VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

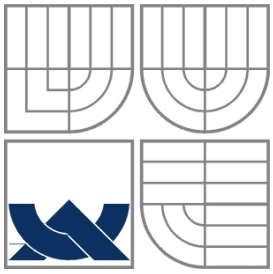
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GRAFICKÁ NADSTAVBA PRO SYSTÉM ZÍSKÁVÁNÍ ZNALOSTÍ

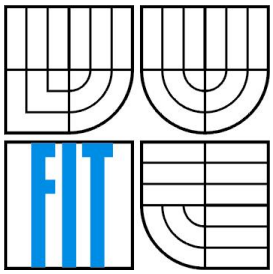
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Michal Gálet
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GRAFICKÁ NÁDSTAVBA PRO SYSTÉM ZÍSKÁVÁNÍ ZNALOSTÍ

GRAPHICAL USER INTERFACE FOR DATA MINING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Michal Gálet

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. Jaroslav Zendulka, CSc.

BRNO 2007

Zadání

Grafická nadstavba pro systém získávání znalostí

Vedoucí:

Zendulka Jaroslav, doc. Ing., CSc., UIFS FIT VUT

Oponent:

Burget Radek, Ing., Ph.D., UIFS FIT VUT

Přihlášen:

Gálet Michal, Bc.

Zadání:

1. Seznamte se rámcově s problematikou získávání znalostí z databází.
2. Seznamte se s prototypem systému pro získávání znalostí vytvořeném na FIT VUT.
3. Seznamte se se způsobem zadávání kroků procesu získávání znalostí v podobě blokového schématu, např. u produktu SAS Enterprise Miner.
4. Navrhněte koncepci řešení komponenty systému, která umožní interaktivně vytvářet blokové schéma procesu získávání znalostí.
5. Po dohodě s vedoucím diplomové práce realizujte vymezenou část navrženého řešení a ověřte jeho funkčnost.
6. Zhodnoťte dosažené výsledky a diskutujte další postup řešení navržené komponenty.

Část požadovaná pro obhajobu SP:

Body 1 až 4.

Kategorie:

Databáze

Literatura:

- Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Second Edition. Elsevier Inc., 2006, 770 p., Chapter 1.
- Data Mining Using Enterprise Miner Software: A Case Study Approach First Edition.
- Michele.J.: Předzpracování dat pro dolování. Diplomová práce FIT VUT, 2004.
- Forgáč, M.: Systém pro získávání znalostí z databází. Ročníkový projekt FIT VUT, 2005.
- Doležal, J.: Jádro systému pro dolování dat v prostředí Oracle. Diplomová práce FIT VUT, 2006.

Licenční smlouva

Licenční smlouva je uložena v archívu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Předmětem této práce je vytvoření grafické nadstavby nad existujícím systémem pro získávání znalostí z databází. Grafické uživatelské rozhraní umožňuje uživateli specifikovat dolovací úlohu pomocí blokového schématu, kde každý element představuje určitý krok v procesu dolování dat. Řešení je postaveno na NetBeans platformě a vedle grafické editace umožňuje práci koncipovat do projektů, spouštět dolovací úlohu, zobrazovat výsledky a ukládat DMSL dokumenty do souborů. Systém je navržen s důrazem na modularitu, rozšiřitelnost a možnost provádění automatických aktualizací aplikace.

Klíčová slova

Získávání znalostí, DMSL, Oracle Data Mining, Java, NetBeans Platform, NetBeans Visual Library

Abstract

The issue of this MSc. project is a design and implementation of a graphical user interface for a data mining system. The application is a front-end for an existing data mining library and allows the user to visually define the data mining process using a graphical editor and a component palette. The solution is built on a NetBeans Rich Client Platform and takes many benefits from using rich components available in this platform. The user may define his work in the projects, edit and persist DMSL documents, run the mining tasks and view the mining results. The system is designed to emphasize the modularity and extensibility of the solution.

Keywords

Data Mining, DMSL, Oracle Data Mining, Java, NetBeans Platform, NetBeans Visual Library

Citace

Gálet Michal: Grafická nadstavba pro systém získávání znalostí. Brno, 2007, diplomová práce, FIT VUT v Brně.

Grafická nadstavba pro systém získávání znalostí

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.

……
Jméno Příjmení
Datum

Poděkování

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

Obsah

1 Úvod

V současné době všechny informační systémy pro ukládání perzistentních dat používají databázové systémy (DBS). S rostoucími nároky na tyto informační systémy se databázová vrstva stává čím dál komplexnější a může obsahovat informace, které jsou skryté anebo nedostupné jednoduchými dotazy. Vzniká tak potřeba pro vytvoření specializovaných nástrojů, které tyto skryté a potenciálně zajímavé informace dokážou extrahovat.

Nejstaršími databázovými systémy byly databáze se síťovým modelem dat, kde byly relace realizovány přímými fyzickými adresami. V případě chyby byly nekonzistentní záznamy neopravitelné. V 80. letech byly vyvinuty relační databázové systémy, které se používají dodnes. Byl vyvinut a standardizován dotazovací jazyk SQL pro definici, modifikaci a získávání dat. S příchodem objektově orientovaných jazyků došlo k potřebě ukládání dat založených na objektovém modelu dat. Současné objektové databáze nejsou zatím populární, protože relační model je léty prověřený a robustní. Výrobci relačních DBS obohacují své produkty o objektovou vrstvu a nabízejí tak objektový přístup. Dochází k vývoji specializovaných modulů pro určité aplikace, jako je zpracování temporálních dat, multimediální databáze, prostorové nebo vědecké databáze.

Prudký rozvoj v této oblasti a vysoký nárůst objemu uložených dat byl důvodem pro vyvinutí nástrojů a technologií pro analýzu těchto dat. V 90. letech se objevují OLAP (Online Transactional Processing) nástroje pro budování tzv. datových skladů, které dokážou shromažďovat, předzpracovat a analyzovat data z tzv. operačních databází. Produkční data z operačních databází a historická data se přesunuly do datových skladů, které obsahují funkce pro analýzu dat a kladou nároky na rychlost odezvy systému na úkor přesnosti. Na rozdíl od OLAP nástrojů, které uživatel používá interaktivně a analýzu definuje „ručně“, jsou systémy pro získávání znalostí z databází zaměřeny na automatizované nalezení zajímavých a potenciálně užitečných vzorků v datech. Vzniká tedy potřeba vyvinutí graficky přitažlivých aplikací s jednoduchým uživatelským rozhraním, které umožní zadávat dolovací úlohu jednoduchou formou.

Firma Oracle vyvinula pro svůj databázový systém modul pro podporu získávání znalostí z databází nazvaný Oracle Data Mining (ODM). Obsahuje nástroje pro předzpracování dat a provádění dolovacích úloh. Proces dolování v datech se provádí na straně serveru a přístup z jazyka Java je od verze 10.2 realizován pomocí standardizovaného rozhraní Java Data Mining (JSR-73).

Pro definici dolovací úlohy je na fakultě a v této práci používán jazyk DMSL založený na jazyku XML, který dovoluje definovat celý dolovací proces. Standardizace jazyka pro dolování dat by měla stabilizovat vývoj nástrojů a tím zjednodušit a zefektivnit jejich implementaci. Tento jazyk umožňuje definovat data pro dolování, typ dolovací úlohy, měření zajímavosti vytěžených dat a vizualizaci výsledků dolování.

Cílem této práce je vytvoření grafického uživatelského rozhraní pro systém získávání znalostí z databází, který je postaven na jazyku DMSL. Systém by měl být realizován na platformě Java a využívat jádro pro dolování implementované panem Jindřichem Doležalem v předchozím roce (Jádro systému pro dolování z dat v prostředí Oracle [3]). Aplikace by měla umožnit uživateli jednoduchým způsobem pomocí blokového schématu definovat a provést úlohu dolování dat v prostředí Oracle.

Kapitola 2 seznámí čtenáře s problematikou získávání znalostí z databází včetně předzpracování dat a používaných algoritmů. Kapitola 3 popisuje koncepci DMSL jazyka pro definici dolovací úlohy a uvádí strukturu DMSL dokumentů. Kapitola 4 popisuje stávající systémy pro dolování dat vyvíjené na fakultě informačních technologií a zaměřuje se na systém postavený na ODM, který je podkladem pro tuto práci. Kapitola 5 čtenáři představí ideu Rich Client Platform se zaměřením na platformu NetBeans. V této kapitole jsou popsány hlavní části platformy použité při řešení. V kapitole 6 je popsán obecný návrh systému, který řeší grafickou nadstavbu pro jádro využívající ODM. V této kapitole je popsána analýza a konceptuální model řešení. V kapitole 7 jsou popsány zajímavé části programového řešení a to, jak věci fungují. Kapitola 8 podrobně uvádí postup vytvoření dolovacího modulu a jeho následné integrace do výsledné aplikace. Kapitola 9 zhodnocuje výsledné řešení, přínos použití technologií, které byly vybrány pro řešení a návrh pro další vývoj aplikace.

2 Získávání znalostí z databází

V této kapitole popíšeme proces pro získávání znalostí z databází, metody a techniky, které se používají v tomto oboru informačních technologií. Pochopení této problematiky je důležité pro analýzu, specifikaci a implementaci aplikace, která je předmětem této práce.

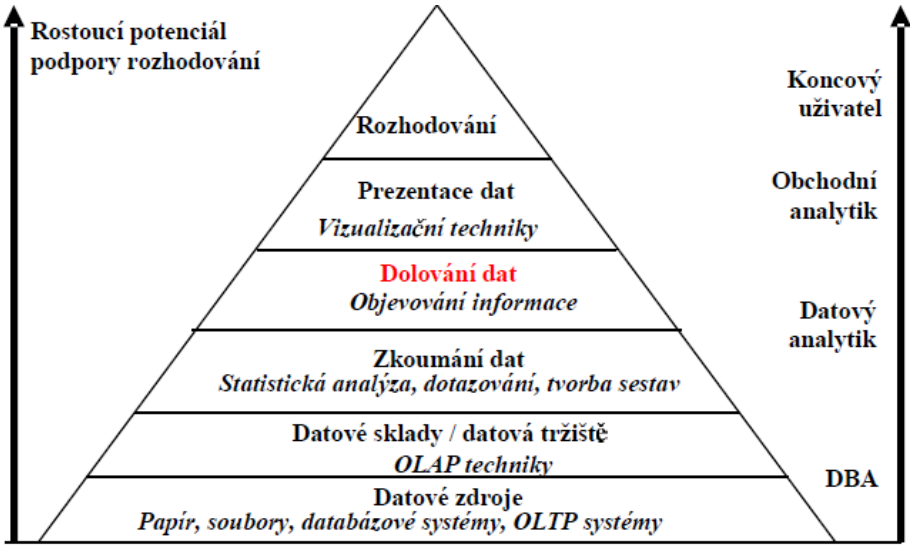
Můžeme říci, že získávání znalostí z databází je extrakce (neboli „dolování“) zajímavých (netriviálních, skrytých, dříve neznámých a potenciálně užitečných) modelů dat a vzorů z velkých objemů dat. Tyto modely a vzory reprezentují znalosti získané z dat. [1]

Často se můžeme setkat s pojmem dolování dat, který se používá jako synonymum k získávání znalostí z databází, který vychází z anglického Knowledge Discovery in Databases (KDD). Tato oblast informačních technologií se začala rozvíjet na začátku 90. let minulého století pod pojmem information discovery, information harvesting nebo knowledge extraction. Hlavním rysem této oblasti je netriviálnost získaných informací. Znamená to, že nejde o informaci, kterou lze získat SQL dotazem, ale je nutné použít nějaký inteligentní postup.

Příklad aplikace získávání znalostí:

- **Analýza trhu** – nalezení zákazníků s podobnými vlastnostmi (záliby, výše příjmu apod.). Zdrojem pro dolování je většinou databáze transakcí prováděných kreditními nebo úvěrovými kartami.
- **Analýza nákupního košíku** – nalezení tzv. asociačních pravidel, tj. zboží, které zákazníci kupují společně. Užitečná je časová analýza pro sledování trendů.
- **Finanční analýza a řízení rizik** – jedná se o predikci vývoje cen a predikci rizik při poskytnutí půjčky.
- **Detekce podvodů** – sledování neobvyklých a odlišných vzorů dat. Časté uplatnění je pro detekci pojistných podvodů nebo vyhledávání profesionálních pacientů ve zdravotnictví.

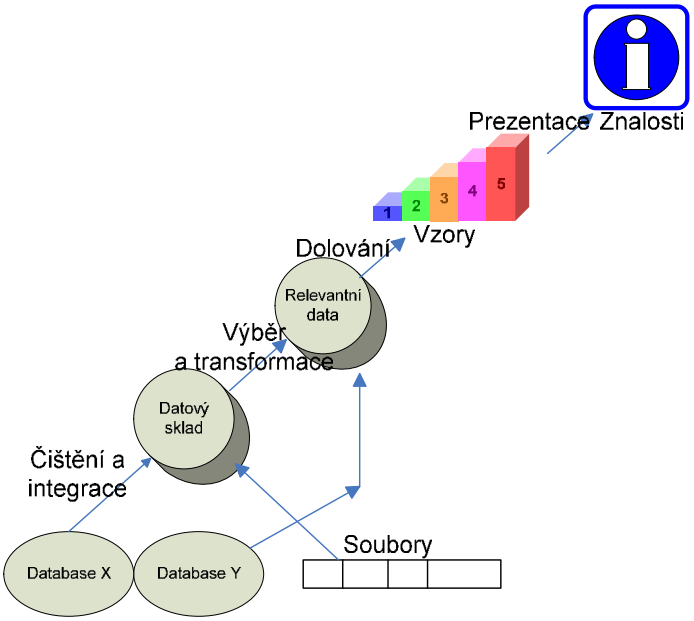
V současné době nabývá na významnosti pojem *business inteligence*. Jedná se o metody a nástroje, které slouží pro podporu rozhodování v business sféře. Na obr. 1 je znázorněn význam dolování dat v této oblasti. S rostoucí abstrakcí ubývá dostupnost prostředků. Osa na pravé straně zobrazuje role uživatelů, kteří přicházejí do styku s příslušnými nástroji. Tento obor by měl být v budoucnu motivací pro vytváření nástrojů podobných jako je řešení této diplomové práce s důrazem na jednoduchost uživatelského rozhraní a efektivitu práce.



Obrázek 1 - Vztah dolování dat a business inteligence (převzato z [1])

2.1 Proces získávání znalostí

Získávání znalostí je proces, který se provádí v několika krocích. Tyto kroky se mohou provádět v iteracích. Jelikož dnešní databáze jsou rozsáhlé a uchovávají data nejrůznějšího druhu, často bývají uložená data tzv. zašumělá nebo nekonzistentní. Častým problémem pro dolování dat je chybějící hodnota atributu. Tyto problémy mohou vznikat vlivem lidského faktoru, migrací dat mezi databázemi nebo integrací z různých úložišť. Podstatným krokem v tomto procesu je proto čištění a integrace dat.



Obrázek 2 - Proces získávání znalostí

Čištění, integrace, výběr a transformace se souhrnně označují jako předzpracování dat. Čištění dat (Data Cleansing) slouží k odstranění zašumělých a nekonzistentních dat. Integrace dat (Data Integration) je krokem, který spojuje data z různých zdrojů do jednoho zdroje, typicky do datového skladu. Transformace dat (Data Transformations) provádí transformaci dat a upravuje data tak, aby byla vhodná pro dolovací metody (např. normalizace hodnot může zlepšit výsledek dolování). Výběr dat (Data Reduction) zmenšuje objem dat pro dolování, například pomocí agregace, shlukováním, nebo odstraněním nezajímavých atributů.

Předzpracování dat může výrazně zlepšit kvalitu vydolovaných vzorů a tím i výsledek dolování. Obecně platí, že data, která chceme použít pro dolování, musí co nejpřesněji modelovat realitu, kterou reprezentují, být konzistentní, důvěrná, aktuální, dostupná a prospěšná pro danou úlohu dolování.

2.1.1 Čištění dat

Jedná se o odstranění problémů nekompletních, zašumělých nebo chybějících hodnot. Důvodem těchto problémů může být porucha na přístroji pro sběr dat, lidský faktor, chyba komunikačního kanálu apod. Úkolem čištění dat je doplnění chybějících atributů, vyhlazení zašumělých hodnot, odstranění extrémních hodnot a vyřešená konzistence.

