

# Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 3, 842 16 Bratislava 4

---

## Znalostný manažment na báze technológie .NET

(Tímový projekt)

**Analýza, špecifikácia požiadaviek a hrubý návrh**

**Odbor:** Softvérové inžinierstvo

**Autori:** Bc. Ondrej Hirjak

Bc. Peter Nociar

Bc. Michal Okresa

Bc. Ľuboš Pazdera

Bc. Juraj Petráš

Bc. Richard Schwartz

**Tím:** Lucky Number 7

**Dátum:** 16. novembra 2006

## Obsah

1.	Úvod.....	1
1.1	Použitá notácia .....	2
2.	Analýza .....	1
2.1	Úvod do manažmentu znalostí .....	1
2.1.1	Základné pojmy .....	2
2.1.2	Základné prvky manažmentu znalostí .....	3
2.1.3	Životný cyklus znalostí.....	5
2.1.4	Clips .....	6
2.1.5	Protégé.....	10
2.1.6	Iné znalostné systémy .....	16
2.2	Indexovanie a vyhľadávanie .....	17
2.2.1	Inverzný index .....	17
2.2.2	Hľadanie v indexoch .....	17
2.2.3	Správnosť a priepustnosť .....	18
2.2.4	DotLucene – analýza vyhľadávacieho systému.....	19
2.2.5	The Gnome Data Mine .....	23
3.	Predbežná špecifikácia požiadaviek .....	1
3.1	Prípady použitia.....	2
3.1.1	Identifikácia hráčov (používateľov systému).....	2
3.1.2	Prípady použitia.....	3
4.	Návrh .....	1
4.1	Hrubý návrh systému.....	1
4.1.1	Fázy manažmentu znalostí.....	1
4.1.2	Vyhľadávanie a metamodel .....	1
4.1.3	Príklad fungovania navrhovaného systému.....	2
	Prílohy.....	2
A.	Použitá literatúra.....	2

## Tabuľky

Tab. 1: Súhrn kompatibilných a nekompatibilných vlastností Protégé a CLIPSu.....	15
Tab. 2: Funkcionálne a nefunkcionálne požiadavky na systém.....	1

## Obrázky

Obr. 1: Hierarchia pojmov v znalostnom manažmente.....	3
Obr. 2: Rozdelenie manažmentu znalostí na bloky.....	4
Obr. 3: Model SECI.....	6
Obr. 4: Príklad používateľského rozhrania programu Protégé 3.2.....	11
Obr. 5: Príklad používateľského rozhrania programu Protégé 3.2 – editor rámcov .....	12
Obr. 6: Príklad rozhrania programu Protégé 3.2 – vizualizácia vzťahov medzi triedami ontológie.....	13
Obr. 7: Všeobecný graf závislosti správnosti od priepustnosti.....	19
Obr. 8: Diagram prípadov použitia.....	2
Obr. 9: Rozdelenie manažmentu znalostí na skupiny procesov.....	1
Obr. 10: Proces transformácie dokumentácií do metamodelu .....	2
Obr. 11: Redakčný systém – vyhľadávanie v archíve dokumentov.....	2
Obr. 12: Produkčný systém – generovanie dokumentov na základe projektu. (príklad iného možného použitia metamodelu).....	2

# 1. Úvod

Znalosť môžeme považovať za organizovanú informáciu využiteľnú na riešenie problémov.

Pre organizácie sa stávajú znalosti stále dôležitejšími pretože:

- Organizácie sa stávajú znalostne intenzívnymi
- Znalosti umožňujú byť na čele zmien
- Zložitost' riešených úloh narastá
- Globalizáciou trhu sa musia organizácie prispôbovať rýchlejšie
- Organizácie dbajúce na rozvoj znalostí majú väčšiu šancu prežiť
- Znalosti sú mobilné (v hlavách zamestnancov) a je vhodné zaviesť zdieľanie znalostí v rámci organizácie

Oproti ostatným podnikovým zdrojom (finančným, materiálnym, ...) majú znalosti svoje špecifiká medzi ktorých napríklad patrí:

- Sú nehmotné a ťažko merateľné
- Sú pomíňajúce sa
- V procesoch sa znalosti nespotrebovávajú, niekedy naopak používaním rastú
- Majú veľkú šírku dopadu v organizáciách
- Nemôžu byť kúpené na trhu alebo burze
- Nie je možné ich riadiť (znalostne orientovaní manažéri neriadia znalosti, ale iba prostredie v ktorom sa nachádzajú)

P. Senge (Senge, 1990) vo svojej práci považuje schopnosť učiť sa rýchlejšie ako konkurenti za jedinú a rozhodujúcu konkurenčnú výhodu. Tá je skrytá v ľudskom kapitály, ktorý je nositeľom znalostí a stáva sa najdôležitejším výrobným prostriedkom súčasnosti.

Z týchto informácií nás môže jednoducho napadnúť idea: „Prepojme tých, čo vedia s tými, čo potrebujú vedieť“. Takáto formulácia by sa dala nazvať aj definíciou manažmentu znalostí, ale tú si spomenieme v inej kapitole. My sa v našom projekte nechceme zameriavať na riadenie znalostí z pohľadu ľudských zdrojov, ale skôr upriamime našu pozornosť na podporu tvorby dokumentácie. Ideu preformulujeme na neformálne heslo tímu Lucky 7, ktoré sa s nami bude spájať až do konca projektu:

**„Prepojme to, čo vieme s tým, čo potrebujeme vedieť“.**

V konečnom dôsledku využijeme ľudský kapitál, ktorý bude zužitkovaný vo vytvorenej dokumentácii. Naším cieľom bude:

- Získavanie znalostí a lokalizácia expertov z neštruktúrovaných, resp. štruktúrovaných dát

- Pomoc firmám v ich snahe o štrukturalizáciu, filtráciu, kontextualizáciu báz dokumentov a ďalších znalostných artefaktov, učenie sa z nich, ako aj navyšovanie informácií uložených v informačných systémoch a zúročovanie znalostí pracovníkov organizácie

## 1.1 Použitá notácia

### Diagramy prípadov použitia



- prípad použitia: činnosť, ktorú vykonáva užívateľ v softvérovom systéme



hrac

- užívateľ systému, nejedná sa o konkrétnu osobu, popisuje užívateľa vo vzťahu k jednotlivým prípadom použitia



- väzba medzi užívateľom a prípadmi použitia, šípka znázorňuje smer inicializácie, nie tok alebo prenos údajov

## 2. Analýza

S ohľadom na rozsah projektu a jeho súčasti je fáza analýzy rozčlenená do dvoch častí.

Prvá časť tejto kapitoly sa venuje manažmentu znalostí. V tejto podkapitole nadviažeme na motivačný úvod a uvedieme základné informácie o manažmente znalostí. Uvedieme si aj všeobecne známe fakty, pretože všetci členovia tímu sa s touto problematikou stretávajú prvýkrát. Nakoľko je manažment znalostí obsírna a náročná oblasť na pochopenie, považujeme túto podkapitolu za veľmi dôležitú na úvodné preniknutie do problémovej oblasti a pochopenie domény problému. Jej súčasťou je aj analýza systémov, ktoré slúžia na implementáciu riešení z tejto oblasti.

