umožňující procesu běžícímu na jednom počítači zavolat podprogram na jiném počítači a to bez nutnosti explicitně programovat detaily této interakce. Pokud je daný software napsán s použitím principů objektově orientovaného programování, mluví se o RPC jako o RMI neboli Remote Method Invocation (vzdálené volání metod). [14]

SOAP – Simple Object Access Protocol, následník XML RPC. SOAP je protokol pro výměnu zpráv na bázi XML skrz počítačové sítě, nejčastěji za pomoci protokolu HTTP. Obvykle se používá v rámci třívrstvé či SOA (Service Oriented Architecture) architektury. [14]

Webové služby – Dle standardu W3C je webová služba definována jako softwarový systém navržený k síťové komunikaci mezi počítači. Obsahuje rozhraní, které je popsané v strojově zpracovatelném formátu jako například WSDL. Aplikace napsané v různých programovacích jazycích a běžící na různých platformách mohou využívat webové služby k výměně dat po síti (například Internetu) stejným způsobem, jako by se jednalo o meziprocesovou komunikaci na jednom stroji. Toho je dosaženo respektováním otevřených standardů. [14]

WSDD – Web Service Deployment Descriptor je XML soubor obsahující informace o webové službě – hlavně obsluhující třídu, povolené metody a případné mapování uživatelských objektů. [4]

WSDL – Web Services Description Language je forma XML určená k popisu webových služeb. WSDL popisuje veřejné rozhraní webové služby a tedy je takovým návodem, jak s danou službou komunikovat. Obsahuje abstraktní popis podporovaných operací, podporovaných zpráv a uživatelských datových typů. [14]

Příloha B: Instalace

Před započetím instalace samotného programu je nutné nainstalovat a nakonfigurovat 4 další aplikace:

Apache http server + PHP

Web: http://httpd.apache.org/ + http://www.php.net/

Při instalaci postupujte dle návodu na výše uvedených stránkách. Pro korektní práci programu je nutné upravit konfigurační soubor *php.ini* – je povolit zavedení knihovny s funkcemi pro práci s PostgreSQL, což se dělá smazáním středníku před příslušným řádkem (pokud je přítomen).

Pro zjednodušení instalace, konfigurace a administrace je možné použít například program *EasyPHP* (http://www.easyphp.org/?lang=en)

Pro nainstalování webového obchodu stačí zkopírovat obsah adresáře web z instalačního média do adresáře *htdocs* (případně toho, který jste nastavili v konfiguračním souboru jako default). V souboru *lib/session_soap.php* je nutné nastavit adresu aplikačního serveru (standardně *http://localhost:8080/axis/services/*) a cestu k adresáři, do kterého se mají ukládat obrázky zboží.

PostgreSQL server

Web: http://www.postgresql.org/

Při instalaci postupujte dle návodu na stránkách. Nastavení, která provedete, si pečlivě zapamatujte, po instalaci samotné aplikace budete některá muset zadat do konfiguračních souborů (například přístupové jméno a heslo).

Vytvořte databázi *rocproj* (můžete zvolit libovolný jiný název, jen je nutné ho zadat do konfiguračního souboru RPsettings.xml – viz. dále) a pomocí programu *PgAdmin* neimportujte soubor *db.backup* (klikněte pravým tlačítkem na název databáze a z menu vyberte položku *restore*. Všechny volby s výjimkou *Verbose messages* by měly být nezaškrtnuty). Pokud program *PgAdmin* nemáte k dispozici lze místo něj využít skriptu *create_schema.sql*, který rovněž vytvoří databázové schéma.

Apache Tomcat + Apache AXIS

Web: http://jakarta.apache.org/tomcat/ + http://ws.apache.org/axis/

Při instalaci postupujte dle návodu na stránkách. Po instalaci zkopírujte adresář *axis* z instalačního média do adresáře *$TOMCAT_HOME/webapps* (kde *$TOMCAT_HOME* je instalační adresář Tomcatu). V souboru *axis/conf/RPsettings.xml* vyplňte položky *user* a *password* – uživatelské jméno a heslo k databázovému serveru a případně položku *databáze*, pokud jste při instalaci databázového serveru nenechali standardní jméno. Soubor by měl vypadat následovně:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd"
>
<properties>
<entry key="user">postgres</entry>
```

```
<entry key="password">1234</entry>
<entry key="url">jdbc:postgresql</entry>
<entry key="className">org.postgresql.Driver</entry>
<entry key="database">rocproj</entry>
</properties>
```

Dále je nutné zadat cestu *$TOMCAT_HOME/webapps/axis* do proměnného prostředí počítače (Systémové proměnné) pod klíč AXIS_HOME. To se provede v případě operačního systému Windows následovně:

Vybereme z ovládacích panelů položku *Systém*, v ní kartu *Upřesnit*, kde klikneme na tlačítko *Proměnné prostření* a prostřednictvím tlačítka *Nová* (Systémová proměnná) přidáme novou položku s názvem AXIS_HOME a hodnotou $TOMCAT_HOME/webapps/axis (kde *$TOMCAT_HOME* je instalační adresář Tomcatu)

Jython

Web: http://www.jython.org/

Při instalaci postupujte dle návodu na stránkách. Po nainstalování je nutné zkopírovat adresář *bean* (obsahuje zkompilované třídy *BeanObjednavka.class, BeanZakaznik.class* a další) do adresáře *$JYTHON_HOME/lib*.

Nakonec doplňte do konfiguračního souboru *axis/conf/RPsettings.xml* (viz. výše) následující řádek:

```
<entry key="JYTHON_HOME">$JYTHON_HOME</entry>
```

Kde *$JYTHON_HOME* označuje adresář, do kterého jste Jython nainstalovali.

Klient RP

Pro instalaci je možné využít instalátor nebo ručně zkopírovat soubory z instalačního média. Po instalaci je nutné upravit konfigurační soubor *settings.xml* (nachází se v kořenovém adresáři programu) – nastavit adresu aplikačního serveru (položka **SOAPadr**, standardně *http://localhost:8080/axis/services/*), adresu souboru s rozšířeními (položka **MODadr**, standardně http://locahost/modules.xml) a konečně adresu nápovědy (položka **HELPadr**, standardně http://locahost/RPHelp.jar). Soubor by měl vypadat následovně:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd"
>
<properties>
<entry key="MODadr">http://localhost/modules.xml</entry>
<entry key="SOAPadr">http://localhost:8080/axis/services/</entry>
<entry key="HELPadr">http://localhost/RPHelp.jar</entry>
<entry key="LookAndFeel">
com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel</entry>
</properties>
```