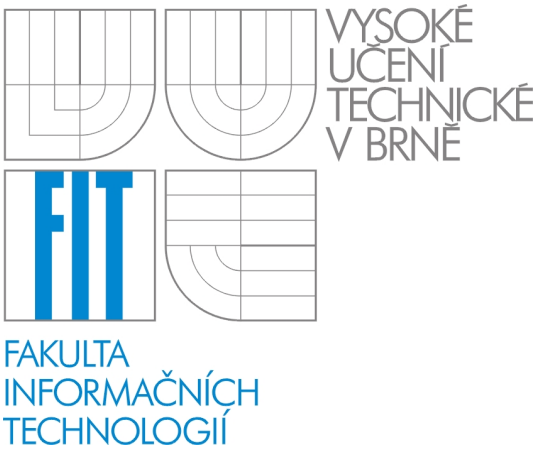


Projekt do předmětu SID

Praktické využití UML 2.0

2009 / 2010

Ing. Jaroslav Dytrych, xdytry00@stud.fit.vutbr.cz
Fakulta informačních technologií
Vysoké učení technické v Brně

Obsah

1 Úvod

UML (Unified Modeling Language) je grafický jazyk pro vizualizování, specifikování, konstruování a dokumentování artefaktů softwarově intenzivních systémů [1]. Lze jej využít ve všech fázích vývoje systému, ale největší význam má při specifikaci požadavků a návrhu systému.

Aktuální verze UML 2.0 poskytuje širokou škálu možností a její specifikace je velmi obsáhlá. V praxi je často potřeba pouze určitá podmnožina tohoto jazyka, která postačuje i pro modelování středních až větších systémů [1].

Vzhledem k omezenému rozpočtu na projekt je velmi důležité, aby bylo modelování v UML věnováno přiměřené úsilí. Pokud vytvoříme složité diagramy, kterým věnujeme mnoho času, následně zbudou malé rozpočtové prostředky a s blížícím se termínem dokončení může dojít na situaci, kdy diagramy pro svoji složitost nebudou odpovídajícím způsobem využity a systém bude vytvořen spíše živelným způsobem. Složité diagramy je rovněž obtížné aktualizovat tak, aby odpovídaly aktuálnímu stavu vývoje a byly konzistentní. Pokud jsou však diagramy na příliš vysoké úrovni abstrakce a v nedostatečném rozsahu, různé aspekty systému nebudou specifikované a může dojít k nedorozuměním a chybám. Volba úrovně abstrakce a detailů je tedy velmi důležitá.

Protože UML je jazyk určený pro modelování systémů a nikoliv pouze softwaru, lze s jeho pomocí modelovat celou firmu, aniž by v ní byly využity počítače. Modelování firemních procesů (business proces) a obchodního modelu firmy lze tedy oddělit od modelování firemního informačního systému. Při vývoji potom postupujeme tak, že nejprve vytvoříme model firmy a následně vytvoříme model nového informačního systému, ve kterém můžeme zohlednit veškeré procesy ve firmě. Pokud bychom ihned modelovali nový informační systém, mohli bychom využít pouze neucelené požadavky a informace od zákazníka. Pro malé projekty toto může být postačující, ale s velikostí projektu potom rostou nedorozumění, chyby v návrhu, implementaci a následné reklamace či přepracování softwaru.

V této práci se budu zabývat vysvětlením základních diagramů jazyka UML a jejich praktickým využitím podle [1], přičemž přístup z dané knihy rozšířím o další důležité či zajímavé poznatky a informace. Práce předpokládá určitou minimální znalost UML a nejjednodušší konstrukce jsou proto vysvětleny méně a spíše s důrazem na uvedení důležitých či zajímavých informací.

2 Objektově orientovaný přístup

UML využívá objektově orientovaný přístup k vývoji systému. Základní filosofií tohoto přístupu je analogie s reálným světem. Vše kolem nás jsou objekty, tedy lidé, zvířata, věci apod. Tyto objekty vzájemně interagují, tedy komunikují, působí na sebe (silou) apod.

Protože objekty reálného světa jsou velmi složité a můžeme se na ně dívat z mnoha aspektů (kámen je složen z atomů, má svůj název, může to být nástroj, stavební materiál apod.), při jejich modelování je nutná určitá abstrakce. Tato abstrakce nám umožňuje uvažovat pouze ty aspekty objektů, které jsou pro nás významné. Každý objekt potom bude mít pouze vybrané vlastnosti (atributy) a může provádět pouze vybrané akce (metody objektu).

Objekty seskupujeme do tříd, což jsou skupiny objektů se stejnými vlastnostmi a metodami. Třídu lze chápat i jako určitou obdobu uživatelského datového typu.

3 Pohledy

Na každý systém se můžeme dívat z různých pohledů. Základními pohledy jsou intuitivně pohled zákazníka, který se zaměřuje především na funkcionalitu (představa uživatelského rozhraní), a pohled vývojáře, který uvažuje, jak systém navrhnout a implementovat. Tyto 2 pohledy by však pro vývoj složitějšího systému nebyly dostačující. Na projektu je angažována celá řada lidí, z nichž pro každou skupinu jsou důležité jiné informace (jiný pohled). Z tohoto důvodu máme specifikovanou řadu pohledů na systém, z nichž každý je složen z určitých diagramů ve variantách pro daný pohled. Základními pohledy pro vytváření obchodních modelů firmy jsou [1]:

- Externí pohled
- Interní pohled

Základními pohledy pro modelování informačních systémů jsou [1]:

- Externí pohled (pohled uživatele)
- Strukturální pohled (pohled na strukturu)
- Behaviorální pohled (pohled na chování a stavy systému)

- Interakční pohled (pohled na interakce uvnitř systému)

Pokud prostudujeme všechny pohledy, měli bychom mít dostatek informací o celém systému. Je to obdoba pohledů na krychli, kdy s využitím několika pohledů získáme informace o celém jejím povrchu. Uvedené pohledy tedy tvoří určitý celek. Pokud bychom se na systém podívali z jiných pohledů, tyto by opět mohly utvořit ucelenou skupinu, která by nám poskytla dostatek informací.

Pokud prodáváme „krabicovou“ verzi samostatného programu, nemusíme se při modelování zabývat nasazením u zákazníka (může být řešeno na úrovni jednoduchého instalátoru a manuálu). Pokud však vytváříme podnikový informační systém, je třeba se zabývat i jeho nasazením. Toto zahrnuje umístění odpovídajících programů (částí systému) na jednotlivé počítače. Jedná se o zcela odlišný pohled na systém, který mapuje jeho části na konkrétní počítače. Počet různých pohledů se tedy mění podle složitosti systému a toho, jaké fáze jeho životního cyklu budeme modelovat.

Alternativní sadou pohledů pro modelování IT systému tedy může být i tzv. 4+1 pohled na ahchitekturu, který zahrnuje následující pohledy [9]:

- Pohled případů použití
- Logický pohled
- Pohled procesů
- Implementační pohled
- Pohled nasazení

Tyto pohledy se od výše uvedených liší tím, které diagramy jsou v nich obsaženy. Stále v nich však najdeme základní nejvyužívanější diagramy a tyto pohledy tedy pouze jiným způsobem utvoří stejný celek, zde rozšířený i o pohled nasazení.

4 Praktický příklad

Pro vysvětlení praktického využití UML jsem zvolil příklad informačního systému velkoobchodu s maloobchodem. Nebudu zde uvádět zadání ani detaily, protože cílem této práce není navrhnout takovýto systém, ale vysvětlit jednotlivé prvky UML a to pokud možno na jednom příkladu, aby jednotlivé diagramy spolu souvisely. Příklad jsem tedy značně zjednodušil a uvedu z něj pouze určité segmenty tak, abych pokryl jednotlivé prvky diagramů, ale zachoval jednoduchost a přehlednost. Za účelem pokrytí prvků diagramů jsem v některých případech „zbytečně“ zvolil využití některých konstrukcí (paralelní provádění akcí, které mohou proběhnout sekvenčně apod.). Uvedené diagramy také nejsou kompletní, ale jsou uvedeny pouze jejich části (např. chybí některé třídy v diagramu tříd apod.). Tímto však neutrpí správnost či smysluplnost využití UML, ale smysluplnost a správnost návrhu systému využitého jako příklad, což vzhledem k cíli práce nevadí. Tímto jsem docílil zkrácení a zpřehlednění textu (pro správný návrh zcela smysluplného systému a současné vysvětlení všech prvků diagramů by byl třeba několikanásobně větší prostor).

V každém diagramu UML mohou být uvedeny poznámky, které umožňují vysvětlit nejasnosti a uvést doplňující informace k modelu. V obrázcích v tomto textu jsem však tyto poznámky využil především pro přehledné popisy a vysvětlování konstrukcí diagramů.

5 Modelování obchodního (business) systému

Obchodní systém zahrnuje organizační model firmy a procesy ve firmě. Protože tento model slouží jako jeden ze základních zdrojů informací pro návrh a modelování informačního systému firmy (než naprogramujeme určitou činnost, je třeba jí porozumět), do modelu vybíráme pouze ty aspekty, které budeme později potřebovat pro návrh tohoto systému.

Pokud budeme navrhovat informační systém pro objednávky, účty, skladové hospodářství, pokladnu apod., ale nebudeme se zabývat správou zaměstnanců a mezd, není třeba modelovat všechna místa ve firmě a organizační vztahy mezi zaměstnanci. Např. informace, že nadřízeným údržbáře je správce budovy nás tedy nezajímá.

Model obchodního systému se skládá ze dvou pohledů – externího a interního.

5.1 Externí pohled

V externím pohledu se na modelovaný systém díváme jako na uzavřenou černou krabici (black box) a zabýváme se interakcí systému s jeho okolím. V našem příkladu zde budou popsány interakce se zákazníky, dodavateli, obchodními partnery a dalšími systémy (např. státní správa). V tomto pohledu se využívají dva diagramy:

- diagram případů použití,
- diagram aktivit.

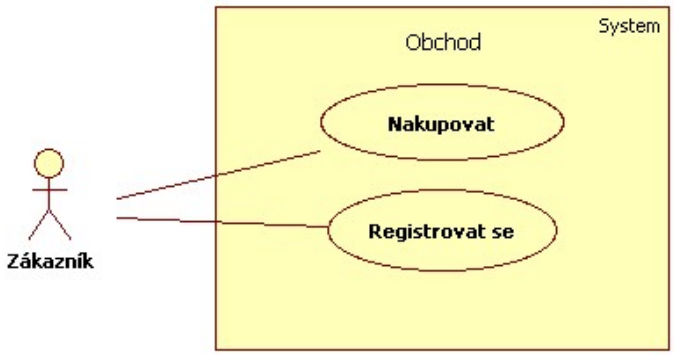
5.1.1 Diagram případů použití

Diagram případů použití popisuje, co mohou externí objekty (zákazník, dodavatel, ...) se systémem dělat, tedy např. že zákazník může nakupovat.

Základními prvky diagramu jsou:

- aktér (externí objekt, např. zákazník),
- případ použití,
- asociace (vztah aktéra k případu použití),
- ohraničení systému.

Ukázka zjednodušeného diagramu případů použití je na obrázku 1.



Obrázek 1: Diagram případů použití pro obchodní systém

Asociace mezi aktérem a případem použití vyjadřuje, že aktér je zainteresován do daného případu použití. Nespecifikuje však, jakým způsobem jej může provádět. Pokud tedy existují asociace mezi jedním případem použití a dvěma aktéry, z diagramu nelze určit, zda činnost mohou provádět jednotlivě či společně. Obvyklý význam však je, že daný aktér může provést daný případ použití bez účasti jiného aktéra (jiné situace je třeba dobře zdokumentovat). Asociace je obousměrná (může-li zákazník nakupovat, lze říci, že nakupování může provádět zákazník).

Mezi aktéry nesmí být zahrnut samotný systém či jeho databáze (vyjádřili bychom, že systém používá sám sebe), ale mohou zde být zahrnuty externí systémy obchodních partnerů a další externí objekty, které mohou se systémem interagovat.

V diagramu případů použití můžeme využít i další konstrukce, které popíšu v kapitole 6.1.1.

5.1.2 Diagram aktivit

Diagram aktivit slouží k vyobrazení průběhu určité aktivity, přičemž využívá funkcionální přístup. Základy jeho syntaxe jsou převzaty z Petriho sítí a vývojových diagramů. Umožňuje detailně specifikovat průběh konkrétního případu použití.

Prvky diagramu aktivit jsou:

- aktivita (ohraničení celého diagramu),
- výchozí uzel,

- akce (jeden krok v aktivitě, např. 1 výpočet),
- volání aktivity (vnořená aktivita uvedená v jiném diagramu),
- příjem události (akce čekání na událost),
- příjem časové události (akce čekání na konkrétní čas),
- zaslání signálu,
- hrana toku řízení,
- tok objektů,
- rozhodovací uzel,
- slučovací uzel,
- větvení (fork; paralelní toky),
- spojení (join; rozvětvených toků),
- koncový uzel toku,
- koncový uzel aktivity,
- segment aktivity (obvykle sloupce (swimlanes) nebo mřížka (grid)).

Celý diagram popisuje jednu aktivitu. Pokud je ihned za výchozím uzlem uzel čekání na událost, výchozí uzel nemusí být vyobrazen (okamžitě se přejde na čekání na událost). Události mohou být signály, akce od uživatele apod.

Spouštění jednotlivých akcí je obdobné jako u Petriho sítí. Předpokládejme, že v každém výchozím uzlu je jedna tečka. Akce může být provedena, pokud má tečku na každém vstupu, a po jejím provedení se uvolní tečka na každém výstupu. Skrze rozhodovací uzel projde tečka pouze na jeden z jeho výstupů, slučovací uzel ihned propustí tečku z libovolného vstupu na výstup. Uzel větvení při příjmu tečky na vstupu uvolní tečku na každém výstupu, uzel spojení uvolní tečku na výstupu, pokud má tečku na každém vstupu (synchronizace).

Pokud je některá akce složitá, můžeme ji popsat samostatným diagramem a následně ji volat z jiných diagramů. Toto umožňuje znovupoužitelnost, začlenění jedné aktivity do druhé a strukturování složitých diagramů. Pokud je diagram složitý, můžeme jej na nejvyšší úrovni vytvořit hodně abstraktní (např. zákazník přijde, nakoupí, odejde), přičemž jednotlivé akce budou voláním aktivit z jiných podrobnějších diagramů (např. zákazník otevře dveře, vstoupí, vezme si vozík, ...). Příliš mnoho úrovní vnoření však může způsobit nepřehlednost a při konzultaci se zákazníkem potom může dojít ke značnému listování při procházení aktivity.

