VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

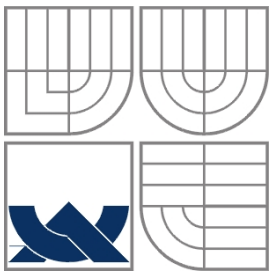
SYSTÉM PRO DOLOVÁNÍ Z DAT V PROSTŘEDÍ ORACLE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

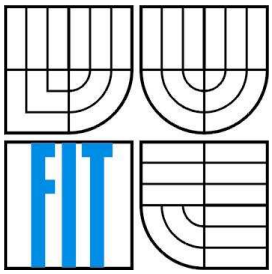
AUTOR PRÁCE
AUTHOR

Bc. MICHAL KRÁSNÝ

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO DOLOVÁNÍ Z DAT V PROSTŘEDÍ ORACLE

DATA MINING SYSTÉM IN ORACLE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MICHAL KRÁSNÝ
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. JAROSLAV ZENDULKA, CSc.
SUPERVISOR

BRNO 2008

Abstrakt

Tento diplomový projekt se zabývá systémem dolování z dat v prostředí Oracle. Jedná se o klientskou aplikaci, využívající služeb serveru Oracle Data Mining Server 10.g Release 2 (10.2). Je naprogramovaná v jazyce Java, grafické uživatelské rozhraní je po staveno na platformě NetBeans.
Teoretická část práce uvede čtenáře do problematiky získávání znalostí, v praktické části je nejdříve shrnuta funkce původní verze systému, jsou popsány jeho nedostatky, zdokumentovány změny, vedoucí k jejich řešení a navržen další rozvoj. Cílem projektu je upravit dolovací systém tak, aby se zvýšila použitelnost aplikace.

Klíčová slova

Získávání znalostí z dat, dolování dat z relačních databází, Oracle Data Mining Server 10.2, DMSL, Java, NetBeans Platform, NetBeans Visual Library, jádro systému dolování dat.

Abstract

This semestral project deals with the system of Knowledge Discovery in Databases. It is a client application which uses the Oracle Data Mining Server's 10.g Release 2 (10.2) services. The application is implemented in Java, the graphical user interface is built on the NetBeans Rich Client Platform.
The theoretical part introduces the Knowledge Discovery in Databases, while the practical part describes functionality of the original system, it's deficiencies, documents sollutions of theese deficiencies and there are proposed improvements for further development. The goal of this project is to modify the system to increase the application usability.

Keywords

Knowledge discovery in data, data mining in relational databases, Oracle Data Mining Server 10.2, DMSL, Java, NetBeans Platform, NetBeans Visual Library, Datamining System Core.

Citace

Krásný Michal: Systém pro dolování z dat v prostředí Oracle. Brno, 2008, diplomová práce, FIT VUT v Brně.

Systém pro dolování z dat v prostředí Oracle

Prohlášení

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

……
Michal Krásný
31.7.2008

Poděkování

Rád bych poděkoval panu Doc. Ing. Jaroslavu Zendulkovi, CSc. za jeho pomoc a podporu při řešení této diplomové práce.

Obsah

1 Úvod

Postupné narůstání objemu dat v databázích a jiných úložištích nám dává možnost tato data zkoumat zpracovávat novými způsoby a získávat z nich různé druhy informací, pro jejichž ukládání nebyl databázový systém původně navržen. Tyto informace jsou nazývány znalosti a proces jejich extrakce je označován pojmem „Získávání znalostí z databází“ (Knowledge Discovery in Databases - KDD). Tento proces se začal rozvíjet koncem 80.let a byl původně vyvíjen pro marketingové účely. Záhy si však našel uplatnění i v ostatních odvětvích, jako je bankovnictví, průmysl, zdravotnictví, sociální výzkumy a podobně. Pojem „dolování z dat“ původně označoval pouze část celého procesu získávání znalostí, nyní se však běžně používá jako synonymum. Stejně tak bude chápán i v následujícím textu.

Konkrétním příkladem znalostí, které se dají z dat získat, může být například pravděpodobnost, s jakou bude klient banky řádně splácet úvěr. Mohou to být také charakteristiky, podle kterých lze od sebe odlišit různé sociální skupiny, nebo to mohou být pravidla, která říkají, že k nějakému druhu koupeného zboží si zákazníci obvykle koupí jiný druh zboží. Pracovník banky se na základě takovýchto informací může rozhodnout, zda poskytnout klientovi půjčku. Sociologická data najdou svá uplatnění ve výzkumu a znalosti potřeb zákazníků pomohou například prodejci počítačů sestavit takové modely, které budou žádanější a tím i konkurenceschopnější.

Nástroje, sloužící k analýze dat, se dělí na dvě hlavní skupiny:

Online Analytical Processing (OLAP)

Tyto techniky využívají datových skladů, které poskytují různé statistické údaje. Analýza probíhá z velké části interaktivně, většinu rozhodnutí provádí člověk. Systémy OLAP jsou vhodné pro zjišťování agregovaných údajů, ale získat komplexní znalosti může být pracné a časově náročné.

Prostředky pro dolování z dat

Slouží k automatizovanému získávání znalostí z dat, většinou určitým způsobem spolupracují s databázovými servery, datovými sklady, nebo do nich mohou být přímo integrovány. Pomocí algoritmů mohou být automaticky získávány komplexní znalosti. Analytik zadává parametry, rozhoduje o kvalitě výsledků, ale samotné dolování provádí počítač.

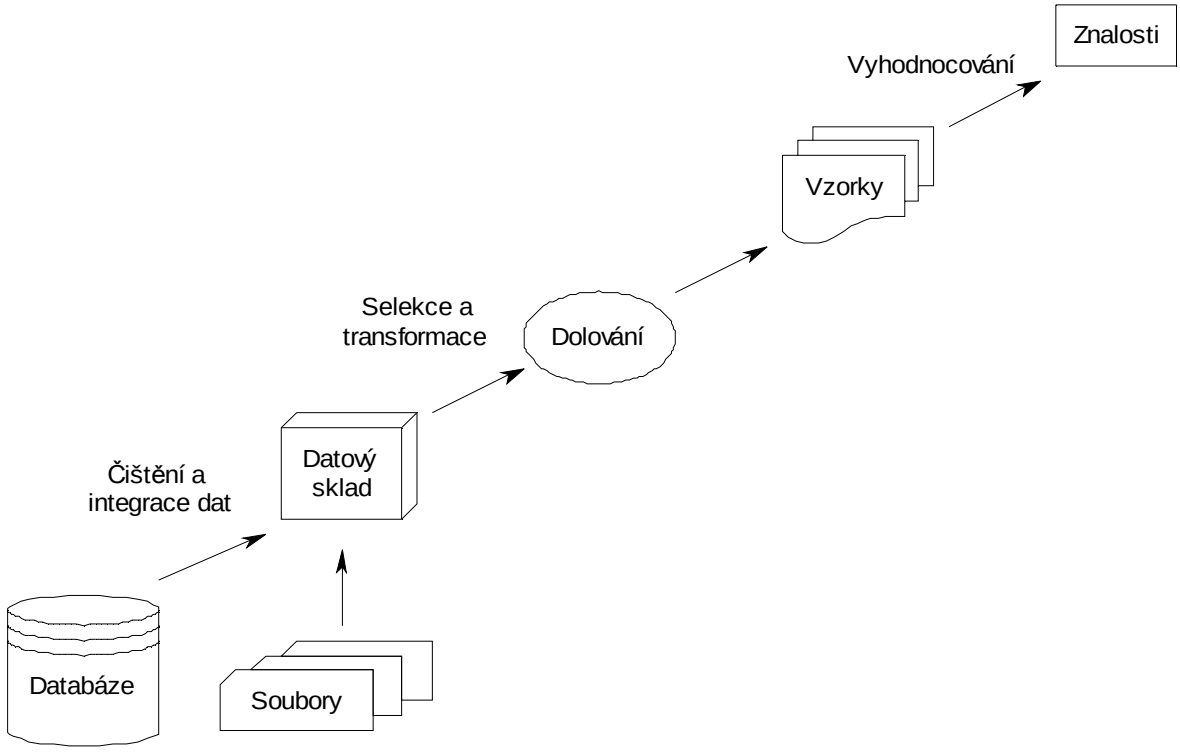
Tato práce se zabývá systémem pro dolování z dat, který umožňuje definici dat, jejich zpracování, přípravu, samotné dolování a prezentaci výsledků. Je pokračováním prací Ing. Doležala [3] a Ing. Gáleta [4]. Výsledná aplikace je zaměřena na definici a přípravu dat, dolování a prezentaci výsledků budou zajišťovat moduly, které budou postupně vznikat později.

Součástí projektu je i návrh a implementace rozhraní, které umožní připojení těchto modulů. Podporu systému zajišťuje databázový server Oracle.

Text je organizován do 9. kapitol. Ve druhé kapitole je čtenář seznámen se samotným procesem dolování dat, třetí kapitola shrnuje použité technologie s důrazem na jazyk DMSL. Ve čtvrté kapitole je popsán stav vyvíjeného systému před tímto projektem. Detailněji jsou rozebírány ty části, kterých se bude týkat další vývoj. Pátá kapitola rozebírá nedostatky původního řešení, navrhuje změny a odůvodňuje je. Šestá kapitola dokumentuje implementaci provedených rozšíření a změn systému. V sedmé kapitole je popsáno grafické uživatelské rozhraní nových komponent. Průběžně uživatele vede k tomu, aby si mohl sestavit a spustit vlastní proces přípravy dat pro dolovací úlohu. Osmá kapitola je návod na vytvoření nového modulu a jeho integraci do aplikace. Bude sloužit budoucím vývojářům modulů. Devátá kapitola shrnuje dosavadní vývoj a navrhuje další rozvoj projektu.

2 Získávání znalostí z dat

Proces získávání znalostí z dat se skládá z více kroků. Nejprve je třeba data připravit a vyčistit, potom najít znalosti a z nich vybrat jen ty důležité a zajímavé. V této kapitole jsem vycházel z [1] a [2].



Obr. 2.1 – Proces dolování dat

Příprava a čištění dat

Surová data mohou obsahovat chyby, mohou být neúplná nebo mohou být nekonzistentní. Zdroje dat je tedy nutné projít a rozhodnout, jak opravit případné nesrovnalosti. Chybějící data se dají odhadnout, nebo doplnit z jiných zdrojů, v případě nekonzistentních dat můžeme tyto položky vyloučit. Ve výsledné statistice se projeví minimálně. Čištění dat nemusíme provádět pouze před integrací do datového skladu, může být součástí prováděných transformací.

Integrace dat

Data nemusíme dolovat pouze z jedné databáze, můžeme mít více zdrojů – například více databází, více souborů. Může být výhodné data organizovat do datových skladů. Ty poskytují mnohem širší

služby než databáze a bývají od provozních databází odděleny. Velmi často se na datový sklad díváme jako na nejnižší komponentu procesu dolování.

Selekce dat

V tomto kroku se pro operaci dolování vyberou jen ta data, která jsou pro naši úlohu relevantní. V praxi to může znamenat například výběr pouze určitých sloupců z databázové tabulky nebo řádků, které mají nějakou vlastnost.

Transformace dat

Pro dolovací úlohy je třeba data transformovat do vhodného tvaru. Některé algoritmy mohou například vyžadovat normalizované vstupy, může být potřeba zahladit extrémní hodnoty nebo převést data do tvaru, který dolovací úloha vyžaduje.

Dolování

V tomto časově náročném kroku získáváme z dat tzv. vzorky. Jsou to analytikem neohodnocené výstupy dolovacích algoritmů, které je potřeba vyhodnotit a vybrat z nich jen ty zajímavé a užitečné. Dolování může být částečně interaktivní a analytik může mít možnost proces nějak řídit nebo usměrňovat. Dolování bývá iterativní proces, kdy na základě výstupů uživatel pozmění parametry úlohy a spustí proces znovu.

Vyhodnocení výsledků

Čím je na vstupu větší objem dat, tím větší množství vzorků získáme. Je proto nutné z nich vybrat pouze ty, které jsou pro nás nějakým způsobem zajímavé nebo užitečné.

Pochopení a prezentace výsledků

Velmi důležitou částí dolování je pochopení a prezentace získaných znalostí. Výsledků dolování využívají často lidé bez matematického nebo technického vzdělání. Znalosti je proto nutné vhodným způsobem vizualizovat.

2.1 Příprava a předzpracování dat

Surová data v databázích nemusí být vždy vhodná pro zpracování a musí být upravena. Obvykle obsahují více informací, než je pro naši úlohu potřeba, mohou být ve tvaru, který se pro dolovací úlohu nehodí a vyskytují se v nich hodnoty, které negativně ovlivňují kvalitu získaných znalostí. Takovým hodnotám říkáme extrémní, neboli odlehlé.

Následující podkapitoly popisují některé metody, které k přípravě a předzpracování dat slouží.

2.1.1 Čištění

Řeší problémy, kdy ve vstupních datech máme chybějící hodnoty, nesmyslné hodnoty (šum) nebo takové hodnoty, které jsou sice platné, mohou však negativně ovlivnit výsledek dolování.

Jako jednoduchý příklad si můžeme představit problém, kdy chceme zjistit, jaký má přibližně plat běžný pracující občan. Pokud bychom spočítali pouze aritmetický průměr platů všech pracujících, pak nám malá skupina velmi vysokých platů ovlivní celkový výsledek. Tyto vysoké platy jsou příkladem odlehlé hodnoty.

Stejné je to i v problematice dolování. Když neodstraníme odlehlé hodnoty, pak získáme sice matematicky správné výsledky, ty ale nemusí odpovídat faktu, který jsme si přáli zjistit.

2.1.1.1 Chybějící hodnoty

S tímto jevem si můžeme poradit několika způsoby. Nejjednodušším přístupem je **ignorování celé n-tice**. Kromě toho, že přicházíme o ostatní nenulové vstupy, se může stát, že atributy s nulovou hodnotou mají mezi sebou nějakou souvislost, například pocházejí ze stejného zdroje. Ignorováním celých n-tic bychom tak mohli přijít o nějakou zajímavou informaci.

Dalším triviálním způsobem nulových hodnot je **manuální doplnění**. Uživatel může data projít ručně a doplnit hodnoty na základě svých znalostí. Tento přístup je ovšem na příliš velké množství dat nepoužitelný.

Zajímavějším přístupem je nějaký druh **automatické náhrady**:

Globální konstantou

Například databázové hodnoty NULL můžeme nahradit číslem 0. Tato nová hodnota musí být volena tak, aby neměla nepříznivý dopad na výsledky.

Průměrnou hodnotou

Jde o obyčejný aritmetický průměr všech nechybějících atributů.

Průměrnou hodnotou n-tic patřících do téže třídy

V praxi to může znamenat, že chybějící cena výrobku bude doplněna průměrnou hodnotou výrobků téhož typu.

Nejpravděpodobnější hodnotou

Tato hodnota se odhadne podle ostatních nenulových atributů. Jde vlastně o aplikaci dolovací úlohy při přípravě dat (klasifikace, regrese...).

2.1.1.2 Šum v datech

Jedná se o druh náhodných chyb, mohou být způsobeny poruchou zařízení nebo lidskou chybou. Například lékař do formuláře zadá teplotu pacienta 3.72 místo 37.2.

Plnění (binning)

Metoda provádí lokální vyhlazování setříděných numerických data. Vstupy se setřídí do tzv. „košů“ tak, že každý z nich obsahuje přibližně stejný počet hodnot. Následně se každá z těchto hodnot nahradí průměrnou hodnotou koše, mediánem nebo nejbližší krajní hodnotou koše.

Regrese

Data jsou nahrazena hodnotami, danými regresní křivkou.

Shlukování

Tato metoda je sama o sobě dolovací úlohou, může však posloužit i k detekci odlehlých hodnot. Hodnoty, které po aplikaci shlukové analýzy nepatří do žádného shluku můžeme prohlásit za odlehlé. Blíže je tato metoda popsána v kapitole 2.2.3.

2.1.2 Integrace

Data z více zdrojů je před samotným dolováním potřeba integrovat do jednotného úložiště. Přitom může dojít k nejednotnosti. Identifikátor v jednom zdroji se může jmenovat *cust_id*, zatímco v jiném zdroji by to bylo *idZakaznika*.

Konflikty schématu

Máme-li data z více zdrojů, může se stát, že v jednom zdroji může být stejná informace vyjádřena jiným schématem. Například adresa se v jedné databázi může skládat z atributů ulice, číslo a město, zatímco v druhé databázi to může být jeden celistvý atribut.

K řešení konfliktů schématu je potřeba mít přístup k metadatům

Konflikty hodnot

Příkladem takového konfliktu jsou různé formáty (datumů), různé stupnice a jednotky (známky, teplota...), různé konvence a podobně. Řešení těchto konfliktů vyžaduje transformaci do jednotné formy.

Konflikty identifikace

Stejný prvek může být ve dvou různých úložištích být identifikován rozdílně nebo různé prvky mohou být identifikovány stejně. Přiřazujeme–li například zaměstnancům identifikátor, unikátní pouze pro jednu pobočku, pak při sloučení dat dojde k porušení této unikátnosti.

Konflikty redundance

V databázích, zejména těch hůře navržených, se mohou vyskytovat odvozené atributy nebo jinak redundantní data. Například z roku narození lze odvodit věk nebo z města bydliště můžeme odvodit kraj. Takovéto redundance jsou nežádoucí a našim cílem je odstranit je.

Příkladem ze skutečné dolovací úlohy je problém, kdy asociační analýza v datech pojišťovací společnosti našla pravidla se 100% spolehlivostí, která říkala, že pokud má klient uzavřenou více než jednu pojistku (číselný atribut), pak má uzavřenou nějakou pojistku (atribut true/false).

