Analytické a návrhové vzory a jejich aplikace

Bára Bühnová

25. února 2013

Obsah

Seznam obrázků

Kapitola 1

Analytické vzory

1.1 Analýza podle vzorů

Pro provedení kvalitní analýzy jsou velmi podstatné dlouhodobé zkušenosti, které bývají často předávány v podobě rad z analytika na analytika. Zobecněním těchto zkušeností vznikají právě analytické vzory. Analytický vzor si můžeme představit jako část analytického modelu, kterou je možné použít v podobných aplikačních oblastech. Použití v konkrétním projektu pak předpokládá vytvoření instance vzoru (nalezení účastníků vzoru mezi objekty modelovaného systému).

Analytické vzory pomáhají při modelování obchodní doménové oblasti navrhovaného systému. Pomáhají nám hledat nové upřesňující otázky, pomocí kterých jsme schopni najít mezery ve znalosti doménové oblasti a vytvořit tak celistvý pohled na modelovaný systém. To je pro nás klíčové při specifikaci požadavků na nový systém, protože perfektním pochopením doménové oblasti předcházíme následným nedorozuměním, která jsou velice drahá.

1.1.1 Metody aplikace vzorů

Při aplikaci analytických a návrhových vzorů je možné postupovat několika různými způsoby. Podle [9] lze nejběžnější techniky pro analýzu a návrh podle vzorů rozdělit na následující typy:

- *Nesystematická aplikace vzorů*
  Vzory jsou aplikovány pouze při výskytu určitého problému. Vzor tak poskytuje jednorázové řešení konkrétní situace.
- *Systematická aplikace vzorů*
  Systematické techniky aplikace vzorů definují postup nasazování vzorů. Tyto techniky mohou být dvou typů.
  - *Katalog vzorů (Pattern Language)*
    Katalog vzorů často poskytuje vzory týkající se konkrétní oblasti z obchodní domény (analytické vzory) nebo domény řešení (návrhové vzory). Katalog dále definuje vztahy a souvislosti mezi vzory, případně možnosti jejich kombinace. Vzory jsou většinou aplikovány metodou *dosazení do výsledného modelu*, kterou si představíme níže.

– *Vývojový proces (Development Process)*
Systematický vývojový proces definuje postup pro aplikaci vzorů, jehož podstatou je *skládání vzorů jako základ aplikačního modelu.* Proces definuje nejen formalizovaný postup, ale i modely a nástroje, které pomáhají k jeho automatizaci.

Dosazení vzorů do výsledného modelu
Při dosazování vzorů do výsledného modelu nejprve pomocí standardních prostředků vytvoříme analytický nebo návrhový model. Poté vybereme adepty vzorů, které by mohlo být dobré použít. V případě analytických vzorů to mohou být vhodné soubory vzorů podle doménové oblasti. Následuje systematické procházení adeptů a hledání vhodných míst pro jejich nasazení (hledání problému, který konkrétní vzor řeší). Zde se často dostaneme do situace, kdy je vzor bohatší než zkoumaná část systému a nasazení vzoru tak přináší možné rozšíření systému o vlastnosti zvyšující jeho flexibilitu a znovupoužitelnost. Po nalezení vhodných vzorů tyto vzory zasadíme do aplikačního modelu. To probíhá na základě vyhledání účastníků vzoru a tím vytvoření instance vzoru.

Skládání vzorů jako základ modelu
Zmíněný postup definuje metodika *POAD (Pattern Oriented Analysis and Design)* [9], která navrhuje rozdělit analýzu a návrh podle vzorů do tří etap:

- *Analytická fáze*
  Během analytické fáze dochází na základě analýzy požadavků k výběru vhodných vzorů pro konkrétní modelovanou oblast.
- *Návrh na vyšší úrovni*
  V rámci návrhu na vyšší úrovni dochází k vzájemnému propojení instancí vzorů, které byly vybrány v předchozí fázi. Spojování vzorů probíhá podle formalizovaného postupu s podporou diagramů pro realizaci propojení vzorů.
- *Zpřesnění návrhu*
  Fáze zpřesnění návrhu rozvíjí hrubý aplikační model, který je výstupem předchozí fáze. Do tohoto modelu jsou doplňovány detaily, které nebyly pomocí vzorů pokryty. Dochází také k optimalizaci vazeb.

1.2 Analytické vzory (Fowler)

Níže diskutované analytické vzory byly představeny Martinem Fowlerem v [4]. Vzory jsou děleny na dvě části. První část obsahuje 65 analytických vzorů a druhá část 21 pomocných vzorů, které mohou být využity při nasazování analytických vzorů. Zde se budeme věnovat vzorům první části. Tyto vzory jsou rozděleny do 9 souborů. Každý soubor se týká určité doménové oblasti (například modelování organizační struktury). Vzory každého souboru jsou představovány formou evoluční řady. Tato řada začíná zavedením jednoduchých vzorů, které jsou dále rozvíjeny a kombinovány, čímž vznikají poměrně komplexní vzory. Výčet souborů analytických vzorů je následující:

- Accountability
- Observations and Measurements

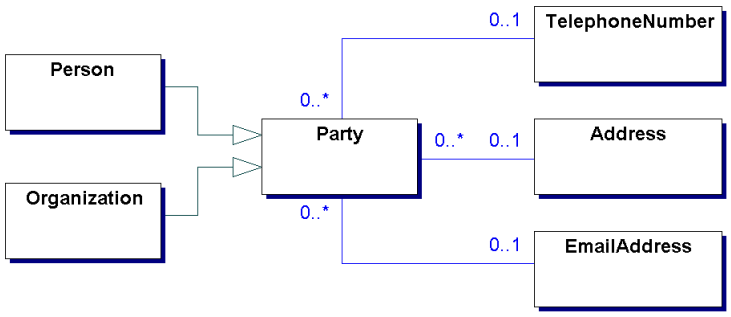
- Observations for Corporate Finance
- Referring to Objects
- Inventory and Accounting
- Planning
- Trading
- Derivative Contracts
- Trading Packages

V následujícím textu se u každého souboru seznámíme s jedním nebo dvěma vzory. Pro větší názornost je použití většiny z těchto vzorů ukázáno na ilustrační aplikaci v kapitole 3.2.

1.2.1 Accountability

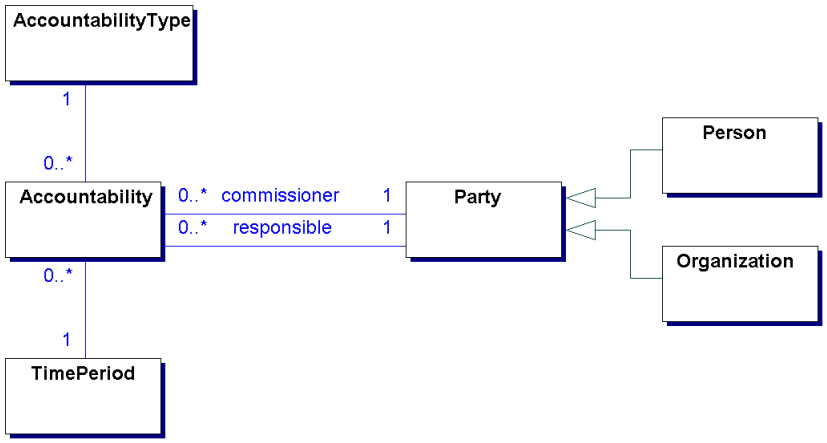
Vzory sady *Accountability* se používají při modelování zodpovědností mezi osobami či organizacemi. Tyto zodpovědnosti mohou být pojaty velmi široce a lze pomocí nich vyjádřit mnoho různých vztahů reprezentujících například organizační strukturu společnosti nebo hierarchii nadřízenosti mezi zaměstnanci. Vzory sady *Accountability* nalézají vhodné uplatnění v podnikových informačních systémech.

Název vzoru: Party
Číselná identifikace: 2.1
Vzor *Party* definuje společný nadtyp osoby a organizace pro případy, kdy v modelu vystupují v rovnocenné pozici a účastní se stejných operací. Jako příklad je často uváděn telefonní seznam. Položkou telefonního seznamu je záznam o osobě či organizaci. S těmito položkami provádím stejné operace (zatelefonovat, smazat, přesunout) nezávisle na typu položky, proto je vhodné odkazovat na ně jednotně přes jejich nadtyp *Party*. Struktura vzoru je na obrázku 1.1.



Obrázek 1.1: Analytický vzor Party

Název vzoru: Accountability
Číselná identifikace: 2.4
Vzor *Accountability* pomáhá definovat zodpovědnost určitého typu mezi dvěma účastníky (objekty *Party*) po stanovený časový interval. Poskládáním těchto dvojic zodpovědností pak vzniká celistvý hierarchický model. Zodpovědnost je založena na vztahu *Commissioner–Responsible*, za kterou můžeme dosadit například vztah *Nadřízený–Podřízený* a tím díky vzoru *Accountability* vybudovat zaměstnaneckou hierarchii v organizaci. Struktura vzoru je na obrázku 1.2.



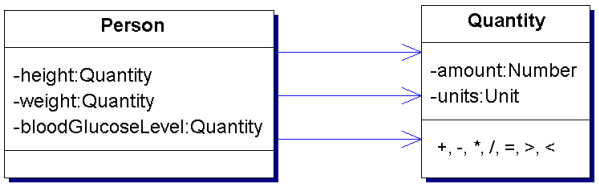
Obrázek 1.2: Analytický vzor Accountability

1.2.2 Observations and Measurements

Při modelování informačních systémů se často setkáváme s objekty, u nichž je třeba uchovávat mnoho kvantitativních a kvalitativních informací. Příkladem může být pacient navštěvující lékařskou ordinaci. Lékař při každém vyšetření potřebuje uchovat o pacientovi několik různých údajů jako je výška, váha, kvalita zraku. Tyto údaje by jistě bylo možné zaznamenat ve formě atributů objektu *Pacient*. Existují však sofistikovanější možnosti řešení, které přinášejí vzory sady *Observations and Measurements*.

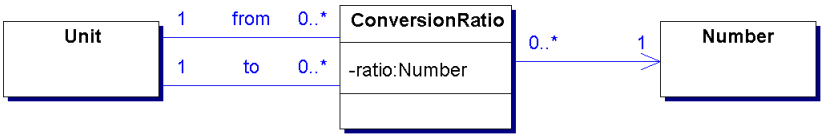
Název vzoru: Quantity
Číselná identifikace: 3.1
Nejčastější formou uchovávání kvantitativních informací je pomocí číselné hodnoty, pro níž se jednotky považují za zřejmé. Zaznamenáme-li u váhy pacienta číslo 70, neuvádíme již, že se jedná o váhu v kilogramech. To však není příliš univerzální přístup bez ohledu na to, jak často je využíván. Mnohem přesnější by bylo vyjadřovat kvantitativní či kvalitativní informace pomocí dvojice *hodnota–jednotky*. Takový přístup nabízí vzor *Quantity*, který navíc definuje povolené operace mezi hodnotami uchovávanými ve stejných jednotkách. Struktura vzoru je na obrázku 1.3.

Název vzoru: Conversion Ratio
Číselná identifikace: 3.2



Obrázek 1.3: Analytický vzor Quantity

Pokud používáme u určité vlastnosti více jednotek, budeme chtít nadefinovat funkci převodu mezi těmito jednotkami. Mimo jiné i proto, že díky tomu budeme moci používat operace určené vzorem *Quantity* i na hodnoty zapsané v různých jednotkách. Vzor *Conversion Ratio* definuje převodovou tabulku mezi jednotkami, díky níž jsme schopni kdykoli zjistit, zda mezi jednotkami lze převádět a jakou použít *multiplikační konstantu (ratio)*. Struktura vzoru je na obrázku 1.4.



Obrázek 1.4: Analytický vzor Conversion Ratio

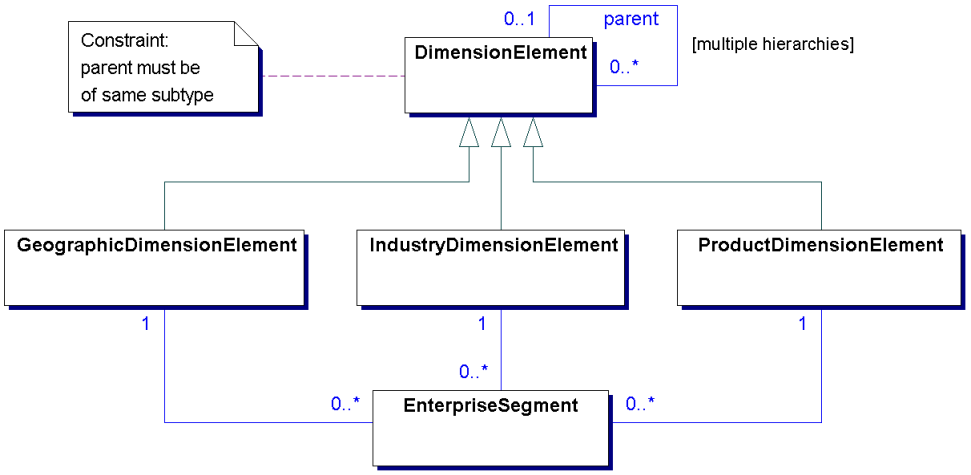
1.2.3 Observations for Corporate Finance

Oblast měření a pozorování se nemusí týkat jen pacientů u lékaře. Stejný princip lze využít i v mnoha dalších oblastech. Jednou z nich je sledování finanční úspěšnosti podniku. Sada vzorů *Observations for Corporate Finance* představuje rozšíření předchozího souboru vzorů se zaměřením na tuto oblast. Hlavním rozdílem je zde skutečnost, že cílem není pozorovat a měřit jednotlivé objekty, ale různé jejich kombinace a skupiny určené pomocí definovaných kritérií.

Název vzoru: Enterprise Segment
Číselná identifikace: 4.1
Vzor *Enterprise Segment* pomáhá rozčlenit společnost na pozorované části. Tyto části vznikají dynamicky v závislosti na definovaných úhlech pohledu (*dimension*). Takovým úhlem pohledu může být typ vyráběného produktu, geografická oblast prodeje či průmyslový sektor, do kterého výrobky dále putují. Struktura vzoru je na obrázku 1.5.

1.2.4 Referring to Objects

Vzory sady *Referring to Objects* se zabývají otázkou pojmenování a identifikace objektů z pohledu uživatele systému. V reálném světě se na objekty odkazujeme nejčastěji pomocí



Obrázek 1.5: Analytický vzor Enterprise Segment

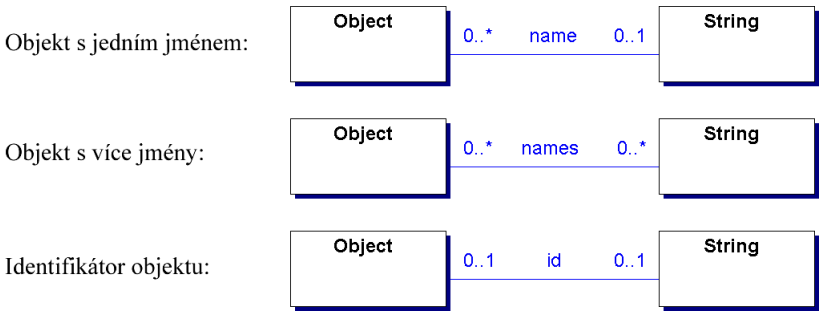
jména, přestože tento přístup nezaručuje jednoznačnou identifikaci. Na své přátele odkazujeme často jen pomocí křestního jména či jména s dodatečným upřesněním (například „Petr z práce“). Je to pro nás lépe určující než odkazovat se na ně pomocí jednoznačného identifikačního čísla. Informační systém by měl tyto přístupy umožňovat také, aby co nejvíce korespondoval s reálným světem. Sada vzorů *Referring to Objects* se nezabývá jen pojmenováním objektů, ale také rozpoznáním ekvivalence mezi objekty se stejným či různým názvem, jejich slučováním a rozdělováním.

Název vzoru: Name
Číselná identifikace: 5.1
Základním vzorem pro pojmenování objektů je vzor *Name*. Ten diskutuje možnosti vztahu objektu a jeho jména. První možností je přiřadit objektu pouze jedno jméno, které může být strukturované. Druhou možností je přiřadit jednomu objektu více jmen. To může reflektovat situaci, kdy jednoho člověka pojmenovává jinak jeho matka, partnerka a přátelé. Na tuto situaci navazuje vzor *Identification Scheme*. Poslední možností je přiřadit objektu jednoznačné identifikační číslo. V některých situacích je to dokonce naprosto přirozené. Příkladem může být kód výrobku. Alternativy vzoru jsou na obrázku 1.6.