Zaslání signálu umožňuje asynchronní volání jiné aktivity. Kromě toku řízení je v diagramu možné vyjádřit i tok objektů.

Pokud některá tečka dosáhne koncového uzlu toku, daný tok končí, ale aktivita může stále pokračovat jinými toky. Pokud některá tečka dosáhne koncového uzlu aktivity, všechny toky jsou ukončeny a aktivita končí.

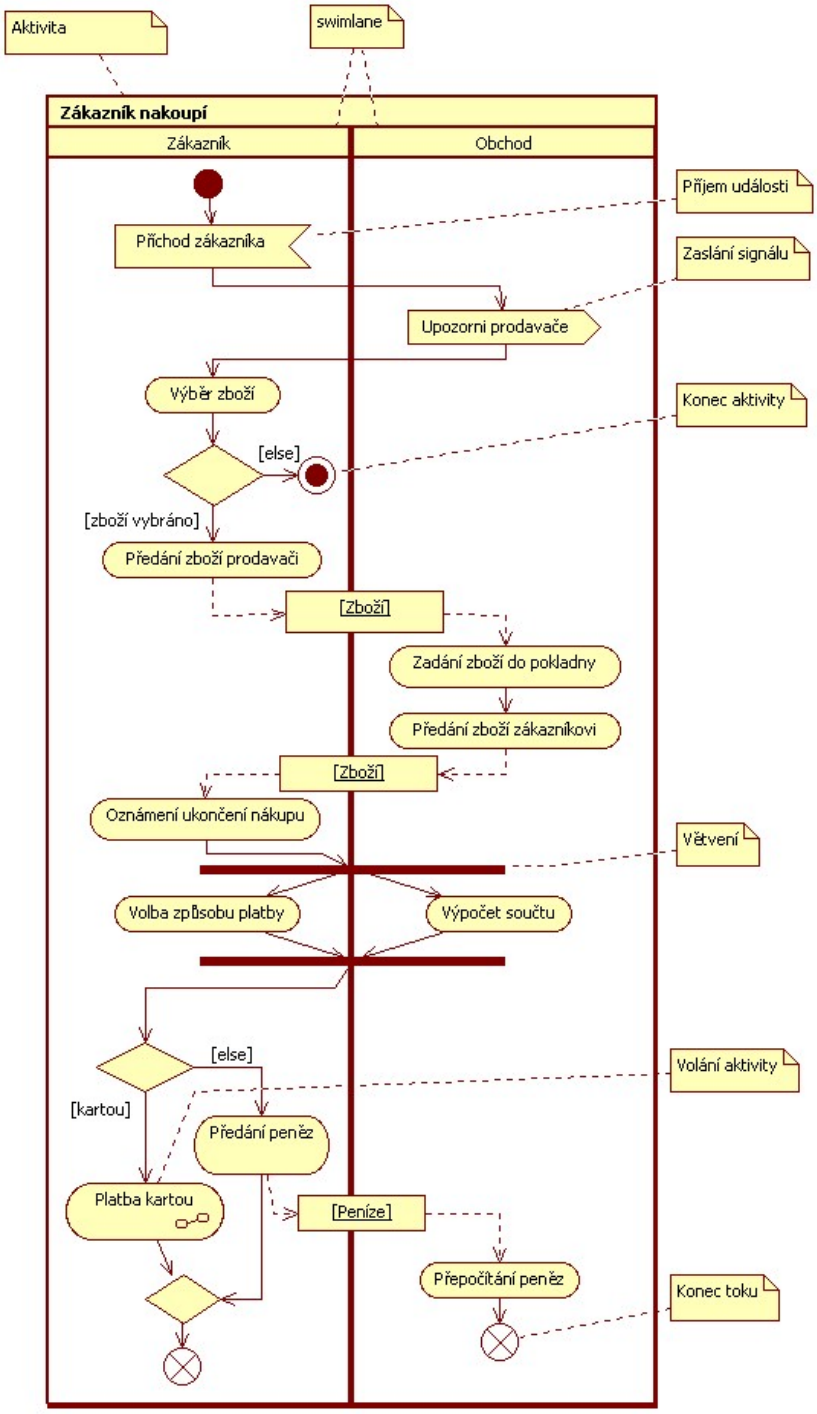
Kromě uvedených má diagram i další prvky (např. výjimky) a umožňuje některé složitější konstrukce jako výrazy u uzlů spojení (částečná synchronizace). Tyto prvky jsou však méně využívané a nebudu se zde jimi zabývat. Více informací lze nalézt v [3].

Diagram aktivit pro případ použití nákupu zboží je na obrázku 2.

Předpokládejme, že máme velkoobchod s maloobchodní prodejnou. Po příchodu zákazníka do prodejny je třeba upozornit prodavače (např. zvukový signál při otevření dveří), zatímco si zákazník začne vybírat zboží. Pokud si zákazník nic nevybere, odchází a případ použití končí. Pokud si zákazník něco vybral, předá košík prodavači, který zboží zadá do pokladny a vrátí jej zákazníkovi, který si jej naskládá do tašky. Zákazník následně oznámí prodavači, že ukončil nákup a ten vypočítá součet ceny, přičemž si zákazník zvolí způsob platby. Následně je provedena platba a zákazník odchází.

Všimněme si, že v diagramu se nevyskytuje prodavač, ale celý modelovaný systém. Nerozlišujeme zde, které operace provádí prodavač, které pokladna a které jiná součást obchodu. Na obchod pohlížíme jako na objekt, jehož vnitřní struktura bude popsána v interním pohledu.

Pro každý složitější či nejasný případ použití je třeba vytvořit diagram aktivit. Případy použití sice můžeme popsat i textově (tabulkou, viz níže), ale při využití více rozhodovacích uzlů či paralelního toku se tento popis rychle stává nepřehledným, zatímco ve vhodně strukturovaném diagramu jsou jednotlivé větve stále přehledně vidět.



Obrázek 2: Diagram aktivit pro případ použití nákupu zboží

Pokud budeme dostatečně dlouho sledovat zaměstnance při jeho práci, snadno vytvoříme diagram, který jeho činnost popisuje. Pokud budeme informace zjišťovat formou textových dotazníků či pohovoru, zaměstnanec může některé rutinně vykonávané akce opomenout, nebo uvést mimo pořadí. Potom může být např. vytvořen terminál, na kterém je nutné zadat způsob platby před zadáním položek zboží, což je méně uživatelsky přívětivé (co se stane, pokud zákazník zjistí, že nemá dostatečnou hotovost?). U složitějších činností, kterým návrhář systému nerozumí, mohou být tyto diagramy klíčové pro správný návrh systému.

5.1.3 Diagram sekvence

Diagram sekvence slouží k popisu určité sekvence interakcí. Podobně jako diagram aktivit se zde využívá k popisu případu použití. Oproti diagramu aktivit ale umožňuje vyjádřit posloupnost akcí v čase a přehledněji zobrazuje interakce. Jedná se o dvourozměrný diagram, kde osa y tvoří časovou osu a na ose x jsou uvedeny jednotlivé interagující objekty.

Diagram má následující základní prvky:

- komentáře,
- objekty,
- vytváření objektů,
- zprávy.

Komentáře sice mohou být ve všech UML diagramech, ale v tomto diagramu mají velký význam. Umožňují uvedení aktivit interagujících objektů včetně podmínek, cyklů apod.

V diagramu pro obchodní systém bude uveden tento systém, externí uživatelé, systémy a další interagující objekty. Konstrukce pro vytváření objektů zde není obvyklá (bude vysvětlena níže u diagramu pro IT systém).

Každý objekt je zobrazen jako obdélník, který může být označen třídou a názvem (oddělujeme dvojtečkou), a čárou života, která zobrazuje jeho existenci v čase (lifeline). Obdélník umísťujeme do místa vytvoření objektu, nebo do horní části diagramu (nad počátek aktivity), čímž vyjádříme, že objekt před zahájením aktivity již existoval. V místě, kde je objekt zrušen, se čára ukončuje křížkem (X), aby toto místo bylo odlišitelné od konce zobrazeného úseku čáry, která pokračuje dále v časovém kontinuu. Výjimkou je aktér, u kterého můžeme místo obdélníku využít ikonu aktéra (viz níže v modelu IT systému).

Úseky, ve kterých je objekt aktivní, jsou na čáře života objektu zobrazeny obdélníkem (rozšíření čáry). Pokud objekt předá řízení jinému objektu (žádost o nějakou akci, volání funkce apod.), zůstává dále aktivní. Pokud objekt provádí více aktivit současně, tuto skutečnost lze zobrazit více obdélníky s částečným překryvem.

Interakce probíhají formou zasílání zpráv. Tyto zprávy mohou mít jako parametry objekty (business object), což mohou být jak objekty reálného světa, tak i datové objekty s potřebnými informacemi. Zprávy mohou být synchronní (čeká se na odpověď) i asynchronní. Odpověď na zprávu (např. návratová hodnota funkce) do diagramu obvykle nezakreslujeme (uvádí se k signatuře zprávy před rovnítko), ale pokud je třeba návratovou hodnotu zdůraznit nebo je odpovědí, která není přímo návratovou hodnotou dané funkce (zprávy), můžeme ji uvést.

Kromě základních prvků lze v diagramu využít také:

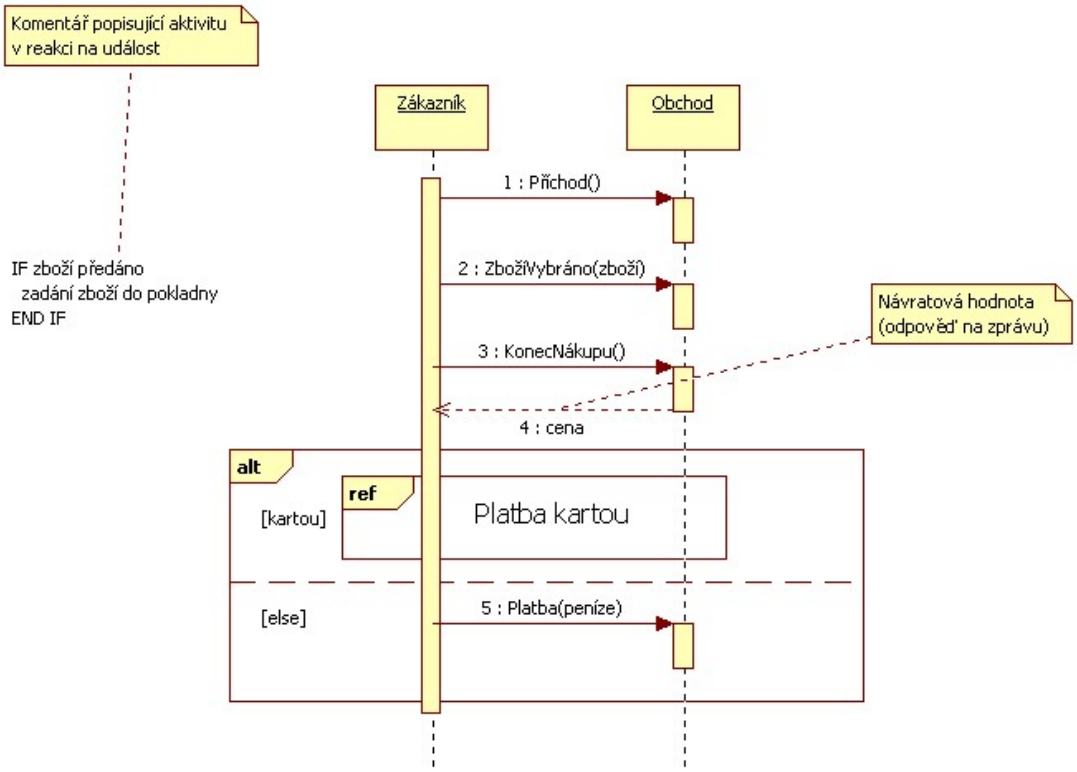
- podmínky (opt),
- větvení (alt),
- cykly (loop),
- seskupování,
- paralelismus (par),
- reference (ref),
- zpětná volání,
- rekurzivní volání,
- ...

Podmínky, větvení, cykly, seskupování a paralelismus jsou vyobrazeny jako rámce ohraničující určité části interakce (tzv. kombinované fragmenty [9]). Pokud jsou využity ve větší míře, diagram se stává značně nepřehledným. V takovém případě je lepší aktivitu strukturovat a diagramem interakce vyjádřit části aktivity s interakcí, zatímco pro části se složitějším tokem řízení využijeme diagram aktivit. Můžeme také využít přehledový diagram interakce (viz níže).

Reference umožňují volání jiného případu použití v rámci dané sekvence. Zpětné volání umožňuje objektu, který provádí nějakou požadovanou aktivitu, zaslat zprávu (vyvolat akci) objektu, který jej volal (z objektu A je volána funkce objektu B, ve které je volání funkce objektu A). Při rekurzivním volání objekt zasílá zprávu sám sobě.

Vyobrazit lze i odmítnutí zprávy, vypršení času apod. Tyto prvky lze nalézt v [3].

Příklad diagramu sekvence k případu použití nákupu zboží je na obrázku 3.



Obrázek 3: Diagram sekvence pro případ použití nákupu zboží

Rozhodujeme-li se, zda případ použití popsat diagramem aktivity, interakce či oběma, měli bychom zvážit interaktivnost případu. Pokud případ použití probíhá tak, že aktér spustí nějakou akci, kterou systém vykoná, využijeme diagram aktivit. Pokud při případu použití dochází k velkému množství interakcí mezi aktérem a systémem, využijeme diagram sekvence. Pokud je v případu použití hodně interaktivity i akcí prováděných systémem a aktéry, využijeme oba diagramy.

5.2 Interní pohled

Interní pohled slouží k popisu vnitřku obchodního systému (vnitřek krabice, na kterou se díváme v externím pohledu). Základními diagramy v tomto pohledu jsou:

- diagram balíčků,
- diagram tříd,
- diagram aktivit.

Je nutné upozornit na výraznou odlišnost diagramu balíčků a diagramu tříd pro obchodní systém a pro IT systém. Narozdíl od jiných typů diagramů budou tyto v obou variantách popsány samostatně.

