Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Metody a nástroje pro tvorbu informačních systémů na internetu

Disertační práce

Autor: Ing. Ondřej Volráb
Školitel: Doc. Ing. Prokop Toman, CSc.
Obor: Informační management

Praha 2008

Děkuji svému školiteli Doc. Ing. Prokopovi Tomanovi, CSc. za odborné vedení této práce a Prof. RNDr. Jiřímu Vaníčkovi, CSc. za cenné konzultace.

Obsah

1. Úvod

Internetově orientované informační systémy představují zatím poslední evoluční stádium zdaleka nejen pro podnikové informační systémy. Jde bezesporu o výsledek dlouhodobých snah o integraci autonomních systémů, hledání efektivnějších způsobů řízení procesů a potřeb komunikace v reálném čase. Těmto trendům se přizpůsobují i metodiky vývoje informačních systémů a jejich architektury. Prosazování modelem řízené nebo servisně orientované architektury naznačuje, že ve vývoji software se stále více uplatňují postupy vedoucí k opakovatelně použitelným a vzájemně spolupracujících řešením se zachováním volnosti vnitřních vazeb. V doprovodu stále propracovanějších technik se tak daří v aplikované informatice uplatňovat inženýrské přístupy, které pomáhají vytvářet kvalitnější software a lépe řídit jeho vývoj.

Všeobecný rozvoj vědy, a především vývoj informatiky, otevřel mimo jiné nové možnosti v oblastech zpracovávání a využívání vstupních dat, které nebylo dříve možné v informačních systémech vhodně reprezentovat pro jejich nejednoznačnost. Přitom určitá míra nepřesnosti, neurčitosti a vágnosti je přirozeným rysem okolního světa i běžného jazyka a rozhodně se nejedná o nějakou nevýhodu, která by omezovala či znesnadňovala komunikaci. Zohlednění jistého typu nepřesnosti přináší výhody zejména v systémech, které jsou značně závislé na lidském faktoru. V uplynulých letech se podařilo tímto způsobem např. zvýšit bezpečnost brzdových systémů v autech, zefektivnit regulaci provozu japonského metra, automatizovat vyhledávání bodu pro zaostřování ve fotoaparátu, vyvinout systém pro rozpoznávání ručně psaného textu na mobilních zařízeních nebo zkvalitnit analýzu investic na kapitálových trzích.

S neustálým nárůstem objemu elektronicky zpracovávaných dat, které jsou často ve složitých vzájemných vztazích, je nutné je dlouhodobě uchovávat a zároveň k nim mít rychlý přístup se mění také role databázových systémů. Znovu se objevují výhody objektově orientovaných databázových systémů, které jsou známé sice řadu let, ale pro které zatím nebylo vhodné tržní prostředí. Obě vývojové cesty, tj. evoluční relačně-objektová i revoluční čistě objektová se stále sbližují a vytvářejí tak prostředí pro své využití ve specifických oblastech, jako jsou transakční cache mechanismy (vyrovnávací paměť), *embedded* (vestavěná/vnořená) databáze nebo in-memory (v paměti) databáze. Prosazují se samozřejmě i v oblastech datových

skladů a v dalších běžných typech transakčních databázových systémů, což potvrzují i přední místa v žebříčcích největších (objemem dat i počtem transakcí) databází na světě [Winter].

Propojením uvedených výsledků výzkumu a vývoje z oblasti reprezentace neurčitosti dat a jejich efektivního ukládání do databází je možné poskytnout informačním systémům kvalitnější vstupy pro rozhodování. Je však potřeba si uvědomit, že využití poznatků vědy v praxi je vždy omezováno dostupnými metodami a nástroji. Pojmem *dostupný* je zde myšlena nejen licenční politika a otevřenost veřejnosti, ale především srozumitelnost běžným uživatelům. V této oblasti je vhodné využívat pokud možno standardů pro formulaci myšlenek, řešení, jejich struktury, výhod i omezení. S pomocí běžně akceptovaných standardů jakými jsou UML nebo návrhové vzory je možné bariéry mezi výsledky výzkumu a praxí úspěšně překonávat, a tím nové myšlenky a postupy reálně začít využívat.

Název disertační práce odráží autorův všeobecný zájem o postupy aplikovatelné při vytváření a rozšiřování informačních systémů. Pro účely zlepšení konkrétní oblasti je však pozornost zúžena především na modelování neurčitosti. Naopak oproti původním předpokladům budou výsledky této práce z větší části využitelné nejen v oblasti internetově orientovaných informačních systémů.

1.1. Motivace

Téma disertační práce spadá zejména do oblasti internetových informačních systémů, resp. metod a nástrojů pro jejich tvorbu. Jedním z hlavních důvodů je rozšířenost tohoto typu systémů a lze tedy předpokládat, že i malé zlepšení v souvisejících postupech může mít široký pozitivní dopad. Dalším důvodem je relativně vysoký stupeň interakce těchto systémů s uživatelem a to je jistě oblast, kde jsou prostory pro zlepšení z pohledu návrhu systémů značné. V neposlední řadě jsou pro tuto oblast inspirativní některá její specifika, která vybízí k návrhům takových informačních systémů, které pokud možno oddělují svojí funkcionalitu, logiku a ukládaní dat od komunikace s uživatelem. Zároveň je pro ně typická rychlá změna v modelovaných procesech i datech, a tak jsou jejich tvůrci nuceni přicházet s opakovatelně využitelnými postupy a implementacemi.

Konkrétní oblast, kterou se tato práce zabývá, se týká modelování neurčitosti v prostředí objektově orientovaných databázových systémů. Vzhledem k tomu, že

obecně rozeznáváme více typů neurčitosti, je třeba na tomto místě uvést, že práce se zabývá neurčitostí, kterou lze vyjádřit jako kvantitativní míru splnění nějaké vlastnosti, kterou nevnímáme jako binární (dvouhodnotovou). Jde tedy o interpretaci výsledku nějakého jevu (fuzzy neurčitost). Neurčitost stochastického typu (pravděpodobnost, že jev nastane), případně kvantová neurčitost (ovlivnění stavu systému měřením výsledku) nejsou v této práci uvažovány.

Jak již bylo v úvodu práce uvedeno, jsou nám známy jak postupy pro modelování fuzzy neurčitosti, tak metody a nástroje pro ukládání komplikovaných dat velkých objemů. Propojující článek, mezi těmito oblastmi však dosud chybí. V současné době je tak využití principů fuzzy přístupu pro velké objemy dat značně omezeno. Touto problematikou se proto zabývá řada prací, které však zatím nenabídly uspokojivé řešení, které by nebylo pevně svázáno s konkrétní platformou, a které by zohlednilo dynamickou povahu kvantitativního hodnocení výsledků jevů. Společnou nevýhodou dosud navržených řešení je pak jejich forma, která není běžným pracovníkům v oboru informačních technologií (IT) dostatečně blízká.

Řešení uvedeného problému umožní tvůrcům široké škály informačních systémů rozšířit metody zpracování velkých objemů dat pomocí principů fuzzy logiky. Vzroste tím využitelnost dat a zároveň se bude moci rozhraní informačních systémů více přiblížit uživatelům, resp. jejich způsobu formulace požadavků. Zpřístupnění navrženého řešení ve všeobecně přijímané podobě jako jsou návrhové vzory [Alur 2003], [Gamma et al. 2003] navíc zlepší IT pracovníkům obecné povědomí o pokročilejších technikách zpracování dat.

Bezprostřední využitelnost výsledků této disertační práce je předpokládána zejména pro pracovníky v následujících rolích:

- business konzultant (povědomí o možnostech, které fuzzy přístup nabízí),
- analytik (konkrétní postup jak v IS modelovat fuzzy neurčitost),
- programátor (konkrétní způsob implementace se zachováním přenositelnosti).

Kromě uvedených rolí bude možné výsledky začlenit také přímo do dnes rozšířených architektur jako je MDA, kde mohou posloužit při konstrukci transformačních pravidel mezi jednotlivými úrovněmi modelu.

1.2. Cíle práce

Hlavním cílem disertační práce je **návrh obecně použitelného postupu pro aplikaci principů fuzzy logiky v objektově orientovaném databázovém systému ve formě návrhového vzoru**. Aplikace tohoto návrhového vzoru se předpokládá zejména v oblasti internetových informačních systémů, jejichž specifika budou v navrženém řešení zohledněna. Pro dosažení hlavního cíle byly stanoveny následující dílčí cíle:

1. V rámci rešeršní části zpracovat současný stav řešené problematiky a získat tak teoretické zázemí a reálné vstupy pro naplnění dalších dílčích cílů.
2. Vymezit principy fuzzy logiky zahrnuté do výsledného řešení.
3. Vytvořit vlastní návrhový vzor a formulovat jej ve standardním formátu.
4. Navrhnout postup pro integraci návrhového vzoru do stávajících metod a nástrojů pro vývoj software.
5. Ověřit využitelnost návrhového vzoru prostřednictvím několika případových studií.

Vytvoření opakovaně použitelného návrhového vzoru pro oblast internetových IS umožní softwarovým analytikům a vývojářům relativně snadnou implementaci fuzzy přístupu do vlastních IS, a to jednotným postupem. Tím se zvýší nejen znuvupoužitelnost dílčích komponent, ale také to přinese zlepšení spolupráce mezi členy vývojových týmů a jejich zastupitelnost.
Očekávaným vedlejším přínosem disertační práce je, že poslouží jako ukázka možností, jakými lze výsledky primárního výzkumu zpřístupnit srozumitelnou formou do praxe.

1.3. Metodika

Při zpracovávání disertační práce byla použita kombinace logických a empirických metod výzkumu. Logické metody byly použity pro oblast analýzy vstupů a formulaci navrženého řešení. Empirické metody následně pomohly ověřit aplikovatelnost vytvořeného řešení v reálném prostředí.
První část práce má rešeršní charakter. Pro její zpracování byla použita analýza domácích a především zahraničních monografií, sborníků vědeckých konferencí,

odborných časopisů, on-line informačních zdrojů a softwarových prototypů. Analýza se zaměřila na oblasti bezprostředně související s tématem práce a jejím cílem byl rozbor jejich klíčových vlastností, vztahů a procesů z pohledu využití pro modelování neurčitosti v databázových systémech. Autor v této části práce těžil také ze své aktivní účasti na řadě konferencí a z praktických zkušeností, které získal jako spoluřešitel několika grantových projektů.

V druhé části práce probíhá vymezení rozsahu vlastního řešení dané problematiky. Formulace požadavků funkční i mimofunkční povahy je založena na detailní analýze a popisu stěžejních principů fuzzy logiky a objektově orientovaných databázových systémů. Technika formálního zápisu byla následně zkombinována s postupy běžně používanými v praxi, jakými jsou scénáře použití a sběr požadavků [Robertson 2006].

Stěžejní část práce obsahuje vlastní jádro řešení, které je formalizováno do podoby návrhového vzoru, tj. všeobecně uznávaného standardu pro popis řešení opakujících se problémů. Pro návrh prezentovaného původního autorova řešení byla vyžita syntéza dílčích poznatků o principech fuzzy logiky, objektově orientovaných databázových systémů a dosavadních přístupů k řešení této problematiky. Autor rovněž zužitkoval své praktické zkušenosti s vývojem databázových systémů na různých vývojových platformách (Java, .NET, Smalltalk). Riziko příliš velkého zobecnění, které při syntéze hrozí a může ohrozit primární cíl práce, se autor pokusil omezit ověřením v případových studiích.

Poslední část práce má z větší míry empirickou povahu. Pomocí experimentů na vybraných typech databázových systémů a programovacích jazyků je ověřována aplikovatelnost obecného vzoru ve specifickém prostředí. Riziku neprůkaznosti experimentů kvůli podobnosti jednotlivých prostředí, tj. neprokázání obecnosti použití vzoru, se autor snaží vyhnout volbou odlišně konstruovaných databázových systémů a programovacích jazyků. Na základě výsledků lze předpokládat, že vzor bude použitelný i pro další typy platforem a podařilo se tak naplnit hlavní cíl disertační práce.

1.4. Použitá terminologie

V rámci této práce je použita řada pojmů, jejichž význam je dán kontextem, ve kterém jsou užity, nicméně přesto je u některých z nich vhodné provést přesnější vymezení. Autor práce se v žádném případě nesnaží touto cestou definovat zvolené pojmy na obecné úrovni, ale naopak vysvětlit, v jakých souvislostech a rozsahu je užívá v této práci.

- **Databázový systém**

Pojem databázový systém je použit v běžném významu, tj. jako „kolekce vzájemně souvisejících dat uložených pohromadě v jednom či více počítačových souborech“ [IEEE]. Zdůrazněn je softwarový pohled, který je s ohledem na řešenou problematiku klíčový. Oproti běžným zvyklostem je ovšem pojem použit obecněji a to i pro systémy, které sice zajišťují perzistenci dat, avšak nesplňují veškeré obvyklé předpoklady plnohodnotného databázového systému (např. embedded databáze). Autor si je vědom určitého zjednodušení, ale domnívá se, že je účelné a přispěje k lepší čitelnosti práce.

- **OODBMS**

Objektově orientovanými databázovými systémy (OODBMS) jsou uvažovány systémy založené na třídně-instančním datovém modelu [Merunka et al. 1999]. Do této kategorie nejsou (z pohledu této práce) zahrnovány objektově-relační, XML ani jiné typy hybridních systémů.

- **Internetový informační systém**

S pojmem internetový informační systém se lze v literatuře setkat obvykle v kontextu webových aplikací, prezentací, on-line obchodů, popřípadě systémů založených výhradně na protokolu HTTP. Uvedené části však představují v mnoha případech pouze prezentační vrstvu celého informačního systému. V této práci je pojem internetový informační systém používán obecně pro všechny *vícevrstvé aplikace, které využívají na úrovni přenosu dat některý z internetových standardů komunikace*. Zároveň jsou zohledněna specifika, která tyto systémy mají. Jde především

o bezstavové prostředí, heterogenní software a hardware prostředí, metody autentizace/autorizace, způsob použití aplikace, nároky na dobu odezvy a další.

- **Fuzzy přístup v objektové databázi**

Fuzzy přístup v databázi označuje snahu o využití principů fuzzy logiky pro reprezentaci, ukládání a získávání komplexních nepřesně či vágně definovaných informací z objektově orientovaných databázových systémů.

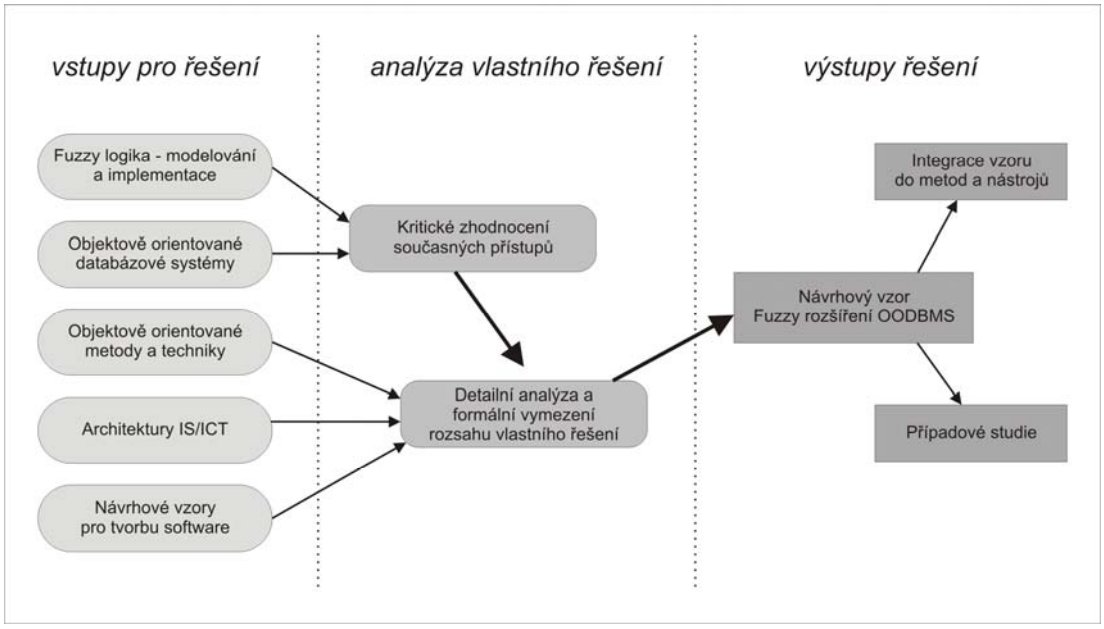
- **Návrhový vzor**

Popis řešení opakujících se problémů při vývoji software, který používá některou z běžně známých forem zápisu (struktury) jako je např. GoF [Gamma et al. 2003].

1.5. Struktura práce

Disertační práce je členěna v souladu s postupem řešení zvolené problematiky. Jednotlivé kapitoly lze rozdělit do následujících skupin:

1. *Analýza informačních zdrojů*.
2. *Vymezení rozsahu vlastního řešení a jeho formulace*.
3. *Návrhový vzor fuzzy rozšíření OODBMS*, který představuje jádro řešení.
4. *Integrace vzoru do metod a nástrojů*.
5. *Případové studie* pro ověření reálné využitelnosti navrženého řešení.



Obrázek 1 - Struktura řešení prezentovaného v disertační práci

1.5.1. Analýza informačních zdrojů

V tomto oddíle práce je shrnut současný stav poznání v oblastech, které jsou využity při analýze a následné konstrukci vlastního řešení. Přehled mapuje teoretická východiska, která kombinuje s technologiemi z oblasti databázových systémů a fuzzy logiky. Na závěr je uvedeno kritické zhodnocení současných přístupů.

Kapitola **„Historické souvislosti“** – obsahuje stručný přehled datových modelů využívaných v databázovém zpracování dat. Dále shrnuje základní myšlenky fuzzy logiky a jejího možného využití s vazbou na databázové systémy.

Kapitola **„Objektově orientované metody a techniky“** – nabízí přehled dnes používaných postupů pro vývoj software. Účelem kapitoly je vytvořit představu o prostředí, do kterého je potřeba navržené řešení začlenit. Nerespektování metodických principů by totiž vedlo k vytvoření v praxi nepoužitelného návrhu.

Kapitola **„Architektury IS/ICT“** – obdobně jako v předcházející kapitole jde o stručnou charakteristiku hlavních proudů dnešního vývoje, a to z pohledu architektur informačních systémů. Adaptace myšlenek, na kterých jsou postaveny opět vytvoří lepší předpoklady pro přijetí navrhovaného řešení.

Kapitola **„Návrhové vzory pro tvorbu software“** – obsahuje vymezení smyslu návrhových vzorů pro software, jejich obvyklou strukturu, a také nejčastější formy, které jsou akceptovány širokou veřejností. Doplněna je také základní klasifikace vzorů a jejich organizace.

Kapitola **„Současný stav řešené problematiky“** – z výsledků dosavadního výzkumu řešené problematiky jsou popsány možnosti využití jazyka UML při modelování neurčitosti na úrovni tříd a jejich vazeb. Dále je uveden přehled možností modelování neurčitosti v různých typech databázových systémů. V závěru kapitoly jsou uvedeny postupy pro aplikaci fuzzy přístupu, které lze využít na úrovni rozhraní pro perzistenci dat.

Kapitola **„Kritické zhodnocení současných přístupů“** – shrnuje dílčí úspěchy, které dosavadní řešení přinesla a kriticky hodnotí jejich nedostatky, mezi které patří zejména velmi omezená přenositelnost a obtížně realizovatelné požadavky na strukturu využívaných dat.

1.5.2. Vymezení rozsahu vlastního řešení a jeho formulace

Na základě analýzy informačních zdrojů a poznání současného stavu problematiky vymezuje autor rozsah svého původního řešení, které se snaží dosavadní řešení propojit a dopracovat do podoby, která naplní stanovený cíl.

Kapitola **„Modelové situace pro navrhované řešení“** – obsahuje schématické znázornění základních požadavků na řešení z pohledu konečného uživatele. Účelem je specifikovat rozhraní a možnosti vnějšího chování systémů vytvořených s pomocí navrhovaného řešení.

Kapitola **„Funkční pokrytí“** – vymezuje základní principy fuzzy logiky, které navrhované řešení musí respektovat, aby mohlo plnit svůj účel. Obsahuje přehled operací s fuzzy množinami a způsob použití fuzzy operátorů pro logické operace.

Kapitola **„Mimofunkční pokrytí“** – doplňuje předchozí kapitoly o požadavky vycházející z dnes používaných metodik, architektur a technologií.

1.5.3. Návrhový vzor fuzzy rozšíření OODBMS

V tomto oddíle disertační práce jsou shrnuty výsledky teoretického a praktického výzkumu, které autor během svého doktorského studia realizoval.

Kapitola **„Návrhový vzor Fuzzy rozšíření OODBMS“** – představuje jádro vlastního řešení. Vychází z oddílu analýzy informačních zdrojů a oddílu vymezujícího rozsah řešení. Podoba vzoru je samozřejmě zpětně ovlivněna také zkušenostmi z jeho praktické aplikace viz následující oddíl. Vzor je formulován pomocí GoF (formátu návrhových vzorů pojmenován po svých čtyřech autorech jako Gang-Of-Four), který patří mezi nejznámější a pro účely popisovaného je velmi vhodný. Vzor je rozdělen na dílčí části:

Účel – Vymezení účelu vzoru a problému, který je řešen.
Motivace – Ukázka řešení problému za použití navržených tříd a vztahů.
Použití – Popis situací, za kterých může být vzor aplikován.
Struktura – Grafické znázornění struktury vzoru.
Součásti – Popis tříd, které jsou součástí vzoru a jejich odpovědnosti.
Spolupráce – Schématické znázornění spolupráce jednotlivých součástí.
Důsledky – Výhody a nevýhody vzoru. Možnosti změn jednotlivých částí.
Implementace – Popis implementačních technik, úskalí, možností.
Příklad – Fragmenty kódu v jazyce Java jako ukázka implementace.

Známá použití – Odkazy na známá uplatnění některých částí vzoru.

Příbuzné vzory – Provázanost s dalšími návrhovými vzory a jejich role.

1.5.4. Integrace vzoru do metod a nástrojů

Oddíl shrnuje způsoby integrace vytvořeného vzoru do nástrojů pro podporu vývoje a rovněž do dnes preferovaných softwarových architektur SOA resp. MDA.

Kapitola „**Integrace s CASE nástroji**“ – autor zpřístupňuje vytvořený vzor ve formátu XMI a zavádí UML profil pro notaci Fuzzy UML.

Kapitola „**Integrace do architektur**“ – postupy jak propojit vývoj fuzzy orientovaných IS s architekturami SOA a MDA.

Kapitola „**Integrace z pohledu metodik**“ – poznámky k integraci do metodik.

Kapitola „**Omezení při integraci**“ – poznámky k omezením, na která je možné při integraci narazit.

1.5.5. Případové studie

Na oddíly s formulací návrhového vzoru a jeho integraci navazuje oddíl případových studií, které mají potvrdit využitelnost navrženého postupu a poskytnout přehled o způsobech kombinace obecného a přenositelného řešení s implementačními detaily konkrétních platforem (programovacích jazyků a OODBMS).

Kapitola „**Integrace vzoru do metod a nástrojů**“ – uvádí návaznosti řešení na stávající metody vývoje software.

Kapitola „**Versant Object Database“** – ukázka aplikace vzoru se specifickými vlastnostmi databázového systému Versant a programovacího jazyka Java 5.

Kapitola „**Gemstone/S“** – ukázka aplikace vzoru se specifickými vlastnostmi databázového systému Gemstone/S a programovacího jazyka Smalltalk.

Kapitola „**ObjectStore“** – ukázka aplikace vzoru se specifickými vlastnostmi databázového systému ObjectStore a programovacího jazyka Java.

Kapitola „**db4objects“ –** ukázka aplikace vzoru se specifickými vlastnostmi databázového systému db4objects a programovacího jazyka C# z rodiny .NET.

2. Literární rešerše

Zpracování literárních a dalších informačních zdrojů do podoby literární rešerše je orientováno nejprve na přehled historických souvislostí databázových systémů a jejich datových modelů. Následně jsou uvedeny klíčové vlastnosti fuzzy logiky a motivace jejího využívání v databázových systémech. S ohledem na cíl práce je dále zařazen přehled současného prostředí vývoje IS z pohledu používaných metodik a architektur. Samostatná kapitola je věnována problematice návrhových vzorů, jejich struktuře a nejběžnějším formám, které jsou v praxi akceptovány. Stěžejní část literární rešerše tvoří informace o současném stavu řešené problematiky fuzzy modelování v prostředí databázových systémů.

2.1. Historické souvislosti

Při snaze popsat, přestavit či zatřídit databázové systémy v dnešní době zjišťujeme, že jde o stále komplexnější seskupení softwarových programů, které již zdaleka nezajišťují pouze samotné ukládání a poskytování dat, jak bylo původně zamýšleno.
Za pravděpodobně první pokus o systém hromadného zpracovávání dat na počítačích lze považovat tzv. agendový přístup [Merunka et al. 1999]. Rozdělení dat o objektech reálného světa se odehrává na úrovni samostatných souborů, ve kterých je pro každý objekt uložen záznam skládající se z položek popisujících vlastnosti objektu. Velkou nevýhodou tohoto přístupu je problematické provádění obou základních operací, tj. vyhledávání i aktualizace záznamů, jelikož procházení dat bývá značně neefektivní. Za další nevýhody je nutné uvést obtížné sdílení agend mezi různými aplikacemi, redundanci, nízkou flexibilnost, omezené zabezpečení, integritní problémy atd. Jisté kvalitativní zlepšení agendového přístupu přinesl tzv. integrovaný přístup, který využíval v nových agendách ve větším rozsahu agend již existujících. Ostatních charakteristiky se však nezměnily. Přes uvedené problémy existuje dodnes řada specifických oblastí, ve kterých nachází tento přístup své široké uplatnění. Za všechny je nutné jmenovat např. svět WWW.
Databázové zpracovávání velkých objemů dat se začalo využívat v období 70. let. Ve většině případů nahradilo agendový přístup, jelikož přineslo výhodu v podobě oddělení vlastních dat od aplikací, které s nimi pracují. Základní myšlenkou databázového zpracovávání dat je existence nezávislé báze dat, která je

obhospodařována programem, který zajišťuje veškerou komunikaci s okolím. Tím je dosaženo následujících vlastností databázového zpracování:

I. struktury aplikačních programů jsou odděleny od datových souborů,
II. přístup k datům je možný jen prostřednictvím programů databázového systému a nikoliv přímo,
III. dotazy do báze dat nejsou pevné, tj. v aplikačních programech lze sestavovat dotazy bez znalosti konkrétního způsobu uložení dat,
IV. je zajištěn víceuživatelský přístup a ochrana před zneužitím.

Pod pojmem databázový systém se rozumí vlastní báze dat spolu s programovými prostředky pro její obsluhu.

2.1.1. Síťový a hierarchický datový model

S příchodem databázových systémů vznikla otázka, jak data v nezávislé bázi organizovat co nejefektivněji. Odstínění od aplikačních programů s sebou přineslo potřebu navrhnout dostatečně elegantní, a především univerzální datový model [Wikipedia 2006]. Mezi první takové modely patřil síťový datový model (např. IDS, IDMS) založený na množině záznamů a soustavě spojek mezi dvěma záznamy. Podobným typem byl také hierarchický datový model (např. IMS), kdy jde vlastně o modifikaci síťového datového modelu s tím rozdílem, že každý záznam je odkazován nejvýše jedním jiným záznamem (stromová struktura). Databáze založené na těchto typech datových modelů se někdy označují také jako navigační databáze. Podobně jako v případě agendového přístupu se i pro tyto modely dnes nachází široké uplatnění. Využívají je například technologie XML, DOM apod.

2.1.2. Relační datový model

Bezesporu nejrozšířenějším je dnes v databázích relační datový model [Merunka et al. 1999, Wikipedia 2006] vycházející z prací E. F. Codda publikovaných od konce 60. let v laboratořích IBM. Mezi první produkty s podporou nového modelu patřily INGRES, System-R (první implementace SQL), SQL/DS (později DB2) a další jako Oracle, Informix, MS SQL Server, Sybase atd. Relační datový model se odklonil od metod provázaných seznamů jednotlivých záznamů a navrhl jako základní jednotku relační tabulku. Data rozložená do více tabulek (relací) jsou provázána klíčovými hodnotami a pro manipulaci s nimi lze použít relační algebru. Relační algebra je

založena na pěti základních operacích: kartézský součin, sjednocení, rozdíl, selekce a projekce. Pro snazší manipulaci s relacemi byly zavedeny další operace jako např. spojení relací (kombinace kartézského součinu a selekce). Vymezení těchto množinových operací a operací mezi relacemi lze nalézt např. v [Pokorný 2004].

Tento koncept odstranil problémy sekvenčního přístupu a obtížné změny datové struktury, se kterými se potýkají síťový a hierarchický datový model. Implementace relačního datového modelu vycházejí povětšinou ze základních pravidel, která E. F. Codd definoval počátkem 70. let. Klíčovou výhodou se měl pak stát standardní dotazovací jazyk SQL. Naneštěstí se podařilo ideu všeobecně dodržovaného standardu SQL zrealizovat jen částečně, a tak se lze dnes setkat s jeho mnoha variantami. Přesto se zcela jistě podařilo vytvořit standard, který posunul vývoj celé oblasti databázových systémů.

2.1.3. Objektové databáze

Objektové databáze se poprvé objevují přibližně v polovině 80. let, kdy začala vznikat potřeba co nejelegantnějšího ukládání datových struktur z objektových programovacích jazyků. Počáteční výkonnostní deficit ve srovnání s relačními databázemi byl postupně smazán a dnes lze hovořit o plnohodnotných alternativách. Moderní objektové databázové systémy samozřejmě podporují řádově stovky až tisíce konkurenčních uživatelů, optimistické/pesimistické řízení transakcí i dnes velmi žádanou práci v clusteru [Barry 2004], [Rashid 2004].

U objektových databází je potřeba rozlišovat modely implementace vlastního uložení dat. Rozšířenější variantou jsou databáze založené na relačním datovém modelu, který je postupně doplňován o objektové vlastnosti (např. Oracle 10g [Lacko 2002], [Kyte 2005]). Plné využití objektového přístupu však umožňuje teprve databáze založená na čistě objektovém datovém modelu (např. Gemstone, Versant, ObjectStore). Objektový datový model je vhodnější především v situacích, kdy jsou do databáze ukládány značně strukturovaná data, která jsou vzájemně v mnoha vztazích. V takovém případě je možné využít principů, které jsou známé z klasických objektových, logických a funkcionálních programovacích jazykům [Merunka et al. 1999], [Merunka 2002], [Merunka 2004]. Mezi klíčové vlastnosti patří:

- Schopnost vytvářet množiny (kolekce) objektů, jejichž prvky mohou být objekty různých datových typů (tříd).
- Skládání objektů mezi sebou a využívání principu zapouzdření jejich vnitřní struktury.
- Odvozování nových tříd objektů od existujících pomocí mechanismu dědění.
- Definování atributů odvozených z ostatních datových složek objektu.
- Polymorfní chování objektů umožňující tvořit dotazy nad množinami objektů rozdílných tříd.

Rozvojem databázového zpracovávání dat postupně vznikla zcela nová oblast informatiky, která ovlivnila i ostatní oblasti jako jsou metody analýzy, návrhu, implementace, architektura aplikací a další. V současných informačních systémech je úloha databázového systému v mnoha směrech jen těžko nahraditelná.

2.1.4. Fuzzy logika

Fuzzy logika se poprvé objevila v roce 1965 v článku, který publikoval Lotfi A. Zadeh [Zadeh 1965]. Tehdy došlo k definici základního pojmu fuzzy logiky, a to tzv. fuzzy množiny. Termín fuzzy je do češtiny nejčastěji překládán jako neostrý, matný, mlhavý, neurčitý či vágní. Vyjadřuje snahu fuzzy teorie o pokrytí reality v její nepřesnosti a neurčitosti.

Klasická teorie množin připouští pouze dva stavy prvku vůči množině. Prvek do množiny buďto zcela patří, nebo vůbec nepatří. Fuzzy logika tento koncept rozšiřuje [Navara 2002]. Fuzzy množina je taková množina, která kromě úplného členství a žádného členství připouští rovněž členství částečné. Míra členství daného prvku ve fuzzy množině je vyjádřena stupněm příslušnosti. Hodnota stupně příslušnosti každého prvku je získána pomocí tzv. funkce příslušnosti.

Význam fuzzy logiky lze vnímat ve dvou rovinách. První rovinou je potřeba uchopit a pracovat s nepřesnými či mlhavými daty. Druhou rovinou je celkový přístup k popisu reálného světa. Při používání přesných popisů totiž dochází k idealizování skutečnosti, a tedy odklonu od reality [Novák 2000].

Klasický přístup vede ke snaze popsat skutečnost pouze dvěma stavy (patří/nepatří do množiny). Pokud není možné situaci jednoznačně rozhodnout, pak je řešený problém dekomponován a opět posouzen z hlediska jediných dvou možných stavů. V situaci, kdy již není možné či účelné problém dále dekomponovat, dochází

k odklonu od reality. Z uvedeného vychází i princip inkompatibility, který vyslovil profesor Zadeh:

„S rostoucí složitostí systému klesá naše schopnost formulovat přesné a významné vlastnosti o jeho chování, až je dosáhnuta hranice, za kterou jsou přesnost a relevantnost prakticky vzájemně se vylučující jevy.“

2.1.5. Využití fuzzy logiky v databázových systémech

Jedním z hlavních cílů výzkumu v oblasti databázových systémů posledních let je snaha sémanticky obohatit datový model. Klasické datové modely postrádají schopnost reprezentovat a manipulovat s nepřesnými informacemi reálného světa. Účelem zavedení fuzzy logiky do datového modelování je zlepšení tohoto stavu. Vznikla celá řada přístupů, a to především s ohledem na nejpoužívanější relační datový model.

Nicméně, rychlý rozvoj informatiky dal příležitost databázím proniknout do oblastí, kde je potřeba pracovat s komplexními objekty a jejich složitými vzájemnými vztahy. Jak již bylo uvedeno, zde se ukázalo, že čistě objektové paradigma je pro tento účel velice výhodné. Jelikož klasický relační datový model a jeho rozšíření o fuzzy přístup nenabídl uspokojivé řešení, většina výzkumu v této oblasti se soustředila na fuzzy objektově orientované datové modely za účelem komplexnost dat a jejich neurčitost řešit zároveň.

2.2. Objektově orientované metody a techniky

Do současné doby bylo vyvinuto několik vzájemně odlišných metod pro objektově orientovanou analýzu a návrh software. Zohlednění jejich principů, skladby, ale i pojmů a notací umožní lépe naplnit hlavní smysl této disertační práce, tj. vytvořit použitelné řešení pro zahrnutí fuzzy logiky do tvorby databázově orientovaných IS. Mezi dnes nejpoužívanější metody a techniky dle [Polák et al. 2003] patří:

OMT – Object Modeling Technique, jejímž autory jsou J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy a W. Lorensen. Metoda byla poprvé publikována nakladatelstvím Prentice Hall v knize „Object-oriented Modeling and Design" v roce 1991. Technika je zajímavá tím, že je v podstatě hybridní technikou, zahrnující jak objektové, tak i klasické nástroje. Nejčastěji se používá pro návrh databázově orientovaných aplikací s objektovým klientem a relačním serverem.

Coad-Yourdon Method, jejímiž autory jsou P. Coad a E. Yourdon. Metoda byla poprvé publikována v časopise American Programmer v roce 1989 a posléze v knihách obou autorů v nakladatelství Yourdon Press. Od počátku je k dispozici jako stejnojmenný CASE prostředek podporující jazyky Smalltalk a C++. Z uvedených technik je nejúspornější co do počtu používaných pojmů.

OOSD – Object-Oriented Software Development, jejímž autorem je G. Booch. Metoda byla poprvé publikována v roce 1991 v knize nakladatelství Benjamin/Cummings „Object-Oriented Design with Applications". Tato poměrně velmi komplexní metodologie je určena pro týmový vývoj rozsáhlých aplikací v jazyce C++ a Smalltalk, a ze všech vyjmenovaných metod pokrývá nejvíce objektových vlastností.

OOSE – Object-Oriented Software Engineering autora Ivara Jacobsona. Metoda je velmi kvalitně popsána ve stejnojmenné knize. (Alternativní název metody je „Objectory".) Vzhledem k tomu, že metoda má původ ve skandinávské škole, je velmi zajímavá pro aplikace z oblasti simulace a řízení. Jacobsonova metoda se jako první začala zabývat problematikou získávání a modelování informací v prvních fázích životního cyklu ještě před budováním konceptuálního diagramu. Úvodní technika této metody - „use case" - byla adoptována i do ostatních objektových metodologií.

Unifikovaný modelovací jazyk UML podporují od května 1996 autoři G. Booch, J. Rumbaugh a I. Jacobson pod záštitou firmy Rational Inc. Metoda začala být publikována průběžně na internetu a v sérii knih, vydávaných firmou Rational lnc, která dodává také vlastní CASE nástroj Rational Rose. Metoda, která je doporučovaným průmyslovým standardem pro notaci (způsob kreslení) diagramů, je ve vývoji a zatím představuje sjednocení myšlenek původních metod svých autorů na platformě OMT. Ve srovnání s původní OMT je patrný posun směrem k větší míře podpory objektových vlastností a určitý odklon od původních „hybridních" vlastností OMT - například zrušení datového modelování pomocí DFD. Je však třeba mít na pamětí, že UML je ve skutečnosti jen jazyk, tedy návrh standardu k zakreslování nejrůznějších objektových diagramů.

Metoda Martin-Odell autorů J. Martina (spolu s E. Yourdonem známý i z dřívějšího období) a J. Odella. Metoda je publikována v sérii knih nakladatelství Martin Books, a představuje podobně jako unifikovaná metoda pokus o sjednocení

dosavadních objektových zkušeností předchozích metod. Používá velké množství nejrůznějších diagramů a pojmů.

Object-Oriented Structured Design notation autorů A. I. Wassermanna, P. A. Pirchera a R. J. Mullera, publikovaná poprvé v časopise IEEE Computer č. 23(3) v roce 1990. Metoda je zajímavá především notací používaných diagramů, která ovlivnila následné metodologie.

OOER notation autorů K. Gormana a J. Choobineha, poprvé publikovaná na 23. mezinárodní konferenci o informatice (computer science) na Havaji v létě 1991. Metoda používá notaci ER diagramů v klasické Chenově syntaxi, rozšířené o objektové vlastnosti. Je výhodná pro použití v objektově orientovaných databázích.

Mezi další používané přístupy lze zařadit:

BORM – Business Object Relation Modeling autorů R. P. Knotta, V. Merunky a J. Poláka. Metoda určená nejen k samotnému vývoji software, ale také k analýze požadavků a modelování business procesů. BORM pokrývá všechny fáze životního cyklu vývoje software se zvláštním důrazem na fáze úvodní.

WebML z Milánské univerzity je určeno pro modelování specifických webově orientovaných aplikací. Poskytuje více různých pohledů na obsah aplikace, a to prostřednictvím tří základních modelů – datového, hypertextového a prezentačního.

Již delší dobu se pro projekty menšího rozsahu stále více uplatňují také tzv. agilní metodiky. Mezi hlavní představitele dle [Buchalcevová 2005], [Kadlec 2004] patří **Extrémní programování** (XP), **SCRUM**, **Lean Development**, **Feature-Driven Development**, **Agilní modelování** a další.

2.3. Architektury IS/ICT

V současném vysoce konkurenčním prostředí představuje IS/ICT jeden z rozhodujících faktorů, které ovlivňují úspěšnost organizací. Pokud má IS/ICT plně podporovat podnikové procesy a realizovat jejich potenciál, je třeba definovat architekturu IS/ICT, která bude umožňovat integraci dílčích prvků. V této kapitole je vymezen pojem architektura IS/ICT a jsou uvedeny techniky pro její zachycení. Podrobněji jsou charakterizovány dva současné nejvýznamnější trendy - Modelem řízená architektura a Architektura orientovaná na služby [Buchalcevová 2005].

Účelem je vymezit struktury používané dnes při tvorbě software tak, aby bylo možné do nich zahrnout prvek fuzzy přístupu.

2.3.1. Charakteristika architektury IS/ICT

Architektura IS/ICT představuje významný faktor úspěšného IS/ICT. Význam architektury IS/ICT bývá odvozován od role architektury ve stavebnictví a urbanistice. Chápání architektur se v mnohých publikacích liší. Metodika MMDIS rozlišuje dvě úrovně architektur IS/ICT - globální architekturu, která je hrubým návrhem celého IS/ICT, a dílčí architektury, které představují detailnější návrhy IS/ICT z hlediska různých dimenzí (funkční architektura, procesní architektura, datová architektura, technologická architektura, softwarová architektura, hardwarová architektura, architektura služeb) [Voříšek 1997], [Voříšek 2003].

Globální architektura (Enterprise architecture) zachycuje strukturu a procesy na úrovni celé organizace. Model globální architektury je reprezentací těchto struktur a procesů. Dobrý architektonický model zachycuje organizaci z různých pohledů a akceptuje i změny v budoucnu. Přínos vytváření globální architektury spočívá ve vyšší kvalitě výsledného řešení, větší rychlosti řešení a menších nákladech. Tím, že vytvoříme globální architekturu IS/ICT, umožníme vývojářům vytvářet systémy, které pracují navzájem konzistentně a efektivně. Systém také mnohem lépe odpovídá potřebám, protože vývojáři pracují se znalostí obecných business vizí a celkové infrastruktury. Výsledná řešení jsou rychlejší, protože důležitá rozhodnutí již byla realizována, byla vytvořena celková infrastruktura pro komunikaci, sdílení zdrojů apod. Globální architektura explicitně ukazuje, jakou funkcionalitu a jaká data je možné znovupoužít v rámci organizace.