V druhej časti si povieme niečo o vyhľadávaní informácií v dokumentoch a rovnako ako v prvej časti aj tu sa budeme venovať aj existujúcim riešeniam z danej oblasti.

### 2.1 Úvod do manažmentu znalostí

Zavádzania manažmentu znalostí do praxe podnecujú hlavne tieto tri faktory (Duffner, 2000):

1. Od času potrebného na nasadenie výrobku na trh je závislý úspech či neúspech daného produktu. Pre spoločnosti je nevyhnutné využiť výhody prvotného iniciátora na trhu alebo aspoň rýchle vytvorenie konkurencie. Z tohto dôvodu je pre firmy nevyhnutné plne využitie ich znalostí týkajúcich sa trhu, zákazníkov, konkurenčných firiem a dostupných technológií.
2. Iná situácia vzniká pri fúzii alebo reštrukturalizácii firiem, ktoré hlboko závisia na nadobudnutých znalostiach. Pri týchto operáciách je veľmi dôležité vedieť preniesť znalosti z jednej organizácie do druhej a určiť, ktoré zo znalostí sú užitočné pre čo najrýchlejší profit firmy a čo najdlhšie prežitie na trhu.
3. Moderné informačné technológie poskytujú obrovské množstvo informácií a pomerne jednoduchý prístup k nim. Týchto informácií je veľké množstvo a je priveľmi zložitá nájsť požadovanú informáciu dostatočne rýchlo na to, aby sme ju mohli využiť a aby námaha vynaložená na jej získanie neprekročila užitočnú hodnotu tejto informácie.

Pri snahe o presné vymedzenie manažmentu znalostí narážame na komplikáciu súvisiacu s faktom, že manažment znalostí čerpá zo širokej škály disciplín (Paralič, 2006):

- Umelá inteligencia, expertné a znalostné systémy
- Počítačom podporovaná spolupráca (groupware)
- Informatika, kognitívne vedy, filozofia
- Správa dokumentov
- Systémy pre podporu rozhodovania
- Reinžiniering firemných procesov
- Riadenie ľudských zdrojov
- Organizačná kultúra a pod.

Formálne manažment znalostí môžeme zdefinovať takto: „Znalostný manažment zahŕňa efektívne prepojenie tých, čo vedia s tými čo vedieť potrebujú a premenu osobných znalostí na znalosti organizácie“ (Trunecek, 2001). Prípadne: „systematický proces hľadania, výberu, organizovania, destilovania a prezentovania informácií spôsobom, ktorý zlepšuje porozumenie pracovníka špecifickej oblasti záujmu. Je to umenie identifikovať latentné znalosti a nájsť cestu k ich zdieľaniu“ (Devenport, 1998).

Zložitosť manažmentu znalostí nie je možné zjednodušovať, lebo je nevyhnutná k pochopeniu jeho podstaty. Práve preto je možné ho vnímať z rôznych druhov pohľadu (Paralič, 2006):

1. konceptuálny pohľad – zahŕňa najmä problémy súvisiace s definíciou manažmentu znalostí, znalostné princípy, resp. celkový rámec. Ide o teoretickú rovinu.
2. procesný pohľad – definovanie a pochopenie jednotlivých znalostných procesov. Základ úspechu implementácie manažmentu znalostí do akejkoľvek organizácie spočíva aj vo formalizovaní, distribuovaní, zdieľaní, aplikovaní a obnove organizačných znalostí.
3. technologický pohľad – výskum spôsobov, akými môžu jednotlivé informačné, komunikačné a znalostné technológie pomôcť pri manažmente znalostí (intranet, systémy DMS, help-desk aplikácie, ...).
4. organizačný pohľad – orientuje sa na vyriešenie problematiky charakterizácie znalostnej organizácie a pre ňu vhodnej formálnej aj neformálnej organizačnej štruktúry, úloh a zodpovedností, organizačného učenia.
5. implementačný pohľad – úvahy o rôznych metódach a postupoch ktoré umožňujú dosiahnuť úspešné zavedenie manažmentu znalostí.
6. manažérsky pohľad – rôzne postupy, ktoré vedú k zavedeniu manažmentu znalostí do praxe, najmä manažérske postupy, meranie a hodnotenie intelektuálneho kapitálu, prémie, platové a motivačné systémy, vytvorenie vhodnej podnikovej kultúry.

### 2.1.1 Základné pojmy

Postupné narastanie dôležitosti a ceny elementu je možné vidieť na (Obr. 1).

#### ***Dáta***

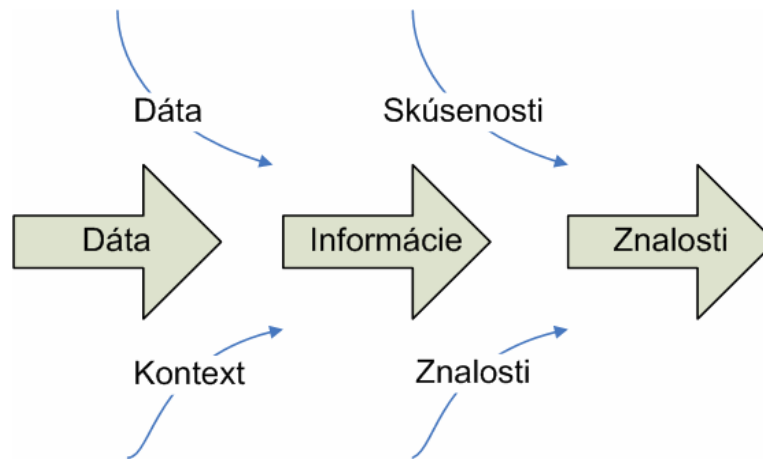
Dáta sú súbor znakov predstavujúcich diskkrétne objektívne fakty.(napr. 60%) Ako také majú dáta veľmi malú výpovednú hodnotu, ale dajú sa jednoducho uchovávať a spracovávať vo výpočtových systémoch.

#### ***Informácie***

Informácie sú dáta ktoré sú vložené do kontextu. (napr. „Ceny akcií spoločnosti XYZ vzrástli o 60%.“) Informácie majú väčšiu hodnotu ako dáta no môžu spôsobovať dvojznačnosť alebo stratiť hodnotu, ak príjemca informácie nerozumie kontextu.

**Znalosti**

Znalosti sú informácie vložené do konkrétneho kontextu a sú hodnotné pre toho kto ich pozná. Umožňujú mu vykonávať činnosti, ktoré by bez týchto znalostí vykonávať nemohol. (napr. „Ak ceny akcií spoločnosti XYZ vzrastú o 60% je výhodné ich predať.“)



Obr. 1: Hierarchia pojmov v znalostnom manažmente

Podľa (Nanoka et al, 1995) sa znalosti delia na dva typy: explicitné a tiché.