2.1.3 Transformace

Transformací připravujeme data do nějaké vhodnější reprezentace. Bývá nástrojem, používaným při integraci dat.

- **Normalizace** – transformace dat do vhodnějšího intervalu, obvykle <-1,1> nebo <0,1>.
- **Odvození dat** – z několika zdrojových sloupců odvodíme jeden. Například z atributů *mzda* a *prémie* můžeme vytvořit atribut *celková odměna*.
- **Agregace dat** – sumarizace dat. Například z předchozího příkladu použijeme atribut *celková odměna* a vypočteme z něj atribut *roční mzda*. Rozdíl mezi agregací a odvozením je ten, že při agregaci sumarizujeme data v nějakém rozměru (čas, pobočka), zatímco odvození pouze sloučí více atributů do jednoho.
- **Vyhlazení** – jedná se o odstranění šumu, viz kapitola 2.1.1.2.
- **Generalizace** – zdrojová data jsou nahrazena koncepty z konceptuální hierarchie. Například kategorický atribut *ulice* mohou být na vyšší úrovni nahrazeny názvem města nebo kraje. Cit [2].

2.1.4 Redukce

Dolování je výpočetně náročné zejména kvůli velkému množství vstupních dat. Cílem redukce je tyto vstupy omezit tak, aby toto omezení mělo minimální dopad na kvalitu výsledků.

- **Diskretizace** – je metoda, která mapuje souvislé hodnoty do diskrétních intervalů.
- **Agregace datové kostky** – typická operace pro datové sklady.
- **Výběr podmnožiny atributů** – výběr pouze těch atributů, které budou pro dolování potřeba.
- **Redukce dimenzionality** – jde o způsob komprese dat. Dojde k překódování dat do takového formátu, který je redukovaný, umožňuje nám však stále provádět potřebné operace.
- **Redukce počtu hodnot** – možné hodnoty se nahradí nějakým redukovaným modelem.

2.2 Typy dolovacích úloh

Dolovací úlohy lze rozdělit do dvou kategorií – popisné a prediktivní. Popisné charakterizují vlastnosti dat a prediktivní hledají takové informace, které můžeme využít pro předvídání budoucího chování dat a následně toto chování předvídají..

2.2.1 Charakterizace a diskriminace dat

Charakterizace

Je disciplína, ve která hledá význačné vlastnosti určitých skupin dat, nazývaných třídy. Úlohu lze typicky popsat dotazem nad datovým skladem. Příkladem může být dotaz: „Čím se vyznačují zákazníci, kteří nejvíce utrácejí?“

Diskriminace

Na rozdíl od charakterizace vymezuje jednu třídu od ostatních popisem vlastností, ve kterých se tyto třídy liší. Například můžeme chtít zjistit, v čem se liší dobří klienti od rizikových.

2.2.2 Klasifikace a predikce

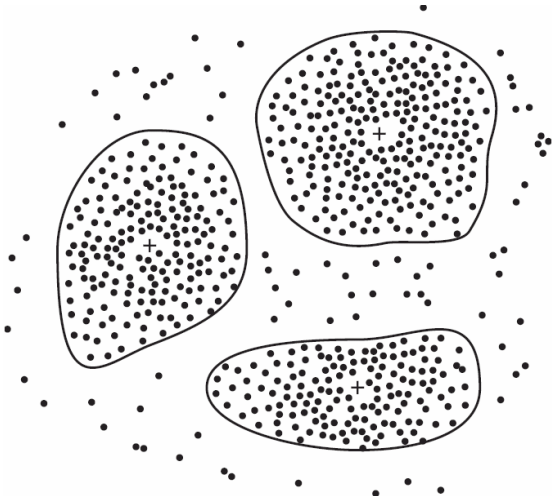
Obě tyto metody jsou prediktivní a předem znají třídy dat, se kterými budou pracovat. Klasifikace nejprve hledá na trénovacích datech model, který přiřadí objekt na základě jeho atributů do některé třídy. Po vytvoření modelu na datech se známou třídou následuje fáze testování, která hodnotí kvality nalezeného modelu. Je-li nalezený model dostatečně kvalitní, mohou být podle něj přiřazovány nové objekty do známých tříd. Tato fáze se nazývá aplikace modelu. Příkladem modelu může být rozhodovací strom nebo neuronová síť.

Jednoduchým příkladem může být banka, která si rozdělí klienty do dvou tříd na spolehlivé a rizikové. Klasifikace nejprve najde, na základě jakých vlastností se tito klienti liší a následně vytvoří rozhodovací strom, který dokáže s určitou přesností rozhodnout, do které skupiny klient patři. Přijde-li do banky nový klient žádat o půjčku, pracovník banky se ho nejprve zeptá na potřebné údaje a na jejich základě pak může stanovit míru rizikovosti úvěru. Tímto dokáže podle předchozích zkušeností předvídat budoucí chování klienta, proto se jedná o prediktivní metodu.

Rozdíl mezi oběma metodami je ten, že klasifikace pracuje s diskrétními daty, zatímco predikce se spojitými hodnotami.

2.2.3 Shluková analýza

Pracuje s předem neznámými daty, která se snaží prvky na základě jejich vlastností seskupit do shluků, neboli tříd. Prvky v jednom shluku jsou si maximálně podobné a co nejvíce se liší od prvků v jiných shlucích. Na rozdíl od klasifikace a predikce se tato úloha neučí na známých datech s pracuje s prvky, které nemají přiřazené třídy, naopak vytvoření a následné přiřazení těchto tříd je cílem.



Obr. 2.2 – Shluková analýza, převzato z [2]

2.2.4 Asociační analýza

Asociační analýza je proces, ve kterém hledáme tzv. asociační pravidla. Jde o vztahy mezi daty, které se dají nejlépe vysvětlit na nákupním košíku. Máme-li seznam nákupů (transakcí), které zákazníci uskutečnili v obchodním domě, pak můžeme hledat vztahy mezi jednotlivými výrobky (položky v transakcích). Hledáme množiny takových výrobků, které zákazníci často kupují dohromady v jednom nákupu (silné množiny). Asociační pravidlo tedy může v běžné řeči znít:

„U zákazníka, který si koupí chléb je 40% pravděpodobnost (spolehlivost), že si koupí sýr. Toto pravidlo se objevuje ve 2% (podpora) všech uskutečněných nákupů.“

Frekventovaná množina prvků

je taková množina, která má vyšší podporu, než je nejnižší námi definovaná podpora. Na příkladu supermarketu by to byla taková skupina výrobků, která se v nákupním košíku objevila dostatečně

často. Nezajímá nás, že 3 zákazníci z celého měsíčního obratu supermarketu měli v košíku totéž zboží, tato skupina má pro nás příliš malou podporu a není tedy frekventovaná.

Silná asociační pravidla

jsou taková, která mají vyšší spolehlivost, než je námi stanovená minimální spolehlivost. Opakem jsou pravidla slabá.

Asociační pravidla se hledají ve dvou krocích:

1. **Nalezení frekventovaných množin** – Nejpoužívanějším algoritmem pro získání těchto množin je Apriori, nebo některá z jeho variant. Tento krok je jádrem celého dolování.
2. **Vygenerování silných asociačních pravidel** – Z frekventovaných množin vygenerujeme všechna asociační pravidla a ponecháme jen silná.

2.2.5 Analýza odlehlých hodnot

Vyhledává neobvyklé, extrémní nebo něčím výjimečné prvky. Příprava dat pro tuto úlohu se liší tím, že při jejich čištění a transformaci nesmí být použity metody, které by odlehlé hodnoty vyhladily nebo je nějakým způsobem poškodila normalizace.

Tato metoda se využívá například při odhalování bankovních podvodů nebo třeba při analýze datových toků, kdy jsme schopni detekovat neobvyklé chování některých uživatelů a snáze odhalit bezpečnostní incident.

2.2.6 Evoluční analýza

Zkoumá opakující se vzory vývoje objektu v čase, intervaly mezi těmito opakováními a rychlost, s jakou ke změnám dochází. Evoluční analýza může být použita například pro odhad vývoje cen na burze.

3 Použité technologie

Tato kapitola popisuje rámcově popisuje použité technologie. Měla by sloužit k základnímu zorientování, podrobné informace lze dohledat v [5][6][7][8][9][10].

3.1 Oracle Data Mining

Celý dolovací systém je postaven na dvouvrstvé architektuře klient-server. Jako server je použit databázový server Oracle s vestavěnou platformou Oracle Data Mining 10.2 [6][7], která slouží jako podpora získávání znalostí. Značení verzí se může zdát poněkud matoucí, protože od verze 10.1 (=10.g) se verze 10.2 zásadně liší v API, které je zcela nekompatibilní. Naopak verze 9.i a 10.g jsou až na drobné rozdíly velice podobné, proto by mohlo dojít ke zmatení programátorů, kteří se snaží v této problematice zorientovat. Firma Oracle již však verze 10.1 a nižší nepodporuje a snaží se je kvůli jednotnosti maximálně omezit. Na internetu lze však najít spoustu odkazů právě na tuto verzi. Od verze 10.2 je rovněž změněn původní název Data Mining Server na Data Mining Engine (dále DME).

DME pracuje tak, že se klient nejprve uloží vstupní data na server a vytvoří a odešle popis dolovací úlohy. Celý další výpočet je plně v režii serveru, po jeho dokončení si pouze klient zažádá o výsledek. Se serverem se dá komunikovat dvěma způsoby, buď pomocí rozhraní PL/SQL nebo pomocí knihoven jazyka Java. V tomto projektu je použit druhý způsob.

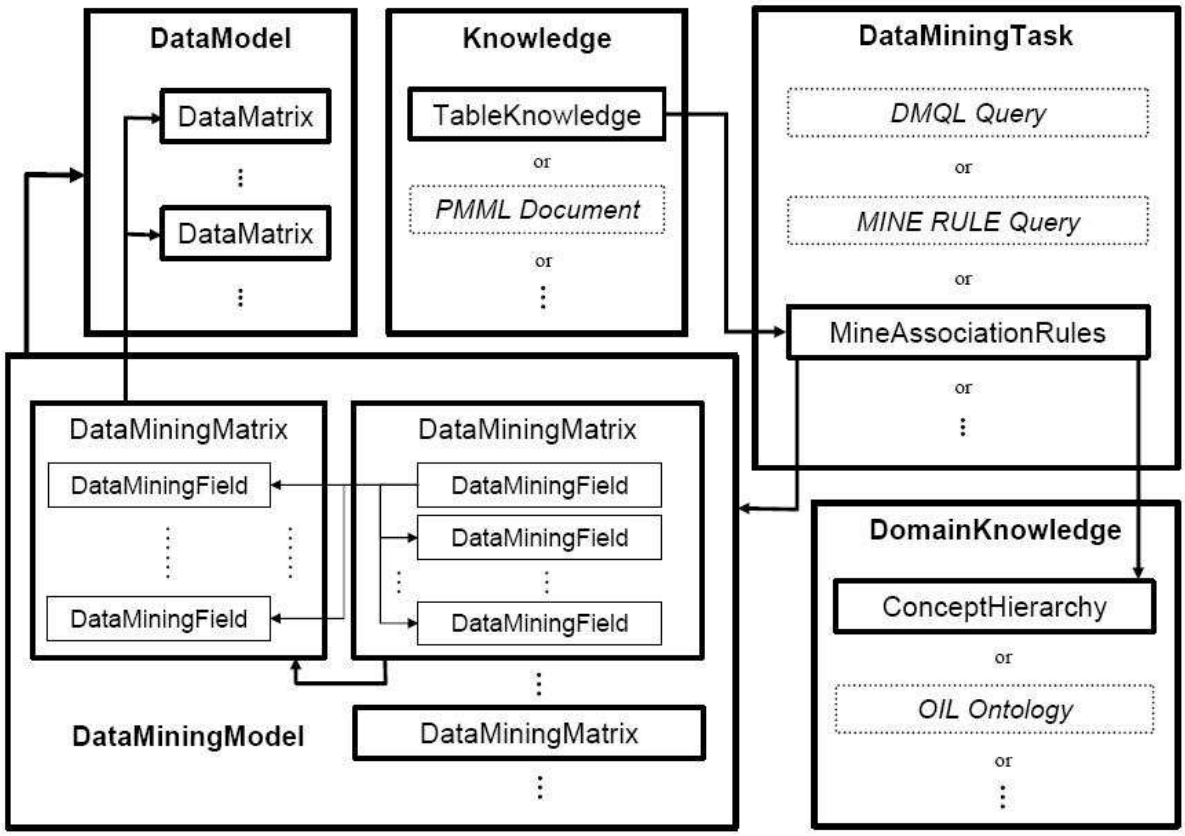
3.2 DMSL

V projektu je k ukládání dat použit jazyk DMSL[5]. Je to je jazyk, založený na XML, který postihuje celý proces dolování dat, od definice vstupních dat, přes jejich transformace, definice dolovacích úloh až po popis získaných znalostí. Je určen k ukládání metadat, nikoliv samotných dat, i když by tímto směrem mohl být rozšířen. Data se nacházejí v databázových tabulkách na serveru, DMSL pouze popisuje kde.

Pro použití v této aplikaci jazyk DMSL byl mírně upraven. Některé jeho části byly modifikovány nebo omezeny a grafická nadstavba, implementovaná panem Gáletem [4] využívá ve značné míře komentářů, kde ukládá informace, potřebné k vizuální reprezentaci celého procesu. V této podkapitole bude popsán obecně podle verze autora [5], změny budou zdokumentovány v následujících kapitolách.

3.2.1 Struktura jazyka

DMSL je jazyk odvozený z XML a stejně jako u XML tvoří jeho obsah tzv. elementy. Abychom mohli popsat celý proces dolování, potřebujeme mít možnost definovat získání dat, popsat transformace, znalosti o vstupních datech (domain knowledge), dolovací úlohu a získané znalosti. Jazyk DMSL má pro každou z těchto pěti částí odpovídající element a navíc jeden element pro definici funkcí, které budou v procesu použity. Vztahy všech elementů jsou vidět na následujícím obrázku:



Obr. 3.1 – Závislosti elementů, převzato z [5]

3.2.2 FunctionPool

Slouží k uchovávání definic funkcí, které mohou být použity při čištění dat a transformacích. Každý FunctionPool má svůj název, podle kterého může být jednoznačně identifikován, na nějž se mohou odkazovat jednotlivé elementy. Kromě funkcí ze samotného FunctionPoolu lze použít i takzvané inline funkce, které jsou definovány přímo v místě použití a nemusí mít název.

3.2.2.1 Function

Tento element definuje samotnou funkci. Funkce jsou děleny podle dvojího pohledu na externí-interní a normální-VIMEO**.** VIMEO funkce jsou takové, které mají jako výstup výčtovou hodnotu Valid, Invalid, Missing, Empty, Outlier. Na každý z těchto výstupů lze reagovat čtyřmi způsoby. Nedělat žádnou změnu (implicitní), ignorovat vstupní hodnotu (celý řádek), nahradit konkrétní pole globální konstantou nebo použít průměrnou hodnotu. Ta se vypočítá z všech odlišně ohodnocených záznamů. Je-li například nějaká položka označena jako Outlier, tak se průměr vypočítá ze všech záznamů jiných, než Outlier.

Kromě těchto druhů funkcí ještě existuje LinearNomalization, která zajišťuje lineární normalizaci. Původní motivace pro přidání lineární normalizace mezi funkce byla snadná dostupnost, že je to jedna z nejpopulárnějších transformací [5].

Interní funkce mají hlavičku a tělo, externí pouze hlavičku. Hlavička funkce poskytuje informace o návratovém typu a je v ní definován seznam a vlastnosti atributů. Na pořadí argumentů záleží, protože se podle něj přiřazují vstupy při volání.

Tělo funkce se skládá z implicitní návratové hodnoty n-tic (Tuple), které seřazeným vstupním argumentům přiřadí hodnoty a pokud vstupy těmto hodnotám odpovídají, je funkce vyhodnocena podle výrazu, který této n-tici odpovídá. Neodpovídá-li žádné n-tice, použije se implicitní návratová hodnota. Hodnoty, které jsou přiřazeny jednotlivým vstupům lze definovat buď výčtově nebo jako interval.

Příklad těla interní funkce:

```
<Tuple>
<Interval defines="interval">
  <RightMargin value="30" closure="closed"/>
</Interval>
<Value value=" Valid "/>
</Tuple>
<Tuple>
<Interval defines="interval">
  <LeftMargin value="100" closure="open"/>
</Interval>
<Value value=" Outlier "/>
</Tuple>
<Value value="Invalid"/>
```

Na příkladu je dobře vidět jak n-tice fungují. Funkce má jeden argument, který je nejprve vyhodnocen podmínkou první n-tice. Je-li vstup menší nebo roven 30, pak je jako návratová hodnota

použit řetězec „Valid“. Pokud ne, je vyhodnocena následující n-tice, která testuje, zda je vstup větší než 100 a pokud ano, pak je vrácena hodnota „Outlier“. Pokud hodnota vstupního argumentu neodpovídá ani této n-tici, pak je vrácena implicitní hodnota „Invalid“. Vstupů může být pochopitelně více než jeden a jako návratová hodnota může být použit výraz.

3.2.3 DataModel

DataModel je element, který slouží definici vstupních dat. Má své jednoznačné jméno, může se odkazovat na nějaký FunctionPool. Skládá se z datových matic.