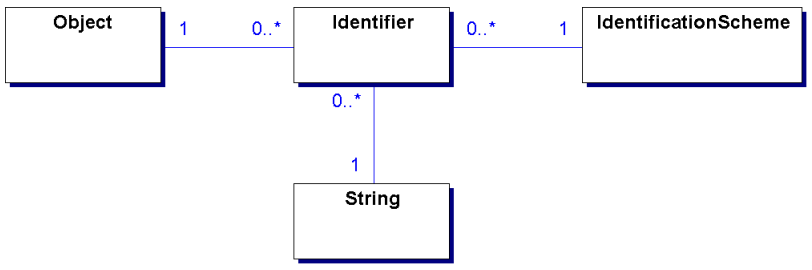
Název vzoru: Identification Scheme
Číselná identifikace: 5.2
Vzor *Identification Scheme* použijeme v případě, že se na objekt odkazujeme podle několika, často pevně daných, identifikačních schémat. Tato situace vzniká často při integraci více systémů využívajících stejnou evidenci objektů, pro něž každý subsystém využívá vlastní způsob identifikace. Struktura vzoru je na obrázku 1.7.

1.2.5 Inventory and Accounting

V mnoha systémech máme zájem sledovat pohyb určitých jednotek (nejčastěji peněz) mezi různými místy (nazvěme je účty). Příkladem může být sledování nakládání s finančními



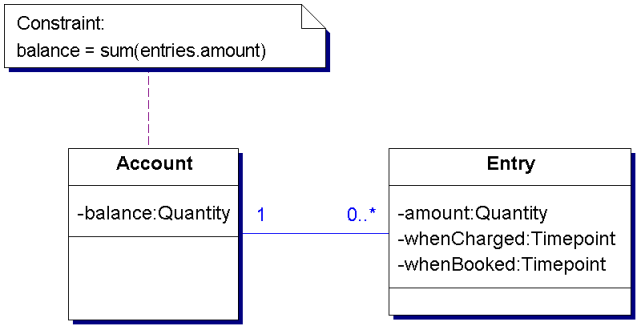
Obrázek 1.6: Alternativy analytického vzoru Name



Obrázek 1.7: Analytický vzor Identification Scheme

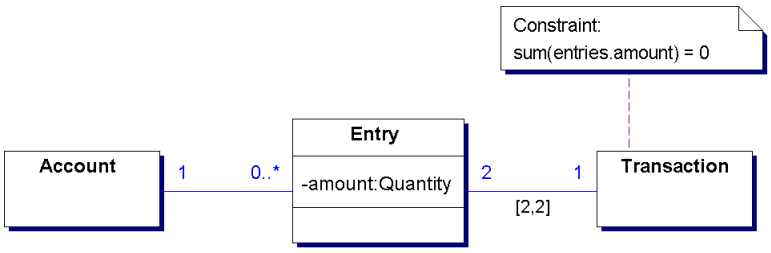
prostředky ve společnosti. Díky sadě vzorů *Inventory and Accounting* můžeme určit účty, které nás zajímají, a efektivně sledovat změny jejich obsahu včetně příčin těchto změn.

Název vzoru: Account
Číselná identifikace: 6.1
Vzor *Account* definuje objekt sloužící jako kontejner s častou změnou obsahu. Takový objekt si můžeme představit jako účet, na kterém se pohybuje výše peněz. Díky vzoru *Account* získáme nejen informace o aktuálním množství peněz na účtu, ale také celou historii změn stavu účtu. Struktura vzoru je na obrázku 1.8.



Obrázek 1.8: Analytický vzor Account

Název vzoru: Transactions
Číselná identifikace: 6.2
Zasaďme účet popsaný výše do informačního systému banky a nadefinujme zvláštní účet i pro peníze přicházející a odcházející v hotovosti. Pak můžeme nadneseně říci, že v tomto systému peníze procházejí finančním koloběhem aniž by někde vznikaly či zanikaly. V takovém systému je jistě užitečné sledovat cestu peněz mezi účty. K tomu však vzor *Account* nestačí, protože eviduje pouze změny stavu účtu, ne jejich příčiny (z jakého účtu peníze přišly, na jaký účet peníze putují). Vzor *Transaction* přináší právě toto rozšíření díky propojení každého vkladu na účet s výběrem stejné částky z účtu jiného. Struktura vzoru je na obrázku 1.9.

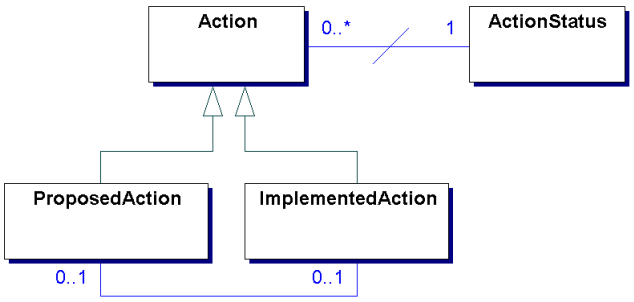


Obrázek 1.9: Analytický vzor Transactions

1.2.6 Planning

Sada vzorů *Planning* přináší některé základní vzory pro podporu plánování akcí. Patří k nim vzory pro odlišení naplánovaných a realizovaných akcí, vzory pro sestavování protokolů naplánovaných akcí či vzory pro podporu přiřazení zdrojů naplánovaným akcím. Složitější vzory pak nabízejí možnost definovat spouštěcí podmínky připravených protokolů a stanovit jejich očekávané výsledky.

Název vzoru: Proposed and Implemented Action
Číselná identifikace: 8.1
Vzor *Proposed and Implemented Action* je určen pro případy, kdy máme zájem uchovávat informace o akci ve dvou podobách (jak byla akce *naplánována*, jak byla *realizována*). Takový přístup nám v budoucnu umožní porovnat plán s realitou a získat tak cenné zkušenosti pro plánování dalších akcí (například volba větší časové rezervy). Tento vzor počítá i s eventualitou, že k realizaci akce dojde bez předchozího plánu, případně že plán nikdy nebude realizován. Struktura vzoru je na obrázku 1.10.



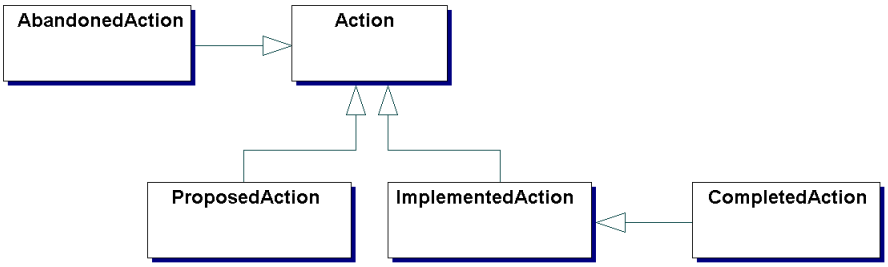
Obrázek 1.10: Analytický vzor Proposed and Implemented Action

Název vzoru: Completed and Abandoned Actions
Číselná identifikace: 8.2
Vzor *Completed and Abandoned Actions* jde v tomto přístupu ještě dále a odlišuje i *dokončené (completed)* a *zrušené (abandoned)* akce. Tento přístup je nutný v systémech, kde je složité odlišit, zda akce stále běží nebo byla zrušena, případně zda byla akce úspěšně dokončena či nikoliv. Struktura vzoru je na obrázku 1.11.

1.2.7 Trading

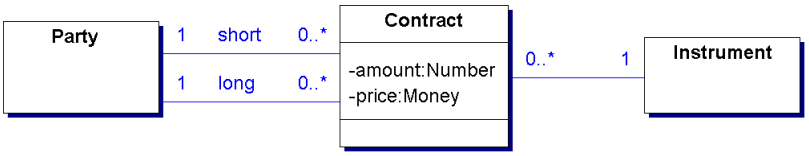
Vzory sady *Trading* přináší podporu pro obchodování a uzavírání kontraktů. Základní vzory se zaměřují na definování jednotlivých *kontraktů*, složitější vzory pak na odhad vhodné ceny v závislosti na vývoji trhu a typu obchodního vztahu (prodej, koupě), případně na možnosti seskupování kontraktů do dynamických skupin podle definovaných kritérií.

Název vzoru: Contract
Číselná identifikace: 9.1



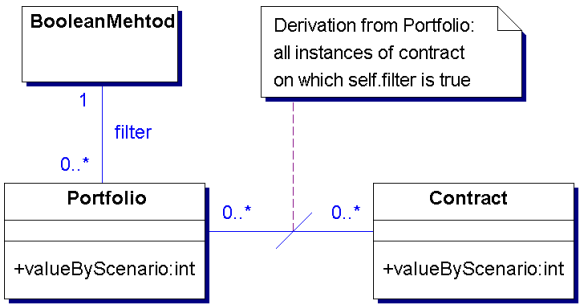
Obrázek 1.11: Analytický vzor Completed and Abandoned Actions

Vzor *Contract* definuje kontrakt jako vztah mezi *prodávajícím (short)* a *kupujícím (long)* účastníkem obchodu. Předmětem tohoto vztahu je obchodované zboží se stanovenou prodejní/nákupní cenou. Struktura vzoru je na obrázku 1.12.



Obrázek 1.12: Analytický vzor Contract

Název vzoru: Portfolio
Číselná identifikace: 9.2
V některých případech je vhodné seskupovat kontrakty do *skupin (portfolií)*, na které lze pohlížet jako na celky. U takových skupin můžeme hodnotit jejich celkový obchodní přínos podle zadaného scénáře. Vzor *Portfolio* využívá k určení skupiny soubor podmínek nazývaný *filter*. Díky filtru umíme pro každé portfolio rozhodnout, zda je kontrakt jeho součástí či nikoliv. Struktura vzoru je na obrázku 1.13.

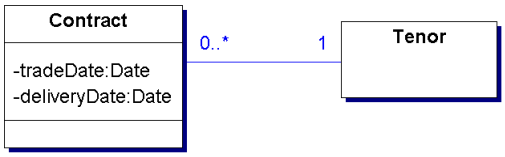


Obrázek 1.13: Analytický vzor Portfolio

1.2.8 Derivative Contracts

Rozšířením předchozí sady vzorů jsou vzory pro podporu odvozených kontraktů. Nejčastější formou jsou kontrakty uzavírané s časovou rezervou. Tato rezerva může být pevná či pohyblivá. V případě pevné časové rezervy se stanoví den naplnění kontraktu, určí se pevně cena a na základě toho se podepíše smlouva. Obchod se realizuje přesně ve stanovený den. Situace při uzavírání kontraktů s pohyblivou časovou rezervou je poněkud komplikovanější. Kupující při dohodě kontraktu získává (kupuje) možnost naplnění kontraktu během stanoveného časového termínu. Záleží na něm, zda a případně kdy této možnosti využije.

Název vzoru: Forward Contracts
Číselná identifikace: 10.1
Vzor *Forward Contracts* definuje kontrakt sjednávaný s časovým předstihem. Kontraktu je tedy přiřazen časový předstih (*Tenor*) a dále dvě data reprezentující den uzavření smlouvy a den naplnění obchodu. Struktura vzoru je na obrázku 1.14.



Obrázek 1.14: Analytický vzor Forward Contracts

1.2.9 Trading Packages

Kapitola *Trading Packages* se zamýšlí nad členěním systému s komplikovanou doménovou oblastí na balíčky (*Packages*). Hlavní otázkou jsou zde vztahy mezi balíčky a jejich vzájemná viditelnost. Vzory představené v této kapitole mají už spíše architektonický charakter a tvoří přechod mezi první a druhou částí vzorů z [4][1].

[1]První část knihy představuje analytické vzory, druhá část knihy se věnuje pomocným vzorům.

Kapitola 2

Návrhové vzory

2.1 Návrh podle vzorů

Návrhový vzor je popisem efektivního řešení obecného problému, které je znovupoužitelné na další problémy podobného typu. Tato obecná řešení skrývají velkou sílu v tom, že vznikla praxí zkušených návrhářů, kteří se s daným problémem mnohokrát setkali. Méně zkušení vývojáři se tak mohou vyvarovat chyb a ušetřit podstatné množství práce při prvním setkání s danou situací. Řešení problémů při návrhu však není jediným důvodem používání vzorů. Velká síla vzorů spočívá ve flexibilním přístupu, který vzory přináší. Seznámení se s touto problematikou otevírá nový pohled na navrhovaný software, ve kterém lépe objevujeme adepty na budoucí problémy, které se mohou objevit se snahami software rozšířit. Výsledná aplikace je tedy díky použití a studiu vzorů stává flexibilnější vzhledem k následnému rozšiřování.

Vzory pomáhají přispět ke kvalitnímu návrhu také tím, že se snaží dodržovat pravidla pro předcházení obecně známým rizikům spojeným s objektovým návrhem. Seznam těchto rizik spolu s popisem můžeme nalézt v knize [6]. Mezi nejčastější rizika podle autora této knihy patří závislost objektů na třídě, závislost na specifické operaci, závislost na SW a HW, závislost na objektové implementaci, závislost na algoritmu, těsné vazby objektů, přeceňování dědění a změny bez přístupu ke kódu třídy.

2.1.1 Metody aplikace vzorů

Jak jsme již naznačili v kapitole 1.1, systematickou aplikaci vzorů je možné provést dvěma základními způsoby. Tyto způsoby se liší v načasováním nasazení vzorů. Je otázkou, zda je vhodnější nejprve vytvořit popisné diagramy a poté hledat možné problémy a řešit je za pomocí vzorů, či návrh začít volbou vhodných vzorů, jejich spojením a poté doplněním chybějících tříd. Metody aplikace vzorů při návrhu se od aplikace analytických vzorů v ničem podstatném neliší, proto dále odkazujeme na text kapitoly 1.1, kde byla tato problematika blíže rozvedena.

2.1.2 Návrhové vzory vs. analytické vzory

Na tomto místě se nabízí otázka, jaký je rozdíl mezi analytickými a návrhovými vzory. Hlavní rozdíl je v typu problémů, které nám příslušné vzory pomáhají řešit. Analytické vzory pomáhají při modelování obchodní doménové oblasti. Jsou tedy na příslušné obchodní doméně závislé. Nejsou však závislé na použité implementační technologii. Vzory

pro vyjádření konkrétní obchodní domény (například organizační struktury) využíváme bez ohledu na to, jaká technologie bude použita na vývoj aplikace. Naopak návrhové vzory pomáhají řešit implementační problémy, které jsou často nezávislé na obchodní doménové oblasti. Pokud řešíme problém propojení webového klienta s jádrem aplikace, není pro nás příliš podstatné, zda se to týká aplikace pro objednávání vstupenek do kina, či vyhledávání v knihovní databázi.

2.1.3 Vzory našeho zájmu

V následujících kapitolách se zaměříme na návrhové vzory související s vývojem webových aplikací. Jejich výčet jistě není úplný, přinese však čtenáři dostatečný rozhled v této oblasti. V současnosti je aplikace návrhových (i jiných) vzorů velmi populární téma, proto je nabídka vzorů velice široká. Z důvodu jednoduchosti a přehlednosti jsme se pokusili vybrat několik vhodných katalogů vzorů a z každého z nich stručně představit několik reprezentantů. Při zájmu o více informací je možné najít úplný výčet vzorů každého katalogu v uvedené literatuře.

2.2 Návrhové vzory GoF

Návrhové vzory GoF (GoF Design Patterns) získaly svůj neoficiální název podle čtveřice odborníků (*Gang of Four*), kteří se podíleli na knize [5], v níž byly představeny. Tato kniha je považována za přelomovou publikaci, která poprvé zachytila návrhové vzory včetně kvalitního popisu a vizuálních diagramů. Návrhové vzory se sice používaly již před vydáním této knihy, často se ale předávaly jen ve formě zkušeností z analytika na analytika a nebyly nikde souhrnně publikovány.

Kniha [5] představuje celkem 23 návrhových vzorů, které jsou strukturovány podle dvou úhlů pohledu – podle účelu a podle rozsahu platnosti. Podle účelu dělíme vzory na:

- vzory pro tvorbu objektů (*Creational*) – zachycují proces vytváření objektů
- vzory struktury (*Structural*) – ukazují přístupy k vytváření struktur objektů
- vzory chování (*Behavioral*) – zachycují komunikaci mezi objekty

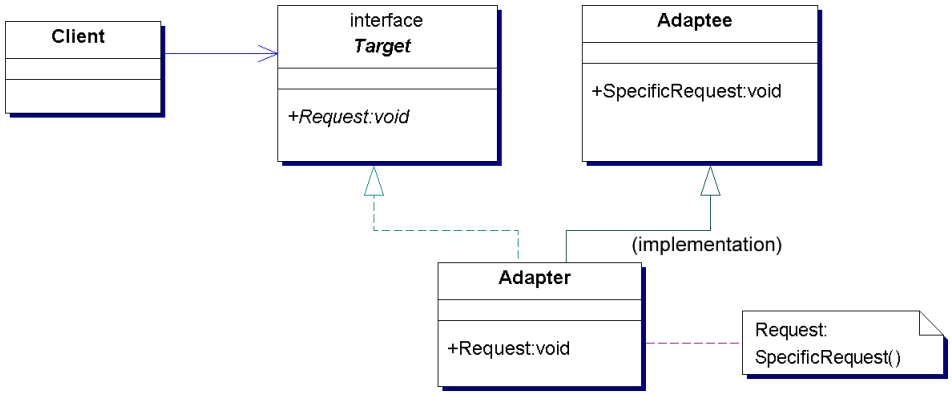
Podle rozsahu platnosti dělíme vzory na:

- vzory s platností v rozsahu tříd (*Class*) – popisují vztahy a interakce mezi třídami
- vzory s platností v rozsahu objektů (*Object*) – řeší interakce mezi objekty, které často nastávají až za běhu aplikace

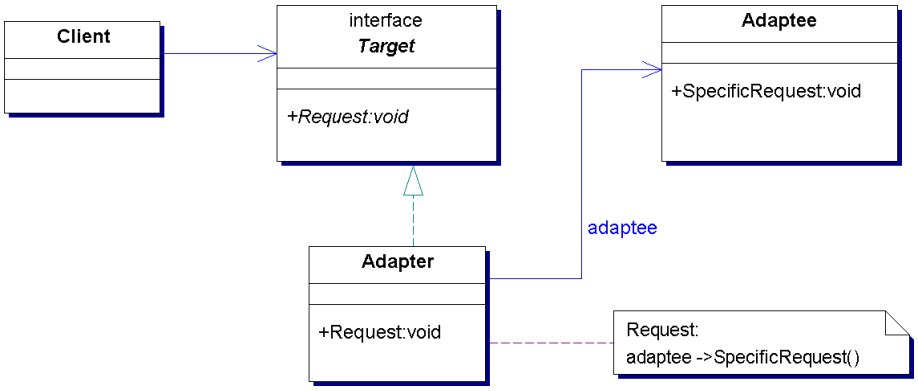
V následujícím textu si představíme 4 návrhové vzory, které nalézají uplatnění zejména při návrhu webových aplikací. Jsou to vzory *Adapter*, *Facade*, *Proxy* a *Observer*.

