PRAKTICKÉ ZKUŠENOSTI S BUDOVÁNÍM INTERNETOVÉ APLIKACE V PROSTŘEDÍ IBM VISUALAGE FOR JAVA A IBM WEBSPHERE APPLICATION SERVER

Pavel Janík

IMC Zlín a.s., Zarámí 4422, Zlín, e-mail janik@imc.cz

Abstrakt

Příspěvek se snaží poskytnout zájemcům o OO technologii na bázi Java pohled vývojáře "zevnitř". Tento pohled se opírá o praktické mnohaleté využívání OO technologie v oblasti analýzy a návrhu a o dvouletý praktický vývoj v jazyku Java. Dotkneme se použitých vývojových nástrojů a metodiky, otázek volby architektury. Součástí příspěvku je i příklad použití popisované metodiky a nástrojů při realizaci aplikace pro velkou logistickou firmu.

1. Proč modelování?

Ve vývoji programovacích technik dochází zákonitě vždy nejprve k rozvoji programovacího jazyka a teprve s určitým odstupem se pozornost přesouvá k přípravným fázím – návrhu resp. k analýze. To platí i pro OO svět: V 60. letech se o OO programování hovoří v souvislosti s jazykem Simula, v 70. letech přichází od Xeroxu SMALLTALK, v 80. letech je to C++.
V oblasti modelování probíhá přerod funkčního modelování na datové modelování (vzpomeňme např. Jacksonovo strukturované programování s centrální myšlenkou "všechno se dá vyjádřit daty"). Teprve v 80. letech se formují nové metodiky OO návrhu a OO analýzy.

Objektové chápání přitom není nic nového. Jde o dlouhodobě známé principy řešení složitosti používané již ve starém Řecku. Nové bylo spojení těchto myšlenek do ucelené metodiky, o které se poprvé pokusili Peter Coad a Edward Yourdon a potom jejich následovníci.

Základním cílem použití některé z modelovacích technik je řešení složitosti. Čím složitější je řešený systém, tím více vystupuje do popředí potřeba výkonné modelovací metodiky a nástrojů.

2. V čem se odlišuje OO svět?

Hlavní rozdíl je v granularitě. Uvažujme, že běžná Enterprise aplikace v podniku střední velikosti obsahuje cca 180 základních programů, které spravují 450 databázových tabulek. Zdrojový kód v jazyku tzv. 4. generace představuje cca 240 tis. řádků kódu. Je zřejmé, že takovou aplikaci je bez určitých modelovacích a dokumentačních nástrojů obtížné sestavit a ještě obtížnější udržovat.

V OO verzi stejného projektu jsme po analýze došli k následujícím číslům: 400 základních (business) tříd a 800 ostatních tříd, cca 10 tis. metod, cca 3 mil. řádků kódu. Takovou aplikaci nelze vytvořit ani udržovat bez výkonné modelovací a návrhové metodiky, bez využití programovacích vzorů, bez využití integrovaných vývojových nástrojů.

