VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

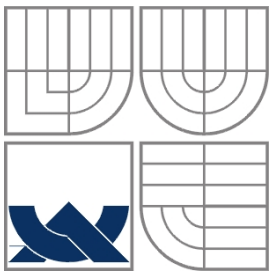
INFORMAČNÍ SYSTÉM PRO VYHODNOCOVÁNÍ AKTIVIT PRACOVNÍKŮ VE STROMOVÉ HIERARCHII V CACHÉ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

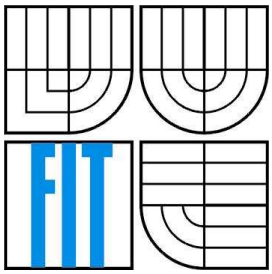
AUTOR PRÁCE
AUTHOR

BC. JAROSLAV BURGET

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM PRO VYHODNOCOVÁNÍ AKTIVIT PRACOVNÍKŮ VE STROMOVÉ HIERARCHII V CACHÉ

INFORMATION SYSTEM FOR EVALUATION OF EMPLOYES' ACTIVITY IN TREE HIERARCHY USING CACHÉ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. JAROSLAV BURGET

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAROSLAV RÁB

BRNO 2008

Abstrakt

Informační systémy jsou nedílnou součástí téměř všech větších obchodních subjektů. Tento dokument má za účel seznámit s návrhem modelu v UML a implementací informačního systému pro vyhodnocování aktivit pracovníků ve stromové hierarchii. První část zahrnuje princip multilevel marketing firem, popis zadání systému a průběh jeho modelování v nástroji CASE Select Component Architekt. Druhá část se zabývá popisem implementovaného systému formou webové aplikace nad objektovou databází Caché.

Klíčová slova

Informační systém, stromová hierarchie, UML model,objektová databáze, Caché, Caché ObjectScript, Caché ServerPages.

Abstract

At most of the times, information systems are the corner stone for companies nowadays. This document's intention is to introduce the model design phase in UML and the implementation of an information system for evaluating the activity of empolyees in a tree hierarchy. The first part is made of business' multilevel marketing principle, a description of the system's specification and a progression of the system's modeling in a CASE tool, Select Component Architect. The second part describes the implemented system through a web application, which is based on Caché, an object database.

Keywords

Information systém, tree hierarchy,UML model, object database, Caché, Caché ObjectScript, Caché ServerPages.

Citace

Burget Jaroslav: Informační systém pro vyhodnocování aktivit pracovníku ve stromové hierarchii v Caché. Brno, 2008, diplomová práce, FIT VUT v Brně.

Informační systém pro vyhodnocování aktivit pracovníků ve stromové hierarchii v Caché

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaroslava Rába
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jaroslav Burget
15.1.2008

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu práce Ing. Jaroslavu Rábovi za cenné rady a připomínky při řešení tohoto projektu.

Obsah

1 Úvod

Firmy řízené stromovou hierarchií jsou světovým fenoménem již několik desetiletí. V kosmetice, spotřebním průmyslu a zejména finanční sféře zaujímají velký podíl trhu. V České republice téměř všechny finanční podniky a ústavy spolupracují s multilevel marketing (MLM) firmami, jelikož se jedná o nejpřímější oslovení koncového zákazníka. Díky síti zaměstnanců pracujících nezávisle na ústředí firmy může tato síť pokrýt obrovské spektrum zákazníků (Spilka, 2007).

Na žádost několika pracovníků finanční firmy jsem se rozhodl navrhnout a vytvořit informační systém dle specifikace jejich požadavků. Hlavním kritériem bylo zefektivnění a zrychlení evidence klientů a jim poskytují smluv. Finanční firma je založena na stromové hierarchii (MLM) a na českém trhu působí již přes 10 roků. Ke konci roku 2007 do svých řad přijala přes 60 000 zaměstnanců přičemž více než 5 000 je stále aktivně se podílejících na vývoji firmy.

Vyvíjený informační systém bude testován a využíván úzkou skupinou uživatelů kteří tvoří jednu „větev“ toho stromu. Tito uživatelé potřebují neustálou aktualizaci svých klientů a smluv, aktualizace dat týkajících se služeb od samotných finančních institucí (banky, pojišťovny, spořitelny) a v neposlední řadě také přístup k těmto datům bez ohledu na lokální instalace v počítači. Informační systém je proto kompletně navržen objektově pomocí jazyka UML 2.0 v nástroji CASE Select Component Architect a pro samotnou implementaci jsem zvolil objektovou databázi Caché. Z důvodů přístupnosti dat na internetu se jedná o portálové řešení využívající objektovou databázi Caché a prostředí klienta je implementováno pomocí Caché ServerPages využívajících Caché Object Script.

2 Řízení zaměstnanců ve stromové hierarchii

Základním principem těchto subjektů je přijímaní nových pracovníků a vytváření stromové struktury jak do šířky, tak do hloubky. Každý stávající zaměstnanec počínaje zakladatelem má právo přivézt do systému libovolný počet svých podřízených, kteří tvoří jeho podstrom. Rekurzivně i tito podřízení přivádějí další zaměstnance.

2.1 Síťový marketing versus „pyramidové struktury“

Široká veřejnost zná firmy řízené stromovou hierarchií pod pojmy MLM – multilevel marketing, síťový marketing ale někteří si je pletou s pyramidami (letadly) což jsou nelegální víceúrovňové struktury lidí, jejichž cílem je vydělat peníze na lidech, kteří se připojí po nich. Letadlo je založené na tzv. duplikačním efektu. Kdokoliv přivede někoho nového, vybere od něj vstupní poplatek, jehož část si nechá a část pošle na přerozdělení člověku, který ho do systému přivedl. Ten si část nechá a část pošle výš. A tak to jde až k jedinci na vrcholku pyramidy, který daný systém založil. Vzhledem k tomu, že trh je omezený, nemůže tento systém fungovat do nekonečna a po poměrně krátké době se zhroutí. Problémem je, že letadlo nevytváří žádnou přidanou hodnotu, pouze přerozděluje peněžní prostředky. Pyramidové hry jsou sice ve všech vyspělých zemích zakázány, stále je však obtížné systém rozpoznat a organizátory usvědčit.

Oproti tomu MLM je systém, který dává dohromady dvě marketingové strategie - přímý marketing a franchising. Subjekty, které tento systém využívají, zabývající se především přímým prodejem svých produktů (se kterým je multilevel marketing velmi úzce spjat) prostřednictvím svých obchodních zástupců, mají více či méně podobné systémy přerozdělování provizí. Hlavním rozdílem je, že tyto firmy na rozdíl od typických pyramid a letadel opravdu prodávají nějaké produkty - vytvářejí určitou přidanou hodnotu. Veškeré peníze (nebo jejich drtivá většina), které se v systému pohybují, jsou vygenerovány prodejem firemních produktů. To, že firma fungující na bázi MLM prodává určité produkty, její smysl na trhu nejen moralizuje, ale především legalizuje. Prodáváním produktů (zprostředkováním služeb) získávají provize sami zaměstnanci, kteří produkt poskytli a také zaměstnanci ve struktuře nad nimi (Clothier, 1995).

2.2 Stromové hierarchie ve finančním sektoru

Finanční sektor je na české trhu zastoupen několika MLM společnostmi, které pokrývají většinu dostupných finančních produktů. Rozdíl mezi MLM společnostmi prodávajícími produkty je v tom, že zaměstnanci finančních společností svým klientům nenabízí přímé produkty ale zprostředkovávají služby.

Zaměstnanci jsou zprostředkovateli mezi finančními institucemi (bankami, pojišťovnami, spořitelnami atd.). Všechny nabízené služby by klient sice mohl získat sám v konkrétní finanční instituci, ale pro širokou veřejnost není jednoduché se objektivně orientovat v množství nabízených služeb různých konkurujících si institucí. Zaměstnanci MLM společností jsou speciálně školeni pro přehled a aktuální orientaci ve finančním sektoru. Jsou schopni svým klientům nejen nestranně nabídnout kompletní seznam nabízených služeb různými finančními institucemi, ale jsou také oprávněni tyto služby zprostředkovat. Tím je koncový zákazník ušetřen problémů týkajících se samostatného vyřizování všech náležitostí.

Za takto zprostředkované služby získávají zaměstnanci provize odpovídající jejich pozicím a typům zprostředkovaných služeb. A stejně jako v ostatních MLM společnostech tyto provize získávají zaměstnanci ve struktuře nad nimi. Funkčnost přerozdělování provizí pro konkrétní případ vyvíjeného informačního systémů je slovně i modelově popsán v následující kapitole.

3 Model informačního systému

3.1 Úvodem

Návrh informačních systémů je obtížnou záležitostí pokrývající mnoho různorodých oblastí. Kvalitní objektově orientovaný návrh aplikace je nezbytným předpokladem jeho úspěšné implementace (Kanisová,Müller, 2006). Existuje několik metod a způsobů, jakými se dají informační systémy modelovat. Pro svůj projekt jsem si vybral jazyk UML 2.0 pomocí nástroje CASE Select Component Architect (SCA). Nejzajímavější a nejdůležitější modelové části a diagramy jsou uvedeny v textu diplomové práce a kompletní model systému je v XML exportu a HTML stránkách nástroje CASE přiložen v přílohách na CD.

3.2 UML

UML – Unified Modeling Language je unifikovaný modelovací jazyk, který má, na rozdíl od převážně textově orientovaných programovacích jazyků, vlastní grafickou syntaxi (pravidla pro sestavování jednotlivých elementů jazyka do větších objektů) a sémantiku (jednoznačná pravidla určující jednotlivým syntaktickým výrazům jejich význam).

V současné době má jazyk UML největší význam převážně při návrhu softwarových systémů. Pro objektově orientovaný návrh je samozřejmě možné použít různé podpůrné prostředky, zejména další odlišné typy diagramů, avšak UML přesně specifikuje, co má daný diagram obsahovat, což je nejpodstatnější zejména při sdílení informací mezi jednotlivými analytiky a vývojáři. Je nutné, aby vytvářené grafy měly vnitřní konzistenci a přesně danou sémantiku, což u jiných typů grafů nemusí být obecně zaručeno. Existuje několik typů UML diagramů které se liší podle toho, jaké se pomocí nich plánují či zpracovávají úlohy. Tyto diagramy se od sebe odlišují především repertoárem použitých značek, způsobem jejich vzájemného propojení a také s nimi související sémantikou.

Jazyk UML je standardizován a podporován celou řadou vývojových nástrojů, což mohou být samotné aplikace určené pouze pro práci s UML, nebo také integrované vývojové prostředí, které v mnoha případech umožňují převod informací mezi UML diagramem a algoritmem již zapsaným v programovacím jazyce (Kanisová,Müller, 2006).

3.2.1 Základní elementy jazyk UML

UML je založen na třech elementech, které jsou z uživatelského hlediska reprezentovány grafickými značkami. Podle své funkce se nazývají:

- Předměty

- Digramy
- Relace

Předměty jsou elementy zpracovávaného modelu, které se dále člení do různých podkategorií. Tyto podkategorie jsou: strukturní abstrakce UML modelu (programové třídy, aplikační rozhranní, objektové rozhranní, případy použití, komponenty atd.), chování (komunikace mezi jednotlivými objekty), seskupení (seskupuje čísti diagramu na nižší úrovni, balíčky) a poznámky (pro bližší specifikaci vlastnosti a chování dalších elementů UML digramu)

Relace slouží k vzájemnému propojení, určování vztahů mezi různými předměty. V UML existují tyto typy relací: asociace (modelují obecnou, přesně definovanou závislost předmětů), závislost (jsou používány aby změna v jednom předmětu způsobila změnu v předmětu jiném), generalizace (modeluje vztah, kdy je jeden předmět specializací jiného) a realizace (jedná se o druh vztahu, ve kterém jeden předmět představuje dohodu, za jejíž splnění je odpovědný jiný předmět)

Diagramy zachycují různé aspekty modelovaného systému, který nemusí být vyjádřen pouze jedním UML diagramem, protože je možné například pomocí balíčků sdružovat více diagramů do jednoho hierarchicky organizovaného celku. Typů diagramů existuje velké množství, uvedu například: diagram případů použití**,** diagram tříd**,** diagram objektů**,** diagram komponent**,** diagram nasazení**,** diagram aktivit**,** stavový diagram**,** diagram spolupráce a sekvenční diagram**.** Každý z těchto výše vyjmenovaných typů diagramů obsahuje poněkud odlišný repertoár dostupných grafických značek (Kanisová,Müller, 2006).

3.2.2 Iterativní přístup k modelování

Pro modelování informačního systému je vhodné požití modelovacích kroků sdružených do takzvaných iterací. Výsledkem každé iterace je přírůstek funkčnosti systému. Je proto vhodné jednotlivé iterace volit tak, aby bylo možné přehledně rozfázovat jednotlivé kroky návrhu. Samotná každá iterace se potom sestává z několika kroků, jejichž počet si můžeme sami volit dle potřeby návrhu. To, jaká část aplikace bude vyvinuta jednotlivými přírůstky, je definováno pomocí množin realizovaných případů užití. Jádrem přírůstkového stylu práce je schopnost odpovídat rychle a flexibilně na změny stavu projektu.

V informačním systému který je zde modelován jsem použil iterativní přístup a během analýzy navrhnul tři iterace. Počínaje diagramem případů užití je model z důvodu přehlednosti rozdělen do těchto tří iterací a v samotné práci jsou kombinovány diagramy jednotlivých iterací a celkové výsledné diagramy modelu. Rozlišení o kterou iteraci se jedná je popsáno následující notací, kdy každý diagram případ užití je označen v diagramu:

- I.- diagram první iterace
- II. - diagram druhé iterace
- III. - diagram třetí iterace

3.3 Požadavky systému

Požadavky systému nejsou součásti jazyka UML a oblast uživatelských požadavků je v UML podporována pouze vazbou na případy užití. Z důvodu zachycení funkčních i nefunkčních požadavků jsou v této části definovány na základě konzultací se zaměstnanci firmy. V nástroji CASE SCA, ve kterém je model navržen, jsou zastoupeny diagramy užití a procesním modelováním. V této části jsou uvedeny všechny stěžejní funkční požadavky a rozdělení.

3.3.1 Neformální specifikace

Přáním zaměstnanců je navrhnout a vytvořit informační systém finanční společnosti která je řízena stromovou hierarchií.

Systém stromové struktury: jedná se o MLM společnost, kde každý zaměstnanec má právo do společnosti přivézt nové zaměstnance, kteří se ve stromové hierarchii zařadí bezprostředně pod něj. Rekurzivně každý takový nový zaměstnanec může přivézt do společností další zaměstnance a vytvořit si tak svůj vlastní podstrom. V případě výpovědi zaměstnance a jeho odebrání se systému se všichni jeho přímí „podzaměstnanci“ stávají přímými „podzaměstnanci“ jeho přímého nadřízeného a všichni klienti a smlouvy které mazaný zaměstnanec zprostředkovával přechází také jeho přímému nadřízenému.

Rozdělení uživatelů

Hlavním správcem systému je osoba (nadále uváděn jako správce), která má na starost udržování aktuálního stavu zaměstnanců a vztahů mezi nimi. Správce má právo zaměstnance do systému vkládat, odebírat nebo editovat jejich parametry tak aby zůstala zachována stromová struktura společnosti uvedená výše. Přidáním zaměstnance do systému vytvoří jeho přihlašovací jméno a heslo. Každý zaměstnanec je v rámci firmy identifikován jedinečným číslem. Správce ve stromové hierarchii vystupuje jako kořen stromu s právy zobrazit, přidat, smazat a editovat veškerá data podřízených. Dalším funkčním požadavkem na správce je správa číselníků:

- Finanční subjekty (číselník bank, finančních institucí, pojišťoven atd.)
- Typy smluv (číselník nabízených typů smluv)
- Položky smluv (číselník které položky se budou vztahovat ke kterým typům smluv)
- Body smluv a pozice (určuje pozici zaměstnance a body udělené za vykázanou práci)

Ostatní uživatelé systému jsou všichni zaměstnanci firmy, kdy je každému v okamžiku vložení do systému vytvořen přihlašovací účet. Každý z těchto zaměstnanců musí mít svého nadřízeného

zaměstnance a může mít libovolný počet podřízených zaměstnanců o kterých mu systém musí nabízet aktuální přehled. Každý zaměstnanec:

- Má společností přiděleno jedinečné identifikační číslo pod kterým vystupuje a které je uváděno na každé jeho zprostředkované smlouvě.
- Spravuje svoje klienty (fyzické a právnické osoby) o kterých potřebuje uchovávat informace. Tito klienti si skrze zaměstnance objednávají služby (stavební spoření, pojištění, povinné ručení (viz číselník typů smluv)
- zaměstnanec za každou poskytnutou službu klientovi (konkrétně za úspěšně podepsanou a vyřízenou smlouvu) dostává určité bodové hodnocení (dle tabulek podle typu smluv, pojistné částky atd.) a za tyto body má odpovídající finanční prémie. Díky stromové hierarchii také všichni zaměstnanci v hierarchii nad ním získají toto bodové hodnocení, ale finanční ohodnocení se za body získané ze svého podstromu je závislé na pozici zaměstnance vůči svým přímým podřízeným. Konkrétní případy výpočtu jsou uvedeny v modelu.
- Každý bod má svoji finanční hodnotu podle pozice na které se zaměstnanec nachází. Tato pozice se určuje podle celkového počtu bodů, které zaměstnanec získal od svého nástupu ke společnosti. (konkrétní hodnoty k získání pozice jsou uvedeny v číselníku)
- Pokud získá body zaměstnanec v hierarchii pod zaměstnancem, je aktuálnímu zaměstnanci tento bod také započítám, ale finanční ohodnocení za tento bod je dán rozdílem pozic těchto zaměstnanců – tento princip platí až po kořen stromu.
- Pokud některý ze zaměstnanců firmu opustí (správce vyřadí zaměstnance ze stromu), všichni jeho „podřízení“ jsou přiřazeni nadřízenému tohoto zaměstnance a nárok na správu klientů a mluv tohoto odcházejícího zaměstnance získává také jeho nadřízený.

Funkční požadavky na systém

Systém každému zaměstnanci bude umožňovat:

- uchovávat databázi jeho klientů s možností přidání odebrání klientů a editace jejich osobních dat. Zobrazování formou tabulek s možností řazení a vyhledávání.
- přidávat, odebírat a editovat klienty dle potřeby – možnost řazení smluv podle různých parametrů smlouvy.
- u každého klienta uchovávat typy smluv které má uzavřeny, data sjednání a respektive ukončení smluv a také základní důležité informace na těchto smlouvách dle číselníkových hodnot. Konkrétnímu zaměstnanci umožnit zobrazit, editovat, mazat a přidávat smlouvy a také vidět data klientů a smluv zaměstnanců, kteří jsou ve stromové hierarchii pod ním.
- vypsání jednotlivých klientů a smluv, různě způsoby seřazení podle parametrů.

- za každou smlouvu ukazovat přidělený počet bodů a také celkový počet bodů a pozici zaměstnance. Získané body přidělit celé části stromu a umožnit orientačně vypočítat finanční odměnu za body které zaměstnanec získal sám a které získat ze své části podstromu.
- vidět svoje přímé podřízené (ty které sám do firmy přivedl) a kolik bodů získává od každého z nich (respektive od podvětve stromu každého z nich). A také systém musí zobrazovat jeho přímého nadřízeného.
- Přístup k systému si zaměstnanci přejí přes webové rozhraní.

3.4 Procesní modelování

Úvodním krokem analytické práce jsem zvolil procesní modelování, přestože není součástí standardu UML. Významným přínosem procesního modelování je podchycení souvislostí mezi firemními elementárními procesy, které budou následně přetransformovány v konkrétní případy užití. Pro vytvoření diagramů procesního modelování jsem použil objektově orientované metodiky Select Perspective vytvořené Stuartem Frostem a Paulem Allenem z firmy SELECT Software Tools. [Select Perspective: The Practical Methodology for delivering next-generation applications, Princeton Softech, 2000].

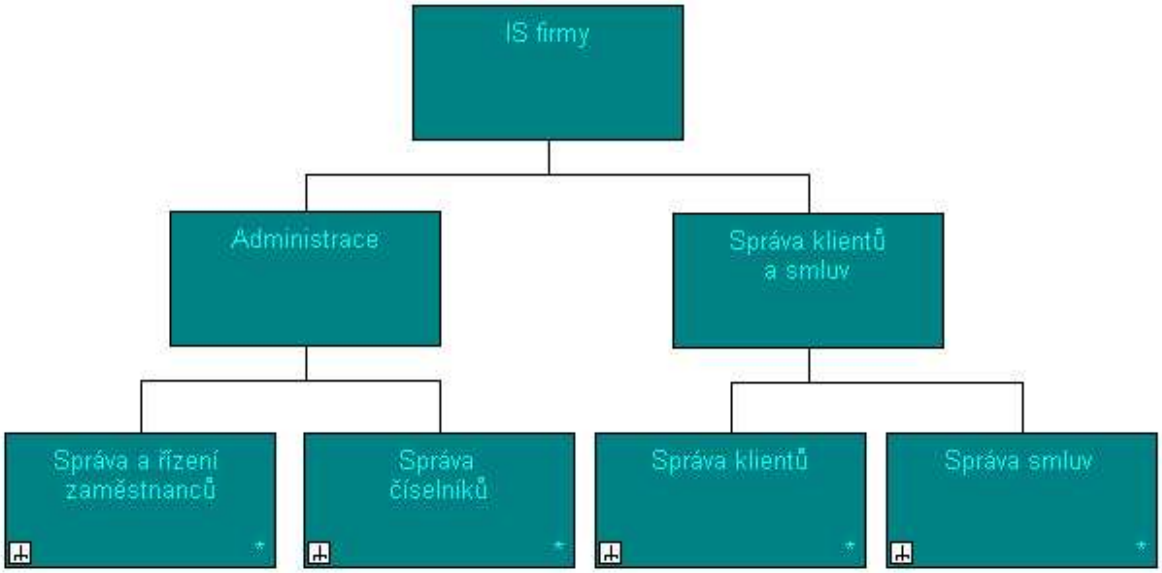
Kanisová a Müller (2006) uvádí, že přínos metodiky Select Perspective je v tom, že samotné techniky jazyka UML vhodně doplňuje technikami tak, aby zabezpečila pokrytí celého životního cyklu tvorby aplikace. Kromě UML se jedná zejména o vizuální modelovací techniky modelování firemních procesů (BPM – Business Process Modeling) a datové modelování (Data Modeling).