**Tiché znalosti**

Sú také ktoré sa veľmi ťažko definujú a šíria. Vznikajú na základe skúseností, pozorovania, myslenia a intuície. Ich častým znakom je, že sa nedajú merať a sú nediskutabilné.

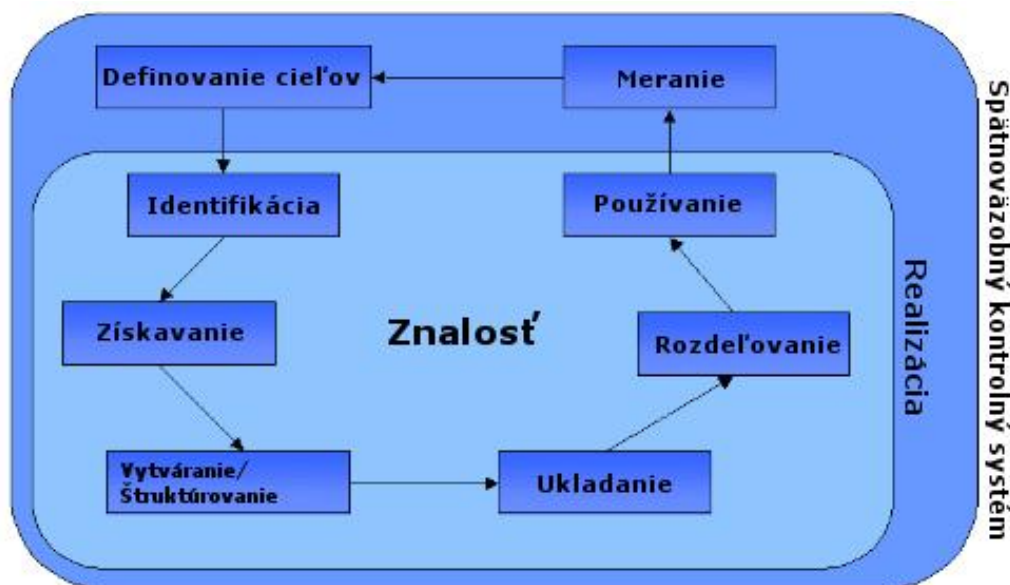
**Explicitné znalosti**

Môžu byť kategorizované, uchovávané a šírené. Sú to napríklad: správy o zákazníkoch, databáza kvalifikácií a zručností zamestnancov a pod. Väčšina programov na báze MZ pracuje práve s týmito znalosťami.

### 2.1.2 Základné prvky manažmentu znalostí

(Probst et. al, 1997) popisuje základnú úlohu manažmentu znalostí pomocou dvoch okruhov ako je znázornené na obrázku (Obr. 2).





Obr. 2: Rozdelenie manažmentu znalostí na bloky

Vonkajší okruh predstavuje spätnoväzobný kontrolný systém, ktorý reprezentuje tradičný proces riadenia a pozostáva z **definovania cieľov, realizácie a merania**.

Realizácia predstavuje vnútorný okruh a skladá sa z **identifikácie, získavania, vytvárania / štruktúrovania, uchovávaní, rozdeľovania (distribúovania) a používania** znalostí.

Výhody takéhoto členenia MZ sú nasledovné:

- rozdeľuje proces riadenia na logické entity,
- ponúka začiatočný bod pre realizáciu,
- poskytuje definované usporiadanie „znalostného problému“ a tým možnosť rozdeliť riešenie na menšie logické celky.

V krátkosti si jednotlivé časti popíšeme.

### ***Definovanie cieľov***

V tomto kroku sú definované strategické a funkčné ciele znalostného manažmentu pre konkrétnu organizáciu. Sú definované strategicky dôležité okruhy znalostí.

### ***Identifikácia znalostí***

V tomto procese sa identifikujú a analyzujú hlavné dátové a informačné zdroje ktoré existujú v prostredí organizácie. Cieľom je vytvoriť transparentný náhľad na tieto zdroje tak aby sa predišlo vzniku nedostatočnému prístupu alebo redundancie.

### ***Získavanie znalostí***

V tomto kroku sa získavajú znalosti z analyzovaných zdrojov. Zdroje môžu byť ľudia, dokumenty, databázy a ich rôzne kombinácie. Tiež môžu byť rozdelené na interné a externé.

### ***Vytváranie a štruktúrovanie znalostí***

Táto časť slúži na analyzovanie a ohodnotenie získaných znalostí a vytvára sa štruktúra znalostí a organizačnej pamäte.

### ***Ukladanie znalostí***

Znalosti sú ukladané do organizačnej pamäte. Táto pamäť zahŕňa všetky druhy znalostí z rôznych zdrojov.

### ***Rozdeľovanie znalostí***

Hlavná otázka tejto časti je: Kto potrebuje tú ktorú znalosť a za akým účelom? Ako môžeme zjednodušiť proces rozdeľovania znalostí? Je dôležité nezabudnúť, že cieľom MZ je poskytovanie správnych znalostí správnym osobám za čo najkratší čas, s cieľom podporiť rozhodovanie danej osoby.

### ***Používanie znalostí***

Tento blok má za úlohu zabezpečiť, aby všetci dostupné znalosti pri plnení ich úloh. Novo vygenerované znalosti by mali byť znovu začlenené do systému.

### ***Meranie***

V tejto časti sa vyhodnocuje, ako dobre boli splnené zadané ciele.

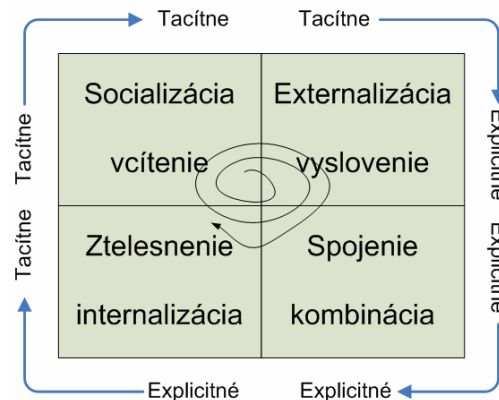
Týchto osem častí popisuje prvky činnosti znalostného manažmentu a malo by sa na ne nahliadať ako na celok a nie ako na navzájom izolované prvky.

## **2.1.3 Životný cyklus znalostí**

(Nonaka et. al., 1995) vytvorili na základe prípadových štúdií model niekoľkých spôsobov, ako organizácie vytvárajú znalosti.

Model SECI (Paralič, 2006) (socializácia, externalizácia, kombinácia, internalizácia) zahŕňa (Obr. 3):

- 2 formy znalostí – nevyjadrené a explicitné,
- dynamickú interakciu – prenos,
- tri úrovne sociálnej agregácie – individuálnu, skupinovú a kontextovú,
- 4 procesy tvorby znalostí – socializácia, externalizácia, kombinácia a internalizácia.



Obr. 3: Model SECI

Organizácie vytvárajú znalosti cez interakcie medzi explicitnými a nevyjadrenými znalosťami, pri ktorých sa znalosti rozširujú čo do kvantity aj kvality.