5.2.1 Diagram balíčků

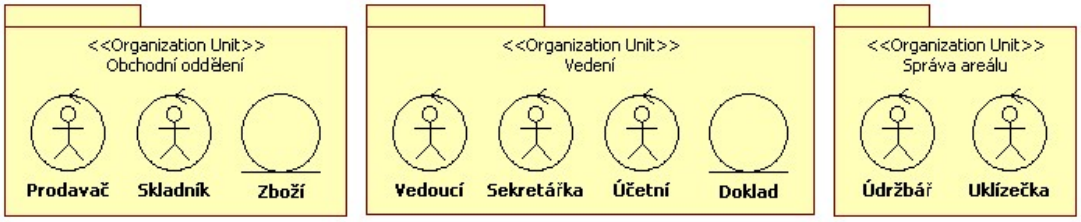
Diagram balíčků pro obchodní systém slouží ke zobrazení jednotlivých součástí systému a jejich seskupení do organizačních jednotek (např. údržbář a uklízečka patří pod správu areálu). Základními prvky diagramu jsou:

- balíček,
- třída.

Balíčky reprezentující jednotlivé organizační jednotky podniku mají stereotyp `<<Organization Unit>>`.
V tomto diagramu jsou obvykle 2 typy tříd:

- zaměstnanec (stereotyp `<<Worker>>`),
- objekt zájmu (business object; stereotyp `<<Business Object>>` či `<<Entity>>`).

Mezi objekty zájmu patří zboží, doklady a další objekty, se kterými se v systému pracuje. Pro třídy je místo stereotypů doporučeno využívat symboly, které jsou uvedeny v příkladu na obrázku 4.



Obrázek 4: Diagram balíčků pro model obchodního systému

5.2.2 Diagram tříd

Diagram tříd pro obchodní systém umožňuje vyobrazit vztahy mezi jednotlivými zaměstnanci (pracovními místy), objekty a vnějšími systémy. Zobrazuje tedy statickou strukturu organizace. Základními prvky jsou:

- třída,
- asociace,
- generalizace.

Třídy jsou stejných typů jako v diagramu balíčků, ale rozšířené o typ pro externí systémy. Při konstrukci můžeme vycházet z diagramu balíčků a diagramu případů použití.

Asociace zobrazují vztahy mezi třídami. Zatímco asociace jsou obousměrné (komunikace mezi zaměstnanci je obousměrná, pokud zaměstnanec manipuluje s objektem, objekt je manipulován zaměstnancem), jejich názvy mohou být pouze jednosměrné (zaměstnanec ověří kartu zákazníka, karta však neověřuje zaměstnance, ale je ověřena zaměstnancem).

Generalizace umožňuje definovat, že některá třída je speciálním případem jiné třídy (příjemka a výdejka jsou speciálními případy dokladu, které se týkají změny množství zboží na skladě).

Příklad diagramu je na obrázku 5.

Pro vysvětlení příkladu je nutné dodat, že prodavač může zboží ihned vydat (maloobchodní prodej), nebo pouze vystavit výdejku, podle které skladník zboží vydá ze skladu (velkoobchodní prodej).

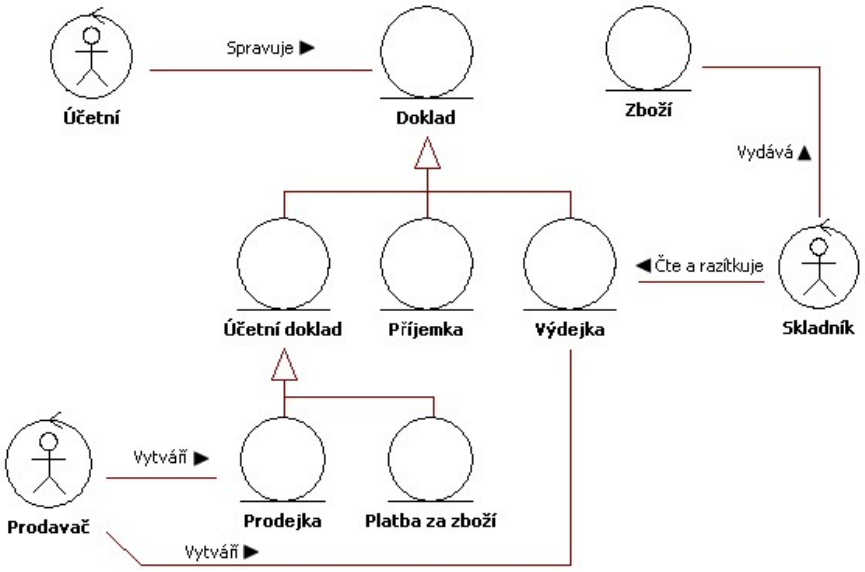
5.2.3 Diagram aktivit

Diagram aktivit v interním pohledu popisuje činnosti zaměstnanců, zařízení, informačních systémů a jiných součástí organizace.

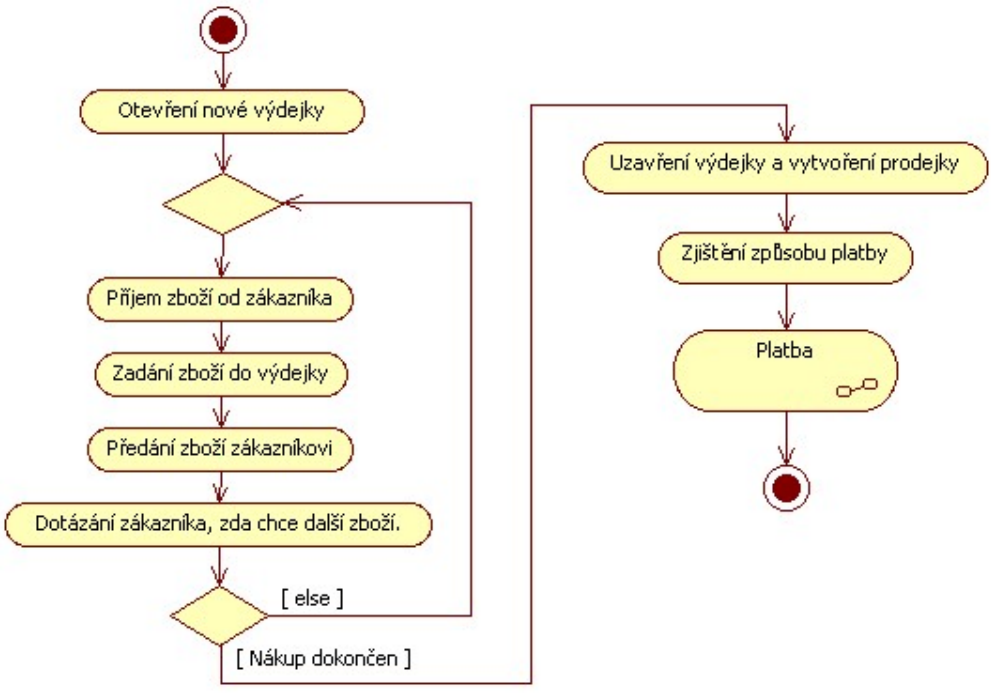
Prvky diagramu jsou stejné, jako u diagramu aktivit pro externí pohled. Díváme se však na fungování uvnitř obchodního systému. Zatímco v externím pohledu jsme popsali např. postup zákazníkova nákupu, v interním pohledu popíšeme činnost prodavače.

Z externího pohledu se dozvíme, že něco v systému zaregistruje seznam vybraného zboží a spočítá jeho celkovou cenu. Z interního pohledu se dozvíme, že prodavač otevřel nový doklad, postupně do něj přidal položky zboží, uzavřel jej a spočítal celkovou cenu.

Příklad diagramu pro případ použití nákupu zboží je na obrázku 6.



Obrázek 5: Diagram tříd pro model obchodního systému



Obrázek 6: Diagram aktivit pro případ použití nákupu zboží

6 Modelování IT systémů

Při modelování IT systémů se zaměřujeme především na programové vybavení, ale i na jeho nasazení na daném hardwaru. Zabýváme se tedy výhradně činnostmi, ve kterých jsou zapojeny výpočetní prostředky. Nezabýváme se činnostmi z hlediska obchodních procesů (prodavač bere zboží z pásu, zadává a vrací na pás), ale z hlediska software (prodavač zadá množství a EAN kód).

Při modelování IT systému vycházíme z požadavků zákazníka a znalostí z dané domény (znalost profese a činnosti zákazníka). Velkou výhodou je model obchodního systému zákazníka, který nám umožní detailní pochopení jednotlivých činností.

Pro model IT sytému můžeme využít různé sady pohledů. V této práci využiji sadu pohledů dle [1] rozšířenou o pohled nasazení. Pohledy v našem modelu tedy jsou:

- externí pohled (pohled uživatele),
- strukturální pohled (pohled na strukturu),
- behaviorální pohled (pohled na chování a stavy systému),
- interakční pohled (pohled na interakce uvnitř systému),
- pohled nasazení.

6.1 Externí pohled

V externím pohledu se na IT systém díváme z pohledu zákazníka. Zabýváme se tedy případy použití a vnějším chováním systému. Na systém nahlížíme jako na uzavřenou černou krabici a nezabýváme se tedy ani jeho vnitřní strukturou ani interními aktivitami.

Základními diagramy tohoto pohledu jsou diagram případů použití a diagram sekvence případu použití. Pokud chceme přehledně zobrazit jednotlivé obrazovky uživatelského rozhraní a přechody mezi nimi, pohled je možné doplnit i stavovými diagramy (viz kapitola 6.3.1) pro jednotlivé případy použití.

6.1.1 Diagram případů použití

Diagram případů použití IT systému je nejdůležitějším ze všech využitých diagramů. Vždy jej vytváříme jako první a konzultujeme se zákazníkem. Výslednou verzi by měl zákazník schválit a podepsat. Činnosti, které v tomto diagramu nejsou obsaženy, potom systém nebude umožňovat. Jedná se tedy o specifikaci toho, co si zákazník objednal, podle čeho bude stanovena cena (rozpočet) a co bude kontrolováno při předání výsledného produktu zákazníkovi.

První předběžnou verzi diagramu bychom měli vytvořit již před vytvářením modelu obchodního systému zákazníka, abychom následně nesledovali a nemodelovali činnosti, které se vůbec netýkají systému, který budeme vytvářet.

Je velmi důležité upozornit na rozdíl mezi aktéry v diagramu pro model IT systému a pro model obchodního systému. Zaměstnanec firmy je součástí obchodního systému a nebude tedy mezi aktéry v modelu obchodního systému. Pokud však pracuje s informačním systémem, bude aktérem v modelu IT systému. Naopak zákazník, který nemůže přímo pracovat s IT systémem, nebude ani aktérem v jeho modelu. Při konstrukci diagramu vždy zjišťujeme, kdo může přímo pracovat s IT systémem. Diagram případů použití z obchodního modelu systému je tedy možné využít spíše pro kontrolu, než jako základ pro diagram případů použití IT systému.

Základní prvky diagramu jsou stejné jako v diagramu případů použití v modelu obchodního systému (viz výše), ale můžeme zde využít i další konstrukce:

- generalizace (specializace) aktérů,
- generalizace (specializace) případů použití,
- vztahy mezi případy použití (include, extend, follows, ...),
- omezení,
- ...

Generalizace (specializace) aktérů slouží k vyjádření, že jeden aktér je speciálním případem jiného. Můžeme např. říci, že neregistrovaný zákazník, registrovaný zákazník a obchodník jsou speciálními případy zákazníka velkoobchodu z našeho příkladu.

Generalizace případů použití umožňuje vyjádřit, že jeden případ použití je speciálním případem jiného. Můžeme např. vyjádřit, že prodej za hotové a prodej na fakturu jsou speciálními případy prodeje.

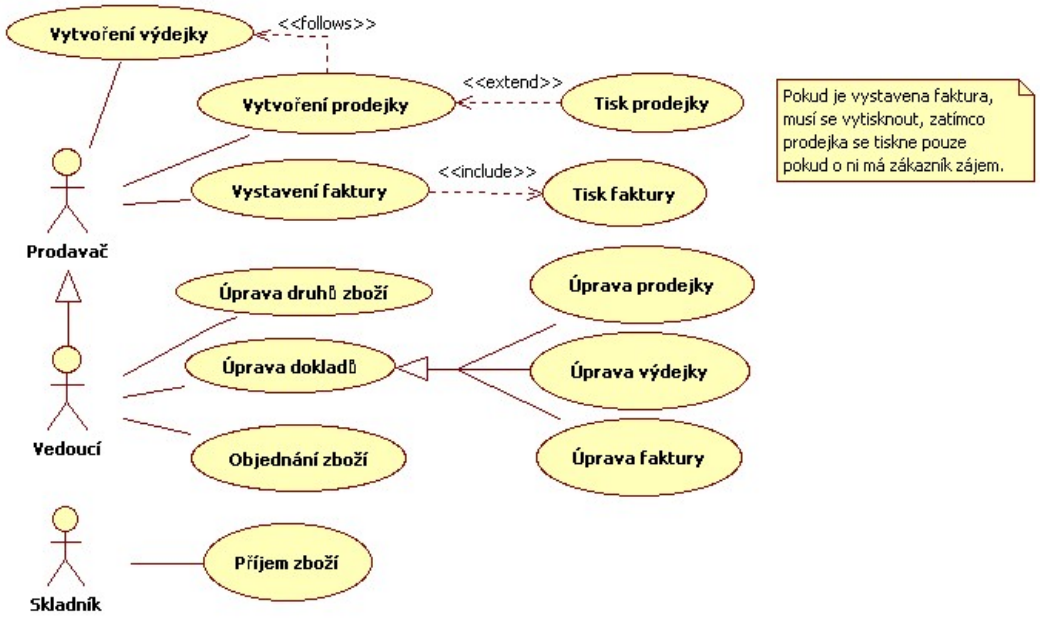
Vztah include (obsahuje či zahrnuje) se využívá k vyjádření, že využití jednoho případu použití zahrnuje i využití jiného. Podle [1] bychom měli nejprve určit základní případy použití a potom z nich „vytýkáním“ extrahovat části, které jsou ve více případech společné (např. vydání zboží při prodeji za hotové i na fakturu). Vztah include lze však využít i ke zdůraznění skutečnosti, že při nějaké činnosti se provede i jiná činnost (např. vystavení prodejky při prodeji za hotové či povinný tisk faktury při jejím vystavení).