Nástroj SCA, kterým jsem procesní modelování realizoval, poskytuje plnou podporu pro vytvoření diagramů firamních procesů a umožňuje jejich provázání s následujícími diagramy UML případů užití. Ze samotných diagramů lze také snadněji odvodit komponenty a prvky dalších diagramů.

3.4.1 Diagram hierarchie procesů

Řešení procesního rozpadu systému je řešeno na diagramu Obr. 1. Znázorňuje modularitu a vzájemné souvislosti jednotlivých firemních procesů na jejich nejvyšší úrovni. Firemní proces je definován jako sekvence činností vytvářející produkt, který organizace potřebuje pro splnění svých firemních cílů. Nejvyšší úrovní tohoto diagramu je samotný informační systém pro vyhodnocování aktivit pracovníků ve stromové hierarchii. Nejnižší úroveň diagramu jsou subprocesy, které jsou dále popsány diagramem procesních řetězců v části 3.4.2 této kapitoly. Jak je patrno z diagramu, je tvořen

dvěma hlavními větvemi procesů. Již tyto větvě definují rozdělení systému z pohledu administrátorského a uživatelského.



Obr. 1: Diagram hierarchie procesů

3.4.2 Diagramy procesních vláken

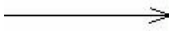
Diagram procesních vláken poskytuje pohled na informační systém z hlediska dynamiky. Výhodou tohoto diagramu a také důvodem k jeho vytvoření je srozumitelnost i pro zadavatele požadavků. Touto technikou jsou namodelovány firemní události, které inicializují spouštění firemních procesů a také výsledky těchto procesů (Kanisová,Müller, 2006). Dalším přínosem je možnost mapování propojení mezi jednotlivými firemními procesy.

Jelikož diagram procesních vláken není uveden ve standardu UML popisuji význam jednotlivých komponent použitých na tomto diagramu pomocí nástroje SCA:

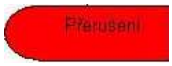
Firemní událost. Reprezentuje událost, která spouští nějakou aktivitu a zahrnuje spuštění příslušného firemního procesu, který na ni reaguje. Pokud je špička šipky přeškrtnutá, tak se jedná o externí událost.

Firemní výsledek, který symbolizuje ukončení procesního řetězce.

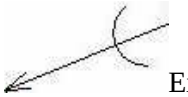
Firemní proces. Elementární firemní proces, který je realizován během procesního řetězce.



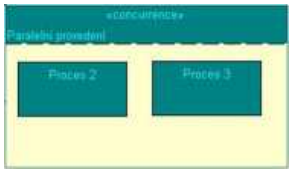
Znázornění závislosti procesů na firemních událostech.



Přerušení procesu. Indikuje možnost, kdy další činnosti (procesy) mohou probíhat, až nastane nová událost. Dokud tato událost nenastane, procesy jsou pozastaveny.



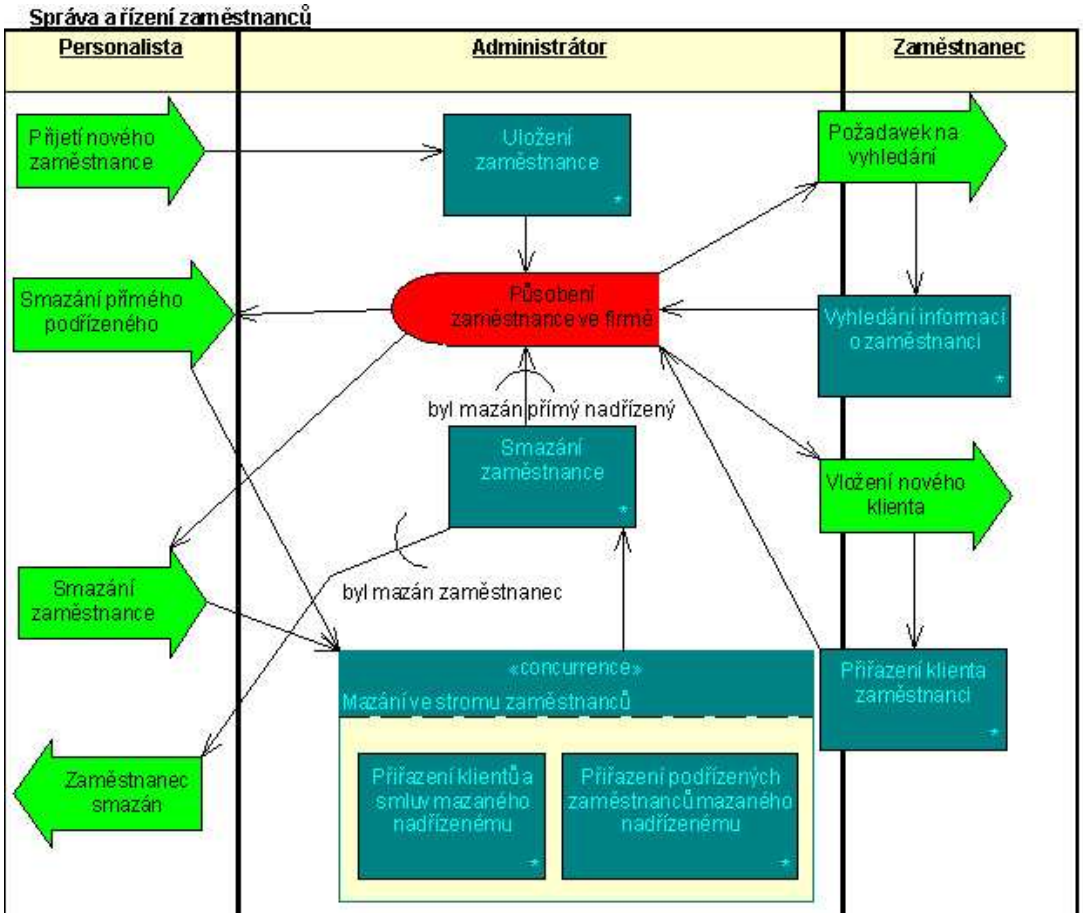
Exkluzivní závislost, použitá pro označení, že je spouštěn pouze jeden z více procesů. Většinou popsána podmínkou, za jaké k této události může dojít.



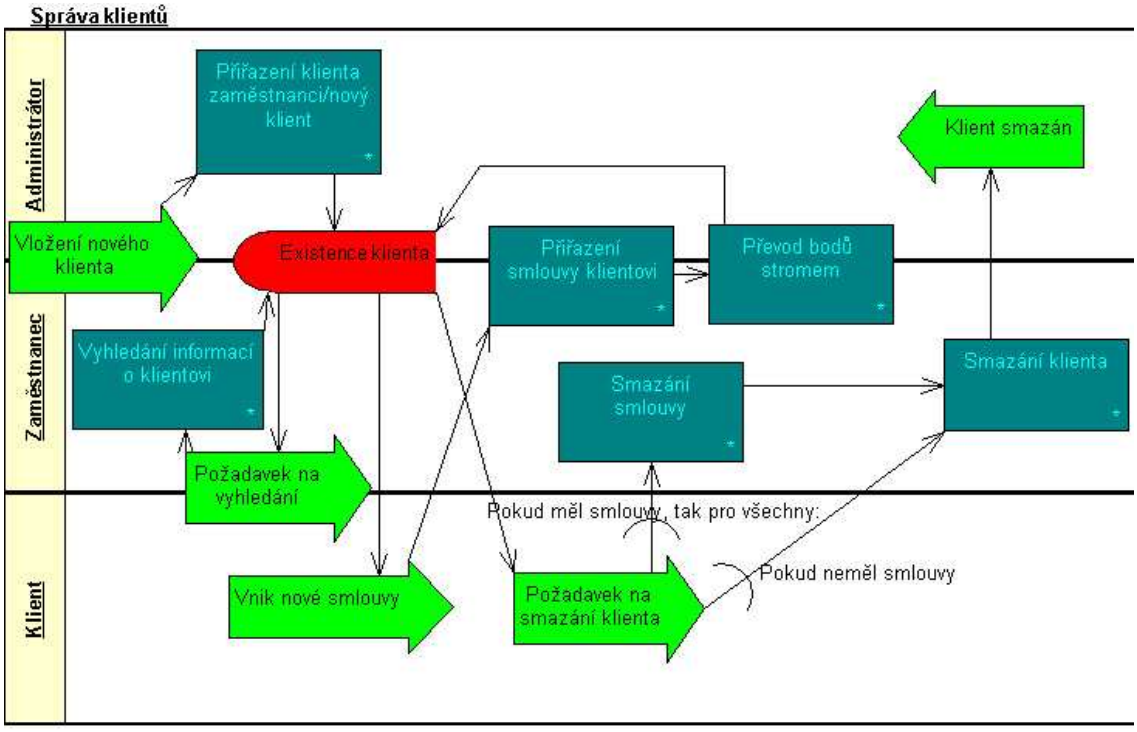
Vyznačuje souběžné procesy, popřípadě že procesy jsou provedeny jako celek, než budou prováděny jiné procesy (události)

Plavecké dráhy. Ukazují, který firemní aktér hraje roli ve vyvolání firemní události nebo firemního procesu. V případě že některá komponenta zasahuje do plaveckých drah dvou firemních aktérů (na Obr. 2 viz Administrátor a Zaměstnanec) může být firemní událost nebo proces vyvolán oběma firemními aktéry.

Diagramy procesních vláken jsou součástí přílohy - Model Systému, vyexportovaném v XML a HTML na přiloženém CD. V této části uvádím tři nejdůležitější diagramy pro pochopení funkčního chodu firmy. Od těchto diagramů se následně odvíjí všechny případy užití na základě kterých je celý systém modelován a implementován. První z diagramu na Obr 2 popisuje proces „Správy a řízení zaměstnance“. Na základě firemních události aktéra „Personalista“ dochází k vložení nového zaměstnance do stromu zaměstnanců. Životní cyklus zaměstnance je dán jeho působením ve firmě, kdy může být přerušen událostmi požadavku na vyhledání, vložení (smazání) klienta, vzniku (zániku) smlouvy s klientem nebo smazáním přímého podřízeného (samotného zaměstnance) ve stromové hierarchii. Tento diagram přehledně znázorňuje přeuskupení stromové struktury zaměstnanců a převázání klientů a smluv z mazaného zaměstnance na jeho předka.

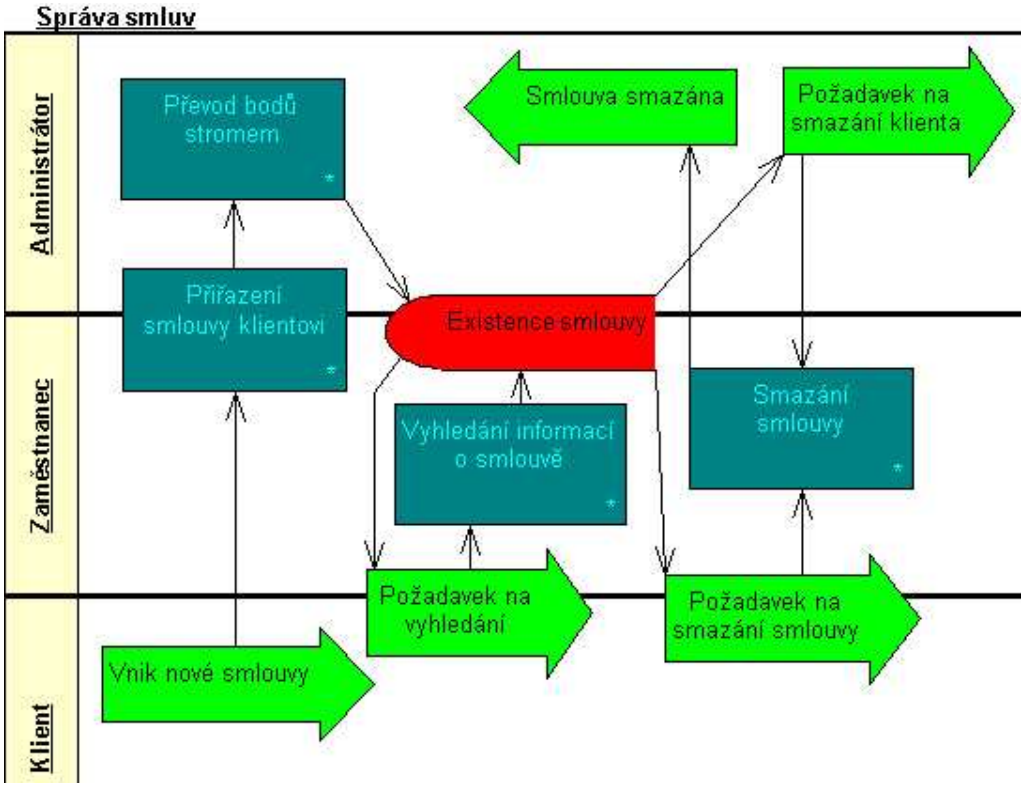


Obr. 2: Diagram procesních vláken – správa a řízení zaměstnanců



Obr. 3: Diagram procesních vláken – správa klientů

Diagram na Obr 3 a Obr 4 znázorňuje souvislost správy klientů a smluv, přestože se jedná o dva samostatné firemní procesy v rámci hierarchie. Do jednoho diagramu je znázorněna správa klientů s reakcí na přidání nové smlouvy ve druhém diagramu je znázorněna nemožnost existence smlouvy bez existence klienta. Zároveň oba diagramy ukazují, že při vzniku nové smlouvy je třeba projít stromem zaměstnanců a upravit body patřičných zaměstnanců.



Obr. 4: Diagram procesních vláken – správa smluv

3.5 Případy užití

K zachycení přesné funkčnosti systému slouží případy užití. Vymezují jednoznačně rozsah prací, kdy každý případ užití popisuje jeden ze způsobů užití systému (Kanisová,Müller, 2006).

Aktéři

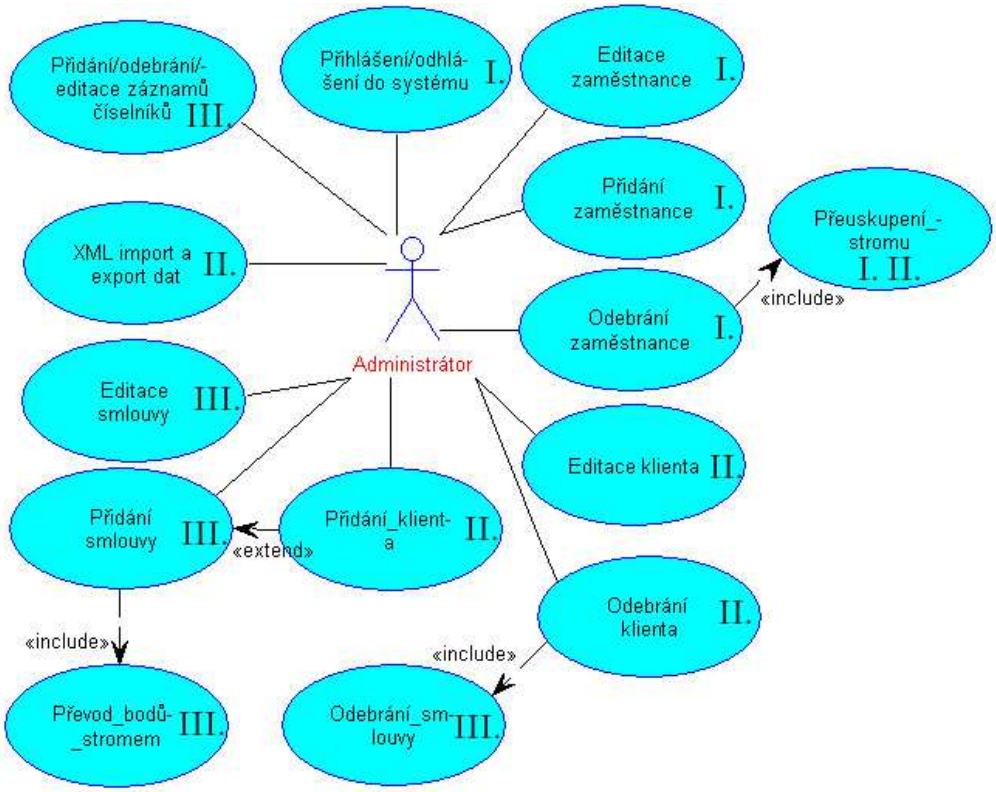
V rámci firemních procesů vystupuje řada firemních aktérů v modelovaném systému se dle předchozích diagramů setkáváme s personálním oddělením, administrátorem, zaměstnancem, vedením firmy popřípadě zástupci finančních institucí a klientem. Ne každý z nich ale komunikuje přímo s daným informačním systémem. Aktérem je rozuměna role, kdy uživatel vystupuje v rámci komunikace se systémem. Aktéry modelovaného systému jsou administrátor a zaměstnanec a všechny akce v systému jsou vyvolány těmito aktéry.

3.5.1 Řízení projektu na základě případů užití

Případy užití definováním funkčních požadavků na systém určují pořadí, ve kterém budou řešeny jednotlivé přírůstky projektu. Na základě definice priorit požadavků jsem naplánoval pořadí vývoje jednotlivých přírůstků systému a rozdělil je do tří iterací. Pro přehlednost robustnosti celého systému jsem do této kapitoly diagramy všech tří iterací spojil ve dva celky dle hlavních aktérů. Některé případech použití zasahují do dvou či tří iterací. Jedná se o případy, kdy je základní funkčnost implementována v rámci jedné iterace a rozšiřující funkčnost v následující (např. vyhledávání nad objekty, řazení atd.)

Rozdělení do jednotlivých iterací:

1. Iterace (Přidání, odebrání a editace zaměstnance, přeuskupení stromu, výpis/vyhledávání podstromu a podřízených, přihlášení/odhlášení ze systému)
2. Iterace (Přidání, odebrání a editace klienta, přeuskupení stromu, XML import a export dat, výpis/vyhledání klientů)
3. Iterace (Přidání, odebrání a editace smluv, převod bodů stromem, přeuskupení stromu, přidání/odebrání/editace záznamů číselníků, výpis/vyhledání smluv)



Obr. 5: Diagram případů užití – Administrátorská část

3.5.2 Popis případů užití

Případ užití je sada scénářů, které spojuje dohromady společný cíl (Kanisová,Müller, 2006). Přidání a odebrání zaměstnance zahrnuje případ užití přeuskupení stromu, jak je patrno z Obr 3. Druhou zajímavou funkčností je takzvané kaskádní mazání, které znázorňují případy užití odebrání klienta a odebrání smlouvy. Vysvětlení principu těchto případů užití je uvedeno v následujících formálních popisech.

Přidání a odebrání zaměstnance – přeuskupení stromu

Přidání zaměstnance do stromové struktury je možné pouze pokud je předem vybrán jeho nadřízený. Samotným přidáním do systému zaměstnanec vytvoří nový list stromu a od této chvíle může být k do systému přidán nový zaměstnanec, jemuž bude nadřízeným a tudíž bude ve stromové hierarchii pod ním.

Odebrání zaměstnance je složitější o přeuskupení existujících zaměstnanců, kteří jsou v hierarchii pod odebíraným zaměstnancem. Pokud zaměstnanec zprostředkoval smlouvy nějakým klientům, tak tito klienti a smlouvy a musí být v systému zachovány a předány patřičnému zaměstnanci.

Název:	**Přidání zaměstnance**
Stručný popis:	Vložení nového zaměstnance jako nový list stromu existujícímu zaměstnanci
Primární aktér:	Administrátor
Předpoklady:	Administrátor přihlášen do systému
Hlavní tok:	Případ použití se spustí, když administrátor vybere v menu položku: přidání zaměstnance. Systém vygeneruje seznam všech zaměstnanců ke kterým může být nový zaměstnanec přiřazen. Systém vygeneruje formulář pro vyplnění osobních informací o zaměstnanci Administrátor vybere nadřízeného ze seznamu zaměstnanců, kterému bude zaměstnanec přiřazen a vyplní požadované pole informacemi. Zaměstnanci je přiděleno unikátní osobní číslo v rámci firmy, kterým se identifikují i jeho klienti a smlouvy. Po stisknutí uložení systém provede kontrolu vyplnění dát a v případě jejich správnosti zaměstnance uloží. Strom je rozšířen o další list.
Alternativní toky:	Validace dat: pokud je zadáno již existující osobní číslo nebo nevyplněno některé z povinných polí, systém neuloží zaměstnance a znovu vygeneruje formulář.

Název: **Odebrání zaměstnance**

Stručný popis: Odebrání existujícího zaměstnance ze stromové hierarchie

Primární aktér: Administrátor

Předpoklady: Administrátor přihlášen do systému, zaměstnanec existuje

Hlavní tok: Případ použití se spustí, když administrátor vybere v seznamu zaměstnanců položku: smazat.
Systém zobrazí varovný dotaz, zda má být zaměstnanec opravdu smazán.
Administrátor potvrdí smazání.
Systém spustí kontrolu existence podřízených, smluv a klientů.
Pokud nalezne alespoň jednoho klienta, spustí alternativní tok č.1.
Pokud nalezne alespoň jednu smlouvu, spustí alternativní tok č.2.
Pokud nalezne alespoň jednoho podřízeného zaměstnance , spustí alternativní tok č.3
Systém smaže vybraného zaměstnance.

Alternativní toky: č.1: systém nalezne přímého nadřízeného zaměstnance k mazanému a všechny nalezené klienty přiřadí tomuto nadřízenému zaměstnanci.
č.2: systém nalezne přímého nadřízeného zaměstnance k mazanému a všechny nalezené smlouvy přiřadí tomuto nadřízenému zaměstnanci.
č.2: systém nalezne přímého nadřízeného zaměstnance k mazanému a u všechny podřízené zaměstnance mazaného vloží ve stromové hierarchii tomuto nadřízenému. Mazaný zaměstnanec je takto „postaven“ mimo strom a může být smazán.

Přidání smlouvy a převod bodů stromem

Zaměstnanec, klient a smlouva tvoří závislou trojici celého systému. Nejvýše v této hierarchii stojí zaměstnanec, poté klient a nakonec smlouva. V praxi to znamená, že nemůže existovat klient, aniž by existoval zaměstnanec který mu služby zprostředkovává a následně nemůže existovat smlouva, aniž by existoval klient, který si tuto smlouvu uzavřel. Tato závislost je funkčně popsána již v předchozím případu užití a v následujících dvou případech užití popisuji princip přidání smlouvy a následného převodu bodů patřičnou částí stromu.