1. Socializácia je proces prenosu nových nevyjadrených znalostí cez zdieľanú skúsenosť, napr. spoločným trávením času, formálne a neformálne stretnutia aj mimo pracovného času alebo zdieľaním nevyjadrených znalostí ako pohľady na svet, mentálne modely a vzájomná dôvera.
2. Externalizácia je proces artikulárne nevyjadrených znalostí na znalosti explicitné. Znalosť tak kryštalizuje, čo umožňuje jej zdieľanie s ostatnými a tým sa stáva základom pre novú znalosť. Príkladom môže byť vytvorenie konceptu pri vývoji nového produktu alebo kontrola kvality, ktorá dovoľuje zamestnancom zlepšovať výrobné procesy.
3. Kombinácia je proces premeny explicitných znalostí do komplexnejších a systematickejších súborov explicitných znalostí. Explicitné znalosti sú zbierané z vnútorného alebo externého prostredia organizácie a následne kombinované, editované a spracovávané za účelom formovania nových znalostí. Nové znalosti sú následne šírené medzi členmi organizácie.
4. Internalizácia je proces stelesnenia explicitných znalostí do nevyjadrených. Cez proces internalizácie sú explicitné znalosti zdieľané v organizácii a sú jednotlivcami konvertované na nevyjadrené znalosti.

## 2.1.4 Clips

### *Všeobecné informácie*

Systém Clips je prostredie pre vývoj pravidlových, respektíve objektovo orientovaných expertných systémov. Používa ho viac ako 5000 používateľov z verejného, ale aj súkromného sektoru. Táto skupina zahŕňa všetky centrá národnej organizácie NASA, všetky úseky armády USA, množstvo federálnych inštitúcií, vládnych dodávateľov, univerzity a súkromné spoločnosti. Pod rozšírenosť tohto systému sa určite podpísal fakt, že je na nekomerčné účely použiteľný bezplatne.

Medzi kľúčové vlastnosti systému CLIPS patrí:

- *Reprezentácia poznatkov:* CLIPS predstavuje dobre použiteľný nástroj pre manipuláciu s poznatkami rôznych typov, pričom podporuje tri rozdielne programátorské paradigmy: pravidlovú, objektovú a procedurálnu paradigmu. Pravidlové programovanie umožňuje, aby boli poznatky reprezentované ako heuristiky, špecifikujúce množinu akcií, ktoré treba vykonať v danej situácii. Objektové programovanie umožňuje modelovať zložité systémy v tvare modulárnych komponentov (tieto možno znova použiť pri modelovaní iných systémov alebo pri vytváraní nových komponentov). Možnosti procedurálneho programovania, ktoré CLIPS poskytuje sa podobajú tým, ktoré nachádzame v programovacích jazykoch ako sú C, Pascal, Ada alebo LISP.
- *Prenositel'nosť (alebo portabilita):* z dôvodu jednoduchej prenositeľnosti medzi operačnými systémami, ale aj kvôli dosiahnutiu maximálneho výkonu je CLIPS napísaný v jazyku C. CLIPS beží pod OS: Windows, Unix a MacOS. Zároveň existujú preportované verzie do jazyka Java a platformy .NET.
- *Možnosti integrácie a rozširovania:* systém CLIPS možno v rámci procedurálneho kódu vyvolaného ako procedúra integrovať do väčšiny používateľských programov v jazykoch ako sú C, Ada, .Net (C#, VB). CLIPS môže byť používateľom ľahko rozšírený použitím niektorého z dobre definovaných protokolov.
- *Vývojové prostredie:* CLIPS ponúka jednoduché prostredie, v ktorom možno zadávať príkazy terminálu CLIPS. Prostredie zároveň umožňuje akúsi formu textového ladenia programu. Prostredie však neobsahuje editor bázy znalostí.
- *Plná dokumentácia:* V rámci systému CLIPS je poskytnutá rozsiahla dokumentácia zahŕňajúca referenčné a používateľské príručky.

### ***Reprezentácia znalostí***

Báza znalostí sa dá rozdeliť do modulov. Modul vytvára samostatnú časť nad ktorou možno usudzovať nezávisle od iných častí systému. Fakty obsiahnuté v jednom module sa dajú zverejniť a tak dosiahnuť ich čitateľnosť v iných moduloch.

### ***Deklaratívne znalosti***

Sú realizované prostredníctvom:

- Premenných faktov
- Trvácnych faktov
- Lokálne a globálne premenné
- Pravidlá

Premennivé fakty sa často modifikujú a popisujú aktuálnu situáciu. Vytvárajú tak bázu faktov. Trvácne fakty sa používajú najmä na reprezentáciu bázy znalostí, možno ich však použiť aj na reprezentáciu bázy faktov.

To, aký tvar budú mať trvácne fakty je definované pomocou šablón. Na nasledujúcom príklade je príkaz vytvorenia šablóny, ktorá popisuje rentabilitu konkrétnej banky.

```
(deftemplate rentabilita "popisuje rentabilitu banky"
  (slot meno-banky (type symbol))
  (slot ROA (type NUMBER) (range 0 100))
  (slot ROE (type NUMBER) (range 0 100))
)
```

Telo šablóny obsahuje tri položky: meno banky, parameter ROA a parameter ROE.

Lokálne premenné je možno vytvárať v pravidlách a majú životnosť len počas vykonávania pravidla. Globálne premenné sú viditeľné v rámci celého modulu.

Pravidlá vytvárajú heuristickú časť usudzovania v expertnom systéme. Pravidlo pozostáva z podmienkovej časti a dôsledkovej časti.

Príklad jednoduchého pravidla:

```
(defrule je-to-kocka
  (pocet_stien 6) (tvar_steny stvorec)
  =>
  (assert (utvar kocka))
)
```

Pravidlo sa vykoná v prípade, že existuje fakt pocet\_stien a je nastavený na hodnotu 6 a zároveň existuje fakt tvar\_steny a je nastavený na hodnotu štvorec. V takom prípade sa vytvorí nový fakt s názvom utvar a hodnotou kocka.

Ako je vidieť na príklade, podmienková a dôsledková časť pravidla je oddelená znakom =>.

### ***Procedurálne znalosti***

Sú realizované prostredníctvom:

- Funkcií
- Generických funkcií
- Message-handlers
- Externé funkcie

Systém CLIPS má množinu vstavaných funkcií z rôznych oblastí (matematické, vstupno-výstupné a iné). Mimo využívania týchto funkcií si môže používateľ nadefinovať vlastnú funkciu. Jej telo bude vykonávané procedurálne.

Príklad jednoduchej funkcie:

```
(deffunction sign (?num)
  (if (> ?num 0) then
    (return 1)))
```

```
(if (< ?num 0) then
  (return -1))
0)
```

Funkcia sign vracia 1 ak je na vstupe funkcie kladné číslo, -1 ak je na vstupe záporné číslo a nula ak je na vstupe 0.

Použitie funkcie vyzerá nasledovne:

```
CLIPS> (sign 5)
1
CLIPS> (sign -10)
-1
CLIPS> (sign 0)
0
CLIPS>
```

Generické funkcie ponúkajú možnosti preťažiť meno funkcie viacerými implementáciami. Podľa druhu argumentov funkcie sa vykoná príslušná implementácia.