Název vzoru: Adapter
Klasifikace: Class nebo Object, Structural
Vzor *Adapter* pomáhá připojit k aplikaci třídu (případně celou komponentu) s nekompatibilním rozhraním (*interface*). To je potřebné zejména v případech, kdy chceme provést integraci s jinou aplikací nebo pokud chceme využít (*reuse*) část jiného systému. Vzor Adapter má dvě varianty, *Class* a *Object*. Varianta *Class* používá jako spojovací prvek (mezi klientem očekávaným rozhraním a neznámým rozhraním) třídu spolupracující přes

dědičnost, varianta *Object* používá jako spojovací prvek vložený objekt. V oblasti webových aplikací nalézá tento vzor uplatnění zejména při vzdálené integraci webových aplikací přes Internet. V tom případě je vhodné kombinovat ho se vzorem *Proxy*. Struktura vzoru ve variantě *Class* je na obrázku 2.1, struktura vzoru ve variantě *Object* je na obrázku 2.2.

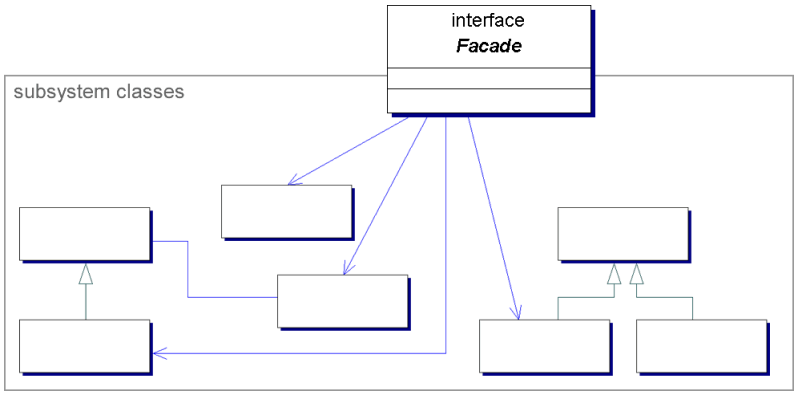


Obrázek 2.1: Návrhový vzor GoF – Adapter, varianta Class



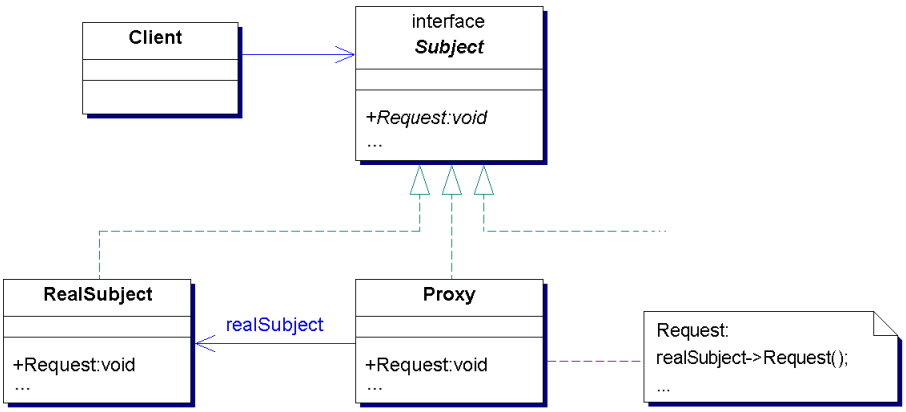
Obrázek 2.2: Návrhový vzor GoF – Adapter, varianta Object

Název vzoru: Facade
Klasifikace: Object, Structural
Vzor *Facade* zavádí zjednodušené rozhraní pro přístup k složitému subsystému, čímž zjednodušuje a zpřehledňuje přístup pro klienta. Klient je tak odstíněn od množství složitých rozhraní, která často poskytují funkcionalitu, kterou klient nepotřebuje znát. Namísto toho je mu zprostředkováno jedno (případně několik málo) rozhraní, které tvoří portál k jemu potřebným funkcím. Příklad z oblastí webových aplikací se sám nabízí. Vzor Facade je ideální pro přístup webového klienta s omezenou funkčností (objednání lístku do kina) ke složitému jádru aplikace (informační systém kina). Struktura vzoru je na obrázku 2.3.



Obrázek 2.3: Návrhový vzor GoF – Facade

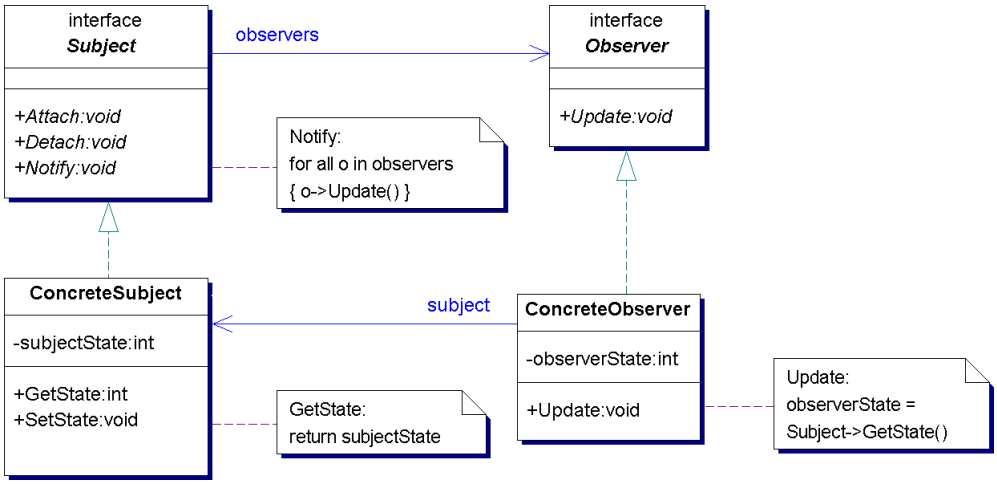
Název vzoru: Proxy
Klasifikace: Object, Structural
Vzor *Proxy* zavádí zástupce objektu, ke kterému chceme získat přístup. Zástupce se navenek jeví jako originální objekt. Při přijetí požadavku provede operace, kvůli kterým byl vytvořen, a přepošle požadavek originálnímu objektu. Stejným způsobem vrátí odpověď. Důvodů použití vzoru *Proxy* může být hned několik. Nejčastější je podpora přístupu ke vzdálenému objektu (po síti). Dalším důvodem může být bezpečnost (zástupce přeposílá pouze bezpečné či autorizované požadavky). Při vývoji webových aplikací je tento vzor nepostradatelný při propojování více aplikací přes Internet. Struktura vzoru je na obrázku 2.4.



Obrázek 2.4: Návrhový vzor GoF – Proxy

Název vzoru: Observer
Klasifikace: Object, Behavioral
Vzor *Observer* pomáhá řešit problém s aktualizací dat (na straně pozorovatelů) závisejících na aktuálním stavu určitého (pozorovaného) objektu. Příkladem může být zobrazování

aktuální teploty vzduchu na webových stránkách v závislosti na naměřené hodnotě uložené na serveru. Vzor *Observer* pomáhá zajistit, aby se vždy zobrazovala aktuální hodnota teploty podle posledního měření bez nutnosti obnovit webovou stránku. Struktura vzoru je na obrázku 2.5.



Obrázek 2.5: Návrhový vzor GoF – Observer

2.3 Návrhové vzory POSA

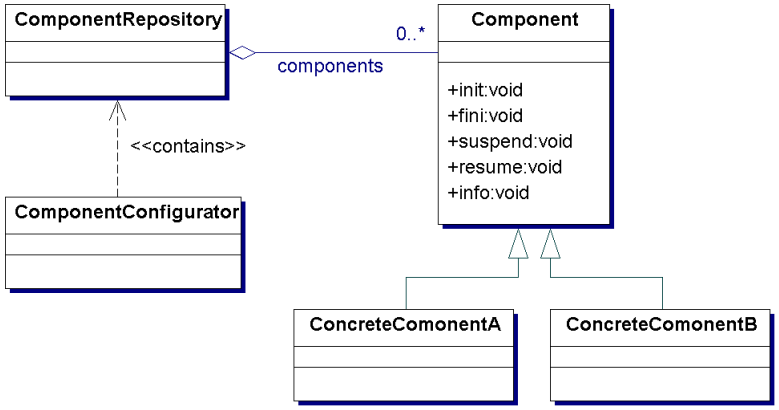
Návrhové vzory POSA (Pattern Oriented Software Architecture) získaly své jméno podle knihy, ve které byly představeny. Pod tímto názvem vyšlo již pět knih (poslední v roce 2007), z nichž se zde zaměříme na vzory představené v druhé z nich, která nese úplný název *Pattern Oriented Software Architecture – Patterns for Concurrent and Networked Objects*. Celkem představuje tato kniha 17 vzorů rozdělených do následujících čtyř skupin:

- Service Access and Configuration Patterns
- Event Handling Patterns
- Synchronization Patterns
- Concurrency Patterns

Vzory sady *Service Access and Configuration Patterns* se věnují definování rozhraní pro efektivní přístup ke službám a komponentám včetně jejich konfigurace jak v lokálních aplikacích tak i v aplikacích propojených sítí. Sada vzorů *Event Handling Patterns* popisuje efektivní způsoby správy událostí v síťových systémech. Vzory *Synchronization Patterns* se zaměřují na otázku uzamykání záznamů pro případ souběžného přístupu více klientů k jedné kritické sekci. Synchronizací se tedy rozumí synchronizace souběžného přístupu. Sada vzorů *Concurrency Patterns* se zabývá problémy vznikajícími v systémech postavených na architektuře s podporou souběžného zpracování (*Concurrency Architecture*). Dále jsou vzory klasifikovány podle toho, zda řeší návrhový (*Design*) či architektonický (*Architectural*) problém.

Ve vztahu k webovým aplikacím jsou pro nás zajímavé pouze soubory *Service Access and Configuration Patterns* a *Event Handling Patterns*. Z těchto dvou souborů si představíme pro ilustraci 3 návrhové vzory.

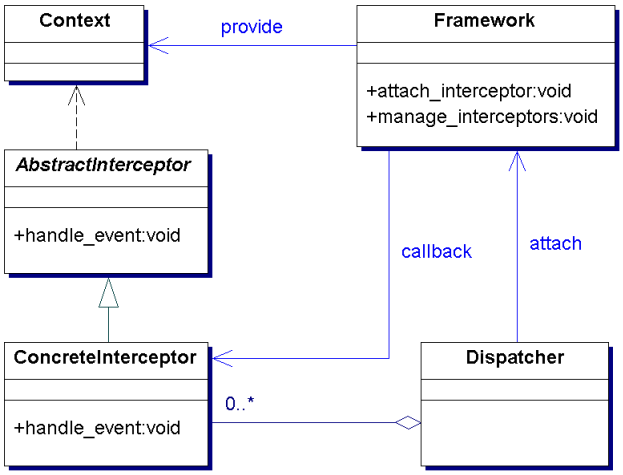
Název vzoru: Component Configurator
Klasifikace: Service Access and Configuration Patterns, Design
Vzor *Component Configurator* umožňuje aplikaci připojovat a odpojovat implementace svých komponent za běhu bez nutnosti změny kódu a opětovné kompilace. U některých aplikací totiž nastává situace, kdy lze nejefektivnější implementaci komponenty (strategie prováděných operací) zvolit až na základě aktuálních podmínek za běhu, nikoli předem. Příklad tohoto principu můžeme pozorovat v případě používání *Java appletů* v aplikaci. Každý *Java applet* může být dynamicky načten, nakonfigurován a po použití opět odstraněn. O správu appletů (roli prvku *Configurator*) se stará *JVM (Java Virtual Machine)* ve spojení s webovým prohlížečem. Vzor *Component Configurator* má čtyři účastníky: *Component* (jednotné rozhraní pro práci s konkrétními komponentami), *ConcreteComponent* (konkrétní komponenta), *ComponentConfigurator* (objekt pro kontrolu připojování a odpojování komponent) a *ComponentRepository* (sklad používaný objektem *ComponentConfigurator* ke správě všech aktuálně připojených komponent). Struktura vzoru je na obrázku 2.6.



Obrázek 2.6: Návrhový vzor POSA – Component Configurator

Název vzoru: Interceptor
Klasifikace: Service Access and Configuration Patterns, Architectural
Ve fázi návrhu je v některých případech těžké odhadnout, jaké služby (funkce) by měl systém podporovat. Pokud by do systému byly zahrnuty všechny služby, které mohou být potřeba, mohl by být systém zbytečné velký a pomalý. Vzor *Interceptor* nabízí řešení. Vzor umožňuje, aby byly služby do systému přidávány postupně za provozu včetně zadání událostí, při jejichž výskytu mají být automaticky spuštěny. Tento vzor má ještě druhé využití a to v případech, kdy vyvíjíme produkt, do kterého by měly být integrovatelné služby, které zatím neznáme. Příkladem může být situace, kdy vyvíjíme nový internetový prohlížeč. Je zřejmé, že by tento prohlížeč měl nabízet podporu pro zobrazování všech

formátů obrázků a animací. To však nestačí, musí také nabízet možnost přidat pluginy pro formáty, které se mohou objevit až po přivedení produktu na trh. Vzor *Interceptor* má pět účastníků: *Framework* (prostředí systému), *Dispatcher* (zprostředkovává aplikaci registraci nových objektů Interceptor), *Context* (obsahuje informace o události, která má být objektem Interceptor spuštěna), *AbstractInterceptor*, *ConcreteInterceptor*. Struktura vzoru je na obrázku 2.7.



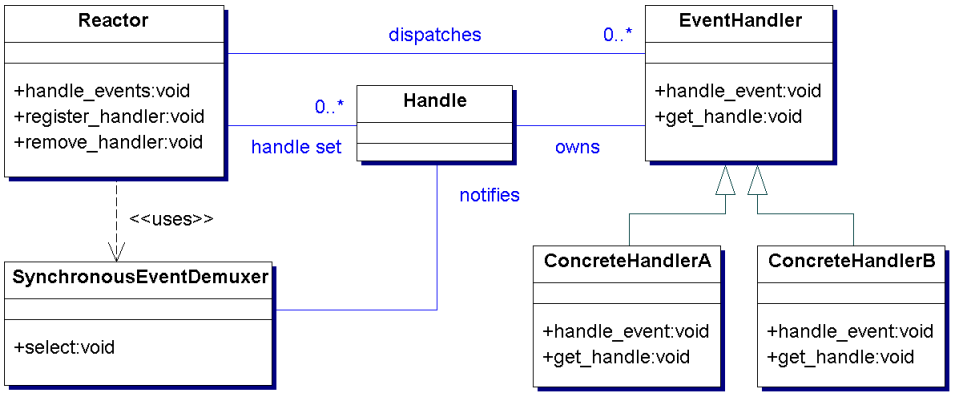
Obrázek 2.7: Návrhový vzor POSA – Interceptor

Název vzoru: Reactor
Klasifikace: Event Handling Patterns, Architectural
Vzor *Reactor* pomáhá v případech, kdy je třeba vytvořit správu požadavků přicházejících k serveru od více různých klientů (např. webových). Vzor pomáhá požadavky roztřídit a přeposlat dál. Využívá přitom převrácený tok řízení, který je znám pod názvem *Hollywood Principle* a jeho myšlenka zní: „*Nevolejte nám, my zavoláme Vám*“. V praxi to znamená, že objekty pro správu požadavků (*EventHandler*) pasivně čekají, až se jim *Reactor* sám ozve. Vzor *Reactor* má šest účastníků s významem odpovídajícím následujícímu popisu. *Reactor* je objekt odchytávající pomocí objektu *SynchronousEventDemuxer* požadavky přicházející od různých klientů. Po detekování nového požadavku upozorní objekt *SynchronousEventDemuxer* příslušný *Handle*, registrovaný objektem *EventHandlerem* u objektu *Reactor*. To je detekováno objektem *Reactor*, který poté přepošle požadavek příslušnému objektu *EventHandler* k vyřízení. Struktura vzoru je na obrázku 2.8.