Vztah extend (rozšiřuje) lze využít k vyjádření, že jeden případ použití může zahrnovat i provedení jiného. Příkladem je tisk dokladu při prodeji za hotové (zákazník doklad může, ale nemusí chtít).

Vztah follows (následuje) umožňuje vyjádřit, že některý případ použití může být proveden jedině po provedení jiného. Omezení a další prvky diagramu již nebudu uvádět, protože se využívají velmi málo, výrazně znepřehledňují diagram a ztěžují jeho pochopení zákazníkem.

Uvedené konstrukce je možné využít i v diagramu v obchodním modelu, ale není to obvyklé ani doporučované.

Diagram případů použití s využitím složitějších konstrukcí je uveden na obrázku 7.



Obrázek 7: Diagram případů použití pro model IT systému

Protože diagram případů použití je vždy základním diagramem pro komunikaci se zákazníkem, je nutné, aby zákazník diagramu porozuměl. Vztah aktéra k jednoduchému případu použití zákazník snadno pochopí (z pohledu „běžného uživatele“ panáček dělá nějakou činnost). Veškeré další konstrukce v diagramu je třeba zákazníkovi s menším či větším úsilím vysvětlit. Generalizace (specializace) uživatelů je obvykle snadno pochopitelná, obsažení jednoho případu použití v druhém také. Další vztahy mezi případy použití, generalizace případů použití apod. jsou již obtížně pochopitelné a často jim správně neporozumí ani vývojáři. Pokud takové komplikované konstrukce zákazníkovi vysvětlíme, diagram sice může pochopit, ale pravděpodobně pro něj bude nepřehledný a snadno v něm přehlédne chyby. Z toho potom plynou nedorozumění, snadnější možnost vniku nekonzistence v modelu a nakonec i nespokojený zákazník.

Informace, které v diagramu nejsou, mohou být stejně důležité jako informace, které v něm jsou. Pokud v diagramu není asociace uklízečky s případem použití „naskladnění zboží“, je vysoce pravděpodobné, že si zákazník nepřeje, aby uklízečka tuto činnost mohla provádět. Z tohoto plynou rovněž vysoké nároky na správnost a úplnost diagramu.

Detaily případů použití

K diagramu případů použití by vždy měl být sepsán i dokument s detaily případů použití. Jeho konkrétní forma není v UML specifikována a není uvedena ani v jiném souvisejícím standardu. Jako vhodná forma je doporučována tabulka, jejíž pole se mohou podle potřeby a organizačních zvyklostí lišit. Příklad této tabulky je tabulka 1.

Identifikátor	UDZ1			
Název	Přidání druhu zboží			
Popis	Uživatel U přidá další druh zboží			
Priorita	2 = střední	**Frekvence**	Několikrát za měsíc	
Počáteční podmínky	Uživatel U je přihlášen do systému a má funkci vedoucího.			
Koncové podmínky	Byl přidán nový druh zboží			
Aktéři	Uživatel U			
Základní posloupnost	**Krok**	**Činnost**		
	1	U vyplní informace o novém druhu zboží		
	2	**Když** je některá informace špatně zadaná: 2.1 Systém zobrazí chybové hlášení a přejde na krok 1		
	3	Systém přidá nový druh zboží a zobrazí hlášení o úspěšném provedení operace		
Alternativní posloupnost	**Krok**	**Činnost**		
	1	U se může kdykoliv vrátit do hlavní nabídky		
Výjimky z normální posloupnosti	**Číslo**	**Popis**	**Ošetřeno krokem**	
	1	U nevyplnil všechny povinné údaje ve formuláři a zvolil uložení nového druhu zboží	2	
	2	U vyplnil některou položku chybně a zvolil uložení nového druhu zboží	2	
Poznámky				

Tabulka 1: Detail varianty případu použití Úprava druhů zboží

6.1.2 Diagram sekvence případu použití

Diagram sekvence případu použití (Use Case Sequence Diagram) se využívá k detailnímu popisu případu použití. Vzhledem k tomu, že se nachází v externím pohledu, jsou v něm obsaženy pouze interakce aktéra (uživatele či jiného systému) s IT systémem.

Diagram má stejné prvky jako diagram sekvence v modelu obchodního systému, navíc se však využívají některé stereotypy. Pro označení objektu reprezentujícího modelovaný IT systém se využívá stereotyp `<<IT System>>`.

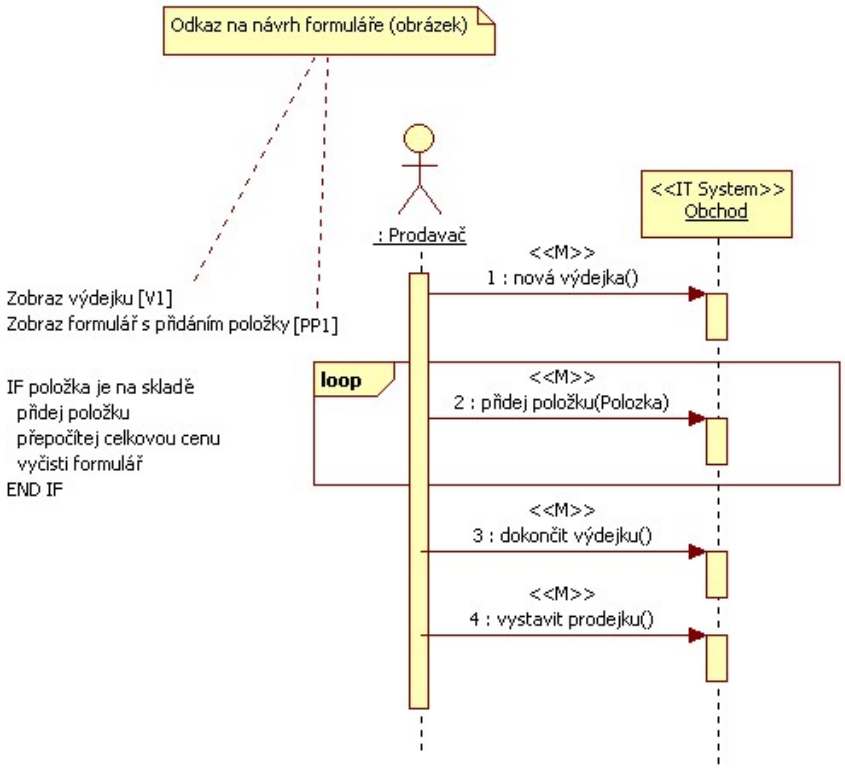
Dle [1] je vhodné rozdělit zprávy (události) na dotazovací (query) a změny (mutation).

Dotazovací zprávy nemění stav systému. Aktér se systému pouze dotazuje na určité informace. Pro tento typ událostí se využívá stereotyp `<<Q>>`. **Zprávy požadující změnu** mění stav systému (např. změna dat v databázi). Pro tento typ událostí se využívá stereotyp `<<M>>`.

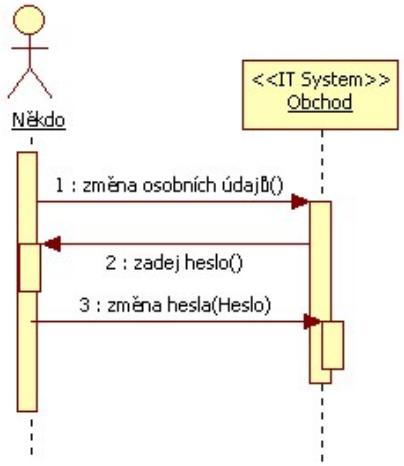
Dalším využitelným prvkem diagramu sekvence je reference na prototyp uživatelského rozhraní. Jedná se o číslo obrazovky (obvykle snímek či náčrt obrazovky) a uvádí se v hranatých závorkách (obvykle v komentáři). Především u formulářů a složitějších obrazovek je grafický návrh obrazovky velmi důležitý. Uživatel často očekává shodu či podobnost s formuláři, na které je zvyklý z předchozího systému či které využíval v papírové podobě.

Příklad diagramu sekvence případu použití maloobchodního prodeje je na obrázku 8.

Pokud potřebujeme popsat případ použití, který může provádět kterýkoliv uživatel (např. změna hesla), můžeme využít aktéra „Někdo“. Tento aktér zastupuje libovolného uživatele systému a i když tato generalizace není v diagramu případů použití explicitně uvedena, můžeme ji podle [1] využít. Příklad takového diagramu je na obrázku 9.



Obrázek 8: Diagram sekvence případu použití maloobchodního prodeje



Obrázek 9: Diagram sekvence případu použití Změna hesla

6.2 Strukturální pohled

Strukturální pohled zobrazuje statickou strukturu systému. Jedná se o typ interního pohledu (zabývá se vnitřní strukturou systému), jehož základním diagramem je diagram tříd.

U větších modelů je vhodné model strukturovat seskupováním prvků do balíčků, k čemuž využijeme diagram balíčků. Tento je potřebný také pokud využíváme návrhové vzory jako např. MVC (Model View Contoller) nebo v případě implementace systému v jazyce Java. Balíčky využíváme také pro jmenné prostory (např. v C++).

6.2.1 Diagram tříd

Diagram tříd slouží ke zobrazení objektů v IT systému a vztahů mezi nimi. Jeho základními prvky jsou:

- třída,
- asociace,
- asociační třída,
- generalizace,
- agregace,
- kompozice.

Třída má název, atributy (vlastnosti) a metody. Asociace vyjadřuje vztah mezi třídami. Asociace může mít název, názvy rolí objektů ve vztahu (např. vedoucí) a kardinality. Kardinalita udává, kolik objektů dané třídy do vztahu vstupuje. Udává se jako rozsah, přičemž minimální kardinalita je obvykle 0 (objekt nemusí být ve vztahu, např. zákazník nemusí mít žádnou objednávku) či 1 (objekt musí být ve vztahu). Maximální kardinalita je obvykle 1 či * (libovolné kladné celé číslo). Pokud je uvedena pouze jedna hodnota, jedná se o minimální a současně maximální kardinalitu, přičemž * reprezentuje nezáporné celé číslo.

Při určování minimální kardinality je třeba obezřetnost, protože může představovat integritní omezení. Pokud např. řekneme, že zákazník nemůže v systému existovat bez objednávky, pravděpodobně později dojde k problémům (jak registrovat zákazníka bez objednávky) a bude nutné provést úpravy v návrhu, případně i v implementaci. Pokud však povolíme objednávku bez zákazníka, také to nebude v pořádku. Obecně platí, že pokud minimální kardinalita není zřejmá, měli bychom volit 0 (nepovinné členství ve vztahu), abychom nezaváděli chybné integritní omezení.

Název asociace může mít stejně jako u diagramu tříd pro obchodní systém udaný směr. Pokud má asociace další vlastnosti (např. počet kusů a cena v asociaci zboží s objednávkou), nestačí nám pouhá asociace, ale musíme využít asociační třídu. Zatímco běžná asociace je realizována tak, že jeden objekt ve vztahu má odkaz na druhý, asociace pomocí asociační třídy je realizována samostatným objektem. Objekty ve vztahu potom mají odkazy na asociační objekt, který je nositelem parametrů vztahu.

Generalizace umožňuje vyjádřit, že jedna třída je speciálním případem druhé. Tento vztah označujeme také jako dědičnost, protože speciálnější třída dědí rysy obecnější třídy. Vždy platí, že na místě obecnější třídy můžeme využít i speciálnější, ale ne naopak (kde pracujeme s dokladem, můžeme pracovat i s objednávkou, ale na místo objednávky nemůžeme vložit libovolný doklad).

Agregace je speciálním případem vztahu, kdy jedna třída je součástí druhé (objekt jedné třídy je součástí objektu druhé třídy). Např. objednávka se skládá z položek. Kompozice je silnější variantou agregace s povinným členstvím, kdy říkáme, že jedna třída se skládá z jiných (např. firma se skládá ze zaměstnanců a nemůže bez nich existovat). U kompozice je složená třída zodpovědná za vytváření a rušení svých složek.

V diagramu tříd mohou být i další prvky:

- rozhraní,
- abstraktní třída,
- vnoření (jeden objekt je součástí druhého),
- navigovatelnost vztahů (objekt A ví o B, ale B neví o A),
- kvalifikovaná asociace (s využitím externího klíče),
- omezení vztahu,
- ...