Message-handlers sú metódy objektov, ktoré sú volané prostredníctvom príkazu sent. Správajú sa rovnako ako funkcie.

CLIPS dokáže byť spúšťaný z externého prostredia a zároveň dokáže clips spúšťať externé funkcie prostredím, v ktorých kontexte je spúšťaný. Túto funkcionality je možné využiť pri realizovaní procedurálnych znalostí.

### ***Objektové znalosti***

Systém CLIPS má možnosť reprezentovať vedomosti pomocou tried. Objektová paradigma je realizovaná prostredníctvom jazyka COOL. Zahŕňa abstrakciu, zapúzdrenosť, dedičnosť, polymorfizmus.

Príklad definície jednoduchej triedy:

```
(defclass A (is-a USER)
  (slot x)
  (slot y (type SYMBOL))
  (slot z (range 3 10)))
```

Trieda A dedí od triedy USER a definuje vlastné členské premenné x, y a z. Premenná y je typu SYMBOL, premenná z je typu Integer a jej hodnota musí byť v rozmedzí od troch do desiatich. Premenná x je neurčeného typu, čo znamená že jej typ bude určený až po vytvorení prvej inštancie triedy a inicializovaní premennej hodnotou.

### ***Inferenčný mechanizmus***

Inferencia je vykonávaná dopredným zreťazením. Každý modul má svoju agendu. Do nej sa ako do zásobníka vložia pravidlá, ktoré pri danom zložení faktov ešte neboli použité a zároveň je splnená ich podmienková časť. Následne sa začne vykonávaním prvého z pravidiel. Ak sa zmení hodnota nejakého faktu, agenda sa naplní znova a všetky fakty sú nastavené ako nepoužité.

Hlavná myšlienka algoritmu usudzovania je vyvodzovanie nových faktov z už známych faktov. Po vyvedení nového faktu sa opäť posúdi situácia, či sa nedajú aplikovať nejaké pravidlá na novú množinu faktov.

### ***Zhodnotenie***

Kladné vlastnosti:

- rýchlosť inferencie,
- podpora viacerých paradigiem programovania,
- možnosť spúšťania externých funkcií.

Záporné vlastnosti:

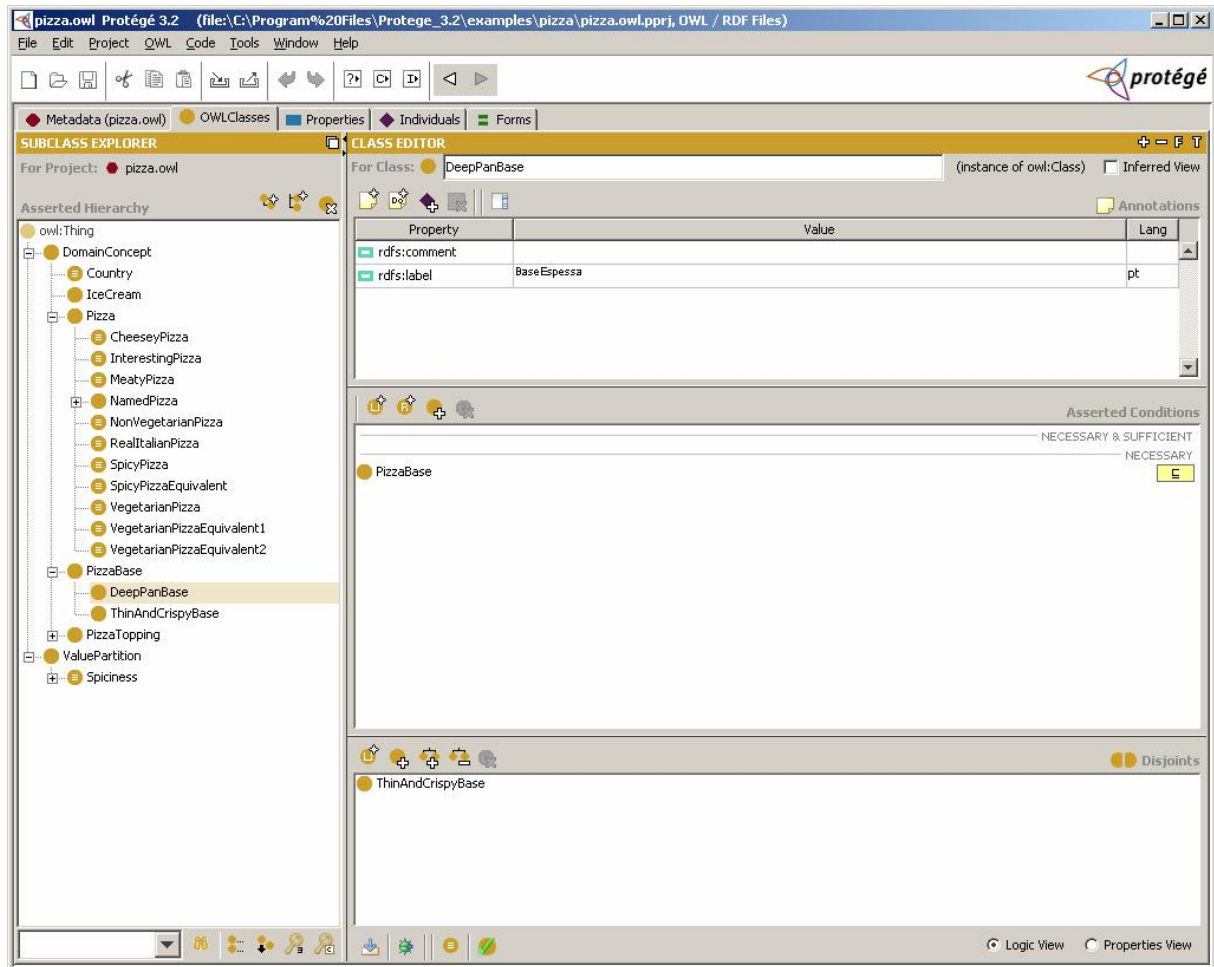
- nemožnosť spolupracovať s externou bázou znalostí,
- neobsahuje editor bázy znalostí,
- len nízko úrovňový nástroj, neobsahuje nástroje na podporu vytvárania ontológií.

### ***Na aké úlohy sa hodí***

CLIPS sa hodí na úlohy menšieho rozsahu, kde je potrebné usudzovanie nad stredne veľkou bázou znalostí. Hodí sa na implementáciu znalostného systému v rámci iného systému, ktorý zabezpečuje podporné prostredie.