3. UML (Unified Modeling Language)

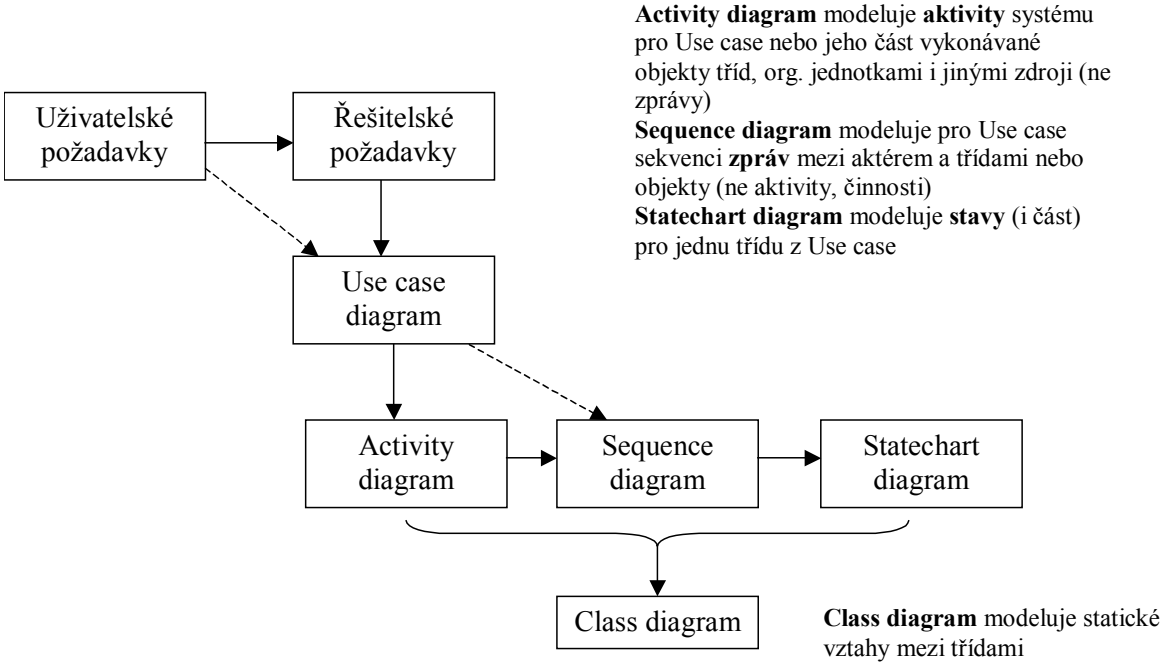
První definice byla zveřejněna v roce 1996 jako otevřený standard pro OO modelování. Tento standard přijala řada předních výrobců software. Výsadní postavení zde má Rational Software Corporation, kde působí duchovní otcové UML pánové Booch, Rumbaugh a Jacobson.

Modelovací standard je však jen jakási unifikovaná tužka a papír (a šablonka na značky). Ale referenční manuál UML neobsahuje nic o cestě, jak s jeho využitím modelovat složitý systém. To je obsahem metodiky.

Takových metodik pro OOA/D/P existuje řada, nejznámější je Rational Unified Process, ale prakticky každá firma zabývající se vývojem CASE nástrojů nabízí také „ten pravý“ proces a ten je samozřejmě optimálně podporován právě v tom jejím CASE. Proto budete při použití CASE nástroje firmy Rational vedeni jinam než v případě CASE nástroje firmy Sterling.

Všechny metodiky však řeší stejnou základní otázku: Jak se dostat úspěšně od požadavku na to, co by měl nový systém dělat, k reálným třídám? Tento postup se navíc u větších systémů člení do vývojových fází, které jsou kroky od hrubého náhledu až po detailní implementaci. V procesu firmy Sterling jsou to například fáze Analysis, System Design, Object Design a Implementation, jinde můžete najít členění na Business model a Design model, atd.

V UML máme pro modelování v rámci celého procesu k dispozici 9 standardních diagramů. V principu lze v kterékoliv fázi použít kterýkoliv z těchto diagramů, přesto jsou některé diagramy předurčeny pro ranější fáze a některé spíše pro návrh a implementaci. Z hlediska kódování v jazyku Java je stěžejní diagram tříd (Class Diagram), který je zdrojem pro automatické generování kódu v CASE nástroji. Vedle diagramu tříd jsou ve fázi analýzy a návrhu důležité UseCase, Sequence, Activity a Statechart diagramy.



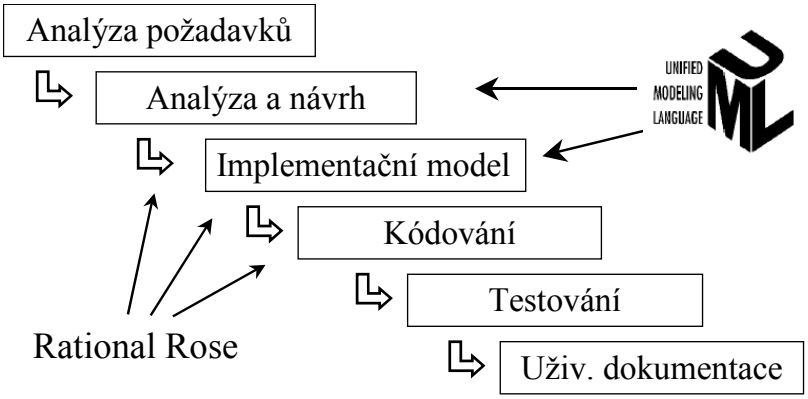
Obr. 1: Cesta od uživatele k Class diagramu