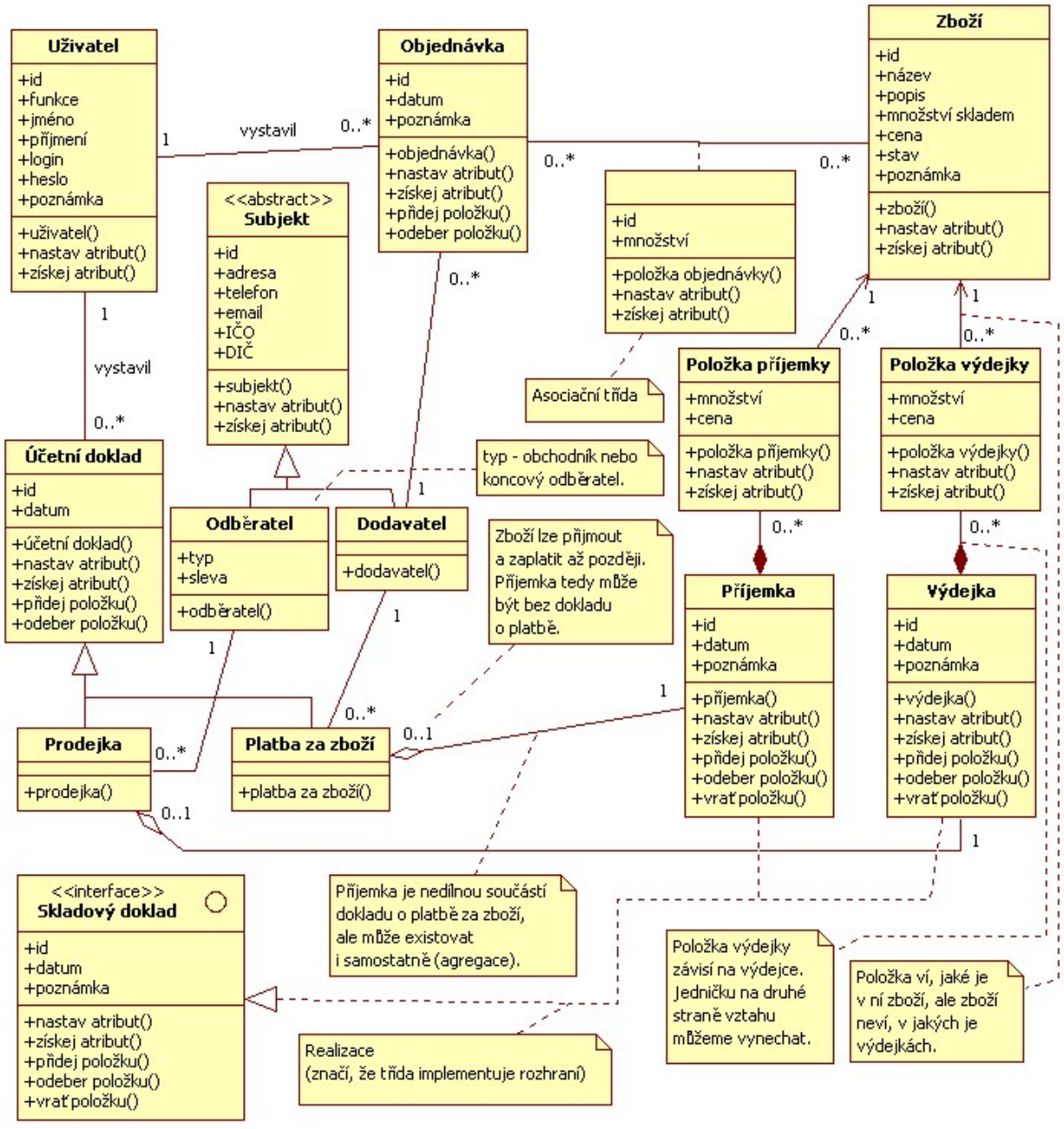
Rozhraní deklaruje veřejné vlastnosti a metody, ale neobsahuje jejich implementaci. Rozhraní se využívá tam, kde je třeba, aby do vztahu mohl vstoupit libovolný objekt, který splňuje daná kritéria (poskytuje dané metody).

Abstraktní třída poskytuje abstrakci jiných tříd. Je možné z ní dědit, ale není možné vytvářet instance této třídy (nelze vytvořit objekt dané třídy). Abstraktní třída může mít implementovanou pouze část metod. Výhodou je, že pokud v nějakém vztahu využijeme abstraktní třídu, může do vztahu vstoupit libovolná její podtřída. Pokud následně potřebujeme přidat metodu, můžeme ji přidat pouze do abstraktní třídy a případně předefinovat v podtřídách,

ve kterých to má smysl. Pokud bychom využili rozhraní, museli bychom novou metodu implementovat ve všech třídách, které rozhraní implementují.

Zbývající prvky diagramu zde nebudu popisovat, protože jejich význam je buď zřejmý, nebo jsou méně využívané. Více informací lze nalézt v [3].

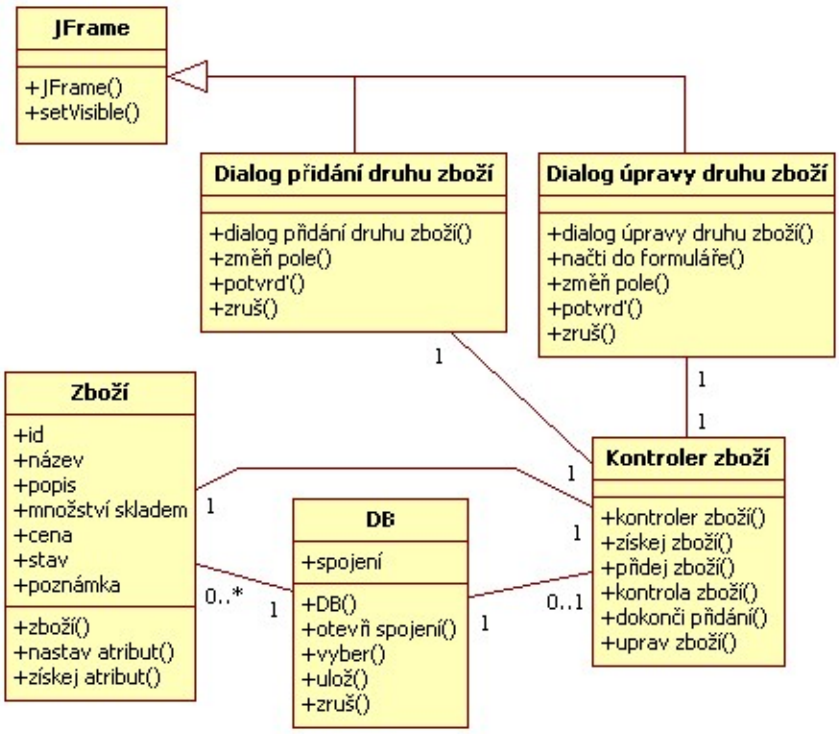
Příklad diagramu tříd je na obrázku 10.



Obrázek 10: Diagram tříd domény

Všimněme si, že názvy atributů a metod jsou s diakritikou a mezerami. Při modelování vytváříme pouze model a nikoliv přímo aplikaci (i když pomocí různých nástrojů lze z modelu kostru aplikace generovat). Můžeme tedy více využít přirozený jazyk, což může být přehlednější. Pokud bychom chtěli z diagramu přímo generovat kostru aplikace nebo jej použít pro specifikaci rozhraní pro vývojáře, museli bychom využít názvy tříd a metod, které budou uvedeny v implementaci.

Uvedený diagram tříd z prostorových důvodů obsahuje pouze (pro zjednodušení vybrané) objekty domény a jedná se tedy pouze o část aplikace s objekty ukládanými do databáze (část 1 balíčku, viz níže). Aplikace však pro svůj chod potřebuje i řadu dalších objektů. Při využití MVC (viz výše) budou potřeba ještě kontroléry (správa objektů a kontroly integrity) a objekty pohledu, které se starají o zobrazení jednotlivých obrazovek aplikace či jejich částí. Bude také potřeba objekt, který bude pracovat s perzistentním úložištěm dat (databází). Velmi malá část diagramu tříd celé aplikace, který je značně rozsáhlý (desítky tříd), je uvedena na obrázku 11.



Obrázek 11: Zjednodušený diagram tříd aplikace

Při návrhu diagramu tříd bychom měli nejprve reprezentovat objekty reálného světa (viz obrázek 10) a následně doplnit další objekty potřebné pro provoz systému. V takto vytvořeném diagramu je vhodné následně vyhledat třídy, u kterých je možné využít generalizaci, čímž můžeme dosáhnout vyšší úrovně abstrakce pro některé operace a následně i lepší modifikovatelnosti systému (dodatečné přidávání podtypů). Při využití generalizace bychom měli zvážit, zda má obecnější třída smysl jako samostatná třída (zda by objekt této třídy reprezentoval něco smysluplného), nebo zda bude tato třída pouze abstraktní. Pokud potřebujeme, aby do vztahu mohly vstoupit objekty různých jinak nesouvisejících tříd, využijeme rozhraní (nemá smysl abstrahovat na příliš vysokou úroveň).

6.2.2 Diagram balíčků

Diagram balíčků umožňuje strukturování statické struktury tříd do balíčků. Zobrazuje balíčky, vztahy a závislosti mezi nimi. Jeho základními prvky jsou:

- balíček,
- závislost,
- generalizace.

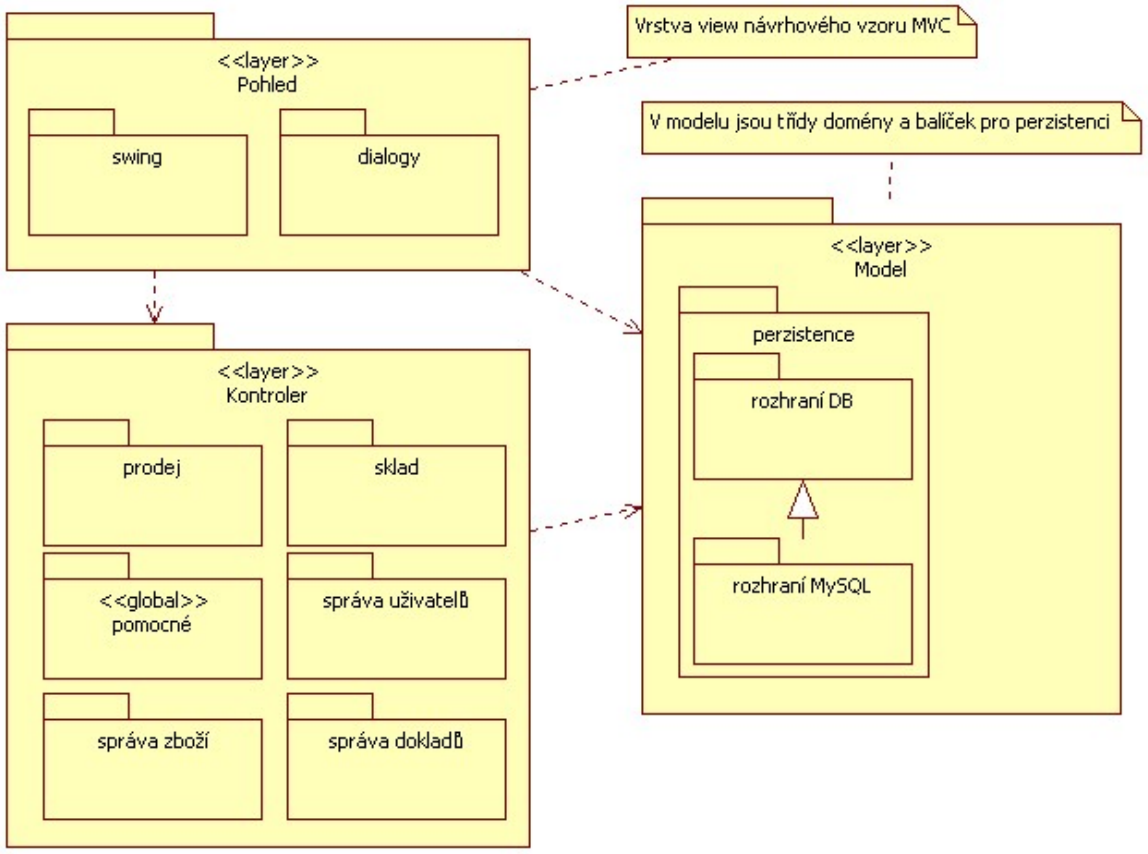
Balíčky můžeme i vnořovat a může mezi nimi být i speciální balíček označený stereotypem `<<global>>`, který obsahuje globální objekty přístupné z celého systému. Na tomto balíčku jsou závislé všechny ostatní, přičemž tyto závislosti se explicitně nezobrazují.

Pro dosažení udržovatelnosti systému a případné znovupoužitelnosti komponent bychom měli minimalizovat závislosti mezi balíčky a zcela odstranit cyklické závislosti (např. s využitím rozhraní).

Pokud využijeme generalizaci, obecnější balíček je automaticky abstraktní (např. balíček rozhraní k databázi je generalizací rozhraní k MySQL a rozhraní k PostgreSQL). Obsahuje tedy především abstraktní třídy a rozhraní.

Další prvky diagramu jako např. omezení se využívají pouze velmi zřídka a nebudu je zde uvádět. Více informací lze nalézt v [8].

Příklad diagramu balíčků je na obrázku 12.



Obrázek 12: Diagram balíčků

Pro vyjádření příslušnosti tříd do balíčků můžeme využít diagram balíčků kombinovaný s diagramem tříd. V tomto diagramu vyobrazíme pouze balíčky (bez vztahů) a do nich vložíme třídy včetně všech závislostí (jedná se spíše o diagram tříd s grafickým seskupením tříd do balíčků).

6.3 Behaviorální pohled

Behaviorální pohled se zabývá vnitřním chováním IT systému. Základním diagramem tohoto pohledu je stavový diagram.

6.3.1 Stavový diagram

Stavový diagram využíváme pro popis chování objektu v systému. Pro každý objekt v systému, jehož chování není zcela zřejmé, je nutné vytvořit stavový diagram. Tento diagram je vhodný i pro popis sekvence obrazovek v případu použití, na kterém se podílí více objektů (viz výše).

Stavový diagram vychází z deterministického konečného automatu. Každý objekt se nachází v nějakém stavu, přičemž jsou definovány podmínky, za kterých může přejít do jiného stavu. V každé situaci musí být možný maximálně jeden přechod (determinističnost).

Základními prvky stavového diagramu jsou:

- počáteční stav,
- stav,
- složený stav,
- přechod,
- koncový stav.

Počáteční stav není normálním stavem, protože v tomto uzlu objekt dosud neexistuje. Rovněž koncový stav je speciální, protože objekt v něm již neexistuje.

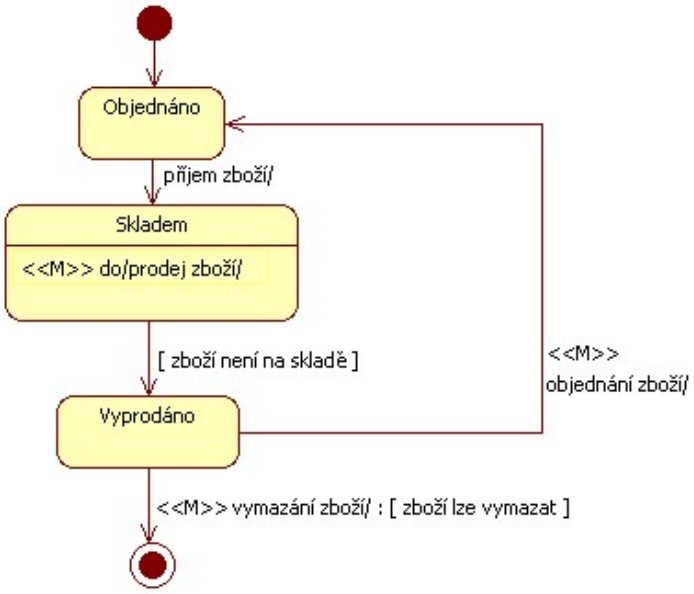
Stav je určen množinou atributů a asociací. Ve stavovém diagramu reprezentuje množinu kombinací těchto hodnot, při které se objekt chová stejně v reakcích na události [1]. V každém stavu objekt provádí nějakou činnost (pasivní čekání je rovněž činnost).