3.2.3.1 Data Matrix

Datová matice podle původního návrhu DMSL definuje výběr dat z jedné databázové tabulky. Ve které databázi, případně na jakém serveru se tato tabulka nachází, není specifikováno, lze to však jistými prostředky odvodit z kontextu, například to může být na aktuálně připojeném serveru v právě zvolené databázi – záleží na nastavení konkrétní aplikace.

Skládá se z výčtu zvolených datových polí a elementu MatrixTreatment. Datová pole specifikují, jaké sloupce budou z této konkrétní tabulky vybrány a uchovávají informace o jejich vlastnostech (typ, granularita...), zatímco MatrixTreatment umožní nad vybranými sloupci provést VIMEO funkci nebo případně normalizaci.

Jméno tabulky značí vstupní databázovou tabulku před výběrem sloupců a aplikaci VIMEO funkcí.

3.2.4 DataMiningModel

Popisuje které dolovací matice budou použity a v případě, že jich je víc také jakým způsobem budou spojeny do jedné výsledné, jak vzniknou odvozené sloupce a umožňuje na všechny sloupce aplikovat transformace, například lineární normalizaci. Každý DataMiningModel se musí odkazovat právě na jeden DataModel a může se odkazovat na jeden nebo žádný FunctionPool.

3.2.4.1 DataMiningMatrix

Dolovací matice je tabulka, která vznikne z jedné nebo více datových tabulek. Skládá se z komponent DataMiningField. Jsou to ekvivalenty k DataField, které jsou doplněny o definici způsobu, jakým pole vznikne, je-li odvozené.

Jméno DataMiningMatrix označuje databázovou tabulku, která vznikne výběrem sloupců z DataMatrix, případně pomocí dotazu.

3.2.4.2 UseMatrix

Tento element slouží jako odkaz na jednu nebo více tabulek, ze kterých DataMiningMatrix vznikne. Je doplněna elementem Query, který definuje, jakým způsobem má být tabulka vytvořena v případě, že vzniká z více vstupních tabulek.

3.2.5 DomainKnowledge

Jedná se o element s volnou syntaxí, obsahuje libovolné znalosti o vstupních datech. Má povinný pouze jeden atribut, který označuje jazyk, ve kterém je jeho obsah zapsán.

3.2.6 DataMiningTask

Specifikuje dolovací úlohu, stejně jako DomainKnowledge má volnou syntaxi a pouze jeden povinný atribut použitého jazyka. Specifikace DMSL dovoluje ukládat více než jedenu dolovací úlohu, proto mi zde chybí atribut jména, který by od sebe jednotlivé úlohy odděloval a pomocí něhož by se dal provázat například s elementem Knowledge.

3.2.7 Knowledge

Poslední element s volnou syntaxí a jedním povinným atributem „language“.

3.3 NetBeans

Jedná se o open source projekt podporovaný zejména firmou Sun Microsystems. Projekt má dvě hlaví části: vývojové prostředí NetBeans a vývojovou platformu NetBeans (The NetBeans Platform).

Vývojové prostředí je komplexní nástroj pro tvorbu programů. Podporuje i jiné jazyky než Java, např. C++, PHP, Ruby a v Javě jsou to všechny tři hlavní platformy – J2SE, J2EE i J2ME.

Druhým produktem je NetBeans Platform. Jedná se o modulární základ pro vytváření středně velkých a rozsáhlých aplikací. Pro řešení tohoto projektu byla použita NetBeans Platform.

3.3.1 NetBeans Visual Library

Knihovna NetBeans Visual Library slouží jako podpora pro tvorbu aplikací, sloužících k vizualizaci. Původně vznikla z Graph Library, která byla orientovaná na tvorbu grafů. Nyní je integrovaná do prostředí NetBeans Platform 6.0 a sjednocuje uživatelské rozhraní aplikacích, postavených na této platformě.

3.3.2 System FileSystem

Jde o virtuální systém souborů, který slouží k připojování modulů do projektu. Každý modul si vytvoří svůj adresář a virtuální adresář, do kterého ukládá data – instance soubory. Ostatní moduly do něj vkládají připojovací údaje a dochází tak k tzv. volné vazbě.

K editaci virtuálního adresáře slouží layer.xml soubory. Každý modul edituje svůj vlastní soubor, který se virtuálně spojí s ostatními. Připojený modul může tedy vkládat a editovat svá data, cizí data pouze čte. Podrobnější informace viz [4].

4 Systém pro dolování z dat vyvíjený na FIT

V současné době je na VUT FIT vyvíjen systém, který slouží k definici a přípravě dat a umožňuje připojení modulů, které zajišťují samotné dolování a prezentaci výsledků. Klient je aplikace, implementovaná v jazyce Java, která se připojuje k serveru Oracle. Definici dat zajišťuje klient, jejich příprava a transformace jsou rozděleny mezi obě vrstvy, protože Oracle přímo nepodporuje například použití VIMEO funkcí při čištění dat. Samotné dolování bude probíhat přímo na serveru. Tato kapitola poskytuje orientační přehled o původní verzi systému, bližší informace viz [3][4].

Konvence: Zkratkou DMSL bude označován jazyk DMSL. Pojmy Dmsl a Dmsl.java budou označovat třídu cz.vutbr.fit.dataminer.core.dmsl.Dmsl.java.

4.1 Dosavadní vývoj

Konvence: V dalším textu bude třeba odlišovat práci Ing. Doležala a Ing. Gáleta. Pod pojmem „jádro“ bude chápána práce Ing. Doležala v celém jejím rozsahu a pojmem „grafická nadstavba“ bude označována práce Ing. Gáleta. Takto byly oba pojmy používány v obou předchozích pracích. V navazujících projektech, které se budou zabývat moduly, bude zřejmě slovem jádro označován celý tento systém, protože z pohledu modulu skutečně jádrem je. Zde ovšem potřebujeme jemnější dělení, použijeme tedy výše zmiňovanou konvenci.

Konvence: Pojmem komponenta, nebude-li uvedeno jinak, bude myšlen jeden blok diagramu, kterým uživatel definuje proces dolování. Výrazy typu „komponenta provádí“ nebo „komponenta spolupracuje“ označují činnost aplikační logiky, která danému bloku náleží.

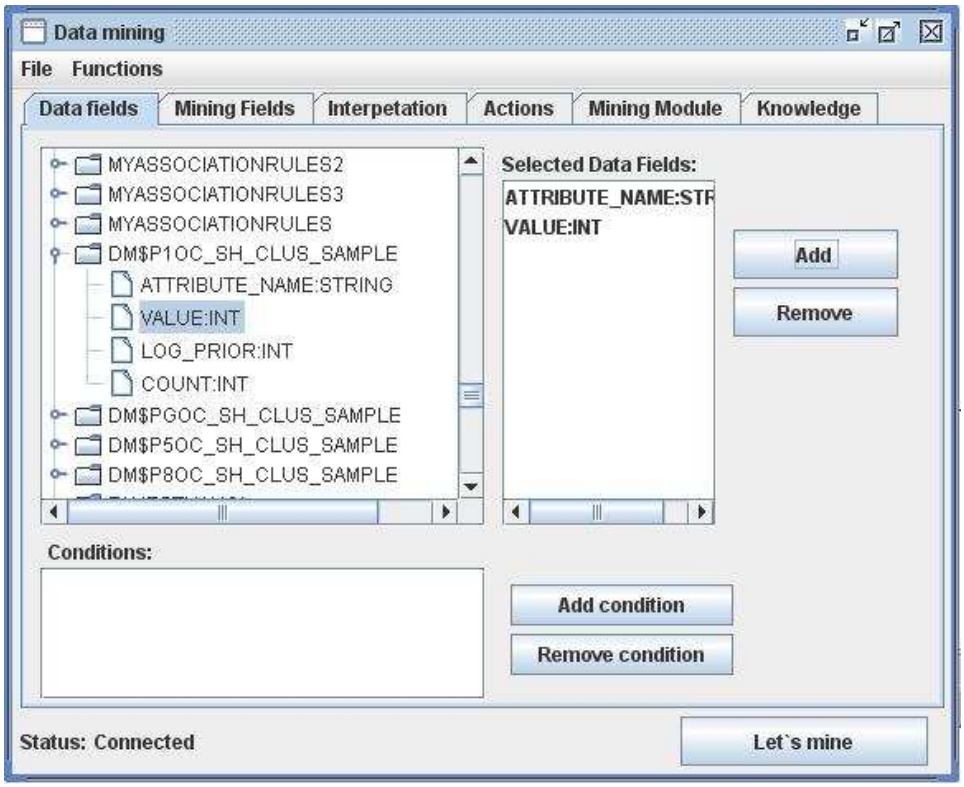
4.1.1 Jádro

Projekt odstartovala práce Ing. Doležala v roce 2006 [3]. Jejím cílem bylo vybudovat jádro systému, které by podporovalo přidávání dolovacích modulů. Toto jádro mělo být základem pro formulářovou aplikaci a počítalo se s tím, že grafická nadstavba by mohla být časem upravena nebo zcela vyměněna.

Systém byl navržen tak, aby připojení modulu nevyžadovalo překlad jádra. Do aplikace se dal připojit v jednom okamžiku pouze jeden dolovací modul a pořadí transformací bylo pevně stanoveno. Nejprve se aplikovaly VIMEO funkce, následovalo odvození sloupců a transformace. Každý krok

bylo možné provést pouze jednou. Rozhraní pro připojení modulů vykazovalo nedostatky, protože například neobsahovalo prostředky pro uložení parametrů modulu.

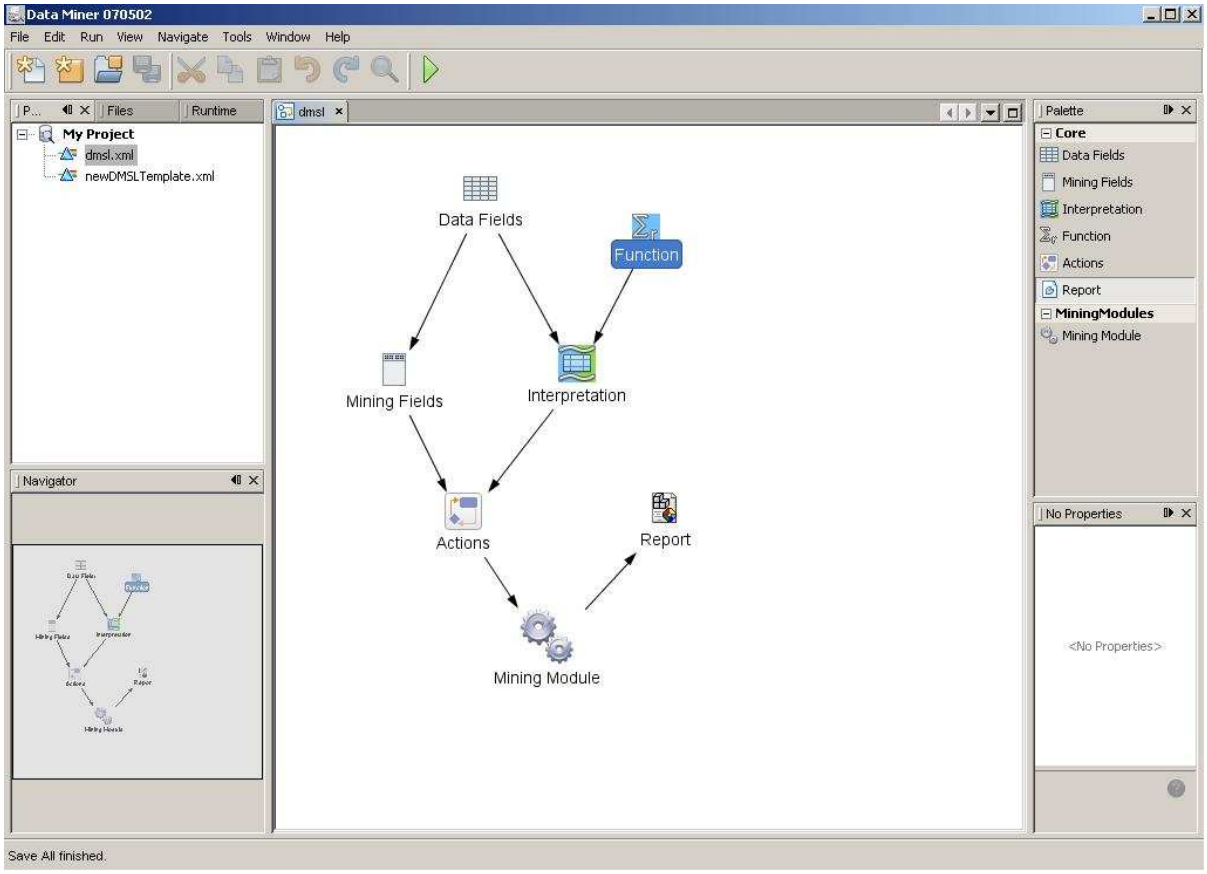
Navzdory výše zmíněným byla práce kvalitní a spoustu problémů řešila velmi dobře. Například funkce, které byly potřeba při čištění dat (VIMEO) a při odvozování nových sloupců nebyly interpretovány přímo aplikací, ale byl z nich vygenerován zdrojový kód v Javě, který byl přeložen kompilátorem a volán jako externí třída. Došlo tak k velmi výrazné úspoře výkonu.



Obr.4.1 – Původní aplikace

4.1.2 Grafická nadstavba

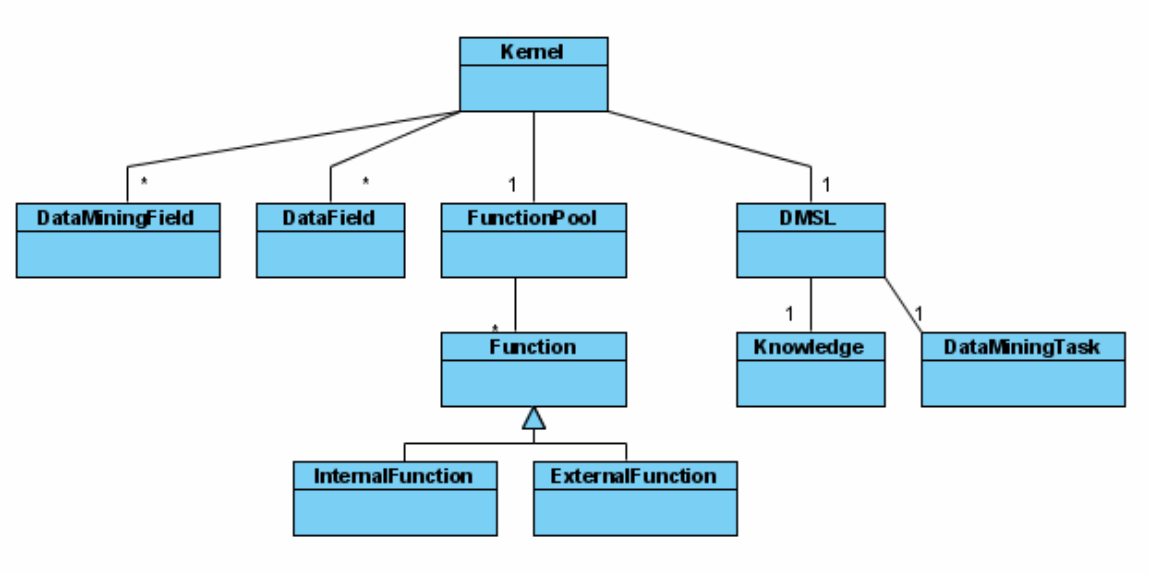
Na tuto práci navázal o rok později svým diplomovým projektem Ing. Gálet. Použil knihovnu NetBeans Visual Library, která je určena k tvorbě grafů a diagramů. Podařilo se mu vybudovat prostředí, které umožňuje vkládat komponenty znázorňující jednotlivé kroky dolování, ty mezi sebou různě propojovat a dal tím základ budoucí aplikace, která má velký potenciál stát se kvalitním komplexním grafickým nástrojem pro dolování dat. Zavedl organizaci do projektů, integroval do DMSL dokumentu grafická metadata ve formě poznámek a nasměroval projekt správným směrem. Byl však omezen přílišnou specifičností jádra a z pohledu dolování nebylo možné přinést další funkčnost.



Obr. 4.2 – Grafická nadstavba Ing. Gáleta

4.2 Architektura původního systému

Jak již bylo v předchozí kapitole zmíněno, systém se děli na jádro a grafickou nadstavbu. Centrální třídou jádra je `cz.vutbr.fit.dataminer.core.system.Kernel.java`, která svazuje a propojuje všechny ostatní třídy. Jak je vidět na následujícím obrázku, Kernel obsahuje datová pole (Arraylist<DataField>), dolovací pole (Arraylist<MiningField>), úložiště funkcí (FunctionPool) a objekt DMSL, který zajišťuje manipulaci s DMSL dokumentem, do kterého se celý projekt ukládá. Myšlenka byla tak, že by pro dobré pochopení Kernel svojí strukturou měl odpovídat DMSL dokumentu, obsahovat volně připojitelný dolovací modul, který by prováděl samotné dolování a instance objektu třídy Dmsl.java by zajišťovala manipulaci se souborem na disku.



.

Obr. 4.3 – Diagram základních tříd

Volání VIMEO funkcí a transformace a všechny parametry těchto operací jsou uloženy v objektu třídy Action.java, který je součástí DataFields i DataMiningFields. U odvozené instance DataMiningField je navíc odkaz na funkci, pomocí které vznikne.