2.4 Návrhové vzory EJB

Návrhové vzory EJB (EJB design patterns) pomáhají řešit problémy objevující se při vývoji aplikací na základě komponentové architektury *EJB (Enterprise JavaBeans)*. Výběr ukázkových vzorů vychází z knihy [**?**]. Tato kniha představuje 20 návrhových vzorů rozdělených do následujících pěti souborů:

- EJB Layer Architectural Patterns



Obrázek 2.8: Návrhový vzor POSA – Reactor

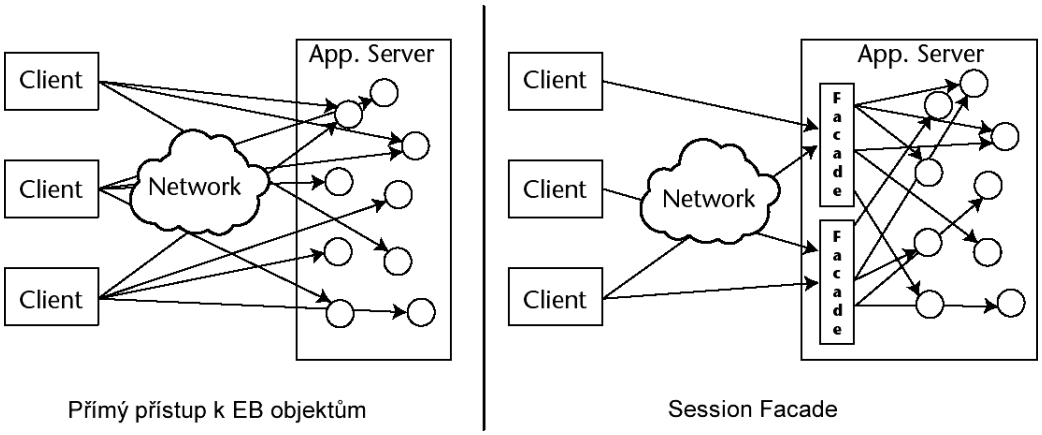
- Inter-Tier Data Transfer Patterns
- Transaction and Persistence Patterns
- Client-Side EJB Interaction Patterns
- Primary Key Generation Strategies

Vzory *EJB Layer Architectural Patterns* se zabývají základními otázkami, které je nutné zohlednit při výběru architektury vytvářeného systému (rozdělení logiky mezi vrstvy). Výsledkem je několik architektonických vzorů, které toto rozdělení vrstev definují. Vzory sady *Inter-Tier Data Transfer Patterns* řeší otázku, jak v distribuovaných aplikacích přenášet data mezi jednotlivými vrstvami (často mezi klientem a serverem). Vzory *Transaction and Persistence Patterns* představuje několik vzorů pro kontrolu transakčního zpracování, konzistence a persistence. Vzory *Client-Side EJB Interaction Patterns* se zabývají otázkou, jak zlepšit udržovatelnost a výkonnost klientských EJB aplikací. Vzory sady *Primary Key Generation Strategies* nabízejí strategie pro generování primárních klíčů v architektuře EJB při zachování přenositelnosti a rozšiřitelnosti způsobu generování.

Z představených vzorů jsme vybrali 4 ukázkové vzory, které v následujícím textu blíže popíšeme. Vzory jsou voleny s ohledem na použití při návrhu webových aplikací.

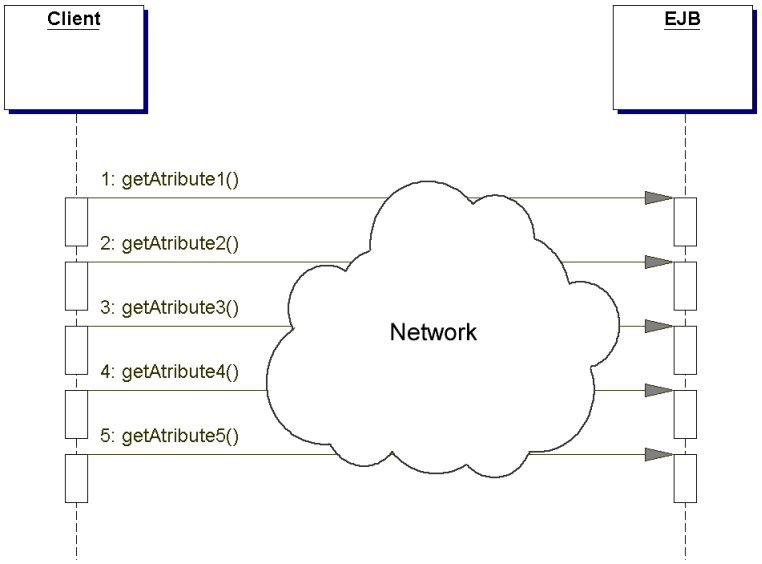
Název vzoru: Session Facade
Klasifikace: EJB Layer Architectural Patterns
Vzor *Session Facade* ukazuje, jak správně rozdělit aplikační logiku v distribuovaném systému tak, aby se minimalizovala závislost mezi klientem a serverem. Nejčastějším přístupem je umístit logiku na stranu klienta. Pokud ale potřebuje klient realizovat i velice jednoduchý případ užití, může to znamenat mnoho volání *entity bean (EB)* objektů na serveru přes síť a s tím spojené zpomalení aplikace kvůli obsluze každého požadavku v samostatné transakci. Pokud bychom naopak podobná volání sdružili a vytvořili z nich funkci uloženou na serveru (v jednotlivých EB objektech), kterou klient zavolá pouze jednou s určitými parametry, může se stát, že EB objekty přeplníme funkcemi, které znepřehlední jejich udržovatelnost a znemožní jejich opakované použití (*reuse*). Vzor *Session Facade* nabízí jako řešení vytvoření vrstvy na straně serveru, která pomocí objektů *session*

bean (SB) obaluje vrstvu EB objektů. Tato vrstva s názvem *Session Facade* tedy implementuje veškerou aplikační logiku a tvoří tak prostředníka mezi klientem a daty na serveru. Struktura vzoru je na obrázku 2.9.

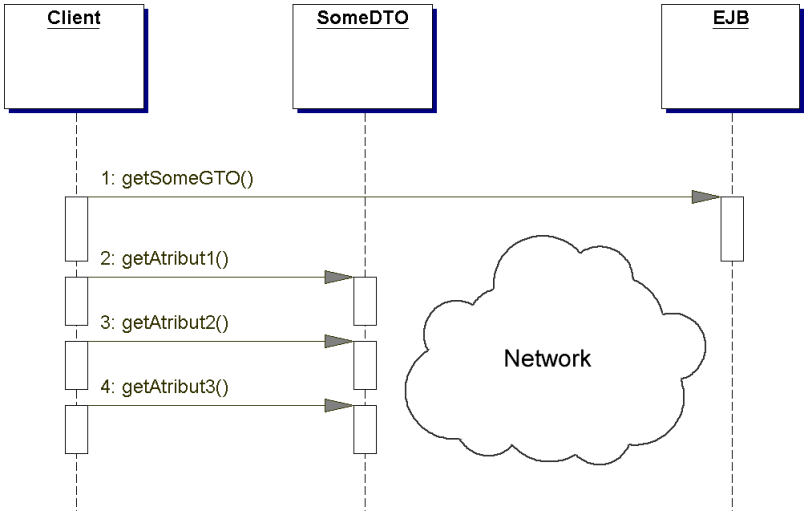


Obrázek 2.9: Návrhový vzor EJB – Session Facade

Název vzoru: Data Transfer Object
Klasifikace: Inter-Tier Data Transfer Patterns
Vzor *Data Transfer Object* řeší problém, jak co nejefektivněji přesouvat data mezi klientem (*servlet*, *applet*) a serverem (*session bean*, *entity bean*, *message-driven bean*) v obou směrech (čtení ze serveru za účelem zobrazení, posílání na server za účelem změny dat). Vzor pomáhá v situacích, kdy je nutné mezi klientem a serverem vyměnit velké množství dat. Pokud se data přesouvají v samostatných požadavcích, dojde ke zbytečně velkému zatížení systému z důvodu obsloužení každého požadavku zvlášť. Vzor *Data Transfer Object* navrhuje seskupit všechna data, která je třeba přenést, do jednoho objektu, který je pak přenesen v rámci jednoho požadavku. Struktura vzoru je na obrázku 2.11.

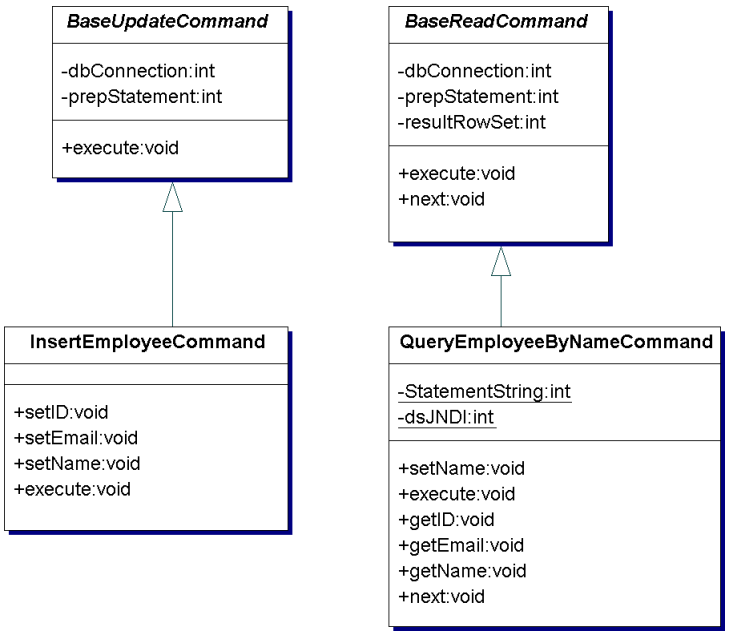


Obrázek 2.10: Neefektivní způsob čtení dat ze serveru



Obrázek 2.11: Návrhový vzor EJB – Data Transfer Object

Název vzoru: Data Access Command Bean
Klasifikace: Transaction and Persistence Patterns
Vzor *Data Access Command Bean* pomáhá zmírnit propojení aplikační logiky uložené v EJB objektech s persistentní logikou uchovávající a spravující data v databázi. Tento vzor radí používat pro správu dat v databázi speciální objekty, které nazývá *Data Access Command Bean*. Tyto objekty řídí přístup k datům v databázi a starají se o jejich správu. V praxi to znamená, že pro databázovou tabulku *Zaměstnanci* můžeme vytvoříme dva objekty *Data Access Command Bean*. První z nich bude vytvářet nové zaměstnance, druhý bude poskytovat funkci vyhledání zaměstnance podle zadaného jména. Pokud bychom funkce pro přístup k této tabulce chtěli dále rozšiřovat, můžeme objekty *Data Access Command Bean* sdružit dědičností pod několik abstraktních objektů, které mohou být například *BaseUpdateCommand* a *BaseReadCommand*. Jednoduchý příklad použití vzoru je na obrázku 2.12.



Obrázek 2.12: Příklad použití návrhového vzoru EJB – Data Access Command Bean

Název vzoru: Business Delegate
Klasifikace: Client-Side EJB Interaction Patterns
Při použití vzoru *Session Facade* dochází k pevnému provázání klienta s vrstvou EJB objektů. Vzor *Business Delegate* toto provázání uvolňuje přidáním prostředníka mezi tyto vrstvy. Prostředníkem je vrstva *Business Delegate*, která pro každou třídu vrstvy *Session Facade* poskytuje klientovi tzv. *delegáta* (jednoduchá Javovská třída na straně klienta), který mu zprostředkovává komunikaci s příslušným SB objektem. Smysl delegáta je například v tom, že se může postarat o vyřízení chybových výjimek či přerušených transakcí a tím usnadnit klientovi komunikaci se serverem. Všimněme si, že tento vzor v mnohém připomíná vzor *Proxy* z katalogu *GoF*.

2.5 Vzory webových služeb

Vzory webových služeb (Web Service Patterns) byly přehledně shrnuty v knize [7]. Tato kniha představuje celkem 15 návrhových vzorů, které jsou rozděleny do pěti skupin podle typu použití:

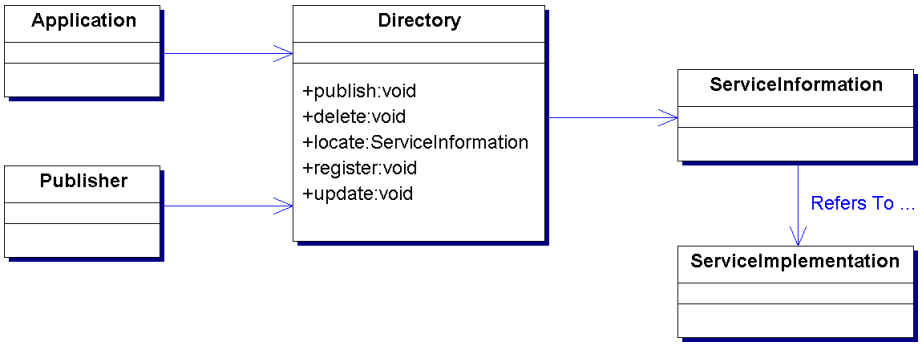
- Learning About Web Services
- Adapting to Web Services
- Determining When Changes Occur
- Refining the Structure of Your Web Services
- Creating Flexibility in Your Web Services

Vzory sady *Learning About Web Services* pomáhají proniknout do souvislostí týkajících se použití webových služeb. Vzory skupiny *Adapting to Web Services* pomáhají porozumět přechodu od implementace objektových Java služeb k neobjektovým webovým službám pomocí takzvaných *„business objektů“*. Vzory souboru *Determining When Changes Occur* popisují způsoby detekce změny v aplikaci. To se většinou týká snahy klienta synchronizovat určitá svá data se serverem. Vzory sady *Refining the Structure of Your Web Services* pomáhající přizpůsobit prostředí webových služeb konkrétním potřebám aplikace. Vzory skupiny *Creating Flexibility in Your Web Services* rozšiřují předchozí základní vzory s cílem optimalizace vytvořeného prostředí.

Z celkových 15 vzorů jsme vybrali 4 jednodušší ukázkové vzory tak, aby byly snadné na pochopení i pro čtenáře, který s architekturou webových služeb nemá mnoho zkušeností. Shodou okolností jsou tři z těchto vzorů variacemi na upravení známých vzorů *GoF* s ohledem na použití v souvislosti s webovými službami.

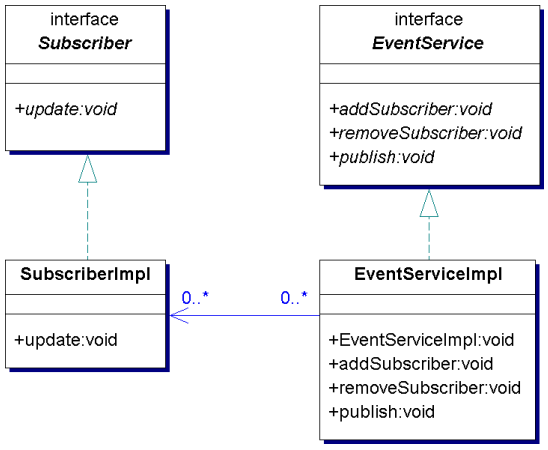
Název vzoru: Service Directory
Klasifikace: Learning About Web Services
Vzor *Service Directory* pomáhá pochopit základní ideu webových služeb. Tou je myšlenka, že pokud budou informace o službách poskytovaných různými servery přehledně popsané pomocí metadat a uložené ve speciálních registrech, je možné přenechat vyhledání vhodné služby a její integraci do zbylé části systému počítači (bez zapojení lidského faktoru). Velkou výhodou je fakt, že aplikace může vhodnou službu vyhledat, integrovat a začít používat za běhu bez aplikačních změn. Metadata popisující službu by měla obsahovat rozhraní, umístění (pro vytvoření vazby), komunikační mechanismus a informace o účelu, pro který byla služba vytvořena. Struktura vzoru je na obrázku 2.13.

Název vzoru: Observer
Klasifikace: Determining When Changes Occur
Vzor *Observer* koresponduje se stejnojmenným vzorem katalogu *GoF* (viz. kapitola 2.2). Publikace [7] přináší zajímavé rady na aplikaci tohoto vzoru v rámci architektury webových služeb. Základní myšlenkou je fakt, že si klienti mohou vyžádat WSDL soubor s definicí rozhraní Observeru zveřejněného v UDDI a vytvořit si jeho implementaci sami. Objekt *Observer* (pozorovatel) i objekt *Observable* (pozorovaný objekt) jsou reprezentovány webovými službami. Výhoda tohoto přístupu je v tom, že *Observer* i *Observable* zná pouze rozhraní toho druhého, nikoliv implementaci, a tudíž nevzniká omezení na architekturu a způsob jejich realizace.