Název: **Přidání smlouvy**

Stručný popis: Vložení nové smlouvy do systému

Primární aktér: Administrátor nebo zaměstnanec (uživatel)

Předpoklady: Uživatel přihlášen do systému. Existence alespoň jednoho klienta zaměstnance, kterému bude smlouva přidána.

Hlavní tok: Případ použití se spustí, když uživatel vybere v menu položku: přidání smlouvy.

Pokud novou smlouvu přidává administrátor, systém vypíše seznam zaměstnanců a administrátor potvrdí výběr jednoho z nich a pokračuje se následujícím testem:

Pokud přidávající nebo administrátorem vybraný zaměstnanec nemá žádného klienta, spustí alternativní tok 1) , jinak:

Systém vypíše všechny klienty zaměstnance pro výběr a vygeneruje formulář pro vložení nové smlouvy.

Uživatel vybere příslušného zaměstnance, vyplní formulář a odešle data.

Po odeslání dat systém zkontroluje jejich správnost a spustí případ užití: Převod bodů stromem

Systém uloží novou smlouvu a všechny změny ve stromě.

Alternativní toky: 1) Zaměstnanec nemá žádného klienta, který by mohl smlouvu zavřít: systém upozorní varovnou hláškou a nabídne vložení nového klienta (případ užití Přidání klienta)

Název: **Převod bodů stromem**

Stručný popis: Průchod stromové struktury zaměstnanců a přidání bodů a finančního ohodnocení patřičným zaměstnancům

Primární aktér: Systém, na základě předchozího případu užití spuštěného administrátorem nebo zaměstnancem

Předpoklady: Nová smlouva byla úspěšně uložena do systému.

Hlavní tok: Případ použití se spustí, když uživatel úspěšně uloží novou smlouvu.

Systém připočte zaměstnanci body do celkového součtu a do bodů získaných svou činností a na základě pozice zaměstnance vypočítá a připočte orientační částku kterou zaměstnanec dostane podle vzorce: body_za_smlouvu * cena_bodu (kdy cena bodu je dána pozicí zaměstnance)

Systém rekurzivně prochází stromem nahoru až po kořen stromu podle následujících pravidel:

Najde přímého nadřízeného zaměstnance.

Uloží body za novou smlouvu do celkového součtu bodů a do bodů získaných ze svého podstromu.

Vypočte orientační částku za smlouvu podle vzorce:

Body_za_smlouvu * (cena_bodu_ přímého_nadřízeného – cena_bodu_stávajícího_nalezeného) Konkrétní případ je uveden

v kapitole testování systému.

Uloží vypočtenou částku do celkového součtu financí získaných ze svého podstromu

Pokud není přímý nadřízený kořen stromu opětovně nalezne přímého nadřízeného a provede vše znovu.

Odebrání klienta a smlouvy – kaskádní mazání

Popis následujících případů řeší vymazání klienta (a tudíž i smlouvy) ze systému.

Název: **Odebrání klienta**

Stručný popis: Odebrání klienta ze systému

Primární aktér: Administrátor nebo zaměstnanec (uživatel)

Předpoklady: Uživatel přihlášen do systému. Existence klienta.

Hlavní tok: Případ použití se spustí, když uživatel vybere v seznamu klientů položku: smazat.

Systém zobrazí varovný dotaz, zda má být klient opravdu smazán.

Uživatel potvrdí smazání.

Systém spustí kontrolu existence smluv

Pokud nalezne alespoň jednu smlouvu, „spustí“ případ užití Odebrání smlouvy.

Systém vymaže klienta..

Alternativní toky: Jestliže má klient jakoukoliv aktivní smlouvu:

systém upozorní o jakou smlouvu se jedná a čeká na potvrzení smazání uživatelem.

Název: **Odebrání smlouvy**

Stručný popis: Odebrání smlouvy ze systému

Primární aktér: Administrátor nebo zaměstnanec (uživatel)

Předpoklady: Uživatel přihlášen do systému. Existence smlouvy.

Hlavní tok: Případ použití se spustí, když uživatel vybere v seznamu smluv položku: smazat.

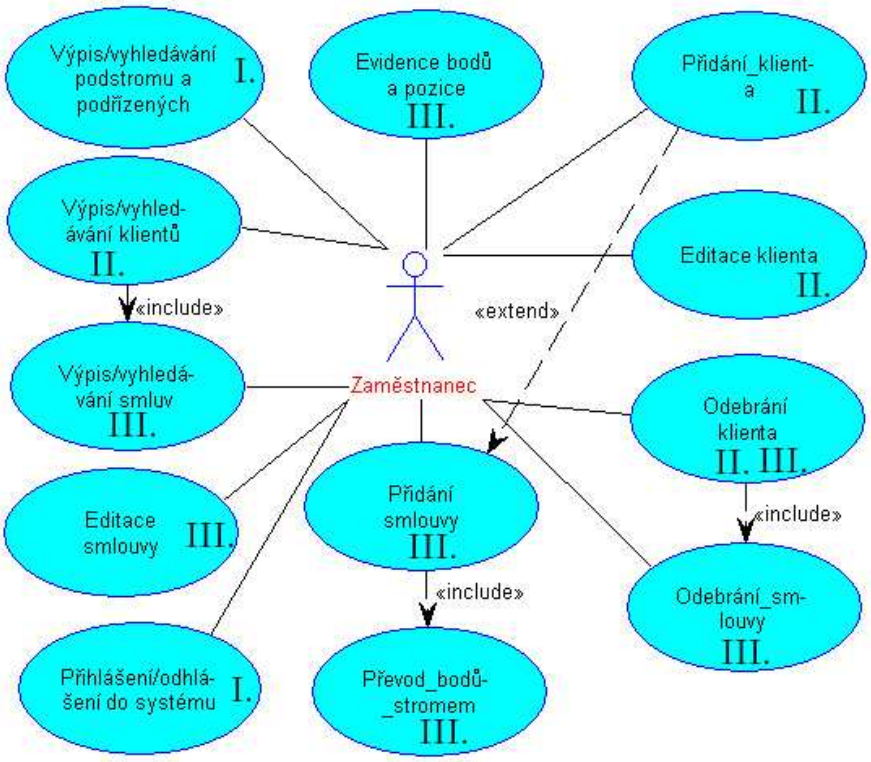
Systém zobrazí varovný dotaz, zda má být smlouva opravdu smazána.

Uživatel potvrdí smazání.

Systém spustí kontrolu zda není smlouva aktivní

Pokud nalezne příznak aktivity, upozorní uživatele a čeká na potvrzení/zrušení mazání.

Systém vymaže smlouvu.



Obr. 6: Diagram případů užití – Zaměstnanecká část

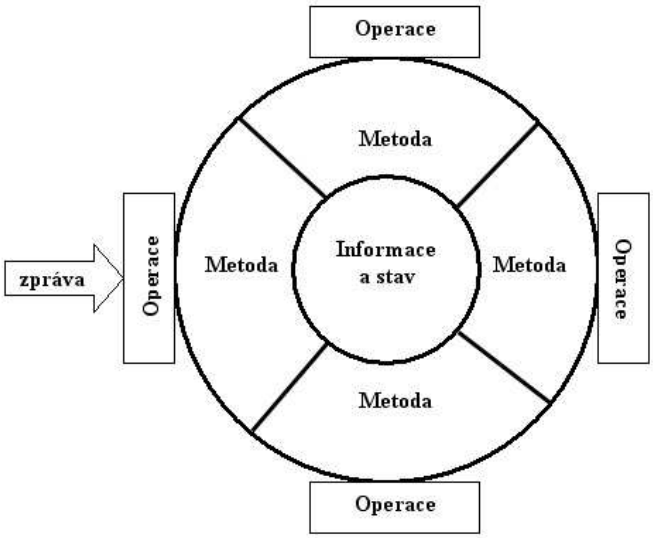
Případy užití zaměstnance vypsané na Obr. 4 jsou opět rozděleny do tří iterací, přičemž některé z nich zasahují do druhé i třetí iterace. Přidání/Editace/Odebrání klienta a smlouvy jsou identické s administrátorskou funkčností pouze s rozdílem, že zaměstnanec má právo tuto funkčnost spouštět pouze pro své klienty a smlouvy. Zajímavá případně složitější funkčnost těchto případů je popsána v administrátorské části. Ostatní z případů užití jsou součástí přílohy.

3.6 Modelování tříd objektů

Objekt je seskupením dat a funkcionality, které jsou spolu spojeny za účelem plnění požadované zodpovědnosti. Každý objekt má svou identitu, vlastnosti, chování a zodpovědnost. Vlastnosti objektu jsou jeho atributy, chování je realizováno metodami a každý objekt má jedinečnou zodpovědnost. Objekt poté poskytuje služby prostřednictvím operací. Objekty mezi sebou komunikují předáváním zpráv což zajišťuje, že změny struktury jsou zapouzdřeny uvnitř objektů a nerozšiřují svůj dopad do dalších částí systému což vede k větší stabilitě celého systému. Zapouzdření je klíčovou vlastností objektů (Kanisová,Müller, 2006). Schéma objektu, jeho vnitřního stavu, metod, operací a způsob komunikace pomocí zpráv prezentuje schéma na Obr. 5.

Objekty jsou organizovány ve třídách, které sdružují jejich společné vlastnosti. Třída představuje šablonu pro skupinu objektů (označovanou jako Instance). Tato Instance popisuje vnitřní strukturu objektu a všechny objekty stejné třídy mají stejně definované operace, atributy a metody.

Modelování tříd je klíčová aktivita objektově orientovaného vývoje, jelikož model tříd dává základ pro funkci jednotlivých objektů.



Obr. 7: Objekt a jeho vlastnosti - zapouzdření

Diagramy tříd zobrazují statickou stránku systému, především vztahy mezi třídami. Vztahy, které jednotlivé třídy navzájem pojí jsou asociace, agregace, kompozice a specializace/generalizace.

3.6.1 Atributy a operace tříd

Atribut je nositelem informace o objektu a je definován svým jménem a formátem a viditelností.

Název atributu jsem zvolil, tak aby jednoznačně pojmenovával danou vlastnost objektu. Z důvodu přehlednosti názvů atributů jsem zvolil u většiny tříd prefix atributu, který je tvořen vždy prvním písmenem názvu třídy, viz příklad Klient.kJméno z digramu na Obr. 6.

Formáty atributů jsou zvoleny na základě domluvy se zaměstnanci a jejich konkrétní typ je viditelný na diagramu tříd. Nástroj CASE CSA umožňuje definovat vlastní typy formátů podle zvoleného implementačního prostředí, na přání zaměstnanců ale není omezena délka řetězců u atributů formátu „string“ a „integer“.

Poslední charakteristikou atributů je jejich viditelnost. UML rozlišuje tři základní typy: Public, Private a Protected, které se liší dle přístupu k těmto atributům. Typ public umožňuje kterémukoliv elementu systému přistupovat k atributu a zbývající dva omezují přístup pouze operacím implementovaným v dané třídě, popřípadě na potomcích této třídy.

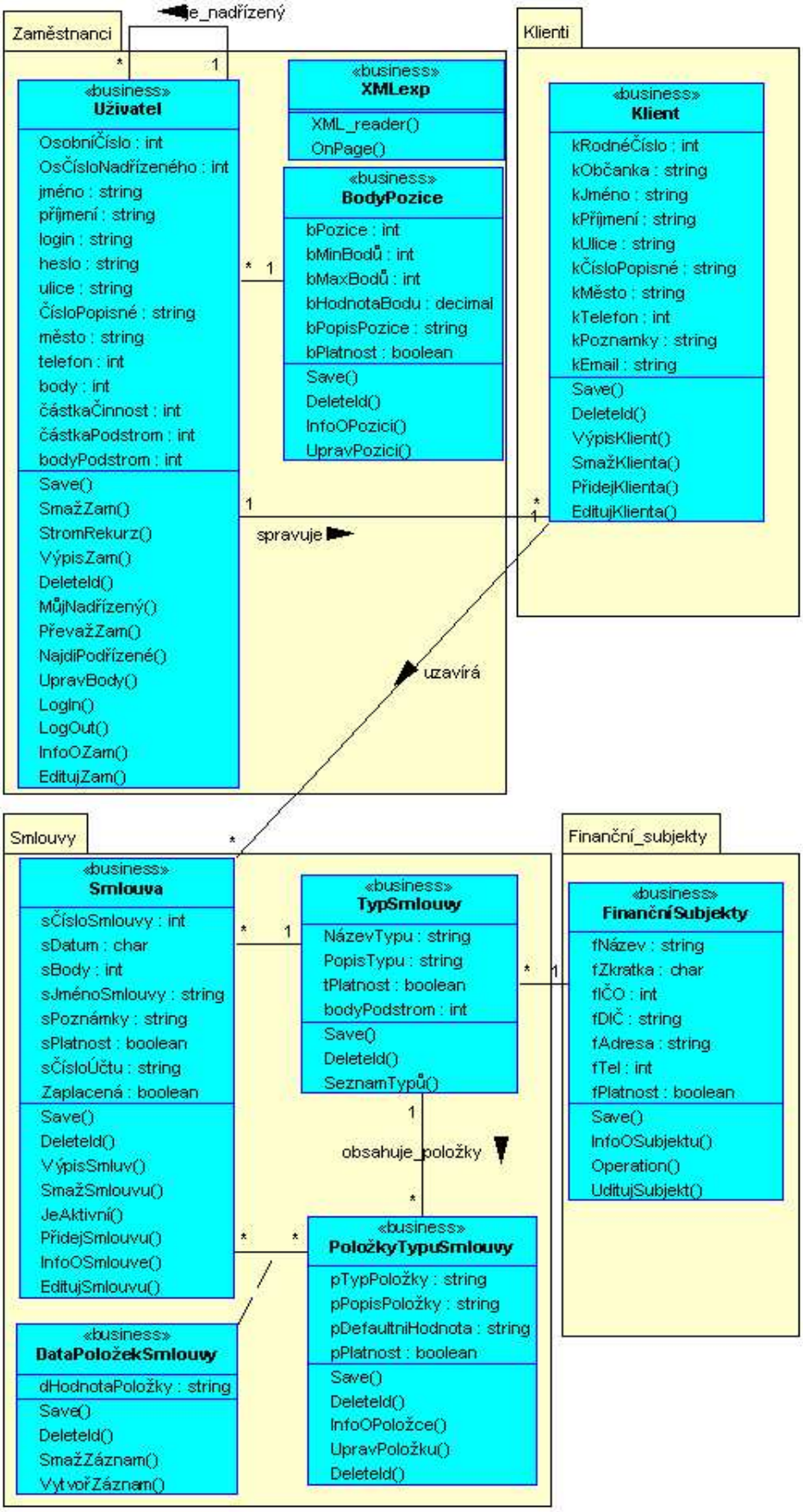
Operace tříd v modelu

Nedílnou součástí struktury objektu je jeho chování, které je definováno operacemi. Charakteristika operace (signatura) je dána jejím názvem, seznamem parametrů a návratovými hodnotami. Z důvodů přehlednosti je v diagramu tříd uveden pouze název operace. Seznam parametrů a návratových hodnot je v modelu definován v příloze.

Abstraktní třída

Asociační třídy dovolují přiřadit atributy, operace a další rysy k asociační vazbě, která řeší vztah mezi třídami typu mnoha ku mnoha. V diagramu tříd na Obr. 6 je modelována asociační třída DataPoložekSmlouvy ke vztahu mezi třídami Smlouva a PoložkyTypuSmlouvy. Každá smlouva je určitého předem daného typu, což je určeno vazbou na konkrétní objekt třídy TypSmlouvy. Ke každému typu smlouvy se vztahují předem definované položky. Uvedená asociační třída řeší problém, jak každé definované položce přiřadit požadovanou hodnotu, kterou má položka nabýt na konkrétní smlouvě. Existence objektu této asociační třídy je přímo spjatá s existencí konkrétní smlouvy a konkrétní položky, kterou tato smlouva obsahuje. Objekty asociační třídy z výše uvedeného důvodu zanikají spolu se smazáním objektů třídy Smlouva nebo PoložkyTypuSmlouvy.

V systému se asociační třída chová stejně jako jiné třídy, může mít proto vztahy na další třídy a má vlastnosti jako standardní třída. V implementaci pak dochází ke skutečnosti, že mezi dvěma objekty v určitém okamžiku je jen jedno jedinečné spojení, které nese informace o vlastnostech, které nebylo možno přiřadit ani jednomu objektu.



Obr. 8: Diagram tříd

3.6.2 Seskupení tříd

Problém dekompozice systému je řešen tvorbou seskupení tříd a používá se pro třídy, jejichž objekty spolu logicky souvisí a komunikují mezi sebou. Tyto třídy jsou pak schopny poskytnout ucelený balík služeb. Využití seskupení v diagramu tříd zjednodušuje udržení kontroly nad celkovou strukturou systému. Seskupení jsou rovněž velmi užitečná pro testování. Ačkoliv je možné testovat chování jednotlivých tříd, je mnohem užitečnější provádět testování jednotek na úrovni seskupení.

Třída má v daném seskupení unikátní výskyt, což znamená, že žádné jiné uskupení nesmí tuto třídu již obsahovat. Z pohledu modelování je možno třídu z jednoho seskupení zobrazit v diagramu jiného seskupení, musí ale nést příznak seskupení do kterého patří. Tohoto se využívá u rozsáhlejších systému, kdy jsou diagramy tříd rozděleny na více celků (Kanisová,Müller, 2006). V modelu systému této diplomové práce je diagram tříd zobrazen v jenom celku a proto seskupení obsahují přehledně všechny třídy které do nich patří. Mezi seskupeními existují závislosti, které vycházejí z asociací mezi třídami těchto seskupení.

Analytická seskupení jsou výhodná pro modelování webových služeb, jelikož tyto služby fungují na bázi komponentového vývoje. Jak je patrno z diagramu tříd na Obr. 6, rozdělil jsem seskupení na 4 celky – Zaměstnanci, Klienti, Smlouvy a Finanční subjekty. Z pohledu implementace téměř odpovídají tyto čtyři seskupení jednotlivým iteracím, ale jednotlivé iterace rozšiřují funkčnost i těch tříd v seskupeních, která již byla implementována dříve.

Seskupení Finanční subjekty jsem dodatečně modeloval samostatně. Finanční sektor je sám o sobě rozsáhlým komplexem, z něhož do tohoto systému zasahuje jen nepatrná část. Z důvodu možného rozšíření jsem již nyní vytvořil samostatné seskupení, které zatím pouze dodává informace o objektech finančních subjektů, které nabízejí jednotlivé typy smluv.

Seskupení Zaměstnanci odpovídá první iteraci systému. Hlavní třída uživatel sdružuje objekty, které se budou přihlašovat do systému pod svým uživatelským jménem a heslem. Administrátor systému je řešen jako kořen stromu celého systému a má proto nejvyšší práva. Je prvním objektem této třídy. Osobní číslo každého zaměstnance je dáno firmou a objekty této třídy jsou vzájemně stromově propojeny vazbou je_nadřízený, která jednoznačně identifikuje nadřízeného každého zaměstnance a zároveň určuje že každý zaměstnanec firmy může mít nula až neomezeně podřízených. Podle neformální specifikace systému se každému zaměstnanci budou načítat body za zřízené smlouvy do atributu body a také za smlouvy zřízené jeho podřízenými (respektive stromem všech jeho podřízených) do atributu bodyPodstrom. Součet všech těchto získaných bodů vyhodnocuje na jaké pozici se zaměstnanec nachází a jakou finanční hodnotu jeho body mají. Na základě této finanční hodnoty je poté do atributů částkaČinnost a částkaPodstrom vypočítávána orientační suma, kterou zaměstnanec již získal za svoje smlouvy a smlouvy ze svého podstromu. Číselník určující pozice a jejich parametry je modelován samostatnou třídou BodyPozice a jeho specifikace je součástí

přílohy diplomové práce. Každý objekt třídy uživatel je vazbou spojen s některým objektem třídy BodyPozice podle bodů, kterých dosáhl.

Třída XMLexp je samostatnou třídou bez vazby na jiné třídy. Důvod jejího zařazení do seskupení je vyvozen z případů užití, protože administrátor je jediným uživatelem systému, který má právo spouštět export a import dat z/do databáze. Druhým důvodem zařazení této třídy do seskupení je samotná funkčnost post-relační databáze Caché, která umožňuje exportovat a importovat data pomocí předdefinovaných funkcí volaných z třídy XMLexp vyděděné ze systémových tříd Caché.

Seskupení Klienti komunikuje mezi seskupením Zaměstnanci pomocí vazby „spravuje“ a zároveň zajišťuje provázanost tříd seskupení Zaměstnanci a seskupení Smlouvy vazbou „uzavírá“. Touto posloupností vazeb mezi objekty třídy Uživatel – Klient – Smlouva je dodržena základní podmínka firmy, kdy nesmí existovat objekt třídy Smlouvy aniž by existoval objekt třídy Klient který tuto smlouvu uzavřel a objekt třídy Klient musí mít vazbu na objekt třídy Uživatel (zaměstnanec).

Během druhé iterace vývoje systému bylo naimplementováno toto seskupení a testováno v návaznosti na prvním seskupení.