Obrázek 1 ukazuje jeden z možných přístupů k OO analýze a návrhu. Jádrem tohoto přístupu je pozice UseCase diagramu jako základního nástroje pro stanovení struktury aplikace. Pokud dokážeme aplikaci rozdělit do samostatných, dostatečně malých a relativně nezávislých částí, dokážeme pak v rámci této části poměrně snadno modelovat vnitřní strukturu a chování včetně vlivu na několik málo zúčastněných tříd. Takovým způsobem používáme UseCase diagram při modelování velkých Enterprise systémů již několik let.

4. Modelovací a vývojové nástroje

Není možné modelovat systém na bázi UML bez využití CASE nástroje. CASE nástroj má několik základních funkcí: inteligentní editor diagramů, zajištění provázanosti dílčích diagramů s celkovým modelem (změna prvku v jednom diagramu se promítne ve všech ostatních diagramech), kontrola modelu, generování kódu., …

Co je k dispozici v oblasti CASE? Já jsem se setkal se dvěma "velkými" CASE nástroji. Nejprve to byl Sterling Cool:Jet, původem od firmy Westmount a později Cayenne Software. Nyní pracuji s CASE Rational Rose. Nenechte se zlákat nabídkou malých a všeobecně dostupných nástrojů. Na Internetu je jich hodně. Ale ty "pravé" nástroje nabízejí opravdu mnohem více (a také za podstatně vyšší cenu). Nenechte se zlákat tvrzením, že potřebujete jen Class diagram a proto je pro vás ideální právě tento malý a levný CASE. OO analýza a návrh, to není překreslování datových tabulek do diagramu tříd. Je to především změna myšlení od souborového nebo relačního k objektovému. A tato změna není možná, pokud vynecháte z úvah některé fáze nebo pohledy modelu a vytvoříte přímo "objektový" statický model ve formě diagramu tříd.



Obr. 2: Pozice UML v procesu používaném v IMC Zlín

Na obr. 2 je znázorněn "vodopád" vývojového procesu. CASE nástroj firmy Rational Software podporuje všechny tři "výkonné" fáze procesu. Samozřejmě všichni dodavatelé CASE nástrojů nabízejí další nástroje své nebo třetích firem pro ostatní fáze procesu.

5. Java

Tady je obdobná situace jako u analytických nástrojů. Máte k dispozici mocný jazyk, který v bezpečnosti, přenositelnosti a výkonnosti převyšuje C++. Můžete použít bezplatně šířené knihovny, nepotřebujete žádné runtime licence. Zdarma dostanete vývojové prostředí JDK x.x.. Nemáte však jednoznačný návod, jak s jejich pomocí naprogramovat složitou aplikaci.

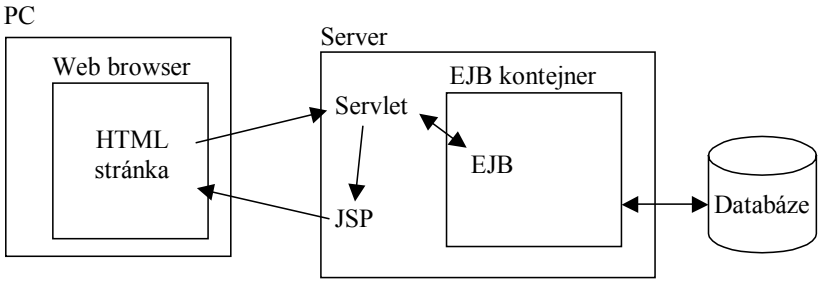
Samozřejmě je možné začít v nějakém textovém editoru psát třídy a pomocí JDK se pokusit s nimi pracovat. Taková práce je však neefektivní a pro rozsáhlejší systém nemyslitelná. Proto se spolu s programovacím jazykem objevila řada lepších nebo horších IDE (integrované vývojové prostředí). Pokud máte k dispozici kvalitní IDE, můžete rychle vyrobit formulář (nebo celou aplikaci) "naklikáním" vizuální části, "naklikáním" ošetření událostí a dopsáním něco málo kódu do řádků označených jako "User defined code".