Tento proces není jednoprůchodový, ale iterativní. Určitý krok čištění může mít za následek opakování některého předchozího kroku. Například při odstranění nekonzistence, může vzniknout potřeba pro odstranění chybějící hodnoty.

2.1.1.1 Chybějící hodnota

Velmi častým problémem je chybějící hodnota atributu, který však může reprezentovat důležité informace pro proces dolování. Většinou se jedná o neklíčové atributy, které dovolují uchovávat prázdnou NULL hodnotu. Existuje několik metod pro ošetření chybějících hodnot.

- **Ignorovat celý řádek tabulky** – tato metoda je vhodná pouze v případě, pokud v prvku relace chybí některé další atributy (nelze odvodit chybějící hodnotu atributu) nebo v případě čištění dat pro klasifikaci.
- **Manuální doplnění chybějící hodnoty** – metoda by byla vhodná, ale kvůli rozsahu dat je prakticky nereálná. Uživatel by rovněž musel mít znalosti, které by uplatnil při nahrazování.
- **Automatické doplnění globální konstantou** – jedná se o obdobu hodnoty NULL v databázích. Používá se hodnota mimo rozsah platných hodnot daného atributu (např. 0 nebo ∞ pro numerický atribut). Pokud by výskyt této odlehlé hodnoty byl nízký, algoritmus pro dolování jí může ignorovat, ale v případě častého výskytu může tato metoda negativně ovlivnit výsledek dolování.

- **Použití průměrné hodnoty atributu** – hodnota pro automatické doplňování se vypočítá jako průměr z hodnot atributu v ostatních prvcích relace (řádcích tabulky).
- **Použití průměrné hodnoty n-tic stejné třídy** – je použita průměrná hodnota atributu z relací, které patří do stejné třídy. Například v případě třídy *vzdělání="vysokoškolské"* se použije průměrná hodnota atributu *příjem* z průměru hodnot relací, které spadají do této třídy.
- **Doplnění nejpravděpodobnější hodnotou** – tato hodnota může být vypočtena použitím odvozovacích nástrojů jako je Bayesovská klasifikace, regrese apod. Jedná se vlastně o klasifikaci nebo predikci s doplňovaným atributem jako cílem. Metoda se jeví jako nejlepší, protože nejvíce zohledňuje okolní informace.

2.1.1.2 Šum v datech

Jedná se o náhodné chyby v datech. Důvodů pro zašuměné hodnoty může být více. Většinou jde o chyby vzniklé poruchou na zařízení pro sběr dat, lidským faktorem, poruchou hardware nebo použitím různých formátů pro kódování. Techniky, které provádějí vyhlazení dat, jsou uvedeny níže.

- **Plnění** – vyhlazování numerických dat je prováděno tak, že setříděná posloupnost zohledňuje hodnoty v blízkém okolí. Tato technika provádí lokální vyhlazení. Setříděné hodnoty se rozdělí do tzv. *košů* stejné frekvence. Hodnoty v koších se pak nahradí průměrem koše, mediánem koše nebo hraniční hodnotou koše.
- **Regrese** – data se nahrazují hodnotami, které jsou dány regresní křivkou. Lze použít lineární nebo vícenásobnou lineární regresi.
- **Rozdělení do shluků (tzv. shlukování)** – nalezení odlehlých hodnot, které nelze zařadit do žádného shluku.

Metody pro odšumění dat můžeme rovněž chápat i jako metody pro redukci dat. Lze je použít i pro *diskretizaci* hodnot. Požívá se metoda pro rozčlenění na intervaly stejné šířky nebo rozčlenění na intervaly stejné hloubky.

2.1.2 Integrace a transformace dat

Jedná se o spojení dat z několika nezávislých úložišť do jednoho a vytvoření jednoho konzistentního zdroje.

V případě integrace dat jde o nalezení atributů různých zdrojů, které k sobě patří. Například atribut pro identifikaci zboží může být v jedné databázi (tabulce) nazván *item_id* v druhé jako *iid*. Tento problém se označuje jako *konflikt schématu*. Další podstatnou částí procesu integrace je odstranění redundance. To znamená odstranění dat, která jsou duplicitní ale i taková, která se dají odvodit z jiných uložených dat. Redundance se dají detekovat z metadat, ale v datech se může vyskytnout i silná korelace, která se detekuje tzv. korelační analýzou. Dalším problémem, s kterým se

musí integrace vypořádat, je konflikt hodnot, kdy jsou odpovídající si hodnoty atributů různé, a konflikt identifikace, kdy v různých úložištích je identifikace objektů různá (např. rodné číslo a pořadové číslo u osob).

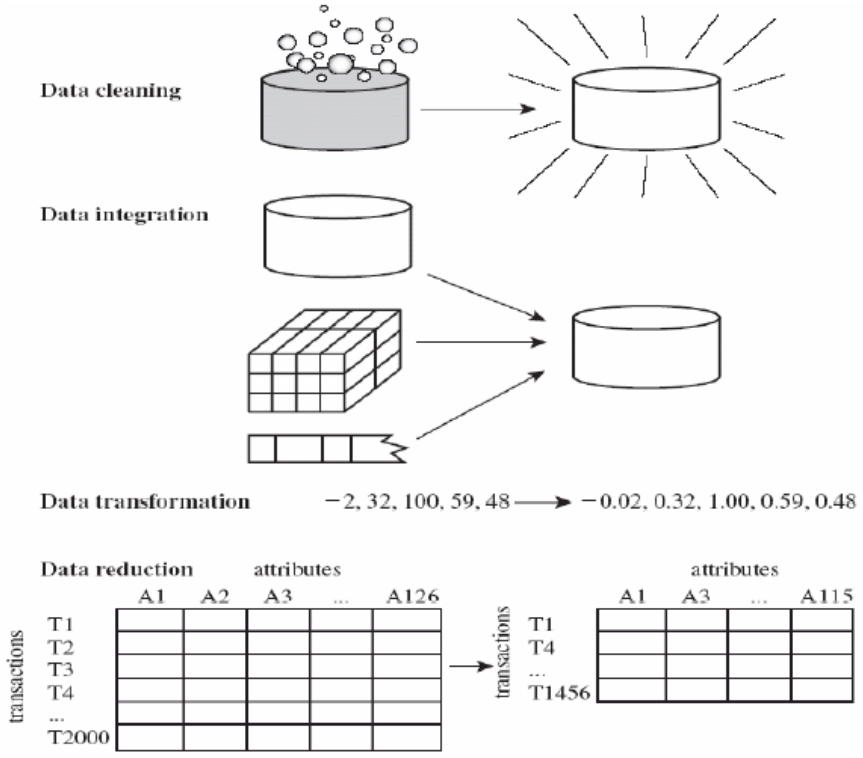
Ve fázi transformace se data transformují tak, aby lépe vyhovovala dolovacím metodám a charakteru dolovací úlohy. Operace, které můžou být zahrnuty ve fázi transformace:

- **Vyhlazení** – odstranění šumu jako v kap. 2.1.1.2.
- **Agregace** – aplikují se sumační nebo agregační funkce typické pro plnění datového skladu. Obvykle se provádí při plnění datové kostky pro analýzu na vyšší úrovni abstrakce a slouží rovněž jako redukce dat.
- **Generalizace** – nahrazení hodnoty atributu jejich obecnější hodnotou jako u hierarchie konceptů.
- **Normalizace dat** – jde o transformaci hodnot tak, že spadají do určitého intervalu hodnot (typicky je to <0.0, 1.0>). Příklad normalizace je uveden na obr. č. 3. Normalizace se provádí typicky u neuronových sítí, shlukování a metody nejbližšího souseda, protože by mohlo dojít k negativnímu ovlivnění výsledku dolování. Normalizace většinou zabrání tomu, aby atribut s velkým rozsahem hodnot překryl svým významem atributy s menším rozsahem hodnot. Existuje celá řada metod pro normalizaci, ale nejčastější jsou min-max normalizace (lineární transformace), z-score (normalizace na základě průměru a odchylky) a dekadickou změnou měřítka (posunutí desetinné čárky tak, aby obor hodnot ležel v požadovaném rozsahu).

2.1.3 Redukce dat

Jelikož je dolování nad velkým množstvím dat časově a výpočetně náročné, je žádoucí zdrojová data vhodným způsobem redukovat. Vhodným způsobem rozumíme tak, že informace obsažená v datech se nezmění nebo nezmění se charakter dat a je zachována integrita dat. Používá se zpravidla 5 technik pro redukci dat:

- **Agregace datové kostky** – sumarizace původních dat. Technika odpovídá operaci roll-up v prostředí datového skladu.
- **Odstranění dimenze** – provádí se, pokud je dimenze pro analýzu nepodstatná nebo málo podstatná. Klíčovým faktorem je správné zvolení množiny atributů pro redukci. Je nutné znát doménu a význam uložených dat včetně závislostí.
- **Redukce počtu hodnot** – data jsou nahrazena modelem a reprezentována parametry.
- **Komprese dat** – ztrátová či bezztrátová komprese dat.
- **Diskretizace a použití konceptuální hierarchie** – hodnoty atributů jsou nahrazeny hodnotami z intervalů nebo hodnotami z nějaké konceptuální hierarchie. Redukuje se počet různých hodnot atributů.



Obrázek 3 - Úlohy předzpracování dat (převzato z [2])

2.2 Typy dolovacích úloh

Řada metod používaných v problematice získávání znalostí vychází z umělé inteligence. Úlohy se rozdělují na 2 typy: deskriptivní a prediktivní. Deskriptivní funkce charakterizují a popisují data podle jejich vlastností uložených v databázi. Prediktivní funkce pracují tak, že na základě dat v databázi jsou schopny předpovědět vlastnosti dat nově příchozích.

2.2.1 Charakterizace a diskriminace

Charakterizace dat provádí sumarizaci obecných vlastností analyzované třídy. Data, která odpovídají třídě, lze z databáze vybrat jednoduchým dotazem. Výsledkem může být například charakteristika určitého zboží a zjištění druhu zákazníka, který si jej kupuje.

Diskriminace naopak porovnává atributy cílové třídy s odpovídajícími atributy jiné třídy a hledá, v čem se nejvíce liší.

2.2.2 Frekventované vzory

Jedná se o nalezení vzorů, které se vyskytují v datech často. Úloha vede k odhalení zajímavých korelací nebo použití asociační analýzy a vyhledávání asociačních pravidel ve tvaru $X \Rightarrow Y$, kde X a Y jsou tvrzení týkající se hodnot atributů. Asociační pravidlo nám říká, jaká je pravděpodobnost

výskytu prvků v transakci. Příkladem může být pravidlo, že zákazníci, kteří si koupí DVD rekordér si koupí i sadu DVD médií.

$$kupuje(X, rekordér) \Rightarrow kupuje(X, DVD\_media)[podpora = 5\%, spolehlivost = 50\%]$$

Podpora a spolehlivost jsou nejčastěji používané míry k vyjádření významnosti zastoupení frekventovaného vzoru v analyzovaných datech a poměřování zajímavosti dolovaných asociačních pravidel. V tomto případě podpora 5 % říká, že 5 % uživatelů nakoupilo společně rekordér a DVD média a v 50 % nákupů, ve kterých si zákazníci koupili rekordér, si zároveň koupili DVD média. Čili údaj 50 % spolehlivosti vyjadřuje významnost zastoupení položek na pravé straně pravidla.

2.2.3 Klasifikace a predikce

Jedná se o prediktivní dolovací úlohy. Cílem klasifikace je nalezení pravidel, která rozlišují a zároveň popisují třídy dat. Tato pravidla se pak použijí k predikci třídy objektu, jehož zařazení neznáme. Model je sestavován pomocí podmínkových pravidel, rozhodovacích stromů nebo jiných prostředků.

Proces klasifikace se sestává ze tří kroků:

1. **Trénování** – na základě trénovací množiny je vytvořen model pro klasifikaci. Tato fáze se označuje také jako učení.
2. **Testování** – ověření kvality modelu testováním pomocí testovací množiny.
3. **Aplikace** – použití modelu ke klasifikaci dat, jejichž třídu neznáme.

Klasifikace se používá k predikci diskrétních tříd. Oproti tomu predikce předpovídá hodnoty spojitých atributů. V tomto případě předpovídáme numerickou nedostupnou hodnotu. Nejčastější metodou predikce je regresní analýza.

2.2.4 Shlukování

Shluková analýza (Cluster Analysis) na rozdíl od klasifikace a predikce analyzuje objekty bez znalosti přiřazení do tříd. Cílem je nalézt třídy objektů, které mají co nejvíce společného tak, aby se objekty různých tříd co nejvíce lišily. Nalezené třídy mají podobu tzv. shluků.

2.2.5 Analýza odlehlých objektů

Jde o nalezení objektů, které se nějakým způsobem významně odlišují od ostatních. Takové datové objekty se nazývají odlehlé (outlier). Tato analýza může například odhalit podvodné zneužití kreditních karet, extrémně velké nebo podezřelé nákupy.

2.2.6 Evoluční analýza

Jedná se o analýzu dat v čase, která hledá modely trendů a jejich vývoj. Zkoumá se vývoj a rychlost změn v čase anebo pravidelnost dat v jistých časových intervalech. Efektivní využití této metody může být při analýze průběhu hodnot akcií a rozhodování o investicích.

2.3 Analýza nalezených vzorů

Systém pro získávání znalostí z databází je schopen generovat obrovské množství vzorů nebo pravidel. Vzniká tak důležitá otázka zajímavosti nalezených vzorů. V praxi je zajímavá pro koncového uživatele pouze malá část. Zajímavé vzory nebo pravidla pak představují znalost.

Aby byl vydolovaný vzor pro uživatele zajímavý, musí mít 4 základní vlastnosti, které určují míru zajímavosti:

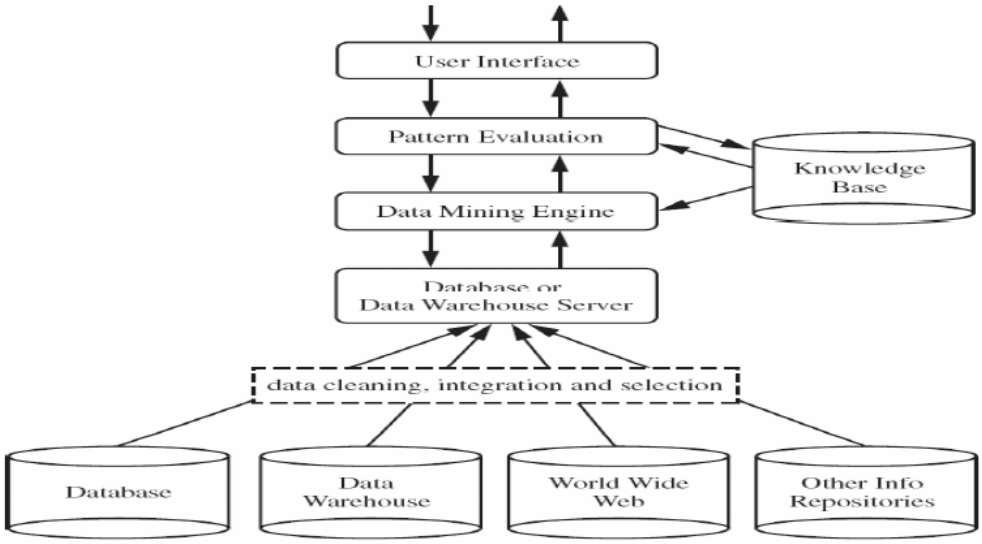
- **Srozumitelnost** – vzor musí být srozumitelný pro člověka
- **Platnost** – pro nová nebo testovací data
- **Užitečnost** – vzor musí mít reálnou užitečnost
- **Novost** – přináší nové poznatky

Užitečným vzorem může být i vzor, který validuje hypotézu, kterou se snaží uživatel potvrdit. Pro určení míry zajímavosti existují objektivní a subjektivní metody. Objektivní metody jsou založeny na struktuře objevovaných vzorů a statistických údajích k nim vztažených. Mezi tyto metody patří dříve zmíněné frekventované vzory a asociační pravidla (míra podpory a spolehlivosti).

Subjektivní míry by měly doplňovat objektivní, které samy o sobě nejsou dostatečným kriteriem pro vyhodnocení zajímavosti. Mezi taková kriteria patří např. novost, neočekávanost apod.

2.4 Architektura systému pro dolování dat

Typická architektura systému pro dolování dat vychází z výše popsaných kroků, které definují proces získávání znalostí.