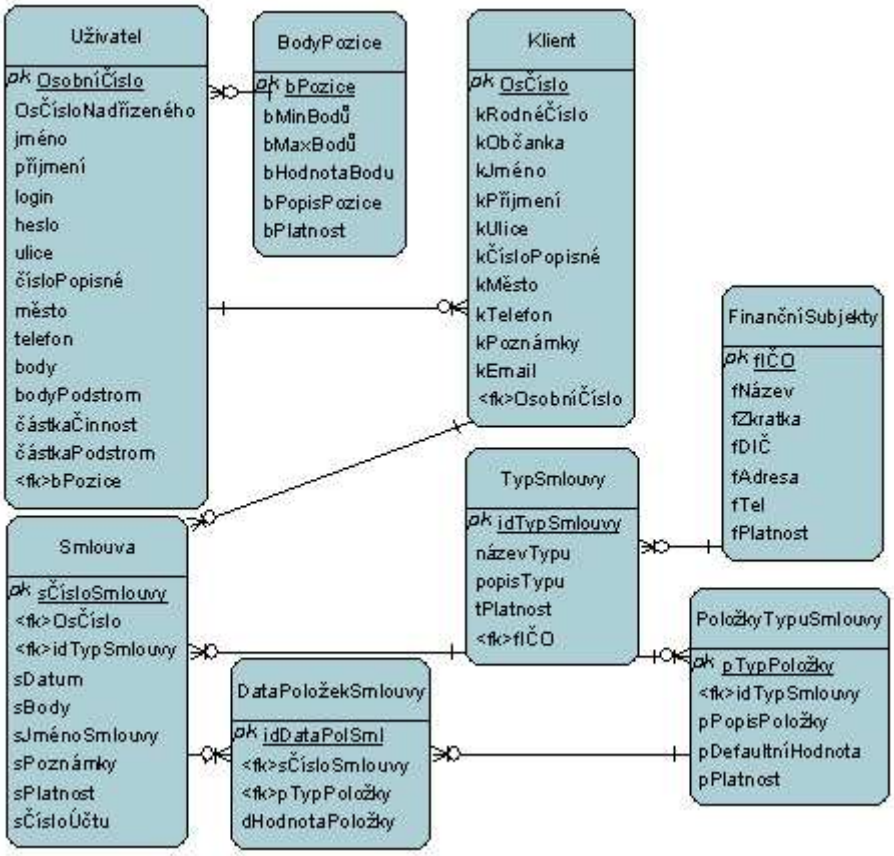
Seskupení smlouvy zahrnuje třídy tvořící dohromady funkčnost třetí iterace. Existence asociační třídy DataPoložekSmlouvy a její význam je popsán výše v této části kapitoly. Boolovský atribut sPlatnost třídy Smlouva zajišťuje rozlišení na aktivní a neaktivní smlouvy. Systém musí umožňovat evidenci jak stávajících aktivních smluv, tak i smluv, které již nejsou aktivní, ale byli s konkrétním klientem v minulosti uzavřeny.

Platnost objektů v číselnících určuje, které objekty budou nabízeny a zobrazovány ve formulářích nových smluv. Existence neplatných objektů je opět nutná z důvodů evidence neaktivních smluv.

3.7 Datové modelování – Logický model

Modelovaný informační systém je v rámci diplomové práce implementován v objektové databázi Caché, ale z důvodu kompletního implementačně nezávislého návrhu uvádím i logický datový model. Jedním z důvodů této části kapitoly je i fakt, že drtivá většina systémů stále používá relační databázové stroje (Oracle,MS SQL Server, Sybase) a jejich nahrazení objektovými systémy je v nejbližší době nepravděpodobné nebo zdlouhavé.

Diagram entit uvedený na Obr. 7 odpovídá datovému uložení podle diagramu tříd navrženému v předchozí části kapitoly na Obr 6. Atributy jednotlivých entit jsou rozšířeny o primární klíče (v diagramu označeny symbolem ***pk***), které zaručují unikátnost a cizí klíče (označeny symbolem <fk> které znázorňují vazbu do jiné entity – jsou primárním klíčem v jiné entitě.



Obr. 9: Logický model - ER

3.8 Model objektové spolupráce

Model statické struktury systému uvedený v minulé části kapitoly pomocí diagramu tříd není schopen popsat chování systému. Dynamickou strukturu systému popisuje model objektové spolupráce, který sestává z interakčních diagramů:

- Sekvenční diagramy
- Diagramy objektové komunikace

Pro dynamické modelování se opět využívá případů užití, tím že se pro konkrétní případy užití hledá jejich realizace. Realizací případu užití se rozumí takové třídy, které uskutečňují chování případu užití pomocí vzájemné spolupráce. Jedná se o převod slovního popisu scénáře na model interakce předem určených tříd.

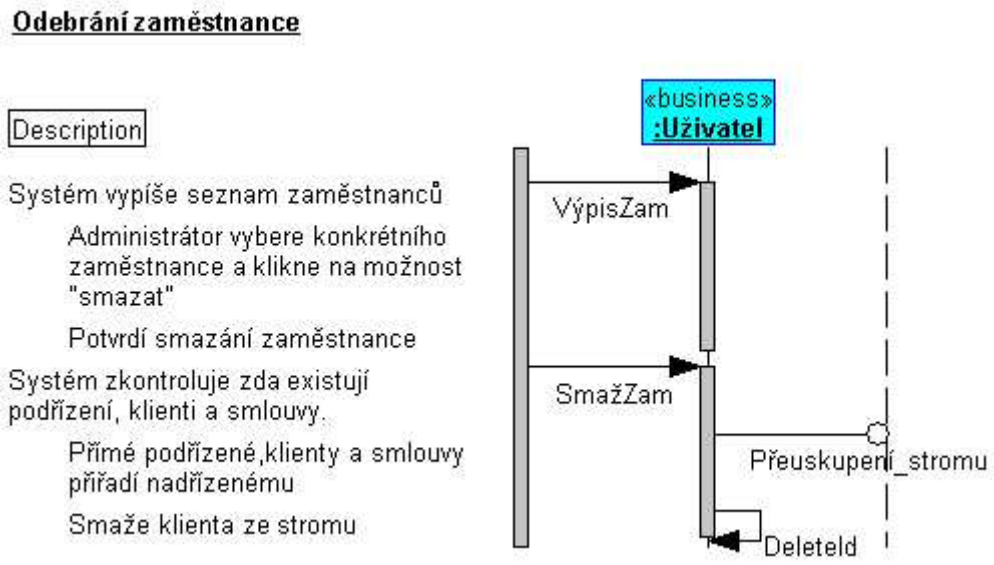
Proces hledání realizace případů užití je soustavným iteračním upřesňováním. Je třeba procházením jednotlivých případů užití modelovat způsob, jak požadované chování zajistit pomocí nalezené množiny analytických tříd a jimi poskytovaných operací (Kanisová,Müller, 2006).

3.8.1 Sekvenční diagramy

Sekvenční diagramy umožňují zachytit celkovou sekvenci chování a je možné je použít i pro zachycení paralelních procesů. Nástroj CASE SCA podporuje zobrazení popisu chování k jednotlivých sekvenčním diagramům a tak usnadňuje přehlednost a spjatost případu užití se samotným dynamickým chováním systému. Tento stručný strukturovaný text není částí standardní notace UML, ale uvádím jej pro přehlednost návrh systému a k popisu funkčnosti jednotlivých sekvenčních diagramů.

Sekvenční diagramy umožňují také znázornění vazeb mezi modelovanými případy užití, tedy vazeb typu <<include>> a <<extend>>. Na digramu je k tomuto případu použit symbol sondy. Sonda představuje odkaz na jiný případ užití, pro který může být zpracován vlastní sekvenční diagram. Pro vazbu typu <<include>> je použit symbol s prázdným kolečkem a pro vazbu typu <<extend>> s plným kolečkem.

Sekvenční diagramy systému jsou součástí přílohy v HTML. V této částí kapitoly opět uvádím jen nejpodstatnější nebo nejzajímavější diagramy z pohledu funkčnosti systému. Mnohé z případů užití jsou si chováním téměř identicky podobné, pouze s rozdílem volání operací nad jinou třídou.



Obr. 10: Sekvenční diagram – odebrání zaměstnance ze systému

Přeuskupení stromu

Obr. 11: Sekvenční diagram – změny ve stromu zaměstnanců při mazání zaměstnance

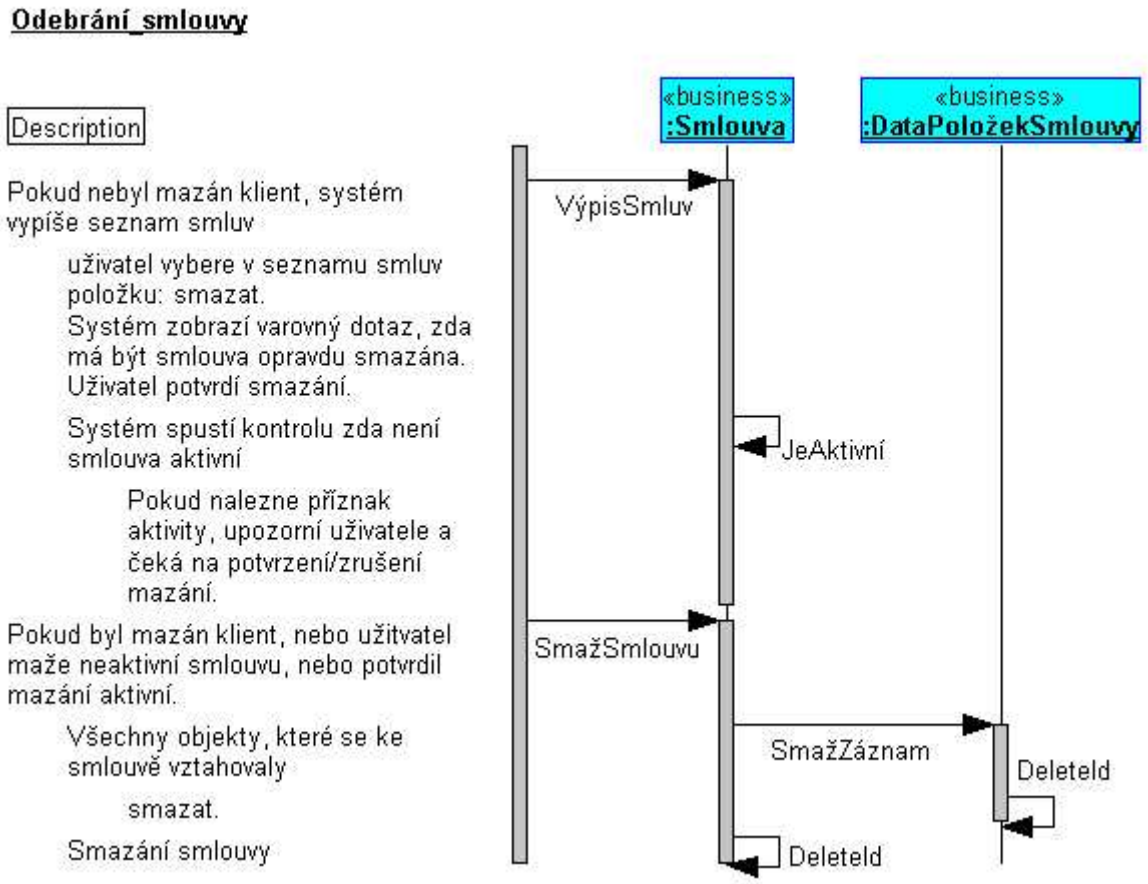
Odebrání klienta

Samotné odebrání klienta, zahrnující i odebrání jeho smluv je v praxi velmi ojedinělé. Tento případ užití je spouštěn převážně za okolností, kdy došlo k pochybení zaměstnance s přijímáním klienta, protiprávním přestupkům atd. Z hlediska modelu je tento případ složitější variantou díky kaskádnímu mazání. Systém musí vyhledat všechny smlouvy, které kdy zaměstnanec uzavřel a následně i všechny datové položky, které jsou na těchto smlouvách. Sonda „Odebrání smlouvy“ umožňuje znovupoužití jiného případu užití a samotný klient je smazán, až když jsou v rámci případu užití znázorněném na diagramu v Obr. 13 odstraněny všechny objekty DatPoložekSmlouvy a všechny Smlouvy klienta.

Obr. 12: Sekvenční diagram – odebrání klienta zaměstnanci

Odebrání smlouvy:

Odebrání smlouvy ze systému je případem, kdy je smlouva uznána za neplatnou (ať již je nebo není aktivní) a má být vyřazena. Jak ukazuje sekvenční diagram na Obr. 13, jsou kaskádním způsobem vymazány všechny objekty, které byly asociovány s konkrétní mazanou smlouvou a teprve poté je smazána smlouva. Případ užití rozlišuje, zda je volán pro samotné smazání jedné konkrétní smlouvy uživatelem, nebo zda je použito vztahu „include“ v rámci případu užití Odebrání klienta.



Obr. 13: Sekvenční diagram – odebrání smlouvy

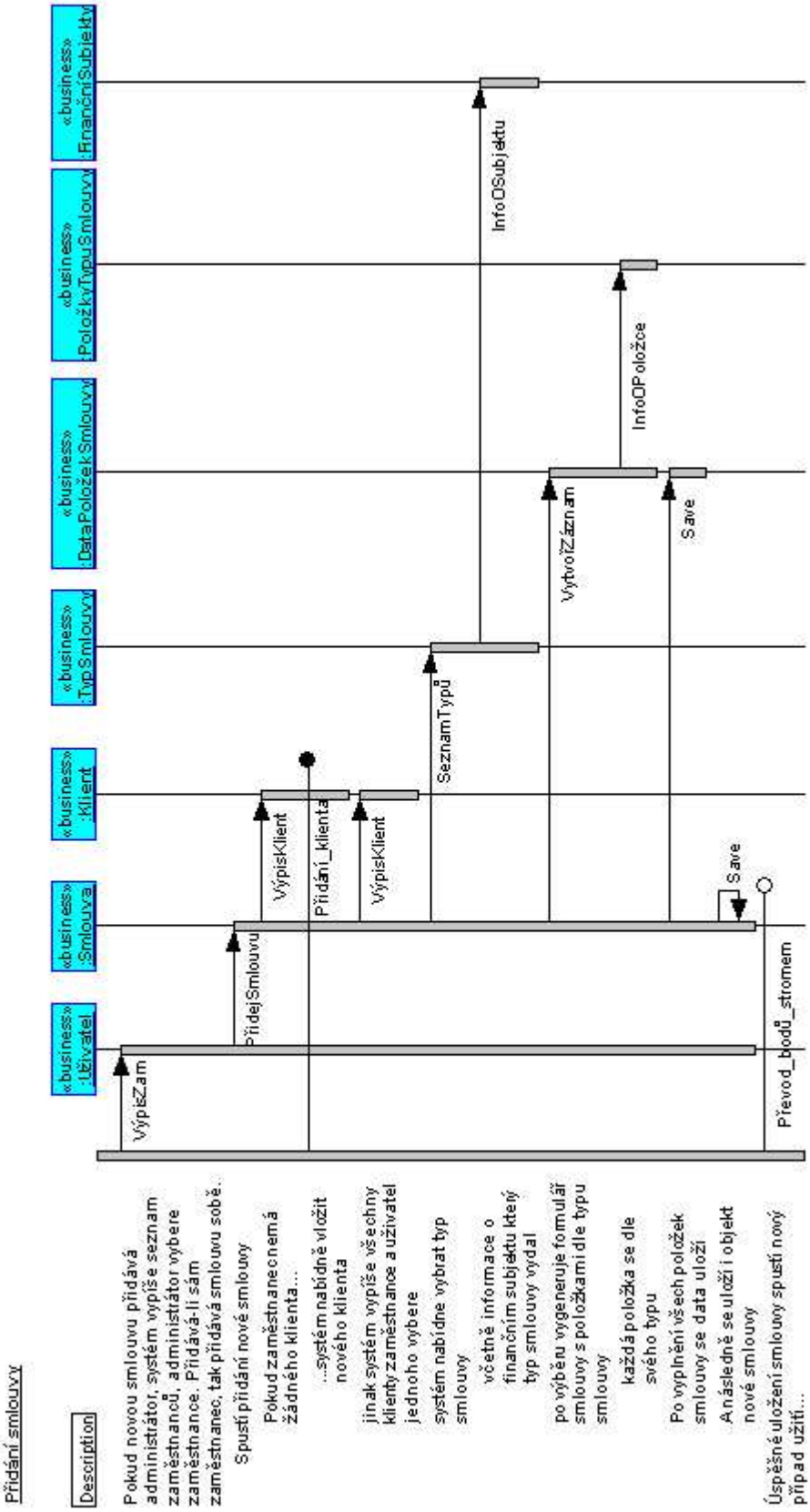
Přidání smlouvy:

Případ užití přidání smlouvy je nejsložitějším případem využití systému a také klíčovým pro celý informační systém. Sekvenční diagram na Obr. 14 popisuje průběh tohoto případu pro variantu, kdy případ užití spouští zaměstnanec nebo administrátor. Z výše uvedených důvodů jej uvádím celý v této části kapitoly. V diagramu jsou znázorněny dvě sondy. Sonda typu „extend“ umožňuje volání případu užití přidání klienta v případě, že uživatel chce přidat smlouvu zaměstnanci, který ještě žádného klienta nemá. Použití relace a sondy typu „extend“ je z důvodu samostatnosti existence případu užití přidání klienta.

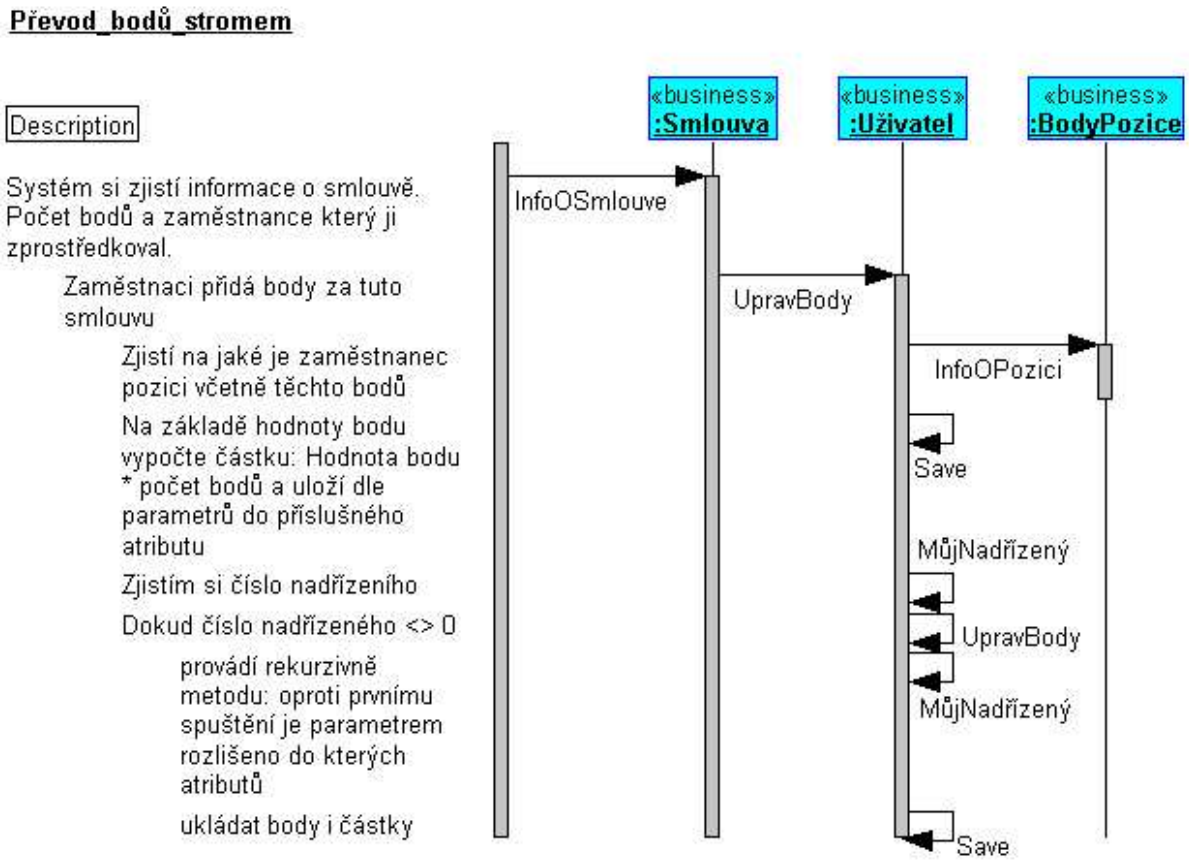
Oproti tomu sonda pro případ užití Převod bodů stromem po úspěšném uložení nové smlouvy je typu „include“, jelikož převod bodů stromem je závislý na vzniku nové smlouvy a jejího bodového

ohodnocení. Automatika převodů bodů stromem a s tím souvisejících výpočtů je znázorněna v následujícím diagramu na Obr. 15.

Položky každé smlouvy jsou automaticky generovány podle náležitosti smlouvy, ke které patří. Pokud je od finančního subjektu uvedena u některého položky přednastavená hodnota, je do formuláře vygenerována a pokud ji uživatel nezmění, následně uložena do dat položek smlouvy.



Obr. 14: Sekvenční diagram – přidání nové smlouvy



Obr. 15: Sekvenční diagram – převod bodů stromem po uložení nové smlouvy

Převod bodů stromem je případ užití, který musí rekurzivně projít celou část stromu od zaměstnance, který smlouvu zprostředkoval až po kořen stromu a přerozdělit body a částky za ně utržené.

V modelu systému je v diagramu tříd popsána metoda UpravBody(), která tuto rekurzi zajišťuje a rozlišuje. Má tři vstupní parametry:

První volání: boolovká proměnná, která pokud je metoda volána poprvé nese hodnotu logické jedničky. Pro všechna další volání logické nuly.

Body smlouvy zprostředkovatele: nese hodnotu bodů, které byly ke smlouvě zadány. Pokud je tato metoda volána poprvé ihned po úspěšném uložení nové smlouvy, jsou tyto body přičteny zaměstnanci rovnou, v opačném případě je s nimi počítáno jako s body získané u podstromu.

Číslo nadřízeného: při prvním volání této metody je v tomto parametru přenášeno osobní číslo zaměstnance, který smlouvu uzavřel. V každém dalším volání osobní číslo nadřízeného získané jako návratová hodnota volání metody MůjNadřízený.

Dokud není v parametru číslo nadřízeného přenesena hodnota 0, což značí kořen stromu, je metoda UpravBody() volána rekurzivně. Tím je docíleno, že zaměstnanci který body uzavřel jsou přičteny body do celkového součtu a do atributu ČástkaČinnost je přičtena hodnota kterou si dle své

pozice vydělal. Všem ostatním zaměstnancům, kterých se týká volání této metody jsou přičteny body do atributu BodyPodstrom a dle rozdílu pozice stávajícího zaměstnance a jeho podřízeného je vypočtena částka která je přičtena do atributu ČastkaPodstrom.

Sekvenční diagramy jsou z hlediska funkčnosti systému jedny z nejpodstatnějších, proto jsem uvedl ty nejdůležitější v této části diplomové práce. Sekvenční diagramy použitelné pro implementaci systému a popisující ostatní případy užití a jejich funkčnost jsou součástí přílohy modelu systému.