Výhody vytváření globální architektury jsou zřejmé, ale přesto se v praxi setkáváme s řadou problémů. Ty pramení ze skutečnosti, že budování IS/ICT se neřídí v rámci celé organizace, ale řídí se jen jednotlivé projekty. Globální architektura často neexistuje, a pokud ano, je často zastaralá nebo s ní nejsou vývojáři seznámeni.

Architektura je většinou prezentována ve formě modelů. Pro modelování architektury systému lze použít různé techniky jako například standardní modelovací jazyk Unified Modeling Language (UML), Zachman Framework a další. V současnosti se při návrhu architektury uplatňují také vzory. Architektonické vzory vyjadřují základní strukturální schéma uspořádání softwarových systémů, poskytují

množinu předdefinovaných subsystémů, specifikují odpovědnosti, zahrnují pravidla pro uspořádání vztahů mezi nimi. Vzory poskytují mocný slovník pro komunikaci mezi architekty a návrháři. Znalost a pochopení vzoru přenáší znalost a zkušenost, která je ve vzoru obsažena.

Pro analýzu a návrh architektury softwarového systému je možné použít také metody definované Institutem softwarového inženýrství viz [SEI]:

- Architecture Tradeoff Analysis Method SM (ATAM),
- Quality Attribute Workshop (QAW),
- Attribute-Driven Design (ADD).

Tyto metody jsou založeny na myšlence, že pro architekturu softwaru jsou určující kvalitativní požadavky (jako výkon, bezpečnost, modifikovatelnost, spolehlivost a použitelnost). Návrh architektury spočívá v aplikaci tzv. „architektonických taktik". Architektonická taktika reprezentuje návrhové rozhodnutí, které umožňuje dosáhnout určité hodnoty atributu kvality.

V současnosti jsou v oblasti architektur dominující dva trendy. Modelem řízená architektura (Model Driven Architecture) a Architektura orientovaná na služby (Service Oriented Architecture), které jsou podrobněji charakterizovány v následujících kapitolách.

2.3.2. Modelem řízená architektura

Myšlenky, na kterých je založena Modelem řízená architektura (Model Driven Architecture, MDA), nejsou nikterak nové. Možnost oprostit se od technologických problémů a zaměřit se na věcné problémy s sebou přinesl už vznik jazyka Cobol a rozvoj strukturovaných metod v 60. letech 20. století. Myšlenka oddělení analytického (konceptuálního) pohledu od pohledu návrhu a implementace je jedním ze stěžejních principů (Princip tří architektur) metodiky MMDIS.

MDA vychází ze skutečnosti, že množství změn v systému klesá s postupem na vyšší úrovně abstrakce. Dopady neustálých změn technologií je možné omezit jen na část modelu - na jeho nižší vrstvy. Při MDA vývoji se po základním Business modelu (CIM) vytvoří nejprve Platformově nezávislý model (Platform Independent Model – PIM), který reprezentuje věcnou funkcionalitu a chování systému. Pomocí MDA nástrojů se PIM následně mapuje na zvolenou platformu (například Corba, Java/EJB,

XML/SOAP) a generuje se Platformově specifický model (Platform Specific Model – PSM). Nakonec se generuje implementační kód pro příslušnou technologii. Tyto generátory využívají především návrhové vzory a šablony, které je možné snadno rozšiřovat v závislosti na modelované oblasti. MDA nástroje umožňují samozřejmě také zpětné inženýrství (reverse engineering), a tak je možné vytvořit modely stávajících systémů např. pro účely integrace aplikací.

2.3.2.1. Business model (CIM)

Výpočetně nezávislý business model popisuje věcné aspekty dané problémové oblasti bez ohledu na to, zda budou automatizovány. Soustředí se na prostředí a požadavky na systém. Vnitřní struktura systému je na této úrovni skryta. Je vytvářen business analytiky.

2.3.2.2. Platformově nezávislý model (PIM)

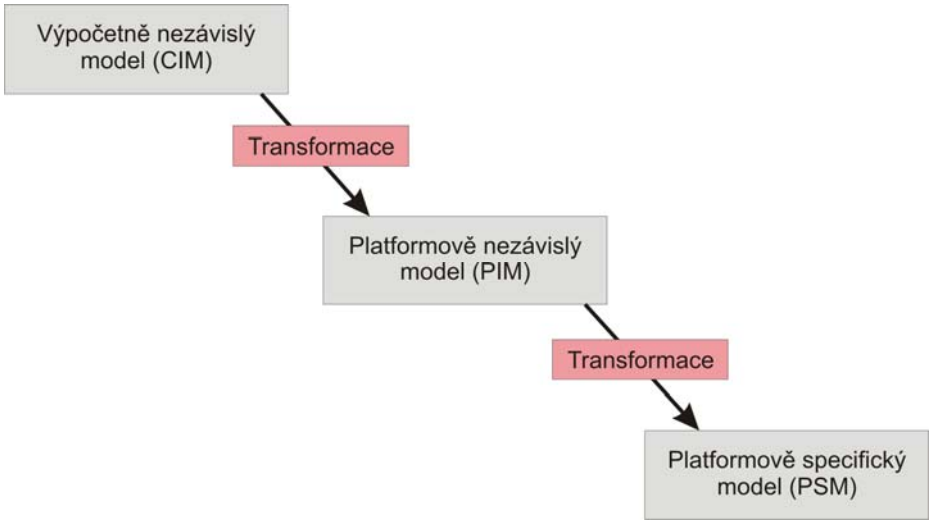
Klíčovou úlohu v MDA plní (Platform Independent Model, PIM). PIM představuje konceptuální model dané problémové oblasti, který je nezávislý na platformě. Platformou se rozumí určitý jazyk, technologie middlewaru a jim odpovídající inženýrské přístupy. Model se soustředí na konkrétní operace systému, ale nezabývá se zatím způsobem jakým je realizovat. Obvykle je pro tuto úroveň modelu používána systémovými architekty notace UML.

2.3.2.3. Platformově specifický model (PSM)

Při zobrazení PIM na určitou platformu vznikají nejen artefakty příslušné platformy (například Interface Definition Language, IDL), ale také Platformově specifický model (PSM), který mnohem lépe zachytí sémantiku řešení pro specifickou platformu. Model reprezentuje jak věcnou, tak technologickou sémantiku aplikace. Je to stále UML model, ale vyjádřený v dialektu UML, který se označuje jako UML profil (viz dále). UML profil odráží technologické prvky cílové platformy [Frankel 2003].

2.3.2.4. Transformace modelů

Největší hodnota MDA tkví v možnosti automatického nebo poloautomatického převodu modelů mezi sebou (viz Obrázek 2). Model CMI je pomocí zobrazení převeden na model PIM, a ten následně na model PSM. Na konci je model PSM transformován na výsledný kód. Výhodou je, že tímto způsobem může vzniknout více platformově závislých modelů (PSM) z jednoho centrálně spravovaného modelu na platformě nezávislém (PIM).



Obrázek 2 – Modely v MDA a jejich převod pomocí transformačních pravidel

MDA zahrnuje následující standardy OMG týkající se modelování:

1. XML Metadata Interchange (XMI),
2. Unified Modeling Language (UML),
3. Common Warehouse Metamodel (CWM),
4. Meta Object Facility (MOF).

Základem MDA je **modelovací jazyk UML** - Unified Modeling Language. Silná stránka UML v této oblasti spočívá v jeho metamodelu a v možnosti převodu modelů do XML na základě standardu **XMI**. UML není jeden jazyk, ale základ pro rodinu jazyků. UML obsahuje vestavěný mechanismus pro rozšíření (stereotypy a připojené hodnoty), který umožňuje definovat a používat dodatečné modelovací konstrukty a vytvářet tak dialekty UML nazývané UML profily. UML profil představuje definovanou množinu stereotypů a připojených hodnot, které rozšiřují elementy UML metamodelu, a je základem pro tvorbu platformově specifických modelů.

K dispozici jsou například následující UML profily.

1. UML profile for Corba - definuje zobrazení z PIM do PSM pro technologii CORBA,
2. UML profile for EDOC - jazyk pro modelování spolupráce komponent a podnikových procesů nezávislý na middlewaru, definuje Enterprise Collaboratíon Architecture a zobrazení z PIM do webových služeb,
3. UML profile for EAI - profil pro systémy komunikující pomocí zpráv.

Meta Object Facility (MOF) je jazykem pro vyjádření konstruktů modelů, tedy jde o metajazyk. Používá stejné modelovací konstrukty pro Diagram tříd jako UML, a tak je možné používat pro jeho tvorbu běžné modelovací nástroje podporující UML. MOF akceptuje realitu, kdy různé modely vyjadřují různé pohledy na systém. Tyto různé modelovací jazyky je však třeba popsat jednotným způsobem. A takovým univerzálním způsobem, jak popsat různé modelovací konstrukty, je právě MOF. Architektura MOF je tvořena 4 metaúrovněmi, jak ukazuje Tabulka 1- Úrovně MOF.

Úroveň	**Popis**
Úroveň M3	MOF, množina konstruktů pro definici metamodelů
Úroveň M2	Metamodely definované pomocí konstruktů MOF, např. UML, CWM, CCM
Úroveň M1	Modely tvořené instancemi konstruktů M2 metamodelu, např. třída Zákazník apod.
Úroveň M0	Objekty a data, tj. instance konstruktů M1 modelu, např. zákazník Jan Novák

Tabulka 1- Úrovně MOF

Common Warehouse Metamodel (CWM) je standardem definovaným pomocí MOF. Standardizuje databázové modelovací jazyky tak, že si nástroje různých výrobců mohou vyměňovat datové modely, pravidla transformace dat, analytické specifikace.

Strategická iniciativa OMG Model Driven Architecture představuje renesanci myšlenek, které byly proklamovány při tvorbě softwaru již dlouhou dobu. Převratný vývoj v technologiích a silný důraz na efektivnost prostředků vložených do informačních technologií však v současné době vytvářejí příznivé klima pro realizaci těchto myšlenek. Navíc jsou tyto myšlenky prosazovány organizací, která má dlouholeté zkušenosti s definováním standardů (UML, CORBA). Cílem MDA je zachovat investice vložené do informačních technologií tím, že se vývojáři zaměří na

vytváření platformově nezávislých modelů, které reprezentují věcnou oblast a nejsou dotčeny změnami v technologiích. Z těchto modelů potom s využitím MDA nástrojů, které se dnes již začínají objevovat, budou automatizovaně generovány platformově specifické modely i vlastní implementace.
MDA vytváří také vhodné podmínky pro vznik opakovaně využitelných návrhů řešení. Ve spojení s automatickým procesem jejich aplikace se zejména v oblastech rychle se měnících požadavků (typicky internetové IS) mohou využít pro optimalizaci refactoringu existujících systémů.

2.3.3. Architektura orientovaná na služby

Snaha o zefektivnění informačních technologií a důraz na přínosy v podnikových procesech vede stále více k prosazování konceptu služeb. Služba informatiky je definována jako „ucelený definovaný výstup procesů informatiky poskytovaný jejím interním i externím zákazníkům. Služby jsou výsledkem kombinace konkrétních instancí procesů a zdrojů, které probíhají v oblasti informatiky" [Bruckner 2001]. Služby vystupují jako spojovací článek mezi podnikovými procesy a IS/ICT. Společnost Meta Group je přesvědčena, že jedním z nejdůležitějších trendů v informatice v následující dekádě bude právě orientace na služby. Otázka orientace na služby při vývoji IS/ICT není jen otázkou technologie, ale podstatně ovlivňuje i metodiky vývoje a nasazení IS/ICT. Ačkoli se technologie i ekonomické prostředí za poslední desetiletí dramaticky změnily, metodiky většinou s těmito změnami nepočítají. Většina metodik byla navržena pro prostředí, kde se aplikace vyvíjejí od tzv. „na zelené louce“, ne pro prostředí založené na službách, kdy je třeba řešení poskládat z různých již existujících zdrojů.
Podle typu služby rozlišujeme aplikační služby (jejich obsahem je poskytovaná funkčnost aplikací IS/ICT), infrastrukturní služby (realizují infrastrukturu pro služby aplikační) a ostatní služby (zajišťují koordinaci a řízení činností apod.). Současný zájem o aplikační služby je spojen zejména s technologií webových služeb, ale služby lze realizovat i pomocí jiných technologií. Webové služby (Web Services) představují evoluční vývoj, který odstraňuje nedostatky komponentových architektur (například CORBA, DCOM). Technologie webových služeb je tvořena množinou průmyslových standardů založených na XML, které specifikují komunikační protokol (SOAP), jazyk pro definici (WSDL) a registr pro publikování a vyžádání

služby (UDDI). Na standardech založené a volně spřažené webové služby mohou být implementovány firmami bez znalosti konkrétních konzumentů, a naopak konzumenti nepotřebují pro využívání služeb znát detaily jejich implementace. Volné spřažení služeb vytváří také podmínky pro budování robustnějších informačních systémů, kde výpadek jedné služby nemusí znamenat selhání celého procesu. Po dobu výpadku např. služby pro komunikaci s hlavním dodavatelem může být automaticky použita alternativní služba komunikující s jiným dodavatelem.

Organizace začínají nabízet určité funkčnosti ve formě webových služeb, často však pouze obalí existující komponenty. Tento přístup ke službám zdola není dostačující. Je třeba se zaměřit na přístup shora, který vyžaduje změny v architektuře IS/ICT. Je třeba zavést Architekturu orientovanou na služby (Service Oriented Architecture, SOA). Podstatou této architektury jsou volně spřažené (loosely coupled) na standardech založené služby [Bloomberg 2003]. Tyto architektury kromě nových architektonických přístupů vyžadují i novou roli v organizaci - roli architekta orientovaného na služby.

Architektura orientovaná na služby je postavena na těchto klíčových principech:

1. podnikové procesy jsou určující pro služby, které jsou zase určující pro technologii,
2. ve skutečnosti služby fungují jako abstraktní vrstva mezi podnikovými procesy a technologií,
3. služby musí umožňovat agilitu podnikových procesů, tj.
4. rychle a pružně reagovat na změny požadavků,
5. úspěšná Architektura orientovaná na služby se stále vyvíjí.

Atraktivita služeb spočívá ve zvýšení produktivity IS/ICT řešení, snížení nákladů vývoje a nasazení a zkrácení času uvedení na trh. Dalším cílem je vyšší přínos z IS/ICT řešení. Architektura orientovaná na služby je důležitá pro podniky, protože představuje rámec, který sjednocuje byznys model s technologiemi a realizuje funkcionalitu zajišťující efektivní podnikání. Bez Architektury orientované na služby se IS/ICT stávají nepropojenou kolekcí balíků, funkcí a obrazovek, které konzumují stále rostoucí zdroje na údržbu a rozvoj. Architektura orientovaná na služby zavádí přímý vztah mezi business operacemi a aplikačními službami, a tak zjednodušuje údržbu systému a jeho refaktorizaci na bázi služeb.

2.4. Návrhové vzory pro tvorbu software

Často používanou definicí pro návrhové vzory je výrok, že jde o *řešení problému v určitém kontextu*. Takovéto definování je dostatečně obecné, ale příliš nevyjadřuje podstatu návrhových vzorů pro tvůrce software. Dle Fowlera [Fowler 2006] je vhodnější se na návrhové vzory dívat primárně jako na způsob rozdělení rad a doporučení k určitému tématu. Jednotlivé části lze následně používat bez nutnosti si pamatovat vše do nejmenším detailů. S tímto chápáním vzorů koresponduje i názor, že návrhové vzory pro software představují dědictví softwarového inženýrství a podílejí se tak nepřímo na vymezení celého odvětví IT.

Každý návrhový vzor by měl pojmenovávat nějaké řešení. Toto řešení by mělo být dostatečně konkrétní alespoň na úrovni oblasti, které se týká. Pokud je jméno vzoru vše co je nutné si vyměnit při diskusi nad řešením určitého problému, pak je pravděpodobné, že název vzoru se stane pevnou součástí slovníku dané profese. Takovým pojmem je kupříkladu „dekorátor", který může představovat dostatečnou odpověď na dotaz jak řešit určitý problém. Detaily řešení je pak možné najít v katalogu vzorů.

Jednou z nejdůležitějších vlastností návrhových vzorů je jejich znovupoužitelnost v různých situacích, resp. při řešení různých problémů. Pokud se dvě samostatná a nezávislá řešení zdají být na první pohled odlišná, ale přesto mají jisté vnitřní podobnosti, pak jde o to, co Christopher Alexander nazývá „jádrem řešení“ ve svém výroku: „Každý vzor popisuje problém, který se v našem prostředí neustále vyskytuje. Potom popisuje jádro řešení daného problému tak, že nám umožňuje toto řešení používat třeba milionkrát, aniž bychom to dělali dvakrát stejným způsobem.“ [Alexander 1977]

Návrhové vzory nejsou samozřejmě jediné techniky sdílení postupů orientovaných na řešení obecných problémů. Podobnou úlohu plní také klasické *návody*, resp. technické dokumentace ve stylu kuchařek. Na rozdíl od návrhových vzorů jsou kuchařky obvykle více orientovány na specifické problémy konkrétních programovacích jazyků a platforem a nesnaží se přistupovat k řešení v obecnější rovině. Dá se říci, že obě techniky se vzájemně velmi vhodně doplňují, protože obě zdůrazňují postup „od problému k řešení“.

Potřeba návrhových vzorů pro software vychází ze zkušenosti, že mnoho projektů končí neúspěchem kvůli nedostatku obecných a běžně používaných řešení dílčích

problémů. I přes nesporné výhody návrhových vzorů je potřeba zdůraznit, že jde o komunikační prostředek, který nemusí být za každé situace vždy nejvhodnější nebo jediný vhodný. Je proto výhodné využít je k pojmenování, třídění a extrahování detailů jednotlivých řešení, ale kombinovat je rovněž s obecnějším popisem kontextu, potřeba a zkušeností, případně s detailnějšími kuchařkami a poskytnout tak ucelenější návrh řešení.

Myšlenka návrhových vzorů pochází z architektury a z oblasti výstavby budov a měst. Princip ovšem zůstává stejný i pro oblast vývoje software, které se bude věnovat následující text.

2.4.1. Podstatné části vzoru

Většina návrhových vzorů je formalizována v určitých pevných strukturách. Variant těchto struktur je celá řada a každý autor si vybírá tu, kterou považuje za nejpřirozenější. Bez ohledu na konkrétní zažité struktury je však možné uvést ty podstatné části vzoru, které jsou společné.

Jak již bylo uvedeno jsou vzory nejčastěji popisovány jako řešení určitého problému. Je vhodné však zdůraznit, že jde primárně o řešení nikoliv problém jako takový. Forma psaní vzoru se proto vždy podřizuje jedinému účelu, tj. vzor je úspěšný, pokud je zformulován tak, aby ostatní mohli řešení zopakovat, pokud je to vhodné. K sestavení správného řešení je samozřejmě nutné nejprve důkladně prozkoumat samotný problém a zároveň se vyvarovat zúžení na konkrétní nástroj nebo programovací jazyk (zde je pak účelnější použít např. výše zmíněné kuchařky).

Protože úspěšné vzory se postupně stávají součástí slovníku profese jsou jejich jména volena tak, aby byla výstižná, obecná a snadno zapamatovatelná. Díky tomu slouží jména vzorů také jako komunikační prostředek napříč technologiemi. Java programátor, který používá konstrukt „listeners“, je totiž snadno schopen se domluvit s .NET programátorem, který používá konstrukt „delegates“ za předpokladu, že si vyjasní, že jde jen o odlišné způsoby implementace návrhového vzoru Pozorovatel (Observer).

Nedílnou součást vzoru představují obvykle také podmínky, za kterých je vhodné vzor použít, ale i podmínky, při kterých je to zcela nevhodné. Řešený problém je totiž vždy zasazen do konkrétního kontextu, který může použití vzoru podporovat či

naopak znemožnit. Není proto výjimkou, že pro jeden problém existuje v závislosti na okolních podmínkách více vzájemně alternativních návrhových vzorů.
Podstatnou a často nejproblematičtější částí návrhových vzorů pro software bývají příklady kódu. Při jejich používání vzniká nebezpečí, že budou považovány nikoliv za doplněk, ale za hlavní a striktní podobu celého vzoru. Většina autorů se tomuto snaží vyhnout uváděním více příkladů typicky na různých platformách.

2.4.2. Obecně používané formy vzorů

Každý z autorů používá víceméně vlastní formu psaní vzorů, nicméně některé z nich se v praxi osvědčily natolik, že jsou (alespoň částečně) přebírány.

2.4.2.1. Alexandrova forma

Kniha Christophera Alexandra „A Pattern Language“ [Alexander 1977] ovlivnila dle mnoha lidí podstatným způsobem pojetí vzorů pro lidskou činnost. Alexander použil pro svoji knihu určitou formu, která je po převzetí do oblasti software známa jako Alexandrova forma. Dnes je možné se setkat s celou řadou variant této původní formy.
Svojí strukturou se Alexandrova forma blíží vyprávění, které je rozděleno jen do několika málo sekcí. Princip zvýrazňování klíčových vět v sekci popisující problém a v sekci řešení umožňuje velmi rychlou orientaci mezi jednotlivými vzory a zároveň vystihuje jejich základní podstatu. Přestože se nejedná o příliš strukturovanou formu zápisu, bývá někdy označována za ne příliš jednoznačnou z pohledu rozdělení řešené problematiky do příslušných sekcí.

2.4.2.2. GOF forma

Forma GOF (Gang of Four) byla zavedena v klíčové publikaci zavádějící návrhové vzory do oblasti vývoje software „Design Patterns: Elements of Reusable Object-Oriented Software“ [Gamma et al. 2003]. Je to značně strukturovaná forma rozdělující návrhový vzor do řady sekcí: Účel, Motivace, Použití, Struktura, Součásti, Spolupráce, Důsledky, Implementace, Příklad, Známá použití a Příbuzné vzory.

2.4.2.3. Portland forma

Název formy je odvozen od původu lidí (Portland, Oregon), kteří se zúčastnili první konference na téma návrhových vzorů. Většina z nich používala podobný způsob zápisu, který je čistě textově orientovaný a zároveň velmi stručný.

Nejprve je v několika málo odstavcích popsán problém a poté následuje klíčové slovo *Therefore* (a proto), za kterým následuje řešení. Odkazy na související vzory jsou uváděny přímo v těle řešení.

Tato forma je používána často v případě, že popisovaný problém ani jeho řešení nejsou příliš komplikované či jako první podoba vzoru před jeho dalším rozpracováním. Příklady doplněné o ukázky v jazyce Smalltalk je možné najít např. na [Cunningham 1994].

2.4.2.4. Coplien forma

Svoje jméno dostala tato forma podle Jima Copliena, ale často se o ní mluví také jako o kanonické formě zápisu návrhových vzorů pro software. Ukázkovým příkladem může být sada vzorů pro telekomunikační systém v AT&T [AT&T 1995]. Opět jde o zástupce stručného popisu problému, jeho kontextu a řešení. Obsahuje základní čtyři sekce, které jsou dle potřeby doplněny o další stručné pasáže.

2.4.2.5. POSA forma

POSA forma je podobně jako GOF forma do značné míry strukturovaná podoba popisu návrhových vzorů, liší se však názvy jednotlivých sekcí: Souhrn, Příklad, Kontext, Problém, Struktura, Dynamika, Implementace, Vysvětlující příklad, Varianty, Známá použití, Důsledky a Odkazy. Důležitým prvkem této formy je úvodní kapitola každého vzoru, která ho shrnuje a popisuje volným textem celou problematiku. Název formy je odvozen od knihy „Pattern-Oriented Software Development“ [Buschmann 1996].

2.4.2.6. Fowlerova EAA forma

Tato forma (P of EAA) nepředstavuje dosud žádný standard, ale pro značné rozšíření návrhových vzorů jejího autora Martina Fowlera do přehledu dnes používaných způsobů zápisu vzorů jistě patří. Je strukturovaná do několika částí: Jak vzor pracuje,

Kdy vzor použít a jeden či dva příklady. Popis jednotlivých částí je volným textem, diagramy a schématy. Příkladem je vzor Notifikace v [Fowler 2002].

2.4.3. Klasifikace vzorů

Návrhové vzory se liší svojí granularitou i úrovní abstrakce. V současné době existují stovky či možná tisíce návrhových vzorů pro software a pro orientaci v nich je potřeba je nějakým způsobem organizovat. Klasifikace vzorů vede k jejich snazšímu pochopení a zároveň podporuje hledání nových vzorů.
V původní publikaci GOF [Gamma et al. 2003] bylo navrženo základní rozdělení 23 základních vzorů na tvořivé, strukturální, vzory chování a dále pak jemnější rozdělení dle oblasti (třída/objekt). Seskupení těchto vzorů dohromady tvoří katalog, který lze přirovnat ke knihovně a nezřídka se i název knihovny stává součástí slovníku softwarových odborníků.

1. **Tvořivé vzory** se zabývají vytvářením nových objektů. *Třídní tvořivé vzory* přenechávají určitou část zodpovědnosti za tvorbu objektů na podtřídy, zatímco *Objektové tvořivé vzory* ji přenechávají jinému objektu.

2. **Strukturální vzory** se zaměřují na skladbu tříd či objektů. *Strukturální třídní vzory* používají ke skladbě tříd dědičnost, zatímco *Strukturální objektové vzory* popisují způsoby, jak se objekty vzájemně skládají.

3. **Vzory chování** charakterizují způsoby, podle nichž třídy či objekty vzájemně jednají a rozdělují si povinnosti. *Třídní vzory chování* používají dědičnost k popisu algoritmů a řízení procesů, zatímco *Objektové vzory chování* popisují, jak skupina objektů spolupracuje při provádění úkolu, který žádný objekt nemůže vykonat samostatně.

2.4.3.1. Katalogy vzorů

Užitečnost a srozumitelnost prvního katalogu návrhových vzorů pro tvorbu software dala podnět ke vzniku mnoha dalších vzorů v nejrůznějších specifických oblastech vývoje SW. Tématu této práce jsou z nepřeberného množství vzorů nejbližší ty, které se týkají tzv. Enterprise systémů, tedy tzv. Enterprise vzory (český překlad

podnikové či obchodní vzory není příliš používán, a proto je ponechán původní anglický název), spolu se vzory pro databázové systémy a vzájemné integrace informačních systémů. Zde jsou ve stručnosti uvedeny některé z klíčových katalogů takovýchto vzorů (odkazy na příslušné publikace je možné nalézt např. na [Fowler 2005]):

1. **Oblast architektury Enterprise aplikací**

Patterns of Enterprise Application Architecture (Martin Fowler) – pohled na architekturu Enterprise aplikací (EAA) pohledem nezávislým na použitých technologiích.

Core J2EE Patterns – První kniha zaměřená speciálně na EAA, a to z pohledu platformy JAVA.

Microsoft Enterprise Solution Patterns – Pohled na EAA z pohledu platformy .NET.

2. **Oblast integrace Enterprise aplikací**

Enterprise Integration Patterns – Návrhové vzory pro systémy zpráv (messaging), které jsou považovány v dnešní době za velmi perspektivní.

Microsoft Integration Patterns – Strategie pro integraci aplikací na technologiích firmy Microsoft.

Microsoft Data Patterns – Vzory pro replikaci a synchronizaci dat, které představují dvě stěžejní oblasti integrace Enterprise aplikací.

3. **Oblast doménové logiky**

Domain Driven Design – Vzory detailně se zabývající doménovou logikou, a které najdou uplatnění zejména v případě velmi komplexních systémů.

Analysis Patterns – Vzory, které ukazují příklady doménových modelů.

Data Model Patterns – Vzory, které se dívají na doménové modely z pohledu datového modelování.

2.5. Současný stav řešené problematiky

Problematikou fuzzy modelování v prostředí databázových systémů se zabývala v minulosti řada autorů. Jako vhodná notace pro jeho reprezentaci se díky kvalitnímu metamodelu ukázal jazyk UML [Arlow 2003]. V tomto oddíle je uveden princip rozšíření tak, jak jej navrhl Ma v přehledu pokročilých technik fuzzy modelování pro objektově orientované databázové systémy [Ma 2005].

Uvedené techniky modelování fuzzy informací poslouží pro návrh vlastního řešení a způsobu jeho integrace do současných metod a nástrojů pro tvorbu informačních systémů.

2.5.1. Modelování fuzzy informací pomocí UML

UML (Unified Modeling Language) je skupinou notací (nejen) objektově orientovaného modelování a je standardem ODMG (Object Data Management Group, dále OMG). Je využíváno v mnoha oblastech softwarového a znalostního inženýrství. Stále více je UML používáno také pro datové modelování na úrovni databáze.

UML poskytuje soustavu modelů pro zachycení mnoha aspektů softwarového systému. Z pohledu modelování databází je relevantní zejména model tříd. Jeho základními stavebními kameny jsou třídy a jejich vztahy.

V běžném (ostrém) modelu tříd popisuje třída množinu objektů se společnou strukturou, chováním a vztahy. Třída má svůj název, přehled atributů a popis hlavních operací (atributy a operace mohou být uváděny volitelně). Objekt α je potom instancí třídy A, jestliže $\alpha \in A$. Hodnota atributu je $f(\alpha)$, kde α je objekt a f je funkce (zobrazení) třídy A do domény příslušného atributu.

Vztahy reprezentují různý typy souvislostí mezi třídami nebo jejich instancemi. Základní typy vztahů jsou agregace (vztah celek-část), generalizace (zobecnění), asociace (obecné spojení) a závislost.

V následujícím přehledu jsou uvedena rozšíření UML diagramu tříd pro účely reprezentace fuzzy informací.

2.5.1.1. Fuzzy třída

Klasickou třídu objektů lze teoreticky vymezit dvěma odlišnými způsoby:

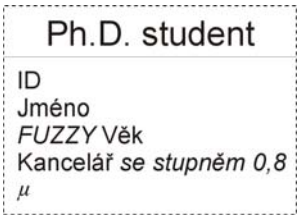
1. Pomocí extenze, tj. třída je definována seznamem svých instancí,
2. Pomocí intenze, tj. třída je definována množinou atributů a jejich přípustných hodnot.

Třídu je možné považovat za fuzzy z několika důvodů. Při vymezení pomocí extenze nemusí pro všechny objekty α platit podmínka $\alpha \in A$, resp. platí podmínka pouze částečné příslušnosti prvku ke třídě $\mu_A(\alpha)$ v intervalu [0,1]. Pak se jedná o tzv. fuzzy objekty a i třída, jejíž instancemi jsou by měla být označena jako fuzzy. Naopak při použití intenze mohou mít fuzzy povahu některé domény atributů. Fuzzy povahu má i třída, která vznikne specializací z jiné fuzzy třídy, nebo naopak generalizací ze tříd, z nichž alespoň jedna je fuzzy. Po vzoru Zvieli a Chena (1986) jsou definovány tři úrovně neostrosti:

1. Neostrost příslušnosti třídy do datového modelu, resp. atributu do třídy.
2. Neostrost příslušnosti instancí ke své třídě. I přes ostré vymezení třídy může dojít k situaci, kdy objekt je instancí této třídy jen s určitým stupněm příslušnosti.
3. Třetí úroveň neostrosti se týká hodnot atributů. Atribut ve třídě má přiřazenu doménu hodnot a pokud je tato doména fuzzy podmnožinou (přípustnost hodnot jen s určitým stupněm), objevuje se neostrost hodnot atributu.

Pro modelování prvního stupně neostrosti by měl být ve Fuzzy UML atribut nebo jméno třídy doplněno dovětkem „se stupněm X“, kde X hodnota z intervalu [0,1] a označuje stupeň s jakým atribut náleží do třídy nebo s jakým třída patří do datového modelu. Například „Zaměstnanec se stupněm 0,6“ a „Číslo kanceláře se stupněm 0,8“ jsou třída a atribut s neostrostí prvního typu. Obecně, atribut nebo třída nebudou vůbec v modelu zahrnuty, pokud jejich stupeň příslušnosti je roven 0. U stupně příslušnosti 1 bude dovětek naopak vynechán. Pro modelování neostrosti třetího typu je doplněn příznak „FUZZY“ před název atributu. Pro neostrost druhého typu je potřeba určit stupeň příslušnosti, s jakým je objekt instancí dané třídy. Pro tento účel je do struktury třídy zaveden nový atribut „μ“, který vyjadřuje příslušnost objektu ke třídě s doménou přípustných hodnot z intervalu [0,1]. Pro rozlišení třídy s druhým stupněm neostrosti je použit čárkovaný obdélník. Obrázek 3 **– Fuzzy třída**

znázorňuje třídu „Ph.D. student“. Atribut „Věk“ zde může nabývat neostrých hodnot, a tak je jeho doména doplněna o příznak „FUZZY“. Ph.D. studenti mohou mít svoji kancelář. Není však zcela jasné, zda studenti atribut „Kancelář“ skutečně mají. Lze však říci, že studenti ji budou mít se stupněm důvěry 0,8. Dále není zcela jasné, zda objekt je či není instancí třídy, která je fuzzy. Z toho důvodu je zaveden do třídy pomocný atribut „μ“, který příslušnost objektu ke své třídě vyjadřuje pomocí stupně příslušnosti z intervalu [0,1].



Obrázek 3 – Fuzzy třída

2.5.1.2. Fuzzy generalizace a specializace

Vztah nadtřída-podtřída je jedním ze základních stavebních prvků objektového modelu. Nová třída (označovaná jako podtřída) je vytvářena z výchozí třídy (označované jako nadtřída) metodou dědění atributů a metod, jejich překrýváním a definicí nových atributů a metod. Protože podtřída je specializací své nadtřídy, tak platí, že libovolný objekt patřící do podtřídy musí zároveň patřit i do nadtřídy. Tato charakteristika může být použita pro rozhodnutí, zda dvě třídy mezi sebou mají či nemají vztah typu nadtřída-podtřída.

Je-li nadtřídou fuzzy třída, tak i odvozená třída musí být fuzzy. Vazba mezi těmito třídami je přirozeně také fuzzy, tj. se stupněm příslušnosti v intervalu [0,1]. Pro vztah podtřída-nadtřída platí:

1. Libovolný (fuzzy) objekt patří do podtřídy se stupněm příslušnosti menším nebo rovným se stupněm své příslušnosti k nadtřídě.
2. Stupeň příslušnosti k podtřídě je větší nebo roven stanovené prahové hodnotě (α řez).

Stupeň příslušnosti třídy do vztahu podtřída-nadtřída je roven minimu stupňů příslušnosti objektů k podtřídě. Nechť A a B jsou (fuzzy) třídy a β je stanovená prahová hodnota. Pak B je podtřídou A, jestliže platí:

$$(\forall e)(\beta \le \mu_B(e) \le \mu_A(e)).$$

Stupeň příslušnosti třídy *B* do vztahu, že *B* je podtřídou *A* je roven $\min_{\mu B(e)\geq\beta}(\mu_B(e))$.

Zde *e* je instancí *A* resp. *B* a $\mu_A(e)$, $\mu_B(e)$ jsou stupně příslušnosti *e* do *A* resp. do *B*.

Výše uvedený vztah fuzzy generalizace předpokládá, že třídy *A* a *B* mají mezi sebou neostrost pouze druhého typu. V případě prvního typu neostrosti je situace odlišná. Předpokládejme třídu *A* se stupněm příslušnosti *memA* a třídu *B* se stupněm *memB*. Potom *B* je podtřída *A*, jestliže platí:

$$(\forall e)(\beta \leq \mu_B(e) \leq \mu_A(e)) \wedge ((\beta \leq memB \leq memA)).$$

To znamená, že *B* je podtřídou *A* pouze pokud platí, že:

- stupně příslušnosti všech objektů do *A* a *B* jsou vyšší nebo rovny stanovené prahové hodnotě,
- stupeň příslušnosti libovolného objektu do *A* je vyšší nebo roven stupni příslušnosti objektu do *B*,
- stupně příslušnosti *A* a *B* musí být vyšší nebo rovny zadané prahové hodnotě,
- stupeň *A* musí být vyšší nebo roven stupni *B*.

Má-li fuzzy nadtřída *A* podtřídy *B1*, *B2*,…,*Bn*, které mají po řadě vlastní stupně příslušnosti *memA*, *memB1*, *memB2*,…*memBn* a stupně příslušnost svých instancí po řadě μ_A, μ_{B1}, $\mu_{B2,\ldots}$, μ_{Bn}, pak platí následující:

$$(\forall e)(\max(\mu_{B1}(e), \mu_{B2}(e), ..., \mu_{Bn}(e))\mu_A(e)) \wedge$$
$$(\max(memB1, memB2, ...memBn) \leq memA) \quad .$$

Tato metoda je použitelná pouze v případě vymezení fuzzy třídy pomocí extenze, tj. výčtem objektů, které k ní přísluší. Při použití intenze je možné použít tzv. stupeň inkluze tříd. Pojem stupeň inkluze byl zaveden v [Ma et al. 2004] pro stanovení redundantních dat ve fuzzy relačních databázích.

Nechť *A* a *B* jsou (fuzzy) třídy a stupeň příslušnosti *B* do vztahu *B* je podtřídou *A* je označen jako $\mu(A,B)$. Pro stanovenou prahovou hodnotu β je *B* podtřídou *A* pokud:

$$\mu(A,B) \geq \beta.$$

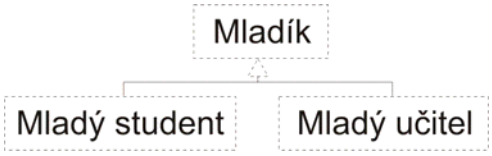
Jsou-li *A* a *B* třídy s neostrostí prvního typu, pak *B* je podtřídou *A*, jestliže

$$(\mu(A,B) \geq \beta) \wedge (\beta \leq memB \leq memA).$$

To znamená, že *B* je podtřídou *A* pouze v případě, že:

- stupeň inkluze *A* vzhledem k *B* je vyšší nebo roven stanovené prahové hodnotě,
- stupeň příslušnosti *A* a *B* je vyšší nebo roven stanovené prahové hodnotě,
- stupeň příslušnosti *A* je vyšší nebo roven stupni příslušnosti *B*.

Postup výpočtu je možné najít v [Ma et al. 2004]. Pro potřeby znázornění vztahu fuzzy generalizace je používána šipka z přerušované čáry, viz Obrázek 4 – **Fuzzy generalizace**. Třídy *Mladý student* a *Mladý učitel* jsou třídy s neostrostí druhého typu. Třídy mohou mít instance (objekty), které mají určitý stupeň příslušnosti. Obě třídy mohou být generalizovány do společné fuzzy nadtřídy *Mladík*.



Obrázek 4 – Fuzzy generalizace

2.5.1.3. Fuzzy agregace

Agregace mezi třídami vyjadřuje vztah *celek-část*. Jednotlivé dílčí části přitom mohou existovat nezávisle. Díky tomu může být každá instance *celku* promítnuta do množiny instancí svých *částí*. Nechť *A* je agregací *částí B1, B2,...Bn*. Pro $e \in A$ je projekce do *Bi* označena jako $e\downarrow_{Bi}$. Potom $\left(e\downarrow_{B1}\right) \in B1, \left(e\downarrow_{B2}\right) \in B2, \left(e\downarrow_{Bn}\right) \in Bn$.

Třída složena z fuzzy *částí* musí být opět fuzzy třídou. Nová třída je agregací svých složek se stupněm příslušnosti v intervalu [0,1]. Pro vztah fuzzy agregace platí:

1. Libovolný (fuzzy) objekt patří do agregace, pokud jeho stupeň příslušnosti k *celku* je menší nebo roven stupni příslušnosti jeho projekce do jednotlivých *částí*.
2. Stupeň příslušnosti k *celku* je větší nebo roven stanovené prahové hodnotě.

Celkem je potom agregace *částí* se stupněm příslušnosti, jehož hodnota je minimem stupňů příslušnosti s jakými tyto objekty přísluší k odpovídajícím *částem*. Nechť *A* je

fuzzy agregací množiny fuzzy tříd *B1, B2,...,Bn* (stupni příslušnosti instancí označeny po řadě jako μ_A, $\mu_{B1,}$ $\mu_{B2,...,}$ μ_{Bn}) a je dána prahová hodnota β, pak:

$$(\forall e)\left(e \in A \wedge B \leq \mu_A(e) \leq \min\left(\mu_{B1}\left(e\downarrow_{B1}\right), \mu_{B2}\left(e\downarrow_{B2}\right),..., \mu_{Bn}\left(e\downarrow_{Bn}\right)\right)\right).$$

Fuzzy třída *A* je *celkem* složeným z fuzzy tříd *B1,...,Bn*, pokud pro libovolnou (fuzzy) instanci platí, že stupeň příslušnosti k třídě *A* je menší nebo roven stupni příslušnosti projekce instance do *B1, B2,...a Bn* odpovídajícímu *Bi* $(1 \leq i \leq n)$. Zároveň pro každou instanci musí být stupeň příslušnosti k *A* vyšší nebo roven stanovené prahové hodnotě.