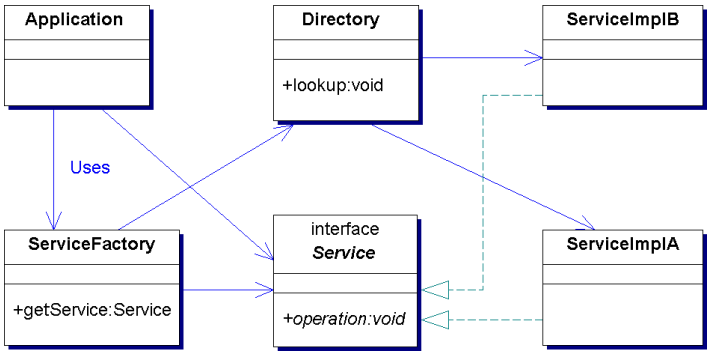
Obrázek 2.13: Vzor webových služeb – Service Directory

Název vzoru: Publish/Subscribe
Klasifikace: Determining When Changes Occur
Vzor *Publish/Subscribe* je rozšířením vzoru *Observer* s několika podstatnými změnami. První z nich je ta, že se pozorovatele neregistrují ke konkrétnímu objektu se zájmem o informace o změně jeho stavu. Pozorovatele se registrují k prostřednické službě, která je zodpovědná za rozesílání informací o výskytu určitých událostí. Tato služba se nazývá *EventService* a objekt, který se u ní registruje (analogie objektu *Observer* z minulého vzoru) se nazývá *Subscriber*. K těmto objektům pak přibývá ještě jeden důležitý objekt, *Publisher*, který zajišťuje mechanismus zveřejnění informace o výskytu události. Pokud by chtěl objekt *Subscriber* požádat o více informací o výskytu události, může kontaktovat právě objekt *Publisher*. Důvod, proč objekt Publisher není zakreslen v diagramu 2.14 je ten, že o něm ostatní objekty podle [7] nepotřebují vědět. Struktura vzoru je na obrázku 2.14. Při bližším pohledu na tuto strukturu je vidět, že je velmi podobná známému vzoru *Observer* z katalogu *GoF*.



Obrázek 2.14: Vzor webových služeb – Publish/Subscribe

Název vzoru: Service Factory
Klasifikace: Creating Flexibility in Your Web Services
Vzor *Service Factory* vychází ze známého vzoru *Factory Method*, který je součástí katalogu *GoF*. Jednou z výhod použití vzoru *Factory Method* je, že se implementace metod a funkcí dají dohledat až při běhu aplikace, nemusí být svázány s volajícími objekty už v době kompilace. Vzor *Service Factory* je obdobou a pomáhá dospět k tomu, aby se aplikace až při běhu mohla rozhodnout, které webové služby použije. Vzor je velice vhodný pro vytváření e-business aplikací, jelikož v této oblasti dochází k velmi častým změnám například při obměně obchodních partnerů. Je tedy užitečné mít v aplikaci obsaženu podporu dynamického užití těch webových služeb, které jsou nejvhodnější pro aktuální situaci. Pro plné pochopení vzoru je vhodné pročíst si nejprve popis vzoru *Service Directory*. Struktura vzoru je na obrázku 2.15.



Obrázek 2.15: Vzor webových služeb – Service Factory

2.6 Antivzory

Antivzory (AntiPatterns) jsou vzory popisující cesty od problémů ke špatným řešením, případně i cestu zpět. Pomáhají tak méně zkušeným návrhářům poučit se z chyb svých kolegů. Takové zkušenosti jsou někdy cennější než zkušenosti obsažené v klasických vzorech pro hledání správného řešení. Důvod je ten, že se mnoho antivzorů na povrch jeví jako vzory přinášející to nejlepší řešení dané situace. Skrývají v sobě ale problém, který méně zkušený návrhář nevidí. Často se antivzor osvědčí při řešení konkrétní specifické situace a v návrháři tak vzniká pocit, že bude možné tento vzor použít i obecně.

Antivzory se velmi často vyskytují v případech, kdy návrhář nemá dostatek zkušeností s technologií, pro kterou je aplikace vyvíjena. Konkrétní technologie v sobě často skrývají detaily, kvůli jejichž neznalosti může návrhář vnést do aplikace chyby či omezení brzdící další vývoj aplikace. Je nutné upozornit, že řešení definované antivzorem nemusí být vždy špatné a cílem antivzoru není odradit od použití příslušného řešení. Cílem antivzoru je upozornit na úskalí, která mohou použitím tohoto řešení vzniknout a dále je na návrháři, aby posoudil, zda jsou daná úskalí pro konkrétní aplikaci aktuální či nikoli.

V v této kapitole si představíme několik antivzorů, které se týkají návrhu webových aplikací. Začneme dvěma architektonickými antivzory, poté si uvedeme jeden prezentační antivzor a na závěr dva EJB antivzory. První dva antivzory jsme čerpali z [3], další pak z [2].

Název vzoru: Sumo Marriage
Klasifikace: Architectural Antipattern
Antivzor *Sumo Marriage* se aplikuje při situaci, kdy dojde k těsnému propletení „tlustého“ klienta s „tlustou“ databází[1], čímž vzniká velice neflexibilní aplikace. Při snaze přesunout takovouto aplikaci na Internet se jeví jako jediné řešení přepsání aplikace. Antivzor *Sumo Marriage* navrhuje místo přepsání oddělení prezentační vrstvy, aplikační logiky a databáze do samostatných vrstev (tedy „rozvod“). Toto řešení nemusí být vždy nejvhodnější. Je jen provizorní, protože pokud v původní aplikaci došlo k silnému provázání základních vrstev, je zřejmé, že implementaci nepředcházel návrh. Proto je pravděpodobné, že rozdělení vrstev vnese do aplikace chyby. Rozumnějším řešením by bylo vyvinout aplikaci znovu s důrazem na kvalitní analýzu a návrh.

Název vzoru: Spaghetti Code
Klasifikace: Architectural Antipattern
Antivzor *Spaghetti Code* se týká spíše fáze implementace. Je však hojně využíván při vývoji webových aplikací, proto si ho zde představíme také. Vzor jednoduše říká: „*Pokud nemáte dostatek času na kvalitní strukturalizaci kódu, nestrukturujte. Ostatní to také nedělají.*“ Tento vzor je uplatňován zejména při používání skriptovacích jazyků (PHP, ASP). K použití vzoru vede několik faktorů. Těmi nejčastějšími jsou stísněné termíny a nízký rozpočet. Antivzor je obvykle používán v menších projektech, které realizuje jedna osoba, což je u webových projektů poměrně časté.

Název vzoru: The Magic Servlet
Klasifikace: Presentation Tier Antipattern
Antivzor *The Magic Servlet* je specifikem J2EE prostředí. Rozšířil se díky zavedení JDBC pro obsluhu přístupu k databázi. Vzor *The Magic Servlet* říká: „Když je možné tak složitou věc jako je přístup k databázi vyjádřit pomocí jednoho řádku kódu, proč rozdělovat kód do více servletů. Kódu v servletu přece nebude mnoho.“ Na základě tohoto antivzoru je často použit jeden servlet pro obsluhu mnoha různých požadavků. Na první pohled se to zdá jako jednoduché přehledné řešení. Při bližším pohledu však zjišťujeme, že velikost servletu často přeroste únosnou mez, což je kritické pro jeho udržovatelnost a opakované použití.

Název vzoru: Everything Is an EJB
Klasifikace: EJB Antipattern
Antivzor *Everything Is an EJB* je speciální variantou známého antivzoru *Golden Hammer*, který říká, že když objevíte vhodný nástroj, máte ho používat na všechny problémy, bez ohledu na to, že mohou existovat jiná vhodnější řešení. Přestože nesprávnost této myšlenky je zřejmá, mnoho návrhářů se tímto vzorem řídí z důvodu své pohodlnosti objevovat vhodnější řešení. Podobným zvykem je i používání objektů *Entity Bean* na ukládání všech dat, která projdou systémem. V některých případech je to sice chytré, často se to však podobá příslovečnému *kanónu na vrabce*. Splní svůj účel, ale zatíží svou zbytečnou složitostí výkon systému. Proto je vhodné před použitím každého objektu *Entity Bean* zvážit, zda výhody jeho zavedení převáží nevýhody.

[1] databáze obsluhuje část aplikační logiky v podobě triggerů a vnitřních procedur

Název vzoru: Round–Tripping
Klasifikace: EJB Antipattern
Antivzor *Round–Tripping* se týká situace, kdy je přes síť přenášeno velké množství dat, která jsou posílána v samostatných požadavcích. Antivzorem je tuto situaci ignorovat s vysvětlením, že to není podstatný problém, který by bylo třeba řešit. To však není pravda. Neopodstatněné zatížení sítě a výkonu aplikace je vždy atribut, který by měl být zohledňován. Řešení situace vzniklé tímto antivzorem nabízí vzor *Data Transfer Object* představený v kapitole 2.4. Tento vzor navrhuje používat pro přenos dat speciální objekt, který data přenese v seskupené podobě.

Kapitola 3

Ukázka použití vzorů na ilustrační aplikaci

Pro ukázku vybraných analytických a návrhových vzorů jsme zvolili aplikaci představenou v knize [1] pod názvem *Ollie's Order Centre*. V následujícím textu se nejprve seznámíme s popisem aplikace a jejím analytickým modelem. Dále tento model rozšíříme o vhodné analytické vzory a poté si přiblížíme nasazení několika návrhových vzorů.

3.1 Popis aplikace

Popis aplikace se skládá ze tří částí. Začneme popisem současného systému přijímání a vyřizování objednávek, který je založen jen na bázi telefonu, faxu, tužky a papíru. Dále nadefinujeme cíl zavedení informačního systému pro podporu chodu objednávkového centra. Nakonec vyspecifikujeme rysy tohoto systému. Popis aplikace je stručným výtahem z [1], kde lze získat detailnější informace o analyzovaném systému.

Vzhledem k tomu, že popis aplikace uvádíme především z důvodu snazšího pochopení celkového analytického modelu, uvádíme na některých místech popisu v závorkách originální názvy popisovaných objektů. Tyto názvy korespondují s názvy použitými v modelech.

Současný stav

Objednávky na zboží jsou přijímány telefonicky operátorem (v [1] nazván *Ollie*), který tvoří prostředníka mezi zákazníkem a jedním z distributorů. Ollie přijímá objednávky pouze od zákazníků, kteří reprezentují nějakou organizaci (ne od samostatných osob). Po zapsání objednávky Ollie odfaxuje objednávku vybranému distributorovi, který zajistí doručení objednaného zboží z jednoho ze svých skladů. Sklad je vybrán na základě zákazníkovy doručovací adresy.

Po úspěšném doručení zboží zákazníkovi distributor odfaxuje do objednávkového centra informace o vyřízení objednávky. Ollie tyto informace převezme a předá účetnímu oddělení.

Z předchozího popisu vyplývá, že nás u každé objednávky (*Order*) zajímají informace o zákazníkovi jakožto organizaci (*Customer*), osobě, která objednávku v zastoupení zákazníka vyřizovala (*CustomerContact*), zvoleném distributorovi (*Distributor*), osobě, která vyřizuje objednávku na straně distributora (*OrderClerk*), skladu, ze kterého má být zboží dodáno (*Warehouse*), a samozřejmě o objednaném zboží (*Item*).

Cíl zavedení informačního systému

Zvýšit efektivitu a přesnost příjímání objednávek, doručování zboží a placení za objednané zboží.

Rysy navrhovaného systému

Rysy navrhovaného systému můžeme podle [1] rozdělit do čtyř skupin. První skupina charakterizuje důležité informace, které chceme v systému evidovat, druhá obsahuje provozní funkce, třetí statistické funkce pro podporu analýzy obchodních výsledků a poslední skupina popisuje interakce s dalšími systémy.

Evidence důležitých informací

- zboží a jeho ceny (*Item*)
- daňové sazby (*TaxCategory*)
- operátoři distributora (*OrderClerk*)
- sklady (*Warehouse*)
- zákazníci (*Customer*)
- obsah jednotlivých skladů (*WarehouseLineItem*)
- objednávky a jejich stavy (*Order*)

Provozní funkce pro podporu obchodu

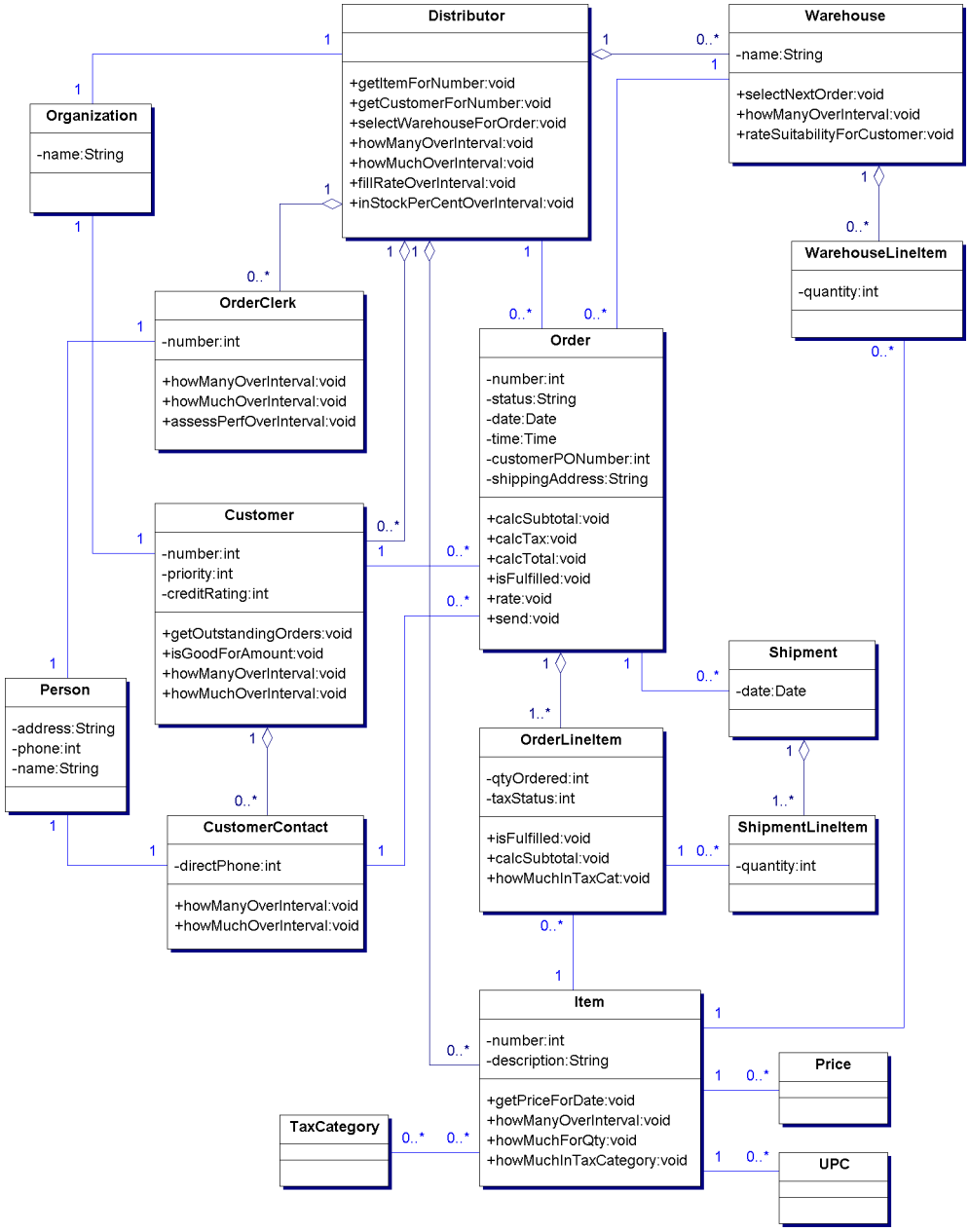
- výpočet celkové ceny objednávky
- dílčí výpočet ceny objednávky (mezisoučet pro vybrané zboží, výše daně)
- výběr objednávky k vyřízení (na základě priority zákazníka a data zadání objednávky)

Analýza obchodních výsledků

- výpočet výkonnosti distributora
- výpočet výkonnosti operátora
- výpočet vytížení skladu

Interakce s dalšími systémy

- skladový systém
  výstup: objednávky
  vstup: doručení objednávky
- účetní systém
  výstup: informace o objednávkách a jejich doručení zákazníkovi



Obrázek 3.1: Analytický model aplikace Ollie's Order Centre

Výsledný model aplikace

Na obrázku 3.1 je uveden analytický model aplikace *Ollie's Order Centre* převzatý z knihy [1] a převedený do syntaxe UML.