3.8.2 Diagramy objektové komunikace

Jsou druhou formou interakčních diagramů, někdy označované jako diagramy objektové spolupráce. Na těchto diagramech jsou objekty znázorněny obdélníky, vazby mezi nimi pomocí spojnic a šipky indikují zprávy zasílané mezi objekty v rámci případu užití. V podstatě se jedná jen o jiný způsob grafického vyjádření případu užití, kdy prvním případem jsou sekvenční diagramy. Diagramy objektové komunikace jsou schopny některé případy užití vyjádřit přehledněji, jelikož je možné lépe znázornit spojení objektů v rámci jejich spolupráce.

3.8.3 Úroveň diagramů interakce

Prvním stupněm kreslení je analytická úroveň, která se vyznačuje tím, že diagramy obsahují pouze bussines objekty.

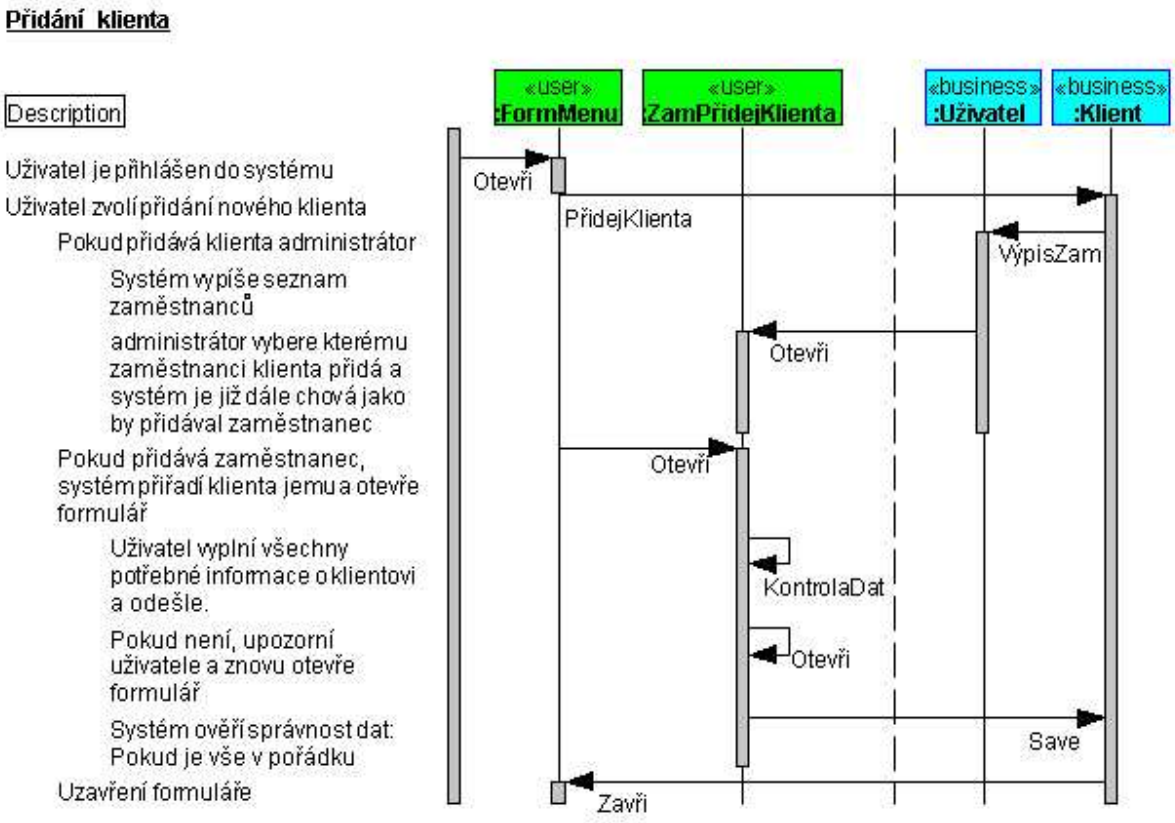
Druhým stupněm jsou diagramy, na kterých je zastoupen návrh (design) v podobě uživatelských formulářů. Tyto diagramy jsou znázorněny komunikací uživatele s formuláři (rozhranním systému) a dále komunikace mezi formuláři a bussines objekty. Jelikož modelovaný systém je plánován jako webová aplikace, je využití komunikace formulářů přehlednější a efektivnější pro pochopení funkčnosti sytému. Diagramy posloupnosti obrazovek jsem proto zařadil do modelu systému a v následující kapitole Implemetace systému v Caché jsem je doplnil i o grafické ukázky samotných webových stránek zformátovaných pomocí CSS stylů a JavaScriptu.

Pro ukázku diagramů interakce obsahujících i formuláře jsem zvolil případy užití přidání klienta a přihlášení do systému, které není zahrnuto jako žádný z případů užití, ale tvoří samostatnou funkčnost celého systému.

Přidání klienta:

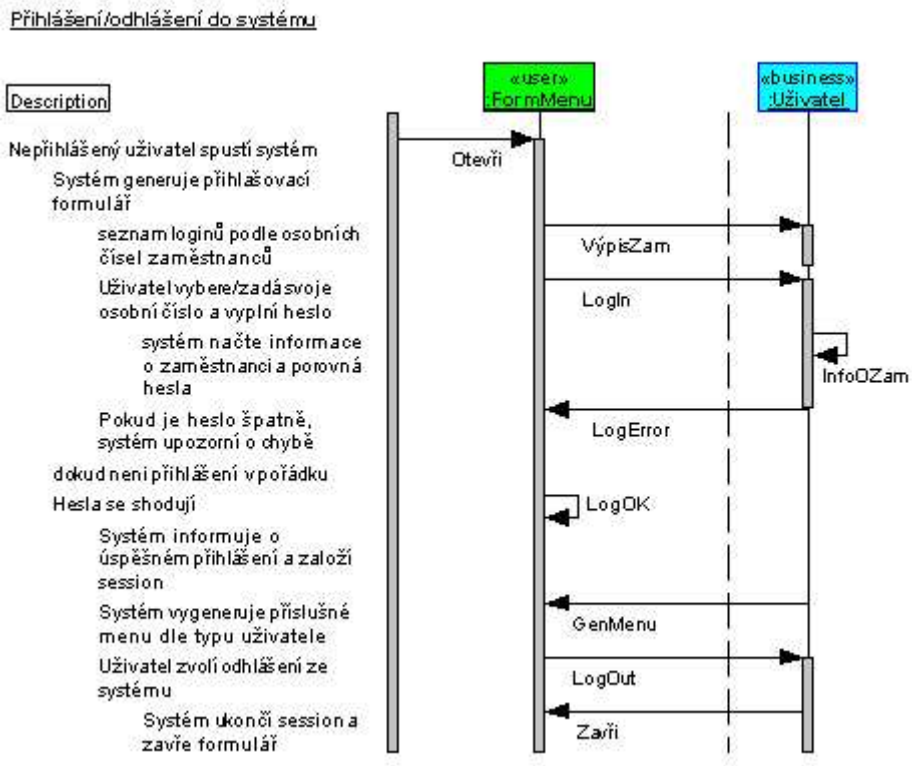
Případ užití přidání klienta může být spuštěn z několik míst v systému. Prvním rozdělení je, pokud jej pouští administrátor nebo zaměstnanec a druhým zda se jedná o samotné přidání klienta, nebo byl případ vyvolán v rámci případu užití přidání smlouvy.

Formuláře jsou v diagramu na Obr. 16 a Obr. 17 znázorněny jako <<user>> a na základě nich jsou volány jednotlivé metody tříd. Každý formulář je navíc odlišen barvou. V kompletním modelu systému jsou některé případy užití popsány pomocí obyčejných sekvenčních diagramů a některé i pomocí formulářů, pokud bylo použití formulářů zpřehledňující.



Obr. 16: Sekvenční diagram s formuláři – přidání klienta

Diagram obrazovek přihlášení do systému:



Obr. 17: Sekvenční diagram s formuláři – přihlášení a odhlášení systému

3.9 Stavové diagramy

Stavové diagramy jsou jednou z nejznámějších technik pro znázornění chování systému. Popisují všechny možné stavy, které může nabývat konkrétní objekt systému. Díky tomu modelují chování objektu napříč všemi případy použití, což žádný z předchozích diagramů neposkytoval. Zároveň znázorňují, jak se stavy objektu mění v závislosti na událostech, které se ho dotýkají (Kanisová,Müller, 2006).

Stavy

Stav objektu je charakterizován tak, že se jedná o konkrétní situaci, která nastala za doby života objektu a objekt během ní vyhovuje nějaké podmínce, realizuje konkrétní operaci nebo čeká na příchod události (stimulu). Tímto stimulem může být událost nebo požadavek na operaci. Objekt v průběhu svého života mění stav. Stav objektu je dán hodnotami atributů objektu, aktuálními spojeními s ostatními objekty a právě prováděnou operací. Každý diagram obsahuje právě jeden stav „start“ a minimálně jeden stav „stop“ kdy do stop stavu musí vést minimálně jeden přechod (Kanisová,Müller, 2006).

Přechody

Stav může obsahovat akce, které jsou nazývány přechody. Akce je považována za nepřerušitelný rychle probíhající proces. Akce stavu je spojena s interním přechodem, který nastává v důsledku jiné události a každý stav může mít libovolný počet akcí a interních přechodů (Kanisová, 2006).

Události

UML považuje událost za něco významného, co se stane v jistém čase a nemá trvání. Události jsou příčinami přechodů. Rozlišuje se několik typů událostí:

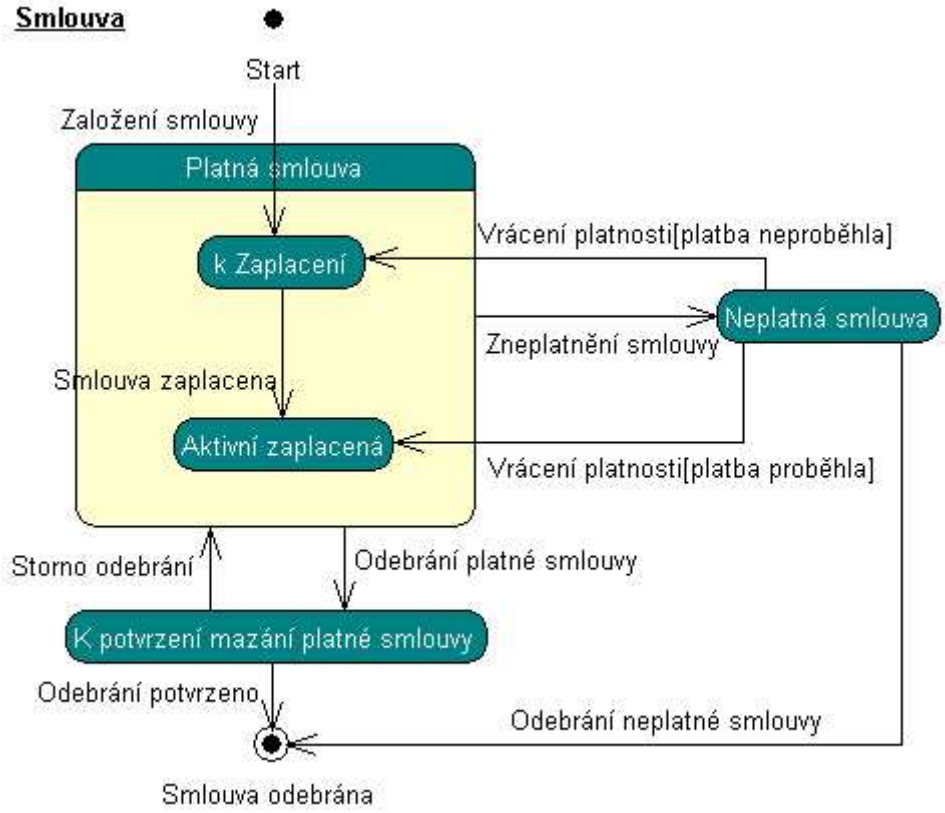
- Událost volání
- Signální událost
- Změnová událost
- Časová událost

Stavové diagramy modelovaného systému nejsou příliš komplikované a pohledu funkčnosti systému nezbytné. Z tohoto důvodu uvádím jen ukázku stavového diagramu objektu smlouvy a zaměstnance. Stavové diagramy číselníků jsou téměř identické, jelikož prochází jen stavy založení, platnosti/neplatnosti objektu a zrušení objektu.

Z pohledu možného rozšíření systému je pravděpodobný nárůst stavů jednotlivých objektů. Stávající požadavky na funkčnost vyhovují zadání zákazníků.

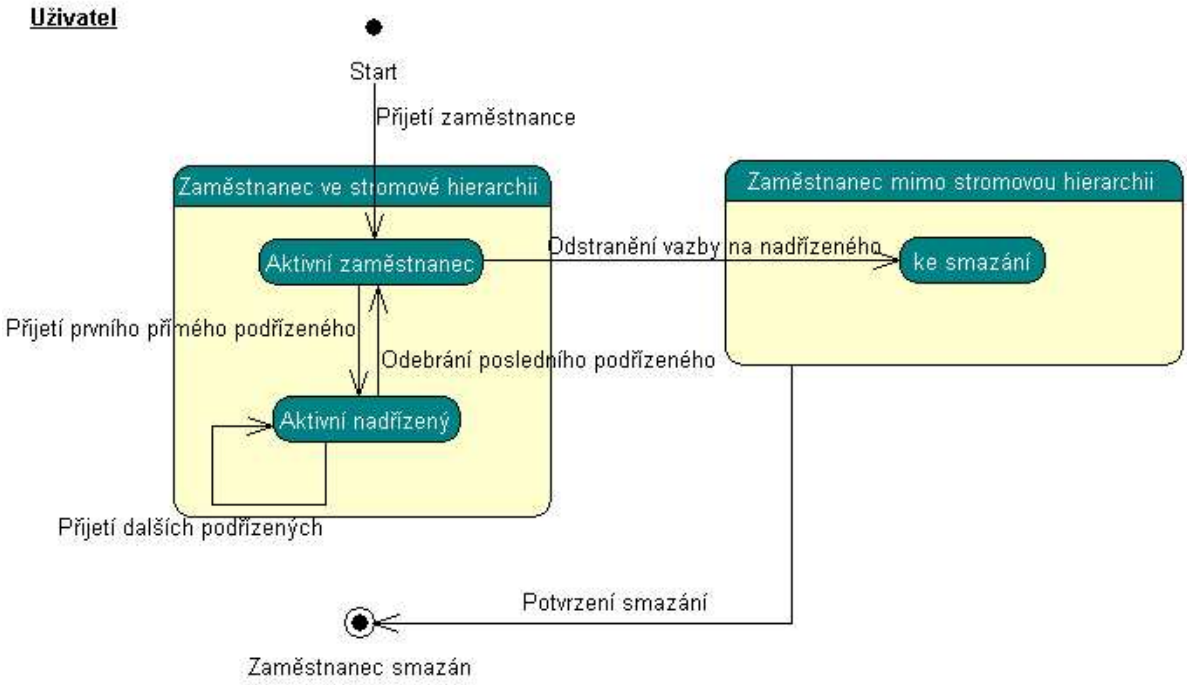
Stavový diagram smlouvy:

Objekt smlouva rozlišuje svoji platnost a neplatnost pomocí atributu sPlatnost. Uživatel může tuto platnost nastavovat libovolně ručně a přepínat mezi nimi. Platnost a neplatnost smlouvy rozhoduje o jejím umístění ve výpisech smluv. Platná smlouva nemůže být vymazána bez potvrzení uživatele, ať již se jedná o zaplacenou nebo nezaplacenou smlouvu.



Obr. 18: Stavový diagram - smlouva

Stavový diagram uživatel (zaměstnanec firmy):



Obr. 18: Stavový diagram - uživatel

3.10 Diagramy aktivit

Představují nejmladší a jednu z nejméně zažitých součástí jazyka UML. Tyto diagramy kombinují myšlenky různých jiných modelovacích technik. Užitečné jsou zejména pro popis chování, které má charakter paralelního zpracování. Zobrazují sekvenci aktivit, které podporují jak sekvenční, tak paralelní chování.

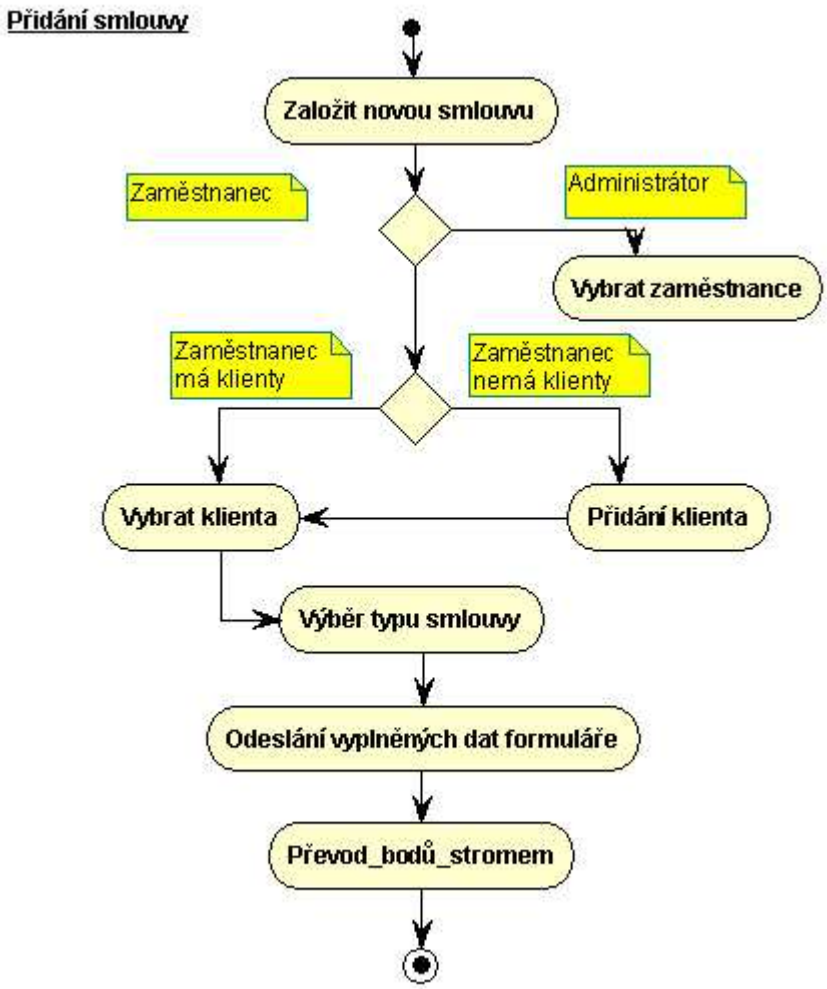
Diagram aktivit je v podstatě variantou stavového diagramu a má podobnou terminologii jako stavové diagramy. Na rozdíl od stavových diagramu jsou procesy modelovány jako kolekce aktivit a přechodů mezi nimi. Využití těchto diagramů je převážně v komunikaci s lidmi se znalostí struktury obchodních procesů (Kanisová,Müller, 2006).

Elementy diagramů aktivit.

Znázornění odlišné terminologie oproti stavovým diagramům je uvedeno následujícími symboly:

- Akce
  Je nedělitelnou jednotnou diagramu. Za akci se považuje stav dělání čehokoliv. Název stavu akce je stav konkrétního pracovního postupu.
- Přechody
  Mezi jednotlivými stavy dochází k přechodům, které nastávají automaticky bezprostředně po ukončení akce.
- Hodnocení přechodů
  Jedná se o element, pomocí kterého lze přerušit lineární zpracování na základě logické podmínky. Tato podmiňuje konkrétní přechod.
- Větvení a spojení
  Prvek umožňující modelovat paralelní chování. Rozdělí přechod na několik paralelně běžících vláken, které se následně spojí.
- Plavecké dráhy
  Vyjadřují kdo provádí jednotlivé aktivity podobně jako Diagramy procesních vláken.

V modelovaném systému jsem použil pro modelování případů užití stavové diagramy. Pouze jako ukázku diagramu aktivit a přehlednost uvádím diagram aktivity přidání smlouvy, protože se jedná o nejsložitější a nejrozsáhlejší s případů užití.



Obr. 19: Diagram aktivit – přidání smlouvy

4 Implementace systému v Caché

Za posledních 25 let vývoje počítačů a programování aplikací můžeme vysledovat mohutný trend přechodu od strukturovaného k objektově orientovanému programování. Toto platí i v oblasti zpracování dat a databází. V osmdesátých létech způsobily revoluci relační databáze, v létech devadesátých s mohutným nástupem objektově orientovaného programování začaly vznikat i objektově orientované databáze, které si kladou za cíl ulehčit a zrychlit práci s daty. Ovšem situace není zdaleka tak jednoduchá, protože relační a objektový přístup je od základu rozdílný. Existuje jak mnoho výhod, tak i mnoho nevýhod pro relační i objektové databáze.

Na současném databázovém trhu existují tři základní typy – relační databáze (Relational Database Management System, RDBMS), objektově-relační ("Object Relational" Database Management System, ORDBMS) a objektové (Object Database Management System, ODBMS). Přes nesporné výhody objektových databází stále nepatří k nejrozšířenějším a většina uživatelů databází volí stále relační databáze, popřípadě s objektovou nadstavbou nad relační databází.

Objektovou databázi Caché jsem si zvolil z důvodu jejího nevelkého rozšíření v ČR a v neposlední řadě také proto, že její unifikovaná datová architektura poskytuje jednu společnou vrstvu jak pro objektový přístup k datům, tak pro přístup prostřednictvím jazyka SQL k datům uloženým v transakčním vícerozměrném databázovém stroji.

4.1 Úvod do Caché

Společnost InterSystems uvedla Caché na trh v roce 1997. Označila ji jako „post-relační“ databázi. Z tohoto pohledu by se pojmem post-relační daly označit všechny databázové platformy, které nejsou založeny na normalizaci a prostých tabulkách, což by zahrnovalo relačně vnořené, vícerozměrné a objektově orientované databáze (Kirsten a spol, 2005).

Objektově orientované technologie jsou ideální pro modelovaní obchodních procesů v reálném světě a to je také jedním z důvodů výběru této databáze.

4.1.1 Kostka Caché

Caché cube – kostka Caché je hlavní přístupovou ikonou k ovládání celého databázového systému. Nabízí nejdůležitější rozcestník ke všem nástrojům a nastavením. Ze všech uvádím nejdůležitější:

- Start/stop Caché – pro spuštění a zastavení databáze.
- Studio – editor s grafickým rozhranním pro tvorbu tříd, stánek CSP a rutin, podrobnější popis studia je uveden v další části kapitoly.