Nechť třídy *A, B1,...Bn* mají po řadě stupně příslušnosti *memA*, *memB1*, *memB2, memBn*. Pak pro neostrost prvního typu platí, že *A* je agregací *B1, B2, ... Bn,* pokud:

$$(\forall e)\begin{pmatrix} e \in A \wedge B \leq \mu_A(e) \leq \min\left(\mu_{B1}\left(e\downarrow_{B1}\right), \mu_{B2}\left(e\downarrow_{B2}\right),..., \mu_{Bn}\left(e\downarrow_{Bn}\right)\right) \wedge \\ memA \leq \min(memB1, memB2,..., memBn) \end{pmatrix}.$$

Symbol β je stanovená prahová hodnota. Tento postup lze použít pokud jsou fuzzy třídy stanoveny na základě extenze. V případě intenze je postup odlišný:

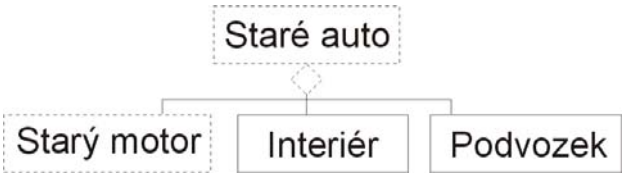
$$\min\left(\mu\left(B1, A\downarrow_{B1}\right), \mu\left(B2, A\downarrow_{B2}\right),...\mu\left(Bn, A\downarrow_{Bn}\right)\right) \geq \beta .$$

Zde $\mu\left(Bi, A\downarrow_{Bi}\right)(1 \leq i \leq n)$ je stupeň s jakým *Bi* sémanticky zahrnuje projekci *A* do *Bi*. Stupeň příslušnosti s jakým je *A* agregací *částí B1, B2,...a Bn* je

$$\min\left(\mu\left(B1, A\downarrow_{B1}\right), \mu\left(B2, A\downarrow_{B2}\right),...\mu\left(Bn, A\downarrow_{Bn}\right)\right).$$

Výraz lze dále rozšířit pro situaci, ve které mají třídy *A, B1, B2,...,Bn* neostrost prvního typu, tj. jde o fuzzy třídy se stupni příslušností. Potom *A* je agregací *B1, B2,...,Bn*, pokud

$$\min\left(\mu\left(B1, A\downarrow_{B1}\right), \mu\left(B2, A\downarrow_{B2}\right),...\mu\left(Bn, A\downarrow_{Bn}\right)\right) \geq \beta \wedge$$
$$stupeň\_A \leq \min(memB1, memB2,..., memBn) \quad .$$

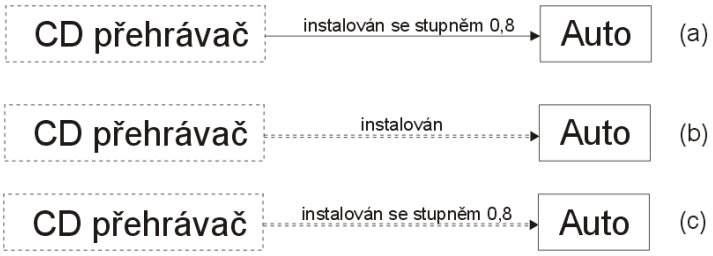


Obrázek 5 – Fuzzy agregace

Pro označení fuzzy agregace je použita značka kosočtverce z přerušované čáry, jak ukazuje Obrázek 5 – **Fuzzy agregace**. Auto je zde složeno z motoru, interiéru a podvozku. Jelikož motor je starý, vyskytuje se v modelu fuzzy třída S*tarý motor*. Třída *Staré auto* je složena z interiéru, podvozku a fuzzy třídy *Starý motor* a stává se tak třídou s druhým stupněm neostrosti.

2.5.1.4. Fuzzy asociace

Ve vztahu asociace mohou být identifikovány dvě dílčí úrovně neostrosti. První dílčí úroveň vyjadřuje stav, kdy samotná asociace mezi dvěma třídami je neostře definovaná, tj. existuje s určitým stupněm předpokladu. Zároveň však nemusí být zcela jisté, zda dvě instance patřící do asociovaných tříd mají mezi sebou vztah, který mezi sebou mají dané třídy. To je druhá dílčí úroveň neostrosti a je způsobena tím, že instance mohou příslušet ke svým třídám jen s určitým stupněm příslušnosti. Obě dílčí úrovně neostrosti mohou nastat v rámci jednoho vztahu mezi třídami najednou. To znamená, že dvě třídy mají mezi sebou fuzzy asociaci na úrovni tříd a zároveň jejich instance mohou mít mezi sebou fuzzy asociaci na úrovni objektů.



Obrázek 6 – Typy fuzzy asociací

K reprezentaci první dílčí úrovně neostrosti mezi asociovanými třídami lze použít slovní příznak „se stupněm X“, kde X je z intervalu [0,1] za názvem asociace. Pro označení druhé dílčí úrovně neostrosti lze použít dvojitou přerušovanou čáru s šipkou. Obrázek 6 – **Typy fuzzy asociací** znázorňuje zápis obou typů neostrosti a jejich kombinace. V části (a) není jisté, zda CD přehrávače jsou do aut instalovány.

V části (b) je jisté, že přehrávače do aut jsou instalovány, ale není jisté, zda tento vztah existuje i na úrovni konkrétního přehrávače a auta. Část (c) kombinuje oba předchozí stavy.

Jak bylo uvedeno výše, mezi třídami mohou nastat tři obecné úrovně neostrosti. Třídy s druhou obecnou úrovní neostrosti mají v případě vzájemné asociace mezi sebou asociaci opět s druhým stupněm neostrosti (pokud je jisté, že tato asociace existuje, tj. v asociaci není první úroveň neostrosti). Potom, instance *e* má stupeň příslušnosti ke třídě *A* označen jako $\mu_A(e)$ a instance *f* má stupeň příslušnosti k *B* označen jako $\mu_B(f)$. Za předpokladu, že vztah asociace mezi *A* a *B* označen jako *ass(A, B)* nemá neostrost prvního typu, pak vztah mezi *e* a *f* označen jako *ass(e, f)* má neostrost druhého typu, tj. stupeň příslušnosti získaný jako

$$\mu(ass(e,f)) = \min(\mu_A(e), \mu_B(f)).$$

První úroveň neostrosti v asociaci lze odhalit již na úrovni designu, a to i v případě, že příslušné třídy jsou definovány ostře. Předpokládejme dvě ostře definované třídy *A* a *B* s asociací *ass(A, B)* prvního typu neostrosti označenou jako *ass(A, B)* se stupněm příslušnosti X. Tedy $\mu_A(e) = 1{,}0$ a $\mu_B(f) = 1{,}0$. Potom:

$$\mu(ass(e,f)) = X\ .$$

U fuzzy tříd s prvním stupněm neostrosti se stejný stupeň neurčitosti předpokládá i pro jejich asociaci. Mějme třídy označené jako „*A* se stupněm X“ a „*B* se stupněm Y“. Potom jejich asociace označená jako *ass(A, B)* má první stupeň neostrosti, tj. *ass(A, B)* se stupněm příslušnosti Z:

$$Z = \min(X, Y).$$

Pro instanci *e* třídy *A* a instanci *f* třídy *B*, kde $\mu_A(e) = 1{,}0$ a $\mu_B(f) = 1{,}0$ platí:

$$\mu(ass(e,f)) = Z = \min(X, Y).$$

Může však nastat také situace, kdy mají třídy první i druhý stupeň neostrosti a jejich vztah je explicitně uvedená asociace s prvním stupněm nepřesnosti. Potom platí:

$$\mu(ass(e,f)) = \min(\mu_A(e), \mu_B(f), X, Y, Z).$$

2.5.1.5. Fuzzy závislost

Závislost je vztahem mezi zdrojovou a cílovou třídou. Týká se pouze tříd a z tohoto důvodu ho druhý a třetí typ neostrosti tříd neovlivňuje.

Vztah fuzzy závislosti je vztahem s určitým stupněm předpokladu. Obdobně jako vztah fuzzy asociace, může být fuzzy závislost určena explicitně na úrovni designu, nebo implicitně zdrojovou třídou. Za předpokladu, že zdrojová třída má neurčitost prvního stupně, bude mít i cílová třída stejný typ neurčitosti. Míra předpokladu, že cílová třída je určena zdrojovou třídou je stejná jako stupeň příslušnosti zdrojové třídy do modelu.

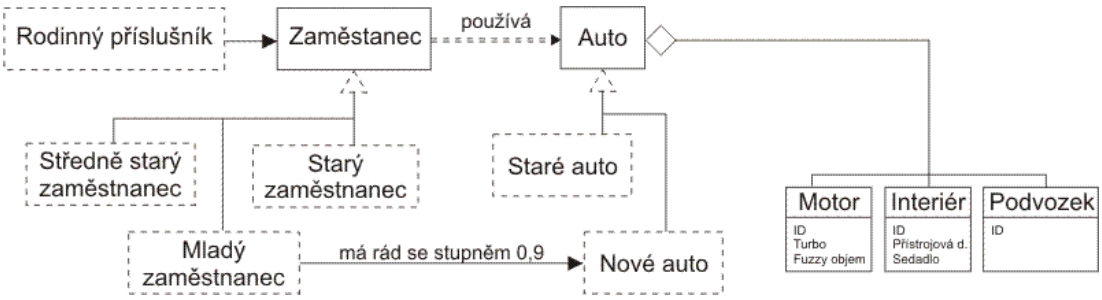
Pro zdrojovou třídu „Zaměstnanec se stupněm 0,5“ předpokládejme závislou třídu „Rodinný příslušník se stupněm 0,5“. Vztah závislosti mezi těmito dvěma třídami by je rovněž fuzzy, a to se stupněm příslušnosti 0,5. Na rozdíl od fuzzy asociace může být ve vztahu fuzzy závislosti nalezen pouze jeden typ neostrosti, a to první.

Obrázek 7 – Fuzzy závislost

Pro označení neostrosti vztahu typu závislost je použita přerušovaná čára s šipkou viz Obrázek 7 **– Fuzzy závislost**.

2.5.1.6. Komplexní příklad

Na obrázku (Obrázek 8 **– Fuzzy UML datový model**) je uveden jednoduchý fuzzy UML model tříd využívající některé z výše představených notací.



Obrázek 8 – Fuzzy UML datový model

Třída *Auto* je nadtřída pro třídy *Nové auto* a *Staré auto*, které jsou jejími fuzzy podtřídami. Tyto podtřídy mohou mít fuzzy instance. Obdobně, třída *Zaměstnanec* má tři fuzzy podtřídy: *Mladý zaměstnanec, Středně starý zaměstnanec* a *Starý zaměstnanec*. Třídy *Zaměstnanec* a *Auto* mají fuzzy asociaci *používá*, která má neostrost druhé úrovně. Dále mají vztah fuzzy asociace s neostrostí prvního typu třídy *Mladý zaměstnanec* a *Nové auto*. Kromě toho je třída *Auto* složena pomocí agregace ze tříd: *Motor, Podvozek* a *Interiér*. Třída *Motor* má tři atributy. Atribut *Id* a *Turbo* mají ostře definované hodnoty, zatímco *objem* je fuzzy atribut, který může nabývat fuzzy hodnot.

2.5.2. Přístupy k tvorbě fuzzy databází

Potřeba reprezentace určitého stupně neurčitosti, nepřesnosti, neznalosti či irelevantnosti dat byla jednou z priorit od samého počátku existence databázových systémů. Během jejich vývoje byla vyvinuta řada přístupů jak pro relační datový model, tak i pro pozdější objektově orientovaný datový model.

2.5.2.1. Neurčitost bez použití fuzzy logiky

V této sekci jsou shrnuty některé základní přístupy pro zachycení nakládání s nepřesnými informacemi v databázových systémech.

2.5.2.1.1. Vícehodnotová logika

Za vůbec první postup, jak reprezentovat nepřesné informace, lze pravděpodobně považovat zavedení NULL hodnoty, které navrhl E. F. Codd v roce 1979. Hodnota NULL vyjadřuje stav, kdy skutečnou hodnotou může být jakákoliv hodnota z domény atributu. Porovnávání této hodnoty s jinou má za výsledek jednu ze tří hodnot: PRAVDA (T), NEPRAVDA (F), MOŽNÁ (m)/NEZNÁMÁ (unknown jak je použito např. v Oracle). Pravdivostní tabulky pro klasické operace AND, OR, NEG jsou všeobecně známé. Později však bylo zavedeno ještě další rozlišení NULL hodnoty, a to na „příznak A“ a „příznak I“. „A příznak“ pro chybějící nebo neznámou relevantní hodnotu a „I příznak“ pro chybějící hodnotu, která není relevantní (např. jméno dítěte u osoby, která nemá děti). Pravdivostní tabulky pro tuto čtyřhodnotovou logiku vypadají takto:

NOT	
T	F
A	A
I	I
F	T

AND	T	A	I	F
T	T	A	I	F
A	A	A	I	F
I	I	I	I	F
F	F	F	F	F

OR	T	A	I	F
T	T	T	T	T
A	T	A	A	A
I	T	A	I	F
F	T	A	F	F

Obrázek 9 - Čtyřhodnotová logika v SQL s rozlišením NULL hodnot

2.5.2.1.2. Výchozí hodnoty

V roce 1982 přišel Date [Date 1986] s ošetřením některých nedostatků v zacházení s NULL hodnotami. Místo nich navrhl použití tzv. výchozích hodnot, které by byly definovány pro každou doménu v relační databázi. Při vložení nové *n*-tice se pro hodnoty atributů, které uživatel nezadal použijí výchozí hodnoty. Koncept výchozích hodnot brzy doplnil koncept NULL hodnot v relačních databázích.

2.5.2.1.3. Intervalové hodnoty

Rozšíření relačního modelu za účelem reprezentace intervalu hodnot v jednom atributu navrhl v roce 1980 Grant [Grant 1980]. Problém opakování *n*-tic byl vyřešen povolením takového stavu, jelikož *n*-tice se stejnými intervaly nemusí být totožné. Relační operátory byly předefinovány ve dvou verzích: PRAVDA (T) a MOŽNÁ (M). Například operátor „<" byl definován takto:

```
[a, b] <T [n, m]      jestliže b < n,
[a, b] <M [n, m]      jestliže a < m.
```

Pro dotazy v tomto modelu je každá *n*-tice zařazena do jedné ze tří kategorií: zcela jistě patří do výsledku (surely-set), možná patří do výsledku (possibly-set) a zcela jistě nepatří do výsledku (eliminated-set).

2.5.2.1.4. Statistické a pravděpodobností databáze

Stěžejní práce na téma statistických a pravděpodobnostních databází (statistical and probabilistic databases) byla publikována Wongem v roce 1982 [Wong 1982]. Hlavní myšlenkou byl předpoklad, že neúplné informace mohou být statisticky porovnávány. Navržený postup považoval dotazy za statistické experimenty, ve

kterých jsou informace neúplné a pozornost je soustředěna na výpočet takové množiny *n*-tic, která minimalizuje oba typy statistických chyb.

Barbara, Garcia-Molina a Porter [Barbara 1992] představili pravděpodobností databáze, ve kterých jsou pravděpodobnosti asociovány s hodnotami konkrétních atributů. V tomto modelu má každý pravděpodobnostní atribut diskrétní pravděpodobnostní rozdělení. V rámci jedné *n*-tice musí být hodnoty pravděpodobnosti normovány, tj. součet pravděpodobností všech možných hodnot musí být roven 1. Stanovení pravděpodobností všech hodnot však může být obtížné. Z tohoto důvodu byl zaveden pojem chybějící pravděpodobnosti, která představuje doplněk součtu známých pravděpodobností do hodnoty 1. V databázi jsou tedy uloženy známé pravděpodobnosti hodnot a dále pak zmíněný doplněk pro hodnoty, jejichž pravděpodobnost neznáme.

Pravděpodobnostní databáze podporují dotazy s podmínkami pro získání jen takových hodnot, jejich pravděpodobnost je vyšší než stanovená mez. Pro výpočet pravděpodobnosti jsou použita uživatelem zadaná kritéria, případně přímá analýza zadaného vzorku dat.

2.5.2.2. Základní modely fuzzy databází

Základní a nejjednodušší model fuzzy relační databáze přidává stupeň z intervalu [0, 1] každé instanci (nebo *n*-tici). To umožňuje udržet homogenitu dat v databázi. Nicméně využitelnost tohoto stupně je dána sémantikou, která se mu přiřadí při zpracovávání dotazů.

Tento stupeň může vyjadřovat stupeň příslušnost každé *n*-tice do relace [Giardina 1979, Mouaddib 1994]. Může ovšem vyjadřovat také něco jiného, jako sílu závislosti mezi dvěma atributy [Baldwin 1983], stupeň splnění podmínky, nebo stupeň významu *n*-tice mezi ostatními [Bosc 1997].

Hlavním nedostatkem těchto fuzzy modelů je, že neumožňují reprezentaci nepřesných informací o určitém atributu konkrétní entity (jako např. hodnoty „vysoký“ nebo „malý“ u výšky člověka). Kromě toho je fuzzy hodnota přiřazena globálně každé instanci (*n*-tici), a tím znemožňuje identifikovat jakým způsobem se na výsledku podílejí jednotlivé atributy.

Buckles-Petry Model – první model, který využívá vztahu podobnosti v relačním modelu. Hodnoty podobnosti nabývají obvykle hodnot v rozmezí intervalu [0,1], kde 0 představuje zcela odlišné a 1 zcela shodné *n*-tice.

Modely možností

Na teorii možnosti (Possibility theory) byla založena řada modelů jako např. *Prade-Testemale* model, *Umano-Fukami* model, *Zemankova-Kaendel* model a asi nejpropracovanější *GEFRED* model. Popis uvedených modelů přesahuje rámec této práce a je možné jej nalézt např. v [Galindo et al. 2006]. Hlavním přínosem těchto modelů je bezesporu zavedení a postupné rozpracování tzv. fuzzy relační algebry. V ní jsou používány tzv. fuzzy operátory, které jsou odvozeny od klasických relačních operátorů.

2.5.2.3. Fuzzy objektově orientované databázové modely

Ve fuzzy modelech nad OODBMS lze využít zkušeností z modelů nad relačními databázemi a zároveň těžit z výhod objektového přístupu.

2.5.2.3.1. Zobecněný objektově orientovaný databázový model

Autoři Tré, de Caluwe, a Van der Cruyssen navrhli v roce 2000 zobecněný objektově orientovaný databázový model, který představuje formální aplikační rámec pro definici fuzzy modelů v OODBMS [Tré 2000]. Model byl získán jako zobecnění standardního objektově orientovaného databázového modelu odpovídajícího ODMG standardu. Hlavní komponentou modelu je typový systém doplněný o systém omezení. ODMG standard dosud trpí některými nedostatky jako je omezená schopnost se vypořádat s omezeními, která slouží především pro zajištění databázové integrity. Navržený aplikační rámec se tento nedostatek snaží odstranit a zároveň umožnit tvorbu fuzzy databázových modelů. Mezi často diskutované nevýhody uvedeného modelu však patří obtížnost při nastavování fuzzy omezení a podmínek.

2.5.2.3.2. Fuzzy objektově orientovaný databázový systém

Vzorovou implementaci fuzzy objektově orientovaného datového modelu (FOOD) představili v roce 1999 autoři Bordogna, Leporati, Lucarella a Pasi [Bordogna 1999]. Model umožňuje reprezentaci vágních hodnot atributů a vtahů různé síly. FOOD

model je definován jako rozšíření grafově založeného objektového modelu za účelem podchycení ostrých i vágních informací pomocí teorie fuzzy množin. Konceptuální schéma v tomto modelu je charakterizováno následující pěticí {**C**, **T**, **A**, **P**, **N**}:

1. **C** je konečná množina názvů tříd (ostré i fuzzy třídy). Fuzzy třídy obsahují objekty, které mají ke své třídě pouze částečné členství.
2. **T** je konečná množina názvů typů (ostré i fuzzy typy). Položky vágních typů označují vágní a nepřesné hodnoty.
3. **A** je množina názvů atributů. Atributy jsou jednoduché, pokud je jejich doménou typ, nebo jsou komplexní, pokud je jejich doménou třída. Atributy mohou být jednohodnotové nebo vícehodnotové.
4. **P** je vlastnost vztahu (relace). P dává do vztahu třídu a její názvy atributů.
5. **H** je vtah dědičnosti.

Nedokonalosti v reprezentaci dat pomocí modelu FOOD si kladou za cíl podchytit následující:

1. Vágní hodnoty atributů jsou definovány v modelu v případech, kdy přesné hodnoty daných atributů nejsou známy.
2. Neurčitá vlastnost a neurčitá vazba umožní přiřadit stupeň neurčitosti, který může být interpretován jako míra tolerance k porušení omezení pro aktuální hodnotu atributu.
3. Zesílená vlastnost a zesílená vazba dovoluje použít mezi dvěma objekty stupeň nejistoty.
4. Fuzzy třídy jsou vhodné pro reprezentaci částečného členství objektu do své třídy.
5. Fuzzy třídní hierarchie reprezentují vágnost v hierarchii definované pro účely klasifikace. Tento případ nastaven, pokud je jak nadtřída, tak podtřída fuzzy.

Implementace aplikačního rámce byla koncipována jako rozšíření objektově orientovaného databázového systému O2, který poskytuje API dostupné z vlastního programovacího jazyka O2C (objektově orientované rozšíření jazyka C doplněné o přímý přístup k datovým typům). Přestože je implementace zatím neúplná,

představuje významný krok ve vývoji fuzzy objektově orientovaných databází a slouží jako inspirace pro další výzkum v této oblasti.

2.5.3. Začlenění fuzzy přístupu do existujících perzistentních rozhraní a systémů

V této části jsou uvedeny hlavní oblasti zájmu během integrace fuzzy přístupu do existujících perzistentních rozhraní a systémů.

2.5.3.1. Začlenění fuzzy přístupu na úrovni rozhraní

V současné době jsou při rozšiřování objektových databázových technologií o fuzzy přístup používány dvě odlišné metody.

První z nich se soustředí na poskytnutí komplexnější palety konceptuálních objektů pro získání lepšího modelu ve smyslu podmínek pro reprezentaci neurčitosti a nepřesnosti [Smets 1997]. Jde o syntézu dílčích technik fuzzy modelování do jednoho komplexního matematického modelu. Takovým modelem v oblasti fuzzy relačních databází je např. GEFRED model [Medina et al. 1994]. Nevýhodou těchto modelů je často jejich velká složitost a obtížná použitelnost.

Alternativním přístupem k rozšiřování objektových databázových modelů o fuzzy konstrukty je ponechání existujících databázových konceptů jako výchozích bodů a výběr těch fuzzy rozšíření, které jsou jim příbuzné s cílem sestavit množinu rozšíření, která budou hladce integrována do současných perzistentních systémů a rozhraní. Předmětem dalšího zájmu bude zejména druhý z uvedených přístupů.

2.5.3.2. Od konceptuálního modelu k modelu databázovému

Při přechodu od konceptuálního modelu k modelu databázovému lze využít architektury MOF, která dovoluje na jednotlivých úrovních obohacovat sémantiku modelů. Jak již bylo výše naznačeno, MOF rozeznává následující úrovně:

- Úroveň M3 – vrstva *meta-meta modelu* slouží ke specifikaci meta-modelů úrovně M2.
- Úroveň M2 – vrstva *meta modelu*. Typickým modelem úrovně M2 je UML meta-model, který popisuje samotné UML.
- Úroveň M1 – vrstva *modelu*. Sem je možné zařadit např. vlastní modely v UML.

- Úroveň M0 – vrstva *dat*. Modely na úrovni M0 slouží k zachycení objektů reálného světa.

Rozšíření UML jsou realizována na úrovni M2, a přestože byl tento přístup před časem kritizován [Atkinson 2000], většina současných rozšíření jde touto cestou. Vazba mezi vrstvami v UML architektuře je koncipována výhradně pomocí vztahů typu *instance-of* (instance třídy). Přesněji, elementy úrovně M1 jsou instancemi elementů na úrovni M2 a elementy úrovně M0 jsou instancemi jak M1, tak M2 úrovní (tento postup je známý jako přístup volného meta-modelu). Hlavním mechanismem rozšiřování UML je koncept stereotypu, který definuje virtuální podtřídu UML meta-třídy. Umožňuje definici nových meta-atributů a rozšířenou sémantiku. Profilem je pak stereotypový UML balík, který obsahuje požadovaná sémantická rozšíření.

2.5.3.3. Fyzické uložení modelů v objektových databázích

Objektové databáze se ve svých modelech fyzického ukládání dat dosti značně liší. Používané jsou architektury od serverově orientovaných dotazovacích jazyků (např. CA-Jasmine), až po modely založené na objektových cache systémech, které přesouvají zátěž spojenou se zpracováváním dotazů na stranu klientských aplikací. Tyto aplikace se pak starají o vlastní výpočet hodnot příslušností a vyžadují proto specifické cluster řešení. Bez ohledu na navržené standardy (např. ODMG, JDO), neexistuje dosud společná obecně používaná architektura. A právě z tohoto důvodu musí být realizace fuzzy rozšíření vždy bedlivě prověřena vůči stávající databázové architektuře.

Jednou z hlavních vlastností objektových databází je navigační přístup, který využívá referencí mezi databázovými objekty. Jde tedy o určité zobecnění mechanismu ukazatelů z objektových programovacích jazyků. Databázové reference používají mechanismus nepřímého přístupu ze sekundární paměti do primární [Tarr 1995]. To vede k tomu, že se v mnoha případech objektové databáze snaží udržet objekty na stejných fyzických adresách pro minimalizaci operací spojených se změnou všech referencí na ně. Z výkonnostních důvodů bývají rovněž pohromadě uchovávány i objekty stejné třídy.

2.5.3.4. Reprezentace tříd a jejich asociací prostřednictvím α řezů

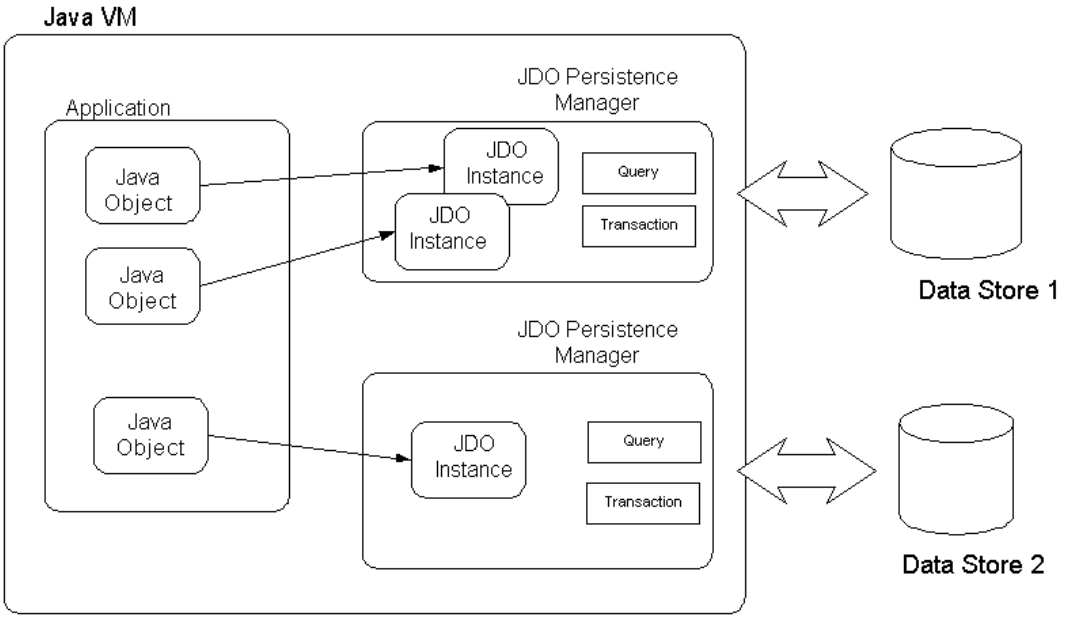
Stupně příslušnosti ve fuzzy třídách nebo stupně participace na fuzzy asociacích jsou obvykle reprezentovány prostřednictvím intervalu [0, 1]. To znamená, že každý objekt třídy nebo asociace může mít teoreticky jiný stupeň příslušnosti. Při práci s velkým množství objektů mohou být operace s fuzzy množinami časově velmi složité. Časovou složitost je možné účinně snížit, pokud je možné se omezit jen určité předem stanovené hodnoty funkce příslušnosti, tj. fuzzy množiny v databázích reprezentovat pomocí α řezů. Tato technika byla navržena v [Boss 1999].

Jde vlastně o určitý optimalizační kompromis mezi ostrými a fuzzy množinami. Uplatnit ho lze v případě, že ztráta rozlišovací úrovně celého intervalu hodnot stupně příslušnosti není podstatná. Zároveň je potřeba zohlednit tento způsob reprezentace fuzzy množin v návrhu datových struktur ukládaných objektů.

2.5.3.5. Fuzzy přístup s využitím JDO

Java Data Objects (JDO) API je standardním Java modelem abstrakce pro zajištění perzistence dat [Eckel 2001]. Model byl vytvořen v rámci Java Community Process [Sun 2006], ve kterém je nyní koncentrováno úsilím původní ODMG skupiny. JDO poskytuje vývojářům standardizované API pro ukládání objektových modelů do nejrůznějších typů úložišť (včetně relačních, objektově-relačních a čistě objektových databází). Poskytuje obecné rozhraní bez ohledu na konečné fyzické řešení vlastního ukládání dat. JDO úzce souvisí s obecnějším DAO návrhovým vzorem pro oddělení jednotlivých vrstev aplikace a jeho popis lze najít např. v [Johnson et al. 2004], [Sun 2004] a [Spell 2002]. O tom, že se dnes zdaleka nejedná o koncept použitelný pouze na platformě Java, svědčí mnoho implementací v dalších jazycích jako např. v PHP viz [Castagnetto 2001], [Kosek 1998], [Zend 2006], nebo v prostředí jazyků .NET viz např. [Písek 2003], [Microsoft 2003].

Základní schéma architektury JDO je zobrazuje Obrázek 10. Aplikační objekty jsou spravovány pomocí tzv. JDO manažera perzistence, který zajišťuje ukládání a načítání objektů z/do datových zdrojů (RDBMS, OODBMS apod.). Manažer perzistence se stará rovněž o správu transakcí a provádění dotazů definovaných pomocí jazyka JDOQL, který vychází ze standardu OQL 3 organizace OMG.



Obrázek 10 - Architektura JDO

Instance schopné perzistence musí v JDO náležet do třídy, která implementuje rozhraní `PersistenceCapable`. Třídy mohou rozhraní implementovat přímo, nebo jim může být doplněno nástroji, které automaticky modifikují zdrojový Java kód, případně generovaný byte kód.

Navigační přístup mezi perzistentními objekty může být realizován pomocí manažera perzistence, resp. voláním jeho metody s názvem `getExtent`. Ta vrací kolekci všech instancí dané třídy. JDO dále poskytuje metodu `makePersistence` pro převod objektu do perzistentního stavu, a to na principu dosažitelnosti, tj. každá další instance, která je s daným objektem propojena je rovněž převedena do perzistentního stavu.

Přidání podpory fuzzy přístupu k třídám prostřednictvím JDO vyžaduje dva typy rozšíření. Na jedné straně musí být rozšířeno programové API rozhraní pro explicitní manipulaci se stupni příslušnosti (formou zpětně kompatibilní se standardním chováním). Na druhé straně musí být obohacen dotazovací jazyk tak, aby byl flexibilnější a zároveň neporušil původní syntaxi nebo sémantiku.

Rozšíření navigačního přístupu je v zásadě otázkou úpravy třídy tak, aby mohla uchovávat hodnoty příslušnosti pro každý objekt. Dosáhnout této změny bez zásahu do sémantiky Java kolekcí je možné díky obecnosti Java kontejneru, který je založen na ukládání jakéhokoliv referenčního typu, tj. např. instance základní třídy `Object`. Jde o obdobu principu uvedeném v [Sicilia 2002]. Původní JDO manažer perzistence

je nutné zapouzdřit do nové třídy `FuzzyPersistenceManager`, které poskytuje stejné rozhraní a zároveň vnitřně zpracovává stupně příslušnosti.

```
Extent e = null;
try {
      pm.currentTransaction().begin();
      e = pm.getExtent(myclasses.X, true, "asc:min=0.2");
} catch(javax.jdo.JDOException) {…}
```

Druhý parametr (`true`) předaný funkci `getExtent` označuje, že mají být získány i instance všech známých podtříd. Třetím parametrem (`asc; min=0.2`) jsou nastaveny vlastnosti fuzzy množiny výsledku (řazení a provedení α řezu na hladině 0.2).

Příklad průchodu kolekcí objektů se stupni příslušnosti do fuzzy množiny:

```
FuzzyExtent fe = (FuzzyExtent)e;
it=e.fuzzyIterator();
while(it.hasNext()) {
      FuzzyObject aux = (FuzzyObject) it.next();
      X anX = (X) aux.getObject();
      double mu = aux.getMembership();
}
```

Při iteraci pomocí `fuzzyIterator()` je možné z objektů získat jejich stupně příslušnosti do fuzzy množiny specifikované databázovým dotazem. Pokud je namísto metody `fuzzyIterator()` použita výchozí JDO metoda `iterator()`, pak je realizován jen základní průchod kolekcí. Získání objektů tímto způsobem ponechá sémantiku JDO rozhraní nezměněnu a zároveň je garantována zpětná kompatibilita.

Dotazovací jazyk JDOQL používá pro definici dotazů Java syntaxi. Jde v podstatě o booleovské filtry nad kolekcemi objektů. Jelikož dotazy jsou zadávány jako řetězce, je z důvodů požadavku na zachování nezměněné syntaxe vhodné umístit fuzzy operace přímo do operátorů. Rozšířený dotaz tedy může vypadat následovně:

```
String filter =
       "address.state == state && " +
       "salary >= && " +
       "department.name.startsWith(deptName) && " +
       "projects.contains(proj) && " +
       "proj.budget > 10000000";
Extent extent = pm.getExtent(
       ProductiveEmployee.class, true, "asc:min=0.01");
Query query = pm.newFuzzyQuery(extent, filter);
((FuzzyQuery)query).interpretAllFuzz();
query.declareImports("import Project");
query.declareVariables("Project proj");
query.declareParameters("String state, String deptName, int sal);
Collection result = (Collection)query.execute(
   "Georgia", "Network", new Integer(100000));
```

V uvedeném příkladu je třída `ProductiveEmployee` podtřídou zaměstnanců, kteří za poslední čtvrtletí pracovali dobře na základě nepřesných kritérií. Instance jsou nejprve filtrovány na minimální stupeň příslušnost 0.01 a poté předány s běžným JDOQL dotazem do objektu, který umí zpracovávat fuzzy operace. Volání metody `interpretAllFuzzy()` signalizuje procesu pro zpracování dotazu, že všechny operátory ve filtrech mají být interpretovány ve fuzzy termínech. Alternativně může být interpretace fuzzy operací v určitých filtrech vyžádána pomocí rozhraní `FuzzyQuery`. Tento postup rozšiřování JDOQL je podobný tomu, který je použit v fJDOQL [Sicilia 2002] a dává fuzzy přístup jako volitelnou vlastnost, jelikož ostatní operace nemusí s hodnotami příslušnosti pracovat.

2.5.3.6. Vliv fuzzy přístupu na fyzické struktury dat v ObjectStore

Databázový systém ObjectStore [Progress 2006] je jedním z nejvyspělejších a nejstabilnějších produktů na trhu objektově orientovaných databázových systémů. Nabízí vysoce výkonnou architekturu nazvanou Virtual Memory Mapping Architecture (VMMA), která využívá tzv. seskupování (clustering) objektů, které jsou často používány společně [Hansen 1999]. To umožňuje vývojářům navrhnout struktury dat, které jsou vysoce optimalizované z pohledu doby přístupu k nim. Architektura VMMA nabízí klient-server přístup, který může být dále optimalizován pro minimální přesuny dat mezi objektovou databází a klientem, a to pomocí vhodné distribuce objektů v blocích s pevnou velikostí (označovaných jako *clustery*).

Jednotlivé bloky se nacházejí v úložištích s proměnlivou velikostí (označovanými jako *segmenty*).

VMMA spoléhá na mechanismus v klientské aplikaci, který produkuje tzv. výpadky stránek ve virtuální paměti procesu pokaždé, když je použit ukazatel nebo reference na perzistentní objekt. Pokud se objekt v adresovém prostoru klienta nachází, pak je přímo přenesen do adresového prostoru aplikace. Pouze v případě, že objekt na klientovi nalezen není, pak dochází k obsluze výpadku ze sekundárního úložiště.

Cache afinita je obecným termínem pro označení míry, jakým data používaná v programu přesahují data získaná v rámci předchozích požadavků [Visnick 2003]. Cache afinita je kritická pro výkon aplikací, jelikož minimalizuje přesun dat mezi klientem a serverem díky velkému množství požadavků, které jsou vybaveny z lokální cache na klientovi. Datová afinita závisí na množině databázových stránek, které klient v daný čas potřebuje (pracovní oblast). Tudíž objekty, které jsou obvykle používány dohromady, by měly být umístěny ve fyzickém úložišti spolu tak, aby mohly být získány ve stejných datových stránkách a tím minimalizovaly klientské požadavky na databázi. Naopak zřídka používané objekty musí být drženy odděleně od těch často používaných. Termín *clustering* slouží k označení procesu seskupování dat, která jsou čtena, resp. aktualizována obvykle ve stejný čas. Dokumentace k ObjectStore uvádí několik kritérií pro design fyzického ukládání, včetně indexů, výběru vhodných struktur a dokonce i refactoringu designu tříd.

Při práci s třídami využívajících fuzzy přístup vystupují parametrické dotazy často jako filtry, které používají stupně příslušnosti k výběru objektů v závislosti na daném α řezu. Jelikož fuzzy dotazy nejsou součástí ObjectStore, zajištění filtrování zůstává na klientovi a tudíž je nutné před vyhodnocením dotazu získat celou kolekci stupňů příslušností. Pokud by byl použit objektový *clustering*, stupně příslušnosti by reprezentovalo pole uvnitř fyzických struktur objektu, takže každý fuzzy dotaz by vyžadoval přenos celé objektové struktury, a tím by způsobil značnou degradaci výkonu. Tato situace ukazuje nutnost oddělení fuzzy zobrazení od zbytku informací v objektech podporujících fuzzy přístup. Tato separace objektů a jejich stupňů příslušnosti je konkrétní realizací techniky „Head-Body Split“ popsané ve [Visnick 2003]. Jeho obecný databázový návrhový vzor shrnut v následujícím jednoduchém schématu delegace pomocí Java deklarace:

```
//Původní třída

public class FuzzyClass {
  //field declarations
  private X1 x1;
  private X2 x2;
  ...
  private XN xN;

  //membership grade
  private double mu;

  //methods
  ...
}
```

```
Výsledek rozdělení

public class FuzzyClass {
  //membership grade
  private double mu;
  //helper instance
  private FuzzyClass_Crisp aux;
  //constructor
  public FuzzyClass(...) {
    aux = new
      FuzzyClass_Crisp(...);
    ...
  }
  //accessors for membership
  public double getMu() {
    returm mu;
  }
  ...
  //methods delegated to the
  //FuzzyClassCrisp
}

public class FuzzyClass_Crisp {
  //all method and
  //field declarations
  //except those releated
  //to fuzziness
}
```

Jakmile je třída rozdělena na dvě, databázový designér musí alokovat instance třídy `FuzzyClass_Crisp` v oddělených fyzických jednotkách tak, aby byla nutná pro filtraci pouze jednoduší část instance dané třídy s fuzzy chováním. Tím je podstatným způsobem snížen přesun potřebných dat.

V případě fuzzy asociací by měla být kolekce, která udržuje přiřazení párů instancí uložena izolovaně v nezávislých blocích. Klient tak může získat celou fuzzy podmnožinu kartézského součinu, vybrat požadovaná propojení a poté vrátit podmnožinu párů instancí, které jsou relevantní s ohledem na jejich stupně příslušnosti. Odůvodnění pro tuto techniku jsou analogická k technice „Isolovaný index“ popsané v [Visnick 2003]. Souhrnně lze tedy říci, že objektové architektury založené na *cache* systémech vyžadují provedení výpočtu se stupni příslušnosti na

straně klienta, a proto by měly být stupně příslušnosti a fuzzy asociace potřebné v pracovní oblasti aplikace umístěny do clusteru pohromadě.

2.5.4. Modelování funkcí příslušnosti

Průběh funkce příslušnosti by měl v ideálním případě přesně odpovídat požadované klasifikaci prvků do fuzzy množiny. Modelování ideálních funkcí příslušnosti by však nezřídka vedlo k vysokým požadavkům na výpočetní výkon. V praxi se proto uplatňují zjednodušené tvary funkcí příslušnosti, které klasifikaci příslušnosti prvků do fuzzy množin urychlují. Je potřeba si však uvědomit, že míra zjednodušení zcela zásadním způsobem rozhoduje o reálné použitelnosti IS pracujících s vágně definovanými pojmy.

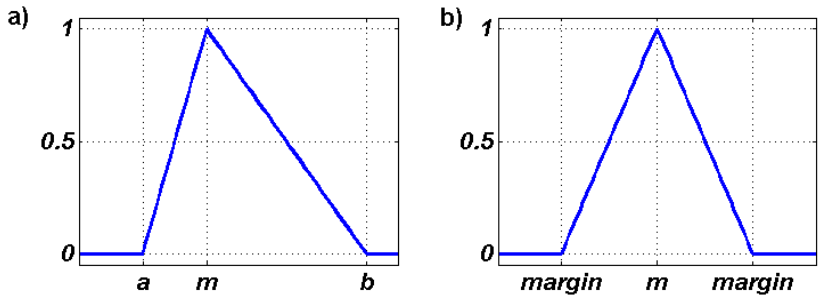
V následujících odstavcích jsou proto uvedeny nejběžnější typy aproximovaných funkcí příslušnosti a některé postupy vedoucí k odvození jejich průběhu.

2.5.4.1. Typy funkcí příslušnosti

V následujícím přehledu jsou uvedeny některé nejběžnější typy funkcí příslušnosti pro použití v později navrženém řešení.