## **2.1.5 Protégé**

Protégé (Protégé, <http://protege.stanford.edu/>) je voľne dostupná platforma s otvoreným zdrojovým kódom, ktorá obsahuje sadu nástrojov na konštrukciu doménových modelov a znalostných aplikácií s ontológiami. Vo svojej podstate Protégé implementuje bohatú sadu objektov pre modelovanie znalostí a funkcií, ktoré podporujú vytváranie, vizualizáciu a manipuláciu s ontológiami reprezentovanými v rôznych formátoch. Protégé je škálovateľný a je ho možné jednoducho upraviť, aby podporoval ďalšie znalostné modely a vstupné údaje. Protégé je rozšíriteľný pomocou zásuvných modulov a aplikačného programového rozhrania (API) založeného na JAVA platforme. To umožňuje vytvárať ďalšie znalostné nástroje a aplikácie. Príklad používateľského rozhrania je na obrázku (Obr. 4).

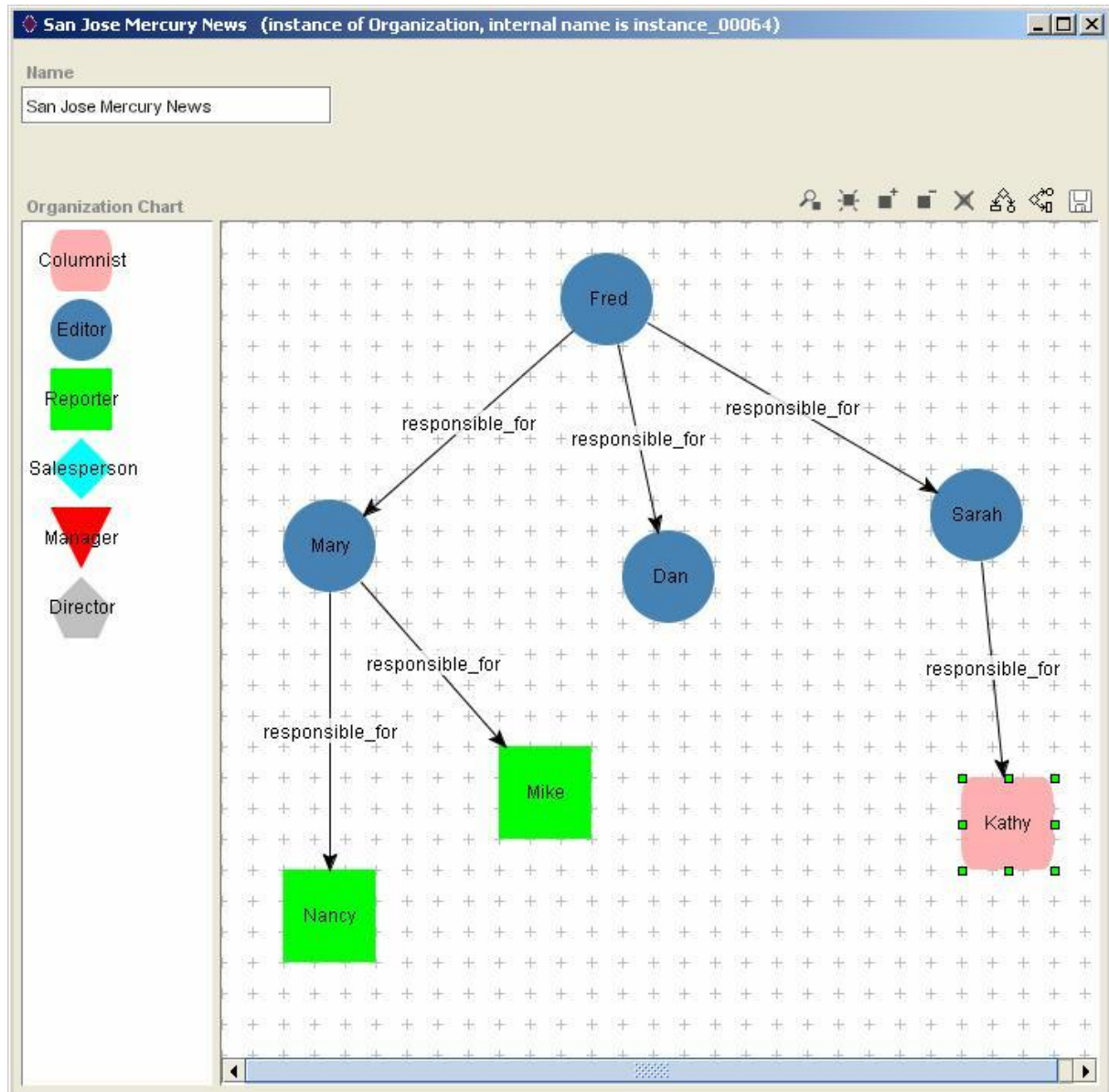


Obr. 4: Príklad používateľského rozhrania programu Protégé 3.2.

Platforma Protégé dva hlavné spôsoby modelovania ontológií:

1. *Protégé editor rámcov* dáva používateľom možnosť vytvoriť a naplniť ontológie, ktoré sú založené na rámcoch v súlade s OKBC (Open Knowledge Base Connectivity protocol). V tomto modeli, ontológia pozostáva zo sady tried (konceptov) organizovaných do hierarchie. Je to množina slotov (parametrov) priradených k triedam, ktoré opisujú ich vlastnosti (premenné) a vzťahy a množina inštancií týchto tried, ktoré uchovávajú špecifické hodnoty pre ich premenné. Príklad používateľského rozhrania je na obrázku (Obr. 5).