Obrázek 4 - Typická architektura systému pro dolování dat (převzato z [2])

- **Datové zdroje** – aplikační vrstva pro získání dat z různých zdrojů

- **Databáze znalostí** – většinou obsahuje prahové hodnoty, pomocí kterých uživatel řídí dolovací algoritmus. Může rovněž být použita pro hodnocení zajímavosti nalezených vzorů.
- **Dolovací mechanismus** – jádro dolovacího systému. Skládá se z řady modulů implementujících algoritmy dolovacích úloh.
- **Modul pro hodnocení vydolovaných vzorů** – vyhodnocuje zajímavost vzorů a modelů. Často bývá integrován do samotného dolovacího mechanismu tak, aby eliminoval nezajímavé vzory v rané fázi.

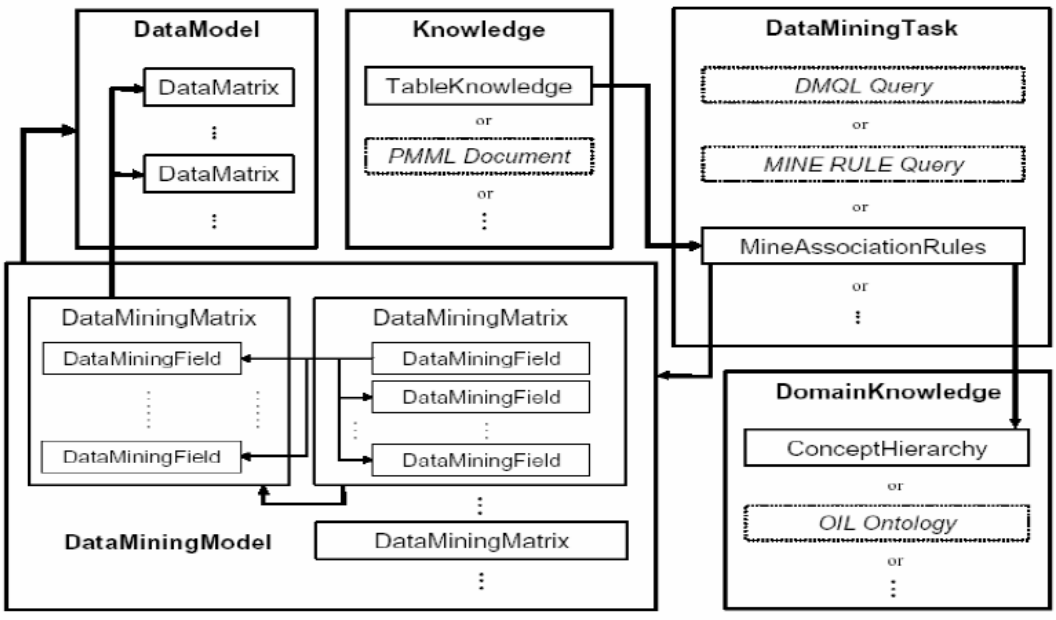
Uživatelské rozhraní – jedná se o komponentu systému, která zajišťuje komunikaci mezi uživatelem a celým systémem. Tato část by neměla být podceňována, protože často je první částí hodnocení kvality celého systému. Umožňuje uživateli definovat jednotlivé kroky procesu dolování, prohlížet data, která budou zdrojem pro dolování, získat statistické údaje o datech, zvolit modul (metodu) dolování apod. Důležitou součástí je vizualizace, popř. export vydolovaných vzorů a znalostí. Důraz by měl být kladen na intuitivnost uživatelského rozhraní.

3 Jazyk DMSL

Jazyk DMSL (Data Mining Specification Language) je jazyk postavený na bázi jazyka XML, který dovoluje definovat celý dolovací proces, od definice vstupních dat až po prezentaci vydolovaných znalostí. Jazyk DMSL klade důraz na proces předzpracování dat a umožňuje oproti ostatním dolovacím jazykům jej detailně specifikovat vlastními prostředky. Výhodou použití jazyka XML je snadná rozšiřitelnost o nové prvky a deskriptivní vlastnost zápisu (čitelný i pro člověka). Jazyk je obecně zaměřený a dovoluje např. definovat různé zdroje dat, od obyčejných souborů až po databáze.

Struktura jazyka se skládá z 5-ti částí:

- **Model dat (Data Model)** – reprezentuje vstupní data.
- **Dolovací model (Data Mining Model)** – zahrnuje datový model a transformace, které se na něm provádějí.
- **Znalosti o doméně (Domain Knowledge)** – model obsahující znalosti o datech, jako jsou vtahy, integritní omezení, obory hodnot apod.
- **Dolovací úloha (Data Mining Task)** – specifikuje dolovací funkce, které se budou provádět nad daty.
- **Znalosti (Knowledge)** – vydolované znalosti.
- **Funkce (Function Pool)** – definice funkcí.



Obrázek 5 - Vztahy DMSL elementů (převzato z [3])

Datový model obsahuje popis všech dat, která se použijí v dalších krocích procesu dolování. Dovoluje definovat data v textových souborech, v souborech tabulkových procesorů nebo

v databázích. Zajímavým prvkem je možnost data ohodnotit tzv. VIMEO hodnotami. Znamená to, že hodnota atributu může obsahovat příznak platný (**V**alid), neplatný (**I**nvalid), chybějící (**M**issing), prázdný (**E**mpty) nebo nezajímavý (**O**utlier). Tyto hodnoty jsou přiřazovány tzv. VIMEO funkcemi. Funkce se definují pomocí elementu *VIMEOValues* a jsou spojené s příslušným atributem. Hodnoty pak můžou být určitým způsobem zpracovány, a způsob zpracování se definuje elementem *ValueTreatment*.

Dolovací model obsahuje výběr z 1 až N datových matic. Atribut pro dolování může být odvozen z datového modelu nebo odvozen z funkce (příkladem může být odvození atributu *věk* z atributu *rodné číslo*). Dolovací model rovněž umožňuje definovat transformace.

Element pro doménové znalosti obsahuje znalosti o datech, jako jsou vtahy, integritní omezení, obory hodnot apod. Element dovoluje použít vlastní způsob definice, nejlépe v jazyku založeném na XML.

Element pro specifikaci dolovací úlohy je taktéž obecný a umožňuje definovat vlastní dolovací modul. Je důležité použít bohatý a rozšiřitelný jazyk, protože se jedná o stěžejní část procesu dolování.

FunctionPool element slouží k definici vlastních funkcí. Funkce mohou být interní nebo externí. Interní funkce mají hlavičku i tělo definované v rámci DMSL dokumentu. Externí funkce mají v dokumentu definovánu pouze hlavičku, kompletní definice je v externím zdroji.

4 Systém pro dolování dat vyvíjený na FIT

V předchozích letech byl na fakultě informačních technologií vyvíjen systém pro získávání znalostí z databází v rámci diplomových nebo ročníkových prací studentů. V současné době je systém kompletní a zahrnuje všechny kroky procesu dolování. Umožňuje výběr dat, předzpracování dat, provádění dolovací funkce a zobrazení výsledku. Existují dvě implementace těchto nástrojů. Obě jsou implementovány v prostředí jazyka Java a postaveny na jazyku DMSL, který představuje definici dolovací úlohy. Díky platformě Java jsou systémy přenositelné a využívají veškeré výhody tohoto řešení.

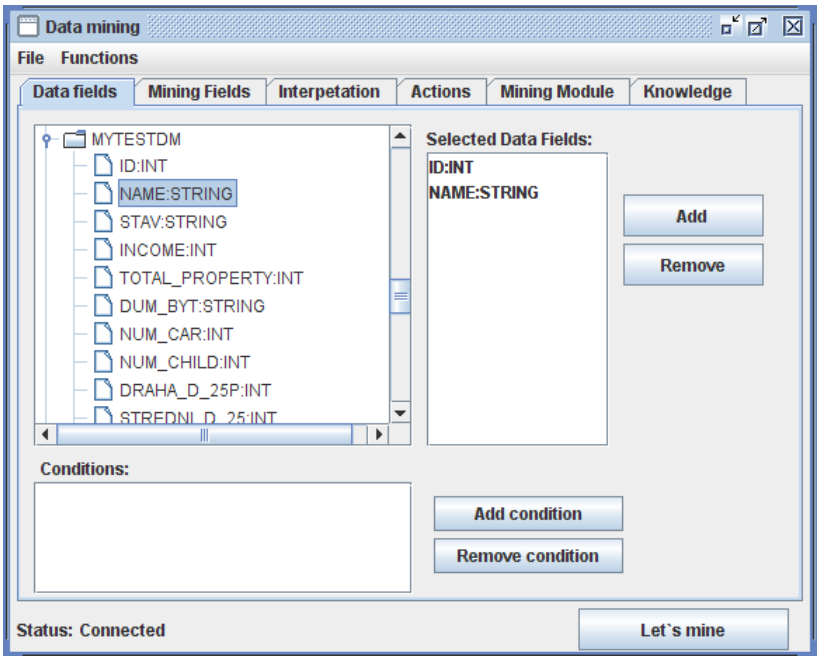
Prvním systémem je nástroj, který pracuje pouze s databází MySQL. MySQL neobsahuje žádnou podporu pro dolování dat, a veškerá komunikace se provádí standardním SQL jazykem přes JDBC rozhraní. Dolování je prováděno pomocí pomocných tabulek, které obsahují jak zdrojová data, tak i mezivýsledky dolování. Systém má modulární architekturu a mohou být do něj snadno integrovány další moduly implementující dolovací úlohy. Jelikož databáze slouží pouze jako úložiště, veškerá funkčnost systému pro získávání znalostí je implementována v tomto nástroji. Uživatel má možnost definovat vlastní funkce, jak interní, tak externí implementované v jazyce Java.

Druhým systémem je nástroj realizovaný v prostředí Oracle a využívá jeho modul pro podporu dolování dat Oracle Data Mining (ODM). Databáze Oracle je zde využita pro uložení dat, proces čištění a provádění dolovací úlohy. Předmětem této práce je rozšíření tohoto systému.

4.1 Jádro systému pro dolování dat v prostředí Oracle

V předchozím roce byl navrhnut a implementován nástroj pro dolování dat v prostředí Oracle (viz. [3]). Systém měl být realizován jako jádro, které používá DMSL pro definici dolování a komunikuje s databází Oracle, resp. s Oracle Data Mining modulem. Pro účely demonstrace bylo vytvořeno uživatelské rozhraní využívající jádro. Toto uživatelské rozhraní je inspirováno prvním systémem zmíněným výše. Do jaké míry je funkčnost rozhraní shodná, nebo zda byly nějaké části systému znovupoužity, není bohužel nikde dokumentováno.

Základním prvkem uživatelského rozhraní je Tabbed Pane komponenta, pomocí které uživatel přepíná panely odpovídající krokům v procesu dolování. Na obr. 6 je obrazovka této aplikace. Je patrné, že uživatelské rozhraní je pro uživatele neintuitivní a množství grafických komponent je matoucí.



Obrázek 6 - Nástroj pro dolování dat v prostředí Oracle

Následuje popis důležitých panelů aplikace:

- **Průzkumník databáze (Data Fields)** – zde se definují zdrojová data. V Tree komponentě jsou zobrazeny všechny tabulky schématu, ve kterém je uživatel připojen.
- **Dolovací pole (Mining Fields)** – definice datových polí (atributů), které budou použity dolovacím modulem. Uživatel zde definuje název pole, a jakým způsobem toto pole vznikne.
- **Explicitní interpretace (Interpretation)** – definice VIMEO funkcí, které určují zpracování jednotlivých hodnot atributů.
- **Zpracování (Actions)** – definice chování při zpracování VIMEO hodnot.
- **Dolovací modul (Mining Module)** – výběr dolovací úlohy a specifikace parametrů modulu.
- **Zobrazení výsledků (Knowledge)** – zobrazení výsledků dolování.

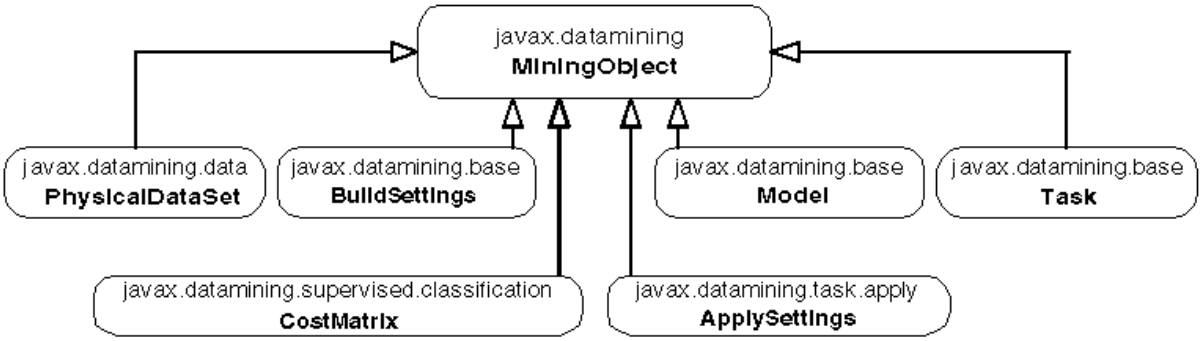
Systém poskytuje rozhraní pro definici vlastních funkcí, které mohou ohodnocovat nebo modifikovat vstupní data. Uživatel může vytvořit vlastní interní funkci nebo použít externí funkci jako zkompilovanou Java třídu.

4.1.1 Oracle Data Mining

Firma Oracle vyvinula pro svůj databázový server Oracle Database 10g Enterprise Edition nástroje a moduly pro podporu získávání znalostí z databází nazvané Oracle Data Mining (ODM). Koncepce systému pro dolování se od ostatních výrobců liší v tom, že funkcionalita je přenesena na databázový server. Tenký klient umožňuje zadávat dolovací úlohu, prezentovat výsledky apod. Tento přístup má mnoho výhod:

- Data zůstávají v databázi. Zpracování dat se provádí na straně databázového serveru včetně respektování přístupových práv. Data nejsou přenášena.
- Nízká zátěž přenosového pásma.
- Škálovatelnost a vhodnost pro aplikace náročné na výkon a bezpečnost.
- Řetězení výsledků dolování.
- Podpora pro více platforem.

Data Mining Server (DMS) je částí Oracle databáze, dolovacím strojem a úložištěm metadat dolování. Komunikace se serverem a přístup k dolovací funkcionalitě může být realizováno pomocí PL/SQL nebo pomocí Java API. Od verze 10.2 databázového serveru firma opustila proprietální řešení pro Java API a implementovala Java Data Mining standard. Oracle Data Mining Java API je implementací Java Data Mining (JDM) standardu ve verzi 1.0 podle JSR-73. Veškerá funkčnost DBMS_PREDICTIVE_ANALYTICS PL/SQL balíčku je přístupná z tohoto rozhraní pomocí tzv. *object factories*.



Obrázek 7 – Přehled JDM API

ODM podporuje všechny obecně používané dolovací úlohy, jako je detekce anomálií, měření významnosti atributů, asociační pravidla, shlukování, klasifikace, regrese, dolování v textu a BLAST (analýza DNA).

Firma Oracle nabízí velké množství nástrojů pro podporu získávání znalostí z databází, které pracují jako klient pro DMS. Tyto nástroje podporují vygenerování PL/SQL skriptů, které se dají použít v podnikových aplikacích. Tímto přístupem lze snadno zpřístupnit dolovací úlohy v tenkých

klientech a umožnit tak uživatelům jednoduše používat proces dolování bez znalosti celé problematiky.

5 Rich Client Platform

Vývoj desktopových aplikací na platformě Java je spojený s nutností programování ve Swing a AWT prostředí. Vytvoření robustní a rozsáhlé aplikace vyžaduje velmi dobrý návrh a disciplínu programování, protože jinak se další vývoj a udržování aplikace stává velice obtížným úkolem.

Swing je dobrá knihovna pro programování aplikací, ale jakmile se použije pro vytvoření komplexní grafické aplikace, je potřeba implementovat spoustu menších částí souvisejících s pohodlným ovládáním, na které je uživatel zvyklý z běžně používaných aplikací. Jedná se zejména o vytváření různých menu, kontextově citlivých akcí, dokovacího mechanizmu pro různá okna apod. V průběhu implementace často dochází k úzké vazbě mezi částmi systému a aplikace se tak stává nemodulární a těžce udržovatelná. Implementace těchto podpůrných částí je časově náročná a odvádí vývojáře od řešení aplikační logiky k implementaci generických GUI komponent.