1. Trojúhelníkové

Trojúhelníkové fuzzy množiny (na univerzu X) jsou určeny svojí spodní mezí a, horní mezí b, typickou hodnotou m a platí pro ně, že $a < m < b$. Rozdíl hodnot $b - m$ se nazývá okraj (angl. margin).



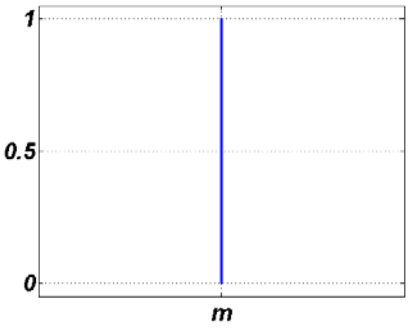
Obrázek 11 - Trojúhelníkové fuzzy množiny: a) obecná, b) symetrická

Průběh funkce:

$$A(x)=\begin{cases}0 & x\le a\\ (x-a)/(m-a) & jestliže \quad x\in(a,m\rangle\\ (b-x)/(b-m) & x\in(m,b)\\ 0 & x\ge b\end{cases}$$

2. Jednoprvkové

Pro všechny hodnoty univerza je stupeň příslušnosti roven 0 s výjimkou hodnoty *m*, kde je 1. Jde o reprezentaci ostré (crisp) hodnoty.



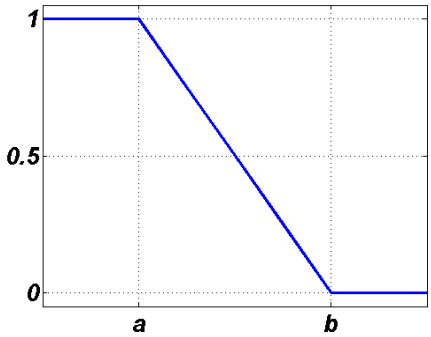
Obrázek 12 - Jednoprvková fuzzy množina

Průběh funkce:

$$\text{sign}(x)=\begin{cases}0 & x\ne m\\ 1 & x=m\end{cases} \quad jestliže$$

3. L funkce

Funkce je definována dvěma parametry. Parametr *a* je hraniční hodnotou, kdy stupeň příslušnosti již není 1 a parametr *b* značí nulový stupeň příslušnosti. Průběh klesání je lineární.



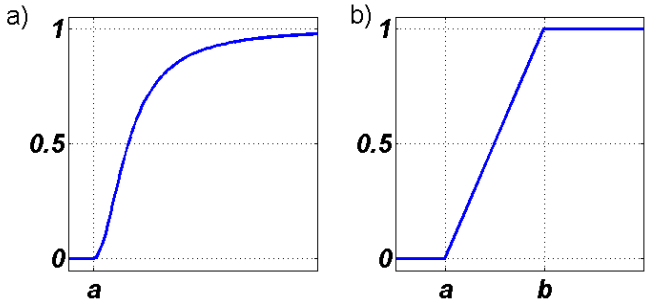
Obrázek 13 - L fuzzy množina (pravostranná)

Průběh funkce:

$$L(x) = \begin{cases} 1 & \\ \dfrac{a-x}{b-a} & \textit{jestliže} \quad a < x \le b \\ 0 & \end{cases} \qquad \begin{array}{l} x \le a \\ \\ x > b \end{array}$$

4. Gamma funkce

Funkce je definována dolní mezí a a hodnotou $k > 0$. Používá se buď obecná, nebo lineární Gamma funkce.



Obrázek 14 - Gamma fuzzy množiny: a) obecná, b) lineární

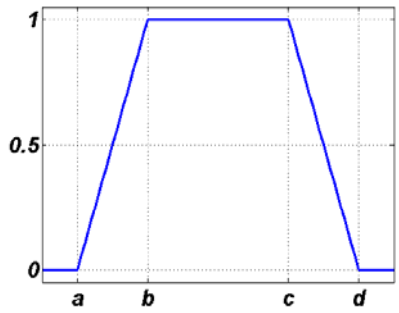
Průběh funkce:

$$\Gamma(x) = \begin{cases} 0 & \\ \dfrac{k(x-a)^2}{1+k(x-a)^2} & \textit{jestliže} \end{cases} \qquad \begin{array}{l} x \le a \\ x > a \end{array} \quad \text{(obecná)}$$

$$\Gamma(x) = \begin{cases} 1 & \\ \dfrac{x-a}{b-a} & \textit{jestliže} \quad a < x \le b \\ 0 & \end{cases} \qquad \begin{array}{l} x \le a \\ \\ x > b \end{array} \quad \text{(lineární)}$$

5. Lichoběžník

Funkce definována dolní mezí a a horní mezí b. Dále pak horní mezí d a dolní mezí c svého jádra (se stupněm příslušnosti 1).



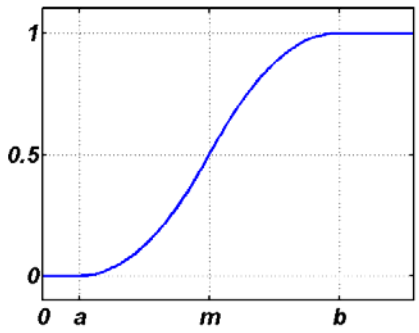
Obrázek 15 - Lichoběžníková fuzzy množina

Průběh funkce:

$$T(x)=\begin{cases}0 & (x\le a)\ nebo\ (x\ge d)\\ (x-a)/(b-a) & x\in(a,b\rangle\\ 1 & x\in(b,c)\\ (d-x)/(d-c) & x\in(b,d)\end{cases}\quad jestliže$$

6. S funkce

Funkce určena svojí dolní mezí *a*, horní mezí *b* a inflexním bodem *m* tak, že platí $a<m<b$. Typická hodnota *m* je obvykle dána jako $m=(a+b)/2$. Růst funkce je pomalejší, pokud se vzdálenost *a* a *b* zvětšuje.



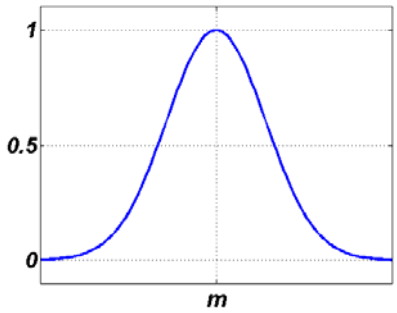
Obrázek 16 - S fuzzy množina

Průběh funkce:

$$S(x)=\begin{cases}0 & x\le a\\ 2\{(x-a)/(b-a)\}^2 & x\in(a,m\rangle\\ 1-2\{(x-a)/(b-a)\}^2 & x\in(m,b)\\ 1 & x\ge b\end{cases}\quad jestliže$$

7. Gaussova funkce

Klasická Gaussova křivka daná svojí střední hodnotou *m* a hodnotou parametru *k*, který je větší než 0. Čím je *k* větší, tím „užší“ je tvar, který graf funkce tvoří.



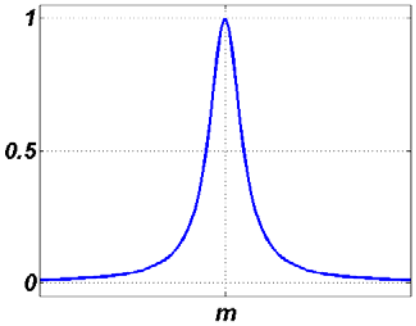
Obrázek 17 - Gaussova fuzzy množina

Průběh funkce:

$$G(x) = e^{-k(x-m)^2}$$

8. Pseudo-exponenciální

Funkce určená podobně jako Gaussova svojí střední hodnotou m a parametrem k, který je větší než 1. Čím je parametr k větší, tím rychleji funkce roste a tvar, který graf funkce tvoří, je „užší“. Tato funkce má tedy podobné vlastnosti jako Gaussova funkce, bude však obvykle výpočetně méně náročná.



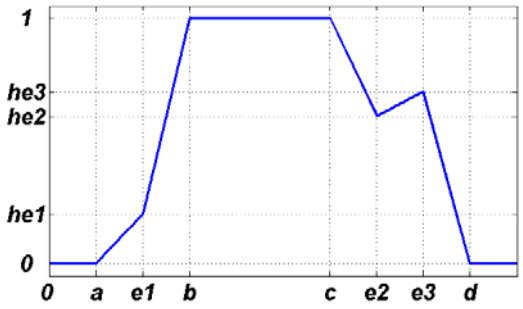
Obrázek 18 - Pseudo-exponenciální fuzzy množina

Průběh funkce:

$$P(x) = \frac{1}{1 + k(x-m)^2}$$

9. Po částech lineární funkce

Průběh funkce je definován čtyřmi hodnotami lichoběžníku a, b, c, d a dále seznamem bodů e_i, $i = 1, 2, \ldots, N$ mezi a a b a/nebo mezi c a d. U každého bodu e_i je zároveň uveden stupeň příslušnosti he_i.



Obrázek 19 - Fuzzy množina určená po částech lineární funkcí

2.5.4.2. Stanovení vhodného typu funkce příslušnosti

V předchozím shrnutí bylo uvedeno několik základních typů funkcí příslušnosti, které jsou z pohledu informačních systémů relativně snadno použitelné. Pro určení fuzzy množin, které nejlépe vystihují určitou vlastnost objektů univerza, je možné použít některou z následujících základních metod:

1. Horizontální metoda

Metoda založená na odpovědích N expertů na stupeň příslušnosti prvků univerza x_1, x_2, …, x_n. Otázky jsou formulovány ve tvaru:

„Je x akceptováno jako slučitelné s pojmem A?“

Přípustné odpovědi jsou pouze „Ano“ nebo „Ne“. Výsledný stupeň příslušnosti prvku x_i do fuzzy množiny A je tedy dán poměrem mezi počtem kladných odpovědí a celkovým počtem odpovídajících expertů N:

$$A(x_i) = (\text{počet kladných odpovědí})/N \qquad \text{, kde } i = 1, 2, \ldots, n.$$

Tato metoda je schopna poskytnout spolehlivé výsledky zvlášť, pokud je doplněna o statistické posouzení významnosti směrodatné odchylky.

2. Vertikální metoda

Cílem metody je sestavit skupinu α-řezů fuzzy množiny A, ke kterým mohou být prvky univerza X přiřazeny. Experti identifikují odpovídající podmnožiny X, jejichž prvky patří do A minimálně se stupněm příslušnosti α. Fuzzy množina je následně rekonstruována jako kombinace po sobě jdoucích α-řezů.

Podobně jako horizontální metoda může vertikální poskytnout kvalitní výsledky. Společnou nevýhodou obou metod je však relativní izolovanost experimentů.

3. Metoda založená na optimalizaci parametrů

Účelem metody není nalezení zcela nové funkce příslušnosti, ale přizpůsobit některou ze standardních funkcí (viz Kapitola 2.5.4.1) tak, aby co nejlépe odpovídala experimentálním datům. Data jsou ve formátu uspořádaných dvojic (prvek, stupeň příslušnosti) a označeny jako (x_k, $M(x_k)$). Postup odhadu parametrů standardních funkcí je následující.

Předpokládejme funkci příslušnosti $A(x, p)$, kde $x \in X$ (prvek univerza) a p je vektor parametrů funkce. Vektor p je třeba získat se zohledněním všech experimentálních dat (x_k, $M(x_k)$), kde k = 1, 2, …, N. Nejčastěji je jako kritérium pro odhad používána metoda minimalizace čtverců odchylek:

$$\min_p \sum_{k=1}^{N} \left[M(x_k - A(x_K; p) \right]^2 .$$

4. Metoda fuzzy shluků

Jde o metodu analýzy dat, jejímž cílem je najít shluky v nějakém smyslu podobných dat. V klasickém pojetí jsou shluky disjunktní podmnožiny nějaké množiny prvků. Protože mohou shluky pocházet z rozmanitých zdrojů, jeví se požadavek disjunktnosti a ostrosti množin jako příliš silný. To vedlo k zavedení fuzzy shlukové analýzy, v niž jsou shluky fuzzy množinami prvků, které se mohou překrývat.

Základním pojmem ve shlukové analýze je *vzdálenost* mezi objekty. Může to být součet absolutních hodnot rozdílů souřadnic, maximum z absolutních hodnot rozdílů souřadnic, eukleidovská vzdálenost, nebo libovolná jiná funkce d: $V \times V \to R$, která má tyto vlastnosti:

$$d(x,x) = 0 ,$$

$$d(x,y) = d(y,x) ,$$

$$d(x,z) \le d(x,y) + d(y,z)$$

platí pro všechna $x, y, z \in V$. Právě způsobem definice vzdálenosti bodů se od sebe liší různé metody shlukové analýzy.

Podstatou shlukové analýzy je hledání vhodného rozkladu množiny V, tj. hledání takového systému podmnožin (shluků), který splňuje požadavek, že prvky v jednom

shluku byly vzájemně „blíže“, než prvky z různých shluků. Ve fuzzy shlukové analýze existuje podmínka, že každý prvek $v \in V$ musí patřit alespoň do jednoho shluku, přičemž součet všech stupňů jeho příslušnosti do všech shluků musí být roven 1. Zároveň platí podmínka, že do žádného shluku nesmí v maximálním stupni patřit všechny prvky množiny V a každý shluk musí být neprázdný.

Existuje více algoritmů pro fuzzy shlukovou analýzu z nichž mezi nejpoužívanější patří algoritmus „isodata“, nebo jeho zobecněná podoba nazývaná „c-průměry“. Detailní popis těchto algoritmů je mimo rozsah této práce a lze ho najít např. v [Bezdek 1981].

3. Kritické zhodnocení současných přístupů

Výše uvedené myšlenky, přístupy, zobecněné datové modely, aplikační rámce se zásadním způsobem zasloužily o rozvoj celé problematiky. Stanovily základní předpoklady pro použití fuzzy přístupu v objektových databázích a zároveň definovaly některá omezení a jejich zdroje. Přehled klíčových prací z oblasti vývoje i aplikace lze nalézt např. v nedávno publikovaném přehledu „Advances in Fuzzy Object-Oriented Databases“ [Ma 2005]. Většina z uvedených konceptů je velmi detailně propracovaná a nabízí pro konkrétní podmínky dobře použitelná řešení.

Řada z těchto řešení je však zaměřena především na oblast budování fuzzy informačních systémů tzv. „na zelené louce“. Patří mezi ně i pravděpodobně nejpropracovanější FOOD, FRIL++ a GEFRED modely [Ma 2005]. Tato řešení předpokládají naprostou volnost při návrhu datových a procesních modelů a nejsou zatíženy stávajícími systémy a především formou uložení existujících dat. V okamžiku nasazení modelu s podporou fuzzy přístupu je proto nutné obvykle provést náročnou datovou a procesní migraci stávajících systémů a dále přizpůsobit rozhraní pro zajištění spolupráce s ostatními systémy (např. ETL procesy, databázová a aplikační rozhraní k datům). Takové změny mohou být velice náročné a nezřídka dokonce neakceptovatelné.

Objevují se samozřejmě i přístupy, které se snaží doplňovat existující systémy jako např. model Sicilia, Garcia-Barrocanal a Gutiérrez [Sicilia 2002]. Výhodou tohoto modelu je implementace fuzzy přístupu formou doplňků současných databázových systémů a jejich rozhraní do programovacích jazyků jako Java a C++. Zpětná kompatibilita s přístupem ostrých (crisp) množin je tak zachována. Jejich těsná integrace s konkrétními platformami však omezuje přenositelnost řešení a vyžaduje pro každou objektově orientovanou databázi použít jiný postup.

Bez ohledu na to, zda jde o přístupy revoluční nebo spíše evoluční, nedospěl zatím žádný z uvedených modelů do stádia implementace do jádra některého z komerčních či nekomerčních produktů světa OODBMS. Tato skutečnost naznačuje potřebu se zaměřit především na srozumitelnost, snadnost použití a akceptaci nových přístupů širokou vývojářskou veřejností. Teprve v okamžiku pochopení možností a způsobu jejich realizace se fuzzy přístup v databázích stane alternativou ke klasickým přístupům a navíc s sebou ponese přidanou hodnotu v podobě pokročilé sémantiky.

Na základě reálných potřeb pak může vzniknout poptávka po databázových systémech, ve kterých již bude fuzzy přístup představovat jejich nedílnou součást.

Cestu k zpřístupnění fuzzy principů veřejnosti mohou představovat aplikované modely ve formě aplikačních rámců. Skupina Berzal, Marín, Pons a Vila [Ma 2005] představila v nedávné době aplikační rámec, který by měl být použitelný jak nad OODBMS, tak nad hybridními ORDBMS (objektově relační databázové systémy jako je Oracle). Přestože jde o jeden z mála velmi srozumitelných a použitelných rámců, jeho nasazení je možné jen za určitých předpokladů. Jedním z nejvíce omezujících je zásah do hierarchie tříd, kdy každý objekt, který má být uložen v databázi a má podporovat fuzzy přístup musí být instancí třídy odvozené od společného (fuzzy) předka. Dalším omezením může být některá porušení čistého objektového přístupu při definici fuzzy atributů. Případné změny ve struktuře tříd (refactoring) jsou následkem toho obtížnější. Aplikační rámec zatím není dokončen, ale již nyní pokrývá řadu fuzzy aspektů a byl pro autora této práce inspirací především z pohledu použitelnosti a obecnosti řešení.

Celá řada dalších postupů jak zachytit fuzzy vlastnosti objektů v databázích je založena na principu zapouzdřujících tříd (wrappers), které jsou nejčastěji implementovány pomocí návrhových vzorů Dekorátor nebo Adaptér [Gamma et al. 2003]. Jednoduchost řešení si vybírá daň zejména v podobě velkého nárůstu počtu objektů v databázích, a tím snížení výpočetního výkonu. Pro malé objemy dat nicméně může jít o elegantní řešení. U větších objemů dat je nutné řešit pro konkrétní OODBMS platformu optimalizaci způsobu ukládání, jak je možné vidět např. u řešení pro ObjectStore, které bylo publikováno v [Sicilia 2002] již zmíněnou skupinou Sicilia, Garcia-Barrocanal a Gutiérrez.

Specifickým způsobem implementace fuzzy rozšíření do OODBMS je úprava standardních databázových rozhraní jako JDO nebo ADO.NET. Logika je tak ponechána zcela na klientském systému a nezřídka je nutné porušit čistě objektový přístup (např. textově orientovaná rozšíření jazyka JDQL pro rozhraní JDO).

Mezi největší společné nedostatky výše uvedených řešení lze dále doplnit omezenou podporu odlišných kontextů pro fuzzy výrazy (dotazy), náročnost při změně databázového schématu a nízkou přenositelnost mezi platformami. Opomíjeny bývají rovněž požadavky vycházející z životních cyklů vývoje software, metodik a požadavků na jakost, jak bude uvedeno v následujících částech této práce.

4. Vývoj software s využitím fuzzy přístupu

Řízení softwarových projektů, ve kterých je využíváno technik či postupů, které nejsou zcela běžné bývá často náročnější než je tomu u klasických projektů, kdy je předmětem dodávky více či méně standardizované softwarové řešení. Využití principů fuzzy logiky v softwarových projektech se dotýká téměř všech fází vývoje. Mezi oblasti, na které má největší dopad patří:

- *Sběr požadavků*. Největším úskalím úvodní fáze softwarového projektu bývá sjednocení myšlenkového modelu, tj. jak zákazník a dodavatel chápou základní pojmy, které nemusí být v případě použití fuzzy logiky z počátku zcela striktně definovatelné (např. bonitní klient, zvýšená teplota reaktoru, slabý brzdný účinek). Fuzzifikace [Novák 2000] těchto pojmů velmi pravděpodobně celou úvodní fázi projektu podstatným způsobem prodlouží. Základním východiskem pro řešení nárůstu složitosti řízení takového projektu představuje vhodně zvolená metoda sběru požadavků umožňující rozlišit různé typy projektů. Příkladem může být metoda Volere [Robertson 2006].
- *Odhad pracnosti*. Náročnější implementaci „fuzzy požadavků“ je užitečné zohlednit v parametrech modelů pro odhad pracnosti (např. Metoda funkčních bodů nebo COCOMO (Constructive Cost Model) [Boehm 2000]).
- *Analýza a softwarový design řešení*. V této fázi jsou na analytika kladeny zvýšené nároky z pohledu schopností odhadnout časový vývoj požadavků i jednotlivých definovaných fuzzy pojmů (lze např. očekávat že pojem bonitní klient se bude v čase podstatným způsobem měnit a jak?). Na úrovni softwarového designu je kritické vhodně navrhnout implementaci fuzzy přístupu tak, aby řešení bylo možné znovupoužít při změně požadavků či dokonce platformy. Možnostem, jak postupovat zejména v této fázi, je věnována tato práce.
- *Vlastní vývoj (programování)*. Algoritmizace operací s fuzzy množinami vyžaduje znalosti, které přesahují obvyklé požadavky kladené na aplikačního či databázového programátora.
- *Změnové řízení a dokumentace*. Fuzzy pojmy lze považovat za velmi citlivé vůči vývoji a změně požadavků. Často se mění nejen jejich podoba (průběh funkce příslušnosti), ale také jejich vzájemné vztahy.

- *Testování*. Složitost v předcházejících fázích determinuje náročnější testování na úrovni aplikačních jednotek (unit testing), integračních testů, a především testů splnění uživatelských požadavků.
- *Podpora*. Potvrzení a klasifikace hlášeného nesouladu mezi očekávaným a skutečným „fuzzy chováním“ systému vyžaduje detailnější dokumentaci celého projektu, na jejímž základě lze následně provést odpovídající servisní zásah.

Za cenu nárůstu složitosti návrhu i implementace je možné získat systém, který více odpovídá reálným potřebám a zvláště tam, kde jde o automatizaci tzv. soft skills (např. pocitové a zkušenostní rozhodování) může zásadně rozhodnout o celkové použitelnosti.

5. Vymezení rozsahu řešení

Z důvodů shrnutých v předcházejících oddílech této práce je vhodné navrhnout způsob fuzzy rozšíření datového modelu způsobem, který usnadní jeho průběžné modifikace a umožní přenositelnost aplikací, které jej využijí.

Pro vymezení základních vlastností navrhovaného řešení byla využita struktura charakteristik modelu jakosti software dle standardu ISO popsaná např. v [Vaníček 2006], [Vaníček 2007]:

- *Funkčnost*. Řešení by mělo umožnit definovat příslušnost jednotlivých prvků do fuzzy množin prostřednictvím funkcí příslušnosti, provádět alfa řezy fuzzy množin a podporovat základní operace s fuzzy množinami (negace, konjunkce – trojúhelníkové normy, disjunkce – trojúhelníkové konormy). Kritériem pro rozsah funkčnosti bude přehled uvedený v [Navara 2002].
- *Bezporuchovost (reliability)*. Při výskytu problému uvnitř aplikace (postavené s využitím navrženého řešení) by mělo dojít k řízené propagaci výjimečného stavu do vnějších částí aplikace takovým způsobem, aby ho bylo možné na tento stav korektně zareagovat. Kritériem pro posouzení splnění tohoto požadavku bude kritické zhodnocení možností ošetření chybových stavů ve vybraných programovacích jazycích.
- *Použitelnost*. Základní myšlenkou řešení je možnost jeho opakovaného znovupoužití bez znalostí implementačních detailů, a proto musí poskytovat jasně definované rozhraní, které je intuitivní a jednoduše integrovatelné s cílovým prostředím (programovacím jazykem, nástroji pro vývoj). Pro posouzení rozhraní bude provedeno jeho porovnání s vybranými zástupci běžně používaných rozhraní pro zajištění perzistence dat. Z pohledu konečného uživatele by pak mělo řešení nabídnout relativně komfortní způsob naplnění báze dat neostrými empirickými poznatky.
- *Účinnost*. Časovou i výpočetní složitost by se mělo podařit minimalizovat přesunutím náročných operací co nejblíže k datům, která mají předem neznámou velikost a jsou uložena v externí objektově orientované databázi. K posouzení účinnosti poslouží odhad složitosti při použití řešení za stanovených podmínek.

- *Udržovatelnost*. Navržené řešení by mělo podporovat tzv. body rozšíření pro dodatečnou implementaci dalších operací a principů fuzzy logiky. Zároveň by mělo umožnit transparentní změnu svého vnitřního chování. Možný postup konkrétního rozšíření bude uveden v podobě kroků, které je nutné pro pokrytí nové (nebo změnu stávající) funkcionality provést.
- *Přenositelnost*. Jedním z klíčových požadavků je univerzálnost z pohledu využívané objektově orientované databázové platformy a programovacích jazyků. Obecně navržené fuzzy rozšíření OODBMS by mělo být možné implementovat v podobě např. aplikačního rámce, který bude schopen využít specifických vlastností konkrétní databázové platformy, ale zároveň zachová obecně definované rozhraní.

Stanovením požadavků na navrhované řešení je vymezen jeho rozsah a základní vlastnosti.

5.1. Modelové situace pro navrhované řešení

Softwarový architekt, analytik, programátor a další specialisté si před volbou resp. návrhem konkrétního řešení musí odpovědět na řadu otázek, z nichž některé již byly nastíněny v předcházejících odstavcích. Jednou z nejnáročnějších otázek je definice vhodného rozhraní a jeho použitelnost. Často se vychází z představy nějakého ideálního stavu a tato představa se postupně upravuje až do podoby, která je současnými technickými prostředky a za současného stavu poznání realizovatelná (ve výjimečných situacích se hranice těchto omezení posouvají). K vytvoření této úvodní představy, resp. k jejímu sdílení a dalšímu rozpracovávání, se používá popis modelových situací. Přestože na úrovni sběru požadavků, resp. konceptuální analýzy, používají různé metodiky pro tuto oblast odlišné názvy (Use-Cases v metodikách založených na UML, Object Behavioral Analysis v metodice BORM apod.), jde vždy primárně o pochopení, zda navržené řešení bude odpovídat potřebám a očekáváním. V následujících odstavcích jsou uvedeny základní představy o možnostech navrhovaného řešení podporu fuzzy přístupu v prostředí OODBMS.

5.1.1. Získání fuzzy množiny ze zdrojových dat

Řešení by mělo nabídnou prostředky pro získání fuzzy množiny, která bude určena předpisem funkce příslušnosti. Vstupní data do této množiny budou načtena dle

zadaných podmínek z OODBMS. Výsledek by měl být předán ve struktuře kompatibilní s cílovým prostředím (klientský systém).

Posloupnost kroků:

1. Klientský systém specifikuje funkci příslušnosti (dle kontextu dotazu), která bude použita pro výpočet stupně příslušnosti každého prvku zdrojových dat do výsledné fuzzy množiny (předpis funkce může být načten z OODBMS nebo definován v klientském systému).
2. Klientský systém určí kritéria pro výběr primárních dat z OODBMS ve formě databázového dotazu.
3. Spuštění dotazu nad OODBMS (nativní přístup k OODMBS nebo prostřednictvím obecného rozhraní).
4. Konstrukce fuzzy množiny z odkazu na výsledek databázového dotazu a z předpisu funkce příslušnosti (stanovení stupňů příslušnosti do fuzzy množiny pro každý prvek výsledku).
5. Zpřístupnění fuzzy množiny v klientském systému.

5.1.2. Získání fuzzy množiny s minimálním stupněm příslušnosti prvků

Nezřídka nastává situace, kdy jsou pro konečného uživatele relevantní pouze prvky s určitým minimálním stupněm příslušnosti do fuzzy množiny. V takovém případě je potřeba prvky s nižším než stanoveným stupněm příslušnosti z výsledné fuzzy množiny vyřadit, tj. provést α-řez.

Posloupnost kroků:

1. Klientský systém specifikuje funkce příslušnosti a kritéria pro výběr primárních dat z OODBMS shodným způsobem jako v předcházejícím případě.
2. Konstrukce fuzzy množiny.
3. Stanovení minimálního stupně příslušnost prvků do fuzzy množiny v klientském systému.
4. Zpřístupnění fuzzy množiny v klientském systému.

5.1.3. Získání fuzzy množiny na základě výpočtu pravdivostní hodnoty složené podmínky

Je-li potřeba omezit výběr prvků do fuzzy množiny splněním více podmínek, pak je možné vyjádřit vazbu těchto podmínek pomocí logické formule. Pro každý prvek univerza je následně potřeba vypočíst pravdivostní hodnotu této formule ve fuzzy logice. Výsledkem je hodnota funkce příslušnosti prvku do cílové fuzzy množiny (fuzzy podmnožiny univerza, určené těmi prvky, které splňují všechny stanovené podmínky).

Navržené řešení by mělo nabídnout prostředky pro konstrukci dotazů s libovolným počtem dílčích podmínek. Vazby těchto podmínek (fuzzy operátory) by se neměly omezovat jen na standardní (Zadehovy) logické spojky, ale podporovat i další typy jako jsou součinové nebo Lukasiewiczovy. Kombinacím různých typů logických spojek v rámci jednoho dotazu nebude navržené řešení explicitně zabraňovat (na nebezpečí problémové interpretace získaných výsledků bude upozorněn analytik/uživatel v dokumentaci).

Posloupnost kroků:

1. Klientský systém specifikuje požadované podmínky (dle aktuálního kontextu dotazu), které budou kombinovány a tím bude určena funkce příslušnosti prvků do fuzzy množiny.
2. Volba typů fuzzy operátorů pro sestavení kombinace podmínek (standardní, součinové apod.).
3. Klientský systém určí kritéria pro výběr primárních dat z OODBMS ve formě databázového dotazu.
4. Konstrukce fuzzy množiny.
5. Sestavení a vyhodnocení kombinace podmínek, kde operátory jsou fuzzy negace/konjunkce/disjunkce a operandy jsou fuzzy množiny mezivýsledků.
6. Zpřístupnění fuzzy množiny v klientském systému.

5.2. Funkční pokrytí

V následujícím přehledu jsou vymezeny základní funkce, které je nezbytné do výsledného řešení zahrnout, aby mohl splnit účel, pro který je navrhováno, tj. poskytnout v praxi použitelné fuzzy rozšíření pro OODBMS.

5.2.1. Operace s fuzzy množinami

Mezi základní funkční požadavky zcela jistě patří prostředky pro popis fuzzy množin a logické operace s nimi.

5.2.1.1. Výška fuzzy množiny

Je-li fuzzy množina výšky 1, nazývá se *normální*. V opačném případě se označuje jako *subnormální*.

$$h(A) = \sup Range(A).$$

Výška je dána suprémem pravdivostních hodnot jejích prvků.

5.2.1.2. Nosič fuzzy množiny

Nosič fuzzy množiny A je množina všech prvků, jejichž stupeň příslušnosti do A je nenulový.

$$Supp(A) = \{x \in X : \mu_A(x) > 0\}, \text{ neboli } Supp(A) = \mu_A^{-1}((0,1\rangle).$$

Nosič je ostrá (crisp) množina.

5.2.1.3. Jádro fuzzy množiny

Jádro je množina všech prvků, které určitě patří do fuzzy množiny A. Představují typické prvky (prototypy) pro danou fuzzy množinu. Stupeň příslušnost prvků v jádře je roven 1.

$$core(A) = \{x \in X : \mu_A(x) = 1\}, \text{ neboli } core(A) = \mu_A^{-1}(1).$$

Jádro je ostrá (crisp) množina.

5.2.1.4. Fuzzy negace a fuzzy doplněk

Jestliže daná fuzzy množina reprezentuje určitou vlastnost prvků univerza, pak její funkce příslušnosti vyjadřuje, do jaké míry prvek má tuto vlastnost. *Fuzzy negace*

této hodnoty pak vyjadřuje, do jaké míry prvek tuto vlastnost nemá, neboli nakolik má vlastnost opačnou (popsanou fuzzy doplňkem původní množiny).

Obecná fuzzy negace je každá unární operace $\neg_{\cdot}: \langle 0,1 \rangle \to \langle 0,1 \rangle$, splňující následující axiomy:

$$\alpha \le \beta \Rightarrow \neg_{\cdot}\beta \le \neg_{\cdot}\alpha ,$$

$$\neg_{\cdot}\neg_{\cdot}\alpha = \alpha .$$

Řešení se omezí na tzv. *standardní negaci* označovanou jako $\neg_S$, která je definována vztahem $\neg_S \alpha = 1 - \alpha$. Ostatní typy fuzzy negací bude možné doplnit prostřednictvím bodů rozšíření viz níže.

Fuzzy doplněk je operace na fuzzy množinách definovaná pomocí fuzzy negace:

$$\mu_{\overline{A}^*}(x) = \neg_{\cdot}\mu_A(x) .$$

Označení $\overline{A}^*$ znamená obecný fuzzy doplněk k fuzzy množině A. Standardní doplněk (odpovídající standardní negaci), na který se řešení omezí bude značen jako $\overline{A}^S$. Podmínkami pro provedení operace je definování funkce příslušnosti a vstupní množiny prvků.

Příklad:

Vstup: Fuzzy množina A = {0,4/o1; 0,6/o2; 1/o3; 0/o4}.

Výstup: Standardní doplněk $\overline{A}^S$ = {0,6/o1; 0,4/o2; 0/o3; 1/o4}.

Použité symboly:

o_j, kde j = 1,…n prvky univerza,

n/o_j, kde n je stupeň příslušnosti prvku o_j do fuzzy množiny.

5.2.1.5. Fuzzy konjunkce

Zajímá-li nás stupeň pravdivosti konjunkce dvou vlastností, které jsou popsány fuzzy množinami, lze tuto hodnotu stanovit ze stupně pravdivosti jednotlivých vlastností a to tak, že na ně aplikujeme binární operaci – *fuzzy konjunkci*.

Definiční vlastnosti fuzzy konjunkcí reprezentují přirozené minimální požadavky (komutativita, asociativita, monotonie a okrajová podmínka). Obecná fuzzy

konjunkce je binární operace $\underset{\cdot}{\wedge}:\langle 0,1\rangle^2 \to \langle 0,1\rangle$ splňující následující axiomy pro všechny $\alpha,\beta,\gamma \in \langle 0,1\rangle$:

$\alpha \underset{\cdot}{\wedge} \beta = \beta \underset{\cdot}{\wedge} \alpha$ (komutativita),

$\alpha \underset{\cdot}{\wedge} (\beta \underset{\cdot}{\wedge} \gamma) = (\alpha \underset{\cdot}{\wedge} \beta) \underset{\cdot}{\wedge} \gamma$ (asociativita),

$\beta \le \gamma \Rightarrow \alpha \underset{\cdot}{\wedge} \beta \le \alpha \underset{\cdot}{\wedge} \gamma$ (monotonie),

$\alpha \underset{\cdot}{\wedge} 1 = \alpha$ (okrajová podmínka).

Fuzzy konjunkcí je více typů. Mezi nejpoužívanější zvláštní případy fuzzy konjunkcí patří:

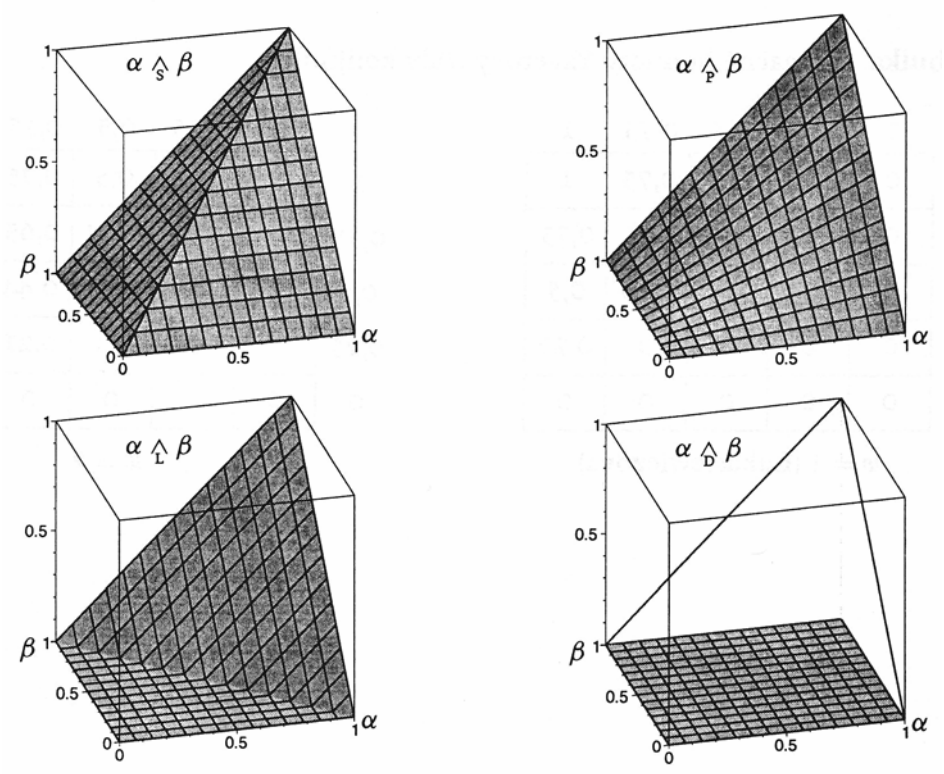
Standardní konjunkce: $\alpha \underset{S}{\wedge} \beta = \min(\alpha,\beta)$.

Součinová konjunkce: $\alpha \underset{P}{\wedge} \beta = \alpha \cdot \beta$.

Lukasiewiczova konjunkce: $\alpha \underset{L}{\wedge} \beta = \begin{cases} \alpha+\beta-1 & pro\ \alpha+\beta-1>0 \\ 0 & jinak \end{cases}$.

Drastická konjunkce: $\alpha \underset{D}{\wedge} \beta = \begin{cases} \alpha & pro\ \beta = 1 \\ \beta & pro\ \alpha = 1 \\ 0 & jinak \end{cases}$.

Standardní fuzzy konjunkce je přitom jediná, která má vlastnost $\alpha \Rightarrow \alpha \underset{\cdot}{\wedge} \alpha$.



Obrázek 20 - Fuzzy konjunkce: standardní, součinová, Lukasiewiczova, drastická

Další typy fuzzy konjunkcí bude možné doplnit prostřednictvím bodů rozšíření viz níže. Postačující podmínkou pro provedení operace budou vstupní množiny prvků a jejich hodnoty stupňů příslušnosti.

Příklad:

Vstup: Fuzzy množina A = {0,4/o1; 0,6/o2; 1/o3; 0/o4}.

Fuzzy množina B = {0,3/o1; 1/o2; 0,1/o3; 0,7/o4; 1/o5}.

Výstup: Fuzzy množina $A \underset{S}{\wedge} B$ = {0,3/o1; 0,6/o2; 0,1/o3; 0/o4; 0/o5}.

Použité symboly:

o_j, kde j = 1,…n prvky univerza,

n/o_j, kde n je stupeň příslušnosti prvku o_j do fuzzy množiny.

5.2.1.6. Fuzzy disjunkce

Zajímá-li nás stupeň pravdivosti disjunkce dvou vlastností, které jsou popsány fuzzy množinami, lze tuto hodnotu stanovit ze stupně pravdivosti jednotlivých vlastností a to tak, že na ně aplikujeme binární operaci – *fuzzy disjunkci*.

Definiční vlastnosti fuzzy disjunkcí reprezentují přirozené minimální požadavky (komutativita, asociativita, monotonie a okrajová podmínka). Obecná fuzzy konjunkce je binární operace $\dot{\vee}:\langle 0,1\rangle^2 \to \langle 0,1\rangle$ splňující následující axiomy pro všechny $\alpha,\beta,\gamma \in \langle 0,1\rangle$:

$\alpha \dot{\vee} \beta = \beta \dot{\vee} \alpha$ (komutativita),

$\alpha \dot{\vee} \left(\beta \dot{\vee} \gamma \right) = \left(\alpha \dot{\vee} \beta \right) \dot{\vee} \gamma$ (asociativita),

$\beta \le \gamma \Rightarrow \alpha \dot{\vee} \beta \le \alpha \dot{\vee} \gamma$ (monotonie),

$\alpha \dot{\vee} 0 = \alpha$ (okrajová podmínka).

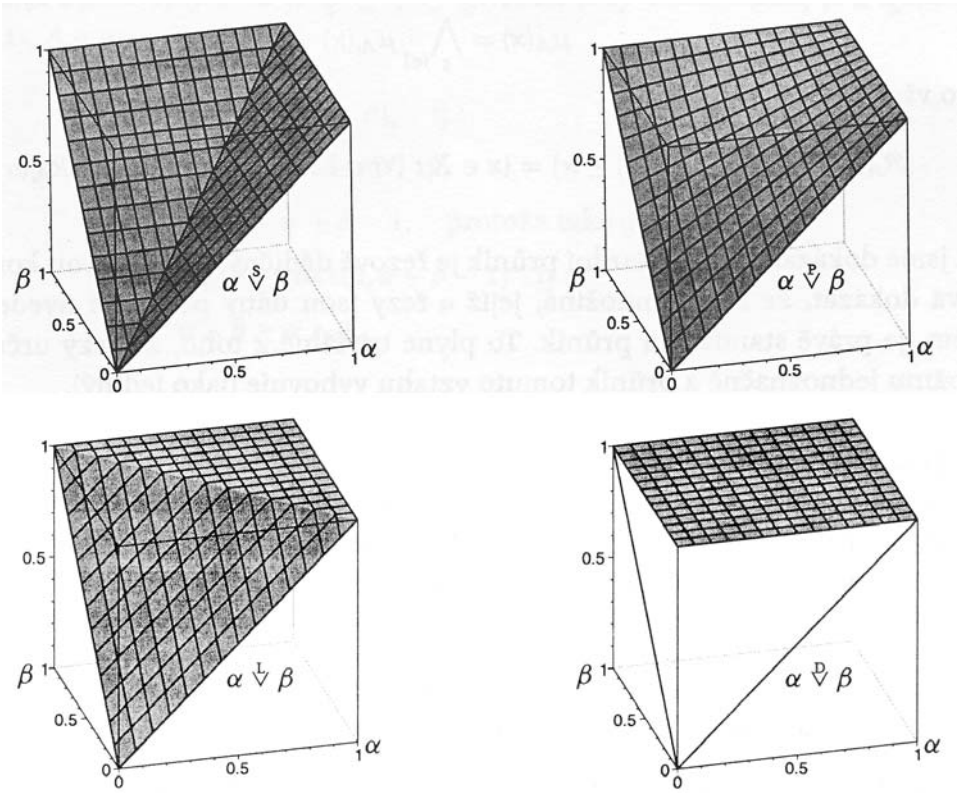
Fuzzy disjunkcí je více typů. Mezi nejpoužívanější zvláštní případy fuzzy disjunkcí patří:

Standardní disjunkce: $\alpha \overset{S}{\vee} \beta = \max(\alpha,\beta)$.