Součástí stavu mohou být i interní přechody, při kterých objekt provede nějakou akci, ale zůstane ve stejném stavu (po provedeném přechodu bude opět provádět stejnou činnost). Mezi interními přechody mohou být i speciální přechody, které se vykonají ihned po vstupu do stavu (entry), opakovaně, pokud nelze udělat jiný přechod (aktivita prováděná v daném stavu), či těsně před jeho opuštěním (exit). Typ speciálního interního přechodu označujeme klíčovým slovem před lomítkem (`entry`, `do` či `exit`).

Složený stav představuje vnoření jiného stavového diagramu.

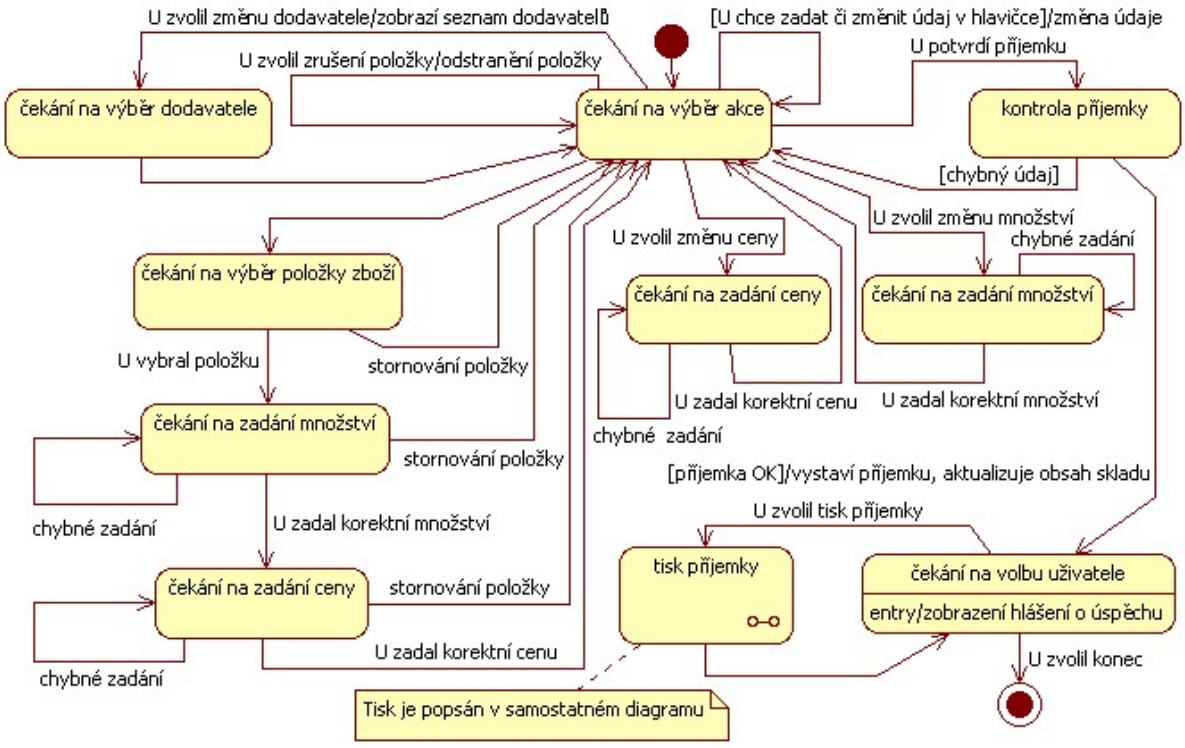
Každý přechod (interní i mezi stavy) může mít název události, podmínku a akci, která bude provedena při přechodu. Jako první se uvádí název události, za ním následuje podmínka v hranatých závorkách, lomítko a akce. Pokud je uveden název události či podmínka, lomítko uvádíme vždy (pro odlišení události od akce). V [1] je doporučeno všechny události (přechody) označit výše uvedenými stereotypy `<<M>>` a `<<Q>>`.

Příklad stavového diagramu pro objekt třídy Zboží je na obrázku 13.



Obrázek 13: Stavový diagram pro objekt třídy Zboží

Příklad stavového diagramu pro popis sekvence obrazovek v případu použití příjem zboží je na obrázku 14.



Obrázek 14: Stavový diagram pro popis sekvence obrazovek při příjmu zboží

6.4 Interakční pohled

Interakční pohled je interním pohledem na interakce objektů uvnitř systému. Tyto interakce jsou obvykle reakcí systému na nějakou vnější událost, která je součástí případu použití.

Diagramy v tomto pohledu obecně označujeme jako diagramy interakce, mezi které patří [9]:

- diagram sekvence (sequential),
- diagram komunikace (communication),
- přehledový diagram interakce (interaction overview),
- diagram časování (timing).

Základními a současně nejčastěji využívanými jsou diagram sekvence a diagram komunikace. V [1] je doporučeno pro dotazovací události využívat diagram sekvence a pro události změny využívat diagram komunikace. Toto pravidlo však není vždy nutné dodržet a často může být výhodné popsat událost změny diagramem komunikace (možné zobrazit, které atributy objektů budou modifikovány) či popsat složitější reakci na dotazovací událost pomocí diagramu sekvence (obvykle při získávání malého množství hodnot, které se získávají složitějšími výpočty s využitím řady objektů).

Přehledový diagram interakce a diagram časování lze využít pro lepší vysvětlení složitějších reakcí na události či některých speciálních případů, ve kterých je jejich využití výhodné.

V systému je obvykle značné množství událostí. Jsou zde desítky případů použití a v rámci každého z nich se provádí řada interakcí tvořících samostatné reakce na události. Pokud bychom měli pro každou reakci na událost sestrojit diagram, model by byl značně složitý a jen obtížně udržovatelný (byl by velmi detailní). Na modelování však obvykle máme málo času a omezený rozpočet. Je tedy vhodné vytvářet pouze diagramy interakce pro složitější či důležité reakce na události.

6.4.1 Diagram sekvence

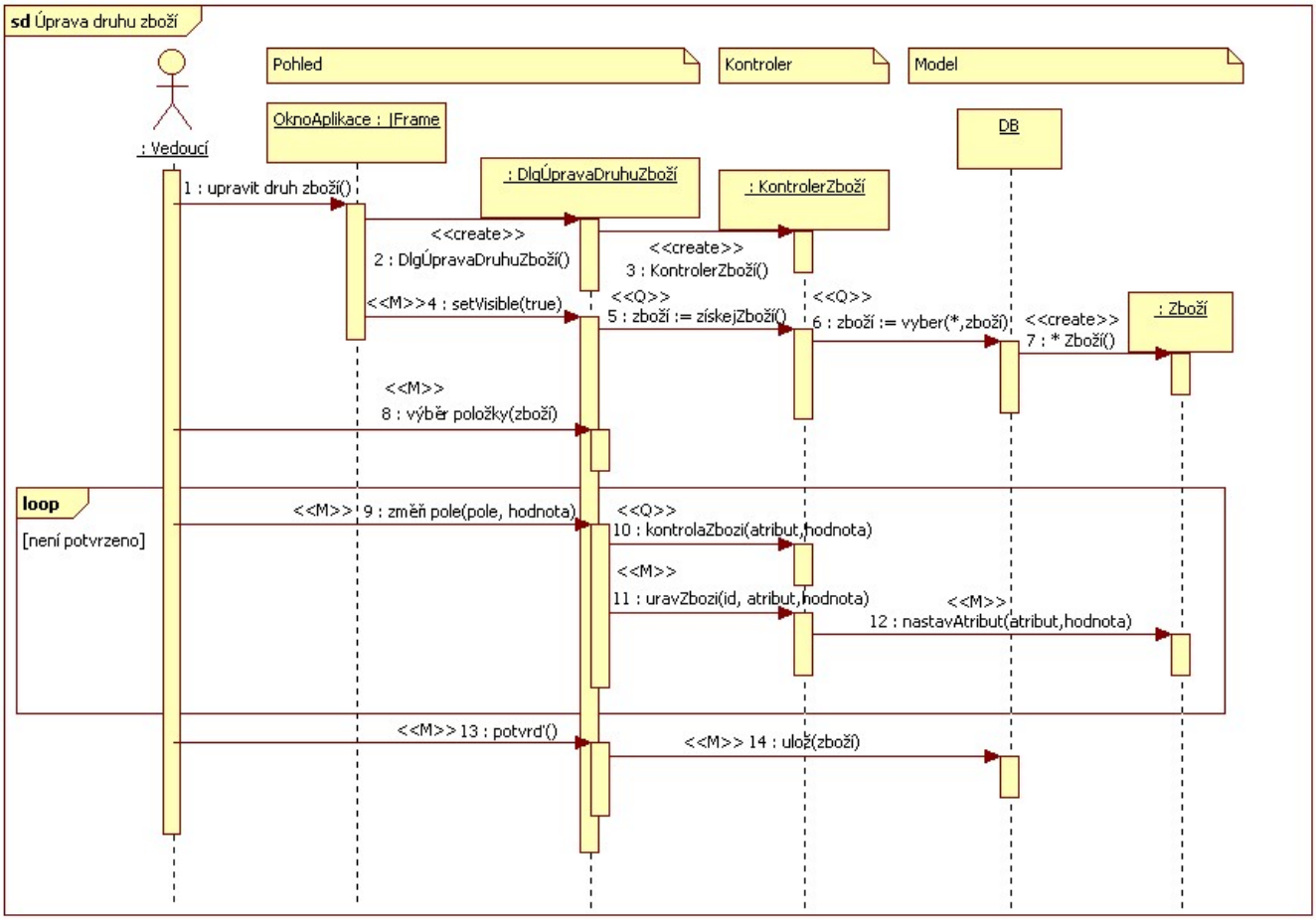
Diagram sekvence umožňuje zobrazit sekvence interakcí mezi objekty v systému. Prvky diagramu jsou stejné jako u diagramu sekvence v externím pohledu v modelu obchodního systému, ale místo celého systému jsou v něm uvedeny jeho jednotlivé objekty.

Diagram obvykle vyobrazuje sekvenci interakcí, která nastane v reakci na nějakou vnější událost, která je součástí případu použití. Zatímco v externím pohledu jsme sledovali celou sekvenci interakcí s aktérem, zde obvykle zobrazujeme pouze to, co nastane v reakci na jednu zprávu (událost) od aktéra. Aktér proto v diagramu nemusí být uveden, ale může zde být vyobrazena pouze daná událost (šipka z vnějšku diagramu). Dle [1] je však doporučeno pro lepší pochopení diagramu aktéra uvádět.

Oproti externímu pohledu se v tomto diagramu častěji využívají složitější konstrukce, protože tento diagram je určen především pro vývojáře a měl by dostatečně detailně popsat, jaké zprávy a v jakých situacích mají být zasílány mezi objekty.

Přibyla zde navíc možnost komunikace objektu s kolekcí objektů (např. objednávka zašle zprávu svým položkám). Pokud je zpráva zasílána mezi objektem a kolekcí objektů, může být zaslána konkrétnímu objektu (selekce) nebo všem objektům v kolekci (iterace). Selekce je implicitní, iteraci vyznačujeme hvězdičkou u zprávy. Pokud je při komunikaci s určitými objekty v daném diagramu vždy využita iterace, můžeme místo hvězdičky u jednotlivých zpráv využít stereotyp `<<every>>` u objektu.

Příklad diagramu sekvence je na obrázku 15.



Obrázek 15: Diagram sekvence pro úpravu druhu zboží

6.4.2 Diagram komunikace

Diagram komunikace rovněž zobrazuje interakce mezi objekty při reakci na událost. Oproti diagramu sekvence má však rozdílné vyjadřovací schopnosti, což umožňuje zdůraznit jiné aspekty interakce. Vše, co lze vyjádřit diagramem sekvence, lze vyjádřit i diagramem komunikace, ale ne naopak.

Diagram komunikace má určitou podobnost se statickým diagramem tříd. Mohou v něm být uvedeny názvy tříd i jejich atributy, což při vyobrazení získávání informací umožňuje zobrazit, které atributy objektu budou čteny (v diagramu sekvence nelze). Narozdíl od diagramu sekvence však nemá časovou osu a sekvenci vykonávání událostí lze vyjádřit pouze pomocí jejich číslování.

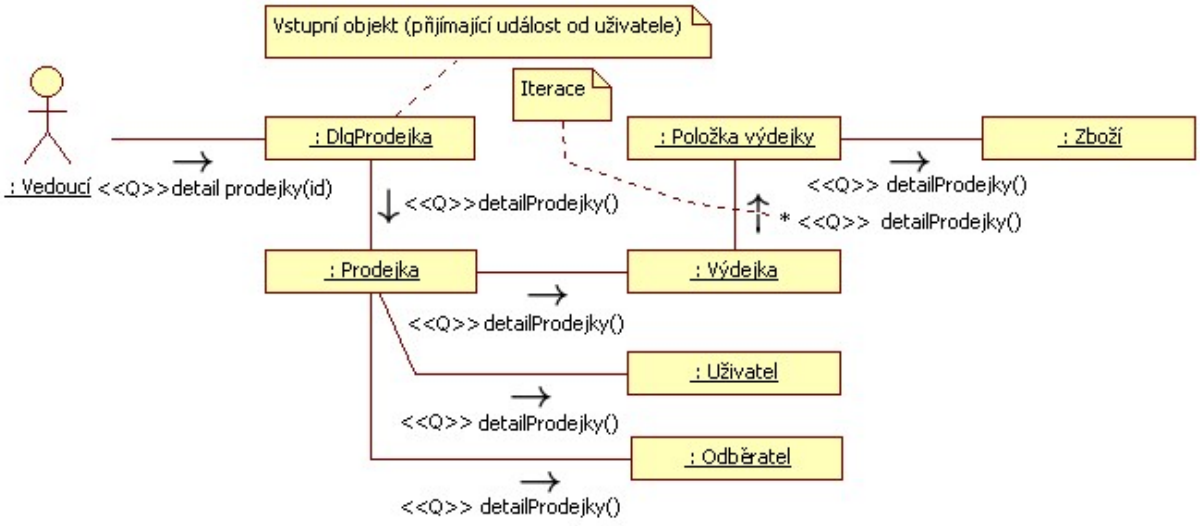
Základními prvky diagramu jsou:

- aktér,
- objekt,
- událost (zpráva).