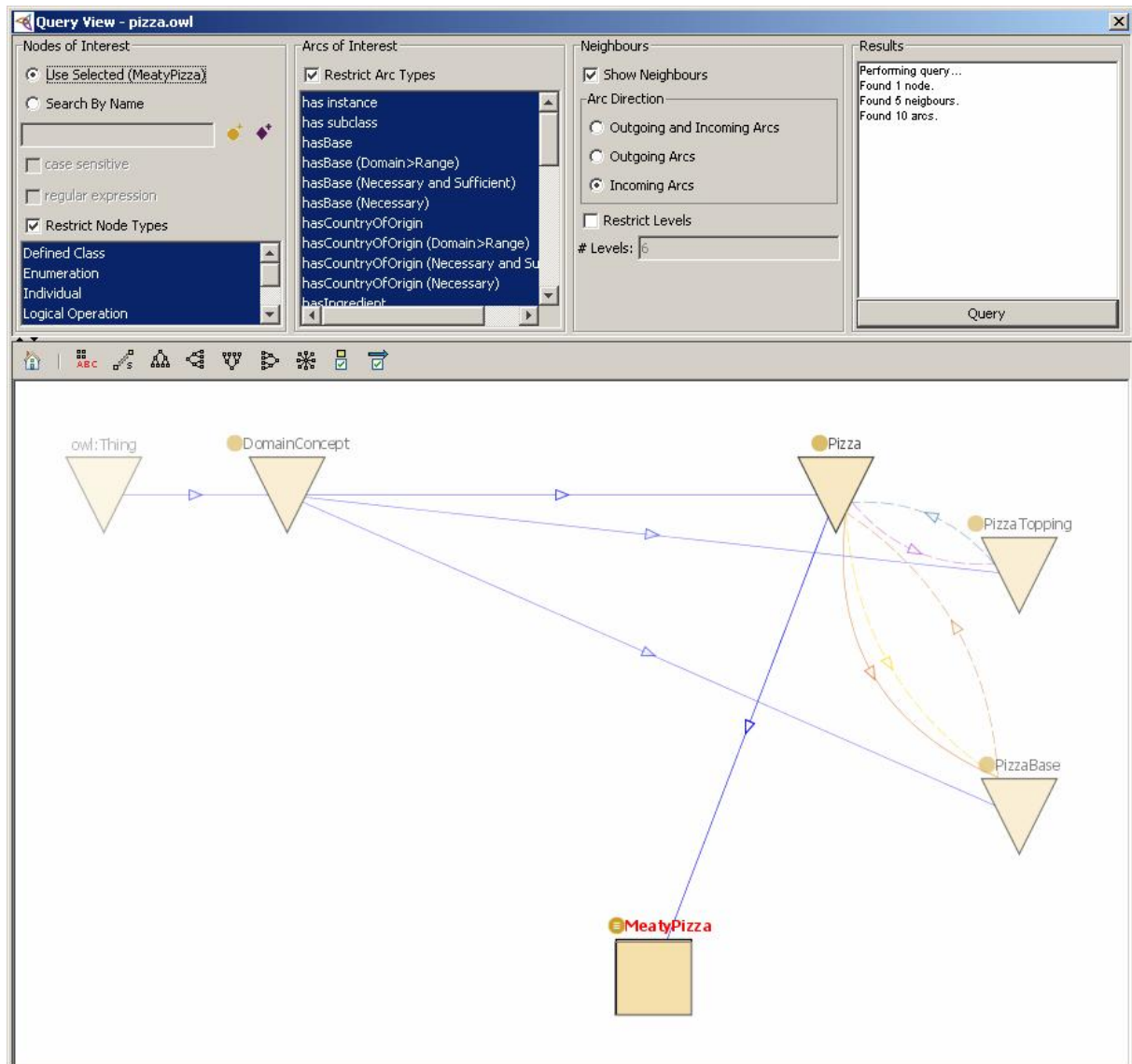


Obr. 5: Príklad používateľského rozhrania programu Protégé 3.2 – editor rámcov

Protégé editor rámcov okrem iného obsahuje:

- Širokú sadu prvkov užívateľského rozhrania, ktoré sa môžu podľa potreby prispôbiť na modelovanie znalostí a zadávanie dát vo formulároch.
  - Architektúru zásuvných modulov, ktorá umožňuje rozšíriť produkt o vlastné navrhnuté prvky ako napríklad grafické komponenty (grafy, tabuľky...), média (zvuk, obrázky a video), rôzne formáty na ukladanie (RDF, XML, HTML, databázové rozhrania...) a doplnkové podporné nástroje (napr. manažment ontológií, vizualizácia ontológií, inferencia a logické myslenie ...)
  - Aplikačné programové rozhranie (API) založené na platforme JAVA, ktoré umožňuje zásuvným modulom a iným aplikáciám pristupovať, používať a zobrazovať ontológie vytvorené Protégé editorom rámcov.
2. *Protégé OWL editor*- dáva používateľom možnosť vytvoriť ontológie pre sémantický web, obzvlášť v jazyku OWL (Web Ontology Language) (Bechhofer,

<http://www.w3.org/TR/2004/REC-owl-ref-20040210/>). OWL ontológia môže zahŕňať opis tried, premenných a ich inštancií. V takejto ontológii, OWL formálna sémantika špecifikuje ako odvodiť jej logické súvislosti t.j. fakty, ktoré nie sú doslova prítomné v ontológii ale sú vyvedené zo sémantiky. Tieto vyvodenia môžu vychádzať z jedného dokumentu alebo viacerých distribuovaných dokumentov, ktoré boli skombinované použitím definovaného OWL mechanizmu (Smith, <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>) Príklad používateľského rozhrania je na obrázku (Obr. 6).



Obr. 6: Príklad rozhrania programu Protégé 3.2 – vizualizácia vzťahov medzi triedami ontológie

Protégé OWL editor okrem iného umožňuje:

- Ukladať a načítavať OWL a RDF ontológie.
- Upravovať a zobrazovať triedy, premenné a SWRL(Semantic Web Rule Language) pravidlá
- Definovať logické vlastnosti tried ako OWL výrazy.
- Vykonať logické operácie ako napríklad opisné logické triedenie.

Flexibilná architektúra Protégé-OWL umožňuje jednoducho konfigurovať a rozširovať tento nástroj o ďalšie funkcie. Protégé-OWL je úzko integrovaný s Jena (Java API pre RDF a OWL) a má otvorený zdrojový kód JAVA aplikačného programového rozhrania pre vývoj vlastných prispôbených komponentov užívateľského rozhrania alebo nezávislé Sémantické Web Servisy.

### ***Protégé a Clips***

Protégé používa textový formát CLIPSu (viď kapitola 2.1.4) ako jeho predvolený formát pre ukladanie a načítavanie tried a ich inštancií v ontológii. Predvolené súbory Protégú (.pont a .pint) sú vlastne súbory CLIPSu, ktoré obsahujú zvlášť triedy a ich inštancie. Protégé nepoužíva CLIPS ako taký. Používa len jeho formát pre ukladanie. Je teda možné načítať súbory Protégú priamo do CLIPSu a naopak. Sú však dve veci, ktoré je potrebné mať na zreteli:

1. Kompatibilné rozšírenia Protégú oproti CLIPS formátu
2. Nekompatibilné rozšírenia Protégú oproti CLIPS formátu

### ***Kompatibilné rozšírenia Protégú oproti CLIPS formátu***

Ako sa Protégé vyvíjal, ukázalo sa výhodným predstaviť a používať koncepty (triedy), ktoré nemajú priamo podporu v CLIPSe. Tieto však nie sú nekonzistentné s CLIPSom a jeho formátom. Príkladom sú Protégom zavedené dovolené triedy (allowed-classes) pre sloty (parametre) konceptov (tried) typu inštancia (Instance). Táto informácia je uložená súbore CLIPSu ako rozšírenie CLIPSu (CLIPS extensions), dodržiavajúc konvenciu, že každý takýto riadok začína znakmi „;+“. Keďže znak „;“ označuje v CLIPS formáte komentár, sú tieto riadky CLIPSom ignorované. Protégé tieto riadky však neignoruje a pri načítavaní takého to súboru ich rozpozná a používa.

Rozšírenia Protégú, ktoré spadajú do tejto kategórie zvyčajne nespôsobujú žiadne problémy pre používateľov CLIPSu. Kompatibilnými rozšíreniami sú:

- *dovolené triedy (allowed classes)* - explicitné obmedzenie pre sloty typu Inštancia,
- obmedzenie pre dokumentáciu slotu,
- vkladanie ďalších projektov.