3.2 Použití analytických vzorů

Pro nasazení analytických vzorů zvolíme metodu dosazování vzorů do výsledného analytického modelu. Budeme tedy systematicky procházet všechny analytické vzory (v našem případě pouze vzory představené v kapitole 1.2) a hledat v nich podobnost s částmi navrhované aplikace. Nasazování analytických vzorů přináší mnoho doplňujících otázek, které nám pomohou zacelit znalostní mezery v doménové oblasti, které bychom jinak nemuseli objevit. Analytické modely znázorňující postupné nasazování popisovaných analytických vzorů jsou uvedeny v příloze A. Je proto vhodné číst následující text spolu s nahlížením do této přílohy.

Accountability

Vzor Party

Pokusme se v analytickém modelu najít adepty na vzor *Party*, tj. dvojice *Person–Organization*. Na první pohled vidíme 3 adepty: *CustomerContact–Customer*, *OrderClerk–Distributor* a *Person–Organization*. Žádná z těchto dvojic se však pro použití vzoru *Party* nehodí, protože se nepodílí na společných operacích. Dokonce lze nahlédnout, že model neumožňuje rovné postavení osoby a organizace. Nabízí se tedy otázka, zda může nastat reálná situace, kdy je rovné postavení osoby a organizace nutné. Touto situací může být objednávka zboží samostatnou osobou, která nereprezentuje žádnou organizaci. V tom případě bychom pohlíželi na zákazníka rovným způsobem bez ohledu na to, zda je osobou či organizací.

Otázka analytika: Přemýšlíte do budoucna o zpřístupnění objednávek i samostatným osobám, které nereprezentují žádnou organizaci?
Odpověď: Ano, v budoucnu tuto možnost pravděpodobně zavedeme.

Vytvoříme tedy nový objekt reprezentující zákazníka jakožto osobu (*CustomerPerson*) a nasadíme vzor *Party*. Účastníka, který tvoří nadtyp objektů *Customer* a *CustomerPerson*, nazveme *CustomerParty*. Zasazení instance vzoru do analytického modelu je zachyceno na obrázku 3.2 v příloze A.

Vzor Accountability

Vzhledem k tomu, že v modelu aplikace nepoužíváme žádnou formu organizační struktury, pro kterou by bylo vhodné zavést zodpovědnosti mezi její účastníky, vzor *Accountability* vynecháme.

Observations and Measurements

Vzor Quantity

Vzor *Quantity* je vhodný ve všech situacích, kdy u objektů zaznamenáváme kvantitativní charakteristiky v různých jednotkách. Jako objekt s několika kvantitativními charakteristikami se nám přímo nabízí objekt *Item*, který reprezentuje nabízené zboží.

Otázka analytika: Líbilo by se vám mít možnost zapisovat některé vlastnosti zboží ve více různých jednotkách?
Odpověď: Ano, tato možnost by byla dobrá například u váhy, kterou různí výrobci uvádějí v různých jednotkách.

Použijeme tedy vzor *Quantity* a seznam vlastností ještě doplníme o rozměry (výšku, šířku, hloubku) a cenu. Zasazení instance vzoru do analytického modelu je zachyceno na obrázku 3.3 v příloze A.

Vzor Conversion Ratio

Po zavedení vzoru *Quantity* se nabízí otázka, zda zavést také vzor *Conversion Ratio*, abychom umožnili provádění aritmetických a jiných operací s hodnotami vlastností zapsaných v různých jednotkách.

Otázka analytika: Napadá vás aritmetická nebo jiná operace, která by měla být prováděna na hodnotách zboží zapsaných v různých jednotkách?
Odpověď: Ano, například výpočet výsledné váhy objednaného zboží kvůli poštovnému.

Zavedeme tedy vzor *Conversion Ratio*. Zasazení instance vzoru do analytického modelu je zachyceno na obrázku 3.4 v příloze A.

Observations for Corporate Finance

Vzor Enterprise Segment

Vzor *Enterprise Segment* bychom použili v případě zájmu sledovat odbyt jednotlivého zboží. To však není cílem navrhovaného systému, a proto to ponecháme na informačních systémech jednotlivých distributorů, ve kterých to už pravděpodobně implementováno je.

Referring to Objects

Vzor Name

Pro případ nasazení tohoto vzoru budeme v modelu hledat objekty, u kterých by mohlo být vhodné použití více jmen. Opět se nabízí objekt *Item*. Ten už dokonce více identifikačních jmen má. Je to jednak jeho *číslo (number)* a jednak jeho *UPC (Uniform Product Code)*. Dokonce můžeme říct, že tato dvě jména reprezentují dvě identifikační schémata, čímž se dostáváme k následujícímu vzoru.

Vzor Identification Scheme

Vzhledem k tomu, že k pojmenování zboží používáme více identifikačních schémat, zavedeme vzor *Identification Scheme*. Situace by byla ještě zajímavější v případě, že by jedno zboží mohlo být dodáváno více distributory.

Otázka analytika: Stává se někdy, že jedno zboží dodává více distributorů?
Odpověď: Ano, ale vzhledem k tomu, že má toto zboží často odlišné pojmenování, považujeme ho za různé zboží.

Sjednotíme stejné zboží nabízené různými distributory pomocí identifikačního schématu, které vyjadřuje pojmenování a identifikaci vybraným distributorem. Zasazení instance vzoru do analytického modelu je zachyceno na obrázku 3.5 v příloze A.

Inventory and Accounting

Vzor Account

Vzor *Account* použijeme v případě, že v modelu objevíme kontejnery, u kterým máme zájem o vedení evidence změn jejich obsahu. Takovým kontejnerem je sklad, jehož obsah se může měnit s prodejem zákazníkovi, nákupem distributorem nebo přesunem mezi sklady. Zasazení instance vzoru do analytického modelu je zachyceno na obrázku 3.6 v příloze A.

Vzor Transactions

Vzor *Transaction* použijeme v případě, že ke změně obsahu skladu dochází přesunem z jiného skladu a my chceme evidovat z jakého. To by však znamenalo zavedení dvou nových abstraktních skladů s neomezenou kapacitou reprezentujících prodej zákazníkovi a naplnění distributorem.

Otázka analytika: Bylo by pro vás zajímavé evidovat, zda ke změně obsahu skladu došlo nákupem, prodejem nebo přesunem mezi sklady?
Odpověď: Ne, to jsou interní informace distributora, které nám nesděluje.

Vzor *Transaction* proto vynecháme.

Planning

Vzor Proposed and Implemented Action

Pokusme se v modelu najít všechny akce, které by mohlo být vhodné plánovat. Je to *objednávka (Order)* a *doručení objednávky (Shipment)*. Podívejme se blíže na stavy, kterými tyto akce prochází.

- Order
  – objednávka přijata od zákazníka
  – objednávka převzata k vyřízení a předána distributorovi
  – objednávka převzata distributorem k vyřízení
- Shipment
  – zboží odesláno zákazníkovi
  – zboží doručeno a přijato zákazníkem

Lze nahlédnout, že dvojice *Order* a *Shipment* společně tvoří akci, kterou můžeme nazvat koupě zboží (*Purchase*). Tato akce přebírá stavy obou původních akcí.

- Purchase
  – objednávka přijata od zákazníka
  – objednávka převzata k vyřízení a předána distributorovi
  – objednávka převzata distributorem k vyřízení
  – zboží odesláno zákazníkovi
  – zboží doručeno a přijato zákazníkem

Ze stavů akce *Purchase* můžeme odvodit, že první stav je plánem akce a druhý stav je začátkem realizace akce. Tento i následující vzor nasadíme do modelu spíše z ilustrativního důvodu. Objednávka totiž neobsahuje podstatné vlastnosti, které by se mohly lišit u naplánované a realizované varianty objednávky. Takovou vlastností by mohlo být plánované

datum doručení objednaného zboží. Zasazení instance vzoru do analytického modelu je zachyceno na obrázku 3.7 v příloze A.

Vzor Completed and Abandoned Actions

Zamysleme se nyní nad tím, zda je v naší situaci vhodné rozeznávat u koupě zboží i dokončené a zrušené koupě. Je zřejmé, že akci *Purchase* prohlásíme za dokončenou po průchodu posledním stavem. Umožňujeme však zrušení akce? V současném modelu ne. Pokud chceme zrušení akce umožnit, budeme muset zavést nový stav a najít místa v posloupnosti stavů, na kterých může tento stav nastat.

- Purchase
  – objednávka přijata od zákazníka
  – [objednávka zrušena]
  – objednávka převzata k vyřízení a předána distributorovi
  – [objednávka zrušena]
  – objednávka převzata distributorem k vyřízení
  – [objednávka zrušena]
  – zboží odesláno zákazníkovi
  – zboží doručeno a přijato zákazníkem

Zasazení instance vzoru do analytického modelu je zachyceno na obrázku 3.8 v příloze A.

Trading

Vzor Contract

Vzor *Contract* se v našem modelu již vyskytuje. Kontraktem je v tomto případě objednávka, kupujícím je zákazník (*CustomerParty*) a prodávajícím distributor (*Distributor*). Obchodovaným artiklem je zboží (*Item*). Vymezení instance vzoru v rámci analytického modelu je zachyceno na obrázku 3.9 v příloze A.

Vzor Portfolio

Vzor *Portfolio* použijeme v případě, že chceme definovat podmínky, podle kterých můžeme ze všech kontraktů vybírat jen ty se zadanými vlastnostmi.

Otázka analytika: Bylo by pro vás výhodné definovat si podmínky, podle kterých můžete následně vybírat vyhovující objednávky?
Odpověď: Taková funkce by pro nás byla určitě zajímavá.

Zavedeme tedy vzor *Portfolio*. Zasazení instance vzoru do analytického modelu je zachyceno na obrázku 3.10 v příloze A.

Derivative Contracts

Vzor Forward Contracts

Použití vzoru *Forward Contracts* se přímo nabízí pro umožnění sjednání objednávky s časovým předstihem.

Otázka analytika: Chtěli byste umožnit svým zákazníkům zadávat objednávky s časovým

předstihem? Například 2 měsíce před plánovanou realizací objednávky.
Odpověď: Ano, už to po nás dokonce někteří zákazníci vyžadovali.

Zavedeme tedy vzor *Forward Contracts*. Zasazení instance vzoru do analytického modelu je zachyceno na obrázku 3.12 v příloze A.

3.3 Použití návrhových vzorů

Při ukázce použití návrhových vzorů bohužel není možné nabídnout pohled na zasazený vzor v kontextu celé aplikace. Důvod je ten, že na rozdíl od analytického modelu je návrhový model velice rozsáhlý. V této kapitole proto budeme postupovat tak, že si nastíníme některé problémy, které mohou při návrhu ukázkové aplikace vyvstat, a ukážeme řešení těchto problému pomocí návrhových vzorů v úzkém kontextu několika tříd. Modely jednotlivých řešení jsou uvedeny v příloze B. Je proto vhodné číst následující text spolu s nahlížením do této přílohy.

Návrhové vzory GoF

Vzor Adapter

Problém: Chceme v systému umožnit funkci vyhledání zboží podle zadaného jména. K těmto účelům nabízíme rozhraní *SearchByName*, které definuje metodu `searchByName(value:String)`. Z dřívějších projektů máme naprogramovanou třídu *ItemDM*[1] realizující efektivní vyhledávání zboží v databázi. Tato třída však implementuje obecnější metodu `search(atribut:Atribut, value:String)`, která vyhledává zboží podle hodnoty zadané vlastnosti. Hledáme tedy řešení, jak tuto třídu zapojit do našeho systému.

Řešení: Využijeme vzor *Adapter (variantu Object)*. Díky tomu přesměrujeme volání metody `searchByName(myValue)` na nový objekt pomocí `itemDM->search(name, myValue)`. Výsledná instance vzoru je na obrázku 3.13 v příloze B.

Vzor Facade

Problém: Rozhodli jsme se vytvořit k aplikaci Ollie's Order Centre tenkého webového klienta, přes nějž budou moci zákazníci zadávat objednávky sami přes webové rozhraní. Tomuto klientovi však chceme zpřístupnit pouze omezenou sadu funkcí a viditelně ho oddělit od zbytku aplikace.

Řešení: Použijeme vzor *Facade*. Nadefinujeme rozhraní zpřístupňující pouze několik nezbytných funkcí jako je výpis zboží, vyhledání zboží na základě jména a zadání objednávky a k tomuto rozhraní připojíme tenkého webového klienta. Výsledná instance vzoru je na obrázku 3.14 v příloze B.

Vzor Proxy

Problém: Když se zamyslíme nad funkcemi poskytovanými třídou *Warehouse* (především nad funkcí `selectNextOrder`), musí nás napadnout, že je objednávková aplikace nemá právo implementovat. Správně by měla implementaci těchto funkcí přenechat objektům

[1] ItemDM je třída pro správu objektů třídy Item, zkratka DM zastupuje výraz Data Manager

skladového systému (konkrétního distributora), se kterým spolupracuje. Volání metody třídy *Warehouse* objednávkovou aplikací by tedy mělo být přesměrováno na objekt skladového systému realizující tyto funkce.

Řešení: Tuto situaci řeší vzor *Proxy*, díky kterému může být každý požadavek na volání metody třídy *Warehouse* přesměrován na objekt skladového systému. Po vrácení výsledku vzdáleným objektem je tento výsledek zobrazen v objednávkové aplikaci.

Výsledná instance vzoru je na obrázku 3.15. Pro přehlednost jsme sklad objednávkové aplikace reprezentující Proxy objekt nazvali *WarehouseOA*[2] a sklad skladového systému *WarehouseWS*[3].

Vzor Observer

Podívejme se pro účely představení tohoto vzoru na objednávkovou aplikaci z pohledu skladového systému, se kterým aplikace spolupracuje. Teoreticky může být k jednomu skladovému systému připojeno více objednávkových aplikací (různých společností) realizujících příjímání objednávek a následný prodej zboží.

Problém: Každá objednávková aplikace potřebuje v libovolný okamžik znát přesný počet kusů konkrétního zboží na skladu skladového systému, aby nepřijala objednávku na zboží, které není na skladě.

Řešení: To lze řešit pomocí vzoru *Observer*, který umožňuje na každou položku skladu (*WarehouseLineItemWS*), evidující ve skladovém systému množství konkrétního zboží na vybraném skladě, připojit pozorovatele jednotlivých objednávkových aplikací (*WarehouseLineItemOA*). Výsledná instance vzoru je na obrázku 3.16 v příloze B.

Návrhové vzory POSA

Vzor Reactor

Problém: Řešíme problém, jak implementovat správu požadavků přicházejících k objednávkové aplikaci od více webových klientů (zákazníků prohlížejících sortiment a realizujících objednávky na webových stránkách).

Řešení: Jako možné řešení můžeme použít vzor *Reactor*. Vzhledem k tomu, že se tento vzor týká pouze implementačních detailů, bude jeho instance vypadat stejně jako vzor samotný s výjimkou tříd *ConcreteHandler*, za něž lze dosadit všechny možné správce událostí od správy událostí GUI prvků po události, jako je přihlášení uživatele nebo odeslání objednávky.

Návrhové vzory EJB

Následující vzory se týkají případu, že bude při implementaci použita architektura EJB.

Vzory Session Facade a Business Delegate

Problém: Chceme efektivně oddělit aplikační logiku od klientské prezentační vrstvy a serverové datové vrstvy (Entity Bean objekty).

[2] zkratka OA vychází z výrazu Order Application
[3] zkratka WS vychází z výrazu Warehouse System

Řešení: Na základě vzoru *Session Facade* zavedeme mezi klienta a vrstvu *Entity Bean* objektů novou vrstvu *Session Bean* objektů obalující *Entity Bean objekty* a kontrolující přístup k nim. Na obrázku 3.17 je možný příklad propojení klienta a serveru přes *Session Facade*. Na obrázku 3.18 je pak rozšíření tohoto modelu o vzor *Business Delegate*.

Vzor Data Transfer Object

Problém: Pro zobrazení webové stránky s výpisem aktuální nabídky zboží je třeba přenést ze serveru data týkající se nabízeného zboží. Otázkou je, jak tato data přenést co nejefektivněji vzhledem k zatížení aplikace a přenosové sítě.

Řešení: Použijeme vzor *Data Transfer Object*. Definujeme si zvláštní objekt s názvem *ListOfItemsDTO*. V prvním kroku do tohoto objektu načteme z databáze data týkající se všeho zboží, které má být zobrazeno. Na straně klienta pak tento objekt rozložíme a zboží vypíšeme na webovou stránku.

Vzory webových služeb

Následující vzor se týká případu, že budou při implementaci využity webové služby.