Aktér může být z případu použití nebo „Někdo“. Objekty jsou zobrazovány obdobně jako ve statickém diagramu objektů. Každý objekt tedy může mít uveden název, třídu a atributy.

Událost má vyznačený směr a může mít parametry. Pokud se jedná o vztah mezi objektem a kolekcí objektů, obdobně jako u diagramu sekvence můžeme využít selekci či iteraci (hvězdička u události).

Při konstrukci diagramu komunikace vycházíme z diagramu tříd, přičemž si vyznačujeme, které třídy budou v reakci na událost zahrnuty. Nejprve označíme třídy, jejichž atributy potřebujeme, a potom označíme třídy, které potřebujeme pro komunikaci (propojení vyznačených tříd). Následně z aktéra a označených tříd sestrojíme diagram a vyznačíme v něm propagaci události. Příklad vytvořeného diagramu komunikace je na obrázku 16.



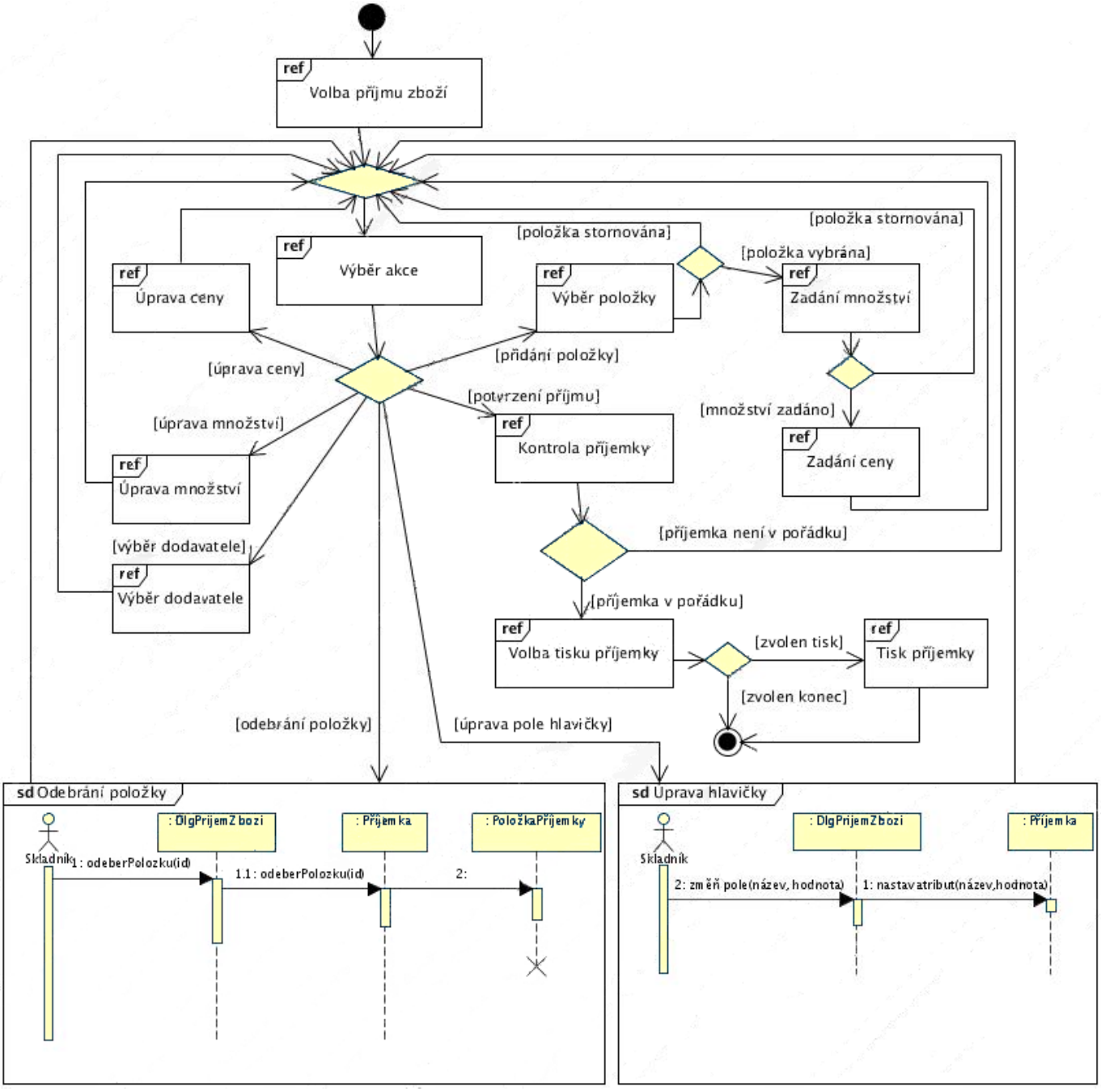
Obrázek 16: Diagram komunikace pro zobrazení prodejky

6.4.3 Přehledový diagram interakce

Přehledový diagram interakce je kombinací diagramu aktivity a diagramů sekvence. Základem je syntaxe diagramu aktivity, ve kterém vybrané uzly akcí nahradíme odkazy na diagramy sekvence či komunikace nebo přímo vloženými diagramy.

Výhodou je možnost snadného vyjádření většího množství podmínek, cyklů, paralelismů a dalších konstrukcí se současným zobrazením důležitých interakcí. Pokud bychom v takové situaci využili pouze diagram sekvence, byl by značně nepřehledný. Pokud využijeme sadu diagramů se vzájemnými odkazy nebo podrobnější členění aktivity, vytvořená sada diagramů bude rovněž nepřehledná (ztrácí se souvislosti). Přehledový diagram interakce tedy poskytuje přehled v situaci, kdy je aktivita rozčleněna do více sekvenčních diagramů.

Příklad diagramu je na obrázku 17.



Obrázek 17: Přehledový diagram interakce pro příjem zboží

6.4.4 Diagram časování

Jazyk UML není primárně určen pro modelování systémů pracujících v reálném čase (real-time). Je zde sice dostatek syntaxe, chybí však potřebná sémantika, z čehož plynou určitá omezení. Pro modelování těchto systémů lze využít zvláštní profil UML pro plánovatelnost, výkonnost a čas. Tento profil zahrnuje řadu stereotypů, omezení a označených hodnot [7].

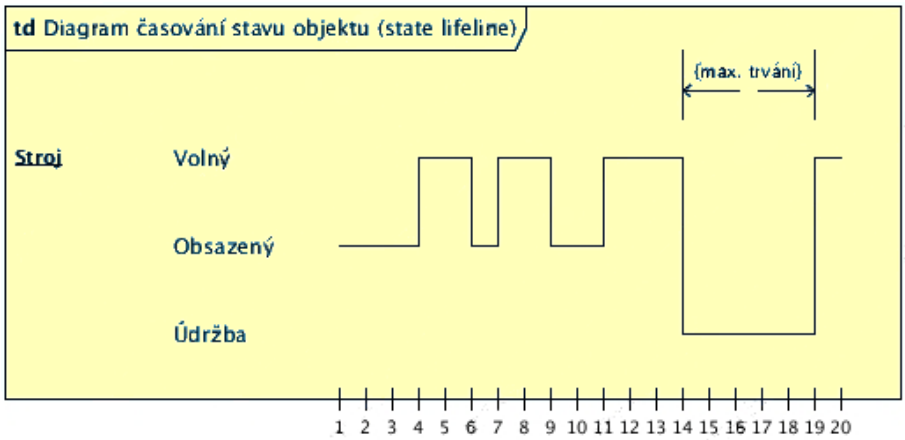
Diagram sekvence sice umožňuje zobrazit časový průběh, jeho časová osa však není lineární. Vzdálenosti zasílání zpráv jsou dány tím, jak zrovna diagram nakreslíme, přičemž dbáme na to, aby byl přehledný. Pokud je mezi dvěma objekty zasláno hodně zpráv v krátkém čase a následně dvě zprávy s delším časovým odstupem, v diagramu sekvence budou vzdálenosti dány tím, co po každé zprávě následuje (zda na jejich základě přijímající objekt posílá další zprávy třetímu objektu), případně budou mezi všemi zprávami přibližně stejné vzdálenosti. Lze tedy pouze určit, která zpráva

následuje po které a to navíc pouze na jedné čáře života objektu (lifeline). Na jiné čáře může být časování jiné. Máme tedy dáno pouze částečné uspořádání.

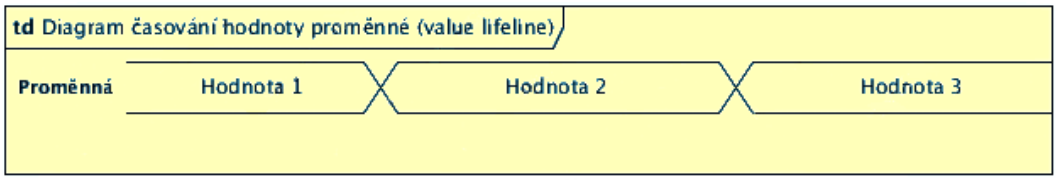
Diagram časování byl zaveden v UML 2.0 a slouží k popisu časování v nějaké situaci (stereotyp `<<SAsituation>>`). Osa x je lineární (či jiná) časová osa a na ose y jsou jednotlivé objekty, úlohy či zdroje a jejich stavy. Základními prvky diagramu jsou časový rámec (ohraničení diagramu) a čára stavu (status lifeline), která zobrazuje, v jakém je objekt (zdroj, úloha apod.) stavu. U zdrojů s hodnotou (proměnných) lze na místě čáry stavu využít čáru hodnoty (value lifeline) umožňující zobrazení změny hodnoty v čase [6]. Využívá se zde také celá řada stereotypů, jejichž vysvětlení je mimo rozsah této práce, Více informací lze nalézt v [7].

Diagram časování je možné využít pro zobrazení plánování činností objektů v čase. Lze zde uvést pořadí zasílání zpráv s úplným uspořádáním, vytížení objektu v čase apod.

Vzhledem k tomu, že příklad informačního systému v této práci nemusí pracovat v reálném čase (bylo by to složité a nemělo by to smysl) a vzhledem k tomu, že plánování času a modelování systémů pracujících v reálném čase je mimo rozsah této práce, uvádím zde diagram časování pouze pro úplnost. Diagramy na obrázcích 18 a 19. slouží pouze pro ilustraci (nepatří k příkladu).



Obrázek 18: Diagram časování stavu objektu



Obrázek 19: Diagram časování hodnoty proměnné

6.5 Pohled nasazení

Pohled nasazení se zabývá nasazením software na konkrétní hardware ve firmě. Jedná se tedy o pohled na celkovou architekturu systému.

6.5.1 Diagram nasazení

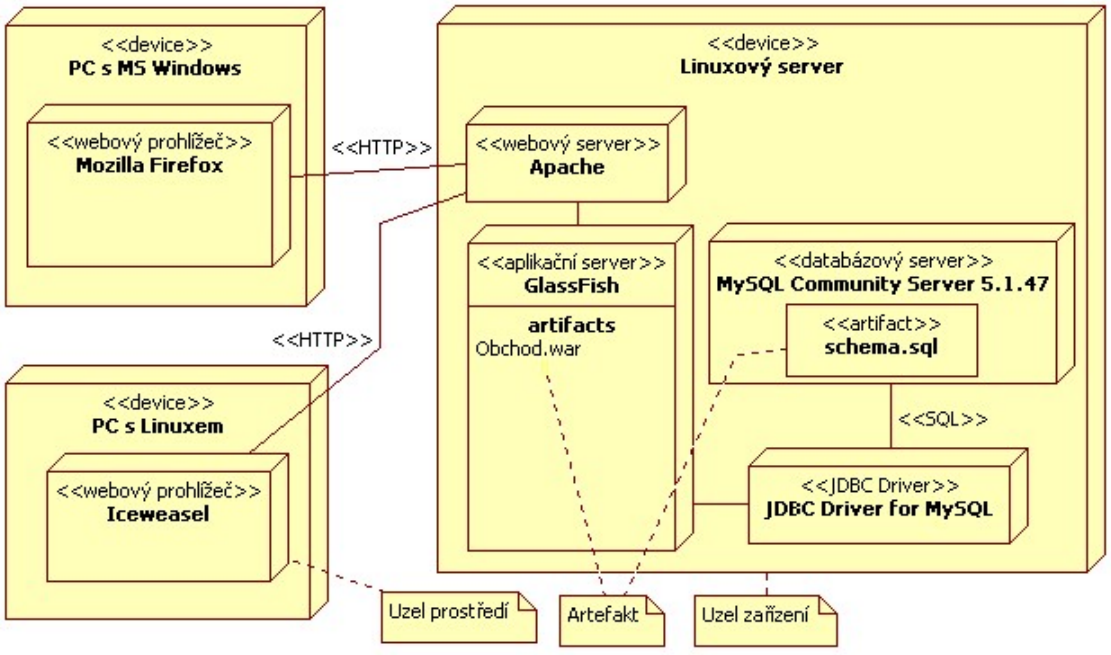
Diagram nasazení zobrazuje nasazení softwarových prvků na fyzickou architekturu (hardware) a komunikaci mezi fyzickými prvky [9]. Jeho základními prvky jsou:

- uzel zařízení (device node),
- uzel prostředí (execution environment node),
- komunikační linka (link),
- artefakt (artifact).

Uzel zařízení reprezentuje konkrétní hardwarové zařízení (PC, přístupový terminál, ...). Uzel prostředí reprezentuje prostředí pro běh součásti systému, jako např. webserver, webový prohlížeč, virtuální stroj jazyka Java apod. Artefakt je konkrétní část systému (soubor, např. .war balíček).

Komunikační linka zobrazuje komunikační linku mezi hardwarovými zařízeními. Může být označena stereotypem s komunikačním protokolem (např `<<HTTP>>`, `<<SQL>>`, apod.).

Příklad diagramu nasazení je na obrázku 20.



Obrázek 20: Diagram nasazení

Dalšími prvky diagramu jsou [8]:

- komponent,
- rozhraní,
- port,
- omezení,
- asociace,
- agregace,
- kompozice,
- ...

Uvedené prvky slouží ke tvorbě složitějších diagramů nasazení. U složitých systémů můžeme namísto artefaktů využít komponenty (modulární části systému s definovaným rozhraním), můžeme vyobrazit vztahy mezi součástmi systému na jednom zařízení apod. Více informací lze nalézt v [8].