Součinová disjunkce: $\alpha \overset{P}{\vee} \beta = \alpha + \beta - \alpha \cdot \beta$.

Lukasiewiczova disjunkce: $\alpha \overset{L}{\vee} \beta = \begin{cases} \alpha + \beta & pro\ \alpha + \beta < 1 \\ 0 & jinak \end{cases}$.

Drastická disjunkce: $\alpha \overset{D}{\vee} \beta = \begin{cases} \alpha & pro\ \beta = 0 \\ \beta & pro\ \alpha = 0 \\ 0 & jinak \end{cases}$.



Obrázek 21 - Fuzzy disjunkce: standardní, součinová, Lukasiewiczova, drastická

Ostatní typy fuzzy disjunkcí bude možné doplnit prostřednictvím bodů rozšíření viz níže. Postačující podmínkou pro provedení operace budou vstupní množiny prvků a jejich hodnoty stupňů příslušnosti.

Příklad:

Vstup: Fuzzy množina A = {0,4/o1; 0,6/o2; 1/o3; 0/o4}.

Fuzzy množina B = {0,3/o1; 1/o2; 0,1/o3; 0,7/o4; 1/o5}.

Výstup: Fuzzy množina $A \overset{S}{\vee} B$ = {0,4/o1; 1/o2; 1/o3; 0,7/o4; 1/o5}.

Použité symboly:

o_j, kde j = 1,…n prvky univerza,

n/o_j, kde n je stupeň příslušnosti prvku o_j do fuzzy množiny.

5.2.2. Fuzzy operátory

Typy fuzzy operátorů (třídy fuzzy operací) je potřeba volit v závislosti na aktuálně řešené úloze.

5.2.2.1. Kontextová definice použitých fuzzy operátorů

Volba nejvhodnějšího typu fuzzy operátorů, by měla co nejlépe respektovat kontext řešené úlohy. Tam, kde je to účelné by tedy měla existovat možnost, jak např. standardní fuzzy konjunkci nahradit součinovou.
Rozhodnutí o použitém typu fuzzy operátorů je účelné ponechat na straně klientské aplikace. Zde jsou totiž k dispozici informace potřebné k výběru nejvhodnějšího typu na základě požadavků konečného uživatele.
Pro pokrytí obvyklých situacích bude vhodné v rámci řešení nabídnout základní sady fuzzy operátorů a případné další kombinace realizovat transparentně formou bodu rozšíření viz dále.

5.2.2.2. Bod rozšíření pro definici fuzzy operátoru

Za účelem možnosti definovat uživatelské typy fuzzy operátorů je vhodné zavést jednotné rozhraní, které deleguje vlastní implementaci výpočtu hodnot příslušnosti na volně spřažené moduly. Jednotné rozhraní poslouží jako šablona pro předání vstupních hodnot funkcí příslušnosti a převezme výslednou hodnotu.
Výhodou bodů rozšíření je otevřenost při doplňování specifických požadavků na chování libovolné fuzzy operace.

5.2.3. Funkce příslušnosti

Stupeň příslušnosti prvek fuzzy získá na základě vyhodnocení kontextově závislé funkce. Řezy fuzzy množiny umožní určit minimální stupeň příslušnosti prvků, které nás zajímají. Na prvky fuzzy množin (objekty v databázi) by neměly být kladeny žádné omezující podmínky z pohledu jejich protokolu.

5.2.3.1. Výpočet stupňů příslušnosti do fuzzy množiny

Výpočet funkce příslušnosti nad vstupní množinou prvků přiřadí každému prvku jeho stupeň příslušnosti do výsledné fuzzy množiny. Do fuzzy množiny by měly být zařazeny i prvky s nulovým stupněm příslušnosti. V opačném případě by mohlo dojít ke ztrátě informace, zda prvek byl či nebyl funkcí příslušnosti vůbec ohodnocen, ale především proto, že výsledná fuzzy množina se může stát vstupem pro další operace

jako je fuzzy doplněk (absence prvků s nulovou příslušností ve zdrojové množině by znamenal zcela nesprávný výsledek doplňku).

5.2.3.2. Stanovení minimální hodnoty stupně příslušnosti

V řadě situacích je potřeba provést řez fuzzy množinou na úrovni minimální příslušnosti prvků k ní, tj. tzv. α-řez. Prvky s nižší než stanovenou příslušností nemusí být pro další zpracování relevantní, nebo mohou být nepodstatné z pohledu počtu potřebných prvků ve výsledku (zajímají nás např. pouze nejlepší tři výsledky, které mají stupeň příslušnosti nad hodnotou 0,9). Řešení by tedy mělo poskytnout prostředky pro omezení fuzzy množin pouze na takové prvky, jejichž stupeň příslušnosti je roven minimální hodnotě, nebo je vyšší.

Příklad:

Vstup: A = {0,4/o1; 0,6/o2; 1/o3, 0,5/o4; 0/o5}.
Minimální stupeň příslušnosti 0,5 (α-řez na hladině 0,5).

Výstup: A = {0,6/o2; 1/o3, 0,5/o4}.

Použité symboly:

o_j, kde j = 1,…n prvky univerza,

n/o_j, kde n je stupeň příslušnosti prvku o_j do fuzzy množiny.

5.2.3.3. Kontextová definice funkce příslušnosti

Průběh funkce příslušnosti, tj. přiřazení stupně příslušnosti prvků do fuzzy množiny, je značně závislé na aktuálním kontextu dotazu. Navržené řešení by mělo tento základní fakt respektovat (v opačném případě by se mohlo stát jen omezeně použitelné). Principiálně je možné algoritmy pro výpočet hodnot funkcí příslušnosti umístit buď přímo do databáze, nebo do nějaké vrstvy nad ní. Navržené řešení by mělo v ideálním případě podporovat obě varianty tak, aby umožnilo jednoduše sdílet často používané algoritmy, ale zároveň ponechalo možnost specifikovat tyto algoritmy v klientské aplikaci v závislosti na aktuálním kontextu úlohy.

Za tímto účelem je vhodné vymezit obecné a jednotné rozhraní, které přijme odkaz na množinu vstupních objektů, pro které provede příslušný algoritmus a následně vrátí výsledek.

5.2.3.4. Typově nezávislý přístup k objektům v OODBMS

Z pohledu využitelnosti stávajících dat bez nutnosti jejich dodatečné konverze je nezbytné, aby byl přístup k objektům nezávislý na jejich struktuře, resp. nebyla vyžadováno žádné konkrétní rozhraní objektů. Zcela postačující by měla být znalost jejich protokolu a v případě, že to bude OODBMS podporovat, tak i jejich vnitřní struktury (některé OODBMS umožňují sestavit databázový dotaz, který pracuje s vnitřními proměnnými objektů, tj. vědomě porušují princip zapouzdření ve prospěch dotazovacího jazyka).

Rozhraní pro specifikaci funkce příslušnosti, výpočet hodnot příslušnosti a operace nad fuzzy množinami budou tedy realizovány transparentně a nebudou vyžadovat specifický způsob ukládání vlastních databázových (doménových) objektů.

5.3. Mimofunkční pokrytí

Řešení by mělo vyhovět i některým požadavkům, které nemají funkční povahu. Základní očekávání z pohledu kvality software dle charakteristik ISO standardu jakosti již byla zmíněna. Další se týkají technického hlediska a zejména pak možných platforem, výkonnosti a napojení na klientské systémy, resp. umístění do vrstev.

5.3.1. Typ databázových systémů

Řešení bude směřovat do oblasti objektově orientovaných databázových systémů (třídně-instančních), jako je např. Versant, Gemstone/S, ObjectStore či O2. Obecný návrhový vzor by měl být použitelný pro všechny shodně, pouze s rozdíly v implementačních detailech, kdy bude možné využít specifik jednotlivých databázových platforem.

5.3.2. Výkonnost

Systémy implementující navržený vzor by měly být schopny poskytovat dostatečný výkon a zároveň minimalizovat paměťové nároky při operacích s fuzzy množinami. Pokud to cílová databázová platforma bude umožňovat, bude vyhodnocení funkcí

příslušnosti realizováno uvnitř databázového stroje (např. Gemstone/S). Klientský systém bude v takovém případě pouze definovat průběh funkcí a vymezí rozsah dat, na která mají být aplikovány.

Jde o kritický parametr realizace podpory fuzzy přístupu v OODBMS. Přestože bude navržené řešení koncipováno obecně, konkrétní vlastnosti databázového systému nakonec rozhodnou, zda a do jaké míry bude možné využít hromadného „fuzzy zpracování dat".

5.3.3. Cílové systémy

Cílovými systémy jsou myšleny programovací jazyky, aplikační rámce a architektury, ve kterých bude možné realizovat API k fuzzy integračním rámcům založených na navrženém řešení. Z pohledu zastoupení jednotlivých technologií v dnešním vývoji podnikových aplikací se předpokládá, že řešení bude využitelné především na platformách Java, .NET a Smalltalk. Další objektově orientované programovací jazyky by měly být použitelné bez podstatných změn v obecném návrhu.

5.3.4. Vrstvy informačního systému

Vytvořením specifického aplikačního rámce pro podporu fuzzy přístupu v objektově orientovaných databázích se již zabývala řada autorů (např. [Ma 2005]). Jednotlivé přístupy se liší zejména cílovou platformou, mírou potřebné modifikace stávajících dat a umístěním výkonné části (výpočet hodnoty funkce příslušnosti, vyhodnocování logických podmínek) v softwarové architektuře. Poněkud stranou zájmu však dosud zůstával jakýsi „prováděcí předpis", který by pomohl rozhodnout řadu z otázek uvedených v předchozích částech této práce. K tomuto účelu lze využít formátu návrhových vzorů. Obecně použitelný vzor totiž může definovat vstupní a výstupní body pro implementaci aplikačního rámce a zároveň ponechává možnost využít v jednotlivých implementacích výhod konkrétních databázových systémů či programovacích jazyků. Poté je možné ponechat až na konkrétní situaci, zda bude výkonná část řešení umístěna do:

- aplikační vrstvy (vyšší flexibilita, snadnost použití),
- mezivrstvy mezi aplikační vrstvou a databází (oddělení od aplikační logiky a použitelné pro více aplikací), nebo do

- vrstvy databázového stroje (odstínění od implementace a vysoká úroveň znuvupoužitelnosti).

6. Návrhový vzor Fuzzy rozšíření OODBMS

Návrhový vzor jak zahrnout podporu fuzzy přístupu do práce s OODBMS je výsledkem autorova výzkumu v oblasti principů a tvorby objektově orientovaných databázových systémů a poznatků z teorie fuzzy množin a fuzzy logiky. Vzor je uveden v klasické GOF struktuře [Gamma et al. 2003]. Na obecný návrhový vzor dále navážou konkrétní možnosti využití vzoru v jednotlivých typech OODBMS.

6.1. Účel

Zavádí do rozhraní s OODBMS pojem fuzzy množina vymezením univerza (množiny vstupních prvků) a funkce příslušnosti definované na univerzu. Poskytuje rozhraní pro základní operace s fuzzy množinami a ponechává volnost při implementaci jejich algoritmů. Podporuje funkce příslušnosti závislé na kontextu řešené úlohy.

6.2. Motivace

Vývoj regulačních či podnikových informačních systémů využívajících k rozhodování prvků fuzzy logiku je velmi často spojeno se zpracováváním velkých objemů dat uložených v databázích. Svojí povahou se obvykle jedná o data značně strukturovaná a s mnoha vazbami, která mají původ např. v komplexních technologických či ekonomických procesech. Pokud jsou data uchovávána a spravována systémy typu OODBMS, je potřeba najít vhodný objektově orientovaný postup pro jejich rozdělení do fuzzy množin, které dále poslouží jako vstup do rozhodovacích podmínek typu JESTLIŽE-POTOM.

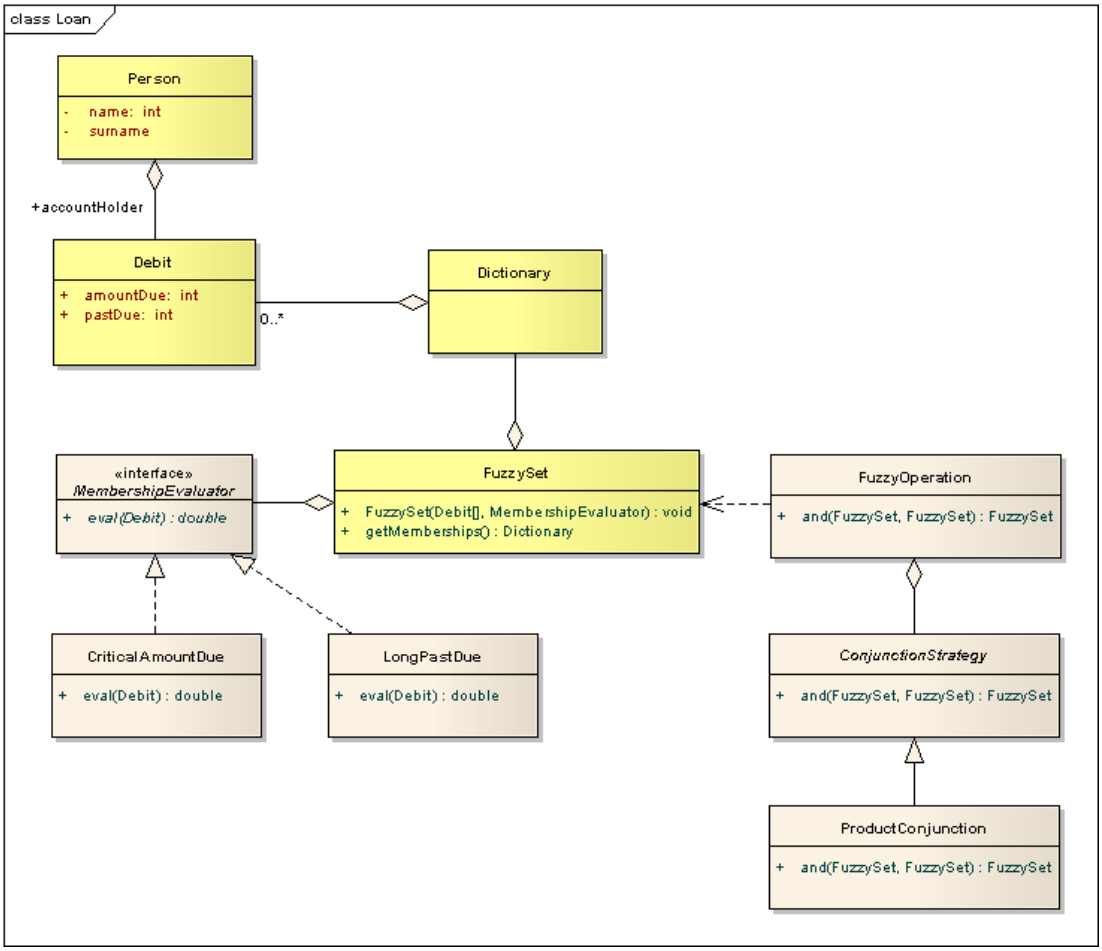
Čím složitější je modelovaný proces, tím větší jsou nároky na konstrukci fuzzy množin a práci s nimi pomocí fuzzy operátorů a dalších typů operací. Vzhledem k podstatně nižší úrovni standardizace OODBMS, než je tomu u RDBMS je rovněž často nezbytné se potýkat s odlišným typem implementace pro každý jednotlivý databázový produkt. Je potřeba se soustředit na řadu zásadních hledisek, mezi které patří:

- Kde a jak má být reprezentována vlastní fuzzy množina. Předpokládá se hromadné zpracovávání velkých objemů dat a tomu musí být přizpůsobena logika ukládání fuzzy množin i další operace s nimi.

- Dle kontextu řešené úlohy je potřeba implementovat algoritmy pro výpočet výsledků operací s fuzzy množinami (doplněk, průnik, sjednocení) v různých variantách. Klíčové je rozhodnutí, jak podporovat nejčastěji užívané typy fuzzy logik (standardní, součinová apod.), ale ponechat si možnost v budoucnu doplnit další typy.
- Příslušnost objektu do fuzzy množin je třeba rozlišovat na úrovni připojení konkrétního klienta resp. až na úrovni jedné databázové transakce. Fuzzy množiny dlouhodobější povahy je naopak vhodné sdílet mezi dílčími transakcemi či dokonce globálně mezi klienty.
- Stupně příslušnosti objektů do fuzzy množin je účelné modelovat jako externí stav objektů, tj. neukládat je fyzicky do stejné datové struktury jako samotný objekt. Prvky fuzzy množin se tak mohou stát libovolné objekty uložené v databázi bez nutnosti zásahu do jejich interních struktur.
- Změny v uložených datech, funkcích příslušnosti a algoritmech operací by neměly být příliš náročné, protože u komplexních datových struktur lze jejich dynamický vývoj předpokládat.
- Fuzzy rozšíření v OODBMS může na různých platformách využívat specifické vlastnosti databázového systému, ale se zachováním struktury tak, aby zůstalo přenositelné a znovupoužitelné.

Jako jeden z mnoha příkladů, které se stali motivací pro návrh obecného řešení je řešení následující situace:

Banky si v interní databázi udržují informace o svých dlužnících včetně doby prodlení se splátkou a výše dlužné částky. Klientů, kteří mají již po termínu splatnosti mohou být tisíce či dokonce desetitisíce. S ohledem na závažnost a finanční rizika je potřeba rozhodnout u každého z nich o dalším postupu ze strany banky.



Obrázek 22 - Zhodnocení závažnosti dluhu

Třída `Debit` reprezentuje dluh u banky. Pro rozhodnutí o dalším postupu je vhodné rozlišit jednotlivé dluhy dle závažnosti a rozhodnout, které spadají např. do kategorie „*vysoká dlužná částka s dlouhou dobou po splatnosti*“. Jde o kombinaci dvou faktorů, které je potřeba vyhodnotit nejprve každý zvlášť a poté výsledky propojit. V klientském systému (např. Scoring systém) je stanovena funkce příslušnosti dluhu do fuzzy množiny „*vysoká dlužná částka*“ pomocí třídy `CriticalAmountDue` (implementace rozhraní `MembershipEvaluator` dovoluje měnit algoritmus dle aktuálních směrnic banky) a do fuzzy množiny „*dlouhá doba po splatnosti*“ pomocí třídy `LongPastDue`. Získané stupně příslušnosti každého dluhu jsou pomocí třídy `FuzzySet` uloženy v databázi.

Fuzzy množiny pro oba faktory dluhu je vhodné zkombinovat tak, aby nízké stupně příslušnosti alespoň do jedné z množin částečně eliminovaly celkovou závažnost dluhu.

Při použití standardní fuzzy konjunkce by oba faktory byly považovány za rovnocenné, naopak při použití Lukasiewiczovy fuzzy konjunkce by kombinace faktorů se součtem menším než jedna nebyly vůbec brány v potaz. Naproti tomu součinová fuzzy konjunkce bere za rozhodující úroveň méně kritického z obou faktorů. Pro ilustraci lze doplnit také chování drastické fuzzy konjunkce, která by pro banku znamenala, že celková závažnost dluhu je pro ni zcela nepodstatná, pokud alespoň jeden z faktorů není na kritické úrovni.

S ohledem na respektování požadavku na eliminaci celkové závažnosti se jako nejvhodnější v tomto případě jeví použití součinové fuzzy konjunkce reprezentované třídou `ProductConjunction` (implementace rozhraní `ConjunctionStrategy` slouží pro snadné rozšíření o další typy fuzzy konjunkcí). Třída `FuzzyOperation` představuje jednotné rozhraní pro operace s fuzzy množinou, pro jejichž vlastní implementace používá vzor Strategie [Gamma et al. 2003].

Výsledná fuzzy množina obsahuje jednotlivé dluhy a stupeň jejich závažnosti s ohledem na stanovená kritéria. Další zpracování těchto údajů pomůže bance rozhodnout, pro které dluhy má ještě volit sankční poplatky, a pro které již vymáhat dlužnou částku např. soudní cestou.

Příklady podobné výše uvedenému podporují snahu návrhového vzoru umožnit co nejvěrnějším způsobem zachytit situace vyžadující odlišné přístupy při vyhodnocování kombinací více fuzzy podmínek. Jen velmi omezené množství současných databázově orientovaných systémů s podporou fuzzy přístupu tyto potřeby respektuje. Podchycení těchto nuancí na obecné úrovni může dát prostor pro využití fuzzy logiky i v situacích, kde standardní fuzzy logické spojky nenabízí uspokojivé výsledky.

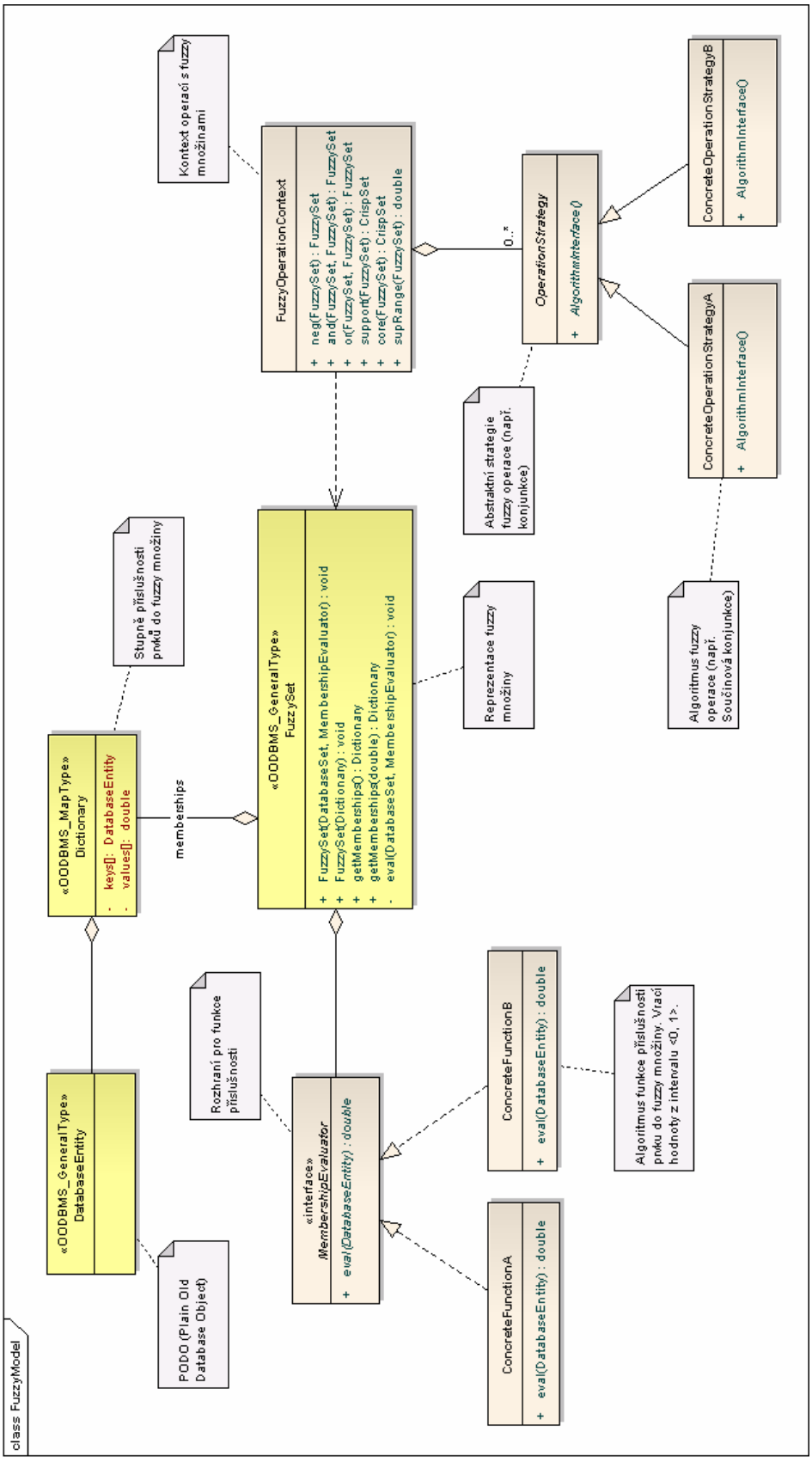
6.3. Použití

Vzor Fuzzy rozšíření v OODBMS je vhodné použít v těchto situacích:

- Máme velké množství objektů uložených v OODBMS a chceme zjistit jejich příslušnost do fuzzy množin na základě stanovených kritérií. Samotné objekty chceme nechat na fuzzy rozšíření nezávislé (objekty nejsou modifikovány, pouze se stávají prvky fuzzy množin).

- Je výhodné, aby nezávisle na umístění jednotlivých komponent (v databázovém systému nebo v klientském systému) bylo možné použít pro reprezentaci fuzzy množin a práce s nimi shodný datový model.
- Výpočet hodnot funkce příslušnosti musí respektovat kontext úlohy. Při souběžném zpracovávání by každá z úloh měla mít možnost definovat vlastní funkci příslušnosti objektů do fuzzy množin.
- Existuje potřeba průběžného doplňování algoritmů jednotlivých fuzzy operací. O výběru nejvhodnějšího typu fuzzy operace musí mít možnost rozhodnout klientský systém.
- Vytvářený informační systém s podporou fuzzy přístupu bude provozován na různých OODBMS platformách, ale měl by přenositelný bez podstatných zásahů do struktury datového modelu.

6.4. Struktura



Obrázek 23 - Model „Fuzzy rozšíření“ pro objektově orientovaný databázový systém

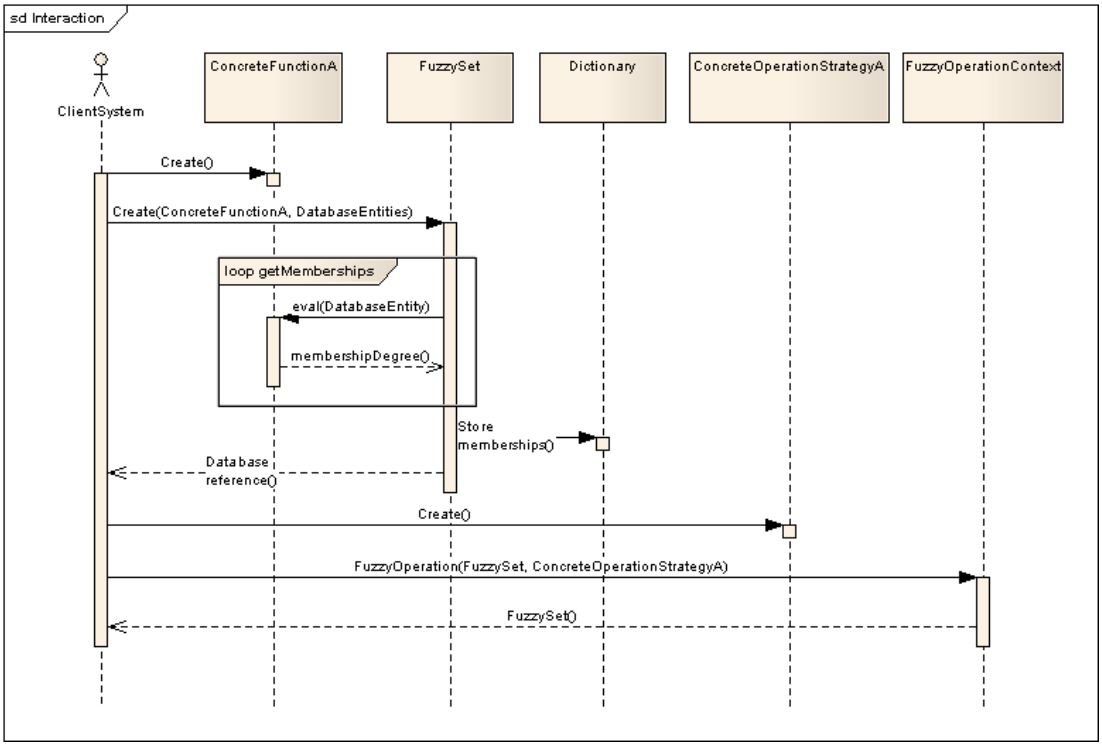
6.5. Součásti

- **`FuzzySet`**
  - Reprezentuje fuzzy množinu pro klientský systém. Objekty typu `FuzzySet` jsou ukládány v OODBMS.
  - Je konfigurovatelná pomocí objektu `ConcreteFunction` pro přiřazení stupně příslušnosti svých prvků.
- **`Dictionary (OODBMS_MapType)`**
  - Definuje interní databázovou strukturu pro prvky fuzzy množiny a vypočtené stupně příslušnosti. Obvykle bývá implementována jako databázová mapa (datové typy odvozené od *ODMG3 Map* návrhu).
- **`DatabaseEntity (OODBMS_GeneralType)`**
  - Objekt, který je uložen v databázi a může se stát prvkem fuzzy množiny.
- **`MembershipEvaluator`**
  - Deklaruje jednoduché rozhraní pro algoritmy přiřazující jednotlivým databázovým objektům typu `DatabaseEntity` stupeň příslušnosti do fuzzy množin.
- **`ConcreteFunction`**
  - Implementuje algoritmus pro stanovení stupně příslušnosti prvku do fuzzy množiny prostřednictvím rozhraní `MembershipEvaluator`.
- **`FuzzyOperationContext`**
  - Představuje logiku operací s fuzzy množinami v závislosti na kontextu použití. U operací, které mohou mít více odlišných implementací (např. typy konjunkcí) udržuje odkaz na objekty typu `OperationStrategy`, kterým přenechává implementaci.
- **`OperationStrategy`**
  - Deklaruje rozhraní (jako abstraktní třídu) pro implementaci algoritmu operace. Zajišťuje nezávislost fuzzy operací na jejich konkrétních variantách.
- **`ConcreteOperationStrategy`**
  - Implementuje algoritmus fuzzy operace pomocí rozhraní `OperationStrategy`. Viz vzor Strategie [Gamma et al. 2003].

6.6. Spolupráce

- Klientský systém vytvoří objekt `ConcreteFunctionA` a předá ho spolu s odkazem na objekty `DatabaseEntity` do konstruktoru třídy `FuzzySet`.
- Objekt `FuzzySet` postupně pro každý objekt `DatabaseEntity` vyhodnotí jeho stupeň příslušnosti a výsledky uloží do databázové mapy `Dictionary`.
- Klientský systém vytvoří objekt `ConcreteOperationStrategyA` (např. pro fuzzy negaci) a předá ho spolu s referencí na `FuzzySet` jako parametry zvolené operace (negace) objektu `FuzzyOperationContext`.
- Objekt `FuzzyOperationContext` dle zvolené strategie provede odpovídající algoritmus a jako výsledek operace vrátí např. `FuzzySet`.

Následující diagram znázorňuje spolupráci mezi klientem, fuzzy množinou a kontextem fuzzy operací:



Obrázek 24 - Spolupráce objektů

Ukázková sekvence předpokládá vytváření objektů typu `ConcreteFunctionA` a `ConcreteOperationStrategyA` na straně klientského systému dle aktuálního kontextu úlohy. Všechny tyto objekty však mohou být ukládány rovněž do OODBMS a při řešení úlohy se již použijí pouze příslušné reference.

6.7. Důsledky

Vzor Fuzzy rozšíření OODBMS má tyto výhody a nevýhody:

1. *Fuzzy množiny v databázi.* Ukládání fuzzy množin přímo do databáze neomezuje jejich použití pouze na počet prvků, se kterými je schopen v paměti pracovat klientský systém. Zároveň jsou však množiny vytvářeny dynamicky a s parametry, které odpovídají kontextu úlohy. Reprezentace fuzzy množiny nepoužívá pro každý prvek tzv. *Wrapper* (využití vzoru Dekorátor případně Adaptér viz [Gamma et al. 2003]), ale namísto toho uchovává stupně příslušnosti všech svých prvků v jediné datové struktuře typu mapa. Tím se tento vzor odlišuje např. od techniky „Head-Body Split" popsané pro OODBMS ve [Visnick 2003].
2. *Nezávislost na rozhraní k databázi.* K vytvoření fuzzy množiny z prvků uložených v databázi není nutný jeden konkrétní typ databázového rozhraní. Může být použito některé ze standardizovaných rozhraní typu JPA, ADO.NET nebo nativní rozhraní, které může těžit ze specifických předností konkrétní databázové platformy. V případě, že to OODBMS podporuje, je možné vzor využít i jako součást aplikační logiky umístěné v samotném databázovém systému.
3. *Kontextově orientované funkce příslušnosti.* Jednoduché rozhraní pro algoritmy určující stupeň příslušnosti prvku do fuzzy množiny ponechává klientskému systému možnost definovat průběh funkce příslušnosti dle kontextu řešené úlohy. Paralelní sestavování dvou fuzzy množin nad stejnými daty se tak může lišit v závislosti na zvolených funkcích příslušnosti.
4. *Volba implementací.* Propojením vzoru Fuzzy rozšíření OODBMS a základního vzoru Strategie [Gamma et al. 2003] je možné implementaci operací s fuzzy množinami průběžně doplňovat a rozšiřovat. Klientský systém může vybírat mezi strategiemi podle preferencí tj. poměru prostoru a času, resp. podle vhodnosti jednotlivých typů operace pro danou úlohu. Vazba mezi operací a algoritmem může zůstat dynamická a klientský systém může implementaci algoritmu v případě potřeby změnit.
5. *Není vyžadována modifikace primárních dat.* Fuzzy rozšíření může využít libovolné objekty uložené v databázi (`DatabaseEntity`), tj. nevyžadují od nich implementaci žádných specifických rozšíření ani modifikaci jejich

datových struktur (změna třídy). Tyto objekty lze označit akronymem PODO (Plain Old Database Object), pro jehož zavedení se autor tohoto vzoru nechal inspirovat akronymem POJO od trojice M. Fowler, R. Parsons a J. MacKenzie pro jednoduše navržené Java objekty a historicky také akronymem PODS (Plain Old Data Structures) používaným pro datové struktury bez zapouzdření a dalších objektově orientovaných vlastností [ISO/IEC 14882]. Postačující podmínkou pro databázové objekty, které se mohou stát prvky fuzzy množin, je jejich vnější rozhraní pro určení stupně příslušnosti.

6. *Ukládání mezivýsledků.* Z pohledu výkonu, resp. časové a prostorové složitosti operací s fuzzy množinami, je užitečné některé mezivýsledky uchovávat, což je možné díky umístění třídy `FuzzySet` v databázi. Při budování *Cache* mechanismů, které mají životnost přesahující jednu transakci je však nutné v návrhu zohlednit postupy aktualizace stupňů příslušnosti u stávajících databázových entit, ale i u nových resp. odebraných.
7. *Dodržení objektového přístupu.* Rozhraní pro fuzzy rozšíření striktně dodržuje čistě objektový přístup a nevyžaduje při své implementaci v programovacích jazycích používání specifických konstrukcí v textové formě (jako např. fuzzy přístup uvedený v [Sicilia 2002]). Objektová struktura i logika operací je navržena tak, aby podporovala dynamickou práci s programovým kódem typu refactoring a další agilní metody.
8. *Přenositelnost na další OODBMS.* Databázově závislé třídy `DatabaseEntity`, `Dictionary` a `FuzzySet` používají pouze základní datové struktury, a přestože se budou konkrétní datové typy v databázích různých výrobců lišit, lze ve většině případů zaručit jejich vzájemnou nahraditelnost. Objekty `DatabaseEntity` jako PODO mohou být instance libovolné třídy, objekty `Dictionary` jsou založeny na libovolné implementaci datové mapy a objekt `FuzzySet` musí být schopen přijmout realizátora rozhraní `MembershipEvaluator`. Pokud OODBMS nepodporuje typ rozhraní (interface), nebo je to výhodnější, lze rozhraní transparentně nahradit abstraktní třídou.
9. *Potlačení specifických vlastností OODBMS.* Vzor má potencionální nevýhodu v tom, že klient předává odkaz na výsledek databázového dotazu

do třídy `FuzzySet`, která se o jeho zpracování již postará sama. Způsob zpracování nemusí vyhovovat všem situacím, např. z důvodu optimalizace výkonu (např. počet načítaných objektů do vyrovnávací paměti, práce s proxy objekty). Řešením by mohlo být přenechání výběru algoritmu na objektech realizující vzor Strategie podobně, jako je tomu u výběru variant fuzzy operací. Klient by pak spolu s výsledkem dotazu do fuzzy množiny předával i strategii načítání dat. Tato nevýhoda je eliminována v OODBMS, které podporují vyhodnocení funkce příslušnosti uvnitř databáze.

10. *Rozšíření na úrovni rozhraní.* Uvedená rozšíření jsou navržena na úrovni datových struktur a logiky umístěné na rozhraní práce s OODBMS. Nemění funkčnost jednotlivých databázových systémů. Nevýhodou je v tomto případě nižší účinnost, než které by bylo pravděpodobně možné dosáhnout změnou principů a architektur OODBMS. Toto omezení je v návrhu částečně kompenzováno jednotným přístupem, který snad usnadní pochopení, rozšiřování a migraci aplikací, které jej použijí.
11. *Zvýšený počet objektů.* Kombinace se vzorem Strategie s sebou přináší zvýšený počet objektů v aplikaci. Pro omezení negativního dopadu lze doporučit postup uvedený v [Gamma et al. 2003]. V případě, že si implementace fuzzy operátorů vystačí vždy pouze s jedním algoritmem, je možné za cenu omezené rozšiřitelnosti sloučit třídy implementující strategie, tj. `ConcreteOperationStrategy` a `FuzzyOperationContext`.

6.8. Implementace

V úvahu je vhodné vzít následující implementační detaily:

1. *Klient/server architektura.* Použitelnost vzoru pro velké objemy dat je podmíněna implementací třídy reprezentující fuzzy množinu jako perzistentní, tj. uloženou v databázi. Perzistence ostatních částí vzoru zůstává volitelná, a je omezena jen vlastnostmi konkrétní OODBMS platformy. V databázových systémech jako je Gemstone/S lze v extrémním případě všechny části vzoru implementovat jako perzistentní a používat pouze vstupně/výstupní zprávy zasílané databázovým objektům (dáno architekturou Gemstone/S, která v sobě spojuje vlastnosti databázového a aplikačního serveru). Ve většině situací bude však výhodnější ponechat alespoň algoritmy

pro přiřazování stupňů příslušnosti na straně klientského systému. Dosáhne se tak větší volnosti při dynamické konstrukci funkcí příslušnosti, resp. jejich parametrů (např. výběrová a preferenční kritéria z webových stránek). Dalším důvodem může být často potřeba spolupráce s externími systémy, na základě jejichž informací lze teprve stupeň příslušnosti prvku do fuzzy množiny určit. Databázový server není obecně schopen tuto činnost efektivně zaručit. Pokud jsou data velmi malého rozsahu a/nebo nejsou uložena v OODBMS, není vyloučena ani implementace vzoru jako zcela neperzistentního a využití logiky na straně klienta (např. v prohlížeči webových stránek, mobilních zařízeních).

2. *Konzistence výsledků mimo databázovou transakci.* Platnost výsledků (vstupní data, vyhodnocení příslušnosti do fuzzy množin, operace s fuzzy množinami) lze v databázových aplikacích obecně zaručit pouze na úrovni databázových transakcí. Pokud má být platnost fuzzy množin delší než databázová transakce (např. při použití jako *cache*), pak je implementaci vzoru potřeba doplnit o mechanismus označování neplatných dat, resp. nástroje pro zajištění jejich konzistence. Jednou z možností je delegování databázového dotazu na třídu `FuzzySet`, která si může uložit parametry pro jeho pozdější zopakování (např. pomocí vzoru Memento). Nevýhodou je zatížení třídy logikou, která se netýká fuzzy množin. Lepším řešením může být přenechání logiky spojené s vykonáním databázového dotazu na zvláštní třídu, která vystupuje vůči `FuzzySet` jako proxy. Logika automatické aktualizace stupňů příslušnosti je zajištěna na úrovni opětovného vyhodnocení fuzzy množiny při změně vstupních dat.
3. *Definice rozhraní* `FuzzySet` *a* `ConcreteFunction`. Pro předání databázového objektu z fuzzy množiny k vyhodnocení jeho stupně příslušnosti poslouží nejčastěji rozhraní `MembershipEvaluator`. Často může nastat situace, kdy je potřeba s každým objektem před vyhodnocením provést nějakou akci a v takovém případě je výhodnější implementovat `MembershipEvaluator` jako abstraktní třídu s šablonovou metodou. Programovací jazyky s polymorfismem založeným na dědičnosti musí obvykle implementovat metody rozhraní/abstraktní třídy s obecným parametrem typu `Object` a následně provádět typově nebezpečnou konverzi.