První problém ale nastane hned v úvodu, kdy si máte zvolit z řady možností: chcete vytvořit JFC Applet nebo JFC Application? Nebo chcete nový AWT applet? Není problém napsat třídu nebo ji vygenerovat pomocí IDE. Problém je vědět jakou třídu potřebujete, jak a s čím bude komunikovat. Potřebujete znát architekturu systému.

Architektura, to je zjednodušeně řečeno problém "co bude kde a jak to bude spolupracovat". Java dává široké spektrum možností a je obtížné zorientovat se. Setkáte se s pojmy tenký a tlustý klient, Applet, HTML klient, dynamické HTML stránky, servlety a JSP, vícevrstvé architektury, Corba, Enterprise Java Beany, …, s databázi můžete pracovat z klienta, ze serveru přímo nebo prostřednictvím vestavěné persistence, atd. Volba architektury se stává klíčovým problémem aplikace v jazyku Java. Jaké jsou základní možnosti?

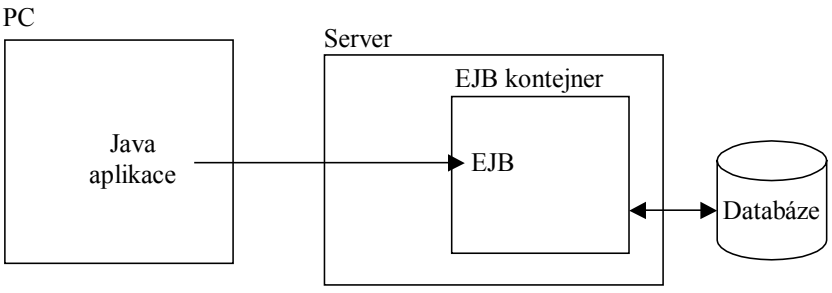
Pro rozsáhlé aplikace je bezesporu nejvhodnější vícevrstvá architektura s aplikačním serverem. Aplikační server poskytuje všem klientům přístup ke třídám jádra aplikace (business logika) a k tomu nabízí řadu výrobcem aplikovaných služeb. Aplikační server poskytuje např. tzv. EJB kontejner, kam můžete umístit své serverové EJB komponenty a ty mohou využívat vlastnosti, které jsou poskytnuty přímo kontejnerem. Sem patří přístup k databázi, řešení transakcí a konkurence atd.

Na straně klienta máme dvě základní volby – Java aplikaci nebo HTML stránku. Architektura může vypadat například takto:



Obr.3: Architektura s HTML klientem

Nebo jiná možnost:



Obr. 4: Architektura s klientem typu Java aplikace

Samozřejmě jsou možné i další varianty počínaje na straně klienta kombinací výše uvedeného (v HTML stránce jsou využity grafické prvky Java) až po "nejtlustšího" klienta, který pracuje přímo s databází. Na straně serveru jsou to možnosti obejít kontejnerovou persistenci (EJB pracuje přímo s databází), umístit na server další pomocné třídy které nejsou EJB, řešit serverovou část na standardu Corba a další. Základní rozdíl v uvedených architekturách:

HTML stránka

- Poskytuje jen omezené možnosti grafického rozhraní. K dispozici je pouze tabulka bez dalších vlastností, vstupní pole, tlačítko, volba, "rádiové" tlačítko, zaškrtávací pole, … Ve srovnání s komfortem např. JavaTable jsou funkčnost a vzhled nesrovnatelné.
- Komunikace HTML klienta se serverem je velmi rychlá, nevyžaduje se instalace JRE (Java Runtime Environment) na klienta (pokud se obejdete bez Java grafických komponent). Umístění řídící logiky na server však znamená určitou zátěž pro server.
- Velmi bezpečná architektura. Při odeslání HTML stránky se na serveru spouští automaticky servlet, který stránku obslouží. Je velmi malá pravděpodobnost, že se někomu podaří neoprávněně vniknout do vašeho systému.