- Terminal – terminálové prostředí v příkazovém řádku, umožňuje přihlášení k lokální i vzdáleným Caché databázím.
- Explorer – spravuje databáze a jmenné prostory s asociovanými třídami, globály a rutinami.
- SQL manager – správa týkající se relačního přístupu k Caché.

4.1.2 Caché studio – vývojové prostředí

Studio se spouští z kostky Caché a vyžaduje připojení k serveru Caché. Caché studio je pracovním centrem, v němž je vyvíjen celý projekt počínaje vytvářením a správou tříd, rutin, stránek CSP až po ladění a správu zdrojových kódů.

Caché studio je rozdělené okny do čtyř částí. Levou stranu tvoří panel se stromovou strukturou aktuálního projektu včetně jeho obsahu – seznamu balíčků, tříd, rutin, Caché serverových stránek atd. Všechny tyto typy tvoří projekt Caché a musí být ve stejném jmenném prostoru. Všechny s těchto částí také mohou být souběžně součástí několika projektů.

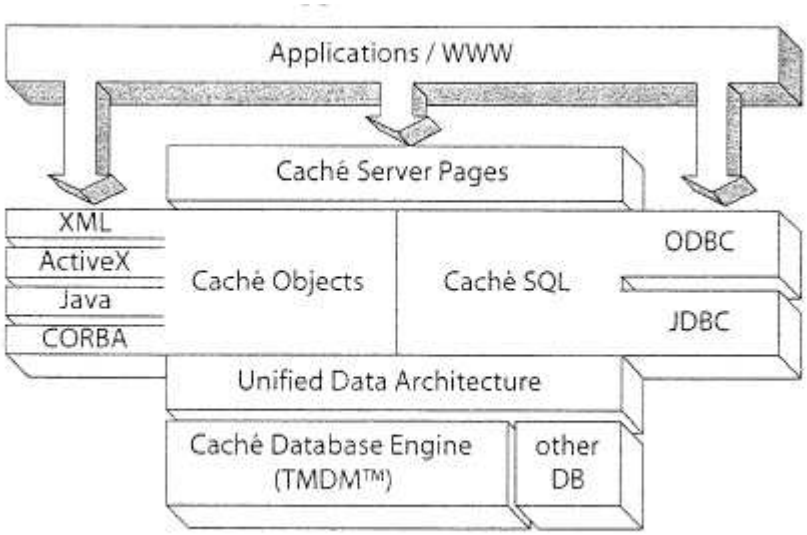
Největší část okna studia je určena pro zobrazení zdrojových kódů jednotlivých souborů. Dalšími dvěma části okna jsou inspektor a výstupní část. Inspektor zobrazuje v tabulkové formě definice objektových tříd a jejich prvků. Výstupní část je monitorovací panel pro výsledky různých akcí probíhajících v Caché studiu, například výsledky kompilací.

Implementace celého systému podle modelu navrženého v kapitole 3 je ve Caché Studiu, z tohoto důvodu věnuji popisu způsobu implementace jednotlivých použitých komponent kapitolu 4.3. Studio Caché je navrženo uživatelsky velmi efektivně, jelikož nabízí pro definování většiny komponent přehledné průvodce.

4.2 Objektový model Caché

Mnozí databázoví architekti a vývojáři řeší problém rozhodování mezi SQL nebo objekty. Caché řeší tento problém integrací obou přístupů. Unifikovaná datová architektura (UDA) poskytuje pro objekty a tabulky integrovanou popisnou vrstvu, která je součástí vícerozměrného databázového stroje. Podporou datových pohledů a rozhraní pro objektové i relační datové modely nabízí UDA nativní přístup k datům jak novým, tak i dříve vytvořeným objektům či relačním aplikacím (Kirsten a spol, 2005).

V této kapitole popisuji objektově orientované aspekty unifikované datové architektury (označované jako „objekty Caché“), použití persistentních objektů a částečně popis jazyka SQL pro Caché. Obr. 20 je vidět systémová architektura Caché uvedená v úvodu.



Obr. 20: Systémová architektura Caché

4.2.1 Vlastnosti objektů Caché

Objektový model Caché vyhovuje standardu ODMG skupiny Object Data Management Group. Základní operace pro objekty Caché jsou založeny na objektových třídách definovaných v Caché Studiu a na jejich následném překladu (kompilaci) do spustitelné podoby. Caché podporuje všechny principy objektových technologií pro vytváření, ukládání, načítání a manipulaci s instancemi objektů.

Dědičnost – Caché podporuje schopnost odvozování jedné objektové třídy od jiné a navíc podporuje i vícenásobnou dědičnost, při níž může být jedna třída odvozena od více než jedné z bázových tříd.

Polymorfismus – jedna metoda může být přiřazena objektům různých tříd, čímž taková metoda bude fungovat odlišně podle konkrétní třídy. Díky polymorfismu jsou objektové aplikace Caché zcela nezávislé na interní implementaci metod v každé objektové třídě.

Persistence – je nejdůležitější vlastností objektové databáze. Objekty jsou uloženy na záznamovém médiu a lze je v případě potřeby snadno a rychle získat. Caché podporuje různé typy objektové persistence. Automatické ukládání ve vícerozměrném datovém modelu Caché (které jsem použil pro vyvíjený systém), uživatelské ukládání v libovolných strukturách a ukládání pomocí brány Caché SQL Gateway v tabulkách externích relačních databází.

V návaznosti na tento základní princip má objektový model Caché některé další vlastnosti:

- Třídy jsou určeny definicemi tříd, které jsou uloženy ve slovníku tříd.
- Základní objekty jsou objekty a literály, přičemž literál nemá oproti objektu identifikátor.
- Objekty Caché svěřují uživatelům přímou kontrolu nad přístupem k jejich vlastnostem a validaci.

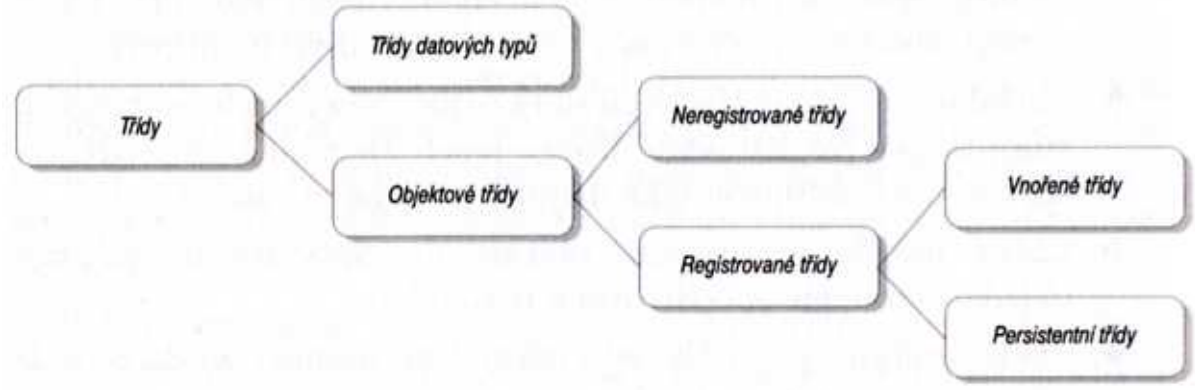
- Třídy mohou obsahovat generátory metod. Tyto slouží k překladu tříd a vygenerování vlastního kódu metod, které třídy využívají v běhovém modulu.
- Caché ukládá persistentní objekty do databáze na základě ukládací strategie specifikované uživatelem.

(Kirsten a spol, 2005)

4.2.2 Typy tříd

Caché rozlišuje mezi třídami definujícími datové typy a objektovými třídami. Třídy datových typů představují literály, což jsou například řetězce, celá čísla a další datové typy definované uživatelem. Objektové třídy obsahují objekty a dělí se dále na registrované a neregistrované třídy(Kirsten a spol, 2005).

Neregistrované třídy neobsahují žádnou vlastní funkčnost a je nutné vytvořit všechny odpovídající metody. Pro využití množiny předpřipravených metod Caché se využívají registrované třídy. Tyto mají registrované objekty, které jsou transientní a nemají tedy programový kód pro uchování informací v databázi. Persistentní objekty mají vlastní objektový identifikátor a lze je tedy uchovávat v databázi nezávisle. Jednotlivé typy tříd dále popisuji jen rámcově a jejich rozdělení je na Obr. 21. Podrobnější popis je věnován pouze persistentním objektovým třídám, které jsou definovány v rámci vyvíjeného modelu.



Obr. 21: Typy tříd v Caché

Třídy datových typů – definují a řídí hodnoty literálů. Datové typy nemají identifikátor a nemohou mít instance, existují pouze jako vlastnosti objektů.

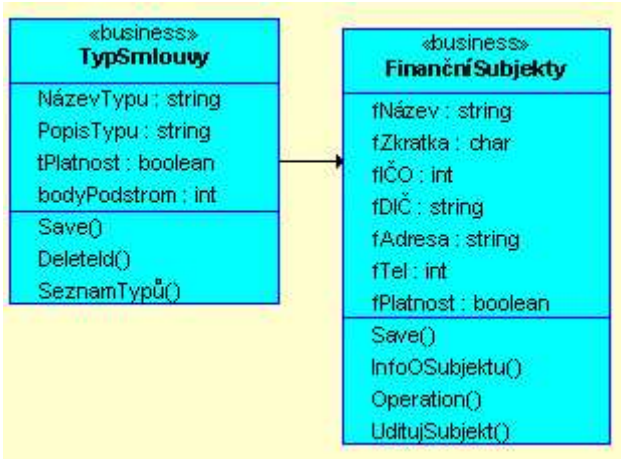
Objektové třídy – definují datovou strukturu a chování objektů určitého typu. Každá taková třída má jedinečný název, vlastnosti, metody a klíčová slova.

Neregistrované objektové třídy – Caché jejich instance nespravuje automaticky. Vývojář musí definovat jejich identifikátor a objektový odkaz. Oproti registrovaným mají omezení v alokaci úložného prostoru.

Registrované objektové třídy – obsahují kompletní množinu metod, které mohou řídit chování objektu z pohledu ukládání. Odpovídající metody se automaticky dědí ze systémové třídy %RegisteredObject.

Vnořené objektové třídy – jejich instance lze dočasně uchovávat v paměti, stejně jako trvale v databázi. Svoji funkčnost dědí ze systémové třídy %SerialObject. Jejich chování umožňuje, aby instance existovaly v paměti nezávisle na dalších objektech. Lze je ukládat do databáze pouze jako vnořené do dalších objektů.

Persistentní objektové třídy – obsahují kompletní podporu pro trvalé uložení svých instancí v databázi. Tato podpora se dědí ze systémové třídy %Persistent. Persistentní objekty mají objektovou identitu a jedinečný objektový identifikátor (OID). Každý persistentní objekt může výt v Caché uložen v databázi nezávisle. Použití persistentního objektu v roli vlastnosti objektové třídy se označuje jako odkaz na persistentní objekt jak je uvedeno na Obr. 22 z modelu systému nad třídami TypSmlouvy a FinančníSubjekty.



Obr. 22: Odkaz na persistentní objekt v Caché

Každá instance třídy TypSmlouvy se takto může odkazovat na instanci třídy FinančníSubjekty, ale každá instance třídy FinančníSubjekty existuje bez ohledu na to, zda a kolik typů smluv poskytuje. Z pohledu objektové třídy TypSmlouvy lze vlastnosti a metody odkazovaného FinančníhoSubjektu použít stejným způsobem jako jeho vlastní metody. Caché může použít tzv. swizzling, s jehož pomocí načte odkazované objekty do paměti.

4.3 Implementace tříd modelu v Caché – část serveru

Hierarchie tříd, typická pro objektovou metodologii, tvoří páteř aplikace. Vytváření hierarchie tříd proto vyžaduje plánování. Tento proces je úzce spojen s analýzou, protože objektový model popisuje problém, nikoli řešení implementace aplikace (Kirsten a spol, 2005). Podle navrhnutého modelu

popsaného v kapitole 3 jsem vytvořil hierarchii tříd. Díky hierarchii tříd je již k dispozici definice modulů a rozhranní aplikace. V této části kapitoly je popsáno vytvoření hierarchie tříd a jejich metod.

4.3.1 Definování tříd

Všechny objektové třídy jsem definoval v Caché studiu. Díky unifikované datové architektuře jsou automaticky vygenerovány tabulky (včetně sloupců a klíčových polí) ze všech definicí tříd, jejich vlastností a metod. Kompletní definice se realizují pomocí jazyka Class Definition Language (CDL) a jsou uloženy v databázi v určitém interním formátu.

4.3.1.1 Balíčky

Caché využívá balíčky pro seskupení souvisejících tříd do jednoho společného názvu. Na libovolnou z třídy určitého balíčku se lze odkazovat plným názvem, který se skládá z názvu balíčku a názvu třídy. Na základe modelu systému jsou třídy rozděleny do čtyř balíčků (zaměstnanci, klienti, smlouvy a Finanční subjekty). V Caché je poté využívána tečková syntaxe a například objekty třídy jsou Smlouva jsou volány: Smlouvy.Smlouva.

Vytvoření balíčků zefektivňuje přehlednost a testovatelnost celého implementovaného systému. S vytvořením nové třídy Caché automaticky vytvoří nový balíček a s odstraněním poslední třídy s názvem balíčku dojde k automatickému odstranění tohoto balíčku. Pro zajištění zpětné kompatibility Caché definuje automaticky dva interní balíčky %Library a User. Libovolný odkaz na třídu bez znaku % poté odkazuje na třídy balíčku User a třídy začínající znakem % patří balíčku %Library. Většinou se jedná o interní předdefinované třídy, z nichž pro návrh tříd modelovaného systému jsem využil třídu %Library.Persistent a %XML.Adaptor.

4.3.1.2 Jazyk Class Definition Language (CDL)

Definice tříd v Caché je možná pomocí přehledných průvodců nebo přímo pomocí CDL. Caché neukládá zdrojový kód definice třídy v jazyce CDL, ale vytváří jeho interní reprezentaci v databázi. V následném zobrazení zdrojového kódu po uložení dojde k odlišnostem ve formátování.

Každá třída je definována svým názvem a balíčkem do kterého je přiřazena, jak je vidět v ukázce z části zdrojového kódu definice třídy uživatel:

```
/// Třída definující jednotlivé zaměstnance firmy
Class ZAMESTNANCI.uzivatel Extends (%Persistent, %XML.Adaptor) [ Owner =
UnknownUser ] {
Parameter XMLNAME = "zamestnanec";

/// definice vlastností - atributů
Property osobnicislo As %Integer [ Required ];

/// definice vazeb – asociační vztahy s jinými třídami
```

```
Relationship klienti As KLIENTI.klient [ Cardinality = many, Inverse = owner ];

Method VypisZam() As %String
{
      /// definice těla metody
}
}
```

Každá třída z modelovaného systému je definována typem (`%Persistent`) aby byla zachována podpora pro trvalé uložení objektů v databázi a každý objekt obdržel svoji jedinečnou objektovou identitu. Jestliže se budou využívat bezpečnostní schopnosti jazyka SQL, je možné nastavit vlastníka třídy.

Dědičnost definované třídy z jiných tříd je uvedena v závorce za typem %Persistent, čárkami jsou odděleny jednotlivé třídy, ze kterých je děděno. Volba This class support XML, která se v definici třídy v CDL zobrazuje jako (`%XML.Adaptor`) sděluje Caché, aby byly automaticky k této třídě přidány metody pro projekci do XML. Objekty třídy děděné z XML.Adaptor jsou poté zahrnuty pomocí exportních metod do XML výstupu. Objektový model Caché je schopen exportovat objekty jednotlivých tříd podle vazeb mezi jednotlivými třídami.

Definovaní vlastností třídy

Definovaní vlastností třídy je další částí ukázky kódu. Názvy vlastností jsem definoval dle diagramu tříd objektového modelu z kapitoly 3. Je možné vybrat ze čtyř typů vlastnosti. Pro modelovaný systém jsem využil základní datové typy podporované v Caché jako „A single value of type“ jsou to %Integer, %String, %Date atd. Jako vlastnost lze vybrat i libovolnou definovanou persistentní či vnořenou objektovou třídu pro zajištění odkazu na objekty nebo na vnořený objekt. Ke každé vlastnosti třídy je možné nastavit několik charakteristik. Nejpoužívanější charakteristikou v implementovaném systému je volba Required, která určuje, že odpovídající vlastnost musí mít před uložením instance hodnotu. Toto nastavení ale nemá vliv v případě transientních objektových tříd. Volbou Indexed je zajištěno, že Caché automaticky vytvoří a bude spravovat index. Využití indexů je například pro zvýšení rychlosti provádění dotazů vyhledávajících záznamy podle konkrétní hodnoty dané vlastnosti. Volbou Unique vytvoří Caché pro tuto vlastnost index automaticky a zajistí, že žádné dvě instance nebudou moci nabít stejné hodnoty této vlastnosti. To zajišťuje jedinečnost hodnot klíčových polí, jakými je například osobní číslo zaměstnance, osobní číslo nadřízeného atd.

Definice vazeb mezi třídami

Volba Relationship určuje obousměrnou vazbu na vlastnost jiné persistentní třídy. Pro asociační vazby mezi jednotlivými třídami modelu jsem použil Relationship. Pojmenování vazby neodpovídá názvům vazeb z diagramu tříd, protože díky obousměrnosti vazby je nutné pojmenovat oba směry vazeb. Nastavením a pojmenováním parametru Inverse dojde ke spojení obou směrů vazby

a objektový model Caché se automaticky stará o vazby mezi objekty těchto tříd a nedovolí smazat objekty které jsou propojeny vazbou s objekty jiné třídy. Hodnotou parametru atributu Cardinality je nastavena kardinalita vazby mezi třídami.

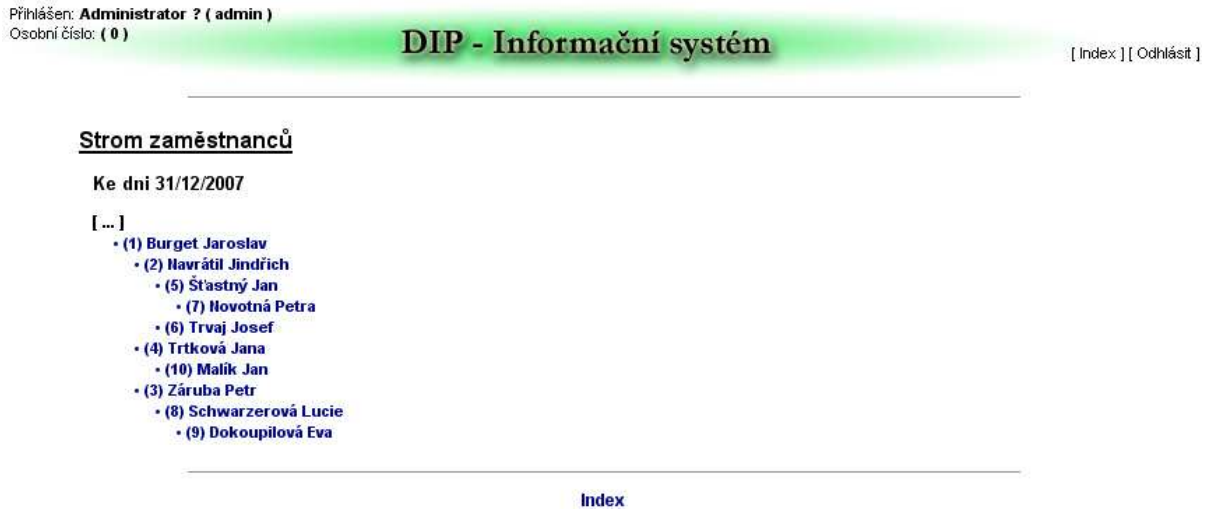
Definice metod třídy

Daný název metody lze zadat v rámci jedné třídy pouze jednou a názvy začínající znakem % jsou vyhrazeny pro metody systémových tříd Caché. Druhým krokem definice metod v Caché je zadání signatury, která popisuje rozhranní metody, počet, význam a typ jejích parametrů a návratovou hodnotu. Pro návratovou hodnotu jsou povoleny všechny datové typy Caché, ale v mnoha případech používá typ %Status, který nese informaci o tom, zda provedení metody bylo úspěšné či neúspěšné. Výběrem chování metody Private se určuje zda bude možné tuto metodu použít mimo třídu jejíž je součástí. Metody takto neoznačené jsou považovány za veřejné a jejich volání nemají žádná omezení. Pokud je v rámci metody volána instance třídy nad kterou je metoda definována, používá se syntaxe ##this.jméno_vlastnosti. Příkaz Quit slouží k ukončení metody Caché ObjectScriptu a přiřazení její návratové hodnoty.

Metody implementovaného systému jsou součástí exportu systému v XML a funkčnost jednotlivých částí systému je popsána v následující části v kapitoly 4.4. V této části uvádím definici metody rekurzivního volání stromu nad třídou uživatel a následně na Obr. 23 ukázku jejího výsledku zformátovaného pomocí HTML v Caché Server Pages.

```
Method Stromrekurz(osobnicislo As %Integer, hloubka As %Integer) As %Status
{
  Set  dotaz  =  "SELECT  %ID  FROM  ZAMESTNANCI.uzivatel  WHERE  oscislonadrizeneho='"
_osobnicislo _ "' ORDER BY prijmeni"
  Set rset = ##class(%Library.ResultSet).%New()
  Do rset.Prepare(dotaz)
  Do rset.Execute()
  while (rset.Next()) {
    set id = rset.Data("ID")
    set p = ##class(ZAMESTNANCI.uzivatel).%OpenId(id)
    for i = 1:1:hloubka
  {w "     "}
  if (hloubka < 6) {
         w "<span class=user_in_tree_small><a href=""zaminfo.CSP?id=", p.%Id(),
         """>&#149",p.LogIn(),"</a></span><br>"
  } else {
         w "<span class=user_in_tree_small><a href=""zaminfo.CSP?id=", p.%Id(),
         """>&#149",p.LogIn(),"</span></a><br>"
  }
  do p.Stromrekurz(p.osobnicislo,hloubka+1)
    } kill p
  Quit $$$OK
  }
}
```

Implementace této metody slouží k ukázce funkčnosti Caché v přístupu k instancím třídy pomocí SQL kombinované s Objektovým přístupe. SQL dotazem je metoda volána pro zaměstnance, jehož osobní číslo je předáno parametrem. Výsledná získaná množina je již objektovým způsobem zpracována pomocí metody %OpenId, která získává objekt z databáze a vytváří v paměti verzi obsahující hodnoty všech vlastností objektů. Následně je podle hloubky stromu zformátován textový výstup se jménem a osobním číslem zaměstnance a metoda je rekurzivně volána tolikrát, kolik je nalezeno zaměstnanců v podstromu. Každý uzel stromu je vytvořen jako hypertextový odkaz k osobním informacím konkrétního zaměstnance. V samotném systému se první volání této metody předává s parametry právě přihlášeného zaměstnance (popřípadě administrátora) a proto je rekurzivně vypsán celý podstrom přihlášeného zaměstnance.