Vyhnout se tomu je možné jazykově specifickými konstrukcemi jako jsou šablonové metody v C++ nebo generické typy v jazyce Java. Naopak jazyky s pozdní vazbou a nezávislým polymorfismem jako Smalltalk mohou definici rozhraní `MembershipEvaluator` v určitých případech zcela vypustit a implementovat `ConcreteFunction` jako běžnou třídu nebo dokonce jako anonymní blok.

4. *Funkce příslušnosti.* Implementace algoritmu funkce příslušnosti je stěžejním faktorem, který zásadně ovlivní použitelnost získaných výsledků. Při výběru algoritmu proti sobě stojí snaha o zachycení co nejvěrnějšího průběhu funkce příslušnosti a obtížnost implementace (případně výpočetní složitost). Je třeba však vzít v úvahu základní princip inkompatibility, podle kterého není snaha o co největší přesnost vždy vhodná. Pro modelování funkce příslušnosti se proto často používají některé jednodušší aproximované funkce jako lichoběžník, trojúhelník nebo L funkce. Implementaci konkrétních funkcí je proto vhodné realizovat jako parametrické varianty základních funkcí. Tím lze dosáhnout mimo jiné i vyšší znuvupoužitelnosti.
5. *Rozhraní k DatabaseEntity.* Aplikace vzoru vyžaduje přístup k databázové entitě minimálně ve dvou krocích. Prvním krokem je výběr databázových objektů, které vstupují do fuzzy množiny. Jelikož se obecně jedná o objekty typu PODO (žádné explicitní požadavky na strukturu a podobu protokolu), lze použít libovolný databázový dotaz, který vybere kolekci (množinu) objektů. Obvykle jde o dotaz na pojmenovaný kořen struktury, extenzi třídy nebo výběr na základě logických podmínek. Pro konstrukci logických podmínek existuje více přístupů (explicitní predikáty, šablony virtuálních atributů, proprietární řetězcové konstrukce a další). Při implementaci je výhodné maximum operací přenechat na databázovém systému. Některé OODBMS dokáží vyhodnotit interně pouze podmínky na instanční proměnné a ostatní řeší načtením dat do klientského systému (např. Versant). Pokud to však databázová platforma dovoluje, je výhodné nechat interně vyhodnotit i podmínky na návratové hodnoty metod (např. Gemstone/S). Příklad interně vyhodnocených podmínek na instanční proměnnou a výsledek metody může vypadat následovně (oba dotazy jsou spuštěny uvnitř databázového systému):

```
// Optimalizovaný dotaz s podmínkou na instanční proměnnou,
// který je vyhodnocen interně v Gemstone/S.
people select: {:person | person.nationality = 'Czech Republic'}.

// Dotaz s podmínkou na návratovou hodnotu metody pro výpočet věku.
// Tato podmínka je rovněž vyhodnocena interně v Gemstone/S.
people select: [:person | person age > 20].
```

6. *Strategie fuzzy operací.* Rozhraní mezi kontextem fuzzy operací a jejich algoritmy je realizováno pomocí vzoru Strategie, a platí tedy pro něj implementační poznámky z publikace [Gamma et al. 2003]. Doplnit lze oblast definice rozhraní strategie a kontextu. Množství předávaných dat je minimální a proto je možné řešit rozhraní prostým předáváním ve formě parametrů. Implementace rozhraní `OperationStrategy` je do značné míry závislé na programovacím jazyku. Je možné ho realizovat jako rozhraní, abstraktní třídu (vhodné, pokud mají algoritmy společné fragmenty), nebo jej zcela vynechat pomocí parametrů šablon v C++ nebo Smalltalk bloků. V jazycích, kde není polymorfismus svázán s dědičností je možné realizovat např. takto:

```
// Metoda pro výpočet doplňku fuzzy množiny ve třídě
// FuzzyOperationContext
negation: fuzzySet strategy: aStrategy
 | mem negBlock |
 mem := Dictionary new.   // Členství prvků v doplňku
 negBlock := self strategies at: aStrategy.
 fuzzySet memberships
     keysAndValuesDo: [:m :v | mem at: m put: (negBlock value: v)].
 ^FuzzySet newWithMemberships: mem

// Blok pro výpočet standardní negace, který je v instanci třídy
// FuzzyOperationContext uložen v katalogu strategií pod symbolem
// #StandardNegation
standardNegation
 ^[:x | 1 - x]

// Získání doplňku fuzzy množiny pomocí strategie standardní negace
operation := FuzzyOperationContext new.
operation negation: fuzzySet strategy: #StandardNegation
```

7. *Výchozí strategie.* Třídu `FuzzyOperationContext` je možné zjednodušit, pokud aplikace ve většině případů používají stále stejné strategie a jen výjimečně se mění. V takovém případě je vhodné definovat výchozí strategie, které se uplatní, pokud nebude specifikováno jinak. Umožní to zjednodušit rozhraní kontextu a zpřehlednit konstrukci fuzzy podmínek.

6.9. Příklad

Jako příklad bude uveden zdrojový text pro modelovou situaci z oddílu *Motivace*. Implementace je v jazyce Java. Jako OODBMS je použit systém Versant s rozhraním JVI [Versant].

Rozhraní `MembershipEvaluator` deklaruje metodu `eval`, která přijímá objekty `Debit` (dluh) a vypočítá pro ně stupeň příslušnosti do fuzzy množiny jako číslo v intervalu [0, 1]. Konkrétní algoritmus pro výpočet je realizován ve třídě, která rozhraní implementuje. Versant rozhraní JVI vrací výsledky databázového dotazu z pohledu jazyka Java netypově, a proto má vstupní parametr metody obecný typ `Object`.

```
public interface MembershipEvaluator {
      public double eval(Object o);    // Rozhraní pro výpočet stupně
}                                      // příslušnosti
```

Při konstrukci fuzzy množiny je nutné předat vedle vstupních prvků také algoritmus, podle kterého se spočítají konkrétní stupně příslušnosti. K posouzení, do jaké míry je výši dluhu možno považovat za kritickou slouží třída `CriticalAmountDue`. Metoda `eval` implementovaná z rozhraní `MembershipEvaluator` nejprve provede přetypování objektu na `Debit` a následně spočítá vlastní hodnotu stupně příslušnosti. V tomto případě jde o Gamma funkci s parametry pro výši dluhu, kdy přestává být stupeň příslušnosti nulový (CRITICAL_AMOUNT) a parametr (K_PARAM), který ovlivňuje rychlost růstu přiblížení stupně příslušnosti k hodnotě jedna.

```
public class CriticalAmountDue implements MembershipEvaluator {
      private static final int CRITICAL_AMOUNT = 2000000;  // Nenulový od:
      private static final double K_PARAM = 0.6;           // Růst funkce

      public double eval(Object o) {
            Debit d = (Debit) o;
            if(d.getAmountDue() > CRITICAL_AMOUNT) {
                  double numerator = K_PARAM *
                                    Math.pow((d.getAmountDue() -
                                                CRITICAL_AMOUNT), 2);
                  double denominator = 1 + numerator;
                  return numerator/denominator;          // Gamma funkce
            }
            else {
                  return 0;
            }
      }

}
```

Pro reprezentaci fuzzy množin „*vysoká dlužná částka*“ a „*dlouhá doba po splatnosti*“ slouží třída `FuzzySet`. Její instance mohou vzniknout po předání vstupních prvků (výsledek databázového dotazu) a funkce příslušnosti, nebo po předání vypočtených stupňů příslušnosti prvků např. z fuzzy operací nad jinou/jinými fuzzy množinami.

```
public class FuzzySet {

	private MapOfObject memberships; //Stupně příslušnosti
	private MembershipEvaluator evaluator; //Funkce příslušnosti v DB

	// Základní konstruktor s databázovou množinou a funkcí příslušnosti
	public FuzzySet(SetOfObject dbSet, MembershipEvaluator memEval) {
		evaluator = memEval;
		eval(dbSet);
	}

	// Konstruktor se stupni příslušnosti z fuzzy operací
	public FuzzySet(MapOfObject memberhipMap)
	{
		memberships = memberhipMap;
	}

	// Stanovení stupňů příslušnosti
	private void eval(SetOfObject dbSet)
	{
		Object o;
		double v;
		memberships = new MapOfObject();
		for (Iterator iter = dbSet.iterator(); iter.hasNext();) {
			o = iter.next();
			v = evaluator.eval(o);	//Výpočet příslušnosti
			memberships.put(o, v);
		}
	}

	// Příslušnosti všech prvků do fuzzy množiny (včetně nulových)
	public MapOfObject getMemberships() {
		return getMemberships(0);
	}

	// Alfa řez fuzzy množinou (dle minimálního stupně příslušnosti)
	public MapOfObject getMemberships(double minValue) {
		if(minValue > 0) {
			MapOfObject mo = new MapOfObject();
			Object o;
			Double v;
			Iterator it = memberships.keySet().iterator()
			for (it; it.hasNext();) {
				o = it.next();
				v = (Double)memberships.get(o);
				if(v >= minValue) {
					mo.put(o, v);
				}
			}
			return mo;
		}
		else {
			return memberships;		// Odkaz na DB reprezentaci
		}
	}

}
```

Po posouzení všech dluhů z pohledu výše částky a doby po splatnosti je možné obě kritéria zkombinovat a určit do jaké míry splňují jednotlivé dluhy obě podmínky zároveň. Kontext, ve kterém budou posuzovány je realizován třídou `FuzzyOperationContext`. Operace s fuzzy množinami, které mají jediný možný algoritmus jsou implementovány přímo. Ostatní operace jsou jako strategie implementovány v samostatných třídách. Pokud klient využívá výchozí strategie, nemusí o způsobu jejich implementace vědět. Naopak, pokud mu standardní chování nevyhovuje, může ho transparentně upravit.

```
public class FuzzyOperationContext {
	// Nastavení výchozích strategií
	// ...

	// Fuzzy konjunkce s výchozím typem „Standardní konjunkce"
	public FuzzySet and(FuzzySet fs1, FuzzySet fs2) {
		return and(fs1, fs2, new StandardConjunction());
	}

	// Fuzzy konjunkce daného typu - implementace jako strategie
	public FuzzySet and(FuzzySet fs1, FuzzySet fs2,
				ConjunctionStrategy andStrategy) {
		return andStrategy.and(fs1, fs2);
	}

	// Jádro fuzzy množiny - implementace přímo
	public SetOfObject core(FuzzySet fs) {
		SetOfObject s = new SetOfObject();
		Iterator it = fs.getMemberships(1.0).keySet().iterator();
		for (it; it.hasNext();) {
			s.add(it.next());
		}
		return s;
	}

	// Další operace s fuzzy množinami
	// ...
}
```

Aby bylo možné mít pro jednu logickou operaci (v tomto případě fuzzy konjunkci) více různých algoritmů, které se mohou lišit v detailech, ale základ sdílejí, je zavedena třída `ConjunctionStrategy`. Logické operace s fuzzy množinami vyžadují postupný průchod přes všechny prvky menší množiny a jednotlivé strategie se liší až v porovnání stupňů příslušnosti každého prvku v jedné a druhé fuzzy množině. Společný kód je tedy implementován ve třídě, která má pro rozlišení strategií abstraktní metodu, ve které se rozhodne, jaký je stupeň příslušnosti prvku ve výsledné fuzzy množině. Implementace této metody je ponechána na potomcích třídy `ConjunctionStrategy`.

```
public abstract class ConjunctionStrategy {

	public FuzzySet and(FuzzySet fs1, FuzzySet fs2) {
		// Výsledné stupně příslušnosti
		MapOfObject finalMemberships = new MapOfObject();
		Double v1, v2;

		// Iterátor přes všechny klíče menší z fuzzy množin
		Iterator it = getIterator(fs1, fs2);

		while(it.hasNext()) {
			Object o = it.next();
			v1 = (Double) m1.get(o);
			if (m2.containsKey(o)) {
				v2 = (Double) m2.get(o);
				finalMemberships.put(o, andValue(v1, v2));
			} else {
				finalMemberships.put(o, v1);
			}
		}
		return new FuzzySet(finalMemberships); // Výsledná fuzzy mn.
	}

	// Implementace ponechána na potomkovi
	protected abstract double andValue(Double m1, Double m2);
}
```

V příkladu týkajícím se závažnosti dluhu bylo uvedeno, že pokud nelze dluh nespadá jednoznačně do jedné z kategorií „*vysoká dlužná částka*“ nebo „*dlouhá doba po splatnosti*“, tak by dopad na celkové vyhodnocení závažnosti měl být částečně potlačen. Namísto standardní konjunkce je proto vhodnější použít konjunkci součinovou, která je implementována jako potomek obecné strategie pro konjunkci.

```
public class ProductConjunction extends ConjunctionStrategy {

	@Override
	protected double andValue(Double m1, Double m2) {
		return m1*m2;		// Součinová konjunkce stupňů příslušnosti
	}
}
```

Všechny části řešení příkladu je nyní možné zkombinovat a provést vyhodnocení všech dluhů, které jsou uloženy v databázi Versant. Vstupní objekty jsou vybrány databázovým dotazem na pojmenovanou kolekci „`AllDebits`“. Vyhodnocení podmínek je realizováno v kontextu s nastavenými funkcemi příslušnosti a určením způsobu kombinace podmínek pomocí strategie pro součinovou konjunkci. Z výsledku jsou odfiltrovány ty dluhy, které nelze ani z poloviny zařadit do kategorie „*vysoká dlužná částka s dlouhou dobou po splatnosti*“. Na závěr je pro každý dluh zvolen další postup vůči dlužníkovi na základě konečného stupně příslušnosti.

```
// Zahájení transakce
session = new TransSession(dbName);

// Dotaz na dluhy uložené v databázové kolekci typu SetOfObject
SetOfObject debits = (SetOfObject) session.findRoot("AllDebits");

// Vyhodnocení "vysoká dlužná částka s dlouhou dobou po splatnosti"
FuzzyOperationContext fcon = new FuzzyOperationContext();
FuzzySet fs = fcon.and(
             new FuzzySet(debits, new CriticalAmountDue()),
             new FuzzySet(debits, new LongPastDue()),
             new ProductConjunction()  // součinová konjunkce
                          );

// Dluhy se stupněm příslušnosti minimálně 0.5
MapOfObject mo = fs.getMemberships(0.5);

// Načtení dluhů z databáze a jejich další zpracování
Double degree = 0.0;
Iterator it = mo.keySet().iterator();
while (it.hasNext()) {
       Debit debit = (Debit) it.next();
       degree = (Double) mo.get(debit);

       // Další postup na základě stupně příslušnosti
       if(degree > 0.8) {
              legalProcess(debit);
       }
       else if(degree > 0.6) {
              penalty(debit);
       }
       else {
              lastRequestForPayment(debit);
       }
}

// Konec session
session.endSession();
```

Během konstrukce fuzzy množin jsou do klientského systému *postupně* přenášeny pouze detaily jednotlivých dluhů potřebné pro výpočet stupňů příslušnosti. Výsledky jsou zpět transparentně ukládány do databáze a proto není řešení z pohledu dostupné paměti klientského systému omezeno počtem evidovaných dluhů v databázi.

6.10. Známá použití

Rozdělení datových struktur pro reprezentaci objektu a jeho stupňů příslušnosti bylo použito v implementaci podpory fuzzy přístupu pro objektově orientovaný databázový systém ObjectStore [Visnick 2003]. Důvodem byla optimalizace datové struktury na fyzické úrovni, tj. zejména zlepšení práce s databázovými indexy.

Třída reprezentující funkci příslušnosti, tj. poskytující algoritmus výpočtu stupně příslušnosti prvku do fuzzy množiny, je používána v systémech založených na programovacím jazyku FRIL++ [Baldwin].

6.11. Příbuzné vzory

Vzor Strategie byl použit pro zachování volnější vazby mezi rozhraním operace a algoritmem její implementace. Vzor Strategie [Gamma et al. 2003] může být rovněž výhodně doplněn vzorem Muší váha [Gamma et al. 2003], protože není potřeba udržovat stavy mezi jednotlivým voláním operací.

Strategie operací lze v případě sdíleného kódu realizovat s využitím vzoru Šablonová metoda [Gamma et al. 2003].

Pro předání výsledku dotazu při konstrukci fuzzy množin je možné použít vzor DAO [Alur 2003] nebo MS Data vzory [MS Data].

7. Integrace vzoru do metod a nástrojů

Reálného uplatnění získá představený vzor až v okamžiku jeho integrace do procesů vlastního vývoje software. Jak již bylo uvedeno, využití fuzzy přístupu má vliv na velkou část přípravných, realizačních i podpůrných etap životního cyklu software. Je třeba počítat s tím, že dopad bude větší, než v případě např. změny některé technologie. Úkolem úspěšné integrace je proto zejména:

- připravit vývojový tým na změny v procesech,
- zajistit dostatečné know-how business konzultantů, analytiků, vývojářů i pracovníků podpory (nejdůležitější je rozptýlení nejistoty a nedůvěry),
- nastavit vhodně parametry modelů pro odhad složitosti a časové náročnosti softwarových projektů,
- poskytnout nástroje pro efektivní práci s fuzzy pojmy a jejich modelování,
- využít výhod opakujících se postupů a automatizovat je za účelem minimalizace chyb.

Příprava vývojového týmu na změny a zajištění proškolení se v principu neliší od jiných změn v podnikovém prostředí. Jejich opomenutí či nedostatečné zajištění by však v tomto případě mělo fatální následky.

Při měření dopadu fuzzy přístupu při vývoje software na jeho celkovou složitost a dobu vývoje je potřeba vhodně upravit parametry použitých modelů. V případě základních modelů, jako jsou funkční jednice (function point analysis), autor práce doporučuje zvýšit hodnocení faktoru negativně ovlivňujícího podmínky realizace do intervalu $F \in [1.0,1.4]$. U odhadů metodou COCOMO je rovněž vhodné zohlednit vliv znalostí a zkušeností programátorů/analytiků s fuzzy logikou do korekce získaných odhadů. Je třeba však dávat pozor na dvojí započtení, pokud jsou dané metody kombinovány.

Metodami, nástroji, architekturami a postupy jsou myšleny konkrétní prostředky, které má vývojový tým k dispozici. V následujících oddílech jsou shrnuty prostředky, které autor sám navrhl a vytvořil během zpracovávání disertační práce. Jejich účelem je pokrytí integrace až do implementační úrovně.

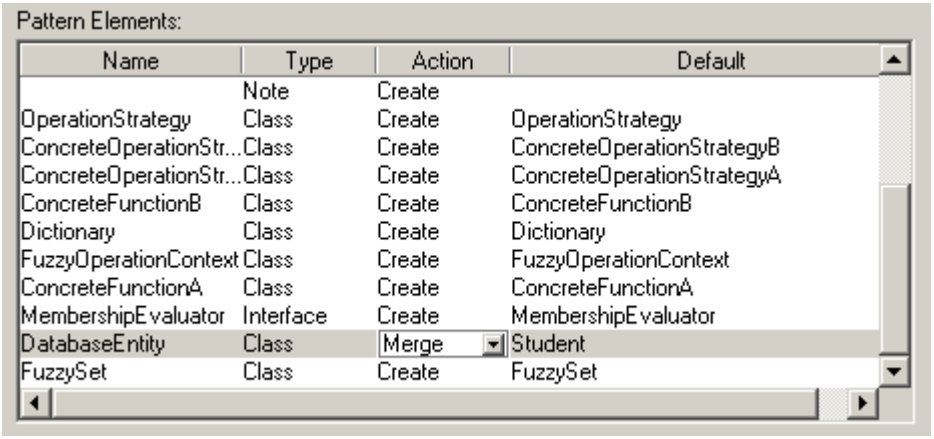
7.1. Integrace s CASE nástroji

CASE (computer aided software engineering) nástroje pomáhají softwarovým inženýrům především s tvorbou modelů a jejich transformacemi, generováním kódu, refaktoringem a konfiguračním řízením, jinak řečeno pokrývají jednotlivé oblasti životního cyklu vývoje software [IEEE]. Existuje celá řada typů CASE nástrojů, které používají různé notace, ovšem sjednocujícím prvkem bývá standard UML, resp. jednotlivá rozšíření (UML profily), která jsou postupně přidávána organizací OMG.

Integraci návrhového vzoru Fuzzy rozšíření OODBMS do CASE nástrojů provedl autor této práce ve dvou krocích:

1. Uvolnění vzoru ve formátu UML (export do XMI viz [OMG XMI]).

   Pro zajištění přenositelnosti vzoru v notaci UML mezi jednotlivými CASE nástroji byl použit OMG standard XMI. Výhodou tohoto formátu je jeho široká podpora z řad dodavatelů CASE nástrojů a jeho rozšíření u veřejnosti. Při načítání XMI souboru lze dokonce v některých CASE nástrojích zvolit sloučení vzoru s již existujícím modelem viz Obrázek 25. Aplikace vzoru se tak zjednodušuje a nezanáší do modelu nadbytečné prvky. XMI podoba vzoru Fuzzy rozšíření OODBMS je umístěna na stránkách autora této práce viz [Volráb web].



Obrázek 25 - Aplikace vzoru v prostředí CASE nástroje Enterprise Architect

2. Vytvoření UML profilu [OMG 2006] pro modelování v notaci Fuzzy UML.
   Ve fázi tvorby konceptuálního modelu je potřeba v CASE nástroji jednoznačně odlišit elementy a vazby, které mohou mít fuzzy povahu, tj. sémanticky korektně reprezentovat stupně příslušnosti. V rešeršní části této práce byla uvedena notace Fuzzy UML, kterou navrhl Ma [Ma 2005]. Přestože jde o často citovanou notaci, nejsou autorovi této disertační práce známy žádné dostupné CASE nástroje, které by ji přímo podporovaly. Za tímto účelem byl vytvořen UML profil, který umožní snadnou integraci této notace do celé řady CASE nástrojů podporujících import profilů (např. Enterprise Architekt, Oracle JDeveloper, Microsoft Visio a další).
   Doplňování UML notace formou UML profilů je standardním a preferovaným způsobem dle OMG, a tím jsou vytvořeny předpoklady pro kompatibilitu a návaznost na další transformační procesy (viz např. integrace v rámci MDA).

7.2. Integrace do architektur

Architektura dle [IEEE] představuje základní organizaci systému zakotvenou v komponentách, jejich vzájemných vztazích, okolním prostředím a principech usměrňujících jeho návrh a vývoj. V praxi se lze setkat s architekturami na různých úrovních abstrakce od modelování business procesů až po implementační.
Provázání návrhového vzoru Fuzzy rozšíření OODBMS provedl autor této práce s dvěma již dříve uvedenými architekturami, a to MDA a SOA, protože je možné je zařadit v oblasti internetově orientovaných aplikací dnes mezi nejpoužívanější.

7.2.1. Provázání s Model Driven Architecture (MDA)

Bezprostředními přínosy vyplývajícím s provázáním navrženého vzoru s MDA jsou:

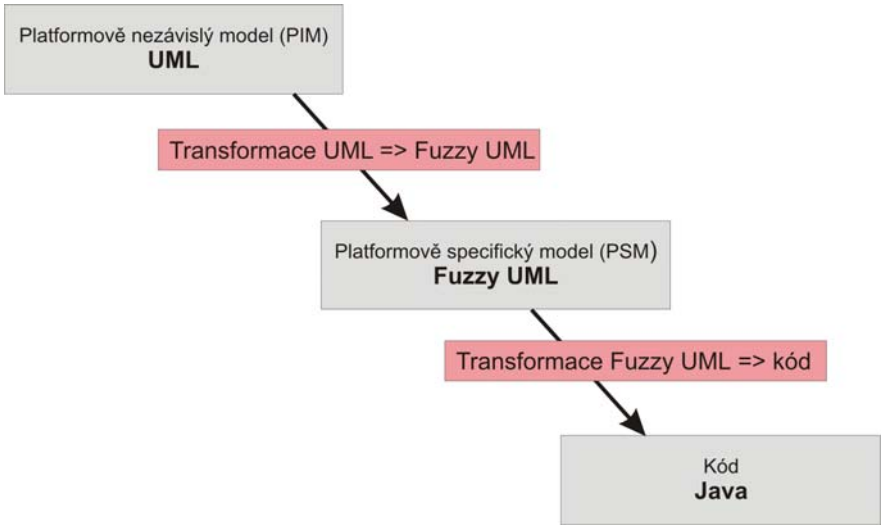
- Transformace z PIM (Platform Independent Model) do PSM (Platform Specific Model), resp. obecného UML modelu do Fuzzy UML profilu.
- Transformace z PSM do kódu, tj. generování kódu ve zvoleném programovacím jazyku a se strukturami specifickými pro konkrétní OODBMS.

Transformace jsou kritickou částí celé MDA architektury. Jde o proces, kdy model *A* vyhovující metamodelu *MA* je převáděn na model *B* vyhovující metamodelu *MB*.

Díky tomu je možné vysledovat původ vytváření a změn jednotlivých částí modelu. K definici transformací slouží rozšíření označované jako MOF/QVT (Query / Views / Transformations), které vnitřně používá především jazyk OCL (Object Constraint Language). Všechny tyto standardy spadají pod organizaci OMG.

Definováním transformačních pravidel na jednotlivých úrovních MDA se zajistí korektní promítnutí změn modelů z vyšších úrovních abstrakce do modelů na nižších úrovních, resp. až na úroveň generovaného kódu.

Předpokladem pro vytvoření transformačních pravidel bylo zavedení Fuzzy UML profilu (viz zavedení profilu v kapitole 7.1), který umožnil rozlišit zdrojové prvky transformace. Nejde o jediný možný přístup, ale byl zvolen jakožto preferovaný sdružením OMG.



Obrázek 26 – Transformační pravidla pro MDA architekturu

Při definici pravidel v MOF/QVT pro transformaci Fuzzy UML modelu je potřeba přihlédnout k určitým omezením, mezi které lze zařadit:

- Existuje více různých implementací MOF/QVT, a tedy konkrétních syntaxí pro zápis transformačních pravidel.
- Počet PSM modelů (různé OODBMS) může být značný.
- Pravidla pro generování kódu jsou dána cílovým programovacím jazykem.
- Ne vždy je možné definování zpětného transformačního pravidla.

Vytvoření uceleného souboru transformačních pravidel bylo mimo stanovené cíle této disertační práce, ale za předpokladu uplatnění výše uvedených principů je možné je doplnit.

7.2.2. Provázání se Service Oriented Architecture (SOA)

Rozšíření funkcionality IS o fuzzy přístup v prostředí podnikové či globální SOA architektury představuje náročný proces. Na rozdíl od spolupráce autonomních fuzzy IS mezi sebou nebo s běžnými IS je infrastruktura SOA tvořena podstatně menšími prvky. Tyto prvky navíc zajišťují obvykle jen dílčí kroky obchodního procesu.

S vyšší granularitou služeb SOA roste také počet rozhraní mezi nimi. Aby SOA přineslo skutečné výhody, musí být zachována jednak transparentnost těchto rozhraní, ale také musí být jednoznačně řečeno, o jaký krok procesu se služba stará. To může být v případě fuzzy přístupu někdy obtížnější specifikovat především z pohledu technologií, na kterých infrastruktura SOA stojí. Při návrhu služeb v rámci SOA lze v zásadě zvolit jednu ze dvou možností:

a) *Fuzzy aplikační logika je z pohledu rozhraní zcela skryta, tj. proces fuzzifikace a defuzzifikace probíhá výhradně uvnitř komponenty poskytující službu.* Výhodou je zachování stávajících rozhraní během změny chování vnitřku „černé skříňky“. Nevýhodou může být naopak potřeba vytvářet služby tak, aby vždy jedna služba zapouzdřila celý fuzzy rozhodovací nebo regulační proces. Takový přístup může vést v mnoha případech k potlačení původní myšlenky návrhu služeb v SOA.

b) *Jednotlivé služby mají rozhraní rozšířena tak, aby fuzzy aplikační logiku mohlo tvořit více vzájemně spolupracujících služeb.* Zde se ovšem v praxi naráží na omezenou sémantiku standardů SOA, které musejí zůstat dostatečně obecné. Ani konkrétní technologie využívané v prostředí SOA, jako jsou webové služby, totiž nemají pro tyto účely vhodné konstrukce. Na úrovni firemních či oborových standardů je však možné definovat např. způsob zachycení fuzzy vstupů/výstupů pomocí WSDL (XML model pro popis webových služeb) i UDDI (registr služeb).

V reálném prostředí budou obvykle oba přístupy kombinovány. Při větším množství spolupracujících služeb může být výhodné uplatnění ESB (Enterprise Service Bus) architektury, která se v poslední době ujala jakožto účinné řešení snižující počet vazeb mezi kooperujícími systémy resp. službami.

Návrhový vzor Fuzzy rozšíření OODBMS byl konstruován tak, aby respektoval různé modely zapojení do prostředí SOA:

1) *Služba databázového systému.* Pokud to zvolený OODBMS umožňuje, lze jako službu zpřístupnit přímo rozhraní objektů (kolekcí) uložených v databázi. Sníží se režie pro komunikaci mezi jednotlivými vrstvami a objem zpracovávaných dat zůstane omezen jen databázovým serverem.
2) *Služba aplikačního serveru.* Klasický model, při kterém je aplikační logika umístěna v samostatné vrstvě nad databázovým serverem. Služba může i nadále využívat výhod plynoucích z návrhového vzoru, ale oproti předchozí variantě je rozhraní služby definováno až v aplikačním serveru. Tento přístup může být výhodnější v případě sdílené fuzzy aplikační logiky.
3) *Služba tenkého klienta.* Ve vysoce distribuovaném prostředí jako jsou klienti v mobilních přístrojích lze za určitých podmínek zpřístupnit službu i z koncových stanic. Příkladem může být instalace mobilní verze OODBMS (např. db4objects) na klientech, kteří využívají pro sběr dat z terénu fuzzy logiku. Jako konkrétní ukázka použití může posloužit sdílení informací o letových podmínkách pro paragliding mezi piloty vybavenými přístroji typu PDA.

7.3. Integrace z pohledu metodik

Využití fuzzy přístupu při vývoji internetově orientovaných IS neklade zvláštní požadavky z pohledu konkrétních metodik. Ať už je projekt řízen pomocí rigorózní metodiky, nebo některé agilní, klíčová zůstává schopnost týmu přijmout a podchytit nárůst složitosti vývoje.

Za tímto účelem autor doporučuje nominovat v rámci projektového týmu některého člena do role doménového specialisty podobně, jako je tomu v případě zavádění nové technologie. Pokud taková role zatím neexistovala, je v návaznosti na to potřeba upravit firemní procesní mapy vývoje tak, aby byla nová role jasně a srozumitelně vymezena.

7.4. Omezení při integraci

Výše uvedené prostředky integrace lze považovat za obecně použitelné, ale často jsou v reálných projektech přítomna určitá omezení, která jejich využití mohou bránit. Jde především o stav, kdy software není stavěn tzv. „na zelené louce“ a z historických důvodů je třeba používat např. taková napojení na datovou vrstvu,

která použití vzoru Fuzzy rozšíření OODBMS znemožní. Podobná situace nastane při pevné vazbě na standardní firemní aplikační rámec.

V takovýchto případech lze doporučit řešení fuzzy části systému formou samostatného modulu či lépe služby ve smyslu SOA architektury.

8. Případové studie aplikace vzoru

Výběr několika případových studií, které byly provedeny, demonstruje praktickou použitelnost obecného vzoru kombinovanou s implementačními detaily jednotlivých platforem. Klíčem k výběru reprezentativních OODBMS a programovacích prostředků bylo nezávislé sdružení ODBMS.ORG (Object Database Management Systems) [ODBMS.ORG], jehož smyslem je podpora výzkumu a tvorba databázových standardů v součinnosti s organizací OMG. Případové studie byly realizovány pomocí technologií, které shrnuje Tabulka 2. Zdrojové kódy všech implementací a testovací příklady jsou dostupné na stránkách autora [Volráb web].

Prostředí/OODBMS	Java 5.0	Smalltalk	C# .NET
Versant	X		
Gemstone/S		X	
ObjectStore	X		
db4objects			X

Tabulka 2 - Přehled technologií v případových studií

Jednotlivé databázové platformy jsou představeny vždy jen z pohledu charakteristik ovlivňujících způsob implementace vzoru. Implementační diagramy tříd, které každou studii doprovází, mají světlejší barvou odlišeny ty části, které jsou implementovány přímo v databázovém systému. Ostatní části jsou pak na rozhraní klientského prostředí.

8.1. Versant Object Database

Versant je vyspělý objektově orientovaný databázový systém, který je optimalizován pro ukládání velkých objemů hierarchických a grafově orientovaných dat z prostředí programovacích jazyků Java a C++ (samostatně je k dispozici i verze určena pro platformu Microsoft.NET s názvem Versant Fast Objects) [Versant].

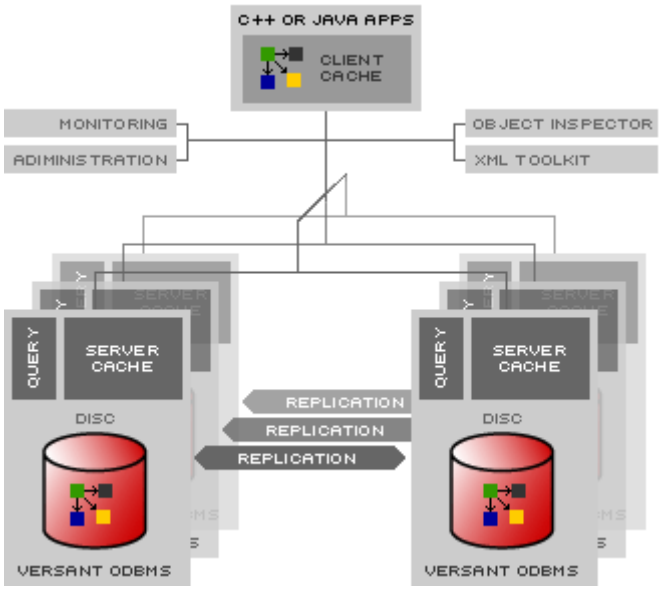
Klíčové vlastnosti

- *Transparentní perzistence objektů v C++ a Java.* Změny v hierarchiích objektů jsou ukládány automaticky při dokončení transakce, a to včetně kolekcí (mapy, slovníky, víceúrovňové mapy).

- *Podpora standardů JDO 2.0, J2EE a ANSI C++*. Standardy využívají Java Versant Interface (JVI) resp. C++/Versant rozhraní.
- *Automatizovaná administrace.* Minimální požadavky za ruční zásahy a automatická optimalizace ukládání dat.
- *Transparentní distribuce dat napříč databázemi.* Škálování databáze pro zvýšení výkonu a maximální velikosti databáze bez úpravy logiky klienta. Podpora synchronní i asynchronní replikace.
- *End-to-end objektová architektura*. Objekty jsou používány v aplikační logice i v perzistentní vrstvě. Nedochází ke konverzím jako u RDBMS.
- *Propracovaný konkurenční přístup.* Zamykání na úrovni objektů pro maximalizaci paralelního přístupu k datům. Podpora optimistického i pesimistického řízení transakcí.
- *Masivní podpora pro multi-threading a multi-session přístup.*
- *Mezinárodní znakové sady.*
- *Manipulace s daty optimalizovaná na rychlost.*
- *Nástroje pro zajištění kritické dostupnosti.*
- *Dynamický vývoj databázového schématu.* Pokročilé funkce migrace objektů při změně jejich třídy.
- *Logické ID objektů pro jejich snadné adresování v paměti.*
- *Podpora SQL (VSQL), ODBC, JDBC připojení pro relačně uložená data.*

Architektura aplikací nad OODMBS Versant

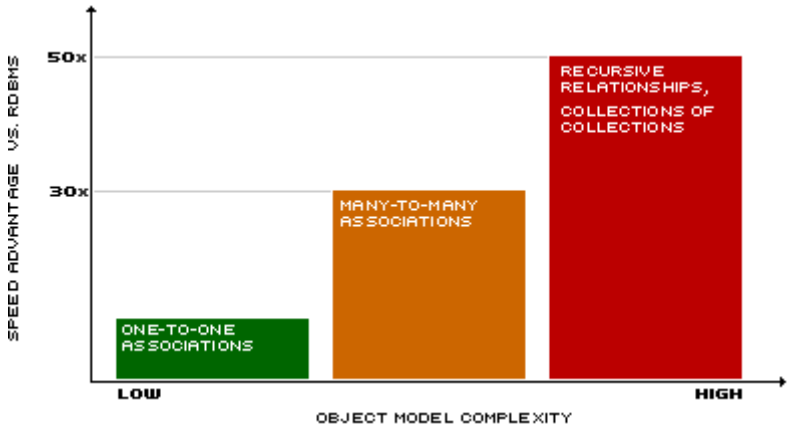
Objekty jsou spravovány prostřednictvím logických identifikátorů, které umožňují fyzický přesun objektů a dělení databáze (partitioning) bez nutnosti jakéhokoliv zásahu do aplikace.



Obrázek 27 - Aplikační architektura Versant. Převzato z [Versant].

Výkon a aplikační výkon

Používají-li aplikace komplexní objektové modely s převážně navigačním přístupem, objektová databáze Versant poskytuje vyšší výkon, než při převodu modelů do relačních databází. U průměrně komplexních objektů lze hovořit u Versant databáze přibližně o 3x rychlejším přístupu. U objektů vysokou úrovní komplexnosti jako jsou vztahy *many-to-many* je přístup obvykle až 30x rychlejší a u vnořených kolekcí a rekurzivních vztahů lze dosáhnout až 50ti násobného zrychlení (viz následující orientační graf).



Obrázek 28 - Výkon Versant vs. RDBMS podle komplexnosti struktur [Versant].

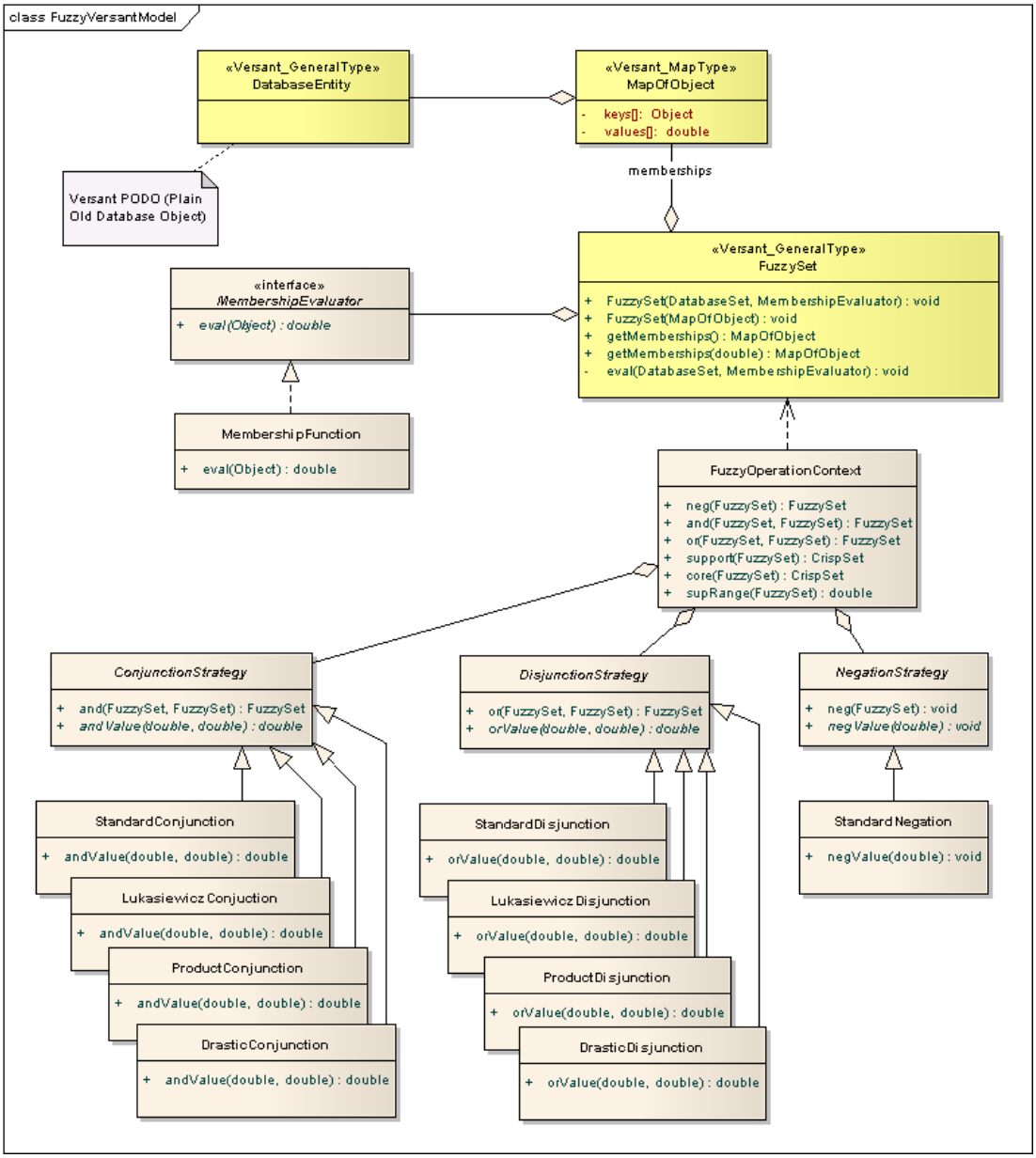
Kompatibilita

Versant Object Database podporuje oficiálně následující platformy: Windows, Solaris, AIX, HP-UX, RedHat Enterprise Linux, ovšem reálně lze provozovat i na řadě jejich variant.