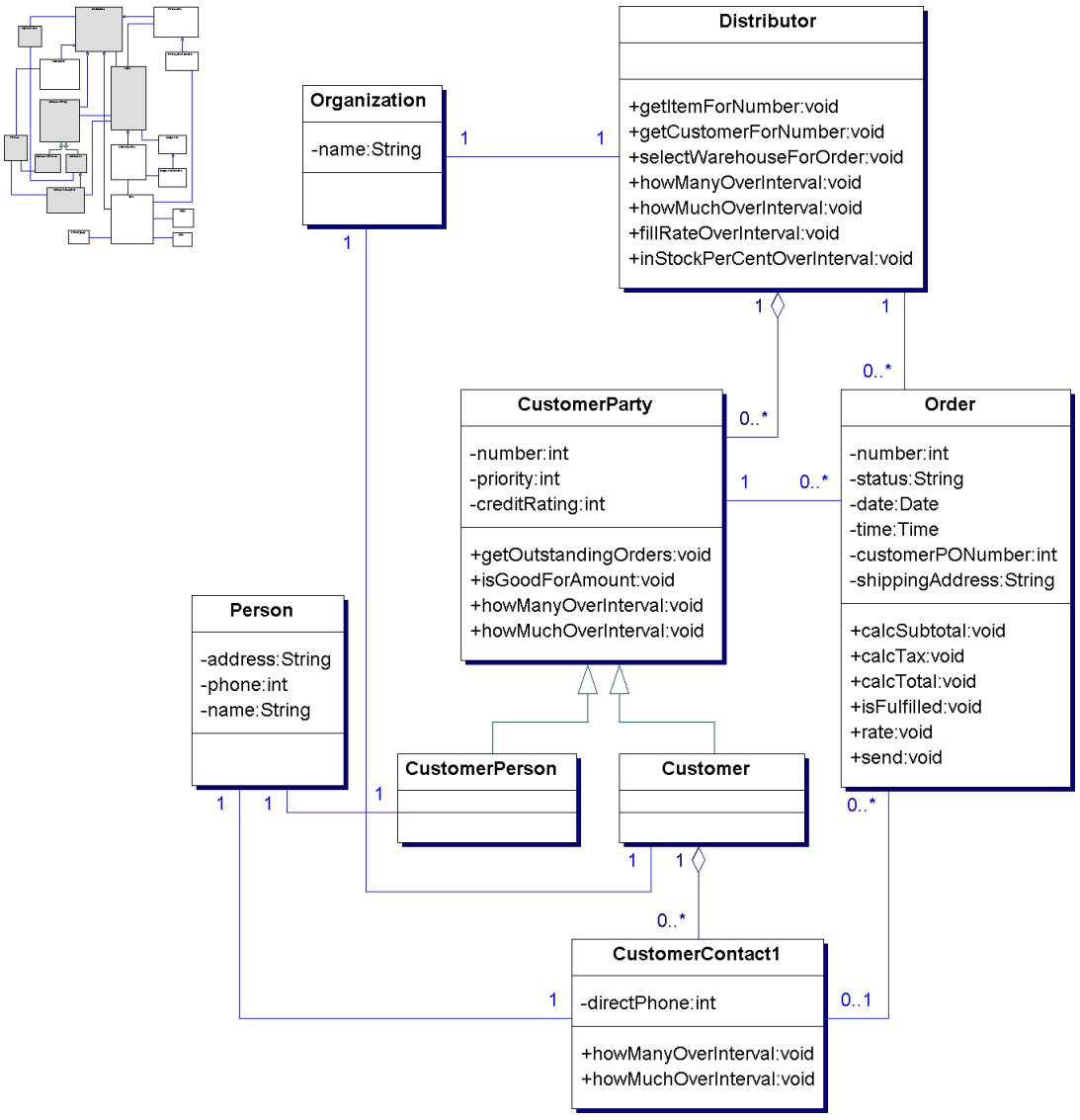
Vzor Service Factory

Problém: Představme si situaci, že máme skladové systémy jednotlivých distributorů připojeny k objednávkové aplikaci ve formě webových služeb. Každý distributor tedy poskytuje webovou službu umožňující jednotlivým objednávkovým systémům komunikovat s jeho skladovým systémem a získávat aktuální informace o jeho změnách. Pak se nabízí otázka, zda by bylo možné vyhledávat nové distributory a připojovat jejich skladové systémy pomocí webových služeb do naší objednávkové aplikace dynamicky za běhu.

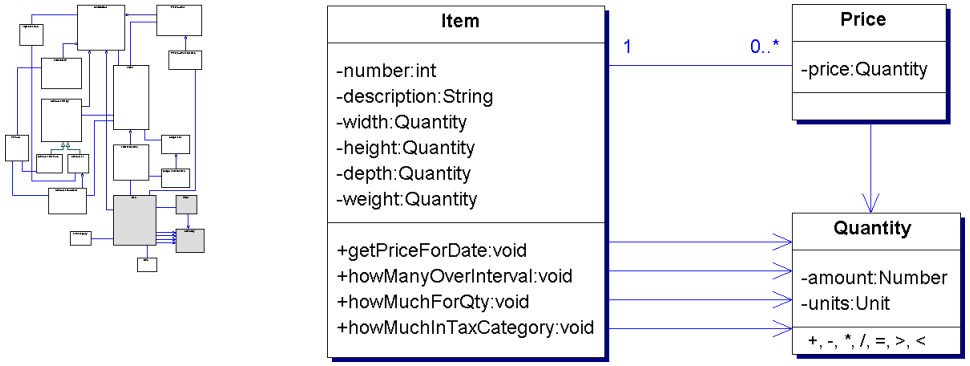
Řešení: Odpověď na tuto otázku je kladná, návod nabízí vzor *Service Factory*. Výsledná instance vzoru je na obrázku 3.19 v příloze B.

Příloha A

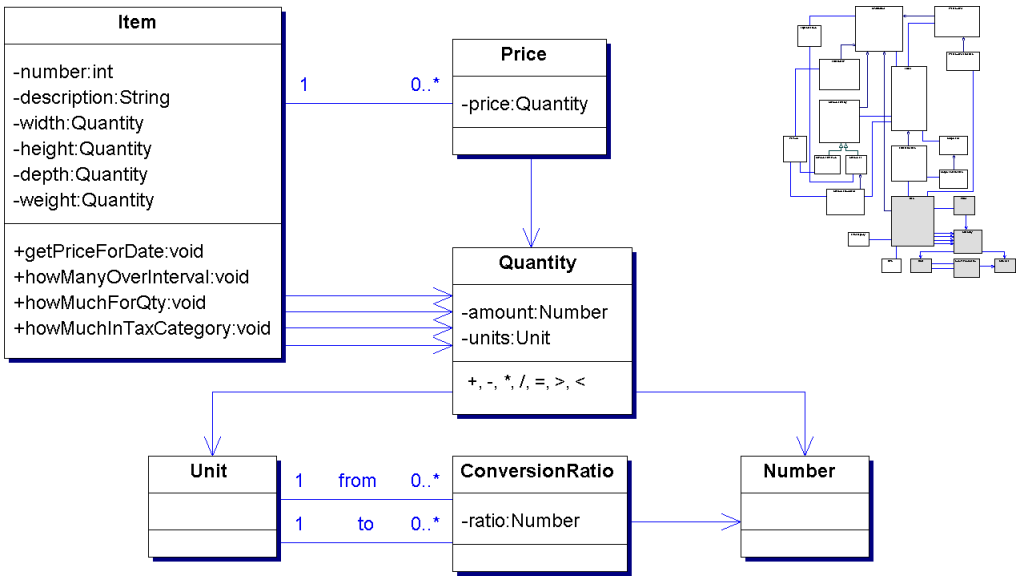
Příloha obsahuje analytické modely ilustrující postupné nasazování analytických vzorů popsané v kapitole 3.2.



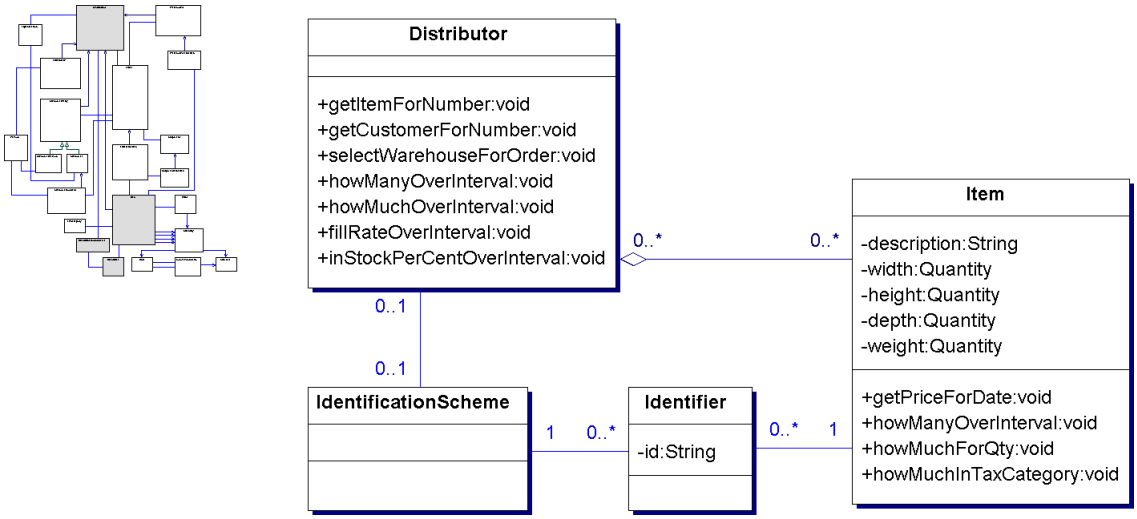
Obrázek 3.2: Zasazení vzoru Party do analytického modelu



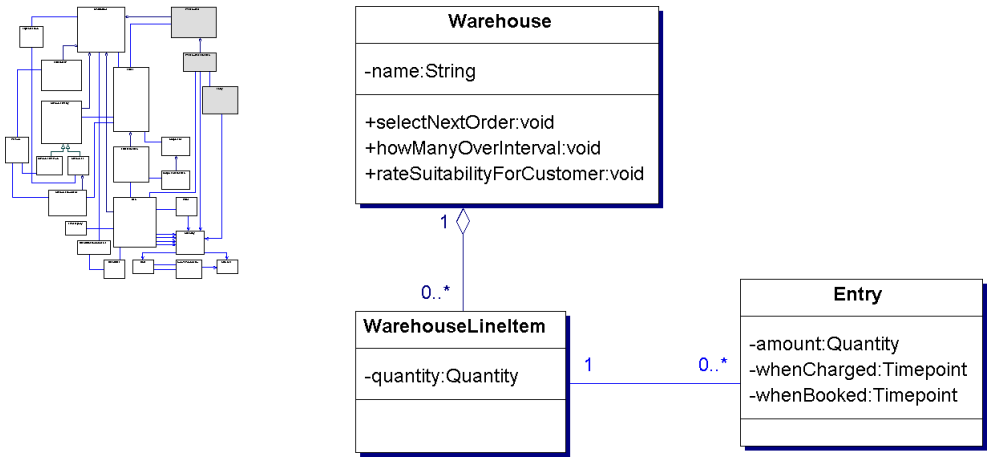
Obrázek 3.3: Zasazení vzoru Quantity do analytického modelu



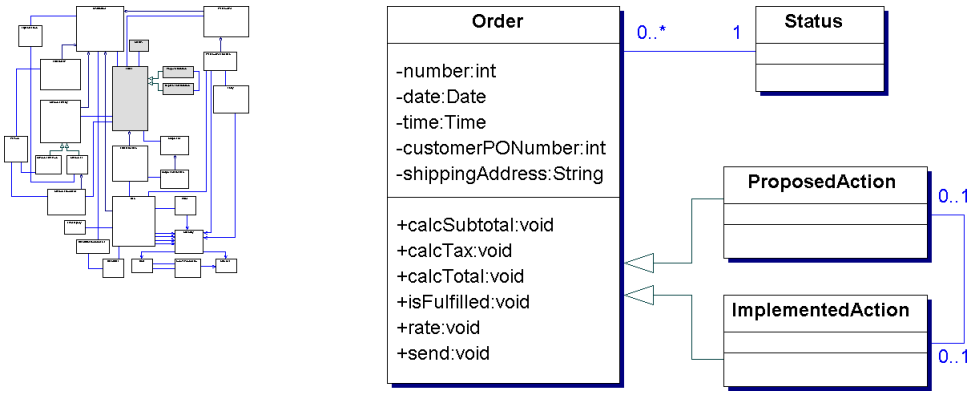
Obrázek 3.4: Zasazení vzoru Conversion Ratio do analytického modelu



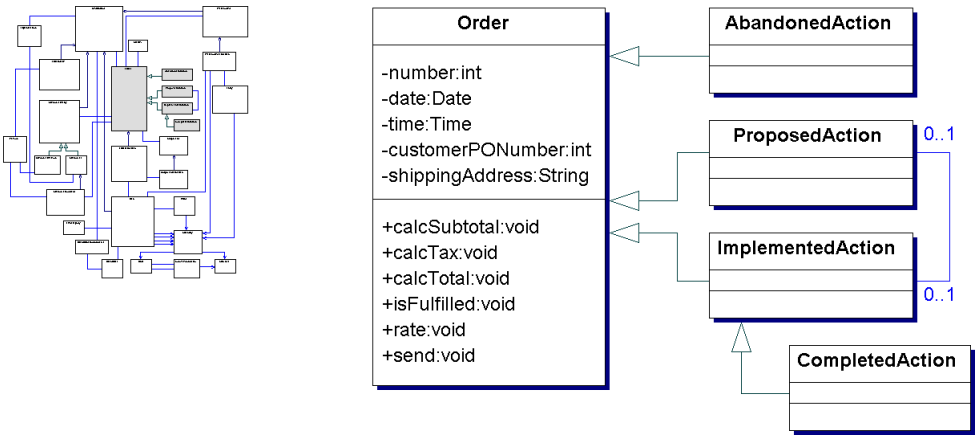
Obrázek 3.5: Zasazení vzoru Identification Scheme do analytického modelu



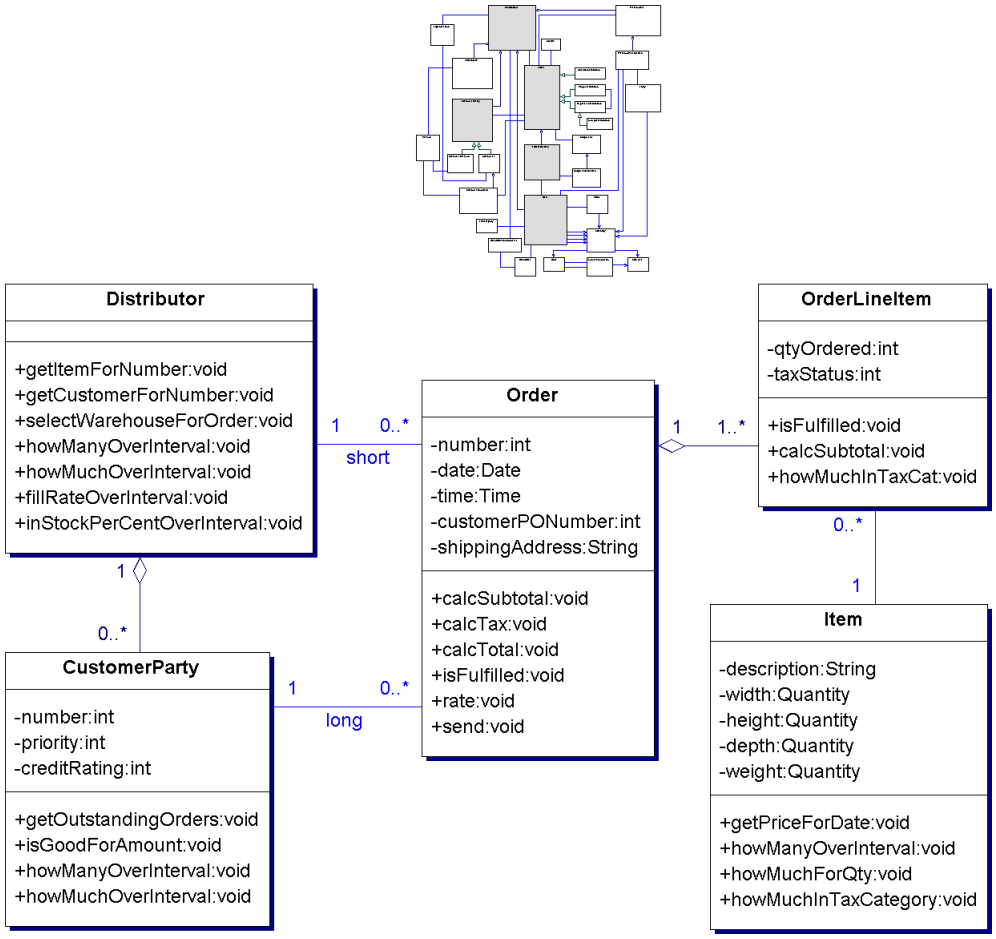
Obrázek 3.6: Zasazení vzoru Account do analytického modelu