Obr. 23: Ukázka výpisu podstromu zaměstnanců pomocí metody Stromrekurz

4.3.1.3 Jazyk Caché ObjectScript

Jazykem Caché ObjectScript jsou definovány metody tříd. Rozlišuje mezi dočasnými daty, která existují pouze v operační paměti a trvalými daty, která jsou trvale uložena v databázích (persistentní data). Persistentní data jsou základem pro vícerozměrný přístup v Caché ObjectScriptu. Oproti tomu dočasná data jsou spojena s určitým konkrétním procesem a nikdy s nimi nemůže pracovat jiný proces. Jsou proto označována lokální proměnnou.

Caché ObjectScript umožňuje definovat proměnné, využívá operátory a výrazy podobně jako většina jazyků využívajících skripty jako jsou PHP, Pearl atd.

Kromě implementace metod tříd je možné pomocí Caché ObjectScriptu definovat Rutiny. Rutiny definují procedury a skládají se z jednotlivých bloků, které jsou vytvářeny a definovány jako zdrojový kód. Při uložení se automaticky překládají do objektové formy nezávislé na platformě. Oproti metodám je rozdílné volání Rutin pomocí syntaxe Do ^Jméno_rutiny (parametry).

4.4 Caché Server Pages – část klienta

V předcházející podkapitole 4.3 jsem popsal způsob definice uložení dat v Objektové databázi Caché, uložení vazeb mezi třídami, jak jsou definovány vlastnosti a metody. Popis jednotlivých tříd a funkčností metod byl již uveden ve třetí kapitole v rámci modelu systému a jejich konkrétní funkčnost včetně ukázek na implementovaném systému je popsána v této kapitole o klientské části systému. Jelikož byl celý systém od počátku navržen jako webová aplikace, využil jsem pro jeho implementaci Caché Server Pages (CSP).

Jednotlivé dynamické stránky CSP jsou součástí celého projektu v Caché a ve Studiu Caché jsou přehledně zobrazeny v části CSP Files. Jejich editace a propojení na Objektovou databázi je efektivně podporováno aplikacemi Caché.

4.4.1 Základy CSP

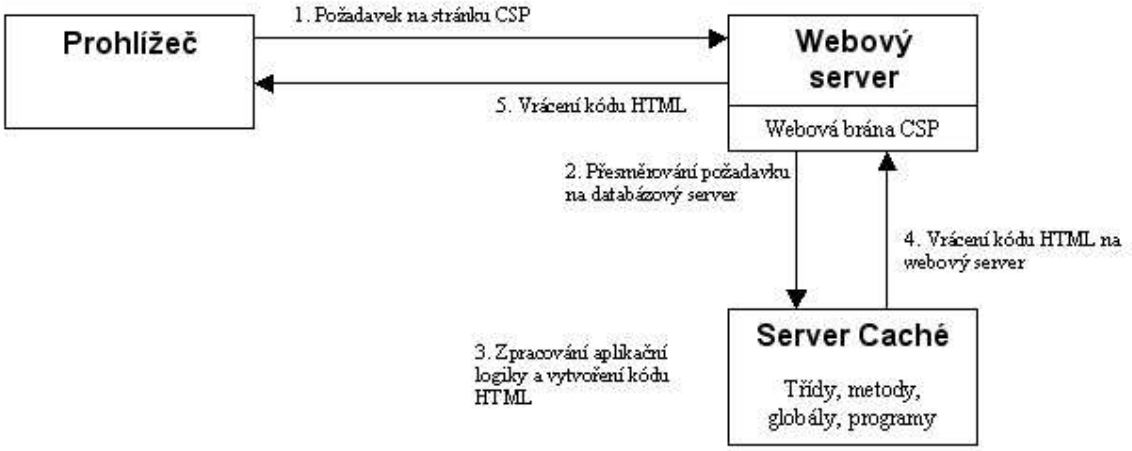
Caché a webový server umožňují dynamicky vytvářet obsah na základě získávání aktuálních informací z databáze a prezentovat je ve webovém prohlížeči. Základní technologií jsou Caché Server Pages (CSP). Dynamický webový obsah je závislý na čase, vazbách v uložených datech, uživatelských v stupech atd. Využívá principu HTML stránek, generovaných na straně serveru Caché pokaždé, když je stránka vyvolána, a vrací individuální obsah stránky CSP. V CSP může být kromě HTML integrován i XML, obrázky a libovolné další binární soubory.

Stránky CSP lze vyvíjet dvěma odlišnými způsoby:

- Jako soubory HTML nebo XML s vnořenými speciálními značkami CSP a prvky jazyka CSP. Caché tyto soubory převádí na definice tříd a překládá je do spustitelného kódu metod – označován jako značkový vývoj.
- Přímo jako třídy Caché, které jsou podtřídami %CSP.Page. Výkonná logika a požadované značky HTML a XML jsou součástí metody OnPage() – označován jako kódový vývoj.

Technologie CSP poskytuje potřebné nástroje pro oboje přístupy a zaručuje bezproblémovou interakci.

Pro spolupráci webového serveru a serveru Caché je nutné na webový server nainstalovat rozhranní Caché. Webová brána Caché je poskytována mimo jiné i pro Microsoft IIS a Apache Web Server. K rozeznání, zda je požadavek z webového prohlížeče směřován pro vlastní webový server nebo pro Caché, využívá brána příponu souboru. Požadavky s příponou .csp nebo .cls jsou směřovány do Caché (Kirsten a spol, 2005). Integrace webového serveru a databázového serveru Caché pro zpracování stránek CSP je na Obr. 24.



Obr. 24: Integrace webového a databázového serveru pro CSP

Webový a databázový server mohou být umístěny na stejném nebo více síťových počítačích a komunikují spolu prostřednictvím protokolu TCP/IP.

4.4.2 Objektový model CSP

Stránky CSP jsou kompletně integrovány do objektového modelu Caché. Každá ze stránek, ať už jde o stránku vnořenou nebo vytvořenou jakou soubor .csp a automaticky zkompilovanou nebo stránku přímo naprogramovanou v Caché studiu, je vždy podtřídou třídy %CSP.Page. Pro systémovou třídu ani pro odvozené třídy CSP nejsou nikdy vytvářeny instance, jelikož kompletní funkčnost je implementována ve statických metodách tříd, které lze spouštět bez vytváření instancí tříd. Caché automaticky vytváří instance pro zbylé třídy související s CSP a dává k dispozici odpovídající objektové odkazy. Objektový odkaz slouží k odkazu na příslušné instance:

- %CSP.Request Informace o způsobu volání http, včetně předaných proměnných CGI.
- %CSP.Session Vlastnosti aktuální relace CSP, včetně aplikačních dat.
- %CSP.Response Vlastnosti ovlivňující, jak Caché odpovídá webovému prohlížeči.

Kromě kódu HTML jsou stránky CSP tvořeny prvky generující dynamický obsah. Tato dynamická data jsou k dispozici ve formě objektů Caché, množin výsledků, proměnných a dalších prvků. Ze všech uvádím ty kódové prvky, které se vyskytují v CSP stránkách implementovaného systému:

- Výrazy ve tvaru #()# jsou nahrazeny hodnotami při generování stránky. Například zobrazení aktuálního data.
- Značky CSP ve formátu <CSP:xxx> poskytují zabudovanou a uživatelem definovanou funkčnost. Například pro podmínky který uživatel má právo na volanou CSP stránku.

- Skripty Caché provádějí kód napsaný v Caché ObjectScriptu a generují tak stránky za běhu. Jsou v kódu uzavřeny mezi skriptové značky ve tvaru <script language=cache runat=server>…</script>

(Kirsten a spol, 2005)

4.4.3 Autentizace

Po popisu základních principů a funkčnosti CSP stránek se od této podkapitoly zabývám již samotným implementovaným systémem. Prvním krokem v implementaci systému bylo vyřešení autentizace uživatelů systému. Caché v tomto směru nabízí řešení ve stavových režimech aplikace. Při práci v tomto režimu existuje privátní relace (session) mezi webovým serverem a databází, takže každý uživatelský požadavek musí být vyřízen v rámci svého vlastního serverového procesu Caché.

4.4.3.1 Rozdělení rolí uživatelů, přihlášení a odhlášení ze systému

Stromová hierarchie firmy každého ze zaměstnanců definuje jako uživatele systému se stejnými právy. Přístup ke konkrétním datům je omezen navíc jejich pozicí ve stromové hierarchii. Kvůli tomuto rozdělení jsem vytvořil dvě role uživatelů přičemž všichni uživatelé jsou objekty třídy uživatel.

První z rolí má jen jednoho uživatele a tím je administrátor. Podle návrhu modelu jsem administrátora definoval jako kořen stromu, který není nikomu podřízen a má přístup ke všem údajům všech zaměstnanců ve stromové hierarchii. Má stejné možnosti jako by byl jakýmkoliv ze zaměstnanců systému a navíc může spravovat číselníky systému. Práva administrátora jsou přehledně zobrazeny v předcházející kapitole 3 v diagramu případů užití.

Druhou rolí jsou všichni zaměstnanci firmy. Každému z nich je ihned při uložení do databáze vytvořeno přihlašovací jméno a heslo. Pozici zaměstnance ve stromové hierarchii určuje číslo jeho nařízeného, které se může v průběhu jeho existence i firmy měnit. Každý ze zaměstnanců má práva znát svého nadřízeného a mít přístup k informacím všech zaměstnanců, které jsou ve stromové hierarchii pod ním.

Přihlášení

Žádné jiné role uživatelů systému nejsou definovány a není možné se k informacím zaměstnanců, klientů a smluv dostat bez přihlášení v roli zaměstnance nebo administrátora. Tato bezpečnost je řešena pomocí stavového režimu Caché, kdy využívám privátní relace (session). Při vstupu na úvodní webovou stránku systému je zkontrolováno, zda je přihlášen uživatel v některé ze dvou výše popsaných rolí. Pokud není nastavena správně privátní relace $Data(%session.Data("prihlasen")) systém vybídne k výběru uživatele systému a zadání přihlašovacího hesla. Po úspěšném ověření hesla ke konkrétnímu uživateli jsou do session uloženy

potřebné přihlašovací informace a je vygenerováno příslušné menu podle role uživatele CSP stránkou index.CSP. Po celou dobu přihlášení uživatele v systému je v horní části obrazovky prohlížeče zobrazeno který zaměstnanec pod kterým uživatelským jménem a osobním číslem je přihlášen.

Odhlášení

Server Caché ve stavovém režimu za určenou dobu automaticky ruší privátní relaci smazáním informací v proměnných session. Tato doba lze nastavit pomocí utilit Caché. Po delší době nečinnosti je proto uživatel systému automaticky odhlášen.

Druhou možností odhlášení je kliknutí na odkaz [Odhlásit] v pravé horní části okna prohlížeče., který odkazuje na CSP stánku logout.CSP. Tato odhlašovací stránka příkazem Caché ObjectScriptu set %session.EndSession = 1 ukončí session a uživatele upozorní na odhlášení ze systému.

4.4.3.2 Zabezpečení CSP stránek a práva dle rolí uživatelů

CSP stránky systému jsou přístupné jen zaměstnancům firmy a administrátorovi. Proto jsem musel zajistit aby jejich obsah nebyl viditelný bez přihlášení. Každá z CSP stránek při jejím volání ověřuje, zda je přihlášen některý uživatel a jestli role uživatele má práva na tuto stránku.

Role uživatelů je rozlišena v session podle toho, zda se jedná o administrátora nebo zaměstnance. V případě, že CSP stránka může být spuštěna jen s administrátorskými právy, je neoprávněný uživatel upozorněn, že nemá právo stránku zobrazit. CSP stránky pro role uživatelů zaměstnanec jsou přístupné i s právy administrátora a proto je pouze kontrolováno, zda je někdo přihlášen.

Po automatickém odhlášení uživatele serverem v případě delší nečinnosti je při pokusu o znovunačtení stránky nebo přesunu na jinou stránku uživatel upozorněn že není přihlášen a je mu nabídnuto opětovné přihlášení.

4.4.4 Administrace

Administrátor má podle modelu systému status kořenu stromu ve stromové hierarchii firmy a může spravovat klienty a smlouvy stejně jako by byl kterýmkoliv zaměstnancem firmy. Oproti běžnému zaměstnanci ale může upravovat více vlastností (atributů) u objektů těchto tříd.

Hlavní a nejdůležitější funkčností administrátorské role je správa uživatelů, správa stromové hierarchie firmy a číselníků Body pozice, Typ smlouvy, Položky typu smlouvy a Finanční subjekty.

4.4.4.1 Menu administrace

Přihlášením administrátora do systému je vygenerováno menu s možnostmi správy systému Obr. 25. Podle správy procesů firmy je tématicky rozděleno do několika kategorií – Zaměstnanci, Klienti, Smlouvy, Správa číselníků a Export/Import dat databáze. Každá z těchto kategorií obsahuje základní

možnosti správy kategorie a odkazuje na konkrétní CSP stránky. Podle modelu systému z kapitoly 3 odpovídají jednotlivé možnosti správy případům užití systému a jejich funkčnost je řízena podle sekvenčních diagramů modelu.

Obr. 25: Ukázka menu po přihlášení Administrátora

4.4.4.2 CSP stránky pro administraci

Implementace jednotlivých případů užití administrátora je řízena modelem systému a jeho iteracemi. Během první iterace implementace systému byly vytvořeny již výše popsané CSP stránky a funkce pro správu uživatelů, přihlašování a odhlašování ze systému a průběžně s nimi také vznikly CSP stránky pro vznik, editaci a mazání zaměstnanců firmy ve stromové hierarchii.

Přidání nového zaměstnance a editace je řešena pomocí formulářů vygenerovaných v HTML a do nich zadané informace jsou ukládány jako vlastnosti objektů třídy uživatel. Odmazání zaměstnance ze stromu v rámci první iterace řešilo přeuskupení stromové struktury přirazením podřízených zaměstnanců nadřízenému odstraněného zaměstnance.

1. Iteraci tvoří tyto CSP stránky:

inchead.CSP (hlavička každé CSP stránky zobrazující informace o přihlášeném uživateli)
index.CSP (zobrazuje přihlašovací formulář, menu zaměstnance nebo administrátora)
editzam.CSP (vygeneruje formulář předvyplněný daty zaměstnance pro editaci)
login.CSP (nastaví session podle přihlášeného uživatele)
logout.CSP (zruší session pro odhlášení uživatele)
pridejzam.CSP (vygeneruje formulář pro přidání nového zaměstnance a uloží vyplněná data)
smazanizamestance.CSP (je voláno pro konkrétního zaměstnance ze seznamu zaměstnanců, předává se jedinečné identifikační číslo, které Caché objektu zaměstnance přidělil. Pokud toto číslo není předáno, je uživatel upozorněn že není koho mazat)

stromzam.CSP (pro administrátora vypíše celý seznam zaměstnanců a z každého zaměstnance vytvoří hypertextový odkaz pro prohlédnutí detailů)

zamestnanci.CSP (vypíše všechny objekty třídy uživatel ve formě tabulky s možností editace a smazání každého z nich)

zaminfo.CSP (vypíše podrobné informace o konkrétním zvoleném zaměstnanci)

zmenheslo.CSP (pro každého uživatele včetně administrátora umožní změnit přihlašovací heslo)

Druhá iterace rozšířila funkčnost systému o přidání, editaci a mazání klientů jednotlivých uživatelů a rozšířila funkčnost některých CSP stránek vytvořených v první iteraci. Samostatnou částí bylo vytvoření XML importu a exportu dat.

Caché nabízí plnou podporu XML exportu a importu tříd již při samotném definování tříd. Pokud je konkrétní třída vyděděna ze systémové třídy %XML.Adaptor, je možné exportovat a importovat její objekty formou XML výstupu jen voláním metod této třídy. Každou z takto vyděděných tříd je potřeba pojmenovat pomocí parametru XMLNAME= "nazev_xml_vystupu";. Pro samotné spuštění exportu do formátu XML jsem vytvořil samostatnou třídu XMLExp. Její podrobnější popis a funkčnost je v následující části 4.4.4.3 Funkce.

2. Iteraci rozšířily tyto CSP stránky:

index.CSP (rozšířeny položky menu o správu klientů a XML)

editklient.CSP (vygeneruje formulář předvyplněný daty klienta pro editaci)

klientinfo.CSP (vypíše podrobné informace o konkrétního vybraného klienta)

pridejklienta.CSP(vygeneruje formulář pro výběr zaměstnance a zadání informací nového klienta)

smazklienta.CSP (smaže vybraného klienta podle jeho OID)

vsichniklienti.CSP(vypíše všechny objekty třídy klient ve formě tabulky s možností editace a smazání každého z nich)

XMLimpinc.CSP a XMLimp.CSP (stránky pro import dat do databáze, první jmenovaná vygeneruje formulář pro vložená XML kódu a druhá slouží pro jeho zpracování a průběžné informování o importu)

zaminfo.CSP (rozšířen o výpis klientů zaměstnanců)

zamklient.CSP (stránka pro evidenční výpis zaměstnanců se seznamem klientů)

smazanizamestnance.CSP(pro smazání konkrétního zaměstnance rozšířena funkčnost, kdy všichni jeho klienti jsou předání jeho nadřízenému v hierarchii)

XMLExp.cls (jedná se o třídu pro XML export která je volána jako webová stránka)

Třetí iterace doplňuje kompletní funkčnost systému dle navrženého modelu. Finální stav třetí iterace ještě není plně dokončen. Je funkční správa číselníků a základní propojení balíčků zaměstnanci, klienti a smlouvy. Průběžně dokončuji funkčnost této třetí iterace.

Správa číselníků je řešena formou tabulek, kdy má administrátor možnost evidovat změny ve finančním sektoru. Během dokončení třetí iterace plánuji umožnit samostatný import a export číselníkových dat nezávisle na zbylých datech databáze aby nebylo nutné číselníky definovat pracně pomocí formulářů.

3. Iterace stávající seznam CSP stránek:

index.CSP (rozšířeny položky menu o správu smluv a číselníků)

editsmlouva.CSP (vygeneruje formulář předvyplněný základními daty smlouvy pro editaci)

klientsmlouva.CSP (stránka pro evidenční výpis klientů a jejich uzavřených smluv)

pridejsmlouvuinc.CSP a pridejsmlouvu.CSP (tyto dvě stránky splňují případ užití přidání smlouvy administrátorem, uvedený na obrázku 14. Administrátor vybere zaměstnance, který smlouvu zprostředkovává a pokud zaměstnanec nemá klienty, umožní mu je systém vložit. V opačném případě nabídne seznam klientů a vygeneruje formulář přidání smlouvy)

smazklienta.CSP (rozšířena funkčnost smazání všech smluv klienta před jeho samotným smazáním)

vsechnysmlouvy.CSP (vypíše evidenční seznam smluv s možností jejich editace, případně smazání)

zamklientsmlouva.CSP (kompletní evidence celého systému, obdoba XML exportu dat formou HTML)

BodyPozice.CSP (číselník bodů a pozic zaměstnanců s možností smazání záznamů)

editBodypozice.CSP (editace existujících záznamů číselníku)

přidejBodypozice.CSP (přidání záznamu do číselníku)

4.4.4.3 Funkce

Funkčnost administrátorské části je řízena modelem systému. Jednotlivé sekvenční diagramy, popřípadě sekvenční diagramy rozšířené o formuláře jsou v třetí kapitole a součástí exportu modelu systému v příloze. Samostatnou částí systému je již výše zmiňovaný export a import dat v XML.

XML export dat spouští třída XMLexp.cls náležící %CSP.Page. Díky tomu je volána z menu administrátora jako CSP stránka. Tato třída obsahuje inicializační metodu XML exportu pojmenovanou OnPage(), jejímž úkolem je pro každý objekt třídy uživatel spustit XMLexport(). Caché se následně postará o kompletní export provázaných tříd s třídou uživatel přes asociační vazby, případně dědičnosti. Do XML exportu jsou ale zahrnuty jen ty třídy, které byly pro XML export

definovány a pojmenovány. Výsledkem tohoto exportu je XML stránka zobrazená v okně prohlížeče s vyexportovanými daty databáze, které je možno uložit do souboru. V případě přání administrátora systému bude doimplementována funkčnost automatického exportu do XML souboru na disk. Export demonstračních dat je přiložen v souboru xml_exp_db.xml na přiloženém médiu. Z důvodu ochrany osobních dat zaměstnanců, klientů a jejich smluv se jedná jen o smyšlená demonstrační data.

4.4.5 Zaměstnanec- uživatelská část

Zaměstnanec firmy se automaticky stává aktivním uživatele systému s rovnocennými právy správy svých klientů a smluv. Na základě přijetí do firmy je mu vygenerováno jedinečné číslo, které slouží zároveň jako přihlašovací login do systému. Implicitně mu heslo vytvoří administrátor a zaměstnanec má právo si jej změnit pomocí položky v menu systému.

4.4.5.1 Uživatelské menu

Z pohledu funkčnosti systému se liší zaměstnanec od administrátora přístupem k jednotlivým CSP stránkám a vygenerováním menu. Zaměstnanec má menu rozděleno stejně jako administrátor do několika tématických kategorií:

Osobní

Tato kategorie umožňuje správu jeho vlastního účtu a osobních informací. Zaměstnanec si může vypsat osobní data včetně svých klientů, smluv a přímých podřízených. Editovat svoje osobní informace a měnit přihlašovací heslo. Každý zaměstnanec má právo znát informace týkající se svého podstromu, jako jsou všichni zaměstnanci v jeho podstromu, jejich klienti a smlouvy. Proto mu systém nabízí vypsat rekurzivně všechny tyto informace.