6.6 Další diagramy

V UML existují i další diagramy, které lze využít k detailnějšímu vysvětlení některých částí systému v rámci některého z výše uvedených pohledů (např. strukturálního pohledu, interakčního pohledu apod.) nebo jako součásti jiných pohledů, které v této práci nejsou uvedeny (např. ze sady pohledů nazvané „4+1 pohled na architekturu“).

6.6.1 Diagram objektů

Diagram objektů zobrazuje objekty v systému v určitém časovém okamžiku jeho provozu. Je vhodný pro vysvětlení složitějších struktur v diagramu tříd. Jeho základními prvky jsou:

- objekt,
- vazba (link),
- agregace,
- kompozice.

Asociace deklaruje, že mezi instancemi daných typů může či musí být vztah, zatímco vazba popisuje konkrétní vztah mezi propojenými objekty. V diagramu se vazba značí stejně jako asociace (mohou v něm být oba prvky, viz [8]) a odlišuje se tím, zda jsou propojeny objekty či třídy.

Příklad diagramu objektů je na obrázku 21.

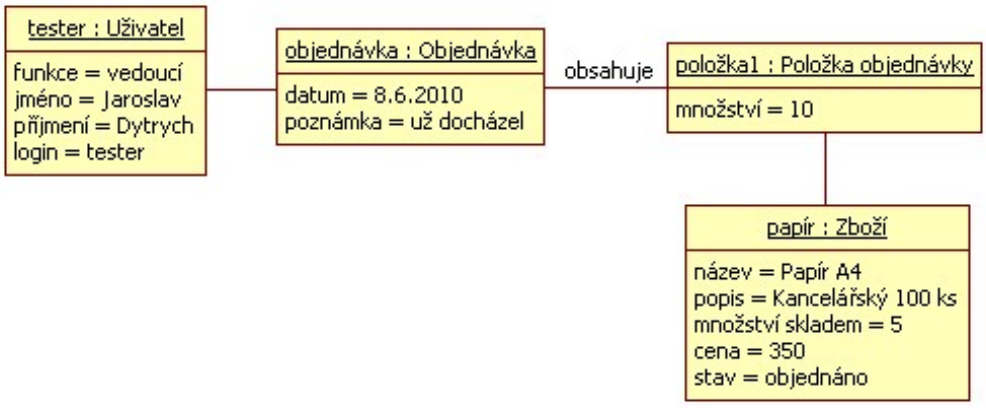
Další prvky diagramu lze nalézt v [8], ale tyto jsou méně typické.

6.6.2 Diagram složené struktury

Diagram složené struktury byl zaveden v UML 2.0 a zobrazuje sestavu instancí objektů existujících za běhu programu a spolupracujících na dosažení určitého cíle. Má několik notací a využívají se v něm různé sady prvků podle účelu (více viz [2]). V jedné z jeho notací je zde určitá podobnost s diagramem komunikace, který také zobrazuje objekty spolupracující na jednom úkolu, ale narozdíl od něj zde nejsou vyobrazeny zprávy, ale vztahy mezi objekty (jak jsou objekty propojeny, aby mohly komunikovat). Diagram složené struktury slouží především ke zobrazení interní struktury (včetně složek a součástí) strukturované kolaborace.

Základními prvky diagramu jsou:

- složka (part),
- třída,
- port,
- konektor,



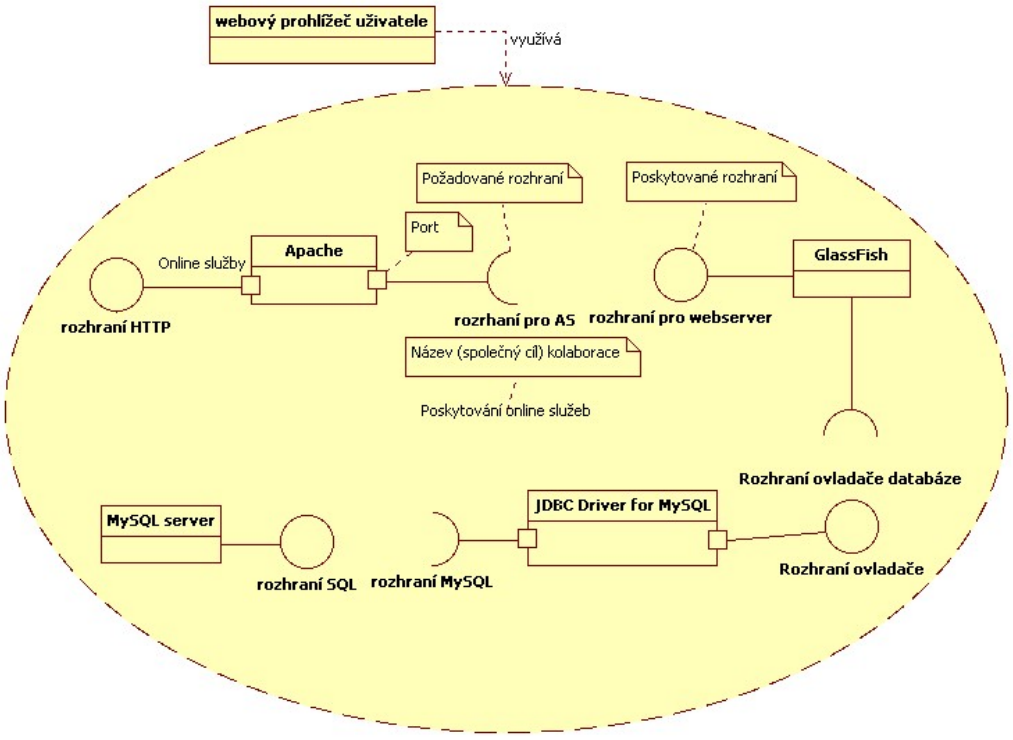
Obrázek 21: Diagram objektů

- rozhraní,
- kolaborace
- využití kolaborace.

Složka je sada instancí vlastněných instancí obsahujícího klasifikátoru (třídy) [8].

Konektor reprezentuje spojení mezi dvěma nebo více objekty. Toto spojení může být instancí asociace nebo může být vytvořené jiným způsobem jako např. předáním identifikátoru objektu v parametru funkce.

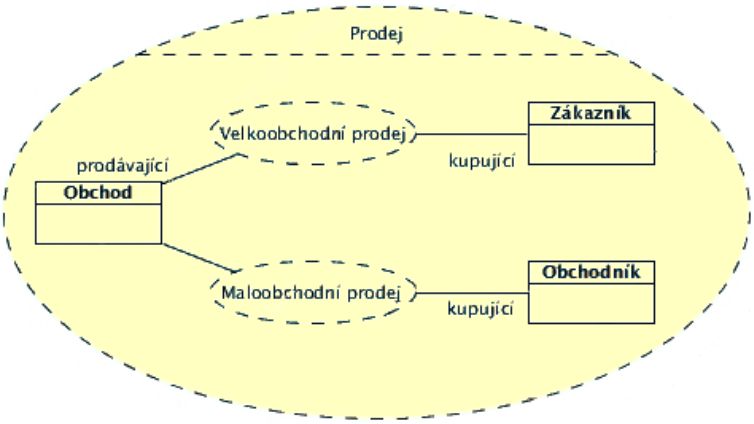
Další prvky diagramu lze nalézt v [8], více informací lze nalézt v [2]. Příklad diagramu je na obrázku 22.



Obrázek 22: Diagram složené struktury

6.6.3 Diagram kolaborace

Diagram kolaborace zobrazuje strukturu spolupracujících prvků, z nichž každý provádí nějakou specializovanou funkci, které ale společně vytvářejí nějakou požadovanou funkcionalitu [9]. Diagram kolaborace pochází z UML 1.0 a v UML 2.0 byl přejmenován na diagram komunikace. Někdy lze pod tímto názvem nalézt i určitou variantu diagramu složené struktury, jehož ukázka je na obrázku 23.



Obrázek 23: Diagram složené struktury ve variantě pro kolaboraci

6.6.4 Diagram komponent

Diagram komponent zobrazuje závislosti mezi softwarovými komponenty (včetně jejich specifikace a implementujících artefaktů). Jeho základními prvky jsou:

- komponent,
- poskytované rozhraní,
- požadované rozhraní,
- port,
- závislost (např. delegace).

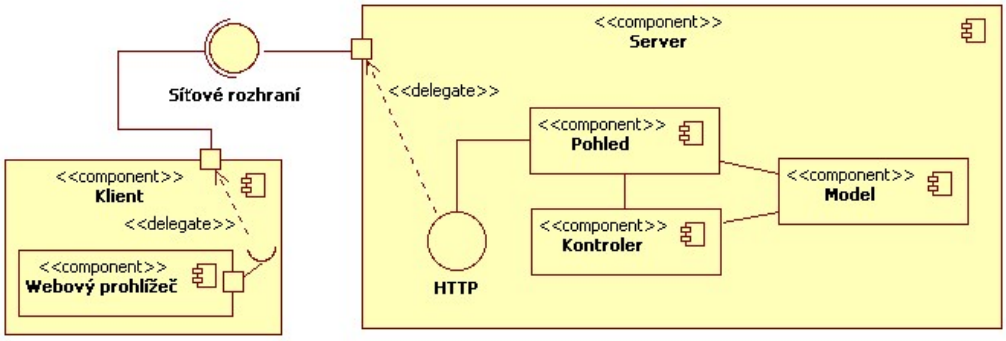
Komponent reprezentuje modulární část systému, která zapouzdřuje svůj obsah. Komponent lze nahradit jiným komponentem se stejným rozhraním a chováním vůči svému okolí.

Poskytované rozhraní je rozhraní, přes které komponent nabízí služby (veřejné metody komponentu, které lze volat). Požadovaná rozhraní slouží k napojení na jiné komponenty, které danému komponentu poskytnou služby, které potřebuje ke své činnosti.

V diagramu je možné modelovat i vnitřní strukturu komponentů (složení jednoho komponentu z jiných). V tomto případě využijeme porty, které oddělují vnitřní rozhraní komponentu od vnějších.

Delegace lze využít např. k delegování vnitřního rozhraní na port (pokud je vnitřní rozhraní stejné jako vnější). V diagramu lze využít i další prvky, jejichž využití je však méně časté. Tyto prvky lze nalézt v [8].

Příklad diagramu komponent je na obrázku 24.



Obrázek 24: Diagram komponent

7 Závěr

V této práci jsou popsány dva základní modely v UML 2.0 - model obchodního systému a model IT systému. Jsou zde uvedeny základní konstrukce všech běžných diagramů UML kromě ER diagramu, který je mimo základní sadu diagramů a využívá se obvykle pro modelování struktury relační databáze (při objektovém přístupu nahradíme diagramem tříd).

Uvedené prvky a konstrukce diagramů jsou postačující i pro modelování středních a větších systémů a přesahují i rozsah, který je postačující podle [1]. Jedná se však stále o malou podmnožinu všech prvků a konstrukcí, které jsou obsaženy ve specifikaci UML. Specifikace v celé šíři, tak jak je uvedena v [4] a [5], je však téměř nevyužitelná, protože si ji většina vývojářů nenastuduje či nezapamatuje. U diagramů využívaných pro komunikaci se zákazníkem je využití složitých konstrukcí, kterým běžný uživatel nemůže porozumět, přímo nežádoucí.

Tato práce je zaměřena především na vývoj webových informačních systémů, ale uvedené konstrukce jsou stejné i pro klasický desktopový software. Pro modelování systémů pracujících v reálném čase a další speciální případy a využití je třeba využít jiné sady pohledů, prvky a konstrukce diagramů, stereotypy a další součásti UML. Pro některé speciální případy jsou v UML definovány profily obsahující potřebné prvky, v jiných případech je volba na návrháři, který musí zvolit vhodné konstrukce. Vždy je však třeba dbát na to, aby diagramy byly dobře srozumitelné pro všechny osoby, které je budou využívat. Jedině tak mohou splnit jeden z hlavních účelů jejich tvorby - usnadnění spolupráce osob pracujících na projektu.

Literatura

[1] Baumann, H.; Grässle, P.; Baumann, P.: *UML 2.0 in Action: A project-based tutorial*. Birmingham: Packt Publishing, Prosinec 2009, ISBN 1-904811-55-8.

[2] Colombi, L.: Composite Structure Diagram. In *Corso di Ingegneria del Software II Anno 2003/04*, Dipartimento di Informatica e Scienze dell'Informazione Università degli Studi di Genova, 2004, [Online; navštíveno 1. 6. 2010].
URL http://www.disi.unige.it/person/ReggioG/ISII04WWW/Composite.ppt

[3] Holub, A.: Allen Holub's UML Quick Reference. In *HOLUB*, Allen I. Holub & Associates, Srpen 2007, [Online; navštíveno 1. 6. 2010].
URL http://www.holub.com/goodies/uml/

[4] OMG Unified Modeling LanguageTM (OMG UML), Infrastructure. Únor 2009, [Online; navštíveno 1. 6. 2010].
URL http://www.omg.org/cgi-bin/doc?formal/09-02-05

[5] OMG Unified Modeling LanguageTM (OMG UML), Superstructure. Únor 2009, [Online; navštíveno 1. 6. 2010].
URL http://www.omg.org/cgi-bin/doc?formal/09-02-03

[6] Sparks, G.; aj.: UML 2 Timing Diagram. In *UML 2 Tutorial*, Sparx Systems Pty Ltd., 2010, [Online; navštíveno 1. 6. 2010].
URL http://www.sparxsystems.com/resources/uml2_tutorial/uml2_timingdiagram.html

[7] Valiente, M. C.; Genova, G.; Carretero, J.: UML 2.0 Notation for Modeling Real Time Task Scheduling. In *Journal of Object Technology*, ETH Zurich, Chair of Software Engineering, Červen 2006, [Online; navštíveno 1. 6. 2010].
URL http://www.jot.fm/issues/issue_2006_05/article2.pdf

[8] Visual Paradigm Gallery. 2010, [Online; navštíveno 1. 6. 2010].
URL http://www.visual-paradigm.com/VPGallery/diagrams/

[9] Zendulka, J.: Úvod do jazyka UML. 2008, [Online; navštíveno 1. 6. 2010].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/AIS-IT/lectures/2008/3_UML.pdf