Příprava dat je jednolitý proces. Nejprve se vyberou dolovací pole a společně s aplikací VIMEO funkcí naplní dolovací tabulka s pevně nastaveným názvem „datatomine“. Hodnoty odvozených sloupců se vypočítávají při plnění tabulky. Následně se aplikují transformace v pořadí Clipping, Normalization a Discretization. Po provedení těchto operací se do objektu třídy Dmsl uloží všechny nově nastavené hodnoty, načte se neaktualizovaný element DMSL DataMiningModel (závažná chyba, viz další kapitola), ten se předá dolovacímu modulu, který je spuštěn.

Kromě dolování se moduly starají i o grafickou reprezentaci nalezených znalostí, protože ta je krajně závislá na konkrétním modulu. Každý modul poskytuje svému okolí (mateřské aplikaci) grafický panel, na kterém jsou výsledky vizualizovány.

4.2.1 Zpracování DMSL dokumentu

K tomuto úkolu je použit XML parser, který vytvoří DOM model skládající se z hierarchicky zanořených uzlů (Node). Každý uzel odpovídá XML elementu. Postupným procházením tohoto modelu jsou načteny všechny jeho části a následně naplněny datové struktury. Ty se nacházejí jednak v samotném objektu DMSL a jednak v jádře Kernel. Vždy před spuštěním dolovací úlohy jsou aktualizovány (není však aktualizován DOM model).

Při ukládání do DMSL se vygeneruje z datových struktur XML řetězec, který je zapsán do souboru. Každá datová struktura má k tomuto účelu metodu.

4.2.2 Interface dolovacího modulu

Aby byly různé dolovací moduly kompatibilní se systémem, musí implementovat společné rozhraní, konkrétně `cz.vutbr.fit.dataminer.core.system.MiningModule.java`. Každý modul má povinnost implementovat tyto metody:

- **executeMiningtask** – Spouští samotnou dolovací úlohu. V parametrech předá modulu DMSL element DataMiningModel, podle kterého by měl modul získat informace o vstupních sloupcích. Popis úlohy TaskDefinition předán není, uživatel ho musí po každém spuštění aplikace zadat znovu.
- **getTaskDefinition** – Získá z modulu parametry dolovací úlohy. Ty se mohou lišit a každá dolovací úloha má možnost si je definovat po svém. Jádro se nijak nestará o jejich formát, pouze zajišťuje, aby se po skončení dolovací úlohy dostaly do objektu třídy Dmsl.java a mohly být uloženy.
- **getKnowledge** – Obdobná metoda, jako getTaskDefinition s tím rozdílem, že vrací nalezené znalosti.
- **getParametersGUI** – Touto metodou modul poskytuje svému okolí grafický panel, pomocí kterého lze zadávat vstupy.
- **getKnowledgeGUI** – Totéž jako předchozí metoda, poskytuje panel se zobrazenými výsledky dolování.

4.2.3 Funkce

Funkce v nepřeložené podobě jsou uloženy v instanci třídy FunctionPool. Tam se dostanou při načtení ze souboru nebo při zadání uživatelem. Před jejich použitím v dolovacím procesu je z nich pokaždé vygenerován Java kód, ten je přeložen, importován jako externí třída a při použití jsou volány zkompilované. Jejich kompilace před každým spuštěním procesu zajistí, že budou pořád aktuální. Název souboru, do kterého se generují je pevně daný - CompiledInternalFuncs.java.

K volání slouží třída `cz.vutbr.fit.dataminer.core.system.Caller.java`. Při volání funkce pomocí metody CallVimeo nebo CallMeth je potřeba zadat typy vstupních sloupců a jejich hodnoty. Typy jsou získány z DataFields, konkrétní hodnoty jsou zjištěny SQL dotazem do databáze.

4.2.4 DME transformace

Jsou implementovány ve třech souborech, které se nacházejí v balíčku `cz.vutbr.fit.dataminer.core.system.dme`, konkrétně Clipping.java, Discretization.java a Normalization.java. Každý z nich zajišťuje odpovídající transformace. Vstupní

parametry jsou seznamy dolovacích polí, objekt reprezentující připojení k databázi a název tabulky, ve které jsou uloženy hodnoty dolovacích polí. Výsledky těchto operací bývají uloženy v pohledech. To, ve kterém pohledu je uložen výsledek transformace záleží na typech použitých transformací. Jméno pohledu s transformovanými daty je návratová hodnota metody, která transformaci zajišťuje. Toto jméno je po ukončení použito jako parametr volání následující transformace. K ukládání jména je použita globální proměnná dataToMine, na konci celého procesu přípravy dat je v ní uloženo jméno tabulky s připravenými daty.

4.2.5 Grafická nadstavba

Grafická nadstavba je implementovaná v prostředí NetBeans a postavená na knihovně NetBeans Visual Library. Skládá se z několika modulů spojených pomocí Module Suite. Jádro je do ní integrováno pomocí tzv. wrapperu. Jedná se o modul, který slouží jako rozhraní pro připojení přeložené třídy.

Aplikace umožňuje organizaci práce do projektů. Každý projekt může obsahovat více DMSL dokumentů. Pro každý z těchto dokumentů se vytvoří zvláštní instance třídy Kernel.java a pro grafická data instance třídy DMSLDataObject. DMSL data jsou načítány parserem implementovaným v jádře ve třídě Dmsl.java. Grafická data jsou při ukládání nejprve uložena do pomocné struktury DMSLMetadata a následně jsou exportována do XML pomocí knihovny XStream. Načítání probíhá stejným způsobem, ale v opačném pořadí. Pro knihovnu XStream je použit wrapper podobně jako tomu bylo u jádra.

Připojení je definované pro každý projekt zvlášť. Jádro rozeznává dva druhy připojení, jedno pro SQL dotazy (JDBC), druhé pro zadávání dolovacích úloh (OraConnection). První připojení je navázáno pomocí ConnectionManageru, druhé je napevno nastaveno ve zdrojových kódech jádra.

Jednotlivé grafické komponenty jsou implementovány tak, že po poklikání na ikonku grafu otevřou dialog, do kterého vloží panel z původní formulářové aplikace.

Celá koncepce původního systému vylučuje vkládání více komponent stejného typu, protože by odkazovaly na tentýž formulář a tím i na stejná data. Omezení na vkládání více komponent stejného typu je implementováno tak, že se jim generují stejné ID (viz kap 4.2.6) a třída NodeAcceptProvider více komponent stejného typu odmítne.

Bližší informace viz [4].

4.2.6 Abstraktní třída MiningPiece

Aby mohla být komponenta grafu vložena do scény, musí dědit z abstraktní třídy MiningPiece.java. Důležité metody:

areRequieredConnectionsSatisfied

Zde může tvůrce nastavit vlastní podmínky, za kterých povolí otevření komponenty.

openDialog

Volá se po poklikání na komponentu. Pomocí metody areRequieredConnectionsSatisfied se otestuje, zda jsou splněny podmínky pro otevření dialogu a pokud ano, zavolá metodu openCustomizingDialog.

openCustomizingDialog

Abstraktní metoda, kterou implementuje tvůrce komponenty. Obvykle otevírá dialog, ale může v ní být teoreticky jakákoliv reakce.

beforeAcceptEvent

Je volána před přijetím uzlu do grafu. Může otevřít dialog pro komunikaci s uživatelem a případně proces vkládání zastavit.

beforeRemoveEvent

Je volána před odebráním uzlu z grafu. Může otevřít dialog pro komunikaci s uživatelem a případně proces odebírání zastavit.

Ostatní metody jsou popsány v API dokumentaci [4] na přiloženém CD.

5 Nedostatky stávajícího řešení a návrh jejich úprav

Původní jádro vzniklo jako samostatný projekt a ze způsobu jeho implementace je patrné, že při jeho tvorbě se nepočítalo s rozšířením za hranice formulářové aplikace. Grafická nadstavba nabízí obrovský potenciál, ve výsledku je to však stále jen vylepšená formulářová aplikace. Tato kapitola obsahuje návrh na změny a rozšíření, které by měly projekt nasměrovat směrem k plnému využití potenciálu, který grafická nadstavba nabízí. Částečně vychází z připomínek Ing. Gáleta v jeho práci[4] v kapitole 9.1. Při návrhu změn bude kladen důraz na modifikovatelnost, pochopitelnost a další rozšiřitelnost.

5.1 Komponenty a práce s grafem

Hlavním cílem tohoto projektu bude umožnit, aby se aplikace naplno využila potenciál definování dolovacího procesu pomocí grafů. Jednotlivé komponenty budou mít možnost být zapojeny v jakémkoliv smysluplném pořadí, budou se moci opakovat a budou moci být různě odpojovány a připojovány do dolovacího procesu. Celý dolovací proces bude ve formě stromu. Jedna komponenta tedy bude mít možnost své výstupy přesměrovat do libovolného počtu podstromů a uživatel bude mít možnost vybrat, kterou větev spustí. Po odpojení podstromu by měla existovat možnost podstrom připojit do jiné části procesu. S tím souvisí řešení konfliktů dolovacích polí připojené a připojované struktury. Dolovací modul bude možné připojit přímo k datovému zdroji bez použití jakýchkoliv transformací. Modul bude mít podobná omezení na vkládání jako jiné komponenty. Bude jich moci existovat více, uživatel si bude moci vybrat které z nich spustí. Při opakovaném spuštění by se neměla znovu přepočítávat aktuální data. Příprava dat a dolování bývá časově náročný výpočet a přepočítávání aktuálních by bylo velkým plýtváním.

Komponenty budou pojmenovatelné, aby graf zůstal přehledný. Popisky komponent budou odděleny od názvů v DMSL dokumentu, aby nedocházelo ke kolizím jmen, ale zároveň uživatel mohl více komponent pojmenovat stejně.

Celý dolovací projekt by měl být uložitelný v libovolném rozpracovaném stavu, nemusí být tedy validní podle stávajících omezení DMSL. Uživatel bude omezován tak, aby nemohl provést nesmyslnou akci. Celý tento mechanismus omezení by měl být intuitivní a pro uživatele přirozený.

Od základů se změní logika propojování transformačních komponent. Současný systém, kde je nutno do Actions soustředit Mining Fields a Interpretations a složitě přiřazovat funkce, není intuitivní a je velice složité ho pochopit bez manuálu. To může odrazovat budoucí uživatele. Vzniknou nové komponenty, které si rozdělí práci vhodnějším způsobem.

Data Fields, které slouží k výběru dolovacích polí jsou řešeny dobře, není nutné je měnit. Dále vzniknou dvě nové transformační komponenty, které oddělí čištění dat (budoucí název Vimeo) od odvozování sloupců a transformací (Transformations).

Transformace a odvozování sloupcí bude vhodné spojit do jedné komponenty. Na intuitivitě to neubírá a ušetří se tím jeden prvek grafu, takže ten se stane přehlednějším. Bude pevně dané pořadí obou akcí – nejprve se provede odvození sloupců, pak se provedou transformace v pořadí clipping, normalizace a diskretizace. Pokud bude potřeba toto pořadí změnit, lze vždy vložit více komponent za sebe a každé z nich přiřadit pouze jednu akci.

Každá komponenta bude mít přiřazenou databázovou tabulku, která bude jejím výstupem. Změní se tak stávající koncept, kdy existuje jedna společná tabulka, případně jeden společný odvozený pohled.

5.2 Datové struktury

Každá komponenta bude mít svůj vlastní prostor v DMSL dokumentu a vlastní jednotku v paměti tak, aby od ostatních byla oddělena. Pomocí jednoho identifikátoru bude nutné provázat komponentu s databázovou tabulkou, paměťovou jednotkou i odpovídajícím úsekem DMSL dokumentu. Bude tedy změněn současný přístup, který ukládá jeden seznam datových a jeden seznam dolovacích polí bez jakýchkoliv dalších informací. Identifikátory budou generovány systémem, uživatel bude moci se systémem pracovat bez toho, aby o jejich existenci musel vědět, protože ten bude své komponenty odlišovat podle jejich popisek.

Zanikne stávající redundance dat, která jsou uchovávána zároveň ve třídě Kernel.java a zároveň ve třídě Dmsl.java. Všechna data budou ukládána pouze do komponenty Dmsl.java, včetně instance třídy FunctionPool. Jakákoliv datová redundance je nebezpečná, má například za následek chybu v rozhraní MiningModule (popsáno dále). Každý dialog každé komponenty by měl mít svůj vlastní dočasný paměťový prostor, který bude používaný ve chvíli, kdy se s komponentou bude pracovat. V momentě otevření dialogu vznikne a po uzavření dialogu se buď aktualizuje do objektu dmsl (tlačítko OK) nebo zanikne (tlačítko Cancel). V aktuální verzi toto oddělení není, tlačítko Cancel není možné rozumným způsobem implementovat.

Aby bylo možné zvládnout novou potenciálně mohutnou datovou strukturu, bude muset být navržena zcela jiným způsobem. Bude přehledná, modifikovatelná a rozšiřitelná a snadno pochopitelná. Ideální řešení je implementovat ji jako co nejvěrnější kopii DMSL dokumentu. Pak bude mít stejnou vyjadřovací sílu a stejné vlastnosti jako jazyk DMSL a komukoliv, kdo bude projekt modifikovat bude stačit pochopit a nastudovat tento jazyk. Projekt se stane modifikovatelnější, protože k úpravě malé části projektu nebude nutné studovat jiný přístup pro řešení v programu, než je DMSL. Nynější řešení je takové, že každé datové nebo dolovací pole přímo v sobě obsahuje

transformace a funkce. Položky obsahují různé příznaky používání, o které je nutno se starat. Toto všechno lze však zjistit přímo z DMSL struktury. Třída Dmsl.java a její zanořené komponenty budou mít funkce, které usnadní vyhledávání, aby zůstal zachován programátorský komfort, který poskytovaly přímé odkazy pomocí referencí v původním projektu. Například bude existovat metoda, která dokáže najít dolovací matici podle jejího jména.

Paměťovou jednotkou i prostorem v DMSL dokumentu bude pro datové komponenty datová matice (DataMatrix) a pro transformační komponenty dolovací matice (DataMiningMatrix). Stejně tak, jako mají komponenty podobné vlastnosti, tak bude vhodné, aby i jejich paměťové ekvivalenty měly podobné vlastnosti. U DataMiningMatrix je celkem logické, že bude použita jako doplněk k transformačním komponentám, zásadní posun však nastává v chápání významu datového modelu (DataModel) a datových matic (DataMatrix).

V původní specifikaci jazyka DMSL [5] sloužila DataMatrix k výběru dat z jedné databázové tabulky. Atribut name označoval vstupní tabulku. Pokud jsme chtěli vybrat sloupce více tabulek, pak jsme potřebovali více elementů DataMatrix, které se spojily v DataMiningMatrix pomocí Query. U prvku DataMiningMatrix označoval atribut name výstupní tabulku, která vznikne aplikací tohoto elementu.

Z hlediska jednotnosti komponent je to závažný problém. Abychom dostáli původnímu předpokladu, že výstupem každé komponenty bude tabulka, pak by musela být práce s výběrovými komponentami mnohem složitější než práce s transformačními. Každá komponenta pro výběr dat z více tabulek by musela mít možnost data spojit, měla by tedy několik DataMatrix spojených v jedné DataMiningMatrix.

Nabízí se tedy řešení, udělat veškerý výběr dat právě v jedné komponentě DataMatrix. Ujednotí se tím práce s maticemi, protože jak DataMatrix, tak DataMiningMatrix budou mít stejné základní vlastnosti. Vstupem budou sloupce z různých databázových tabulek, výstupem bude jedna tabulka. Atribut name u obou matic bude označovat jméno výstupní tabulky a každé grafové komponentě bude přidělena právě jedna matice. DataMatrix bude rozšířena o možnosti spojování dat z více tabulek, podrobnosti viz kapitola 6.

5.3 Rozhraní pro připojení modulů

Původní rozhraní MiningModule, navržené pro formulářovou aplikaci mělo umožňovat připojit k aplikaci externí modul. Nebylo však funkční, protože mělo následující nedostatky:

- Dolovacímu modulu se předá uzel DataMiningModel z DMSL dokumentu. Ten je však neaktuální, obsahuje totiž data načtená ze souboru při otevření, nikoliv data změněná uživatelem.
- Modul se nedozví, ve které tabulce má data.

- Z modulu je výstup pro parametry dolovací úlohy, rozhraní však neumožňuje je do něj dostat jinak, než přes GUI.
- Do modulu nelze nahrát znalosti. Aby bylo možné zobrazit získané znalosti, je nutné znovu spustit dolování. Lze je sice uložit do souboru, ale nelze je již zpětně prezentovat.
- Celý systém počítá pouze s jedním připojeným modulem.

Rozhraní MiningModule bude zrušeno a místo něj bude rozšířena abstraktní třída MiningPiece, která svým potomkům poskytuje prostředky pro připojení do grafu. Dolovací modul se tedy stane běžnou komponentou a pro práci s ním nebude potřeba žádné zvláštní zacházení.