8.1.1. Implementace vzoru

Implementace návrhového vzoru Fuzzy rozšíření OODBMS v prostředí databázového systému Versant je ovlivněna především architekturou a nabízenými režimy perzistence. Architektura ukládání rozděluje objekty na tzv. FCO (objekty uložené jako instance tříd s jedinečným identifikátorem) a SCO (objekty uložené v serializované podobě, tj. jako proud bytů). Režimy perzistence pak umožňují objekty vytvořené v klientském systému ukládat do databáze automaticky, nebo až v okamžiku explicitní žádosti. Třídy, které chtějí přistupovat k FCO objektům a měnit jejich hodnoty, musí být rovněž explicitně určeny.

Jako programovací jazyk pro implementaci logiky vzoru byla zvolena Java 5.0. Jako rozhraní pro komunikaci s databázovým serverem bylo použito JVI a datové typy kompatibilní se standardy ODMG 2.1. Následující schéma zobrazuje konkrétní aplikaci vzoru pro pokrytí výše specifikovaných funkčních a mimofunkčních potřeb.



Obrázek 29 - Model Fuzzy rozšíření pro OODBMS Versant

8.1.2. Součásti a jejich implementace

- `FuzzySet` – Třída typu FCO v režimu volitelné perzistence. Při tvorbě instancí přebírá odkaz na výsledek databázového dotazu a instanci třídy implementující rozhraní `MembershipEvaluator`. Pro stupně příslušnosti prvků používá interní Versant třídu `MapOfObject` (Mapa dle ODMG 2.1).
- `MembershipEvaluator` – Java rozhraní.
- `MembershipFunction` – Pro přiřazení stupně příslušnosti objektu předaného přes rozhraní `MembershipEvaluator` je použito explicitní přetypování `Object` na `DatabaseEntity`. Alternativně lze využít „Java

Generics" a změnit rozhraní tak, aby přijímalo již přímo instance `DatabaseEntity`.

- `FuzzyOperationContext` – Třída přímo implementuje základní algoritmy pro práci s fuzzy množinou. Pro fuzzy logické operace používá vzor Strategie.
- `Conjunction/Disjunction/NegationStrategy` – Abstraktní třídy a algoritmy operací, které lze sdílet mezi strategiemi.
- `StandardConjunction/Disjunction/Negation` – Algoritmy variant fuzzy operací s použitím šablonové metody abstraktních tříd.

8.1.3. Příklad

Využití aplikovaného vzoru lze demonstrovat na příkladu hodnocení bonity klienta při žádosti o půjčku v bance. Do tohoto hodnocení budou obvykle jako parametry vstupovat mimo jiné výše měsíčního příjmu a délka současného pracovního poměru. Dotaz na výběr pouze těch žadatelů, které lze považovat za bonitní z alespoň 80 % lze realizovat v navrženém systému takto:

```
// zahájeni session
session = new TransSession(dbName);

// dotaz na žadatele uložené v databázové kolekci typu SetOfObject
VQLQuery vqlQuery = new VQLQuery(session,
                    "select selfoid from com.versant.odmg3.SetOfObject");
VEnumeration resultEnum = vqlQuery.execute();

// získej referenci na množinu žadatelů
SetOfObject people = (SetOfObject) resultEnum.nextElement();

// vyhodnoť pojem "bonitní žadatel"
FuzzyOperationContext fcon = new FuzzyOperationContext();
FuzzySet fs = fcon.and(new FuzzySet(people, new SufficientSalary()),
                    new FuzzySet(people, new EmployedPerson()),
                    new StandardConjunction() //použij standardní konjunkci
                       );

// zjisti výšku fuzzy množiny
System.out.println(fcon.supRange(fs));

// stanov minimální stupeň příslušnosti do výsledné fuzzy množiny
MapOfObject mo = fs.getMemberships(0.8);

// načti výsledek z DB a zobraz vyhovující žadatele
for (Iterator it = mo.keySet().iterator(); it.hasNext();) {
      Person p = (Person) it.next();
      System.out.println(p + " {Stupeň příslušnosti: " + mo.get(p) + "}");
}
// konec session
session.endSession();
```

8.1.4. Shrnutí

Aplikaci vzoru v prostředí databáze Versant a programovacího jazyka Java je přímočará a nevyžaduje žádné specifické úpravy. S výhodou lze využít zejména transparentního navigačního přístupu napříč hierarchií databázových objektů při vyhodnocování stupně příslušnosti prvků do fuzzy množin. Fuzzy logické operace jsou řešitelné prostou objektovou skladbou a podporují tak v plném rozsahu dynamické sestavování fuzzy podmínek v klientském systému.

8.2. Gemstone/S

Objektový server Gemstone/S [Gemstone 2006] je platformou, která zajišťuje nejen samotnou perzistenci objektů v databázi, ale představuje rovněž aplikační server pro business logiku a integrační server pro spolupráci s RDBMS a dalšími systémy. Nabízí rozhraní z prostředí programovacích jazyků Smalltalk, Java a C.

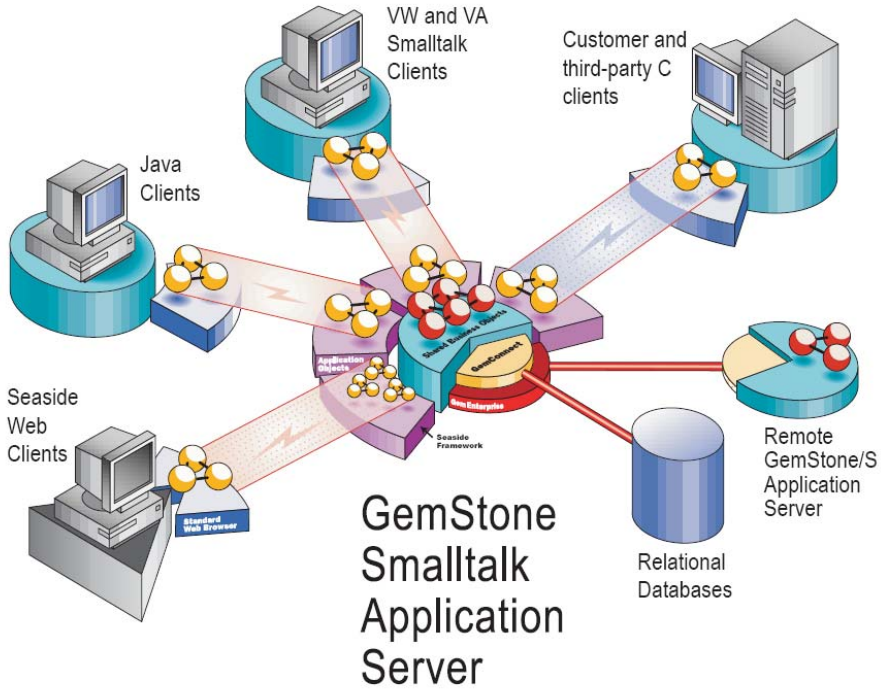
Klíčové vlastnosti

- *Aktivní výkonné jádro.* Uložená data spolu mohou nadále komunikovat v čistě objektovém prostředí podobně, jako v klasickém aplikačním serveru. Komunikace může probíhat zcela uvnitř serveru a není nutné je přenášet na klienta.
- *Intuitivní prostředí.* Snadné použití ve vícevrstvých aplikacích prostřednictvím rozhraní GemBuilder (knihovny a nástroje pro vývoj) a GemConnect (komunikace s RDBMS).
- *Otevřené vývojové prostředí.* Poskytovaná rozhraní umožňují v klientských aplikacích (aplikačních serverech) používat jazyk Smalltalk, Java nebo C.
- *Víceuživatelský přístup.* Podpora optimistického i pesimistického řízení transakcí.
- *Objektový model shodný s prostředím Smalltalk.* Integrace aplikační a perzistentní vrstvy nevyžaduje explicitní převody objektů mezi databázovým a klientským programovacím prostředím. Při použití jazyka Java je přístup rovněž transparentní.
- *Verzování databázových schémat.* V systému může existovat jedna třída ve více verzích. Během vývoje schématu lze pracovat s objekty různých verzí, resp. definovat způsob jejich migrace.

- *Pokročilý model konkurenčního přístupu a správy transakcí.* Zamykání na úrovni objektů a jejich kolekcí. Podpora distribuovaných transakcí v heterogenních prostředích.
- *Polymorfismus uložených objektů není určen vztahy mezi jejich třídami.* Třídy nemusejí mít mezi sebou vztah dědičnosti, postačí průnik jejich protokolů.

Architektura aplikací nad objektovým serverem Gemstone/S

Gemstone/S je součástí tzv. GLASS platformy (GemStone, Linux, Apache, Seaside, Smalltalk), která se snaží konkurovat např. velmi populární LAMP platformě (Linux, Apache, MySQL, PHP). Gemstone/S Object Server jako jádro této platformy spolupracuje se světem klientů ve Smalltalk/Java/C, ale dokáže zprostředkovat také objektový přístup k RDBMS. Prezentační část platformy zajišťuje Seaside, což je aplikační rámec pro tvorbu webových klientů.



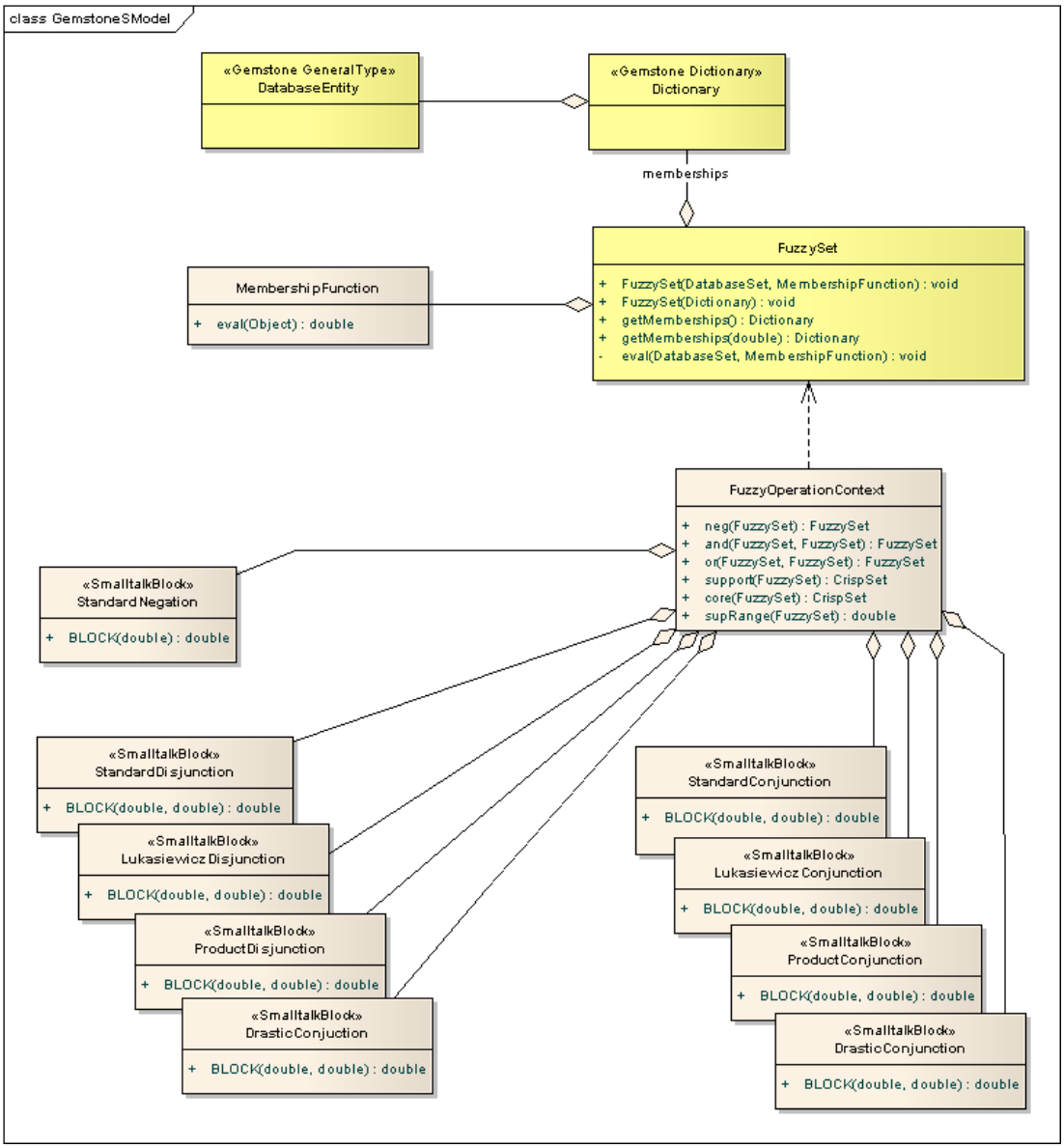
Obrázek 30 – Aplikační architektura Gemstone/S. Převzato z [Gemstone 2006].

Kompatibilita

Gemstone/S Object Server lze provozovat na platformách Solaris, HP-UX, Windows, RedHat Enterprise Linux a AIX. Pro svůj běh vyžaduje server instalaci příslušné Virtual Machine.

8.2.1. Implementace vzoru

Implementace vzoru Fuzzy rozšíření do Gemstone/S serveru ukazuje, že i při obecném návrhu lze využít čistoty a jednoduchosti jazyku jako Smalltalk. Polymorfismus nezávislý na dědění nebo implementaci rozhraní a zapouzdření algoritmů do objektových bloků dávají prostor k efektivní skladbě objektů, která zůstává silně otevřenou a snadno rozšiřitelnou. Nejvíce však ovlivnila implementaci aktivní úloha databázového serveru, ve kterém lze realizovat množinové fuzzy operace a fuzzy logiku, a tím minimalizovat přesun dat mezi serverem a klientem. Jako programovací jazyk byl použit Gemstone Smalltalk (dialekt pro optimalizované databázové dotazy). Schéma znázorňuje jednu z možných implementací vzoru.



Obrázek 31 - Model Fuzzy rozšíření pro Gemstone/S Object Server

8.2.2. Součásti a jejich implementace

- `FuzzySet` – Základní třída pro fuzzy množinu, která pro stupně příslušnosti svých prvků používá databázový typ `Dictionary`.
- `MembershipFunction` – Oproti obecnému vzoru není nutné používat rozhraní `MembershipEvaluator`, ale do fuzzy množiny lze přímo předat instanci libovolné třídy, která má metodu `eval` pro přiřazení stupně příslušnosti. Rozhraní mezi spolupracujícími třídami by bylo možné ještě dále zjednodušit pomocí Smalltalk bloků (podobně jako u strategií pro fuzzy operace viz níže), ovšem v reálných situacích je určení stupně příslušnosti spíše komplexnější operací vyžadující spolupráci řady objektů a nezřídka i externích systémů. Pro tyto účely by již Smalltalk blok nebyl vhodný.
- `FuzzyOperationContext` – Třída, která pro vlastní algoritmy fuzzy operací používá vzor Strategie implementovaný pomocí Smalltalk bloků. Odpadá tak nutnost definovat obecné rozhraní pro strategie a přitom zůstává možnost dynamicky nové algoritmy přidávat a upravovat.
- `StandardConjunction/Disjunction/Negation` – Algoritmy fuzzy operací v podobě Smalltalk bloků, které jsou umístěné v katalogu operací třídy `FuzzyOperationContext`.

8.2.3. Příklad

Zjednodušená skladba objektů ze vzoru je patrná na příkladu webové aplikace typu seznamka. V modelové situaci bude žena klást na aplikaci požadavek, který vyhledá v databázi všechny svobodné muže, kteří jsou buď velmi staří a bohatí (metoda hledání „král“), nebo jsou mladí a na jejich hmotném zajištění v takovém případě nezáleží (metoda hledání „princ“).

Z pohledu implementace je možné rozdělit podmínky na ostré (crisp) a fuzzy. Ostré podmínky (svobodný, muž) poslouží jako primární kritéria pro předvýběr kandidátů. Fuzzy podmínky (bohatý, starý, mladý) mohou být zadány z webového rozhraní (průběh funkcí příslušnosti ve formě např. horizontální reprezentace) a dále jsou předány k vyhodnocení do databáze. U logického spojení podmínek *starý* a *bohatý* lze předpokládat přísnější kritéria, tj. vyžadovat vysoké stupně příslušnosti u obou podmínek zároveň, a proto bude aplikována Lukasiewiczova fuzzy konjunkce. Pro

rozhodování mezi variantami *starý* a *bohatý* NEBO *mladý* postačí aplikace standardní fuzzy disjunkce.

```
"Session z aktualního pripojeni do Gemstone/S"
session := GBSM currentSession.

"Databázový dotaz na (před)výběr svobodných mužů - crisp podmínka"
singleMen := (session at: #People)
                select: {:p | (p.status = 'single') & (p.gender = 'male')}.

"Přiřazení stupňů příslušnosti do fuzzy množin"
fs1 := FuzzySet newWithSet: singleMen memEvaluator: RichPerson new.
fs2 := FuzzySet newWithSet: singleMen memEvaluator: OldPerson new.
fs3 := FuzzySet newWithSet: singleMen memEvaluator: YoungAdultPerson new.

"Nastavení kontextu dotazu (strategie fuzzy operací)"
operation := FuzzyOperationContext new.

"Složený databázový dotaz: [(starý a bohatý) nebo mladý]"
fsResult := operation disjunction:
             (operation conjunction:
                    fs1 with: fs2 strategy: #LukasiewiczConjunction
             )
             with: fs3.

"Jádro fuzzy množiny"
core := operation core: fsResult.

"Nosič fuzzy množiny"
support := operation support: fsResult.

"Výška fuzzy množiny"
height := operation supRange: fsResult.
```

8.2.4. Shrnutí

Implementace vzoru Fuzzy rozšíření OODBMS v prostředí Gemstone/S bylo za pomoci jazyka Smalltalk velmi blízké přirozenému způsobu uvažování o problému a jeho řešení. Zjednodušená struktura objektů, která nevyžaduje zavádění speciálních tříd/rozhraní pro spolupráci nezávislých objektů klade sice vyšší nároky na správnost použití ze strany vývojářů, ale kompenzuje to přehledností, nižší provázaností komponent a jejich snadnější rozšiřitelností. Pokud je logika vyhodnocování fuzzy podmínek umístěna v databázovém systému (jako je tomu v tomto případě), může implementace v Gemstone/S nabídnout při práci s fuzzy množinami vysokou účinnost.

8.3. ObjectStore

Progress ObjectStore Enterprise [Progress 2006] je objektovou databází, která může sloužit jako primární zdroj dat, ale je rovněž optimalizována tak, aby mohla plnit úlohu Cache serveru nad RDBMS. Pro produkční prostředí je k dispozici vedle Enterprise edice také méně robustní (omezení především na úrovni konkurenčního přístupu) edice ObjectStore PSE Pro. Obě edice poskytují shodně rozhraní pro klienty v jazycích Java a C++.

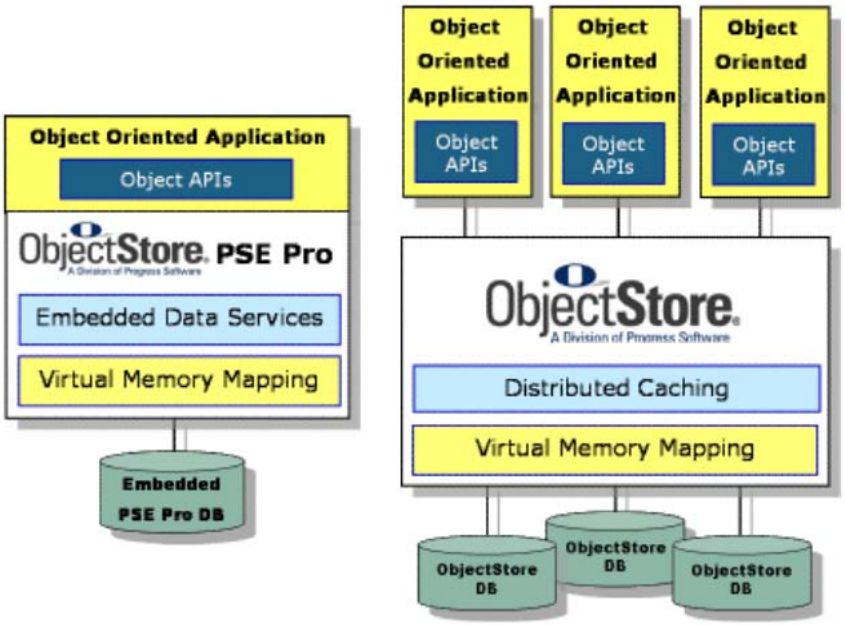
Klíčové vlastnosti

- *Transparentní perzistence.* Hierarchie objektů jsou ukládány a zpětně načítány bez provádění jakýchkoliv konverzí. Navigace mezi objekty je zcela odstíněna od průběžného načítání dat z databáze.
- *Optimalizované typy kolekcí.* Zvláštní typy kolekcí dovolují ukládat až stovky tisíc objektů. Při procházení těchto kolekcí nejsou vytvářeny na straně klienta žádné falešné kopie objektů – data jsou načítána až v okamžiku, kdy jsou skutečně potřebná.
- *Distribuovaná cache.* Změny v kterémkoliv uzlu distribuované aplikace jsou automaticky propagovány do všech částí Cache systému.
- *Architektura Cache-Forward (CFA).* Patentovaný mechanismus pro práci s velkými objemy dat co nejblíže klientské aplikaci. Lokální cache používá virtuální paměť napříč fyzické a diskové paměti.
- *Přístup pro čtení dat během jejich aktualizace.* V režimu pouze pro čtení je možné přistupovat i k datům, které jsou zároveň aktualizována v jiné transakci.
- *Správa vláken.* Jeden kontext spojení do databáze (session) může mít více vláken sdílejících jednu databázovou transakci, nebo může jeden proces s více kontexty používat najednou více nezávislých transakcí.
- *Podpora standardu JDO.* Klienti v prostředí jazyka Java mohou zvýšit přenositelnost svých aplikací pomocí kombinace ObjectStore a standardu JDO.
- *Kompatibilita se syntaxí Java 5.0.* Podpora nových konstrukcí jazyka Java 5.0 je přínosem zejména při práci s typovými kolekcemi, kde lze využít mechanismu generických typů.

Architektura aplikací nad ObjectStore

ObjectStore je k dispozici ve dvou verzích, které se liší v typech aplikací, pro které jsou určeny. Méně robustní verze ObjectStore PSE Pro je určena pro aplikace s jedním procesem a lze ji použít jako tzv. embedded databázi (databáze je součástí aplikace). Je kompaktní a vyznačuje se velmi nízkými paměťovými nároky. Verze ObjectStore Enterprise již obsahuje navíc podporu pro distribuovanou cache, clustering, replikace a on-line zálohování. Není omezena počtem konkurenčních procesů a lze ji použít i jako samostatnou cache pro RDBMS.

Pro vývoj aplikací je podstatné, že obě verze se od sebe neliší v poskytovaném API pro programovací jazyky C++ a Java. Přechod mezi verzemi je převážně konfigurační záležitostí. Z tohoto důvodu byla taky pro následující implementaci návrhového vzoru Fuzzy rozšíření OODBMS použita kompaktnější PSE Pro verze. S výjimkou robustnosti a výkonu by použití této verze nemělo mít od verze Enterprise žádné podstatné odlišnosti.



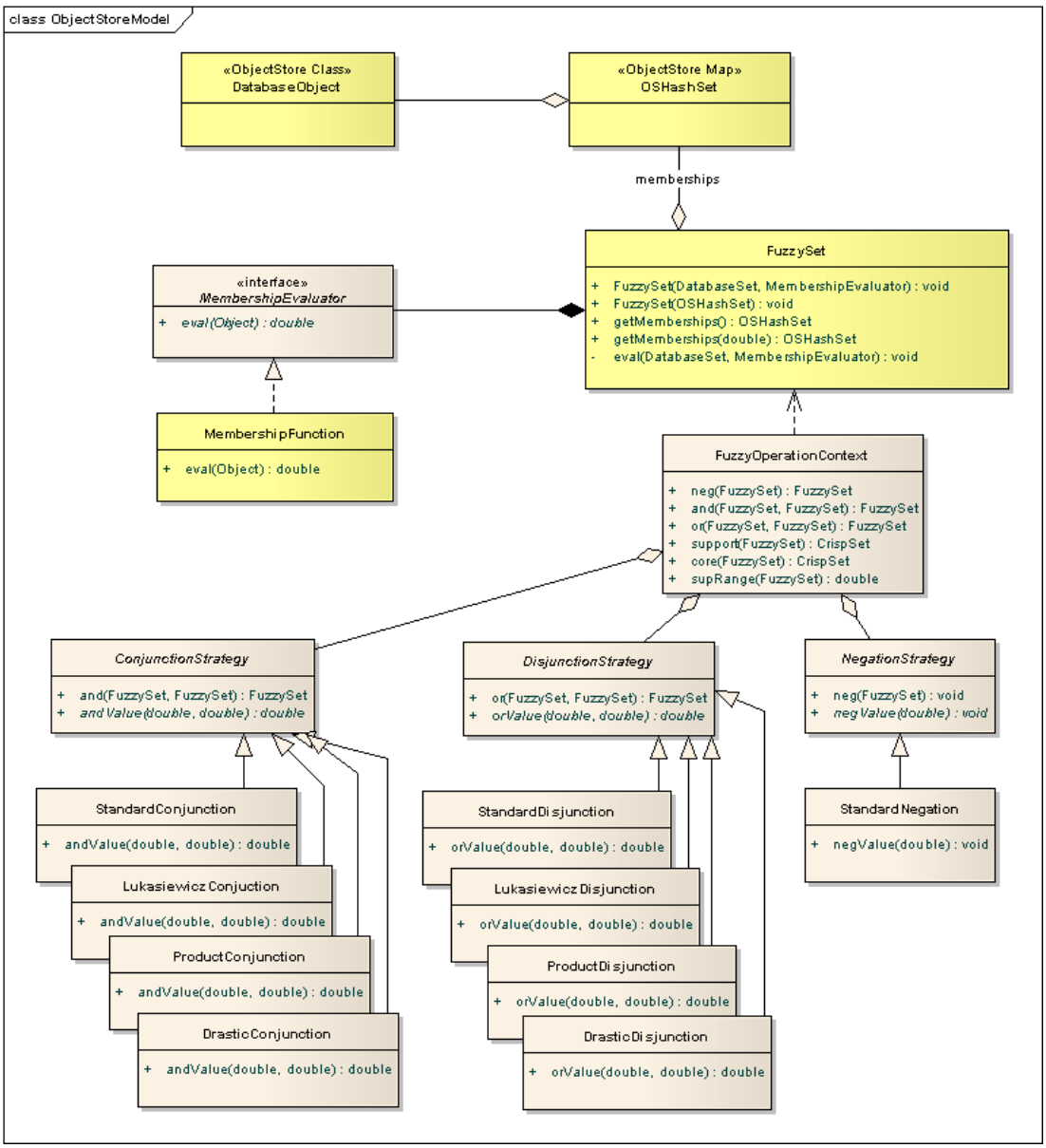
Obrázek 32 - ObjectStore PSE Pro a ObjectStore Enterprise

Kompatibilita

ObjectStore je možné provozovat na platformách: Windows, Redhat Linux, HP-UX, Solaris, Linux a AIX.

8.3.1. Implementace vzoru

Návrhový vzor Fuzzy rozšíření OODBMS v prostředí databázového systému ObjectStore využívá výhodně především propracovaný systém cache pro velmi rozsáhlé kolekce objektů. U kolekcí, které nejsou tak rozsáhlé, je samozřejmě možné použít pro dosažení vyššího výkonu i jiné typy kolekcí, než jsou použity v referenční implementaci níže. ObjectStore pracuje s třídami v několika režimech perzistence (podobně jako systém Versant). Explicitně musí být označeny všechny třídy, jejichž instance jsou ukládány do databáze a dále pak třídy, jejichž metody pracují s vlastnostmi ukládaných tříd. Jako programovací jazyk byla zvolena Java 5.0. Jako rozhraní pro komunikaci s databázovým serverem bylo použito nativního rozhraní.



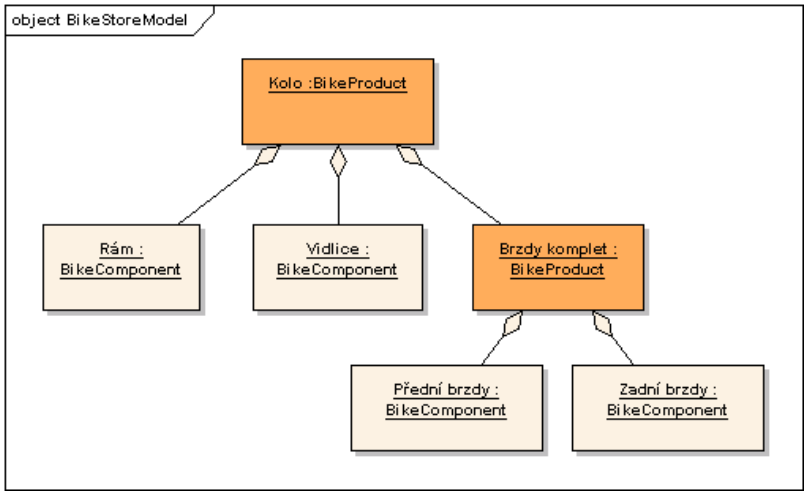
Obrázek 33 - Model Fuzzy rozšíření pro ObjectStore

8.3.2. Součásti a jejich implementace

- `FuzzySet` – Perzistentní třída. Pro stupně příslušnosti prvků fuzzy množiny je použita ObjectStore kolekce `OSHashMap`, která implementuje standardní Java rozhraní `Map`. Je-li možné identitu prvků fuzzy množiny definovat jako řetězec, číslo nebo pole bytů, pak je možné použít místo `OSHashMap` její robustnější varianty jako např. `OSTreeMapInteger`.
- `MembershipEvaluator` – Java rozhraní.
- `MembershipFunction` – Implementace rozhraní `MembershipEvaluator`. Instance mohou být ukládány do databáze spolu s fuzzy množinami, a proto musí být třída označena jako perzistentní.
- `FuzzyOperationContext` – Třída přímo implementuje základní algoritmy pro práci s fuzzy množinou. Pro fuzzy logické operace používá vzor Strategie.
- `Conjunction/Disjunction/NegationStrategy` – Abstraktní třídy a společné algoritmy pro strategie.
- `StandardConjunction/Disjunction/Negation` – Algoritmy fuzzy operací.

8.3.3. Příklad

Navigace v hierarchii objektů bez explicitního načítání dat z databáze je velmi výhodná v situacích, kdy není přiřazení stupně příslušnosti do fuzzy množin zcela triviální. Taková situace může nastat např. v obchodě s jízdními koly. Majitel by se rád zbavil některých kol, která sám sestavuje ze značkových komponent, ale nechce zlevňovat kola s komponentami značky Shimano, protože ví, že se jistě prodají i za stávající ceny. Kritériem pro to, která kola zlevnit je tedy nízký podíl nákupních cen komponent značky Shimano vůči celkové nákupní ceně všech součástí kola.
Jízdní kolo je složeno z jednotlivých komponent, které jsou uloženy v databází. Samostatně prodejné komponenty, resp. skupina komponent je vedená jako produkt. Následující schéma ukazuje, že kolo může být sestaveno z komponent, ale i dílčích produktů.



Obrázek 34 - Objektový diagram skladby jízdního kola

Problém výběru kol ke zlevnění je možné řešit pomocí fuzzy množiny, která vybere všechna jízdní kola s podstatným podílem komponent značky Shimano (funkce příslušnosti dává do poměru nákupní ceny Shimano komponent a celého kola) a následně je vypočten fuzzy doplněk množiny.

```
// připojení k databázi
database = Database.open("BikeStore", ObjectStore.UPDATE);

// zahájení databázové transakce
Transaction transaction = Transaction.begin(ObjectStore.UPDATE);
try {
      // získaní reference dle pojmenované kolekce
      Set products = (Set) database.getRoot("Products");

      // získání Fuzzy množiny všech produktů, ve kterých JSOU
      // převážně zastoupeny komponenty značky Shimano
      FuzzySet fs = new FuzzySet(products, new MainSupplier("Shimano"));

      // doplněk fuzzy množiny - sestavy, ve kterých NEJSOU
      // převážně zastoupeny komponenty značky Shimano
      FuzzyOperationContext fcon = new FuzzyOperationContext();
      FuzzySet fsResult = fcon.neg(fs);

      // minimální stupeň příslušnosti do fuzzy množiny je 0.5
      OSHashMap<Object,Double> mo = fsResult.getMemberships(0.5);

      // načti výsledek z DB a zobraz požadované sestavy a příslušnosti
      Set en = mo.entrySet();
      for (Iterator iter = en.iterator(); iter.hasNext();) {
            Entry entry = (Entry)iter.next();
            System.out.println(entry.getKey() + " : " + entry.getValue());
      }

      // ukončení transakce
      transaction.commit(ObjectStore.RETAIN_HOLLOW);
} catch (Exception e) {
      transaction.abort();  // odvolání transakce v případě výjimky
} finally {
      database.close();     // uzavření spojení do databáze
}
```

8.3.4. Shrnutí

ObjectStore splňuje všechny požadavky pro aplikaci vzoru Fuzzy rozšíření OODBMS a jeho implementace je možná bez specifických úprav. Způsob řešení je na úrovni kódu velmi podobné jako u databázového systému Versant. Za určitých okolností by dokonce bylo možné obě implementace propojit např. využitím návrhového vzoru Abstraktní továrna.

Jednouchý příklad konkrétního využití vzoru v provozní aplikaci naznačuje použitelnost zvoleného řešení i pro netriviální způsoby určení stupňů příslušnosti.

8.4. db4objects

Platforma db4objects [db4objects] je zástupcem OODBMS, které nabízejí vedle komerční licence také bezplatnou GPL licenci a tím přispívají k větší dostupnosti objektových databází i pro nekomerční projekty. S podporou prostředí Java, .NET a Mono nachází db4objects uplatnění jak v klasických vícevrstvých podnikových aplikacích, tak jako embedded databáze v mobilních aplikacích pod J2ME, resp. Microsoft .NET CompactFramework. Poskytovaná rozhraní do programovacích jazyků jsou zaměřena především na maximální jednoduchost použití a transparentnost operací.

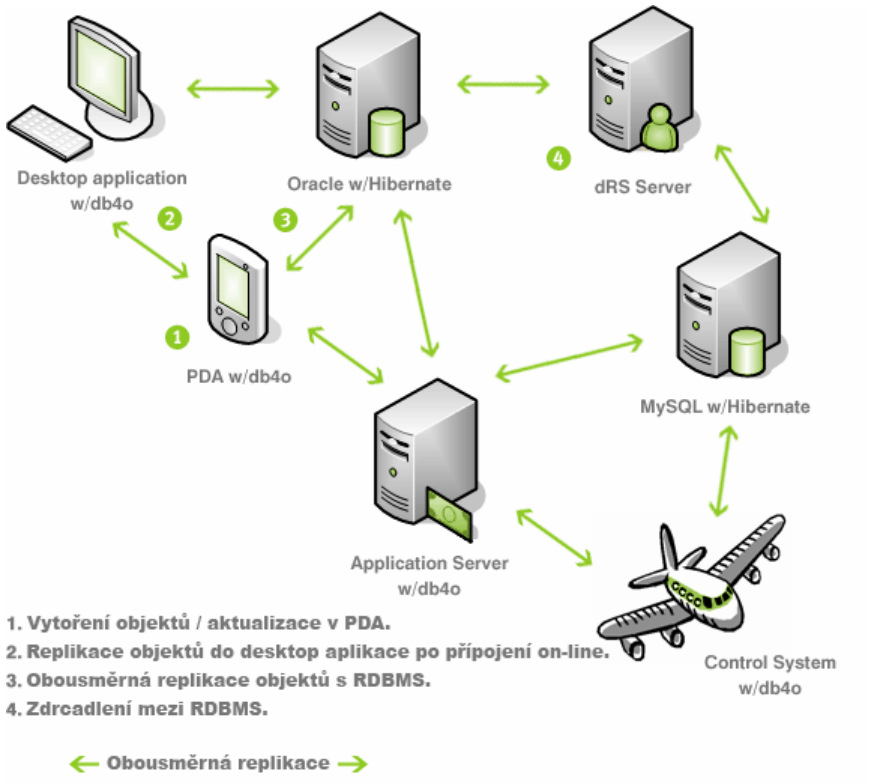
Klíčové vlastnosti

- *Jednoduchost rozhraní.* Klientské systémy používají pro komunikaci s databázovým serverem rozhraní, které vyžaduje pro uložení resp. aktualizaci celé struktury objektů pouze jediný řádek kódu. Pro opětovné načtení dat, resp. formulaci databázových dotazů, je k dispozici více metod, které jsou opět transparentní a zachovávají objektové principy.
- *Dotazovací jazyky.* Databázový systém db4objects podporuje pouze čistě objektové API a vyhýbá se tak problémům textově orientovaných dotazovacích jazyků (typová bezpečnost, refactoring, kontrola při kompilaci). Vedle prototypového přístupu (dotazy QBE) nabízí nativní dotazy pomocí objektových predikátů a rozhraní S.O.D.A. pro dynamickou tvorbu dotazů pro spouštění kódu na straně databázového serveru.

- *Změny ve schématu.* Schéma databáze je dáno objektovou hierarchií. Při její změně jsou instance předchozích verzí migrovány dle zadaných pravidel, nebo automaticky při jejich aktualizaci.
- *Replikace mezi OODBMS s RDBMS.* Spolupráci v prostředí distribuovaných databázových aplikací zajišťuje replikační modul. Synchronizovat lze jak databáze db4objects mezi sebou (např. mobilní zařízení a databázový server), tak databázi db4objects s relačními databázemi prostřednictvím objektově relačního zobrazení zajištěného open source vrstvou Hibernate.
- *Režim embedded databáze.* V prostředí mobilních zařízení může být nasazen db4objects jako embedded databáze, která sdílí stejný virtuální stroj jako klientská aplikace a nevyžaduje žádnou další administraci.

Architektura aplikací nad db4objects

Značná nezávislost databázového systému na platformě a objektově orientovaný replikační proces vytváří vhodné prostředí pro nasazení db4objects v distribuovaném prostředí. Tencí klienti na mobilních zařízeních (J2ME, .NET CompactFramework) mohou synchronizovat data s desktop aplikacemi, stejně jako s relačními databázemi. Architektura a vlastnosti db4objects podporují koexistenci OODBMS a RDBMS v rámci jednoho informačního systému.



Obrázek 35 - Distribuovaná prostředí s db4objects. Převzato a upraveno z [db4objects].

Kompatibilita

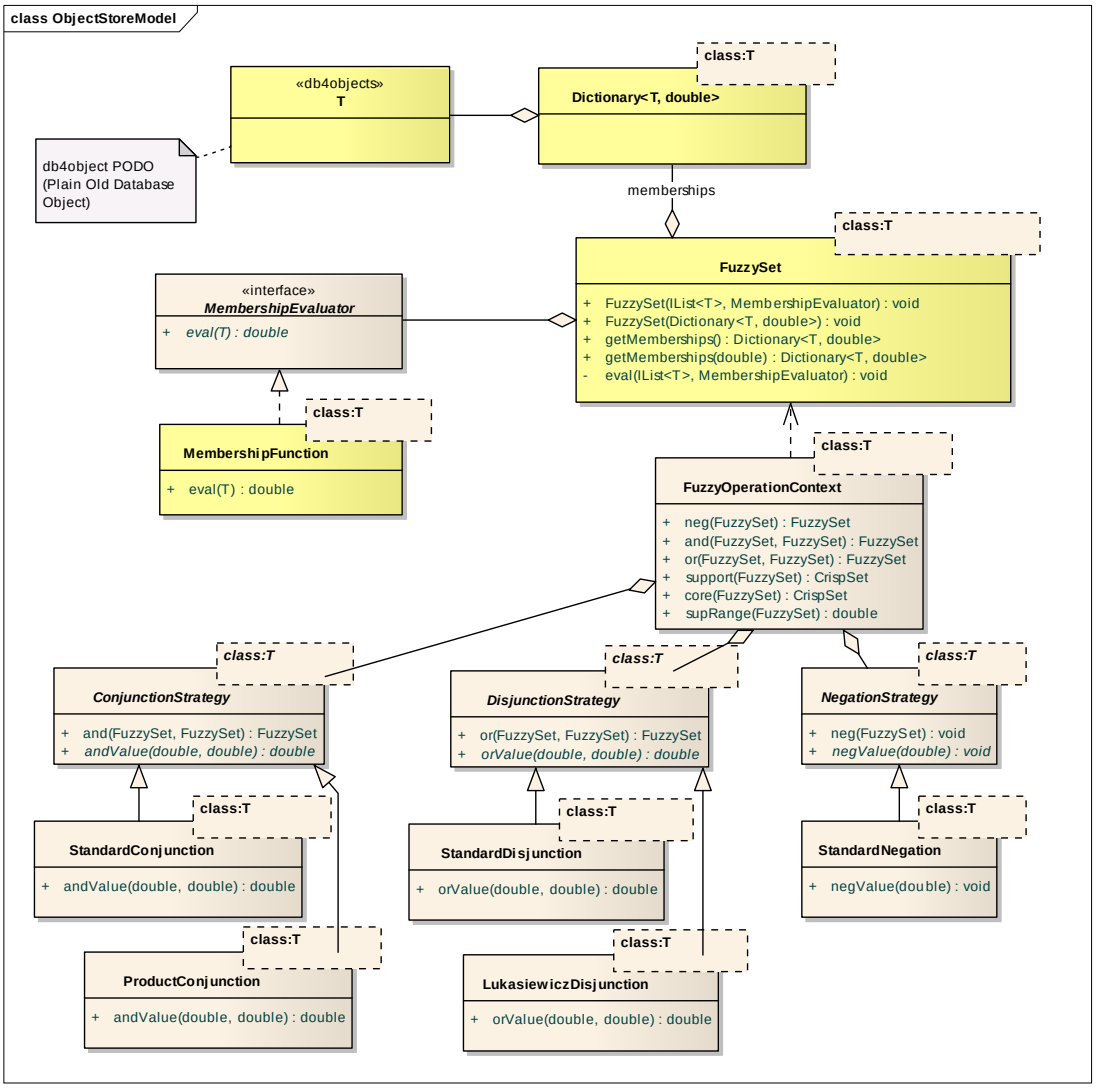
Používání databázového systému db4objects není omezeno na konkrétní operační systémy, resp. je možné je provozovat na libovolném operačním systému, pro který existuje virtuální stroj jazyka Java nebo platformy .NET, a to včetně mobilních verzí (J2ME, Microsoft .NET CompactFramework) a prostředí Mono (port platformy .NET na Linux, Solaris, Mac OS X, Windows a Unixové systémy).