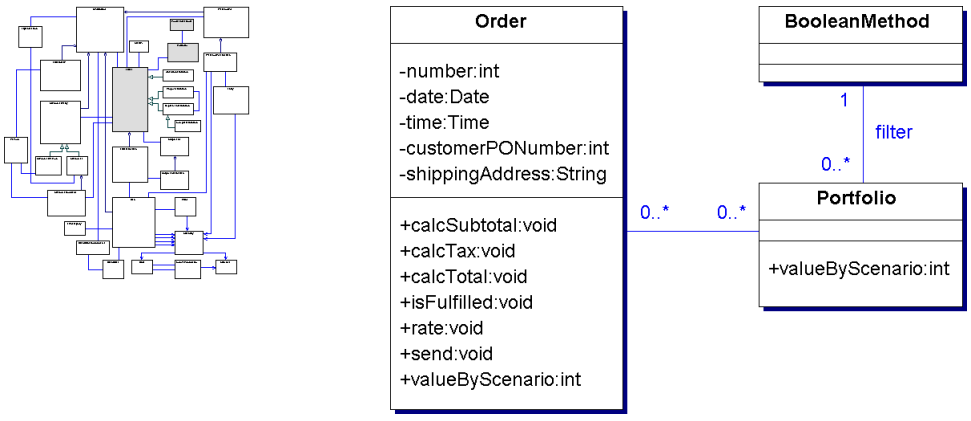
Obrázek 3.7: Zasazení vzoru Proposed and Implemented Action do analytického modelu



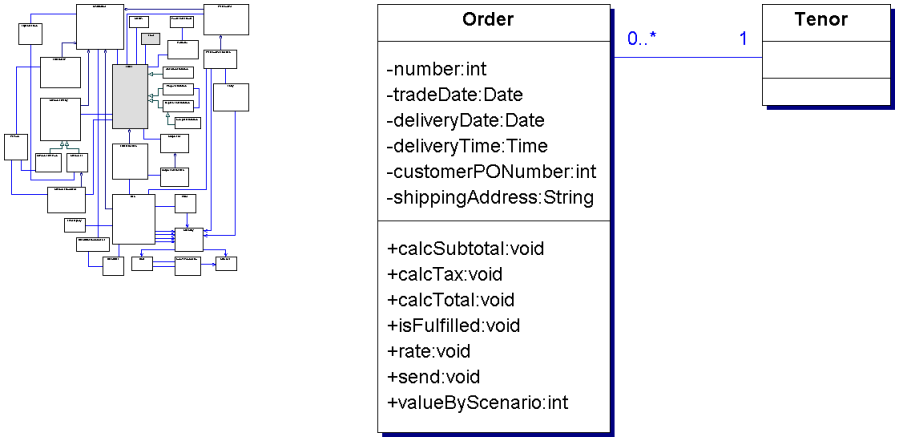
Obrázek 3.8: Zasazení vzoru Completed and Abandoned Actions do analytického modelu



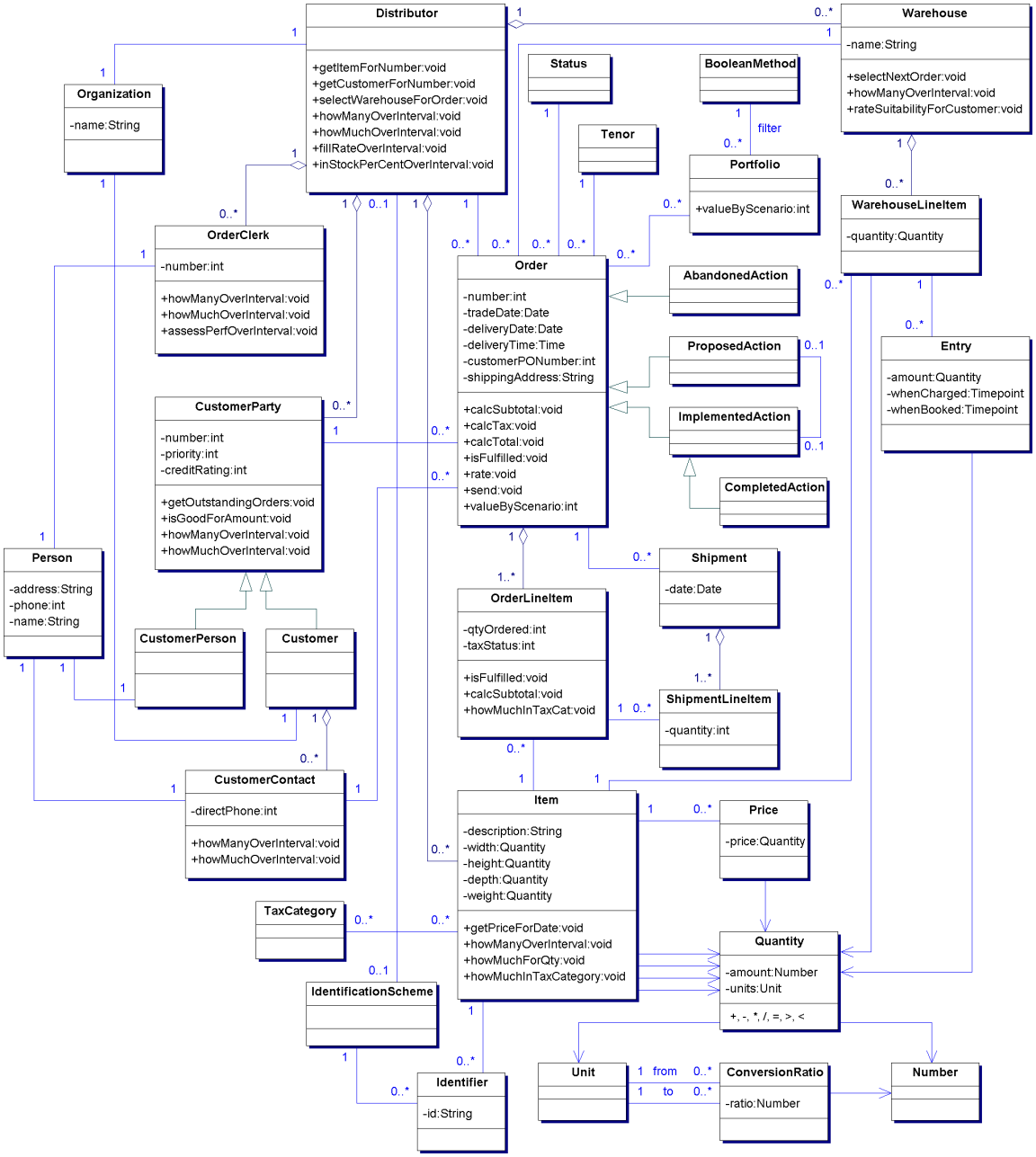
Obrázek 3.9: Zasazení vzoru Contract do analytického modelu



Obrázek 3.10: Zasazení vzoru Portfolio do analytického modelu



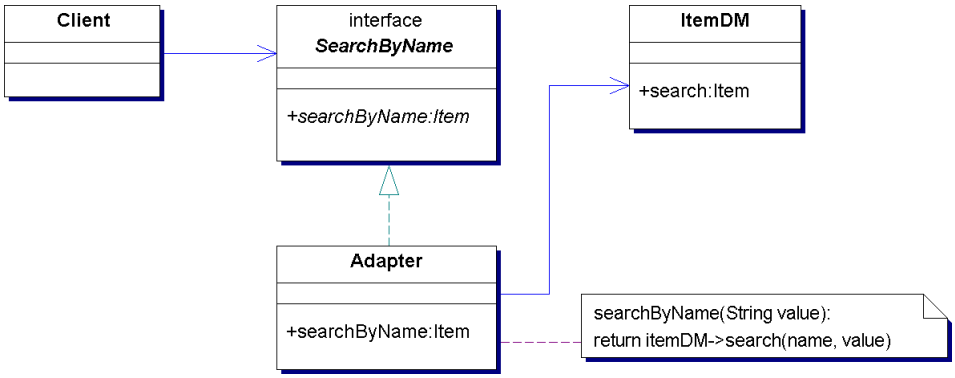
Obrázek 3.11: Zasazení vzoru Forward Contracts do analytického modelu



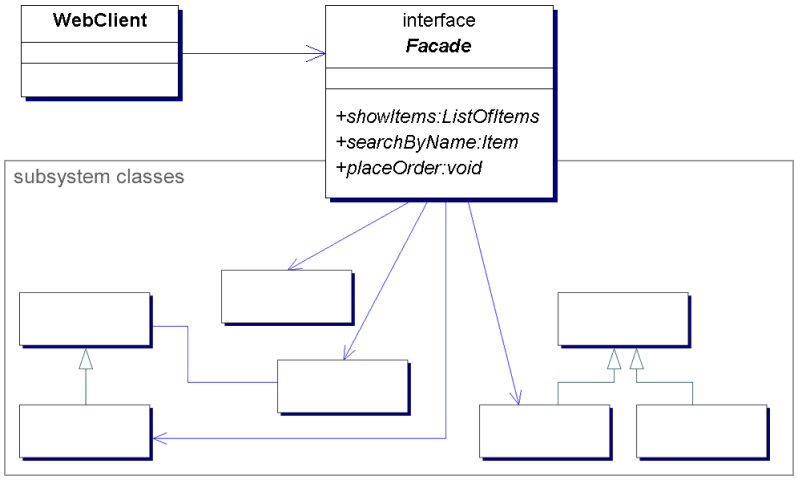
Obrázek 3.12: Výsledný model po nasazení analytických vzorů

Příloha B

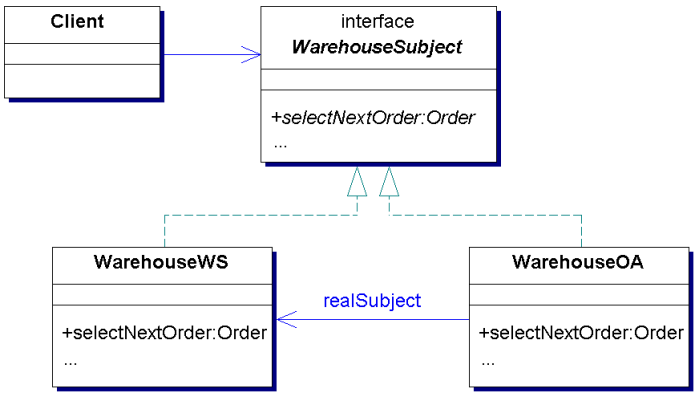
Příloha obsahuje modely instancí návrhových vzorů použitých při řešení návrhových problémů v kapitole 3.3.



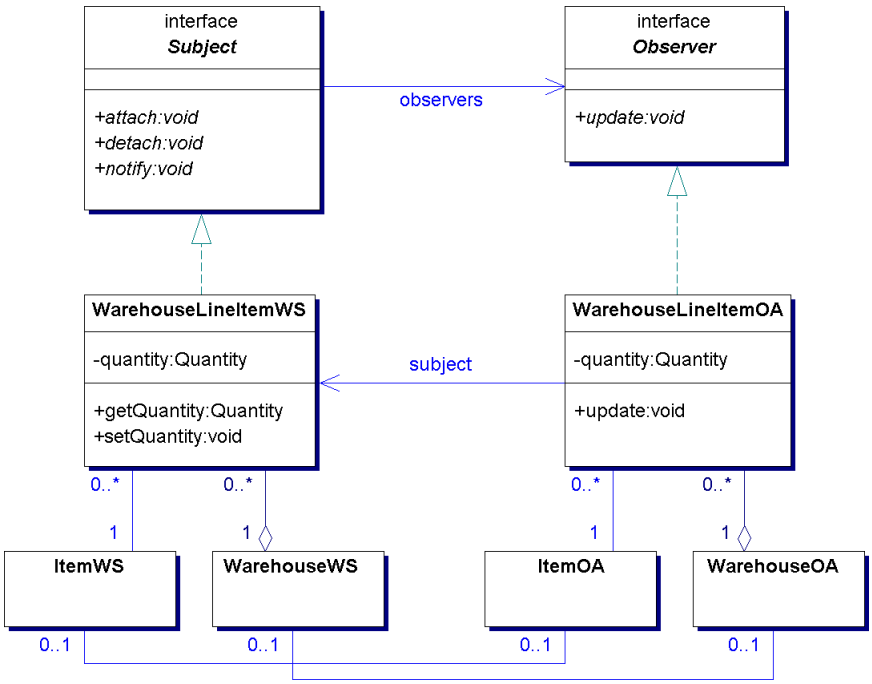
Obrázek 3.13: Instance návrhového vzoru GoF – Adapter, varianta Object



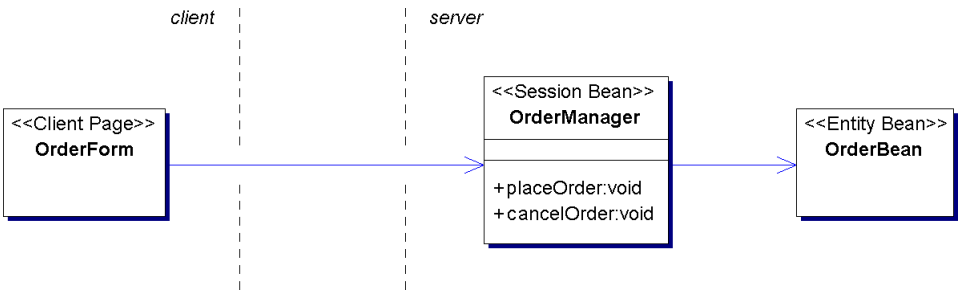
Obrázek 3.14: Instance návrhového vzoru GoF – Facade



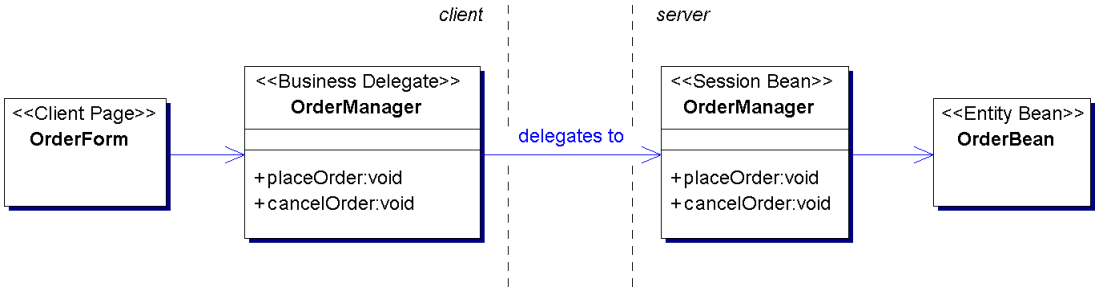
Obrázek 3.15: Instance návrhového vzoru GoF – Proxy



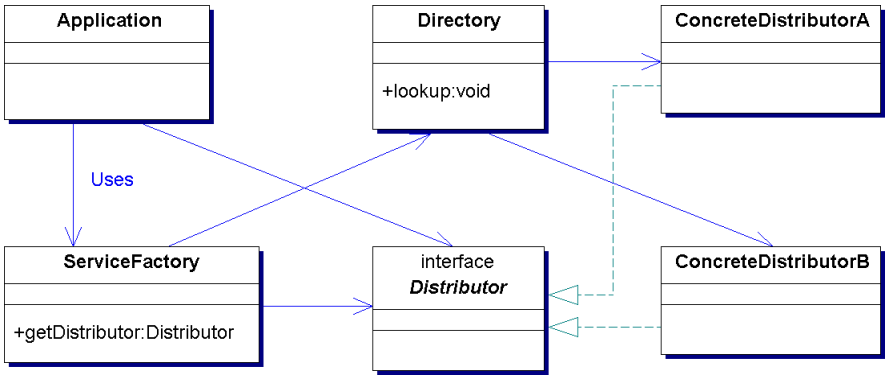
Obrázek 3.16: Instance návrhového vzoru GoF – Observer



Obrázek 3.17: Příklad použití návrhového vzoru EJB – Session Facade



Obrázek 3.18: Příklad použití návrhového vzoru EJB – Business Delegate



Obrázek 3.19: Instance vzoru webových služeb – Service Factory

Literatura

[1] Coad, P.: *Object Models – Strategies, Patterns and Applications.* USA, Upper Saddle River, Yourdon Press 1997.

[2] Crawford, W., Kaplan, J.: *J2EE Design Patterns.* USA, Sebastopol, O'Reilly & Associates 2003.

[3] Cunningham & Cunningham, Inc. – consultancy specialized in object-oriented programming. Slovník pojmů z OOP dostupný na http://c2.com/cgi/wiki?SearchWords (květen 2004).

[4] Fowler, M.: *Analysis Patterns – Reusable Object Models.* USA, Menlo Park, Addison-Wesley 1997.

[5] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns – Elements of Reusable Object-Oriented Software.* USA, Reading, Addison-Wesley 1995.

[6] Kraval, I.: *Design Patterns v OOP.* Elektronická kniha vydaná na http://www.objects.cz, 2002.

[7] Monday, P.: *Web Service Patterns: Java Edition.* USA, Berkeley, Apress 2003.

[8] Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: *Pattern-Oriented Software Architecture – Patterns for Concurrent and Networked Objects.* USA, New York, Wiley & Sons 2000.

[9] Yacoub, S., Ammar, H.: *Pattern-Oriented Analysis and Design – Composing Patterns to Design Software Systems.* USA, Boston, Addison-Wesley 2003.