5.4 Reorganizace třídy Kernel.java

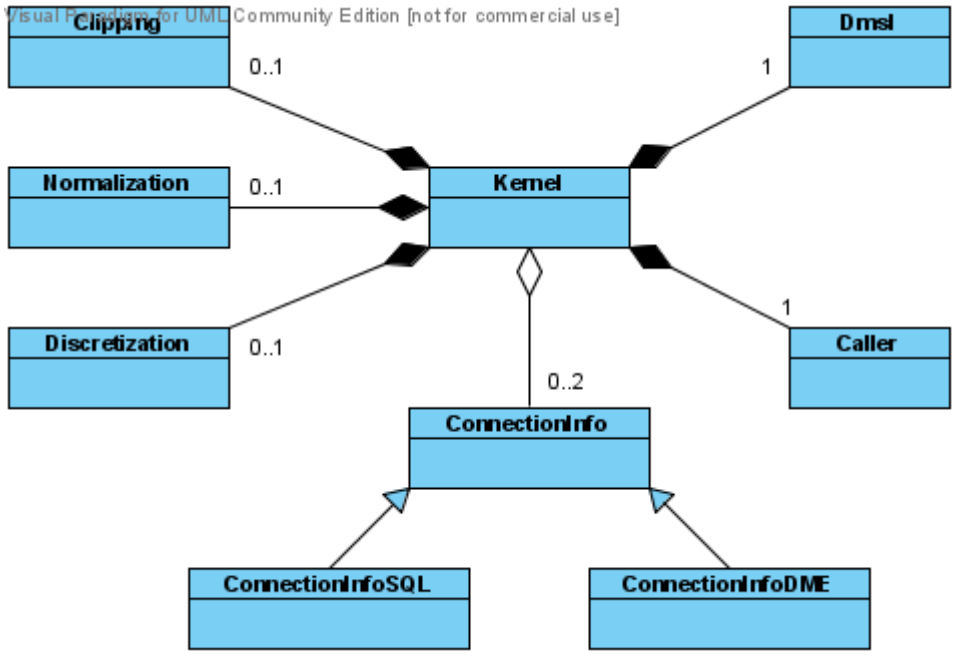
S předchozími změnami souvisí i přestavba třídy Kernel.java. Bude z ní odebrán dolovací modul a metody, které s ním související. FunctionPool bude vnímán jako datová struktura a bude přesunut spolu se všemi ostatními datovými strukturami do Dmsl.java. Spuštění dolovací úlohy bude řešeno jiným způsobem, metoda startProcessing se rozpadne do dílčích metod. Připojení k databázi přestane být napevno zadané ve zdrojovém kódu a bude nastavitelné pomocí GUI. Řešení Ing. Gáleta [4] předpokládá, že každý otevřený DMSL dokument v projektu bude mít svoji instanci třídy Kernel, zruší se tedy návrhový vzor singleton.

6 Popis implementace změn

Tato kapitola dokumentuje konkrétní implementace navrhovaných změn. Motivace k nim jsou uvedeny v kapitole 5.

6.1 Kernel.java

Třída byla upravena podle návrhu v kapitole 5.4. Všechny datové jednotky byly přesunuty do Dmsl.java. Byl odstraněn dolovací modul a instance entit provádějících transformace. Třída má za úkol poskytovat připojení k serveru – jeden typ pro SQL dotazy, jeden pro DME transformace. Vytvoření připojení nezajišťuje Kernel, proto je ve vztahu uvedena agregace. Dále Kernel poskytuje Caller pro volání zkompilovaných interních funkcí a přístup k datové struktuře Dmsl.java. Metoda startProcessing byla nahrazena metodami pro jednotlivé komponenty. Instance Clipping, Normalization a Discretization se vytvářejí při volání metody, obsluhující spuštění běhu komponenty Transformations a zajišťují inicializaci parametrů pro server a jejich odeslání.



Obr.6.1 – Diagram tříd Kernel.java

6.2 Datové struktury

Třída Dmsl.java nyní kromě manipulace se souborem slouží jako centrální úložiště dat. Nedoporučuje se, aby si jakákoliv část programu uchovávala svá data a ta pak musela být aktualizována tak, jak tomu bylo v původním projektu. Jedinou výjimkou jsou datové struktury pro právě otevřené dialogy, které se do Dmsl zapíší v okamžiku stisku tlačítka „OK“ a jim podobné případy.

Skladba celé datové struktury by měla co nejpřesněji odpovídat DMSL dokumentu. Každá DMSL entita má svůj protějšek v datové struktuře. Ve struktuře neexistují agregované položky, příznakové proměnné, ani žádná jiná odvozená data. Vše se vypočítává ve chvíli, kdy je to potřeba. Důvodem k této konvenci je špatná zkušenost s předchozím řešením, kde bylo nutné i pro malou změnu nastudovat velkou část programu a pochopit všechny souvislosti. Programátorský komfort zůstal zachován, protože třídy u kterých to má smysl obsahují metody, které dokáží přistupovat k datům a manipulovat s nimi asociativně.

Konkrétní příklad může být ten, že uživatel smaže MiningField a pomocí dvou metod bude možnos smažat z položky MatrixTreatment všechny VimeoValues a ValueTreatment, které mají jako svůj FieldRef nastavený odkaz právě na mazané MiningField. Výkonnostně to aplikaci nijak neovlivní, předpokládaný počet prohledávaných komponent budou řádově jednotky až desítky.

Každá třída, obsažená v datové struktuře musí implementovat rozhraní DMSLDataStructure a doporučuje se, aby implementovala i rozhraní Cloneable. První uvedené rozhraní vynucuje implementaci metody getDMSLCode, která jako řetězec vrací DMSL kód příslušného elementu. Cloneable je doporučeno z důvodu, že je například při otevření dialogu potřeba vytvářet lokální kopii editované struktury, která se podle způsobu zavření dialogu buď uloží (OK) nebo zahodí (Cancel).

Pokud nějaká třída obsahuje seznam jiných tříd, pak je k tomuto účelu použita třída, implementující rozhraní List. Ta zajišťuje, že uspořádání komponent zůstane zachováno podle pořadí vložení. Je to nutné například v elementu FunctionCall, který obsahuje uspořádaný seznam polí, předávaných funkci jako argumenty.

Byla zrušena třída Actions.java, protože nemá ekvivalent v DMSL a její obsah byl rozptýlen do několika jiných menších tříd. Z výše popsaného přístupu tvoří jedinou výjimku FunctionPool, který se jako jedinou datovou strukturu (včetně obsažených podkomponent) podařilo integrovat do nového řešení beze změny.

Kvůli vysokému počtu nových datových tříd byl pro ně zaveden zvláštní balík `cz.vutbr.fit.dataminer.core.system.dmsl`.

6.3 Změny v DMSL

Tato kapitola vychází z DMSL, upraveného Ing. Doležalem v jeho práci [3] a dokumentuje rozdíly oproti jeho řešení. Bližší informace viz [3], kapitola 7.4 a příloha A.

6.3.1 DataModel

```
<!ELEMENT DataModel (DataMatrix*, Annotation?) >
<!ATTLIST DataModel name               CDATA #REQUIRED
                    functionPoolRef    CDATA #IMPLIED >
```

Element Conditions byl přesunut do DataMatrix. DataModel nemusí obsahovat žádnou DataMatrix, protože uživatel aplikace může uložit projekt, ze kterého odstranil všechny komponenty pro výběr dat.

6.3.2 DataMatrix, Conditions

```
<!ELEMENT DataMatrix (DataField*, MatrixTreatment*, Conditions*, Annotation?) >
<!ATTLIST DataMatrix name CDATA #REQUIRED >

<!ELEMENT Conditions (Part*,Annotation?) >
<!-- <!ATTLIST Conditions EMPTY > -->

<!ELEMENT Part EMPTY >
<!ATTLIST Part      column1 CDATA #REQUIRED
                    column2 CDATA #REQUIRED >
```

Datová matice byla v kapitole 5.2 navržena tak, aby k výběru dat z více tabulek a jejich integraci do tabulky jedné nebyl potřeba celý DataModel, ale pouze jeden element DataMatrix. Hlavní motivací bylo, aby každá komponenta, pracující s DMSL, měla svou odpovídající strukturu v tomto jazyce a tyto struktury měly podobnou stavbu a podobné vlastnosti. Pro výběr dat byla určena DataMatrix, pro transformace DataMiningMatrix.

Atribut name jednoznačně identifikuje DataMatrix, je na něj navázána komponenta grafu a zároveň označuje výstupní tabulku, která podle této DataMatrix vznikne. Musí být unikátní i v prostru každého elementu DataMiningModel, který se odkazuje na DataModel, ve kterém se dotyčná DataMatrix nachází. Je to z důvodu, že DataMiningMatrix se mohou odkazovat pomocí elementu UseMatrix jak na datové matice, tak na dolovací matice. Jednoznačnost jmen zajišťuje systém, uživatel k jejich tvorbě nemá a ani nepotřebuje mít přístup.

Element MatrixTreatment použit není, protože aplikace VIMEO funkcí je považována za transformaci a používá se v elementu DataMiningMatrix. I když ho aplikace nevyužívá, zůstala zachována možnost ho použít při případných dalších rozšířeních. Z DTD tedy nezmizel.

Novým elementem je Conditions, který byl původně rozšířením, navrženým a implementovaným v [3]. Obsahuje sloupce, které se použijí v podmínce WHERE při generování SQL dotazu, který vytváří tabulku.

Příklad:

```
<Conditions>
<Part column1="ZAMESTNANCI.POBOCKA" column2="POBOCKY.ID"/>
</Conditions>
```

Způsobí vygenerování kódu **WHERE ZAMESTNANCI.POBOCKA = POBOCKY.ID.**
Více elementů Part se spojuje pomocí logické spojky AND.

6.3.3 DataField, FieldProperties

```
<!ELEMENT DataField (FieldProperties) >
<!ATTLIST DataField name CDATA #REQUIRED >


<!ELEMENT FieldProperties (Annotation?) >
<!ATTLIST FieldProperties atomicity    (%ATOMICITY;)       #REQUIRED
                          dataType     (%DATA-TYPE;)       #REQUIRED
                          table          CDATA             #REQUIRED
                          column         CDATA             #REQUIRED
                          granularity  (%GRANULARITY;)     #IMPLIED
                          scale        (%SCALE;)           #IMPLIED
                          length       (%INT-NUMBER;)      #IMPLIED >
```

Atribut name u DataField značí jméno sloupce ve výsledné tabulce. Zdrojový sloupec je uveden ve FieldProperties novými atributy table (název zdrojové tabulky) a column (název zdrojového sloupce). Tímto je zajištěna možnost načítat libovolné sloupce z libovolných tabulek, případně vybrat jeden sloupec vícekrát, pokaždé s jiným výsledným jménem.

6.3.4 Matrix Treatment, Transformations

```
<!ELEMENT MatrixTreatment ((VIMEOValues | ValueTreatment | Transformation )*,
                           Annotation?) >
<!ATTLIST MatrixTreatment name CDATA #REQUIRED >
```

Atribut name zůstal zachován pro případné změny v budoucnu, aplikace ho však nepoužívá. MatrixTreatment může být i prázdný.

Byl přejmenován element Tranformation na Transformation, zřejmě se jednalo o překlep.

6.3.5 DataMiningMatrix, UseMatrix

```
<!ELEMENT DataMiningMatrix ((UseMatrix | (Query, UseMatrix*)?),
                                DataMiningField*, MatrixTreatment*, Annotation?) >
<!ATTLIST DataMiningMatrix name CDATA #REQUIRED >
```

```
<!ELEMENT Query ANY >
<!ATTLIST Query     language            CDATA #REQUIRED
languageVersion     CDATA #IMPLIED >


<!ELEMENT UseMatrix (UseField*) >
<!ATTLIST UseMatrix       matrixRef           CDATA #REQUIRED
                          matrixTreatmentRef  CDATA #IMPLIED >

<!ELEMENT UseField EMPTY >
<!ATTLIST UseField name CDATA #REQUIRED >
```

Výběr dat zajišťuje DataMatrix, proto element Query není použit. Je používán pouze jeden element MatrixTreatment, i když jich specifikace dovoluje použít víc. DataMiningMatrix nemusí mít žádné DataMiningField.

Nově DataMiningMatrix nemusí obsahovat element UseMatrix. Je to z toho důvodu, že při definici dolovací úlohy lze jakoukoliv komponentu grafu odpojit a soubor uložit. Taková dolovací matice nemá žádný vstup, nikam neodkazuje, přesto je žádoucí, aby v ní zůstalo uloženo nastavení, protože může být opět připojena.

Element UseMatrix nepoužívá UseField. Implicitně se berou všechna pole ze vstupní matice a výběr konkrétních polí je realizován pomocí elementu DataMiningField. Jestliže nějaká vstupní pole nemají být použita, pak se na ně žádný DataMiningField neodkazuje a jsou ignorována.

6.3.6 DataMiningField

Došlo ke stejné změně ve FieldProperties, jako u DataField, pouze atribut table není z důvodu omezení redundance používán. Vstupní data se berou pouze z jedné tabulky a tu již definuje UseMatrix. Viz kapitola 6.3.3.

6.3.7 DomainKnowledge

Tento element nebyl použit, zůstává však ve specifikaci pro další rozvoj.

6.3.8 DataMiningTask, Knowledge

```
<!ELEMENT DataMiningTask ANY >
<!ATTLIST DataMiningTask  name                CDATA #REQUIRED
                          language            CDATA #REQUIRED
                          languageVersion     CDATA #IMPLIED
                          type                CDATA #IMPLIED >

<!ELEMENT Knowledge ANY >
<!ATTLIST Knowledge       name                CDATA #REQUIRED
                          language            CDATA #REQUIRED
                          languageVersion     CDATA #IMPLIED
                          type                CDATA #IMPLIED >
```

Oběma položkám přibyl povinný atribut name, který slouží jako propojení s dolovacím modulem.

6.4 Rozšíření abstraktní třídy MiningPiece.java

Důležitou změnou v tomto rozhraní bylo využití atributu kernelBinding, který slouží jako identifikátor pro přístup k DMSL elementům a k databázovým tabulkám. U některých komponent existuje, ale není využíván. Aby mohlo být implementovány změny navržené v kapitolách 5.1 a 5.3, musela být třída rozšířena o následující metody:

afterAcceptEvent

Tato metoda je volána po vložení do komponenty scény. Komponenta je nastavena, včetně id a kernelBinding. Touto událostí již nelze proces vkládání do scény zastavit. Příkladem implementace metody pro komponentu SelectData je vložení DataMatrix do DataModelu.

afterRemoveEvent

Tato metoda je volána po odstranění komponenty ze scény. Příkladem implementace metody pro komponentu SelectData je odstranění příslušné DataMatrix z DataModelu.

beforeConnectEvent

Je volána u cílové komponenty, pokud se jí někdo pokusí připojit. Může spojení odmítnout a tím proces připojování zastavit. Například komponenta Transformations, která byla odpojena od zdroje (jiné komponenty) má pořád nastaveny transformace původních sloupců (DataFields nebo MiningFields). Po připojení nového zdroje provede kontrolu, zda zdrojové sloupce svým jménem a typem odpovídají těm z minulého spojení. Pokud ano, spojení povolí, pokud ne, vyvolá dialog a o povolení spojení rozhodne uživatel.

afterConnectEvent

Tato metoda je volána po vytvoření spojení. Nemůže zastavit proces připojování. Příkladem implementace této reakce může být komponenta Transformations, která byla odpojena od zdroje (jiné komponenty) a následně připojena k jinému zdroji s odlišnými vstupními sloupci (DataFields nebo MiningFields). Odsouhlasil-li uživatel vytvoření nového spojení, pak se v této metodě provedena synchronizace.

beforeDisconnectEvent

Analogická metoda k beforeConnectEvent.

afterDisconnectEvent

Analogická metoda k afterConnectEvent.

updateDestPieces

Vynutí synchronizaci všech následníků volající komponenty. Je-li například v komponentě SelectData odstraněn nějaký sloupec, pak je pomocí této metody odstraněn z celého procesu dolování

getOutputPanel

Nahrazuje metodu getParametersGUI ze zrušeného rozhraní MiningModule.java. Pokud není překryta (override), pak vrací null, jinak vrátí objekt typu jPanel. Využívá se pro spolupráci dolovacího modulu s komponentou Report.

6.5 Komponenty a práce s grafem

Bylo odlišen identifikátor komponenty (kernelBinding) od její popisky v grafu (displayableName). Uživatel se nemusí starat o správu těchto identifikátorů, může si pouze komponenty přejmenovat. Na popisky nejsou stanovena žádná omezení, mohou být prázdné i duplicitní.

Omezení počtu vložených komponent stejného typu je implementováno pomocí generování ID. Generuje-li si v metodě beforeAcceptEvent unikátní ID, pak ji aplikace povolí vložit, jinak odmítne. Příklad viz zdrojové kódy některé konkrétní komponenty.

6.5.1 Nové komponenty

Původní komponenty DataFields, MiningFields, Interpretation, Actions a Function byly zrušeny, místo nich vznikly komponenty nové.

6.5.1.1 SelectData

Vznikla jako ekvivalent původní komponenty DataFields, převzala z ní obrázek ikonky a GUI. Slouží k výběru vstupních sloupců a umožňuje vkládat spojovací podmínky Conditions (viz 6.3.2). KernelBinding odkazuje na DMSL element DataMatrix.

6.5.1.2 Vimeo

Tato komponenta funguje jako datový filtr. Všechny výstupní sloupce jsou stejné jako vstupní, pouze je k nim přiřazena nějaká VIMEO funkce. Neumožňuje sloupce přejmenovávat, od toho je komponenta Transformations. Zajišťuje přístup k editacím funkcí. KernelBinding odkazuje na DMSL element DataMiningMatrix. Vimeo reakce „Average“ nebyla implementována.

6.5.1.3 Transformations

Umožňuje nezahrnout do výstupu vstupní sloupce, odvodit sloupce nové pomocí funkcí a aplikovat DME transformaci. Zajišťuje přístup k editacím funkcí. KernelBinding odkazuje na DMSL element DataMiningMatrix. Transformace diskretizace nebyla implementována.

6.5.1.4 Insight

Zobrazí obsah databázové tabulky. KernelBinding neodkazuje na žádný element.

6.5.2 Práce s grafem

Dolovací proces se spouští přes popup menu komponenty položkou Run. Nepřipojená komponenta nemůže být ani otevřena ani spuštěna. Tlačítko Run na hlavním panelu zůstalo pro případné pozdější využití zachováno, je však neaktivní. Pomocí příznaku se zjistí, zda komponenta byla nebo nebyla editována a needitované komponenty jsou přeskočeny. Po otevření souboru je všem komponentám nastaven příznak modifikace.