Podřízení

Nejpodstatnější část podstromu jsou jeho přímí podřízení, na základě jejichž pozice a získaných bodů se počítají finanční částky, které zaměstnanec získá. Z tohoto důvodu existuje samostatná kategorie menu, ve které může zaměstnanec sledovat veškeré podstatné informace o svých přímých podřízených včetně jejich vlastních podstromů. Systém mu umožňuje zobrazit si tyto informace formou tabulek nebo stromového výpisu.

Klienti

Správa klientů a smluv je stěžejním účelem celého systému. Kategorie klienti nabízí všechny potřebné funkce a výpisy pro správu klientů, přidání, odebrání, editaci a výpisy klientů i se smlouvami. V rámci dokončení třetí iterace bude implementováno seřazení klientů podle různých

kritérií (vlastností) a v případě zájmu zaměstnanců je možné řešit i vyhledávací filtry, které jsou popsány v kapitole 6. Možnosti rozšíření systému.

Smlouvy

Kategorie správy a evidence smluv je propojena s kategorií klientů. Stávající funkčnost umožňuje ukládat, evidovat a mazat základní data ke smlouvě a evidovat smlouvy. V rámci dokončení třetí iterace se budou jednotlivé smlouvy lišit podle typu smluv a položek smlouvy, které nadefinovaly finanční subjekty. Opět bude implementováno řazení smluv podle různých kritérií a řešena jejich platnost.

Číselníky

Výpis aktuálního stavu číselníku, které zaměstnanec využívá ve své práci. Jedná se o informace z číselníku Body Pozice, Typy smlouvy a položky typů smluv, Finanční subjekty. Zaměstnanec nemá právo jejich editace.

4.4.5.2 Skripty pro zaměstnance

CSP stránky zobrazující informace popsané v jednotlivých kategoriích menu se liší od administrátorských přístupovými právy nebo podmínkami jejich použití. Stejně jako CSP stránky pro administraci systému vznikali v jednotlivých iteracích z důvodu testování funkčnosti systému. Z tohoto důvodu jsou také popsány z pohledu iterací.

1. Iteraci tvoří tyto CSP stránky:

index.CSP	(rozšířen o zaměstnanecké menu a přihlášení zaměstnanců do systému)
login.CSP a logout.CSP	(rozšířeno rozlišení administrátora a zaměstnance)
podřízení.CSP	(vypíše formou tabulky jen tu část podstromu která patří přihlášenému zaměstnanci s možností výpisu informací o jednotlivých zaměstnancích stromu)
mujedit.CSP	(vygeneruje formulář vyplněný osobními daty zaměstnance pro editaci)
stromzam.CSP	(vypíše stromovou strukturou podstrom přihlášeného zaměstnance)
zaminfo.CSP	(vypíše podrobné informace o přihlášeném zaměstnanci)
zmenheslo.CSP	(umožní zaměstnanci změnit přihlašovací heslo do systému)

Druhá iterace přidává zaměstnanci možnost správy svých klientů a rozšiřuje možnosti výpisu informací o klientech svých přímých podřízených i celého svého podstromu.

2. Iteraci rozšířily tyto CSP stránky:

klientinfo.CSP	(informace o jednotlivých klientech přihlášeného zaměstnance)
mojiklienti.CSP	(tabulkový seznam klientů přihlášeného zaměstnance)

editklient.CSP (editace vybraného klienta přihlášeného zaměstnance)

zampridejklienta.CSP (vygeneruje formulář pro vložení nového klienta přihlášenému zaměstnanci)

smazklienta.CSP (smazání aktuálního vybraného klienta)

Třetí iterace opět rozšířila funkčnost systému o evidenci smluv, stejné jako v administrátorské části. Zaměstnanec může přidávat jednotlivých klientům smlouvy a vyplňovat základní informace k objektům smluv. Během dokončení třetí iterace bude zprovozněna funkčnost pro různé typy smluv a řazení podle různých parametrů. Každý zaměstnanec má právo zobrazit aktuální stav číselníků systému.

3. Iterace se stávajícími CSP stránkami:

editsmlouva.CSP (vygeneruje formulář předvyplněný základními daty smlouvy pro editaci)

mojemlouvy.CSP (tabulkový výpis všech uzavřených smluv)

smazklienta.CSP (rozšířena funkčnost o smazání smluv klienta)

smazsmlouvu.CSP (smazání smlouvy přihlášeného zaměstnance)

zamklientsmlouva.CSP (evidenční výpis klientů a smluv přihlášeného zaměstnance)

zampridejsmlouvu.CSP (vygenerování formuláře nové smlouvy pro klienta zaměstnance. Pokud ještě neexistuje žádný klient zaměstnance, je systémem nabídnuta možnost jeho vložení)

4.4.6 CSS a grafická úprava CSP stránek

Caché ServerPages mají plnou podporu HTML i XML. Mohou být ale integrovány i libovolné další binární soubory. Součástí Caché je i vlastní modul do profesionálního nástroje Dreamweawer firmy Macromedia, který poté umožňuje vkládat a spravovat speciální značky CSP a jiné jazykové prvky CSP (Kirsten, 2005).

Pro grafickou úpravu stránek implementovaného systému jsem využil kaskádových stylů CSS. Samostatný soubor s definicí CSS stylů je pojmenován styl.CSS a je v projektu začleněn mezi ostatní stránky CSP. Caché Studio umožňuje tyto soubory přiřadit přímo do CSP stránek nebo jako samostatnou přikompilovatelnou část projektu ve stromovém rozdělení projektu v kategorii Other. CSS styly jsou definovány stejně jako při použití statických HTML stránek.

Díky schopnosti CSP stránek zpracovávat i binární soubory jsem využil vkládání obrázků formátu jpg ke grafické úpravě designu stránek, přihlašování a odhlašování ze systému.

5 Testování systému

Model systému byl rozvržen do několika iterací a i diagram tříd je rozdělen do čtyř balíčků, které tvoří vlastní tématické okruhy. Testování funkčnosti systému bylo prováděno v rámci všech iterací nad jednotlivými třídami balíčků a následně od druhé iterace bylo testována i funkčnost vazeb mezi jednotlivými třídami balíčků.

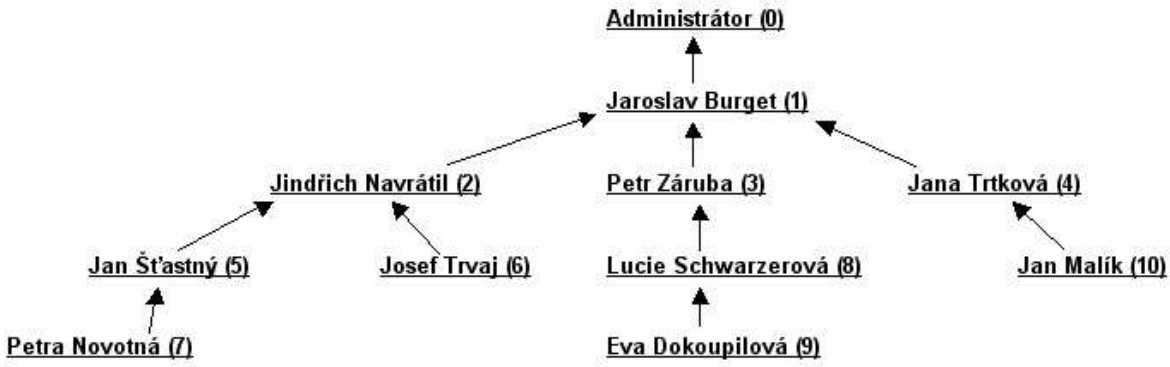
Systém byl testován pod operačním systémem Windows XP professional SP2. Verze Caché byla 5.2 a webový server i webová brána Caché byla instalována na stejném počítači o konfiguraci:

Procesor: AMD AthlonXP 2000+, taktován na 1,67 Ghz

Paměť: 512 MB RAM DDR na 400Mhz

Webové prohlížeče: Internet Explorer 6.0.2900
Mozilla Firefox 2.0.0.11
Opera 9.21

V rámci každé iterace jsem navrhnul několik přejímacích testů validujících případy použití nad demonstračními daty, které jsou uvedeny v exportu databáze v příloze. Tyto data demonstrují strom zaměstnanců o deseti objektech třídy zaměstnanec jak ukazuje Obr. 26. Postupně jsem v každé iteraci doplňoval nově vzniklé třídy o testovací data, kdy někteří zaměstnanci měli klienty a jiní ne. Stejným způsobem byly řešeny i smlouvy. Účelem bylo postihnout všechny možnosti, které v případech užití systému mohly nastat a vyzkoušet jejich správnou funkčnost.



Obr. 26: Demonstrační strom zaměstnanců pro testování

Testování 1. iterace

Pro první iteraci jsem pomocí případu užití – „přidání zaměstnance“ vytvořil strom zaměstnanců na Obr 26. Caché automaticky neupozorní na správnost vyplnění dat ve formuláři při ukládání nového zaměstnance (nebo editaci stávajícího), proto je nutné tuto kontrolu správnosti dat implementovat aplikačně.

Pro správnou funkčnost mazání zaměstnance a přeuskupení stromu strom nabízí všechny varianty mazání zaměstnance:

- Smazání zaměstnance který nemá podřízené
- Smazání zaměstnance který má právě jednoho podřízeného
- Smazání zaměstnance s více podřízenými
- Smazání zaměstnance s podstromem podřízených (rozšířená varianta předchozích)

V konečném stavu první iterace systém prošel všemi testy a umožňuje smazat kteréhokoliv zaměstnance kdekoliv ve stromové hierarchii se správným přeuskupením stromu, kdy jsou podřízení zaměstnanci mazaného ve stromové hierarchii přeřazeni nadřízenému mazaného zaměstnance.

Testování 2. iterace

Rozšíření testů druhé iterace o přidání, editaci a mazání klientů zaměstnanců bylo již rozlišeno podle rolí uživatelů. Ke stávajícímu testovacímu stromu z první iterace jsem některým zaměstnancům přidal klienty tak abych opět postihl všechny varianty následného mazání. V roli administrátora jsem otestoval přiřazení klienta náhodným uživatelům a v roli zaměstnance právě přihlášenému zaměstnanci. Stejným způsobem bylo otestováno mazání klientů administrátorem i zaměstnancem.

Složitější variantou je mazání zaměstnanců, kdy musí systém automaticky přeřadit klienty mazaného zaměstnance jeho nadřízenému. Otestovány byly úspěšně všechny varianty:

- Zaměstnanec neměl žádné klienty
- Zaměstnanec měl jednoho klienta
- Zaměstnanec měl více klientů

Systém umožňuje správně mazat zaměstnance s přeřazením klientů a při mazání je uživatel upozorněn výpisem webové stránky o které klienty se jednalo.

Testování 3. iterace

V rámci třetí iterace je systém zatím rozšířen o přidání objektů smluv do třídy Smlouva a XML import a export dat již všech tříd tří balíčků. Z pohledu testování systému se jednalo o rozšíření funkčnosti testů druhé iterace. Pomocí formulářů administrátora i zaměstnance jsem přidal smlouvy klientům jednotlivých zaměstnanců pro postihnutí všech variant mazání. Následně jsem vyzkoušel následující mazání klientů a zaměstnanců:

- Smazání klienta bez smlouvy
- Smazání klienta s jednou smlouvou
- Smazání klienta s více smlouvami

Ve všech těchto případech došlo i ke smazání smluv klienta, popřípadě upozornění o variantě, že mazaný klient neměl žádnou smlouvu.

- Smazání zaměstnance s klientem bez smlouvy
- Smazání zaměstnance s klientem se smlouvami

Smlouvy mazaných zaměstnanců zůstaly zachovány a spolu s klienty byly předány nadřízenému mazaného zaměstnance.

Testování XML exportu a importu dat proběhlo také v pořádku. Vyexportovaná data byly v pořádku importovány do následně prázdné databáze systému.

6 Možnosti rozšíření systému

Stávající funkčnost systému navržená v modelu v kapitole 3 odpovídá neformální specifikaci získané během konzultací se zaměstnanci firmy. Po dokončení implementace třetí iterace a otestování celkové funkčnosti nad reálnými daty budou zváženy možná rozšíření a vylepšení systému. Tato kapitola nastiňuje některé z možných návrhů rozšíření.

Automatická kontrola platnosti smluv

Většina typů smluv je uzavírána na časově omezenou dobu (stavební spoření, penzijní připojištění atd.). Po tuto dobu jsou smlouvy platné a aktivní. Každému objektu třídy Smlouva by byl přidán nepovinný atribut data ukončení platnosti a přidána funkčnost systému. Právě přihlášenému uživateli systém nabídnul automatické zneplatnění těch smluv, jejichž datum ukončení platnosti by se rovnalo nebo bylo větší systémovému datu.

Upozornění klientů emailem o stavu smlouvy

Systém by umožnil zaměstnanci odeslat informace o stavu smlouvy na email klienta. Po výběru této možnosti v menu u konkrétní smlouvy by systém vygeneroval email se všemi patřičnými informacemi o smlouvě (číslo, typ, datum uzavření, částky a podmínky atd) s možností přidání vlastního komentáře zaměstnancem. Po vyplnění komentáře a stisku odeslání emailu by byla zpráva poslána na email uvedený u konkrétního klienta, který si smlouvu uzavřel.

XML export do souboru

Stávající funkčnost exportu zahrnuje vygenerování XML kódu do webové stánky. Rozšíření funkčnosti by zahrnovalo možnost výběru administrátorem, zda chce XML soubor vygenerovat do webové stránky nebo rovnou do XML souboru na disk. Stejným způsobem by bylo řešeno importování dat v XML. První možností by bylo nakopírování patřičného kódu do webového formuláře nebo načtení obsahu vybraného souboru.

Evidence historie smluv

Dokončením třetí iterace budou případem užití „převod bodů stromem“ řešeny přepočty bodů a orientačních částek obdržených za tyto body. Přáním zákazníků bylo udržovat tyto informace jen rámcově, jelikož částky za tyto body jsou firmou upravovány i dle prémií a dalších faktorů.

Pokud by přáním zákazníků bylo zpřesnit tyto informace, umožňoval by systém editovat částky za obdržené body, popřípadě další výpočty nad těmito částkami (prémie, zvýhodněné smlouvy atd.)

7 Závěr

Cílem této diplomové práce bylo vytvoření modelu informačního systému pro vyhodnocování aktivit pracovníků ve stromové hierarchii a implementace navrženého modelu v prostředí post-relační databáze Caché.

Podle zadání diplomové práce jsem v druhé kapitole nastínil problematiku řízení zaměstnaneckých firem ve finančním sektoru fungujících nad stromovou hierarchií tvořenou ze svých zaměstnanců. Popsal jsem rozdíl mezi nelegálními subjekty zneužívajícími tuto stromovou hierarchii a seriózními finančními multilevel marketingovými společnostmi.

Model systému

Třetí kapitola nazvaná Model informačního systému se zaměřuje na kompletní funkční řešení systému. Jelikož je implementace celého systému v Caché řešena nad objektovou databází, je řízena tímto modelem. Z tohoto důvodu je kapitola modelu systému nejrozsáhlejší částí celé diplomové práce a nejdůležitější a nejzajímavější diagramy modelu jsou prezentovány a popsány přímo v rámci kapitoly. Popisuji základní představení jazyka UML 2.0 (Unified Modeling Language) a jeho elementy. Následně se již zabývám popisem iterativního přístupu k modelovanému systému: model systému a jeho následná implementace je rozdělena do tří vzájemně se prostupujících přírůstků. Každý z těchto přírůstků řeší systém z pohledu jednoho tématického celku – zaměstnanců, klientů a mluv.

Na základě konzultací se zaměstnanci firmy fungující nad stromovou hierarchií jsem formuloval neformální specifikaci systému uvedenou v této kapitole a požadavky této specifikace převedl na model využívající převážně UML 2.0. Systém je navržen a implementován jako webová aplikace nad objektovým úložištěm dat systému Caché. Každý ze zaměstnanců firmy bude mít soukromý heslem ověřený přístup do tohoto systému a bude moci evidovat svoje klienty a smlouvy, které jim zprostředkoval.

Model systému je uvozen diagramem hierarchie procesů firmy a na něm navazujícími diagramy procesních vláken, které jsem vytvořil z důvodu pochopení problematiky procesů ve firmě a následného správného určení všech případů užití, které bude systém realizovat. Tyto jsou rozlišeny z pohledu dvou typů rolí uživatelů – administrátora a zaměstnance. Nejdůležitější z těchto případů užití jsou podrobně popsány a následně modelovány pomocí dalších diagramů. Nejpodstatnější diagram – diagram tříd, řešící fyzické uložení dat v objektové databázi, vazby a metody jednotlivých tříd a je kompletně popsán a zobrazen v diplomové práci. Na základě něj jsem vytvořil i logický model uložení dat.

Dynamické chování systému řeší zbývající část kapitoly počínaje modelem objektové spolupráce, v rámci něhož jsou vytvořeny všechny nejpodstatnější sekvenční diagramy realizující

jednotlivé případy užití. Sekvenční diagramy včetně rozšíření o uživatelské formuláře byly řídícím prvkem pro implementaci webové aplikace v následující kapitole. V poslední části kapitoly modelu jsem uvedl stavové diagramy dvou nejdůležitějších tříd systému a ukázku diagramu aktivit přidání nové smlouvy do systému.

Implementace v Caché

Post-relační databázový systém Caché je jedním z prvních plně objektových systémů vůbec. Vyjma objektového přístupu ke své objektové databázi nabízí i SQL přístup, který umožňuje pohlížet na třídy formou tabulek. Z těchto důvodů jsem si jej vybral pro implementaci modelovaného systému. Ve čtvrté kapitole diplomové práce popisuji princip funkčnosti Caché a jeho možnosti. Popsán je také objektový model Caché včetně vlastností objektů a typů systémových a uživatelských tříd.
Implementací podle jednotlivých iterací se zabývají následující části kapitoly 4.3 a 4.4. Rozlišil jsem vytvoření databázové části (třídy, vazby, metody tříd) a klientské webové části pomocí Caché Server Pages.

Serverová část systému je popsána definicí balíčků, tříd balíčků použitím jazyka Class Definition Language (CDL) a implementací metod jednotlivých tříd použitím jazyka Caché ObjectScript.

Klientská část je realizována pomocí Caché Server Pages (CSP), které generují HTML kód webových stránek prohlížeče přes webovou bránu Caché. Popsán je také Objektový model CSP. V rámci podkapitoly 4.4. je popsána autentizace systému včetně rozdělení rolí uživatelů a přihlášení a odhlášení ze systému. Vznik CSP stránek systému je rozděleno na administrátorskou a zaměstnaneckou část a pro obě části je popsány vznik stránek ve všech třech iteracích.

Implementovaný systém má plně dokončeny a otestovány první dvě iterace. Třetí iterace která řeší správu smluv klientů ještě není dokončena a fungují jen některé z případů užití. Kompletní dokončení systému bude realizováno v nejkratší možné době z důvodů testování v provozu nad reálnými daty zaměstnanců, kteří si systém vyžádali. Jedná se o doimplementování již namodelované funkčnosti.

Testování a možnosti rozšíření systému

První dvě iterace systému byly testovány nad demonstračním stromem zaměstnanců a jejich klientů a jsou plně funkční. Stávající funkčnost třetí iterace již byla otestována také a popis těchto testů je součástí kapitoly 5.

Šestá kapitola navrhuje některá z možných rozšíření systému po dokončení třetí iterace. Tyto rozšíření budou konzultována se zaměstnanci až na základě testování kompletně dokončeného systému a jeho osvědčení se v praxi.

Přínosem této diplomové práce je vytvoření webového informačního systému, který bude použit v praxi skupinou zaměstnanců, kteří mezi sebou tvoří jednu větev stromové hierarchie. Systém je modelován a implementován objektově nad objektovou databází, která je označována za jednu z nejrychlejších objektových databází. Výhodu výběru objektové databáze pro tento systém vidím ve využití modelu systému pro efektivní vytvoření jen na základě modelu, kdy balíčky, třídy a vztahy navržené diagramem tříd mohou být přímo uloženy v databázi a komunikace mezi webovými stránkami objekty v databázi je řešena formou metod tříd. Další výhodou je větší stabilita systému, jelikož objekty tříd mezi sebou komunikují předáváním zpráv což zajišťuje, že změny struktury jsou zapouzdřeny uvnitř objektů a nerozšiřují svůj dopad do dalších částí systému.

Literatura

[1] Hana Kanisová, Miroslav Müller. UML srozumitelně. Brno,Computer Press 2006.

[2] Ivana Rábová. Diagramy UML a Rational Rose, Pef MZLU, 2000.

[3] Jim Arlow,Ila Neustadt. UML a unifikovaný proces vývoje aplikací. Praha, Computer Press 2003.

[4] Wolfgang Kirsten a spol. Caché databáze postrelačního typu a tvorba aplikací. Brno,CP Books 2005.

[5] Peter Clothier. Multi-level Marketing. Praha, Pelikán 1995.

[6] Ilja Kraval. Nejčastější chyby při modelování 1, URL: <http://www.dbsvet.cz/view.php?cisloclanku=2004080401> [platné 2006-12-26].

[7] René Stein. Návrh aplikací v jazyce UML, URL: <http://interval.cz/clanky/navrh-aplikaci-v-jazyce-uml-unified-modeling-language/> [platné 2007-2-6].

[8] Pavel Tišnovský. Nástroje pro tvorbu UML diagramů, URL: <http://www.root.cz/clanky/nastroje-pro-tvorbu-uml-diagramu/> [platné 2007-2-6].

[9] Jan Spilka. Multilevel – cesta k bohatství či do pekel?, URL: <http://www.mesec.cz/clanky/multilevel-cesta-k-bohatstvi-ci-do-pekel/> [platné 2007-10-20].

[10] Síťový marketing versus "pyramidové“ struktury, URL: <http://aktualne.centrum.cz/finance/investice/clanek.phtml?id=137950> [platné 2007-10-20].

[11] InterSystems Caché, URL: <http://www.intersystems.cz/cache/> [platné 2007-10-20].

[12] InterSystems Technologie Caché, URL: <http://www.cache.cz/education/cache_studio/> [platné 2007-10-20].

[13] Metodika Select Perspective, URL: < http://objekty.vse.cz/Objekty/MetodikyANotace-Select> [platné 2007-12-30].

Seznam příloh

Příloha CD obsahující:

1. XML export modelu systému z nástroje CASE Select Component Architect
2. XML export systému z Caché, třídy, rutiny a CSP stránky
3. XML export demonstračních dat systému
4. Dokumentaci diplomové práce ve formátu PDF