V poslední době nabývají na významu nástroje pro vývoj desktopových aplikací postavených na tzv. Rich Client Palatform (RCP). Jedná se o využití existujících a vyspělých komponent frameworku místo toho, aby programátor musel implementovat prostředí od začátku. Vývoj na takové platformě přináší rychlejší proces vývoje, využití otestovaných komponent a nutí vývojáře dodržovat určité návrhové vzory. Většina aplikací má podobné požadavky, jako je menu, nástrojová lišta, správa dokumentů, uživatelská nastavení, systém nápovědy apod.

Typická RCP aplikace se skládá z následujících částí:

- Core – jádro frameworku.
- Workbanch – pohledy, editory, průvodci apod.
- Bundling framework – systém pro řízení vazeb mezi moduly
- Widget set – vizuální komponenty
- Update manager – správce online aktualizací
- Help – systém pro nápovědu

V současné době existují 3 RCP prostředí. Jsou to NetBeans Platform, Eclipse RCP a Spring RCP. Spring je velmi mladý framework (aktuálně ve verzi 0.2) a pro produkční nasazení tudíž nevhodný. NetBeans je platforma postavená na Swing, jejíž komponenty (widgets) jsou vykreslovány pomocí Java 2D. Eclipse oproti tomu je postaven na vlastním vykreslovacím systému SWT, který používá nativní vykreslování použitého prostředí. Právě kvůli použití nativního vykreslování, SWT implementace neexistuje pro platformy jako je IRIX, OpenVMS a některé UNIX systémy. NetBeans běží na jakékoliv platformě, pro kterou existuje implementace Javy. Obě platformy jsou si jinak ve funkčnosti a množství přínosných komponent velmi podobné. Rozhodnutí pro použití té či oné závisí spíše na tom, zda máme zkušenosti s vývojem ve Swing nebo SWT, anebo chceme využít již existující části napsané v jednom z těchto prostředí.

Platforma nabízí velmi kvalitní, robustní a otestovaný software, jehož využití je díky příznivým licencím zdarma (Eclipse Public License, Common Development and Distribution License). Velkou nevýhodou všech Rich Client Platform je absence kvalitní a ucelené dokumentace. To je zapříčiněno tím, že se jedná o poměrně mladou technologii. V době psaní této práce jsou pro NetBeans Platform dostupné pouze demonstrační příklady, API dokumentace a Wiki stránky s častými dotazy uživatelů. Díky otevřeným zdrojovým kódům, může být zdrojem informací rovněž zdrojový kód.

Jelikož pro řešení programové části tohoto projetu je použita platforma NetBeans, bude dále popsána a analyzována pouze tato platforma.

5.1 NetBeans Platform

NetBeans je open source Java platforma podporována firmou Sun. Původně bylo hlavním produktem NetBeans integrované vývojové prostředí pro vývoj Java aplikací. V současné době NetBeans platforma představuje komplexní balík produktů pro návrh a vývoj desktopových, serverových a mobilních aplikací. Jakékoliv části této platformy mohou být jednoduše integrovány ve vlastním produktu postaveném na této platformě. Výsledné řešení se tak z části skládá z velmi dobře otestovaných a robustních komponent, které jsou jednoduše rozšiřitelné.

Platforma je šířena pod CDDL (Common Development and Distribution License) licencí, která dovoluje využití jak v nekomerční, tak i v komerční sféře. Tato licence umožňuje vytvářet a distribuovat moduly a aplikace založené na NetBeans a používat součásti NetBeans ve vlastních aplikacích.

Důvodem pro výběr NetBeans RCP platformy je používaní Swing komponent v jádru pro dolování dat v prostředí Oracle, nad kterým je tato práce postavena. Eclipse sice dovoluje použít Swing komponenty, ale výsledný vzhled je nekonzistentní.

Aplikace postavené na NetBeans platformě dovolují dynamicky instalovat nové moduly bez nutnosti celou aplikaci znovu instalovat pro přidání nové funkčnosti. Vývoj aplikace se skládá z implementace NetBeans modulů, které mohou běžet v rámci NetBeans IDE nebo v rámci NetBeans Platform. Existuje velké množství modulů pro NetBeans, které mohou být využity ve vyvíjené aplikaci.

Vlastnosti NetBeans platformy:

- **Správa uživatelského rozhraní** – platforma nabízí okna, menu, nástrojové lišty a jiné komponenty pro prezentaci. Vývojář píše akce, které systém bude nabízet, a soustředí se tak na vývoj aplikační logiky.
- **Prezentace dat** – platforma nabízí bohatou škálu nástrojů pro zobrazení a editaci dat.
- **Správa nastavení** – ukládání a správa nastavení aplikace je bezpečná, jednoduchá a prováděna automaticky.
- **Grafická editace** – Drag and Drop editace dat apod.

- **Editor textu** – velmi silný editor textu vycházející z NetBeans IDE editoru, který může být upraven a rozšířen pro editaci různých typů souborů.
- **Framework pro průvodce** – nástroj pro vytváření rozšiřitelných průvodců, které vedou uživatele v průběhu definování komplexních úkolů.
- **Široký sortiment dalších komponent** – využití komponent pro verzování, specializované editory, vzdálený přístup ke zdrojům apod.
- **Systém aktualizací** – aktualizace aplikací jsou založeny na internetových technologiích a udržují systém aktuální.

Vývoj v prostředí NetBeans Platform se provádí pomocí průvodců v IDE a zjednodušuje tak psaní konfiguračních souborů.

Jak bylo již zmíněno výše, neexistuje v současné době dokumentace, která by obsáhla platformu jako celek. V následujících měsících by ale měla být publikována kniha *Rich Client Programming: Plugging into the NetBeans™ Platform*[1], která by podle prvních ohlasů měla být oficiální dokumentací k platformě.

5.1.1 Open API a jádro platformy

Samotná platforma se skládá z jádra a množiny rozhraní Open API, které se nacházejí v balíčku *org.netbeans.core.** resp. *org.openide.**. Jádro implementuje řadu hlavních rozhraní Open API potřebných pro samotný běh aplikace. Veškeré části výsledné aplikace se implementují do modulů. NetBeans modul je Java archív obsahující implementaci, která používá Open API, a *manifest* soubor, který obsahuje identifikaci, verzi implementace a závislosti na ostatních modulech.

Příklad *manifest* souboru generovaného vývojovým prostředím:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.0
Created-By: 1.6.0-b105 (Sun Microsystems Inc.)
OpenIDE-Module-Public-Packages: -
OpenIDE-Module-Module-Dependencies: cz.vutbr.fit.dataminer.api/1 > 1.0
 , cz.vutbr.fit.dataminer.core > 1.0, org.openide.dialogs > 7.3, org.o
 penide.util > 7.9.0.1
OpenIDE-Module-Java-Dependencies: Java > 1.5
OpenIDE-Module-Implementation-Version: 070513
OpenIDE-Module: cz.vutbr.fit.dataminer.module.testmodule
OpenIDE-Module-Layer: cz/vutbr/fit/dataminer/module/testmodule/layer.x
 ml
OpenIDE-Module-Localizing-Bundle: cz/vutbr/fit/dataminer/module/testmo
 dule/Bundle.properties
OpenIDE-Module-Specification-Version: 1.0
OpenIDE-Module-Requires: org.openide.modules.ModuleFormat1
```

[1] Boudreau, T., Tulach, J., Wielenga, G.: *Rich Client Programming: Plugging into the NetBeans™ Platform*. Prentice Hall PTR.

5.1.2 System FileSystem

Jedná se o virtuální systém souborů, který slouží jako úložiště konfiguračních údajů aplikace. Moduly využívají tento prostor pro definici vlastních přípojných bodů. Modul vytvoří svůj virtuální adresář a zdokumentuje, jaké virtuální soubory se mohou v tomto adresáři nacházet. Typicky se do těchto adresářů vkládají *instance* soubory. Ostatní moduly, aniž by cokoli věděly o implementaci svého okolí, mohou do těchto adresářů vkládat záznamy. Jedná se o volnou vazbu na základě implementace rozhraní definovaných ve veřejných API vrstvách.

Typickým příkladem je vytvoření akce a vložení do menu aplikace:

```
<folder name="Actions">
  <folder name="Build">
    <file name="cz-vutbr-fit-dataminer-actions-StartMiningAction.instance"/>
  </folder>
</folder>
<folder name="Menu">
  <folder name="RunProject">
    <file name="cz-vutbr-fit-dataminer-actions-StartMiningAction.shadow">
      <attr name="originalFile" stringvalue="Actions/Build/cz-vutbr-fit-
            dataminer-actions-StartMiningAction.instance"/>
    </file>
  </folder>
</folder>
```

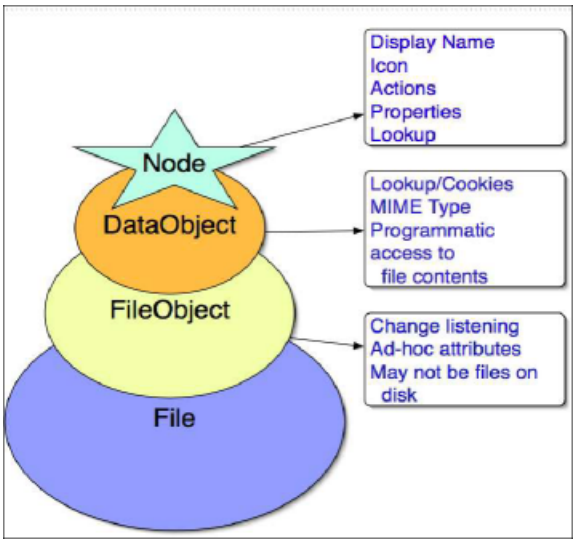
Od tohoto okamžiku se modul pro správu akcí postará o vytvoření instance třídy *StartMiningAction*, a vytvoření příslušných tlačítek, položek menu, nástrojové lišty apod.

Tento prostor můžou moduly modifikovat prostřednictvím *layer.xml* souborů. Záznamy se spojují do jedné konfigurace přístupné všem modulům. Tento přístup úzce souvisí s tím, že moduly mají vlastní zavaděč tříd (class loader), a nemají přístup k třídám (class path) ostatních modulů, i přesto, že běží v rámci jednoho virtuálního stroje Javy. Instance tříd vytváří *System FileSystem*. Tím je zajištěno volné vázání (tzv. loose coupling) a programování do rozhraní místo do konkrétních tříd.

System FileSystem je základním kamenem při implementaci modulů.

5.1.3 FileObject, DataObject a Node

Platforma dokáže rozpoznávat různé druhy souborů, které mají v aplikaci vlastní ikony, položky v menu a chování. Soubory jsou reprezentovány jako *FileObjects*, což je obalová třída kolem *java.io.File* třídy. To, co je v aplikaci zobrazeno, jsou instance třídy *Node*, které poskytují operace a prezentaci různých objektů, mezi které se řadí i soubory. Mezi *Nodes* a *FileObjects* se zařazují *DataObjects*.

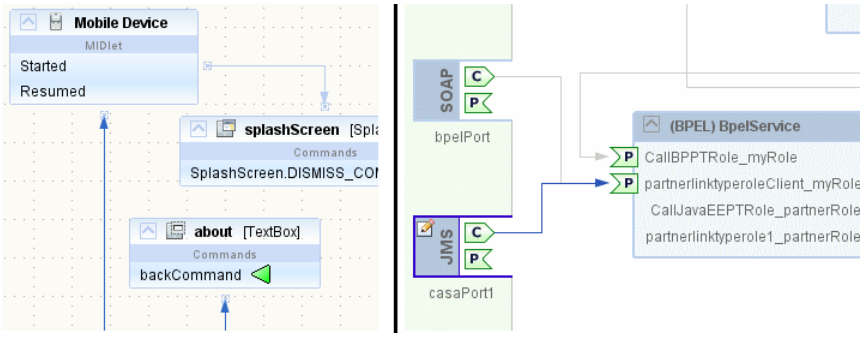


Obrázek 8 - Způsob reprezentace souborů (převzato z [10])

DataObject slouží jako vyšší abstrakce nad souborem. Rozumí obsahu daného druhu souboru. To znamená, že pro každý druh souboru existuje příslušný DataObject.

5.1.4 NetBeans Visual Library

Visual Library je knihovnou NetBeans, která je použitelná pro obecnou vizualizaci včetně podpory zobrazení grafů. Je základem pro veškeré grafické editory NetBeans, jako je vizuální editace konfiguračních souborů (Struts, JSF apod.) nebo definování procesů (BPEL).



Obrázek 9 - Příklad editoru aplikací postavených na Visual Library

Styl programování je podobný programování Java Swing aplikací. Vizuální komponenty se jmenují *Widgets* a vkládají se do stromu reprezentovaného *Scene* třídou. Knihovna rovněž poskytuje náhled aktuální scény, který se využívá v komponentě pro navigaci v řešené aplikaci.

Vlastnosti knihovny:

- Podpora pro grafy
- Vrstvy zobrazení
- Funkce lupy (Zoom)
- Podpora pro dynamickou změnu detailů (Level of Detail)

- Animace
- Editace v místě poklepání
- Různé módy zobrazení prvků
- Export scény do obrázků, PDF souborů
- Podpora tisku

6 Analýza a návrh řešení

Původní aplikace, kterou toto řešení rozšiřuje o grafickou nadstavbu a uživatelské rozhraní, je navrhnuta tak, že odděluje grafické rozhraní od jádra systému. To dovoluje nahrazení původního grafického rozhraní za nové bez nutnosti zásahu do jádra. Přestože jedním z požadavků na výsledné řešení bylo využití knihovny bez zásahu do jádra a programového rozhraní, bylo nutné projekt migrovat na současnou verzi Java, na které běží NetBeans platforma.

6.1 Migrace a refaktoring jádra

Systém bude migrován pro Java verzi 6, což znamená odstranění nekompatibilit a nové sestavení programu. Java sice zachovává v každém svém vydání zpětnou kompatibilitu, ale při pokusech o spuštění knihovny ve verzi 5 a výše docházelo k pádu aplikace. Knihovna rovněž využívá pro kompilaci DMSL funkcí Java kompilátor, který není součástí Java Runtime Environment. Při komunikaci mezi rozhraním kompilátoru (Java Runtime Environment) a implementací (knihovna *tools.jar* z Java Development Kit) vzniká nekompatibilita, která se touto migrací vyřeší.

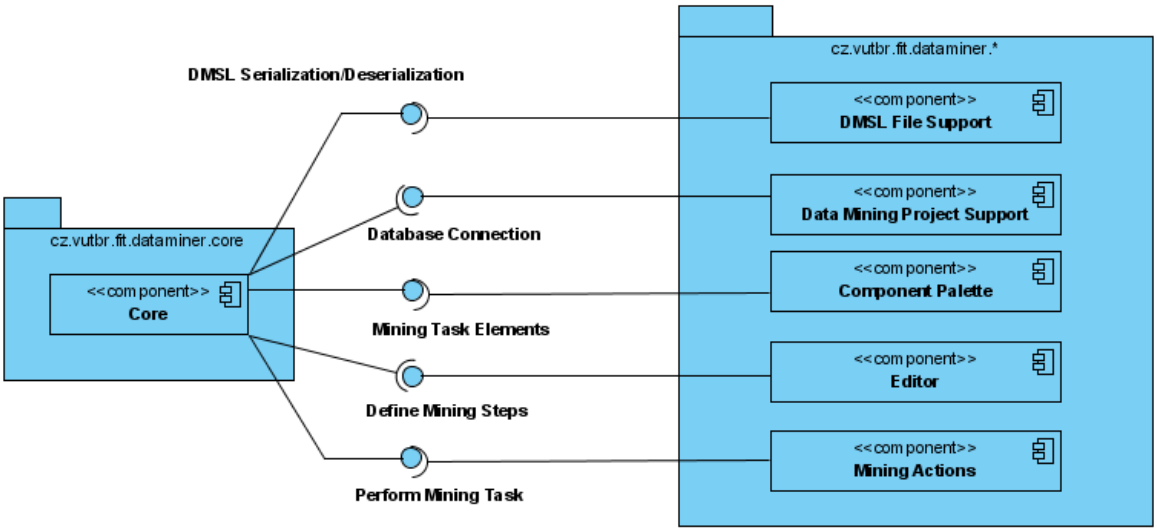
Problémem původního řešení je rovněž absence použití balíčků tříd na hlavní úrovni (tzv. package). Konvence programování v Javě doporučují pojmenování a strukturování tříd do balíčků od nejvyšší úrovně hierarchie. Bude proto zaveden systém balíčků s kořenem *cz.vutbr.fit.dataminer*. Pro existující jádro bude použit balíček *cz.vutbr.fit.dataminer.core*. Pro vyvíjené grafické uživatelské rozhraní budou sloužit sourozenecké balíčky umístěné v *cz.vutbr.fit.dataminer.**.

Jádro systému bude zkompilováno a zabaleno do Java archívu (JAR). Knihovny, na kterých je jádro závislé, budou taktéž distribuovány jako JAR archívy.

6.2 Koncepce systému

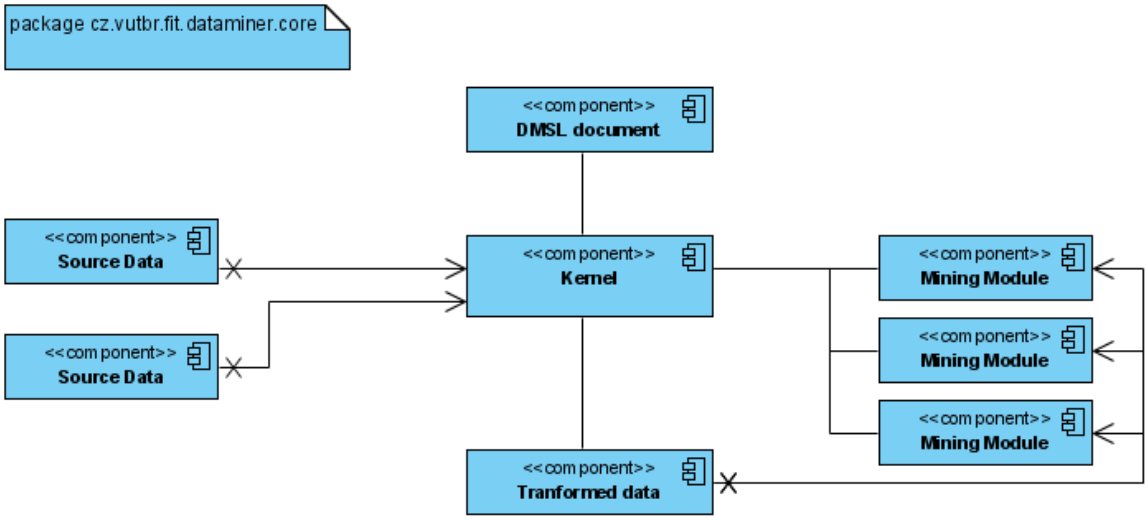
Na nejvyšší úrovni můžeme systém rozdělit na jádro a grafické uživatelské rozhraní. Grafické uživatelské rozhraní by mělo poskytovat podporu pro práci s DMSL soubory, tyto soubory by měly být sdružovány do projektů. V průběhu editace by měla být zobrazena paleta s elementy, které lze umístit do editoru grafu. Editor by měl modifikovat části DMSL dokumentu a určitým způsobem komunikovat s jádrem.

V neposlední řadě by měla aplikace podporovat možnost lokalizace do různých jazyků a kontextově závislý systém nápovědy.



Obrázek 10 - Koncepce systému

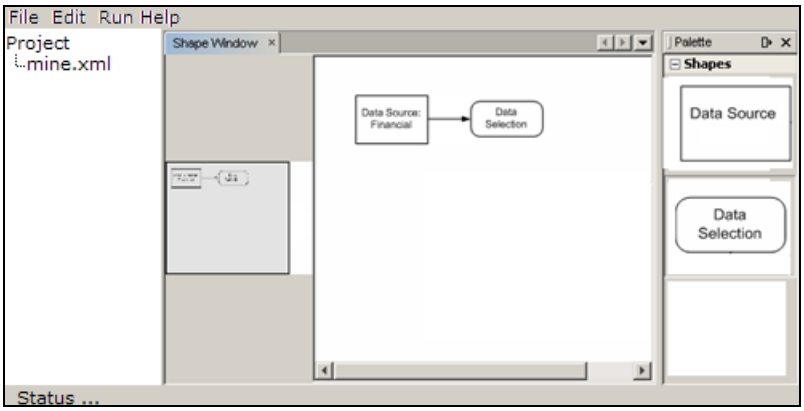
Samotné jádro se pak skládá z jádra pro dolování, dolovacích modulů a tříd pro správu DMSL dokumentů. Jádro pro dolování vlastní DMSL dokument, který obsahuje definici celého procesu dolování – definuje vstupní data, data k dolování, závislosti mezi daty, dolovací úlohu a vlastní funkce pro úpravu dat.



Obrázek 11- Diagram komponent jádra (převzato z [3])

6.2.1 Návrh uživatelského rozhraní

Grafické uživatelské prostředí se bude skládat z části pro zobrazení otevřených projektů a jejich obsahu. Po poklepání na příslušný DMSL soubor se otevře editor s paletou elementů, které lze vkládat do editoru a modifikovat tak samotný dokument. Editor by měl být schopný vytvářet vazby mezi elementy grafu a nabídnout akce pro otevření dialogu, ve kterém uživatel může specifikovat detaily kroku v procesu získávání znalostí.



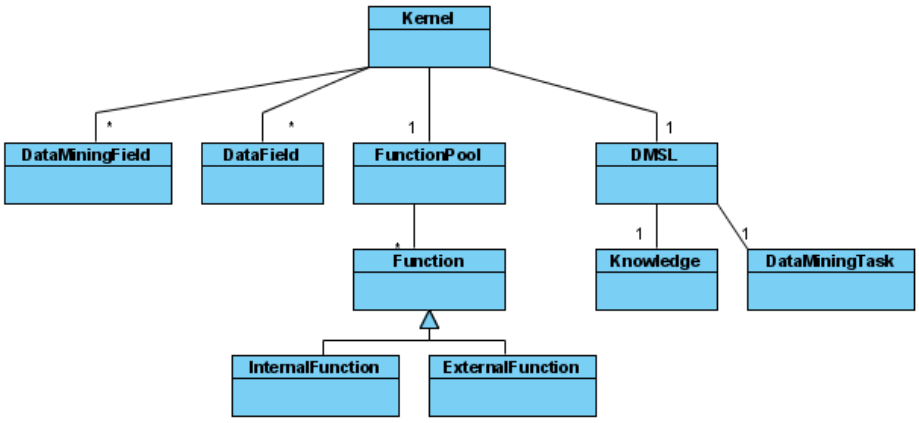
Obrázek 12 - Návrh GUI

Aplikace by měla zobrazit navigační okno pro rychlou navigaci v právě editovaném schématu. Veškeré dostupné akce by měly být přístupné z menu. Často používané akce by měly být přístupné z nástrojové lišty. Akce, jako je například spuštění dolovací úlohy, by měly být povoleny pouze tehdy, jakmile má provedení akce smysl (kontextově citlivá akce).

Veškeré dlouho trvající akce by měly být prováděny v samostatném vlákně a jejich průběh by měl být zobrazen pomocí komponenty zobrazení průběhu (progress bar) nebo zprávami ve stavovém řádku.

6.2.2 Funkčnost jádra

Jádro zajišťuje veškerou režii spojenou s načtením a editací DMSL dokumentu. Umožňuje definovat celý proces dolování právě pomocí DMSL dokumentu, z čehož vyplývá podobnost koncepce jádra se strukturou dokumentu. Jádro má 5 základních komponent vycházejících z elementů DMSL dokumentu. Jádro je vlastníkem dokumentu a uchovává jej v podobě DOM objektu XML. Samotné jádro pracuje pouze se třemi objekty, které obalují DMSL dokument. XML DOM dokument je spravován třídou DMSL.



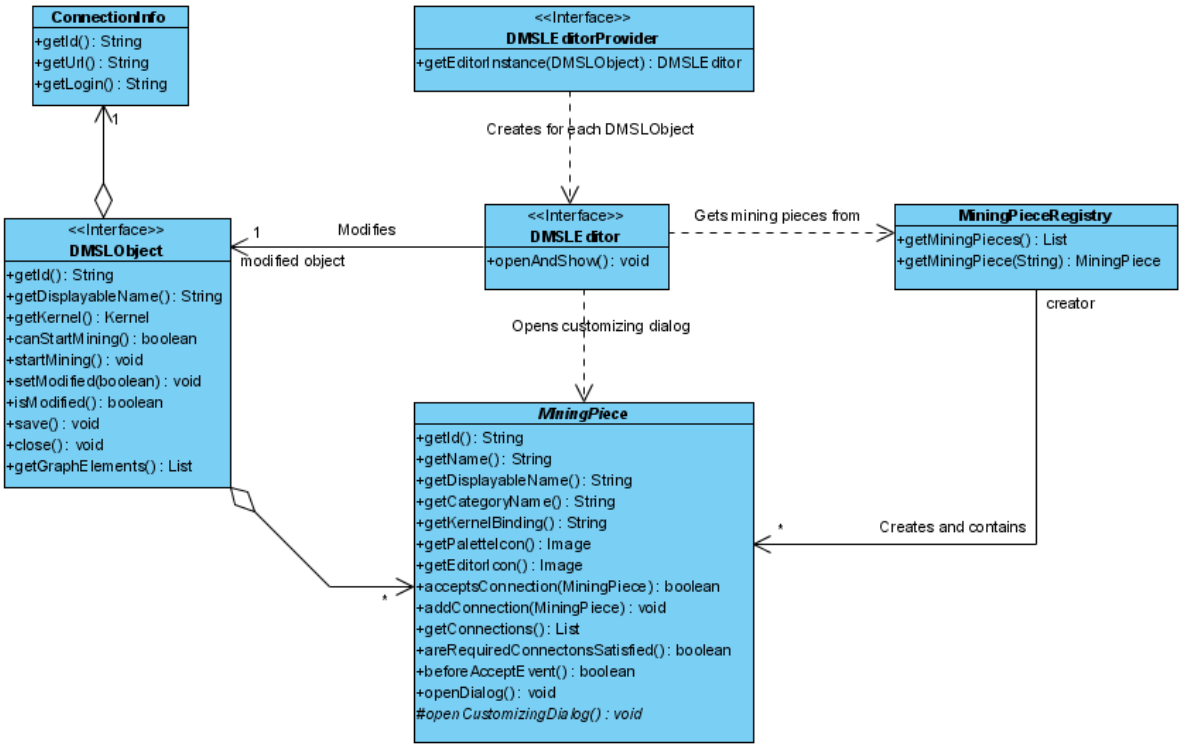
Obrázek 13 - Diagram tříd jádra systému

6.2.3 Komponenty grafického uživatelského rozhraní

Veškeré komponenty by měly být maximálně modulární a závislé pouze na společném API. To znamená, že například podpora pro vytváření projektů by neměla být závislá na editoru a opačně. Takže vazby mezi moduly by měly být definovány deklarativně (*layer.xml* v prostředí NetBeans) anebo přes rozhraní v API (Lookup system). Tímto lze implementaci částí systému nahradit nebo doplnit jinou implementací. Výhody takovéhoto řešení se projeví už v prvních krocích vývoje, při opravách chyb (změny v jednom modulu neovlivní ostatní moduly) a zejména v dalších iteracích vývoje, kdy se rozšiřuje funkčnost.

6.2.3.1 GUI API vrstva

Pro zachování modularity systému je nutné vytvořit společnou API vrstvu grafického uživatelského rozhraní.



Obrázek 14 - Diagram tříd API vrstvy

Hlavní třídou je implementace *DMSLObject* rozhraní, která zastřešuje služby a atributy, které poskytuje DMSL dokument a jádro. Jedna instance *DMSLEditor* třídy může modifikovat jeden *DMSLObject* objekt. Editor vkládá do DMSL objektu instance abstraktní třídy *MiningPiece* (ve skutečnosti se jedná o instance tříd dědících od této abstraktní třídy), které představují elementy blokového schématu (resp. dolovací úlohy). Editor rovněž implementuje metody a akce pro vytváření vazeb mezi instancemi *MiningPiece*, a také poskytuje akce pro otevření dialogu pro editaci vybraného elementu.

6.2.3.2 Podpora pro rozpoznání a práci s DMSL soubory

Pro správnou funkci podpory projektů v prostředí NetBeans je potřeba, aby systém dokázal rozpoznat DMSL soubor mezi ostatními soubory na disku. To lze vyřešit pomocí speciální přípony souboru (např. dokument.dmsl) anebo pomocí tzv. MIME Lookup. V případě MIME Lookup systém analyzuje obsah souboru a podle zaregistrovaných MIME rozpozná, o jaký typ souboru se jedná.

Serializace/Deserializace DMSL dokumentů

Čtení DMSL souboru a jeho ukládání provádí jádro systému. Jelikož editor je grafický, je rovněž nutné ukládat informace o poloze a vazbách mezi *MiningPiece* objekty do perzistentní paměti. To lze implementovat dvěma způsoby:

1. Ukládat tyto meta informace do samostatného souboru, který se načte/uloží při otevření/uložení DMSL souboru. Tento soubor by byl v projektovém zobrazení skrytý před uživatelem. Nevýhodou tohoto řešení je větší režie spojená se správou druhého souboru a špatná přenositelnost DMSL souborů.
2. Druhým, vhodnějším způsobem je uložení meta informací do téhož XML souboru. Problémem je zde definice struktury DMSL dokumentu, která nedovoluje vložení takovýchto informací. DMSL dokument sice v určitých elementech dovoluje vložit jakýkoliv obsah, ale z hlediska sémantiky je toto umístění nevhodné pro tyto informace. Nabízejí se tedy další dvě varianty řešení:
    a. Díky použití jazyka XML, lze DMSL dokument vložit dovnitř rodičovského elementu, který bude obsahovat jak podstrom pro samotný DMSL dokument, tak sourozenecký podstrom reprezentující meta informace. V tomto případě bychom přišli o přenositelnost těchto souborů mezi různými systémy, které pracují s DMSL soubory. Přenositelnost byla jedním z hlavních kritérií pro vytvoření DMSL jazyka a tudíž tato varianta se jeví jako nevhodná.
    b. Posledním řešení je uložení meta informací dovnitř DMSL dokumentu ve formě XML komentáře. Uvnitř XML komentáře může být jakákoliv textová informace, která neobsahuje uzavírací značku XML komentáře. Je tedy možné do tohoto komentáře vložit například další XML dokument a tento zpracovávat jako samostatnou informaci.

V budoucnu by bylo vhodné zvážit rozšíření definice DMSL dokumentu o možnost uchovávat meta informace související s vizualizací nebo prezentací struktury dokumentu.

6.2.3.3 Podpora pro práci v projektech

Pokud vyjdeme z předpokladu, že jeden DMSL dokument odpovídá jedné dolovací úloze s vlastním připojením do databáze, bylo by vhodné DMSL dokumenty zařazovat do projektů. Projekt v této roli slouží jako logická vazba mezi DMSL dokumenty, které mají podobnou dolovací úlohu, sdílejí stejné

připojení do databáze nebo spolu jinak souvisejí. Můžeme pak definovat připojení do databáze se zdrojovými daty pouze jednou pro daný projekt.

Projekt dále může ukládat do perzistentní paměti různá nastavení pro daný projekt. Projektové logické schéma pak zobrazí pouze relevantní soubory a prvky, které by uživatel měl vidět. Ostatní, pomocné soubory jsou filtrovány.

6.2.3.4 Editor, paleta komponent a navigace

Editor zobrazuje DMSL dokument jako orientovaný graf, jehož uzly představují jednotlivé kroky v procesu dolování dat. Editace dokumentu by měla být přístupná pomocí *Open* akce, při výběru DMSL souboru v projektovém prohlížeči. Každý element (uzel) grafu představuje jednu instanci *MiningPiece* implementace.

V průběhu editace je k dispozici paleta komponent, ze které uživatel může přesouvat vizuální elementy do grafu. Jedná se o seznam dostupných implementací abstraktní třídy *MiningPiece*, která je součástí API. Tento seznam se vytváří deklarativně za pomocí *layer.xml* souboru, ve kterém se zaregistruje každá komponenta dolovacího procesu včetně dolovacích modulů, které mají být přístupné uživateli. Přístup k implementacím je prováděn pomocí System File System. Tímto se vyřeší problém závislostí modulů, a jakýkoliv nový modul může být jednoduše nainstalován a veškeré jeho *MiningPiece* implementace budou automaticky přístupné přes paletu a zobrazitelné v editoru. Tímto přístupem se zajistí modularita systému.

Je potřeba zajistit, aby přesunutí elementu z palety a vložení do grafu bylo povoleno pouze tehdy, jakmile tato operace má smysl (např. graf nemůže obsahovat dva elementy *DataFields*).

Editor by měl umožňovat volné přesouvání uzlů grafu a vytváření hran (vazeb) mezi relevantními uzly (elementy). Přípustné a povinné vazby by rovněž měly být definovány deklarativně v *layer.xml* souboru. Po poklepání na příslušný uzel grafu nebo pomocí kontextového menu by se mělo zobrazit modální dialogové okno příslušného elementu. V tomto okně lze pak specifikovat detaily (např. *DataFields* element nabídne výběr různých sloupců tabulek, které se použijí jako zdrojová data).

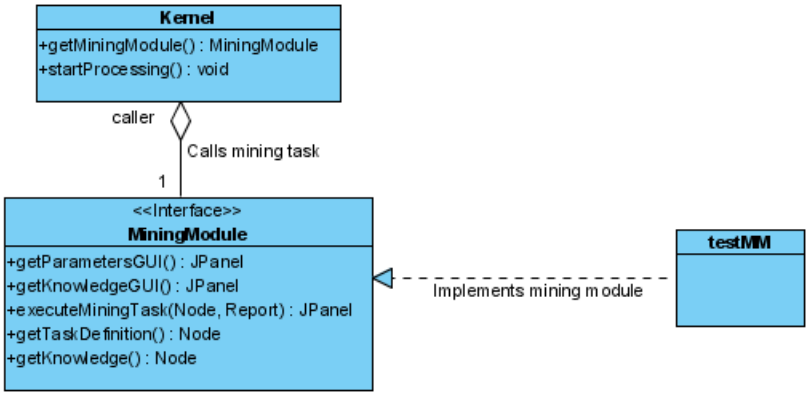
V průběhu editace je vhodné, aby v samostatném okně byl zobrazen náhled grafu ve zmenšeném měřítku. Tento náhled by měl interaktivně zobrazovat aktuální pozici editoru vzhledem k plátnu, na kterém se kreslí graf. V první fázi této práce není k dispozici mnoho elementů, které lze do editoru vkládat, ale s rostoucí komplexností a přidáváním dalších vlastností tato funkčnost přijde jako velmi praktická a uživatelsky příjemná. Pokud bychom se například rozhodli, že se v grafu budou zobrazovat všechny položky obsažené v *DataFields* elementu, může pak graf obsahovat velké množství uzlů.

Implementace vykreslovacího mechanizmu

Pro vykreslování grafu do okna editoru by bylo vhodné použít už existující a robustní řešení místo implementace vlastního mechanismu. Vlastní implementace by byla časově náročná a není předmětem této práce. V prostředí NetBeans se nabízí využití Visual Library knihovny, která obsahuje podporu pro kreslení různých typů schémat včetně grafů s uzly a hranami.

6.2.3.5 Dolovací moduly

Dolovací moduly implementují rozhraní jádra a provádějí určitý typ dolovací úlohy. Jedná se o implementaci *MiningModule* rozhraní.



Obrázek 15 - Rozhraní dolovacího modulu

Vyvstává otázka, jak se tyto části budou integrovat do výsledného řešení. Je potřeba, aby každou novou implementaci dolovacího modulu bylo možno snadno instalovat do aplikace. To znamená, že žádná část systému nemůže být závislá na jakékoliv implementaci dolovacího modulu.

Tento problém řeší v předchozí kapitole zmíněná registrace *MiningPiece* implementace v *layer.xml*. Takže implementace dolovacího modulu se bude prezentovat jako další prvek, který lze umístit do grafu. Samozřejmě je potřeba, aby tyto moduly byly vizuálně odlišitelné od ostatních elementů a byly rovněž umístěny do samostatné kategorie v paletě. To, jakým způsobem se modul prezentuje v grafu a paletě, je na implementaci *MiningPiece*, která obaluje dolovací modul.

Tímto způsobem lze postupně implementovat různé dolovací moduly a jednoduše je instalovat do stávající aplikace bez nutnosti kompilace jakékoliv další části systému a nové instalace celé aplikace. Jedná se o instalaci zásuvných modulů (pluginů) tak, jak to známe z jiných aplikací. Rovněž aktualizace jednotlivých částí je tímto bezproblémová.

6.2.3.6 Aktualizace a instalace zásuvných modulů (pluginů)

NetBeans platforma obsahuje modul Plugin Manager, pomocí něhož lze za běhu velmi jednoduše instalovat nové moduly. Je možné instalovat jak stažené *nbm* soubory, tak moduly vystavené na internetu. NetBeans Module (soubor s příponou *nbm*) je archív určený pro distribuci modulu.

Výsledná aplikace bude podporovat online možnost aktualizace novější verzí z internetu. Rovněž nové dolovací moduly budou přístupné k instalaci online.

6.3 Nástroje pro vývoj

Jelikož v současné době je k dispozici Java ve stabilní verzi 6, bude pro vývoj použit *Java Development Kit 1.6.0*[2].

NetBeans IDE od verze 5.5 pro samotný běh vyžaduje Javu v minimální verzi 5. *NetBeans IDE*[3] v základní instalaci obsahuje nástroje pro automatizovaný překlad (*Ant*), nástroje pro automatické testování (*JUnit*) a podporu pro verzovací nástroje (*Subversion*). Jedná se o nástroje používané v rámci extrémního programování, které jsou užitečné i při vývoji aplikace v malém týmu. Není tedy potřeba instalace dalších součástí potřebných pro překlad ze zdrojových kódů a další vývoj této aplikace.

Projekt se překládá za pomocí nástroje Ant z prostředí IDE. Samozřejmě by bylo možné vytvořit distribuční balík zdrojových kódů tak, aby šlo projekt vyvíjet i z jiných IDE, ale práce by byla velmi neefektivní. NetBeans IDE obsahuje rozsáhlou podporu pro vývoj aplikací pro NetBeans Platform.

Jeden z problému, který bylo potřeba vyřešit, je volba verze platformy NetBeans. V době implementace tohoto projektu byly k dispozici dvě verze, které přicházely v úvahu:

1. NetBeans 5.5 – jedná se o stabilní verzi, která ale v sobě neobsahuje Visual Library pro kreslení grafů. Pro tuto verzi je možné knihovnu zkompilovat, ale při praktickém použití bylo chování této knihovny nestabilní.
2. NetBeans 6 – jedná se o nestabilní vývojovou verzi, která je vydávána v tzv. Milestone verzích. Jedná se o verze, které přinášejí novinky do platformy. Oproti denním sestavením jsou Milestone verze poměrně dobře otestovány. Visual Library je standardní součástí této verze.

Po zhodnocení všech pro a proti byla zvolena vývojová verze NetBeans 6 Milestone 9. Pro budoucí vývoj je lepší zvolit novější verzi, která přináší řadu vylepšení a optimalizací, než trvání na současné stabilní verzi, která za rok bude zastaralá.

Volba verze NetBeans není ve skutečnosti tak dramatická, jako by to bylo v případě Eclipse Rich Client Platform. NetBeans má velmi stabilní vývojový cyklus a změny v API jsou prováděny výjimečně. NetBeans zachovává zpětnou kompatibilitu na úrovni API. To v případě Eclipse neplatí.

---

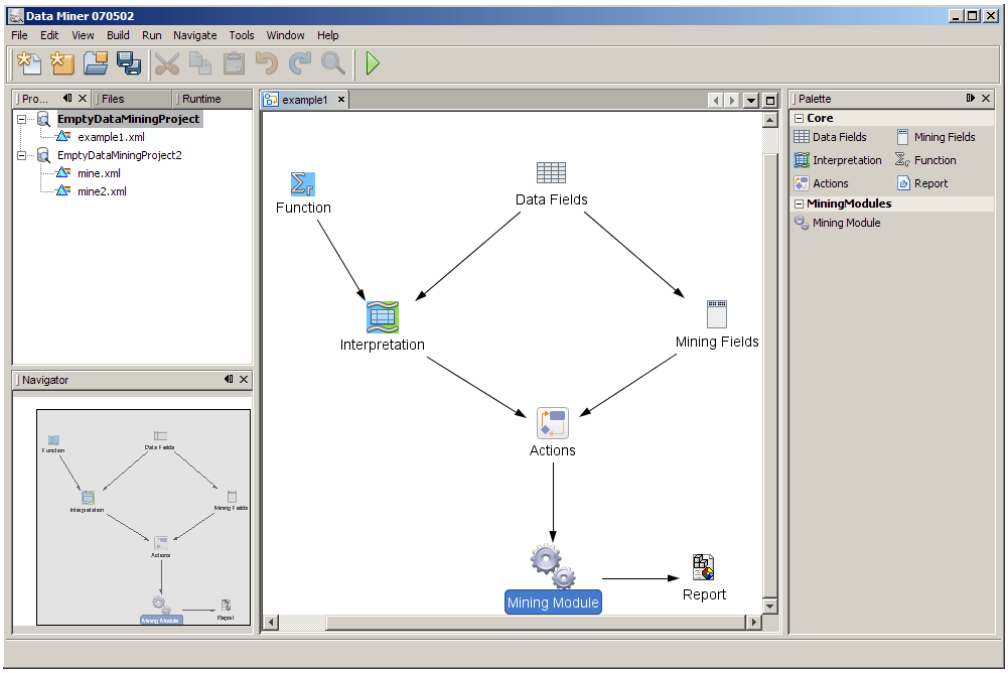
[2] Java Development Kit 1.6.0 je k dispozici ke stažení na stránce http://java.sun.com/javase/downloads/

[3] Netbeans IDE včetně veškeré dokumentace a různých pluginů je k dispozici na adrese http://www.netbeans.org

Na CD nosiči, který je přiložen k této práci, je uloženo celé vývojové prostředí včetně zdrojových souborů. Po spuštění je možné projekt dále vyvíjet ve stejném prostředí jako na začátku a je tak možné porovnat chování v případě migrace na novou verzi platformy.

7 Řešení

Tato kapitola popisuje implementační detaily programového řešení. Jsou zde uvedeny pouze důležité části týkající se architektury systému. Podrobnější informace o implementaci jsou obsaženy v API dokumentaci (vygenerovaný Javadoc) a v dostupné dokumentaci k NetBeans platformě.



Obrázek 16 - Vzhled aplikace s rozvržením komponent

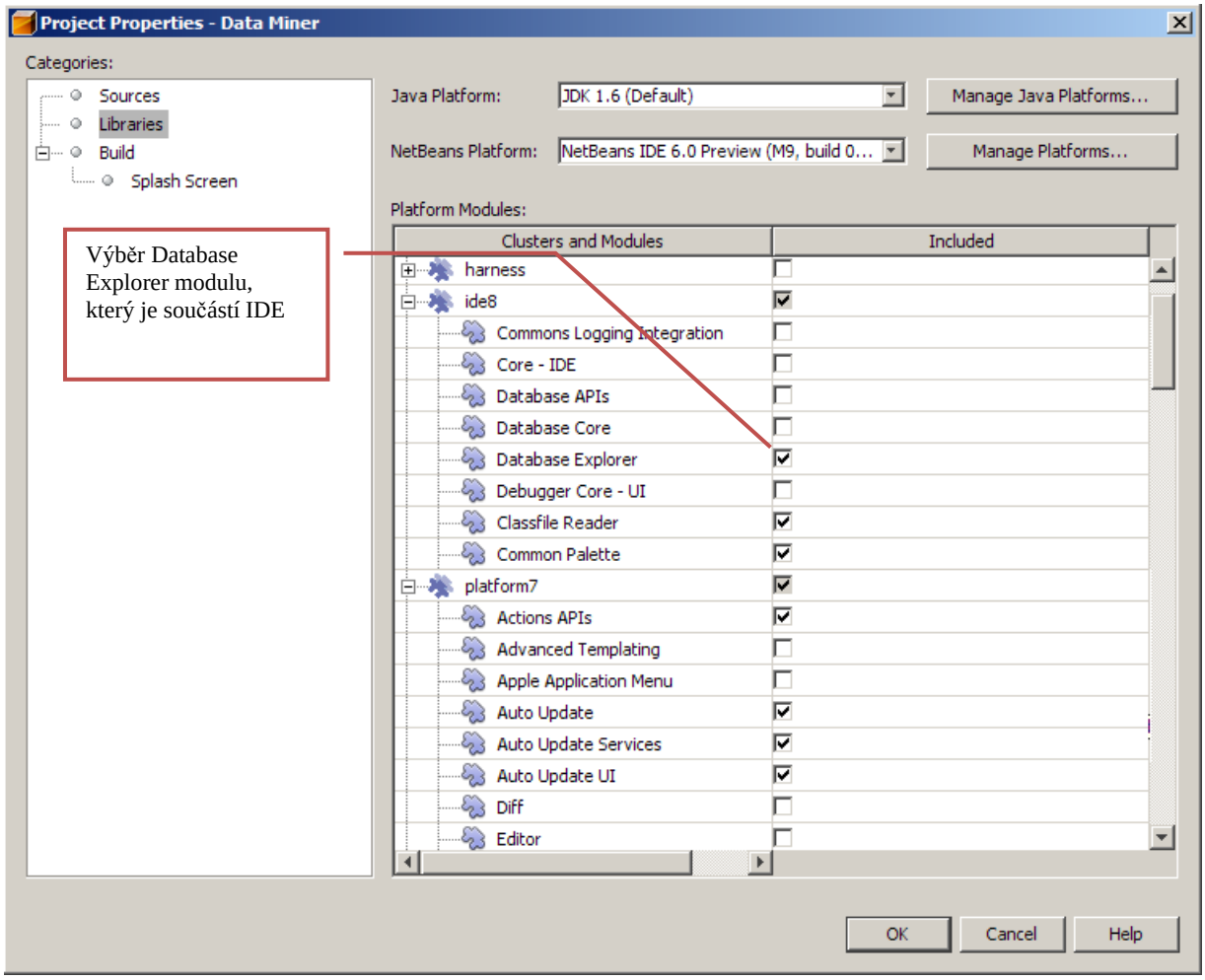
7.1 NetBeans Module Suite

Aplikace postavená na NetBeans platformě se skládá z NetBeans modulů. Tyto moduly musejí být nějak spojeny ve funkční celek. Tuto integraci provádí module suite. Modul tedy může být integrován do samostatného module suite, anebo dokonce může být instalován do samotného vývojového prostředí. Module suite je vlastně integrované vývojové prostředí bez nepotřebných modulů.

Prvním krokem je vytvoření module suite a následně výběr modulů potřebných pro základní běh platformy. Výběr je velmi jednoduchý a skládá se ze zaškrtnutí políčka pro daný modul, který chceme, aby byl zahrnut v naší aplikaci. Vývojář většinou zahrnuje moduly z Open API, které se nacházejí v kategorii *platform7*. V případě, že je potřeba zahrnout do výsledného produktu nějakou konkrétní část IDE, je zde možnost vybrat modul z kategorie *ide8*.

Dalšími parametry, které lze specifikovat, je tzv. splash screen a název aplikace. Přes kontextové menu v projektovém zobrazení lze sestavit celou aplikaci, spustit ji v normálním nebo

ladícím režimu (debug mode), vytvořit ZIP distribuční balík se spustitelnou aplikací (jak v prostředí Windows, tak v systémech Unixového typu) anebo vytvořit JNLP[4] aplikaci.



Obrázek 17 - Dialog Module Suite pro výběr standardních modulů

Na obrázku je zobrazen případ, kdy chceme do řešení zahrnout Database Explorer modul, který v prostředí NetBeans IDE spravuje a zobrazuje databázové ovladače a veškerá připojení do databáze. Takto lze jednoduše integrovat již existující moduly, které jsou součástí IDE, do vlastního řešení.

7.1.1 Podpora pro online aktualizace software

Začlenění podpory pro aktualizaci softwaru a instalaci nových modulů je velmi jednoduché. Stačí v module suite vybrat příslušné moduly. V menu záložce *Tools* se pak objeví položka *Plugins*, pomocí které lze realizovat tyto operace. Veškerá funkčnost spojená s aktualizací dostupná v IDE je od tohoto okamžiku dostupná ve vyvíjené aplikaci.

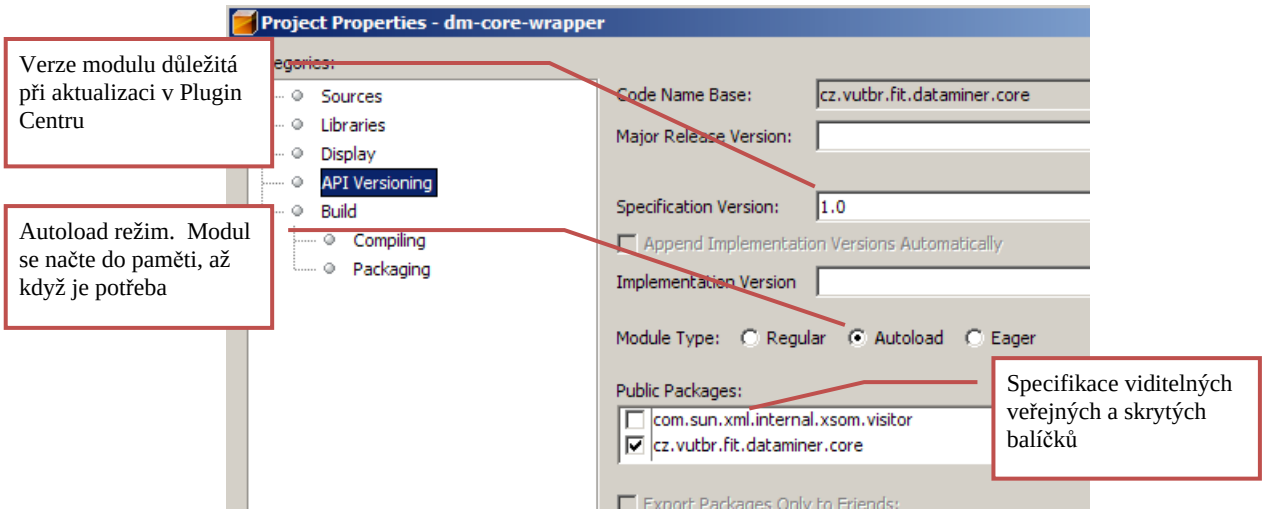
[4] JNLP je protokol určený pro Java Web Start. Takovou aplikaci je možné spustit z prostředí internetového prohlížeče.

7.2 Library Wrapper pro dolovací jádro

Pro to, aby mohly ostatní moduly používat dolovací jádro, je potřeba vytvořit tzv. Library Wrapper Module. Každý modul NetBeans má vlastní zavaděč tříd (classloader), který má k dispozici pouze vlastní třídy (classpath) umístěné v tomto modulu a třídy umístěné v modulech, na kterých je závislý. Jelikož řada modulů výsledného řešení je závislá na třídách v knihovně jádra, je potřeba vyřešit problém závislostí. K tomuto účelu slouží Library Wrapper, který se tváří jako samostatný modul, ale ve skutečnosti obsahuje pouze potřebné knihovny ve formě JAR souborů.

V konfiguraci Library Wrapper modulu je možné specifikovat, které balíčky modul zpřístupní ostatním modulům. V případě dolovacího jádra je vhodné, aby modul zpřístupnil pouze interface dolovací knihovny a ostatní balíčky potlačil.

Jelikož ostatní části aplikace nepotřebují přístup ke knihovnám třetích stran použitým v dolovacím jádře, jsou tyto knihovny skryty a vloženy dovnitř modulu jádra. Tímto je jádro distribuováno jako celek, nemá žádné závislosti na ostatních modulech a může být bez problémů integrováno do jiných řešení.



Obrázek 18 - Dialog pro nastavení vlastností modulu

Veškeré knihovny a API vrstvy je vhodné označit jako *autoload* modul. Tím je zajištěno, že modul se načte do paměti až při prvním použití. Start aplikace je pak rychlejší, protože se nahrávají pouze moduly označené jako *regular*.

7.3 API vrstva

API vrstva aplikace je implementována v samostatném *autoload* modulu. Implementuje třídy podle popisu v kapitole návrhu. Zajímavým prvkem je zde implementace třídy *MiningPieceRegistry*. Ta

prochází *System FileSystem* a hledá instance *MiningPiece*. Tyto instance se registrují v *layer.xml*, což je virtuální systém souborů. Instance jsou hledány v těchto složkách:

```
<!-- All mining pieces should be registered in appropriate category -->
<folder name="MiningPieceRegistry">
  <folder name="Core">
  </folder>
  <attr name="Core/MiningModules" boolvalue="true"/>
  <folder name="MiningModules">
  </folder>
</folder>
```

Přesněji řečeno, instance se vkládají do příslušné kategorie (*Core*, *MiningModules* apod.). Kategorie přispívají k přehlednosti a využívají se při zobrazení v paletě.

Dalším elementem v *layer.xml* je *MiningPieceConfig*, ve kterém se definují možnosti vytváření vazeb mezi uzly v grafu (uzel odpovídá instanci *MiningPiece*). Pokud tyto vlastnosti nejsou definovány, nelze vytvářet příslušné vazby mezi elementy v grafu.

Názorný příklad konfigurace je popsán v kapitole 8, která popisuje příklad integrace dolovacího modulu.

7.4 Podpora DMSL souborů

Dalším krokem je vytvoření podpory pro DMSL soubory. Platforma bude schopna tyto soubory rozpoznávat, přiřadit jim prezentaci a operace. Je to důležité pro zobrazení souborů v projektech. Vytvoření této podpory je poměrně jednoduché a provádí se pomocí průvodce v IDE. Důležitou částí je registrace v layer.xml souboru. Od tohoto okamžiku všechny části platformy jsou schopny pracovat s DMSL soubory standardním způsobem.

Ukázka konfigurace:

```
<filesystem>
  <folder name="Loaders">
    <folder name="application">
      <folder name="dmsl+xml">
        <folder name="Actions">
          <file name="org-openide-actions-OpenAction.instance"/>
          . . .
        </folder>
      </folder>
    </folder>
  </folder>
  <folder name="Services">
    <folder name="MIMEResolver">
      <file name="DMSLResolver.xml" url="DMSLResolver.xml">
        <attr name="SystemFileSystem.localizingBundle"
stringvalue="cz.vutbr.fit.dataminer.filesupport.Bundle"/>
      </file>
    </folder>
  </folder>
  <folder name="Templates">
    <folder name="Other">
      <file name="DMSLTemplate.xml" url="DMSLTemplate.xml">
        <attr name="SystemFileSystem.localizingBundle"
stringvalue="cz.vutbr.fit.dataminer.filesupport.Bundle"/>
```

```
        <attr name="template" boolvalue="true"/>
      </file>
    </folder>
  </folder>
</filesystem>
```

Z ukázky je patrné, že se v sekci *Loaders* pod novým MIME typem *dmsl+xml* zaregistrují akce, které lze s tímto souborem provádět. Následně se přidá nový *MIME resolver*, který dokáže poznat, zda se jedná o DMSL soubor. Zároveň se vytvoří i šablona DMSL souboru, která se použije při vytvoření nového prázdného DMSL souboru.

Node pro DMSL soubor má ve svém *Lookup* zaregistrovanou implementaci *OpenCookie* rozhraní, která provádí otevření editoru pro daný *DataObject*. Standardní akce pro otevření souboru prohledává lookup vybraného *Node* a pokud v sobě obsahuje implementaci *OpenCookie*, použije tuto implementaci pro otevření souboru.

7.4.1 Serializace/Deserializace DMSL souborů

Pro serializaci a deserializaci DMSL dokumentů se používá jádro pro dolování. Knihovna XStream, která dokáže POJO (Plain Old Java Object) třídy převádět do XML perzistentní formy, se používá k ukládání pozice a vazeb mezi instancemi *MiningPiece*. XML dokument s meta informacemi, který tato knihovna vytvoří, se vloží do výsledného souboru ve formě XML komentáře. Tímto přístupem je výsledný soubor validní a přenositelný, a zároveň obsahuje meta-informace pro prezentaci.

7.5 Organizace práce do projektů

V případě, že aplikace má podporovat práci v projektech, stačí v module suite zaškrtnou políčka pro podporu a vizualizaci projektů. Tím se do aplikace dostane Project API a vizuální komponenty pro zobrazení projektů v *explorer* okně. V aplikaci se zpřístupní standardní akce známé z IDE pro otevření projektu, zavření, vytvoření nového projektu apod.

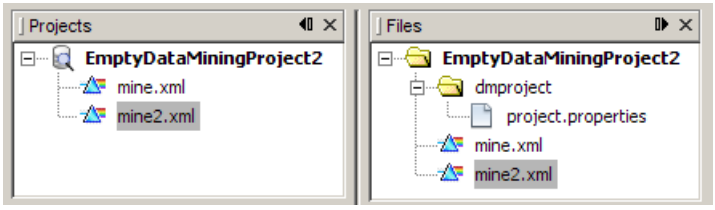
Projekty v prostředí NetBeans jsou tvořeny adresářem na disku s projektovými daty uvnitř. Samozřejmě aplikace musí nějakým způsobem rozpoznat, zda daný adresář je projektem. Proto je prvním krokem vytvoření *ProjectFactory*. Jedná se o implementaci *org.netbeans.spi.project.ProjectFactory* rozhraní, která se zaregistruje do globálního *Lookup* kontextu. Kdykoliv je pak otevřen dialog pro otevření projektu, jsou všechny ProjectFactory dotazovány, zda daný adresář je projektem. Pokud ano, je tato ProjectFactory použita pro správu daného projektu.

Registrace ProjectFactory do globálního kontextu se provádí druhým způsobem, přes adresář *META-INF/services*. V adresáři *META-INF/services* se vytvoří soubor s názvem rozhraní, které implementuje. V tomto případě se soubor jmenuje *org.netbeans.spi.project.ProjectFactory*. Obsahem tohoto souboru je jméno implementující třídy,

cz.vutbr.fit.dataminer.projects.DataMiningProjectFactory. Tímto se při startu aplikace do globálního kontextu dostanou instance daného rozhraní. Takto lze pomocí návrhového vzoru Dependency Injection vytvořit vazbu mezi moduly, které o sobě nic neví, ale komunikují přes společné rozhraní. Pro podporu dalšího typu projektů stačí vytvořit příslušnou implementaci *ProjectFactory*.

7.5.1 Struktura Data Mining projektu

Data mining projekt se rozpoznává na základě přítomnosti podřízeného adresáře *dmproject*. Rozpoznávání by mělo být rychlé, protože se volá při každé změně adresáře v dialogu. Struktura a logické zobrazení projektového adresáře je definováno implementací *LogicalViewProvider* rozhraní. DMSL soubory, které jsou viditelné pro uživatele, se nacházejí v adresáři projektu. Podadresář *dmproject* je skryt a obsahuje soubor *project.properties*, do kterého se ukládají různá uživatelská nastavení projektu.



Obrázek 19 - Logická a fyzická struktura projektu

Soubor *project.properties* obsahuje uživatelská nastavení, která platí pro celý projekt. V současné době obsahuje parametry definující připojení k databázi. V budoucnu lze tento soubor nahradit například XML souborem, pomocí něhož bude možné ukládat libovolné vlastnosti a nastavení projektu.

7.5.2 Šablona projektu a průvodci nového projektu

Z běžně používaných aplikací je uživatel zvyklý na možnost pohodlného vytvoření nového projektu pomocí průvodce. Platforma tuto možnost samozřejmě podporuje. V *layer.xml* se zaregistruje šablona nového Data Mining projektu, která je zabalena v zip souboru. Zaregistruje se zde i implementace rozhraní průvodce, který může provádět činnost v 1 až N krocích. Stačí implementovat rozhraní pro průvodce a výsledná implementace je integrována do standardního průvodce novým projektem.

Průvodce pro data mining projekt obsahuje dva kroky:

1. Specifikace umístění projektu v souborovém systému
2. Definice připojení k databázi (používá se Database Explorer modul)

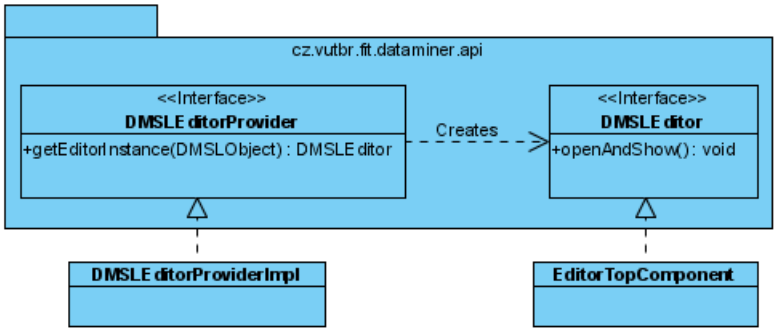
7.6 Editor, paleta komponent a navigace

Editor, paleta komponent a navigace jsou součástí jednoho modulu. Důvodem je to, že tyto části spolu úzce souvisejí a je vhodné je integrovat do jednoho celku. Editor je implementačně nejnáročnější částí řešení. Vizuálně zobrazuje dolovací úlohu pomocí grafu. Je potřeba zajistit aby veškeré změny provedené editací grafu byly zaznamenány a editovaný soubor byl označen jako modifikovaný.

7.6.1 Editor

Editor v prostředí NetBeans je vizuální komponenta, která dědí od třídy *TopComponent*. Její umístění je ve výchozím nastavení v centrální části aplikace a slouží jako hlavní pracovní plocha. Pro to, aby se po poklepání na DMSL soubor v projektové části aplikace otevřel editor, je potřeba implementovat rozhraní definované v API vrstvě. Tímto se eliminují jakékoliv závislosti mezi moduly a je možné nahradit implementaci editoru za jinou, aniž by to nabouralo nebo jinak ovlivnilo zbytek aplikace.

Zpřístupnění editoru ostatním částem aplikace je provedeno obdobným způsobem jako v případě registrace *ProjectFactory* v modulu pro podporu projektů. Editor implementuje rozhraní *DMSLEditor* definované v API vrstvě. Implementace *DMSLEditorProvider* zajišťuje vytváření instancí editoru pro jednotlivé objekty *DMSLObject*.



Obrázek 20 - Implementace API rozhraní editoru

Implementace *DMSLEditorProvider* se zaregistruje do globálního Lookup tak, že se do adresáře *META-INF/services* vloží soubor *cz.vutbr.fit.dataminer.api.DMSLEditorProvider* v němž je uložen název implementující třídy *DMSLEditorProviderImpl*.

Jak bylo popsáno v kapitole pro podporu DMSL souborů, je otevírání souborů řešeno pomocí *OpenCookie*. Toto rozhraní obsahuje pouze jednu metodu *open()*. Ta by měla otevřít DMSL soubor v DMSL editoru. Způsob jakým získá instanci editoru z globálního Lookup je následující:

```
public class DMSLOpenCookie implements OpenCookie
{
  private DMSLObject dataObject;
  public DMSLOpenCookie(DMSLObject obj)
  {
```

```
    this.dataObject = obj;
  }

  public void open()
  {
    DMSLEditorProvider editorProvider =
            Lookup.getDefault().lookup(DMSLEditorProvider.class);
    if (editorProvider != null)
    {
      DMSLEditor editor = editorProvider.getEditorInstance(dataObject);
      editor.openAndShow();
    }
    else
    {
      ErrorManager.getDefault().log("No DMSL editor installed.");
    }
  }
}
```

Jak je patrné z příkladu, použití je velmi jednoduché. Je zachováno volné vázání částí aplikace v jeden celek (tzv. loose coupling). Po otevření editoru se vytvoří paleta komponent, *DMSLObject* se předá scéně pro vykreslení a vytvoří se náhled.

7.6.1.1 Vykreslovací mechanismus GraphScene

Graf DMSL dokumentu se vykresluje za pomocí Visual Library knihovny. Knihovna kreslí za pomocí tzv. scén a jednou z nich je i *GraphScene*, která vytváří graf s uzly a hranami. Díky použití Generics, které jsou součástí Javy od verze 5, je graf definován tak, že uzly tvoří objekty třídy *MiningPiece* a hrany tvoří objekty třídy *Edge*. Jelikož jsou hrany orientované, třída *Edge* v sobě obsahuje odkaz na zdrojový a cílový uzel.

Práce s takovým grafem je velmi jednoduchá, protože se pracuje s konkrétními instancemi *MiningPiece*, a není potřeba žádných tříd, které by tyto objekty obalovaly. Uzly i hrany můžou poskytovat kontextové menu, lze je přesouvat, odstraňovat apod. Po poklepání na příslušný uzel se otevře dialogové okno tohoto *MiningPiece*.

Pří jakékoliv změně grafu je potřeba nastavit příznak, že DMSL soubor byl modifikován. Důležitým prvkem, který bylo potřeba implementovat, je schopnost grafu informovat o každé změně jeho obsahu. To lze implementovat uvnitř grafu a nastavovat tento příznak tam, ale vhodnějším přístupem je použití tzv. návrhového vzoru Observer. Graf dovoluje zaregistrovat implementace rozhraní *ModificationListener*. Tyto implementace pak dostávají události o různých změnách v grafu. Událostí, které rozhraní nabízí, jsou definovány následovně:

```
public static interface ModificationListener
{
  public void pieceAdded(MiningPiece miningPiece, Point location);
  public void pieceRemoved(MiningPiece miningPiece);
  public void pieceOpened(MiningPiece miningPiece);
  public void pieceMoved(MiningPiece miningPiece, Point original, Point dest);
  public void edgeAdded(Edge edge);
  public void edgeRemoved(Edge edge);
}
```

7.6.2 Paleta komponent

Paleta komponent je tvořena instancemi abstraktní třídy *MiningPiece*, které se načítají z *MiningPieceRegistry*. Způsob definice a řazení do kategorií je popsán v části pro API vrstvu v kapitole 7.3.

7.6.3 Navigace pomocí náhledu grafu

Navigace v grafu se provádí pomocí miniatury zobrazeného grafu. Tuto miniaturu poskytuje implementace *GraphScene*. Problémem je integrace této komponenty do standardního panelu *Navigator*, který poskytuje platforma NetBeans. To je realizováno implementací rozhraní *NavigatorPanel*, které poslouchá změnu kontextu, dokud se v aktuálním kontextu neobjeví instance *DMSLObject* rozhraní. Jakmile tato událost nastane, znamená to, že aktuální okno poskytuje náhled. Implementace panelu navigátoru se registruje v *layer.xml* pro daný MIME typ.

7.7 Akce pro spuštění dolovací úlohy

Akce pro spuštění dolovací úlohy se nachází v samostatném modulu. Dokud DMSL dokument neobsahuje dolovací modul, akce nemůže být povolena. Je tedy nutné, aby akce byla citlivá na kontext (context aware action). To je implementováno návrhovým vzorem Observer, kde akce naslouchá na změny objektu *DMSLObject*. Jakmile je v DMSL dokumentu dostupný dolovací modul, akce je povolena. Vyvolání této akce způsobí spuštění dolovací úlohy. Průběh je zobrazen pomocí modulu *Progress Bar*, který je standardní součástí platformy.

8 Příklad integrace dolovacího modulu

Tato kapitola detailně popisuje způsob vytvoření dolovacího modulu a jeho integrace do aplikace. Dolovací modul tak může vytvořit kdokoli další, dokonce bez zdrojových kódů zbytku aplikace. Stačí mít API vrstvu a knihovnu jádra v binární formě. Výsledný modul se pak jednoduše nainstaluje jako zásuvný modul. Veškeré níže popsané kroky se týkají práce v NetBeans IDE.

8.1 Vytvoření modulu NetBeans

Prvním krokem je vytvoření projektu pro NetBeans modul. V dialogovém okně pro vytvoření nového projektu se vybere položka *Module Project* v kategorii *NetBeans Plug-in Modules*. Umístění projektu je libovolné, ale je potřeba zaškrtnou volbu *Standalone Module*. Tato volba znamená, že modul bude vyvíjen jako samostatný plugin.

Tímto se vygeneruje kostra projektu. Po tomto kroku následuje implementace samotného dolovacího modulu pro jádro. Dolovací modul je možné vytvořit nezávisle v samostatném projektu a přidat jej jako JAR knihovnu do vytvářeného NetBeans modulu. Způsob vytvoření dolovacího modulu je popsán v práci pana Ing. Doležala [3], proto se zde nebudeme zabývat tímto problémem.

8.2 Implentace abstraktní třídy MiningPiece

Pro to, 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 potřeba pomocí *setXxx()* metod nastavit jméno pro zobrazení a popis prvku (nejlépe z lokalizačního Resource Bundle). Dále je potřeba inicializovat cesty k obrázkům, které se použijí k vizualizaci v paletě a editoru.

Nyní je potřeba implementovat abstraktní metody. Je potřeba zajistit, aby v případě vložení dolovacího modulu do DMSL dokumentu byl inicializován *Kernel*. To lze provést v metodě *beforeAcceptEvent()*, která se volá před vložením do grafu. Návratovou hodnotou lze tuto operaci přerušit (např. v případě, že uživatel zadal nesprávný údaj apod.):

```
private TestMM miningModule = new TestMM();  // Implementace dolovacího modulu
public boolean beforeAcceptEvent()
{
  getKernel().setMiningModule(miningModule);
  return true;
}
```

Dalším krokem je implementace metody, která se volá při poklepání na tento uzel v grafu a měla by zobrazit dialog pro nastavení parametrů. 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:

```
protected void openCustomizingDialog()
{
  if (miningModule == null)
  {
    // Internacionalizace z Resource Bundle
    String message =
        NbBundle.getBundle(TestMiningModule.class).getString("MSG_NoModule");
    NotifyDescriptor d =
        new NotifyDescriptor.Message(message, NotifyDescriptor.ERROR_MESSAGE);
    DialogDisplayer.getDefault().notify(d);
    return;
  }

  DialogDescriptor descriptor = new
      DialogDescriptor(miningModule.getParametersGUI(), getDisplayableName());
  descriptor.setModal(true);
  descriptor.setOptions(new Object[]{DialogDescriptor.OK_OPTION});

  Dialog dialog = DialogDisplayer.getDefault().createDialog(descriptor);
  dialog.setSize(580,430);
  dialog.setLocationRelativeTo(null);
  dialog.setVisible(true);
}
```

8.3 Integrace do aplikace

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.

9 Závěr

Výsledná aplikace je prvním prototypem produktu, který umožňuje získávat znalosti z databází. Uživatelské rozhraní má díky použití NetBeans platformy velký potenciál. Právě výběr použité technologie byl velmi dobrou volbou. Nutí vývojáře použít určitý styl programování, který klade důraz na modularitu a volné vázání pomocí programování do rozhraní. Jakmile je hotová koncepce aplikace a definována API vrstva, je velmi jednoduché přidávat funkčnost, aniž by tento krok naboural stávající aplikaci. Mezi nevýhody použití tohoto přístupu patří poměrně dlouhá doba potřebná pro studium platformy. Nedostupnost dobré dokumentace lze odůvodnit tím, že se jedná o mladou technologii. V budoucnu tato nevýhoda zmizí, protože programování s využitím Rich Client Platform se stává velmi populárním přístupem a dostupnost dobré dokumentace se stává otázkou času.

Podle mého názoru se použití Rich Client Platform vyplatí při vývoji středně velkých až velkých desktopových aplikací. Vývoj je pak efektivní a využívají se otestované a robustní existující komponenty. Mezi hlavní přínosy bych vyzdvihl důraz na modularitu celého řešení.

Vytvořená aplikace, mimo již zmíněné vlastnosti, obsahuje systém nápovědy Java Help. Jedná se o kontextovou uživatelskou nápovědu, přístupnou přes menu nebo prostřednictvím klávesy F1. Dále je aplikace jednoduše lokalizovatelná do jakéhokoliv jazyku. Stačí k tomu přeložení řetězců uložených v tzv. Resource Bundle, a následně jejich instalace do aplikace. To jsou často opomíjené prvky, které ale určují kvalitu výsledného produktu. Velkým přínosem je systém automatických aktualizací a možnost online instalace rozšiřujících modulů. To je typický příklad využití „rich“ komponent platformy.

Velkou překážkou, která velmi ztížila vývoj, byla absence jakékoliv API dokumentace existující knihovny pro dolování dat, nad kterou je tato aplikace postavena. Právě tato část řešení by měla být nejlépe zdokumentována. Vytvořit knihovnu je velmi obtížný úkol a je rovněž potřeba vytvořit kvalitní návrh API, jelikož je knihovna vždy využívána třetí stranou. Během implementace bylo nutné řešit i jeden neobvyklý problém spojený se zaváděním tříd. Knihovna pro dolování využívá kompilátor Javy z programového rozhraní. Při spuštění v prostředí NetBeans modulu docházelo k výjimkám, které byly spojené s neviditelností některých tříd JRE. To bylo zapříčiněno tím, že moduly používají vlastní zavaděč tříd (class loader), a pravděpodobně některé balíčky JRE byly maskovány. Problém nebyl nikde zdokumentován, a proto bylo použito náhradní řešení takové, že se potřebné balíčky kompilátoru staly součástí aplikace. Tento problém by bylo vhodné zveřejnit v diskusní skupině okolo vývoje NetBeans platformy. Byl by to dobrý příklad toho, jak open source komunita dokáže poskytovat podporu sama sobě, a jakým způsobem se případná chyba platformy opraví.

Samozřejmě aplikace není hotovým produktem a má velmi omezenou funkčnost. Tato omezení jsou dána především návrhem knihovny jádra, která je velmi omezující v použití. Rovněž chybí smysluplná implementace dolovacích modulů pro databázový server Oracle, které by realizovaly konkrétní typ dolovací úlohy. V následující kapitole jsou uvedeny požadavky na funkčnost a návrh architektury pro další vývoj. Na CD disku přiloženém k této práci se nachází mimo jiné Flash animace, která demonstruje použití aplikace včetně online instalace dolovacího modulu.

9.1 Návrh pro další vývoj

Současný stav aplikace není v praxi použitelný, ale další iterací by se mohl stát vcelku použitelným nástrojem pro získávání znalostí z databází. Zde jsou uvedeny požadavky na funkčnost pro další verzi aplikace, které úzce souvisejí s možnostmi jádra pro dolování:

1. **Zobrazení dat** – je potřeba vytvořit podporu pro zobrazení surových dat v jakémkoli kroku získávání znalostí. Bylo by vhodné, aby uživatel mohl prohlížet výsledky různých kroků, které aplikovaly nějakou operaci na vstupní data. Například zobrazit výsledky *MiningFields* uzlu, který aplikoval funkci na vstupní data.
2. **Spuštění dolovací úlohy po určitý krok** – při označení určitého uzlu grafu by mělo být možné spustit dolovací úlohu po tento krok a dále nepokračovat. Informace spojené s tímto uzlem by měly být dostupné tak, jak je to navrženo v předchozím bodě. Obdobnou funkčnost nabízí například produkt SAS Enterprise Miner.
3. **Zobrazení informací o vstupních datech** – při výběru datových polí pro dolování by měla být dostupná informace o velikosti dat a typu hodnot daného pole.
4. **Možnost importu dat z různých úložišť** – v současném stavu lze použít jako vstupní data pouze data uložená v databázi Oracle. Jelikož Oracle databáze je potřebná pouze pro provádění dolovacích operací, není důvod pro toto omezení. Aplikace by měla poskytovat možnost importu jak z Excel nebo CSV souborů, tak z jiných relačních databází.
5. **Vytvoření dolovacích modulů** – aplikace by měla poskytovat sadu běžných dolovacích modulů, které realizují určité typy dolovacích úloh. Tyto moduly by měly plně podporovat serializaci/deserializaci jejich konfigurace a výsledků dolování. Bylo by rovněž vhodné, aby z výsledků dolování šlo vytvořit tiskové sestavy.
6. **Rozšíření definice DMSL dokumentů** – v současné době návrh DMSL dokumentů nedovoluje uložení meta informací spojených s prezentací dokumentu (pozice uzlu, vazby mezi uzly apod.). Tyto informace jsou nyní ukládány ve formě XML komentáře. Bylo by vhodné zvážit rozšíření definice DMSL dokumentu o možnost uchovávat meta informace související s vizualizací nebo prezentací struktury dokumentu.
7. **Statistická analýza dat** – poskytnutí statistických informací o datech.

8. **Možnost aplikovat další transformace na transformovaná data** – aplikace by měla umožnit volnější řazení bloků, které provádějí modifikace v datech. To znamená, že například výstup bloku *Actions* může být použit jako vstup do dalšího bloku *Actions*.
9. **Implementace externích funkcí** – rozšíření aplikace tak, aby podporovala externí funkce podobným způsobem, jak je to u integrace dolovacích modulů.

9.1.1 Refaktoring knihovny jádra

Pro to, aby předchozí body byly realizovatelné, je potřeba zásadním způsobem přepracovat rozhraní jádra pro dolování. V následujících několika bodech jsou shrnuty hlavní prvky, které by knihovna měla splňovat:

1. **Striktní oddělení implementace funkčnosti jádra od veřejného rozhraní** – knihovna není nijak vrstvená a aplikační logika se míchá s reprezentací dolovací úlohy. Dotazy do databáze se skládají ve formě řetězců a neexistuje žádná vrstva zapouzdřující tuto funkci. Vhodným přístupem by mohlo být použití obdoby třívrstvé architektury:
    - Veřejná část knihovny (prezentační vrstva), která slouží jako rozhraní pro komunikaci s okolím, zadávání dolovací úlohy, spuštění úlohy apod.
    - Servisní vrstva (aplikační vrstva) sloužící jako implementace procesů a aplikační logiky.
    - Vrstva pro přístup k databázi by měla implementovat komunikaci s Oracle databází a Oracle Data Mining rozhraním. Vhodným přístupem by mohlo být vytvoření programové abstrakce nad JDBC rozhraním.
2. **Identifikace elementů** – elementy používané v procesu definice dolovací úlohy nejsou nijak svázány a jsou ukládány jako různé kolekce objektů. To je hrubé pochybení v návrhu. Dokonce neexistuje žádná identifikace jednotlivých elementů, ačkoli tato identifikace je součástí DMSL XML dokumentu. Elementy a části DMSL dokumentu by měly být v paměti reprezentovány hierarchickou strukturou tříd, která odpovídá struktuře DMSL.
3. **Konfigurace připojení do databáze s ODM mimo zdrojové kódy** – připojení do Oracle databáze provádějící dolovací operace je „natvrdo“ definováno ve zdrojových kódech knihovny.
4. **Vytvoření kvalitní API dokumentace ve formě JavaDoc**

Literatura

[1] Zendulka, J., Bartík, V., Lukáš, R., Rudolfová, I.: Získávání znalostí z databází. Studijní opora, 2006, Fakulta informačních technologií VUT v Brně.

[2] Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Second edition. Elsevier Inc., 2006

[3] Doležal, J.: Jádro systému pro dolování z dat v prostředí Oracle. Diplomová práce, 2006, Fakulta informačních technologií VUT v Brně.

[4] Hornick, M., Marcadé, E., Venkayala, S.: Book excerpt: Java Data Mining Concepts, 12.3.2007, http://www.javaworld.com/javaworld/jw-02-2007/jw-02-jdm.html

[5] Oracle Documentation Library – Data Warehousing and Data Mining, 14.11.2006, http://www.oracle.com/pls/db10g/portal.portal_demo3?selected=6

[6] White paper: Oracle Data Mining, 14.11.2006, http://www.oracle.com/technology/products/bi/odm/index.html

[7] Hromčík, P.: Systém pro získávání znalostí z databází. Diplomová práce, 2003, Fakulta informačních technologií VUT v Brně.

[8] Giudici, F.: From Pain to Gain. NetBeans Magazine, May, 2007, s. 40-51.

[9] Boudreau, T., Glick, J., Green, S., Vaughn, S., Woehr, J.: NetBeans – The Definite Guide. First edition. O'Reilly, 2002, ISBN 0-596-00280-7

[10] NetBeans Modules and Rich-Client Application Learning Trail, 17.4.2007, http://platform.netbeans.org/tutorials/index.html

[11] Wiki stránky NetBeans Platform, 17.4.2007, http://wiki.netbeans.org/wiki/view/NetBeansPlatform

[12] Dokumentace pro Visual Library, 17.4.2007, http://graph.netbeans.org/documentation.html

[13] Java API Documetation, 17.4.2007, http://java.sun.com/javase/6/docs/api/

Seznam příloh

Příloha 1. CD disk s následujícím obsahem:

- zdrojové kódy aplikace
- API dokumentace
- vývojové prostředí
- sestavená distribuce vyvíjené aplikace ve formě ZIP souboru
- demonstrace použití aplikace ve formě Flash animace
- tato dokumentace v elektronické podobě