Graf tvoří strom, žádná z komponent nemůže mít více, než jednu vstupní hranu. Může mít ovšem libovolný počet výstupních. Pokud je komponenta odpojena, pak zůstane zachováno její nastavení. Opětovné připojení na původní místo uvede graf do původního stavu.

Pokud je komponenta připojena ke zdroji, který obsahuje stejné sloupce jako před odpojením (stejné jméno, typ, u řetězců délka) a nějaké navíc, bude seznam vstupních sloupců rozšířen.

Pokud je komponenta připojena ke zdroji, který některé sloupce z předchozího připojení neobsahuje, pak je vyvolán dialog, kterým uživatel rozhodne, zda povolit připojení komponenty (podgrafu) a sloupce synchronizovat. Tzn. odstranit z podgrafu všechny sloupce, které nejsou v nadgrafu.

Je-li z nějaké komponenty odstraněn výstupní sloupec, projeví se to odstraněním z celého podgrafu podobně, jako u synchronizace.

6.6 Konvence pro umísťování souborů

Kvůli zachování přehlednosti ve zdrojových souborech projektu byla zavedena následující konvence:

- Všechny soubory patřící jiným komponentám než dolovacím modulům, budou ukládány do balíčku cz.vutbr.fit.dataminer.editor.palette.items. Mohou to být obrázky ikonek, zdrojové soubory GUI apod. Soubory, patřící jedné komponentě by měly začínat stejným jménem.
- Grafické uživatelské rozhraní, které nepatří komponentám ani modulům bude ukládáno do balíčku cz.vutbr.fit.dataminer.core.gui. Může to být například GUI pro editaci funkcí.
- Každý modul bude mít vlastní projekt, kde si bude uchovávat své soubory.

6.7 Ostatní

Byl upraven způsob, jakým se aplikace připojovala k databázi. Nastavení JDBC zůstalo zachováno, DME spojení se nastavuje ve vlastnostech projektu – (klik pravým tlačítkem na projekt, položka

properties). Je to nouzové řešení, celý systém správy spojení potřebuje přepracovat, viz návrh v kapitole 9.2.

Z předchozí verze projektu se podařilo zachovat FunctionPool, pouze byly opraveny dvě chyby. Funkce po smazání nešla znovu vložit (chyba grafické nadstavby) a při pokusu o přejmenování funkce se vytvořila její kopie.

7 Grafické uživatelské rozhraní a ovládání programu

V této kapitole popíšu GUI jednotlivých komponent a jejich ovládání. Komponenty budou seřazeny postupně podle návazností a u každého kroku bude krátký příklad. Čtenář si podle návodu bude moci sestavit a spustit ukázkovou dolovací úlohu. Každý příklad počítá s tím, že je program ve stavu po předchozím příkladu. Pokud není na serveru vytvořena tabulka sample, pak ji lze vytvořit pomocí skriptu sample.sql, který je na přiloženém CD. Tato tabulka bude použita v příkladech.

7.1 Založení projektu a nastavení připojení

Po spouštění nejprve založíme nový dolovací projekt. Průvodce otevřeme v hlavním menu položkou File→NewProject. Zvlolíme DataMiningProject a stiskneme tlačítko Next. Na další obrazovce nastavíme jméno a cestu k projektu, opět potvrdíme tlačítkem Next. Poslední obrazovka slouží k nastavení připojení k databázi. Vybereme New Oracle Database Connection, zvolíme jméno, heslo a přihlašovací řetězec.

Příklad:

Jméno: dmuser1
Heslo: dmuser1
URL: jdbc:oracle:thin:@pcuifs1.fit.vutbr.cz:1521:STUD

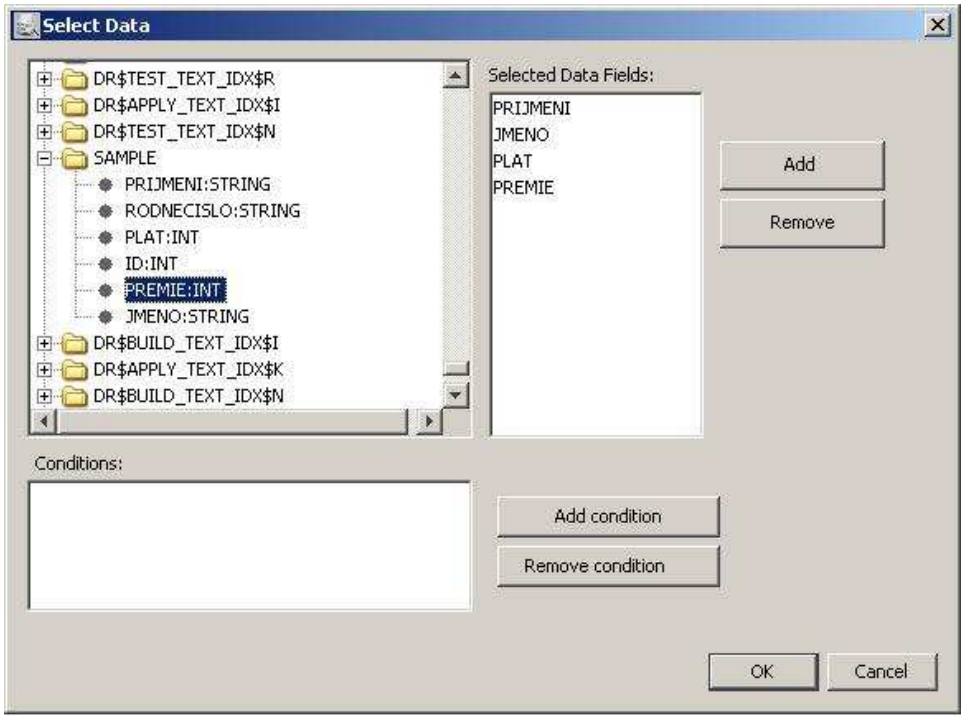
Nyní se vytvořil nový projekt. Zavřeme ho, na disku v domovském adresáři projektu v podadresáři dmproject otevřeme soubor project properties a na konec přidáme řádek :

connection.passDME=dmuser1

Znovu projekt otevřeme. Toto je nouzové řešení, původně bylo připojení k ODM řešeno napevno přímo ve zdrojových kódech.

7.2 Select Data

Slouží k výběru tabulek a sloupců, které budeme vybírat. V levém poli je strom, který zobrazuje tabulky v aktuální databázi (Default schema). Vybraná pole se nachází v seznamu vpravo. Jejich jméno musí být unikátní, mohou se však opakovat pod jiným názvem. Spojovací podmínky (Conditions) se vybírají tak, že uživatel klikne na tlačítko Add condition a v levém seznamu označí dva sloupce.



Obr. 7.1 –Komponenta Select Data

Příklad:

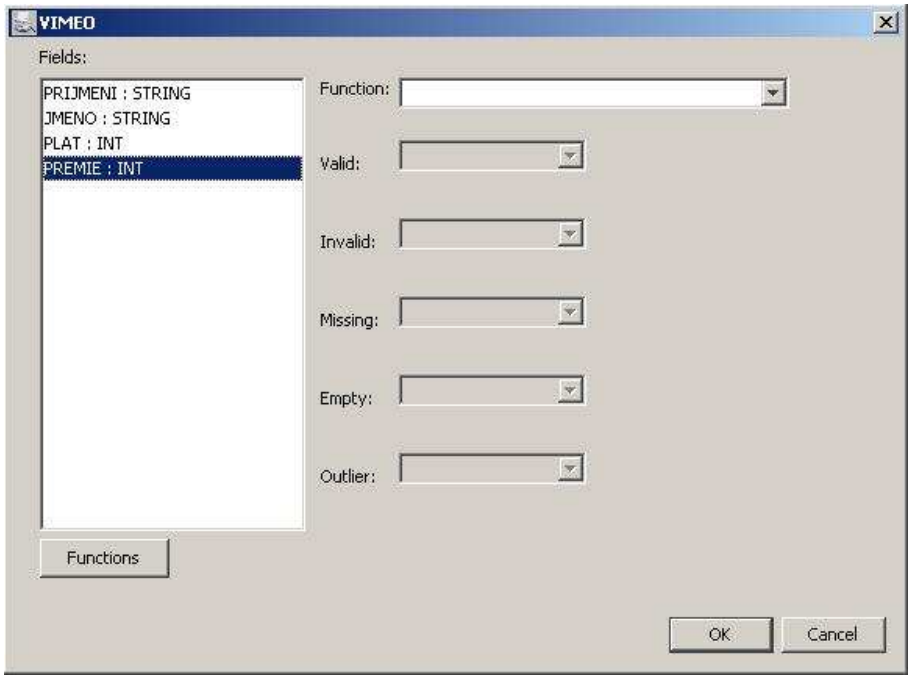
Rozbalíme projekt a poklikáme na soubor dmsl.xml. Z panelu komponent na pravé straně přesuneme do plochy jednu komponentu SelectData. Poklikáme na ikonku a po otevření komponenty si z tabulky SAMPLE vybereme sloupce PRIJMENI, JMENO, PLAT a PREMIE.

Zavřeme tlačítkem OK, klikneme na SelectData pravým tlačítkem myši a v menu zvolíme run. Tím vznikne v databázi tabulka. Na pracovní plochu vložíme komponentu Insight (lupa) a obě komponenty spojíme tak, že podržíme tlačítko CTRL, klikeme na komponentu SelectData a se stisknutým levým tlačítkem myši táhneme šipku nad komponentu Insight nad kterou tlačítko pustíme. Otevřením komponenty Insight můžeme zobrazit výslednou tabulku.

7.3 Vimeo

Zde může uživatel definovat VIMEO funkce a aplikovat je na jednotlivá pole. K definici VIMEO funkcí slouží tlačítko Functions. GUI pro správu a definici funkcí je převzat z původního projektu [3].

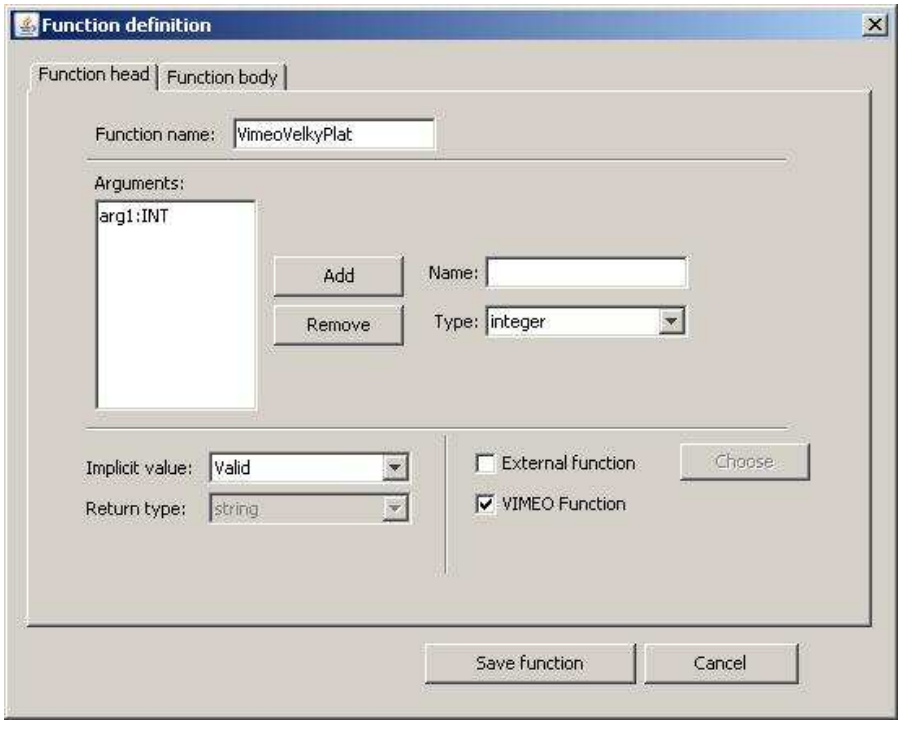
Po kliknutí na některé pole ze seznamu mu můžeme přiřadit VIMEO funkci a následné nastavit reakce na výstupní hodnoty v příslušných roletových menu.



Obr. 7.2 – Komponenta Vimeo

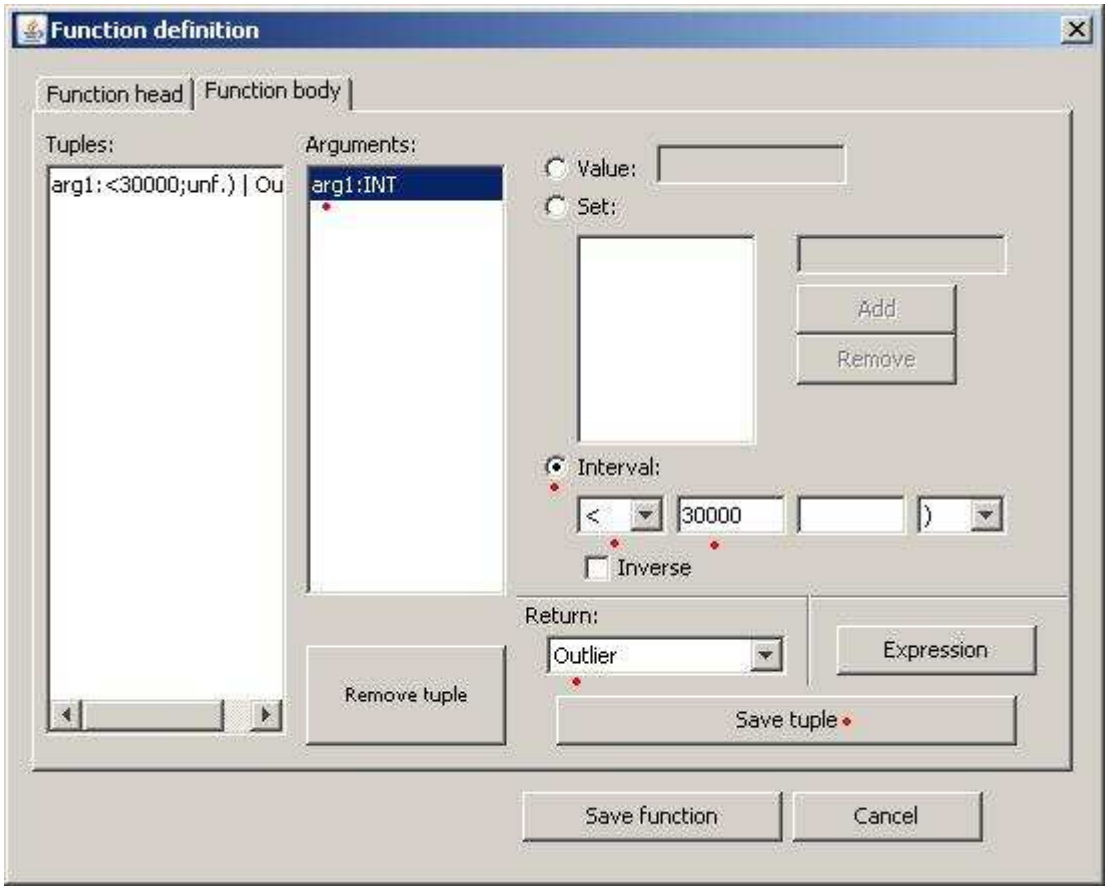
Příklad:

Na plochu vložíme komponentu Vimeo, připojíme k ní Select Data tak, aby šipka vedla směrem do komponenty Vimeo a komponentu otevřeme. Nejprve je potřeba nějakou VIMEO funkci vytvořit. Klikneme na tlačítko Functions, otevře prázdný seznam funkcí. Tlačítkem New přidáme novou funkci.