Java aplikace

- Může poskytnout opravdu komfortní grafické prvky s množstvím inteligence, kontrol, modifikace vzhledu a chování.
- Na klientském PC musí být k dispozici odpovídající JRE. Podpora Javy v Microsoft IE je velmi špatná a pokud chcete v prohlížeči použít komponenty Java verze 2, obvykle musíte instalovat tzv. Plug-in s odpovídajícím JRE.
- Musíte se zabývat otázkami bezpečnosti.

6. IDE pro Javu

Ve své praxi jsem se setkal se dvěma nástroji. První kroky v oblasti Javy jsme dělali s podporou Symantec Visual Café for Java – tehdy ve verzi 3. Pokud ale chcete používat kvalitnější grafické komponenty se množstvím hotových vlastností, neobejdete se bez verze podporující Java 2. Upgrade je možný formou placeného downloadu přes Internet.

I po upgrade ovšem zůstal podstatný problém: nestabilita vývojového prostředí. Velmi negativní zkušenost se opírá o každodenní několikanásobné restartování počítače, obdenní reinstalace IDE a časté reinstalace Windows NT. Existuje závažné a poměrně reálné podezření, že tato nestabilita je způsobena českou lokalizací Windows NT, protože kolegové pracující na anglické verzi se s problémy tohoto rozsahu nesetkali.

V současné době jsme přešli na IDE VisualAge for Java firmy IBM. Základní odlišnosti proti Visual Café jsou ve správě zdrojového kódu. Café pracuje přímo se zdrojem Java, který můžete editovat běžným textovým editorem. Cenou za to je řada problémů při reálné práci, kdy některé změny, konstrukce nebo zásahy do programového kódu způsobí zcela nevyzpytatelné chování výsledku (obvykle nefunkčnost). Naproti tomu VisualAge udržuje zdrojový kód v databázi ve formě jednotlivých fragmentů. Ty jsou navigovány přes hierarchickou strukturu a ke zdrojovému kódu se dostanete teprve po provedení exportu. To má své velké výhody v čistotě vývoje, protože u Café si někdy nejste zcela jisti, které knihovny a verze zdrojů momentálně používáte (bere se vše v souborovém systému, k čemu je nastavena cesta). U VisualAge pracujete striktně se soubory umístěnými v repozitáři a pokud potřebujete nějaký vnější zdroj (obrázek apod.) musíte ho umístit do jednoho přesně

specifikovaného adresáře. Ovšem i S IBM IDE jsme zaznamenali značné problémy se stabilitou. Podezření na českou lokalizaci NT je opravdu silné.

Zvláštní kapitolou jsou nároky IDE na váš počítač. Počítač s Pentium III 800 MHz a 256 MB RAM jsme z počátku považovali za předimenzovaný. V reálném vývoji si však brzy ověříte, že se jedná prakticky o minimální rozumnou konfiguraci.

7. Příklad použití

Nyní se vrátíme k problematice vývojového procesu a ukážeme si nástin praktického použití UML, CASE nástroje a IDE při řešení aplikace.

Tento proces by měl zjednodušeně obsahovat následující části:

- Specifikace požadavků
- UseCase model
- Návrh sekvence obrazovek (screenflow)
- Modelování UseCase v Sequence diagramech
- Identifikace potřebných tříd, jejich základních atributů a metod
- Zpřesnění modelu do implementační úrovně
- Generování kostry kódu
- Kódování

Pro ilustraci použijme fragment z reálného projektu aplikace pro velkou mezinárodní přepravní firmu. který se týká stanovení pravidel pro kalkulaci a vyúčtování přepravy.

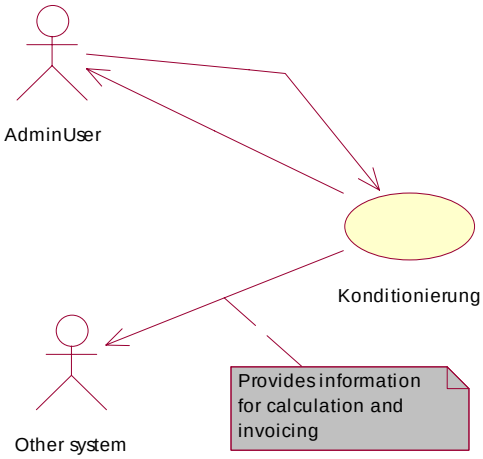
Je naprosto nezbytné ujasnit si se zákazníkem, co vlastně chce řešit. Mnohem lépe než volný nebo formátovaný text je seznam požadavků, tj. seznam jasných, pravidel a omezení, která se na dané úrovni dají dohodnout. Existují metodiky pro zpracování a zpřesnění takovýchto seznamů, ty však přesahují účel tohoto příspěvku. Proto jen některá základní pravidla:

- Seznam nemusí být úplný nebo "správný". Neexistuje správný seznam požadavků. Seznam musí vymezit rozsah řešení a hlavní rysy.
- Seznam je hierarchií požadavků. Nejjednodušší metodou stanovení hierarchie je otázka "proč" existuje daný požadavek. Pokud je odpovědí "protože existuje jiný" dostáváte hierarchii.
- V určité úrovni hierarchie se požadavek stává návrhem konkrétního řešení. Návrh řešení nepatří do seznamu požadavků!

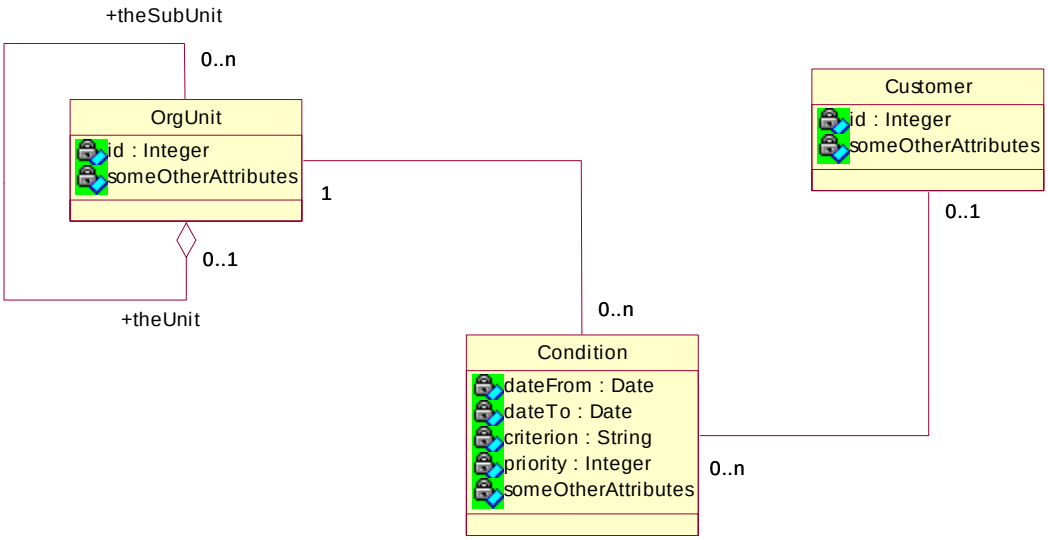
Náš seznam požadavků:
1. Poskytnutí platných parametrů pro kalkulaci a vyúčtování přepravy
2. Správa (vkládání, rušení, aktualizace) těchto parametrů

K těmto dvěma základním požadavkům existuje samozřejmě řada pravidel a upřesnění. Např.
- Parametry mohou být jak obecně platné, tak specifické pro konkrétního zákazníka
- Parametry jsou specifikovány ve vazbě na hierarchickou organizační strukturu přepravce s rysy dědičnosti z vyšších uzlů
- atd.

Základní UseCase diagram může vypadat takto:

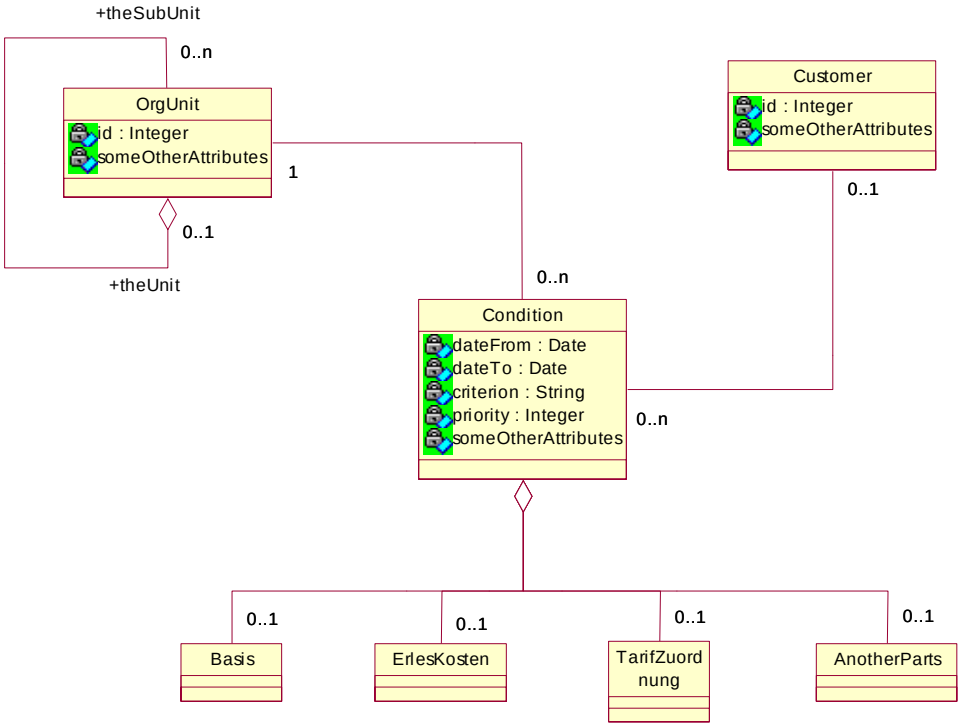


Obr. 5: Základní UseCase diagram



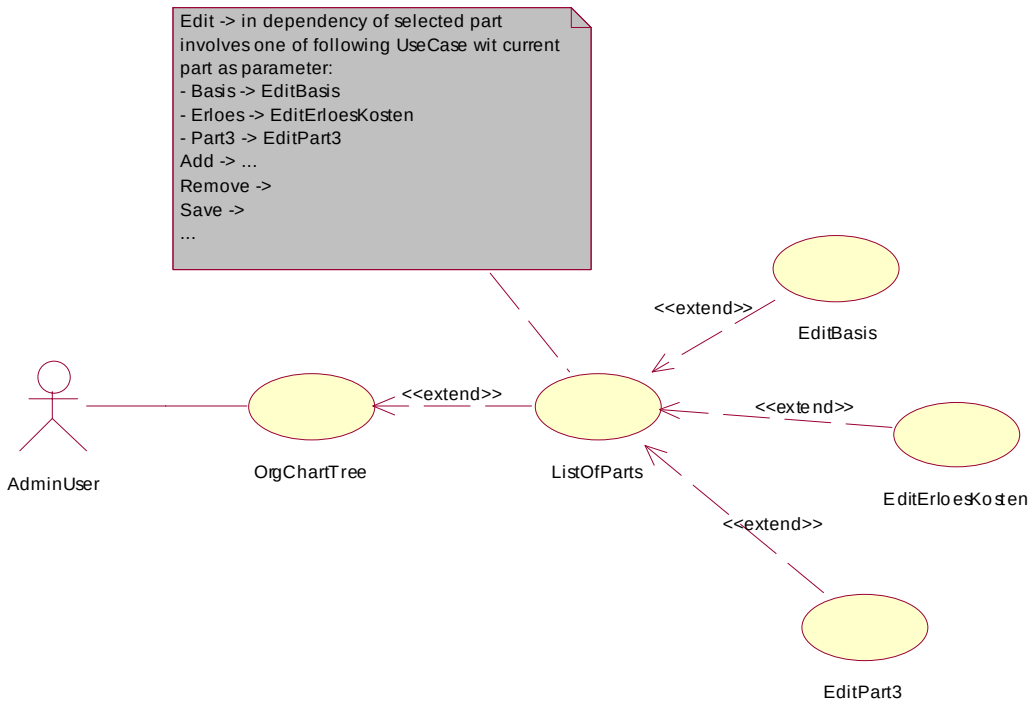
Obr. 6: Základní Class diagram

Na základě možnosti definovat jednotlivé skupiny parametrů na různých úrovních organizačního stromu jsme dále zpřesnili Class diagram takto:



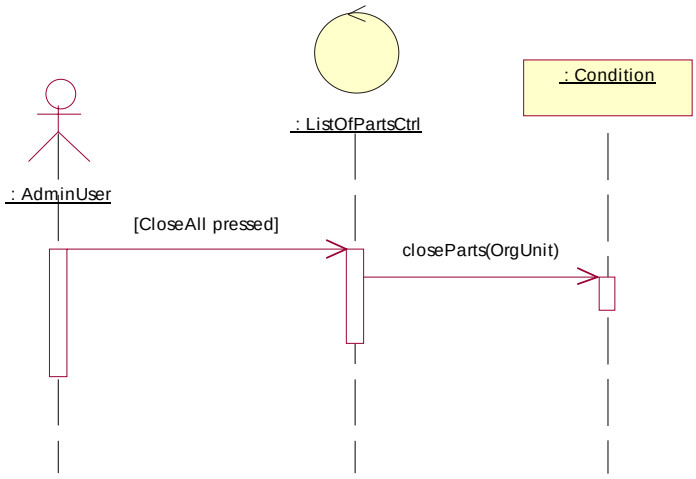
Obr. 7: Části parametrů

S ohledem na předpokládaný způsob práce uživatele při správě takto strukturovaných parametrů jsme dále zpřesnili UseCase diagram pro administraci:



Obr. 8: UseCase diagram s detailní strukturou

Tento diagram je dobrým východiskem pro modelování jednotlivých UseCase v Sequence diagramech. Tím získáte v základním pohledu potřebné business metody a v detailním zpřesnění všechny potřebné třídy a metody pro realizaci daného UseCase. Na obr. 9 je velmi zjednodušený příklad Sequence diagramu pro ukončení platnosti všech parametrů:



Obr. 9: Příklad Sequence diagramu

Zde jsme například identifikovali potřebu metody *closeParts* třídy *Condition.* Opakováním pro všechny způsoby použití daného UseCase, pro všechny UseCase pracující s danou třídou a postupným zpřesněním do všech implementačních detailů dostaneme detailní Class diagram, ze kterého můžeme generovat kostry tříd. Tyto kostry obvykle obsahují definice atributů a metod a dále metody pro přístup ke každému atributu (tzv. getry a setry).

Jak už jsem uvedl, každá vývojová firma rozšiřuje standardní modely o své zvláštnosti. V našem případě je to Activity diagram sekvence obrazovek, na základě kterého si uživatel může udělat reálnější představu o vzhledu projektované aplikace. Pokud přidáte i vzory obrazovek, budete mít později mnohem méně starostí s akceptabilitou vašeho řešení.
Tyto vzory a další diagramy zde není možné s ohledem na rozsah příspěvku prezentovat.

8. Závěr

Tento příspěvek vzhledem ke svému rozsahu nemůže detailně objasnit všechny aspekty uváděné problematiky. Mou snahou bylo přístupnou formou načrtnout problémy, se kterými se při vývoji v prostředí Java setkáte, představit některé možné metodiky a nástroje a na jednoduchém příkladu dokumentovat náš přístup k vývojovému procesu.

Je zřejmé, že pokud vaše aplikace nepřesáhne několik málo jednoduchých obrazovek, dokážete se docela dobře obejít i bez prezentovaných technik a nástrojů. Pokud však uvažujete o rozsáhlejší profesionální aplikaci, musíte se těmito věcmi vážně zabývat. Teprve velká aplikace vám ukáže nedostatky kódu generovaného vašim IDE a teprve při jejím realizování nebo údržbě si ověříte, jak dobrá je použitá metodika, jak dobrá je zvolená architektura a kde jste udělali chyby.

Literatura

1. Schmuller,J.: Myslíme v jazyku UML. Praha Grada 2001
2. Pecinovský,R.-Virius,M.: Objektové programování (1+2). Praha, Grada 1996
3. Duben,J.: Metodika analýzy a návrhu systému pomocí OMT. Skriptum VŠE Praha 1995