8.4.1. Implementace vzoru

Implementace Fuzzy rozšíření OODBMS pro db4objects se od ostatních liší hned v několika detailech. Prvním z nich je mechanismus perzistence dat, který nevyžaduje žádnou explicitní deklaraci (např. dědění společného předka, implementaci rozhraní, deklaraci v konfiguračním souboru, kompilaci třídy v DBMS). Instanci libovolné třídy je možné učinit perzistentní a zároveň je zachována možnost jejího načtení v prostředí Java i .NET. Druhý implementační detail opět vyplývá ze snahy o co nejjednodušší rozhraní a jde o typovost kolekcí získaných jako výsledek databázového dotazu. Je-li dotaz formulován pomocí nativního rozhraní a s pomocí deklarace generického typu, pak je získaná kolekce

typově bezpečná (na rozdíl od většiny OODMBS, které vracejí jako výsledek databázového dotazu kolekce základních objektů). Typový přístup odstraňuje nutnost explicitního přetypovávání na skutečnou třídu objektů a zlepšuje celkovou čitelnost i udržovatelnost programového kódu. Využití generických typů pro implementaci všech částí vzoru tyto výhody dále doplňují o lepší přenositelnost a znuvupoužitelnost. Přináší také zkrácení programového kódu a možnost jeho statické kontroly ještě před kompilací. V konečném důsledku jsou výhody přeneseny i do oblasti formulace databázových dotazů s fuzzy podmínkami.

Jako prostředí pro implementaci v prostředí db4objects byla zvolena platforma .NET (programovací jazyk C# 2.0) a nativní rozhraní pro typové dotazy.



Obrázek 36 - Model Fuzzy rozšíření pro db4objects

8.4.2. Součásti a jejich implementace

- `FuzzySet<T>` – Perzistentní třída, která pro stupně příslušnosti prvků fuzzy množiny využívá standardní C# generickou kolekci typu `Dictionary<T, double>`. Mezi instancemi ukládanými do databáze a instancemi např. pro mobilní zařízení tedy není žádný rozdíl.
- `MembershipEvaluator<T>` – Typové C# rozhraní.
- `MembershipFunction<T>` – Typová implementace funkce příslušnosti. Polymorfismus při přiřazování stupně příslušnosti instancím různých tříd je možné zajistit nastavením typu `T` na společného předka nebo rozhraní.
- `FuzzyOperationContext<T>` – Třída implementuje základní algoritmy pro práci s fuzzy množinou. Pro fuzzy logické operace používá vzor Strategie. Typová definice zajišťuje, že operace budou prováděny jen nad fuzzy množinami obsahující prvky shodných typů.
- `Conjunction/Disjunction/NegationStrategy` – Abstraktní třídy a společné algoritmy pro strategie. Alternativně lze hierarchii objektů (vzor Strategie) nahradit delegáty.
- `StandardConjunction/Disjunction/Negation` – Algoritmy fuzzy operací implementované buď jako samostatné třídy, nebo úsporněji jako C# delegáti.

8.4.3. Příklad

Distribuované rezervační systémy na Internetu stojí téměř vždy před problémem jak nabízet uživatelům neustále aktuální informace bez nutnosti neustálého nákladného a především časově náročného dotazování do centrálního systému. Tvůrci rezervačních systémů proto musí hledat kompromis mezi on-line dotazem a výsledkem posledního ověření dostupnosti. Za tímto účelem vznikla celá řada propracovaných cache mechanismů, které dovolují s určitým stupněm důvěry stanovit, zda je nabídka ještě stále aktuální.

Situaci lze demonstrovat na hotelovém rezervačním systému, který je dostupný na webových stránkách jednotlivých prodejců ubytování (portály cestovního ruchu, cestovní kanceláře). Aktuální informace o dostupnosti jsou drženy v globálním rezervačním systému (např. GoGlobal, Amadeus).

V databázovém systému db4objects jsou uloženy informace o nabízených hotelech včetně adres a GPS pozic. Typický uživatel bude vyhledávat hotel podle místa a minimální kategorie (počet hvězdiček). Dalším hlediskem zejména u velkých měst bývá vzdálenost od centra města, kterou je vhodné vyhodnotit pomocí fuzzy podmínky (vzdálenost GPS souřadnic od souřadnic centra). Nejdůležitějším kritériem pro sestavení nabídky je poté informace o dostupnosti. Tu lze rozdělit na dvě dílčí kritéria:

a) *Kdy byla z centrálního systému naposledy načtena skutečná dostupnost.* Čím starší tato informace je, tím je možné ji považovat za méně důvěryhodnou.
b) *Jak vzdálený je začátek termínu ubytování.* Skutečná dostupnost hotelu se bude rychleji měnit s blížícím se termínem požadovaného ubytování.

V databázi budou nejprve vyhledány hotely vyhovující podmínkám uživatele (hotel v Praze s minimálně 3 hvězdičkami) a následně bude vyhodnocena kombinovaná fuzzy podmínka pro posouzení lokality hotelu a jeho dostupnosti. Z výsledku budou nabídnuty na webových stránkách jen ty hotely, u nichž lze prohlásit, že podmínce vyhovují alespoň z poloviny.

Před samotným prodejem bude samozřejmě nutné on-line dotaz na dostupnost v globálním systému skutečně provést (jako ochranu uživatele), ale tento dotaz se již bude týkat jednoho konkrétního hotelu a proto lze akceptovat režii s tím spojenou.

```
// Připojení k DB
IObjectContainer db = Db4oFactory.OpenFile("hotelsDB.yap");
try
{
   // Nativní a typově bezpečný dotaz na instance třídy hotel,
   // které vyhovují podmínce stanovené ve formě typového
   // anonymního delegáta
   IList<Hotel> hotels = db.Query <Hotel> (delegate(Hotel hotel) {
                    return hotel.Address.City.StartsWith("Praha") &&
                            hotel.Stars >= 3;
                });

   // Fuzzy podmínky (typově bezpečné) na hotely, které jsou
   // blízko centra města a jsou dostupné (jako dostupné budou
   // zobrazeny hotely, pro které byla dostupnost v nedávnosti
   // ověřena v kombinaci s posouzením doby do začátku ubytování)
   FuzzyOperationContext<Hotel> oper = new FuzzyOperationContext<Hotel>();
   FuzzySet<Hotel> fuzzyResult =
       oper.and(
              new FuzzySet<Hotel>(hotels, new NearCenter<Hotel>()),
              oper.and(
                new FuzzySet<Hotel>(hotels, new ValidAvailability<Hotel>()),
                new FuzzySet<Hotel>(hotels, new Available<Hotel>())
                      )
              );
   // Vypiš hotely, které splňují výslednou podmínku alespoň z poloviny
   Console.WriteLine(fuzzyResult.getMemberships(0.5));
}
finally
{
   // Bezpečné uzavření databáze
   db.Close();
}
```

8.4.4. Shrnutí

Db4objects je databázovou platformou v mnoha rysech neobvyklou. Přenositelnost mezi serverem a mobilním klientem, nativní přístup do databáze pro .NET i Java prostředí a další charakteristiky vytváří podmínky pro tvorbu obdobně koncipovaných klientských systémů.

Z představené implementace a příkladu je patrné, že uplatnění vzoru Fuzzy rozšíření OODMBS je možné provést v souladu se specifickými vlastnostmi databázové platformy i programovacího jazyka klientského systému. Nativní dotazy, typovost rozhraní nebo jazykové konstrukty typu delegát jsou všechno techniky, které zvyšují produktivitu i znuvupoužitelnost a zároveň zlepšují sémantiku softwarového řešení, což jsou hlavní důvody pro aplikaci návrhových vzorů.

9. Závěr

Téma disertační práce směřovalo do oblasti zlepšování metod a nástrojů pro tvorbu informačních systémů. Soustředilo se na modelování neurčitosti v datech a na manipulaci s těmito daty pomocí prostředků, které nabízejí objektově orientované databázové systémy (OODBMS). Na základě řady vědeckých prací a setkání odborníků na konferencích v posledních letech lze usuzovat, že propojení uvedených dvou oblastí může přinést do budoucna značnou konkurenční výhodu. Velké objemy dat, které dosud bylo možné zpracovávat je omezeně pomohou lépe porozumět situacím a procesům, které zatím není člověk schopen přesně kvantifikovat a popsat. Zásadní význam může mít zpracování neurčitosti v OODBMS také na kvalitu rozhraní mezi uživatelem a informačním systémem, jelikož komunikace může probíhat z pohledu člověka přirozenějším způsobem a za použití vyjadřovacích prostředků běžného života.

V rešeršní části disertační práce byly shrnuty hlavní přístupy, které používají pro manipulaci s neurčitostí současné databázové technologie. Podrobněji analyzována byla zejména oblast fuzzy logiky, která má ve spojení s moderními OODBMS předpoklady k tomu, aby se vypořádala s komplexností současných informačních systémů (včetně internetově orientovaných). Z analýzy vyplynulo, že **současné přístupy k využití fuzzy logiky v OODBMS jsou silně implementačně závislé a nemají jednotný koncept, přestože z pohledu architektury sdílejí řadu společných principů**.

Současné přístupy k řešení problematiky využití fuzzy přístupu v OODBMS vedlo autora práce k rozhodnutí navrhnout obecnější řešení. Stěžejní částí bylo vymezení těch principů fuzzy logiky, které jsou v současných (implementačně závislých) řešeních úspěšně využívány, a měly by tedy být obsaženy i v obecném řešení. Autor provedl **formální vymezení** ve třech vzájemně provázaných oblastech:

a. **oblast modelových situací,**
b. **oblast funkčnosti a**
c. **oblast mimofunkčních požadavků.**

Uvedené vymezení určuje nejen rozsah řešení, ale rovněž poskytuje prostředky pro jeho průběžné rozšiřování.

Autor v této práci formuluje svůj původní přístup k sjednocení dnes používaných metod a postupů využívání fuzzy přístupu v OODBMS jako **návrhový vzor** „**Fuzzy rozšíření OODBMS**“. Vzor je uveden v klasické GoF struktuře, která je jednou z nejrozšířenějších forem sdílení znovupoužitelných řešení v softwarovém inženýrství. Pomocí vzoru autor navrhuje strukturu celého řešení, jeho součásti, omezení a podmínky, za jakých je možné a výhodné ho aplikovat. Vzor přitom zůstává otevřený z pohledu doplňování dalších fuzzy operací a jejich algoritmů.

Unikátním rysem navrženého řešení je způsob reprezentace fuzzy množin. Díky němu je dosahováno nižší prostorové a časové složitosti operací s fuzzy množinami. Dalších charakteristickou vlastností je podpora více tříd fuzzy operací. Z příkladů průběžně uvedených v práci je zřejmé, že volnost při výběru nejvhodnější třídy fuzzy operací rozšiřuje použitelnost vzoru i pro situace, ve kterých neposkytují stávající řešení uspokojivé výsledky.

Autorova snaha o srozumitelnost, znovupoužitelnost a bezprostřední aplikovatelnost vzoru je podpořena integrací vzoru s vývojovými nástroji a architekturami SOA a MDA. Za tímto účelem byl **vzor zpřístupněn ve formátu XMI určeném pro import/export UML modelů mezi CASE nástroji**. Zároveň **byl vytvořen Fuzzy UML profil, navržen postup zahrnutí Fuzzy UML do transformačních pravidel MDA a vymezeny způsoby konstrukce fuzzy orientovaných služeb v prostředí SOA**. Zavedené metody a postupy zjednodušují návrh nových či rozšiřování stávajících informačních systémů. Poskytují konkrétní nástroje pro modelování neurčitosti a převod modelu do implementace.

Pro ověření použitelnosti, úplnosti a reálné aplikovatelnosti byla autorem realizována sada případových studií v prostředí databázových systémů Versant, Gemstone/S, db4objects, ObjectStore a programovacích jazyků Java, C# a Smalltalk. Výsledky studií prokázaly, že **zavedený návrhový vzor je nezávislý na konkrétním databázovém systému či programovacím jazyku**.

V této práci navržený **obecně použitelný postup pro aplikaci principů fuzzy logiky v objektově orientovaném databázovém systému ve formě návrhového vzoru** umožňuje:

- pracovat s neurčitostí velkých objemů dat jednotným způsobem, a to nezávisle na zvoleném OODBMS a/nebo objektově orientovaném programovacím jazyku,

- sdílet postupy návrhu a implementace fuzzy rozšíření OODBMS,
- modelovat neurčitost dat pomocí Fuzzy UML,
- automatizovat tvorbu zdrojového kódu aplikací (s pomocí CASE nástrojů),
- zvýšit přenositelnost a rozšiřitelnost aplikací,
- omezit nároky na znalosti a zkušenosti vývojářů.

Architektura navrženého řešení zohledňuje některá specifika informačních systémů v prostředí internetu jakou jsou kritická doba odezvy, distribuované a heterogenní prostředí, nestavové chování přenosových prostředků apod. Díky tomu nalezne vzor uplatnění v celé řadě aplikací v režimu on-line služeb. Např. rezervační systémy mohou kombinovat úrovně služeb (nalezení *levné* letenky spolu s hotelem *střední* kategorie), marketingové kampaně vyberou efektivně cílové zákazníky (*často* nakupující *starší* muži s *vyšší* průměrnou útratou). Stejně dobře však mohou využít výhod fuzzy přístupy i ostatní typy informačních systémů. Bankovní aplikace citlivěji posoudí bonitu klienta (dle věku a výše příjmu), nebo mu nabídnou optimalizaci portfolia s ohledem na jeho investiční strategii (*mírně agresivní* investor s *dlouhodobým* investičním horizontem). V poslední době se otevírají nové možnosti především v oblastech mobilních a vestavěných databázových systémech. Příkladem jsou regulační mechanismy moderních automobilů BMW, které sbírají tisíce údajů z desítek senzorů a následně přizpůsobují brzdný účinek pedálů či odrazivost zpětných zrcátek (ochrana proti oslnění).

Výsledky disertační práce byly průběžně publikovány na mezinárodních vědeckých konferencích a jsou rovněž součástí řešení **víceletých grantových projektů podporovaných MŠMT a GAČR**:

- IZMAN – Inteligentní nástroje pro hodnocení relevance a strukturování obsahu obecných i specializovaných zdrojů dat a znalostí (2C06004),
- AMIMADES – Inteligence prostředí pro podporu manažerského rozhodování (402/06/1325).

Autor práce také navázal kontakt ohledně praktického uplatnění navrženého řešení s Bezpečností informační službou ČR.

Další rozvoj myšlenek shrnutých do současné podoby návrhového vzoru je předpokládán ve dvou rovinách. První je postupné doplňování vzoru o další operace fuzzy logiky. Tou druhou je vylepšení integrace s nástroji pro automatickou perzistenci dat.

10. Použité zdroje

Literatura

[Alexander 1977] ALEXANDER, CH., ISHIKAWA, S., SILVERSTEIN, M. et al. *A pattern Language*. 1. vyd. New York: Oxford University Press, 1977. 1216 s. ISBN 978-0195019193.

[Alur 2003] ALUR, D., CRUPI, J., MALKS, D. *Core J2EE Patterns: Best Practices and Design Strategie*. 2. vyd. Palo Alto: Sun Microsystems, 2003. 650 s. ISBN 0-13-142246-4.

[Arlow 2003] ARLOW, J., NEUSTADT, I. *UML a unifikovaný proces vývoje aplikací*. Přeložil Bogdan Kiszka. 1. vyd. Brno: Computer Press, 2003. 387 s. ISBN 80-7226-947-X.

[Atkinson 2000] ATKINSON, C., & KÜHNE, T. *Strict profiles: Why and how*. In A. Evans, S. Kent, & B. Selic (Eds.), „UML" 2000 – The Unified Modeling Language, Third International Konference (Lecture Notes in Computer Science 1939, pp. 309-322). New York: Springer

[Baldwin 1983] BALDWIN, J. Knowledge engineering using a fuzzy relational inference language. In *Proceedings of the IFAC Symposium on Fuzzy Information Knowledge Representation and Decision Analysis*, 1983, s. 15-21.

[Baldwin 2000] BALDWIN, J., CAO, T.H., MARTIN, T.P. et al. Towards soft computing object-oriented logic programming.In. *Proceedings of the Ninth IEEE International Conference on Fuzzy Systems*, s. 768-773.

[Barbara 1992] BARBARA, D., GARCIA-MOLINA, H., PORTER, D. The management of probabilistic data. In *IEEE Transactions on Knowledge and Data Engineering*, 1992, no. 4, s. 487-502. ISSN: 1041-4347.

[Bezdek 1981] BEZDEK, J. *Pattern recognition with fuzzy objective function algorithms*. 1. vyd. New York: Plenum Press. 1981. 272 s. ISBN 978-0306406713.

[Boehm 2000] BOEHM, B. W., ABTS, CH., BROWN, A. W. et al.. *Software Cost Estimation with Cocomo II*. 1. vyd. Indianapolis: Prentice Hall PTR, 2000. 544 s. ISBN 978-0130266927.

[Bordogna 1999] BORDOGNA, G., LUCARELLA, D., & PASI, G. A fuzzy object-oriented data model managing vague and uncertain information. *International Journal of Intelligent Systems*, 1999, vol. 14, no. 7, s. 623-651. ISSN: 0884-8173.

[Bosc 1997] BOSC, P., DUBOIS, D., PIVERT, O. et al. Flexible queries in relational databases: The example of the division operador. In *Theoretical Computer Science*, 1997, no. 171, s. 281-302. ISSN: 0304-3975.

[Boss 1999] BOSS, B., HELMER, S. *Index structures for efficiently accessing fuzzy data including cost models and measurements*. Fuzzy Sets and Systems 108(1), pp. 11-37. 1999.

[Berka 1997] BERKA, M., KUČERA, J., MACUR, J., et al. *WWW-multimediální informační prostředí Internetu*. 1. vyd. Brno: UNIS, 1997. 159 s. ISBN CPISBN-632.

[Bruckner 2001] BRUCKNER, T. *Řízení služeb informatiky v podmínkách outsourcingu*. Disertační práce, VŠE, Praha, 2001.

[Buchalcevová 2005] BUCHALCEVOVÁ, A. *Metodiky vývoje a údržby informačních systémů*. 1. vyd. Praha: Grada, 2005. 164 s. ISBN 80-247-1075-7.

[Buschmann 1996] BUSCHMANN, F. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. 7. vyd. Chichester: Wiley, 1996. 476 s. ISBN 978-0471958697.

[Castagnetto 2001] CASTAGNETTO, J., RAWAT, H., SCHUMANN, S., et al. *Programujeme PHP profesionálně*. Přeložil L. Roubíček. 1. vyd. Praha: Computer Press, 2001. 656 s. Přeloženo z Professional PHP Programming. ISBN 80-7226-310-2.

[Date 1986] DATE, C. J. Null values in database management. In *Relational Database: Selected Writings*. 1. vyd. Boston: Addison-Wesley Publishing Company, 1986. 497 s. ISBN 978-0201141962.

[Eckel 2001] ECKEL, Bruce. *Myslíme v jazyku Java*. Přeložil Bogdan Kiszka. 1. vyd. Praha: Grada, 2001. 472 s. (Myslíme v …, Knihovna zkušeného programátora.) Přeloženo z Thinking in Java 2nd Edition. ISBN 80-247-0027-1.

[Fowler 2002] FOWLER, M., MEUNIER, R., ROHNERT, H. et al. *Patterns of Enterprise Application Architecture*. 10. vyd. Boston: Addison-Wesley Professional, 2002. 500 s. ISBN 978-0321127426.

[Frankel 2003] FRANKEL, D. S. *Model Driven Architecture: Applying MDA to Enterprise Computing*, OMG Press, 2003. ISBN 0-471-31920-1.

[Galindo et al. 2006] GALINDO, J., URRUTIA, A., PIATTINI, M. *Fuzzy databases: Modeling, Design and Implementation*. 1. vyd. Hershey: Idea Group, 2006. 320 s. ISBN 1-59140-324-3.

[Gamma et al. 2003] GAMMA, E., HELM, R., JOHNSON, R., et al. *Návrh programů pomocí vzorů: Stavební kameny objektově orientovaných programů*. 1. vyd. Praha: Grada, 2003. 388 s. ISBN 80-247-0302-5.

[Giardina 1979] GIARDINA, C. *Fuzzy databases and fuzzy relational associative processors*. Technical Report. Hoboken: Stevens Institute of Technology, 1979.

[Grant 1980] GRANT, J. Incomplete information in a relational database. *Fundamenta Informatice*, 1980, no. 3, s. 363-378. ISSN: 0169-2968.

[Hansen 1999] HANSEN, D., ADAMS, D., GRACIO, D. *In the trenches with ObjectStore*. Theory and Practise of Object Systems, 5(1) pp 201-207. 1999.

[Isaacs 1998] ISAACS, Scott. *Dynamické HTML*. Přeložil J. Dudr. 1. vyd. Praha: Computer Press, 1998. 436 s. Přeloženo z Inside Dynamic HTML. ISBN 80-7226-083-9.

[Johnson et al. 2004] JOHNSON, R., HOELLER, J. *Expert One-on-One J2EE Development without EJB*. 1. vyd. Indiana: Wiley, 2004. 552 s. ISBN 0-7645-5831-5.

[Kadlec 2004] KADLEC, V. *Agilní programování: Metodiky efektivního vývoje software*. 1. vyd. Brno: Computer Press, 2004. 278 s. ISBN 80-251-0342-0.

[Kosek 1998] KOSEK, Jiří. *PHP: tvorba interaktivních internetových aplikací*. 1. vyd. Praha: Grada, 1998. 492 s. ISBN 80-7169-373-1.

[Kyte 2005] KYTE, T. *Oracle: Návrh a tvorba aplikací*. Přeložila Anna Rychetská. 1. vyd. Brno: Computer Press, 2005. 694 s. ISBN 80-251-0569-5.

[Lacko 2002] LACKO, Luboslav. *Oracle – Správa, programování a použití databázového systému*. Přeložil Marek Kocan, Michal Brůha. 1. vyd. Praha: Computer Press, 2002. 464 s. ISBN 80-7226-699-3.

[Ma et al. 2004] MA, Z. M., et al. *Extending object-oriented databases for fuzzy information modeling*. Information Systems, 29(5), p421-435.

[Ma 2005] MA, ZONGMIN. *Advances in fuzzy object-oriented databases: Modeling and application*. 1. vyd. Hershey: Idea Group, 2005. 343 s. ISBN 1-59140-385-5.

[Medina et al. 1994] MEDINA, J. M., et al. *GEFRED. A generalized model of fuzzy relational databases*. Information Sciences, 76(1-2), pp87-109. 1994

[Merunka et al. 1999] MERUNKA, V., VOSTROVSKÝ, V. *Databázové systémy*, 2. vyd. Praha: Provozně ekonomická fakulta ČZU v Praze, 1999. 168 s. ISBN 80-213-0543-6.

[Merunka 2002] MERUNKA, V. *Objekty v databázových systémech*. ČZU Praha, Praha 2002. ISBN 80-213-0318-2.

[Merunka 2004] MERUNKA, V. *Objektový přístup v databázové technologii*. Sborník konference Tvorba softwaru 2004. Ostrava: Tanger 2004, ISBN 80-85988-96-8.

[Mouaddib 1994] MOUADDIB, N. *Fuzzy identification database: The nuanced relation division*. In *International Journal of Intelligent Systems*, 1994, no. 9, s. 461-473, ISSN: 0884-8173.

[Navara 2002] NAVARA, M., OLŠÁK, P. *Základy fuzzy množin*. 1. vyd. Praha: Vydavatelství ČVUT, 2002. 136 s. ISBN 80-01-02585-3.

[Novák 2000] NOVÁK, Vilém. *Základy fuzzy modelování*. 1. vyd. Praha: Ben – technická literatura, 2000. 176 s. ISBN 80-7300-009-1.

[Písek 2003] PÍSEK, Slavoj. *ASP.NET – začínáme programovat: podrobný průvodce začínajícího uživatele*. 1. vyd. Praha: Grada, 2003. 228 s. ISBN 80-247-0526-5.

[Pokorný 2004] POKORNÝ, J. *Konstrukce databázových systémů*. 2. vyd. Praha: ČVUT, 2004. 166 s. ISBN 80-01-02898-4.

[Polák et al. 2003] POLÁK, J., MERUNKA, V., CARDA, A. *Umění systémového návrhu*, 1. vyd. Praha: Grada, 2003. 196 s. ISBN 80-247-0424-2.

[Rashid 2004] RASHID, Alwais. *Aspect-Oriented Database Systems*. 1. vyd. Lancaster: Springer, 2004. 176 s. ISBN 3-540-00948-5.

[Robertson 2006] ROBERTSON, S., ROBERTSON, J. *Mastering the Requirements Process*. 1. vyd. London: Addison Wesley Professional, 2006. 416 s. ISBN 0-321-41949-9.

[Saaty 1980] SAATY, T.L. *The analytic hierarchy processes*. New York: McGraw Hill. 1980.

[Sicilia 2002] SICILIA, M. A., et al. *Extending relational data access programming libraries for fuzziness: The fJDBC framework*. In T. Andreasen, A. Motro, H. Christiansen, & H. L. Larsen (Eds.), Proceedings of the Flexible Query Answering Systems International Conference (Lecture Notes in Artificial Intelligence 2522, pp. 314-328). New York: Springer. 2002.

[Smets 1997] SMETS, P. *Imperfect information: Imprecision-uncertainty*. In A. Motro, & P. Smets (Eds.), *Uncertainty management in information systems: From needs to solutions* (pp. 225-254). 1997. Norwell, MA: Kluwer Academic Publishers.

[Spell 2002] SPELL, Brett. *Java: Programujeme profesionálně*. Přeložil Bogdan Kiszka. 1. vyd. Praha: Computer Press, 2002. 1022 s. Přeloženo z Professional Java Programming. ISBN 80-7226-667-5.

[Tarr 1995] TARR, C. *Identity indirection design pattern*. In Proceeding of the OOPSLA ’95 workshop on design patterns for concurent, parallel, and distributed object-oriented systems. 1995.

[Tré 2000] TRÉ, G., DE CALUWE, R., VAN DER CRUYSSEN, B. A generalised object oriented database model. In *G. Bordogna, G. Pasi (Eds.), Recent issues on fuzzy databases*. 1. vyd. Heidelberg: Physica-Verlag, 2000. 236 s. ISBN 978-3790813197.

[Vaníček 2006] VANÍČEK, J. *Software quality requirements*. In *Agricultural Economics*, 2006, vol. 52, s. 177-185. ISSN 0139-570X.

[Vaníček 2007] VANÍČEK, J. *Kvalita informačních systémů z pohledu mezinárodní normalizace (aktuální informace)*. In *Sborník příspěvků z konference Systém řízení kvality a bezpečnosti informačních systémů ve zdravotnictví.* ČZU. 1. vyd. Praha: Provozně ekonomická fakulta ČZU v Praze, 2007. s. 5-18. ISBN 978-80-213-1720-8.

[Voříšek 1997] VOŘÍŠEK, J. *Strategické řízení informačních systémů a systémové integrace*. Praha: Management Press, 1997. ISBN 80-85943-40-9.

[Wong 1982] WONG, E. A. Statistical approach to incomplete information in database systems. In *ACM Transactions on Database Systems*, 1982, vol. 7, no. 3, s. 479-488. ISSN 0362-5915.

[Zadeh 1965] ZADEH, L. *Fuzzy Sets*. Information and Control. 8. pp. 338-353, 1965.

On-line zdroje

[AT&T 1995] AT&T Bell Laboratories. *Fault-Tolerant Telecommunication System Patterns* [online]. poslední aktualizace 1995-03-24 [cit. 2007-08-21]. Dostupné z: <http://users.rcn.com/jcoplien/Patterns/PLoP95_telecom.html>.

[Barry 2004] BARRY&ASSOCIATES. *Object-oriented database product vendors* [online]. poslední aktualizace 2004-01-10 [cit. 2004-01-11]. Dostupné z: <http://www.service-architecture.com/products/object-oriented_databases.html>.

[Bloomberg 2003] BLOOMBERG, J. *Principles of SOA. Application Development Trends*. [online] poslední aktualizace 2003-02-28 [cit. 2006-08-06].
Dostupné z: <http://www.adtmag.com/article.aspx?id=7345>.

[CMMI 2006] CARNEGIE MELLON UNIVERSITY. *CMMI Web Site* [online]. c2006, poslední aktualizace 2006-05-09 [cit. 2006-05-09]. Dostupné z: <http://www.sei.cmu.edu/cmmi/>.

[Cunningham 1994] CUNNINGHAM, W. *The CHECK - Pattern Language of Information Integrity* [online]. poslední aktualizace 1994-01-10 [cit. 2007-08-21]. Dostupné z: <http://c2.com/ppr/checks.html>.

[db4objects] db4objects, Inc. *db4o :: Native Java & .NET Object Database :: Open Source* [online]. c2008, poslední aktualizace 2007-11-05 [cit. 2007-11-05]. Dostupné z: <http://www.db4o.com/>.

[Fowler 2005] MARTIN, F. *Patterns in Enterprise Software* [online]. poslední aktualizace 2005-02-19 [cit. 2007-11-30]. Dostupné z: <http://martinfowler.com/articles/enterprisePatterns.html>.

[Fowler 2006] MARTIN, F. *Writing Software Patterns* [online]. poslední aktualizace 2006-08-01 [cit. 2007-08-20]. Dostupné z: <http://martinfowler.com/articles/writingPatterns.html>.

[Gemstone 2006] GEMSTONE SYSTEMS. *GemStone Smalltalk Object Server*. [online], poslední aktualizace 2006-02-07 [cit. 2006-02-07]. Dostupné z: <http://www.gemstone.com/products/smalltalk/>.

[IEEE] IEEE. *SEVOCAB: Software and Systems Engineering Vocabulary* [online]. poslední aktualizace 2007-10-10 [cit. 2007-10-10]. Dostupné z: <http://comrie.computer.org/sev_display/index.action>.

[ISO/IEC 14882] American National Standards Institute. *ISO/IEC 14882: Programming languages — C++* [online]. 2. vyd. poslední aktualizace 2003-10-15 [cit. 2007-08-18]. Dostupné z: < http://www.usatlas.bnl.gov/~dladams/cpp/INCITS+ISO+IEC+14882-2003.pdf >.

[Microsoft 2003] MICROSOFT. MSDN CZ: Brožury s vývojářskou tématikou [online]. c2003, poslední aktualizace 2003-03-07 [cit. 2003-03-07]. Dostupné z: <http://www.microsoft.cz/akce/msdn_brozury/>.

[MS Data] Microsoft. *Data patterns* [online]. poslední aktualizace 2007-12-11 [cit. 2007-12-11]. Dostupné z: <http://msdn2.microsoft.com/en-us/library/ms998446.aspx>.

[ODBMS.ORG] ODBMS.ORG. *Object Database Management Systems* [online]. c2008, poslední aktualizace 2008-01-02 [cit. 2008-01-02]. Dostupné z: <http://www.odbms.org/>.

[OMG 2006] THE OBJECT MANAGEMENT GROUP. *Catalog of OMG Modeling and Metadata Specifications* [online]. c2006, poslední aktualizace 2006-08-28 [cit. 2006-08-28]. Dostupné z: <http://www.omg.org/technology/documents/modeling_spec_catalog.htm>.

[OMG XMI] THE OBJECT MANAGEMENT GROUP. *XML Metadata Interchange (XMI)* [online]. c2007, poslední aktualizace 2007-12-12 [cit. 2007-12-18]. Dostupné z: < http://www.omg.org/technology/documents/formal/xmi.htm>.

[Progress 2006] PROGRESS Software. *Progress Real Time Division: Progress ObjectStore Enterprise* [online]. c2003-2006, poslední aktualizace 2006-09-01 [cit. 2006-09-01]. Dostupné z: <http://www.objectstore.com/>.

[Scienta 2004] SCIENTA INTELLIGENCE. *FuzzySQL* [online]. c2004, poslední aktualizace 2004-05-12 [cit. 2004-05-31]. Dostupné z: <http://scianta.com/products/fuzzysql.htm>.

[SEI] CARNEGIE MELLON UNIVERSITY. *Software Engineering Institute (SEI)* [online]. c2006, poslední aktualizace 2006-07-10 [cit. 2006-07-10]. Dostupné z: <http://www.sei.cmu.edu/sei-home.html>.

[Stephnes 2004] STEPHENS, Scott. *Go fuzzy with Oracle* [online]. c1995 2004, poslední aktualizace 2004-05-17 [cit. 2004-05-31]. Dostupné z: <http://asia.cnet.com/builder/architect/db/0,39009328,39179658,00.htm>.

[Sun 2004] SUN MICROSYSTEMS. *Design Patterns: Data Access Object* [online]. c1994 2004, poslední aktualizace 2004-01-07 [cit. 2004-01-07]. Dostupné z: <http://java.sun.com/blueprints/patterns/DAO.html>.

[Sun 2006] SUN MICROSYSTEMS. *Java Technology: Sun Developer Networks (SDN)* [online]. c1994-2006, poslední aktualizace 2006-09-01 [cit. 2006-09-01]. Dostupné z: <http://java.sun.com>.

[Versant] Versant Corp. *Versant Object Store* [online]. poslední aktualizace 2007-11-18 [cit. 2007-11-18]. Dostupné z: <http://www.versant.com/developer>.

[Visnick 2003] VISNICK, L. *Clustering techniques in ObjectStore*. Technical white paper. Retrieved September 2003 from http://www.objectstore.net

[Voříšek 2003] VOŘÍŠEK, J. *MMDIS Princip multidimezionality a dimenze řešení IS* [online]. [cit. 2006-06-03]. Dostupné z: <http://nb.vse.cz/~vorisek/FILES/IT215_materialy_k_predmetu/05MMDIS-Princip_multidimezionality_a_dimenze_reseni_IS.zip>.

[Wikipedia 2006] WIKIPEDIA PROJECT, *Wikipedia, the free encyclopedia* [online]. c2001-2006, poslední aktualizace 2006-08-11 [cit. 2006-08-11]. Dostupné z: <http://en.wikipedia.org/wiki/Main_Page>.

[Winter] WINTER CORPORATATION. *Winter Corporation VLDB Survey* [online]. c2007, poslední aktualizace 2007-05-01 [cit. 2007-05-01]. Dostupné z: <http://wintercorp.com/VLDB/>.

[Zend 2006] ZEND TECHNOLOGIES. *Zend Technologies - PHP tools for development, protection and scalability of PHP applications* [online]. c1999-2006, poslední aktualizace 2006-08-14 [cit. 2006-08-14]. Dostupné z: <http://www.zend.com>.

11. Přehled publikovaných prací autora

[Volráb 2004a] VOLRÁB, O. Fuzzy logika v databázových systémech. In *Sborník prací z mezinárodní vědecké konference Agrární perspektivy XIII*. ČZU. 1. vyd. Praha: Provozně ekonomická fakulta ČZU v Praze, 2004. s. 551–555. ISBN 80-213-1190-8.

[Volráb 2004b] VOLRÁB, O. Fuzzy přístup v objektové databázi. In *Sborník příspěvků konference OBJEKTY 2004*. VŠB – Technická univerzita Ostrava. 1. vyd. Ostrava, 2004. s. 319–325. ISBN 80-248-0672-X.

[Volráb 2004c] VOLRÁB, O. Objektový přístup při vývoji informačního systému typu Content Management Systém. In *Sborník příspěvků z doktorského semináře*, ČZU, 1. vyd. Praha: Provozně ekonomická fakulta ČZU v Praze, 2004. s. 38–42. ISBN 80-213-1150-9.

[Volráb 2005a] VOLRÁB, O. Aplikace fuzzy logiky v objektové databázi. In *Sborník příspěvků z doktorského semináře*, ČZU, 1. vyd. Praha: Provozně ekonomická fakulta ČZU v Praze, 2005. s. 603–608. ISBN 80-213-1314-5.

[Volráb 2005b] VOLRÁB, O. Softwarové nástroje pro podporu výuky objektově orientovaných databázových systémů. In *Sborník prací z mezinárodní vědecké konference Agrární perspektivy XIV*. ČZU. 1. vyd. Praha: Provozně ekonomická fakulta ČZU v Praze, 2005. s. 695–698. ISBN 80-213-1372-2.

[Volráb 2006a] VOLRÁB, O. Příprava testovacích dat pro objektový databázový systém. In *Sborník příspěvků z doktorské vědecké konference THINK TOGETHER 2006*, ČZU, 1. vyd. Praha: Provozně ekonomická fakulta ČZU v Praze, 2006. s. 472-476. ISBN 80-213-1474-5.

[Volráb 2006b] VOLRÁB, O. Fuzzy logika na úrovni rozhraní pro perzistenci dat. In *Sborník příspěvků jedenáctého ročníku konference Objekty 2006*. ČZU. 1. vyd. Praha: Provozně ekonomická fakulta ČZU v Praze, 2006. s. 299-306. ISBN 80-213-1568-7.

[Volráb 2007a] VOLRÁB, O. Fuzzy model v objektově orientované databázi Versant. In *Sborník prací z mezinárodní vědecké konference Agrární perspektivy XVI*. ČZU. 1. vyd. Praha: Provozně ekonomická fakulta ČZU v Praze, 2007. s. 1801-1808. ISBN 978-80-213-1675-1.

[Volráb 2007b] PERGL, R., VOLRÁB O. Objektově-orientované modelování znalostních informačních systémů v podmínkách neurčitosti. In *Sborník příspěvků z mezinárodní vědecké konference Objekty 2007*. s. 74-83. ISBN 978-80-248-1635-7.

[Volráb 2007c] VOLRÁB, O., PERGL, R. Využití vzoru při návrhu fuzzy databáze v OODBMS. In *Sborník příspěvků z mezinárodní vědecké konference Objekty 2007*. s. 147-157. ISBN 978-80-248-1635-7.

[Volráb 2007d] VOLRÁB, O. Design pattern for fuzzy and object oriented databases. *Scientia Agriculturale Bohemica*. ISSN 1211-3174.

[Volráb 2008] VOLRÁB, O. Objekty v databázi s db4objects. *Linux+*. ISSN 1732-3681. (přijato k publikaci do čísla 04/2008)

[Volráb web] VOLRÁB, O. *Fuzzy OODBMS design pattern* [online]. c2008, poslední aktualizace 2008-01-05 [cit. 2008-01-05]. Dostupné z: <http://volrab.googlepages.com/>.

Přehled obrázků

Přehled tabulek

Příloha A – Grantové projekty

Autor práce se podílel resp. podílí na následujících grantech zaměřených na problematiku objektově orientovaných databází a využití fuzzy přístupu:

IZMAN – Inteligentní nástroje pro hodnocení relevance a strukturování obsahu obecných i specializovaných zdrojů dat a znalostí

Instituce: MŠMT, Národního programu výzkumu II - program 2C - Informační technologie pro znalostní společnost.

Realizace: 2006 – 2010

Podíl na řešení: Spoluřešitel

Náplň: Dílčím cílem projektu je navrhnout integrující framework pro konceptuální modelování a implementaci fuzzy aspektů v databázových systémech a metodiku jeho nasazení.

AMIMADES – Inteligence prostředí pro podporu manažerského rozhodování

Instituce: GAČR, Grantová agentura České republiky

Realizace: 2006 – 2008

Podíl na řešení: Spoluřešitel

Náplň: Dílčím cílem projektu je získání nových poznatků týkajících se modelování neurčitosti v závislosti na charakteru aplikační domény a využití objektového přístupu pro vývoj potřebných bází dat

Softwarová podpora výuky objektově orientovaných databázových systémů

Instituce: Interní grantová agentura PEF ČZU v Praze

Podíl na řešení: Hlavní řešitel

Realizace: 2005

Náplň: Cílem grantového projektu bylo zefektivnění výuky objektově orientovaných databázových systémů prostřednictvím vytvoření vhodných softwarových nástrojů. Jejich uplatnění ve výuce směřovalo zejména do oblasti zjednodušení a urychlení často opakovaných operací nad databázovým schématem.

Využití objektových databází v aplikacích pro výzkum v zemědělství

Instituce: Celouniverzitní interní grantová agentura ČZU v Praze

Podíl na řešení: Spoluřešitel

Realizace: 2005

Náplň: Projekt byl zaměřen na aplikovaný výzkum v oblasti využití databázových systémů založených na objektově orientované architektuře pro biometrická data z oblasti živočišné a rostlinné výroby.

Ostatní spolupráce:

Autor práce je v současné době v kontaktu s Bezpečnostní informační službou ČR ohledně praktického uplatnění výsledků prezentovaných v této disertační práci. Kontaktní osoba: RNDr. Jaroslav Carbol, carbol@bis.cz