Obr. 7.3 – Definice hlavičky VIMEO funkce

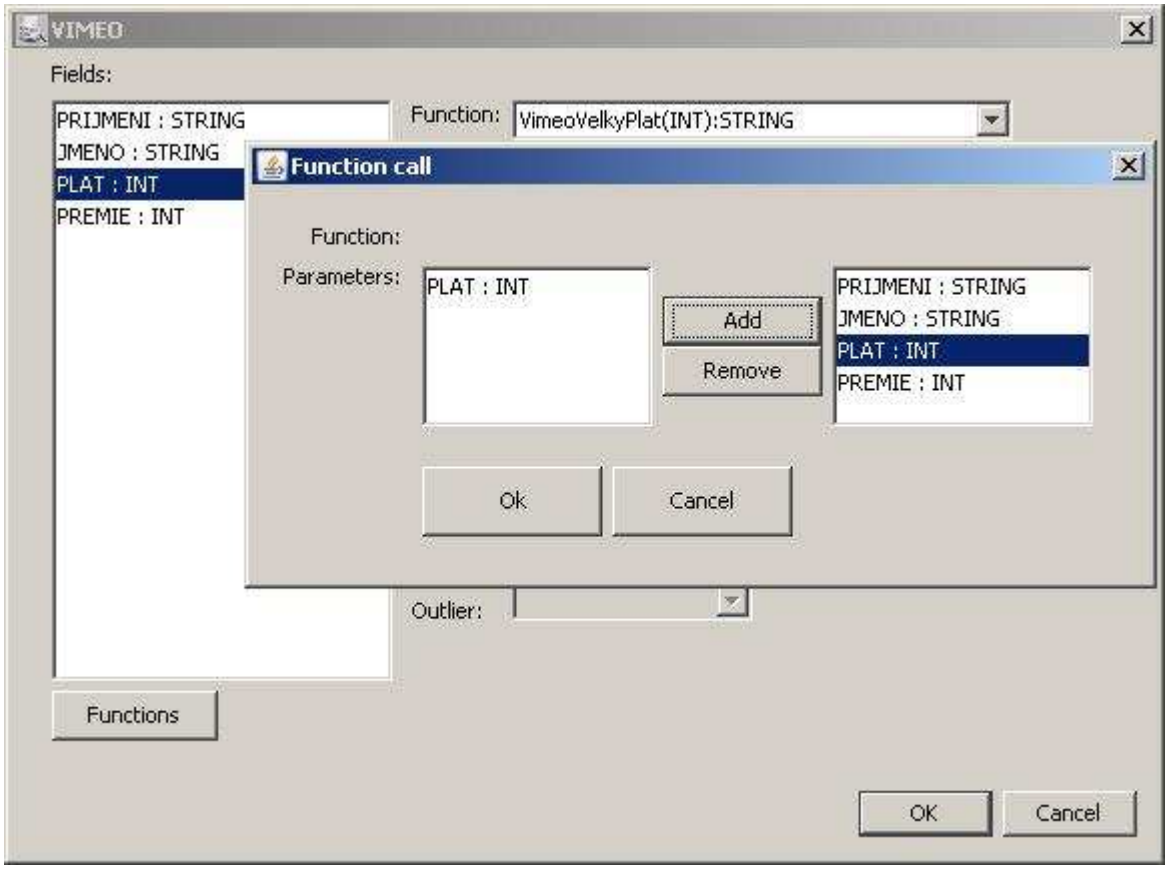
Do functionName zadáme například „vimeoVelkyPlat“. Přidáme jeden argument tak, že do políčka Name napíšeme jméno atributu (arg1), zvolíme jeho typ (integer) a klikneme na tlačítko Add. Zaškrtneme VIMEO Function, protože se jedná o VIMEO funkci a klikneme na Save function. V seznamu funkcí označíme novou funkci a zadáme Edit. Funkci je nejprve třeba uložit, aby se její argumenty daly použít. FunctionPool a jeho GUI bylo použito z původního projektu a tento jeho nedostatek nebyl opraven. Klikneme na záložku Function body.



Obr. 7.4 – Definice těla VIMEO funkce

Na této záložce se definuje tělo funkce. Klikneme v seznamu Arguments na arg1, zvolíme možnost Interval, nastavíme ho od 30000 do nekonečna, jako výstupní hodnotu nastavíme Outlier a klikneme na Save tuple.

Nyní přiřadíme novou VimeoFunkci některému z polí. Na panelu Vimeo klikneme v seznamu na položku PLAT, v roletovém menu Function vybereme naši novou funkci a přiřadíme jí vstupní argument PLAT. Viz následující obrázek:



Obr. 7.5 – Přiřazení vstupních parametrů funkci

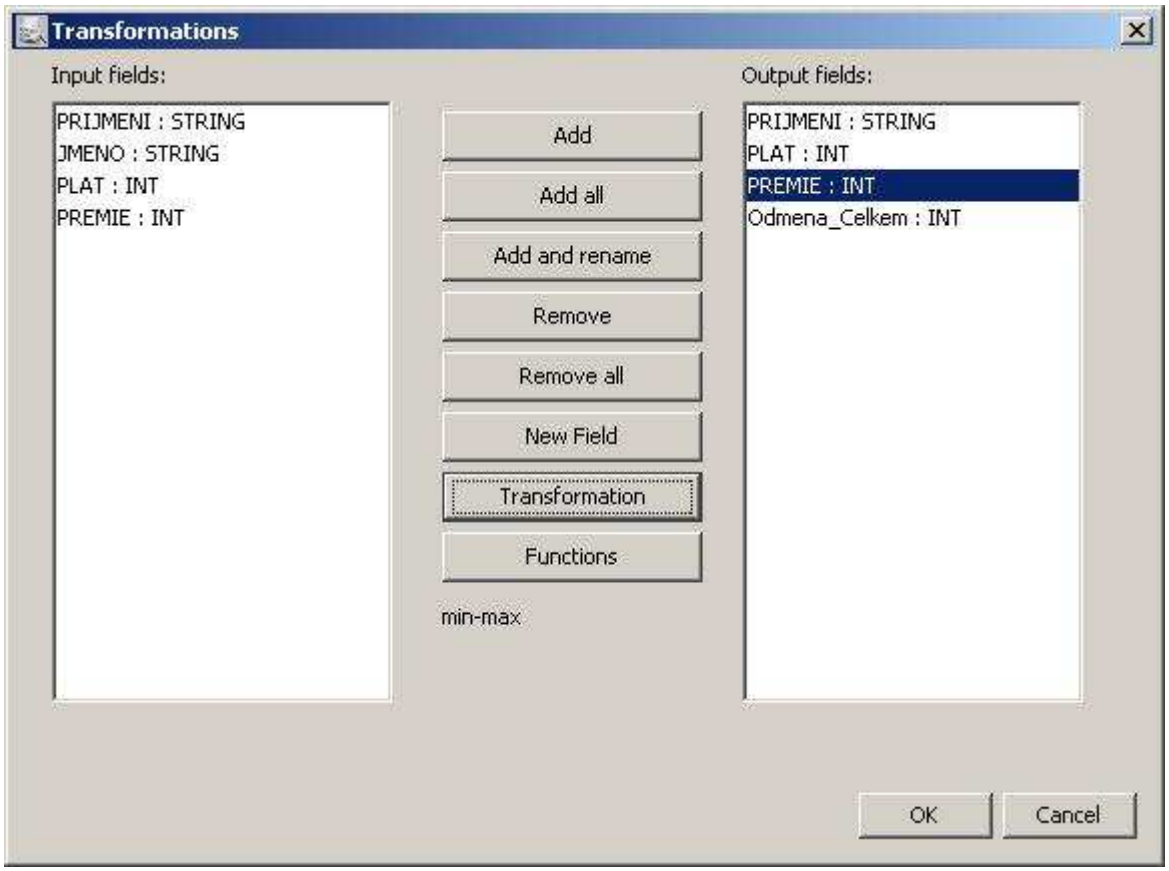
Po stisku tlačítka OK se dostaneme opět na panel Vimeo, na kterém jsou všechna roletová menu povolená. V položce Outlier nastavíme reakci například na Ignore. Po stisku tlačítka OK se dostaneme zpět na scénu.

Nyní bude pravděpodobně potřeba aplikaci restartovat protože po delší nečinnosti padá spojení a nejde obnovit. Po restartu můžeme komponentu Vimeo spustit, připojit k ní Insight a zkontrolovat, že záznam Marek Daněk s platem větším než 30000 opravdu zmizel.

7.4 Transformations

Zde se definují odvozené sloupce a DME transformace. Odvozený sloupec lze definovat tlačítkem New Field.

Pro aplikaci transformace na výstupní sloupec musíme tento sloupec označit v pravém seznamu, kliknout na tlačítko Transformation a vybrat transformaci. Při použití normalizace na sloupec typu INT je tento sloupec automaticky přetypován na REAL, protože výsledek normalizace je reálné číslo. Normalizace jiného než číselného sloupce je ignorována.



Obr. 7.6 – Komponenta Transformations

Příklad

Do pracovní plochy vložíme komponentu Transformations, připojíme ji jako následníka komponenty Vimeo a komponentu otevřeme. Budeme odvozovat nové pole, proto nejprve definujeme součtovou funkci. Klikneme na tlačítko Functions, dále na New a stejným způsobem, jako v předchozí kapitole vytvoříme hlavičku funkce. Nazveme ji například Soucet, bude mít dva vstupní argumenty typu INT,

které pojmenujeme arg1 a arg2. Možnost VIMEO Function necháme **nezaškrtnutou**, protože se nejedná o VIMEO funkci. Stiskneme Save function a funkci stejně jako v minulém příkladě označíme a zmáčkneme tlačítko Edit.

Klikneme na záložku Function body, v seznamu argumentů označíme libovolný z nich, dále klikneme na možnost Interval, ale do rozsahu v tomto případě nezadáme nic. Tímto postupem jsme zvolili, že bude existovat tuple, který nebude mít žádné vstupní omezení (na rozdíl od minulého případu, kdy měl tuple omezení >=30000). Klikneme na tlačítko Expressions a zadáme arg1+arg2. Dále na tlačítko OK a Save Tuple. Funkce by měla být nyní správně nadefinovaná.

Vrátíme se do dialogu Transformations, klikneme na tlačítko New Field, zadáme jméno nového sloupce Celkova_odmena, vybereme funkci Soucet a přiřadíme jí jako vstupní argumenty políčka PLAT a PREMIE.

Zbývá nám definovat transformaci. V pravém sloupci klikneme na položku PLAT, zpřístupní se tlačítko Transformation, kterým zvolíme například min-max normalizaci. Komponentu spustíme a pomocí Insight můžeme zkontrolovat výsledky.

8 Postup při vytváření nového modulu

Tato kapitola popisuje postup vytvoření a integrace nového modulu. Je obdobou [4], kapitoly 8. Některé části mohou být převzaty, protože obě kapitoly popisují tentýž postup, tato je však aktuální pro novou verzi systému. K tvorbě dolovacího modulu je potřeba API vrstva mateřské aplikace a knihovnu jádra v binární podobě.

8.1 Vytvoření modulu NetBeans

Nejprve je třeba vytvořit nový NetBeans modul. V hlavním menu klikněte na File→ New Project, v kategorii NetBeans Modules vyberte Module, nastavte jméno, cestu a vyberte možnost Standalone Module. Tato možnost znamená, že modul bude vyvíjený jako samostatný projekt. Jako předponu Code Name Base použijte cz.vutbr.fit.dataminer.

8.2 Implementace abstraktní třídy MiningPiece

Aby dolovací modul byl přístupný v aplikaci, je potřeba implementovat MiningPiece abstraktní třídu, která se nachází v API vrstvě. Každá třída, která dědí od třídy MiningPiece, představuje prvek, který lze vložit do procesu dolování dat.

V konstruktoru třídy je třeba nastavit parametry pro zobrazení, nejlépe pomocí lokalizačního bundle.

Příklad z komponenty Transformations.java:

```
setDisplayableName(NbBundle.getMessage(Transformations.class, "NAME_Transformations"));
setDescription(NbBundle.getMessage(Transformations.class, "HINT_Transformations"));
setPaletteIcon("cz/vutbr/fit/dataminer/editor/palette/items/Transformations16.png");
setEditorIcon("cz/vutbr/fit/dataminer/editor/palette/items/Transformations32.png");
```

Narozdíl od předchozí verze projektu zde neimplementujeme interface MiningModule.java, je plně nahrazeno rozšířeními abstraktní třídy MiningPiece.java.

Povinná je však implementace metody openCustomizingDialog(), která zajišťuje otevření dialogu pro nastavení parametrů komponenty. Implementace by neměla vytvářet vlastní okna, ale měla by použít prostředky platformy NetBeans pro dialogy, které se starají o zachování jednotného vzhledu a korektní chování. Příklad implementace metody OpenCustomizingDialog() viz zdrojové kódy komponenty Transformations.java na přiloženém CD.

8.3 Integrace do aplikace

Cit. [4]: Aby nový modul mohl být integrován do aplikace, do které se nainstaluje, je potřeba jej zaregistrovat do souboru layer.xml. Tento soubor představuje virtuální systém souborů, který se z různých modulů integruje do jednoho celku. To znamená, že záznamy, které tento modul vytvoří, budou viditelné i z ostatních modulů. Ostatní moduly o dolovacím modulu nic neví, dokonce ani nemají přístup k jeho třídám, i přesto, že běží v rámci jednoho virtuálního stroje Javy. O vytváření instancí se stará System FileSystem, který poskytuje instance ostatním modulům.

Pro to, aby se dolovací modul objevil v MiningPieceRegistry (a tím i v paletě komponent), je potřeba vytvořit pár záznamů v layer.xml:

```
<filesystem>
 <folder name="MiningPieceRegistry">
  <folder name="MiningModules">
   <file name="dataminer-module-testmodule-TestMiningModule.instance" />
  </folder>
 </folder>

 <folder name="MiningPieceConfig">
  <folder name="dataminer-module-testmodule-TestMiningModule">
   <folder name="accept">
    <file name="cz-vutbr-fit-dataminer-editor-palette-items-Actions">
     <attr name="required" boolvalue="true"/>
    </file>
   </folder>
  </folder>
  <folder name="cz-vutbr-fit-dataminer-editor-palette-items-Report">
   <folder name="accept">
    <file name="dataminer-module-testmodule-TestMiningModule"/>
   </folder>
  </folder>
 </folder>
</filesystem>
```

První skupina elementů vytváří instanci třídy, která dědí od abstraktní třídy MiningPiece a reprezentuje dolovací modul. Je vložena do kategorie MiningModules a v paletě bude zobrazena v této kategorii. Druhá skupina elementů pak specifikuje, které dolovací elementy lze spojovat s tímto dolovacím modulem. V tomto případě TestMiningModule vyžaduje spojení z uzlu Actions (atribut required) a musí být akceptován uzlem Report pro to, aby bylo možné spojit tyto dva uzly a Report mohl zobrazit výsledek dolovacího modulu.

Takovýto modul lze snadno vystavit na internet ve formě souboru NBM a zpřístupnit jej k pohodlné instalaci z rozhraní aplikace. Tento soubor lze rovněž distribuovat samostatně a nainstalovat jej z lokálního disku do aplikace. Tato funkce se nachází v aplikaci v menu Tools→Plugins.

9 Závěr

9.1 Zhodnocení výsledků

Cílem projektu bylo vytvořit z původního prototypu aplikace [4] použitelný program. K jeho dosažení bylo nutné provést změny jak v jádře systému, tak v grafické nadstavbě. Hlavním nedostatkem předchozí verze byl oddělený vývoj obou těchto částí. Jádro bylo navrženo pro formulářovou aplikaci, zatímco grafická nadstavba měla mnohem větší potenciál, protože umožňovala definovat dolovací proces pomocí orientovaných grafů. Výsledkem byl tedy produkt se stejnou vyjadřovací silou, jakou měla původní formulářová aplikace.

Aby bylo možné posunout projekt kupředu, bylo nutné zásadním způsobem přestavět jádro. Při návrhu jeho změn jsem vycházel z připomínek Ing. Gáleta [4] v předchozím projektu. Musely být vytvořeny nové datové struktury, přičemž byl kladen důraz zejména na jejich snadnou modifikovatelnost a rozšiřitelnost. Nad těmito datovými strukturami bylo nutné znovu napsat algoritmy jádra. Z původního řešení se podařilo zachovat část, která zajišťovala práci s interními funkcemi. Se změnami jádra souvisí i modifikace jazyka DMSL (formát ukládání dat), aby lépe vyhovoval požadavkům aplikace.

Změny v grafickém uživatelském rozhraní zahrnují především práci s komponentami. Díky podpoře jádra byla podstatně rozšířena možnost jejich spolupráce. Nyní lze definovat mnohem složitější a univerzálnější dolovací procesy. Program plně využívá potenciál grafického uživatelského rozhraní, postaveného na knihovně NetBeans Visual Library.

Rovněž bylo nutné změnit rozhraní pro připojování dolovacích modulů. Původní verze nebyla funkční a ani po opravě by se nehodila do nového konceptu. Byl zrušen přístup, kdy se o moduly staralo jádro a fungovaly jako externí třída, místo něj je na moduly pohlíženo jako na standardní komponenty grafu.

Nepodařilo se implementovat některé algoritmy přípravy dat, konkrétně reakci na VIMEO funkci average, která nefungovala ani v předchozí verzi a transformaci diskretizace.

Celý systém byl přepracován tak, aby byly odstraněny zásadní nedostatky, které bránily dalšímu rozvoji. Program poskytuje základní nástroje pro přípravu dat a funkční rozhraní pro připojování dolovacích modulů. Projekt je poměrně rozsáhlý, k reálně použitelné aplikaci budou podle mého odhadu potřeba alespoň dvě další iterace.

9.2 Další rozvoj aplikace

Aby mohl být program nasazen v praxi pro skutečné dolování bude kromě následujících úprav potřeba i důkladného testování. Po napsání dolovacích modulů by mohl být například použit na

cvičeních. Velmi by to prospělo propagaci projektu mezi studenty. Následující navrhované úpravy jsou seřazeny podle důležitosti, nejdůležitějšími počínaje.

Správa připojení

Tato část projektu je řešena velmi nevhodným způsobem. Na první pohled jednoduchá věc je v tomto projektu poněkud složitější, protože komponenta, použitá v předchozím projektu zřejmě nepodporuje připojení k ODM. Jeho konfigurace je prováděna nouzovým způsobem, v předchozí verzi byla dokonce napevno daná ve zdrojovém kódu. Server navíc po několikaminutové nečinnosti spojení uzavře a chybí mechanismy pro jeho udržování nebo obnovu.

Nejdůležitějším úkolem příští iterace je najít spolehlivé řešení tohoto problému. Aplikace může poskytovat libovolné služby, bez kvalitního spojení se serverem však nebude nikdy opravdu použitelná.

Implementace zbývajících algoritmů přípravy dat

Tuto nedokončenou část bude třeba implementovat přednostně. Výpočet průměrných hodnot popsaný v kapitole 3.2.2.1 a diskretizace jsou důležitými kroky při přípravě dat.

Oddělení více klientů na jednom serveru

Doposud se počítalo, že na server je připojena pouze jedna aplikace. Při připojení více aplikací by došlo ke konfliktu, protože by si vzájemně začaly přepisovat databázové tabulky.

Přepracování třídy FunctionPool.java

Tato třída byla začleněna do původního projektu, nezapadá však dobře do celkové koncepce - má například své vlastní datové struktury a není dobře modifikovatelná. Má poněkud nekvalitní grafické uživatelské rozhraní a neumožňuje práci s více. Špatně jsou řešeny kontroly vstupních argumentů, nefungují použití externích funkcí a práce s jinými datovými typy než s číselnými je problematická. Při na odstranění funkce program nereaguje vymazáním všech datových sloupců, které touto funkcí vzniknou a pod.

Bylo by dobré tuto třídu od základu přepracovat a začlenit ji standardním způsobem mezi ostatní datové struktury. Vhodnou organizací by bylo seskupovat funkce do různých instancí DMSL elementu FunctionPool a ty moci z projektu exportovat a znovu do něj importovat. Mohly by tak vnikat celé knihovny externích funkcí.

Tuto část projektu hodnotím jako nejnáročnější ze všech navrhovaných úprav, rozsahem srovnatelnou s tvorbou celého dolovacího modulu.

Implementace dolovacího procesu pomocí vláken

Při spuštění dolovacího procesu aplikace přestane až do jeho skončení reagovat. Spuštěný dolovací proces neposkytuje informace o svém průběhu, nelze ho přerušit a chování programu nepůsobí dobře na uživatele.

Rozšíření funkčnosti komponenty Insight

Funkce komponenty Insight by v budoucnu mohla být rozšířena o poskytování statistických informací o datech, případně o export dat do souboru.

Komponenty pro výběr dat

Komponenta SelectData neumožňuje spojovat tabulky jinak, než pomocí operátoru "=". Vhodným přístupem by mohlo být zrušení používání DMSL elementu Conditions a využití elementu Query, který nebyl původně určen pro datové matice.

Mohly by vzniknout nové komponenty, které by se dokázaly importovat data z jiných databází, například z MySQL.

Optimalizace

Program je náročný na přenos dat a komunikaci mezi serverem. Například výpočet VIMEO funkcí je celý řešen v aplikaci. Po vyřešení ostatních závažnějších problémů by bylo dobré analyzovat, zda je možné některé funkce klienta implementovat pomocí úložných procedur v databázi.

Dolovací moduly

Budou řešeny jako samostatné diplomové projekty.

Literatura

[1] Han J., Kamber M.: *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001

[2] Zendulka J., Bartík V., Lukáš R., Rudolfová I.: *Získávání znalostí z databází*, studijní opora, Brno, 2006, Vysoké učení technické v Brně, Fakulta informačních technologií.

[3] Doležal J.: *Jádro systému pro dolování z dat v prostředí Oracle*, diplomová práce, Brno, 2006, Vysoké učení technické v Brně, Fakulta informačních technologií.

[4] Gálet M.: *Grafická nádstavba pro systém získávání znalostí*, diplomová práce, Brno, 2006, Vysoké učení technické v Brně, Fakulta informačních technologií.

[5] Kotásek, P.: *DMSL: Data Mining Specification Language*, disertační práce, Brno, 2003, Vysoké učení technické v Brně, Fakulta informačních technologií.

[6] *Oracle Database 10g Release 2 Documentation Library - b14340 Data Mining Application Developer's Guide*, 2007, Oracle, [online]
URL: http://download.oracle.com/docs/cds/B19306_01.zip

[7] *Oracle Database 10g Release 2 Documentation Library – b14339 Data Mining Concepts*, 2007, Oracle, [online], URL: http://download.oracle.com/docs/cds/B19306_01.zip

[8] *NetBeans*, 2008, Wikipedia, [online], URL: http://cs.wikipedia.org/wiki/NetBeans

[9] *NetBeansPlatform*, 2008, [online], URL: http://wiki.netbeans.org/NetBeansPlatform

[10] *Visual Library 2.0 – Documentation*, 2008 [online],
URL: http://graph.netbeans.org/documentation.html

[11] *JavaTM 2 Platform, Standard Edition, v 1.3.1, API Specification*, 2008, Sun Microsystems, [online], URL: http://java.sun.com/j2se/1.3/docs/api/

Seznam příloh

Příloha A: DTD použitého DMSL

```
<!-- External Definitions -->
<!-- ################### -->

<!ELEMENT Annotation (#PCDATA) >
<!ATTLIST Annotation date %DATE-TIME; #IMPLIED >


<!ENTITY % INT-NUMBER "CDATA" >        <!--  content must be an integer number -->
<!ENTITY % REAL-NUMBER "CDATA" >       <!--  content must be a real number -->
<!ENTITY % DATE-TIME "CDATA" >         <!--  content must be a dateTime value -->
<!ENTITY % REAL_OR_DATE-TIME "CDATA" > <!--  content must be a real number
                                            or a dateTime value -->
<!ENTITY % ANY-VALUE "CDATA" >   <!--  content is a value of any type:
                                       integer, real, string, or dateTime -->
<!ENTITY % DATA-TYPE      "integer | int | INT | real | REAL | string | STRING
                           |String | date | Date |DATE |dateTime" >
<!ENTITY % TRUE-FALSE "TRUE | true | FALSE | false" >


<!-- Entities -->
<!-- ######## -->

<!-- Interval closure -->

<!ENTITY % CLOSURE "open | closed" >

<!-- Values -->

<!ENTITY % VALUES "Value | Interval | Set" >


<!-- Value -->
<!-- ##### -->

<!ELEMENT Value EMPTY >
<!ATTLIST Value     value %ANY-VALUE;          #REQUIRED
                    dataType (%DATA-TYPE;)     #IMPLIED >


<!-- Interval -->
<!-- ######## -->

<!ELEMENT Interval (LeftMargin?, RightMargin?) >
<!ATTLIST Interval  defines (interval | complement) "interval"
                    relativeTo (domain | undefined) #IMPLIED >

<!ELEMENT LeftMargin EMPTY >
<!ATTLIST LeftMargin      value %REAL_OR_DATE-TIME; #REQUIRED
                          closure (%CLOSURE;)       #REQUIRED >

<!ELEMENT RightMargin EMPTY >
<!ATTLIST    RightMargin value %REAL_OR_DATE-TIME;  #REQUIRED
             closure (%CLOSURE;)                    #REQUIRED >
```

```
<!-- Set -->
<!-- ### -->

<!ELEMENT Set (Value*) >
<!ATTLIST Set       defines (set | complement) "set"
                    relativeTo (domain | undefined) #IMPLIED >


<!-- Value types -->

<!ENTITY % SCALAR-VAL-INT-TYPE   "interpretedMissing | interpretedEmpty |
                                  invalid | outlier | valid | missing | empty |
                                  Invalid | Outlier | Valid | Missing | Empty" >
<!ENTITY % NULL-INT-TYPE "nativeMissing | nativeEmpty" >
<!ENTITY % VALUE-INT-TYPE "%SCALAR-VAL-INT-TYPE; | %NULL-INT-TYPE;" >


<!-- Value interpretation and treatment -->

<!ENTITY % DEF-TAG-ASSIGN "explicit | pessimistic | optimistic" >
<!ENTITY % USE-FOR-DEF-ASSIGN "primary | secondary | both" >
<!ENTITY % VALUE-TREATMENT "ignore | average | globalconst" >
<!ENTITY % TREATMENT-TARGET "row | field | setItem" >
<!ENTITY % USE-VALUES "originals | substitutes" >
<!ENTITY % TRANFORMATION "normalization | discretization | clipping" >
<!ENTITY % NORM-TYPE "min-max | z-score | linear" >
<!ENTITY % CLIP-TYPE "winsorize | trim " >
<!ENTITY % BIN-TYPE "equi-width | quantile | top-n | custom | Custom" >


<!-- Operators -->

<!ENTITY % UNARY-OPERATOR        " plus | minus " >
<!ENTITY % BINARY-OPERATOR       " plus | minus | mult | div " >


<!-- Expressions -->

<!ENTITY % GENERAL-FUNCTION "InternalFunction | ExternalFunction" >
<!ENTITY % NORM-FUNCTION  "LinearNormalization" >
<!ENTITY % FUNCTION       "%GENERAL-FUNCTION; | %NORM-FUNCTION;" >
<!ENTITY % EXPRESSION     "FieldRef | Value | UnaryOperation |
                           BinaryOperation | %FUNCTION; | FunctionCall" >


<!-- Field properties -->

<!ENTITY % ATOMICITY      "scalar | set" >
<!ENTITY % GRANULARITY    "constant | binary | discreteFinite |
                           discreteInfinite | continuous" >
<!ENTITY % SCALE          "nominal | categorical | ordinal | interval | ratio" >


<!-- DMSL -->
<!-- #### -->

<!ELEMENT DMSL      (Header?, (FunctionPool | DataModel | DataMiningModel |
                     DomainKnowledge | DataMiningTask | Knowledge)+) >


<!-- Header -->
<!-- ###### -->

<!ELEMENT Header ((DMSLProducer | Annotation)+) >

<!ELEMENT DMSLProducer (Annotation?) >
<!ATTLIST DMSLProducer    producer                    CDATA #REQUIRED
                          producerVersion             CDATA #REQUIRED
```

```
                            date                            %DATE-TIME; #REQUIRED
                            authorName                      CDATA #IMPLIED
                            hostOperatingSystem             CDATA #IMPLIED
                            hostOperatingSystemVersion      CDATA #IMPLIED
                            implementationLanguage          CDATA #IMPLIED
                            implementationLanguageVersion   CDATA #IMPLIED >


<!-- Function Pool -->
<!-- ############# -->

<!ELEMENT FunctionPool ((%FUNCTION;)*) >
<!ATTLIST FunctionPool name CDATA #REQUIRED >


<!-- External Function -->
<!-- ################# -->

<!ELEMENT ExternalFunction (FunctionProperties, FieldRef*) >
<!ATTLIST ExternalFunction      name         CDATA #REQUIRED
                                classfile    CDATA #REQUIRED >


<!-- Function Properties -->
<!-- ################### -->

<!ELEMENT FunctionProperties (ArgumentProperties*, Annotation?) >
<!ATTLIST FunctionProperties    resultAtomicity     (%ATOMICITY;) #FIXED "scalar"
                                resultDataType      (%DATA-TYPE;) #REQUIRED
                                isvimeo (%TRUE-FALSE;) #REQUIRED >

<!ELEMENT ArgumentProperties (Annotation?) >
<!ATTLIST ArgumentProperties    atomicity (%ATOMICITY;) #FIXED "scalar"
                                dataType (%DATA-TYPE;) #REQUIRED >


<!-- Internal Function -->
<!-- ################# -->

<!ELEMENT InternalFunction (    FunctionProperties, FieldRef*,
                                Tuple*, Value ) > <!-- (%EXPRESSION;) -->
<!ATTLIST InternalFunction name CDATA #REQUIRED >

<!ELEMENT Tuple ((%VALUES;)+, (%EXPRESSION;)) > <!-- , (%TUPLE-RETURN;) -->


<!-- Other expressions -->
<!-- ################# -->

<!ELEMENT FieldRef EMPTY >
<!ATTLIST FieldRef name CDATA #REQUIRED >

<!ELEMENT UnaryOperation (%EXPRESSION;) >
<!ATTLIST UnaryOperation operator (%UNARY-OPERATOR;) #REQUIRED >

<!ELEMENT BinaryOperation ((%EXPRESSION;), (%EXPRESSION;)) >
<!ATTLIST BinaryOperation operator (%BINARY-OPERATOR;) #REQUIRED >

<!ELEMENT FunctionCall (FieldRef*) >
<!ATTLIST FunctionCall functionRef CDATA #REQUIRED >
```

```
<!-- Data Model -->
<!-- ########## -->

<!ELEMENT DataModel (DataMatrix*, Annotation?) >
<!ATTLIST DataModel name               CDATA #REQUIRED
                    functionPoolRef    CDATA #IMPLIED >


<!-- Data Matrix -->
<!-- ########### -->

<!ELEMENT DataMatrix (DataField*, MatrixTreatment*, Conditions*, Annotation?) >
<!ATTLIST DataMatrix name CDATA #REQUIRED >

<!ELEMENT Conditions (Part*,Annotation?) >
<!-- <!ATTLIST Conditions EMPTY > -->

<!ELEMENT Part EMPTY >
<!ATTLIST Part      column1 CDATA #REQUIRED
                    column2 CDATA #REQUIRED >

<!ELEMENT DataField (FieldProperties) >
<!ATTLIST DataField name CDATA #REQUIRED >


<!-- Field Properties -->
<!-- ################ -->

<!ELEMENT FieldProperties (Annotation?) >
<!ATTLIST FieldProperties atomicity    (%ATOMICITY;)       #REQUIRED
                          dataType     (%DATA-TYPE;)       #REQUIRED
                          column         CDATA             #REQUIRED
                          granularity  (%GRANULARITY;)     #IMPLIED
                          scale        (%SCALE;)           #IMPLIED
                          length       (%INT-NUMBER;)      #IMPLIED >

<!-- Matrix Treatment -->
<!-- ################ -->

<!ELEMENT MatrixTreatment ((VIMEOValues | ValueTreatment | Transformation )*,
                          Annotation?) >
<!ATTLIST MatrixTreatment name CDATA #REQUIRED >

<!ELEMENT VIMEOValues (FieldRef, VIMEOFunction) >
<!ATTLIST VIMEOValues     defTagAssign (%DEF-TAG-ASSIGN;) "pessimistic"
                          useForDefAssign (%USE-FOR-DEF-ASSIGN;) "both" >

<!ELEMENT VIMEOFunction (FunctionCall) >
<!ATTLIST VIMEOFunction id %REAL-NUMBER; #REQUIRED >

<!ELEMENT ValueTreatment (FieldRef*, (%VALUES;)*, TreatAs*) >
<!ATTLIST    ValueTreatment valueIntType      (%VALUE-INT-TYPE;)  #REQUIRED
             treatment                        (%VALUE-TREATMENT;) #REQUIRED
             target                           (%TREATMENT-TARGET;)#IMPLIED >

<!ELEMENT Normalization (Annotation?)>
<!ATTLIST Normalization type (%NORM-TYPE;) #REQUIRED >

<!ELEMENT Clipping (Annotation?)>
<!ATTLIST Clipping fraction      %REAL-NUMBER;       #REQUIRED
                                 type (%CLIP-TYPE;)  #REQUIRED >

<!ELEMENT UserValues ANY >
<!ATTLIST UserValues column CDATA #REQUIRED >

<!ELEMENT Discretization (UserValues*)>
<!ATTLIST Discretization numBins %INT-NUMBER; #REQUIRED
                                  type (%BIN-TYPE;) #REQUIRED >
```

```
<!ELEMENT Transformation (FieldRef,(Normalization | Discretization | Clipping)) >
<!ATTLIST Transformation target (%TREATMENT-TARGET;) #IMPLIED >

<!ELEMENT TreatAs (%EXPRESSION;)* >
<!ATTLIST    TreatAs      id %REAL-NUMBER; #REQUIRED
             useValues    (%USE-VALUES;) "substitutes" >


<!-- Data Mining Model -->
<!-- ################# -->

<!ELEMENT DataMiningModel (DataMiningMatrix*, Annotation?) >
<!ATTLIST DataMiningModel name               CDATA #REQUIRED
                          dataModelRef       CDATA #REQUIRED
                          functionPoolRef    CDATA #IMPLIED >

<!-- Data Mining Matrix -->
<!-- ################## -->

<!ELEMENT DataMiningMatrix ((UseMatrix | (Query, UseMatrix*)?),
                                DataMiningField*, MatrixTreatment*, Annotation?) >
<!ATTLIST DataMiningMatrix name CDATA #REQUIRED >

<!ELEMENT Query ANY >
<!ATTLIST Query     language           CDATA #REQUIRED
                    languageVersion    CDATA #IMPLIED >


<!ELEMENT UseMatrix (UseField*) >
<!ATTLIST UseMatrix       matrixRef          CDATA #REQUIRED
                          matrixTreatmentRef  CDATA #IMPLIED >

<!ELEMENT UseField EMPTY >
<!ATTLIST UseField name CDATA #REQUIRED >

<!ELEMENT DataMiningField (FieldProperties, FieldExpression*) >
<!ATTLIST DataMiningField name CDATA #REQUIRED >

<!ELEMENT FieldExpression (FunctionCall) >
<!ATTLIST FieldExpression id %REAL-NUMBER; #REQUIRED >


<!-- Domain Knowledge -->
<!-- ################ -->

<!ELEMENT DomainKnowledge ANY >
<!ATTLIST    DomainKnowledge language  CDATA #REQUIRED
             languageVersion           CDATA #IMPLIED
             type                      CDATA #IMPLIED >


<!-- Data Mining Task -->
<!-- ################ -->

<!ELEMENT DataMiningTask ANY >
<!ATTLIST DataMiningTask  name               CDATA #REQUIRED
                          language           CDATA #REQUIRED
                          languageVersion    CDATA #IMPLIED
                          type               CDATA #IMPLIED >

<!-- Knowledge -->
<!-- ######### -->

<!ELEMENT Knowledge ANY >
<!ATTLIST Knowledge       name               CDATA #REQUIRED
                          language           CDATA #REQUIRED
                          languageVersion    CDATA #IMPLIED
                          type               CDATA #IMPLIED >
```

Příloha B: Obsah přiloženého CD

- ./dist – spustitelná distribuce aplikace
- ./doc – diplomová práce
- ./doc/javadoc – Javadoc
- ./netbeans – instalace NetBeans Platform na které byl projekt vyvíjen
- ./workspace – zdrojové kódy aplikace
- sample.sql – SQL skript pro vytvoření databázové tabulky, která je použita v kap. 7.