### ***Nekompatibilné rozšírenia Protégú oproti CLIPS formátu***

Protégé sa však v niektorých veciach odklonil od objektovo orientovaného modelu CLIPSu, takže niektoré nové podporované vlastnosti Protégú nemôžu byť priamo integrované v CLIPSe. Najlepším príkladom sú metatriedy (metaclasses) v Protégi – triedy, ktorých inštancie sú samy triedy. Ak sa teda rozhodneme použiť v Protégi metatriedy, výsledné triedy a ich inštancie nemôžu byť použiteľné v CLIPSe žiadnym spôsobom. Súbory zostávajú kompatibilné s CLIPS formátom vďaka komentárom, avšak interpretácia obsahu súboru v CLIPSe je odlišná ako v Protégi (v modeli v CLIPSe chýbajú metatriedy a ich inštancie).

Ďalšou nekompatibilitou je rozsah platnosti mena slotu. Predpokladajme dve úplne nezávislé triedy Jablko a Hruška. Nech obidve majú slot A. V CLIPSe tieto sloty nie sú tým istým slotom (sú nezávislé) a ich obmedzenia sú nezávislé. V Protégi existuje iba jeden slot A. Tento slot je pripojený ku obidvom triedam Jablko aj Hruška. V obidvoch triedach však môžu takisto byť pre tento slot definované odlišné obmedzenia vďaka preťažovaniu. Okrem toho, v Protégé mená slotov a tried sú v tom istom priestore mien, preto musia byť jedinečné. CLIPS dovoľuje triedam a slotom mať to isté meno.

Ďalej Protégé dovoľuje dávať rámcom také mená, ktoré sú neplatné v CLIPSe. Aby CLIPS dokázal správne rozpoznať a načítať obsah súboru, je potrebné voliť iba také mená pre rámce, ktoré sú kompatibilné s CLIPSom.

Súhrn kompatibilných a nekompatibilných vlastností Protégé a CLIPSu je zobrazený v tabuľke (Tab. 1).

CLIPS – vlastnosti	Protégé	Riešenie
Moduly	Nepodporované	
Mená tried a slotov	Pozri vyššie	Dodržiavať naraz obmedzenia Protégé aj CLIPSu (žiadne prázdne mená, odlišné mená pre triedy a sloty)
Viacnásobná dedičnosť	Áno	
Abstraktnosť/Konkrétnosť	Áno	
Reaktívnosť/Nereaktívnosť	Nepodporované	Použiť post-processing na pridanie reaktívnych špecifikácií tým triedam, ktoré to potrebujú
Jednoduché pole/zložené pole	Áno	
Predvolená hodnota obmedzenia	Áno, ale len statické nastavenie (nie dynamické)	
Obmedzenie na ukladanie	Lokálne	
Prístupové obmedzenie	Čítanie aj zápis	
Dedenie obmedzenia	Dedí	
Obmedzenie zdroja	Vyhradené	
Reaktívne obmedzenie súhlasu vzorky	Nepodporované	
Obmedzenie viditeľnosti	Verejné (Public)	
Obmedzenie pre vytvorenie a prístup	Čítanie aj zápis	
Obsluha správ	Nepodporované	Zapisovanie správ a ich „handlerov“ do osobitných súborov

Tab. 1: Súhrn kompatibilných a nekompatibilných vlastností Protégé a CLIPSu

### Zhodnotenie možnosti využitia Protégé

Protégé je voľne šíriteľný znalostný systém pre prácu s ontológiami, a je implementovaný na platforme Java. Na tejto platforme podporuje aj programové rozhranie pre prácu s ontológiami a dovoľuje priamo vstupovať cez toto rozhranie do programu a manipulovať s ontológiami. Nenašli sme však podporu tohto rozhrania pre platformu .NET, na ktorej budeme náš systém vyvíjať. Keďže nevieme ku Protégéu programovo pristúpiť, nie je priamo použiteľný v našom projekte. Na druhej

strane je však úzko previazaný s CLIPSom (kapitola 2.1.4) vďaka spoločnému formátu, v ktorom Protégé ukladá a načítava projekty. Je preto možné ho použiť pri počítačnom návrhu ontológií, vytváraní tried a ich inštancií a následnom použití v programe CLIPS. Využitie Protégé je v tomto smere výhodné, pretože na rozdiel o CLIPSu priamo vizualizuje navrhnuté triedy, inštalácie, ich parametre a vzťahy, čo umožňuje návrhárovi mať lepší prehľad a orientáciu pri ich návrhu.

## 2.1.6 Iné znalostné systémy

Kvôli povahe systému, ktorého súčasťou bude nami vytvorený modul, vybraný znalostný systém by mal mať implementované rozhrania resp. knižnice pre platformu .NET Framework. Teda by mal bežať pod operačným systémom Windows (98, 2000, XP, Vista). Mal by podporovať tvorbu ontológií. Veľmi dôležitým kritériom je jeho používanosť v praxi, ktorá okrem iného zabezpečuje, že systém je funkčný a odladený. Z dôvodu, že systém budeme možno nútený mierne upravovať, prihliadali sme aj na jeho zdokumentovanosť. V prvej iterácii vyhľadania vhodného systému sme dôraz nekládli na reprezentáciu znalostí, u ktorej pri v praxi používaných systémoch rátame s jej bezproblémovosťou a jej naštudovaniu sa budeme venovať neskôr. Okrem bližšie opísaných znalostných systémov (Clips, Protégé) sme analyzovali sme aj iné, avšak tie nepostačovali našim základným kritériam. Uvádžame v krátkosti ich zoznam, a dôvod ich zavrnutia.

- n FreeShell – Výsledok tímového projektu. Hlavným dôvodom, prečo neuspel bola nedostatočne podrobná dokumentácia.  
(<http://www.cs.siue.edu/SeniorProjects/2003/fall/FreeShell/968x726/main.html>)
- n Summer – už sa nevyvíja, posledná verzia DOS. Nesplnil kritérium podpory na platforme .NET.  
(<http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/expert/systems/es/0.html>)
- n DEX – bez ontológií, verzia DOS v Turbo Pascale. Nesplnil kritérium podpory na platforme .NET.  
(<http://www-ai.ijs.si/MarkoBohanec/dex.html>)
- n FEL-Expert - bez ontológií, v praxi nepoužívaný  
(<http://krizik.felk.cvut.cz/felex/>)
- n JAM – zdarma pre nekomerčné účely, Java. Absencia podpory .NET Frameworku.  
([http://www.marcush.net/IRS/irs\\_downloads.html](http://www.marcush.net/IRS/irs_downloads.html))
- n MIKE - už sa nevyvíja, posledná verzia DOS  
([http://kmi.open.ac.uk/people/marc/mike\\_text.html](http://kmi.open.ac.uk/people/marc/mike_text.html))
- n QSIM – Lisp resp. C++, verzia pre Windows v plienkach  
(<http://www.cs.utexas.edu/users/qr/OR-software.html>)
- n Jena 2 – Java. Absencia podpory .NET Frameworku.  
(<http://jena.sourceforge.net/>)

- n JTP – veľmi jednoduchý, Java. Absencia podpory .NET Frameworku.  
(<http://www.ksl.stanford.edu/software/JTP/>)
- n JESS- Systém JESS (<http://herzberg.ca.sandia.gov/jess/>) je v podstate Java nadstavbou nad systémom CLIPS. Vzhľadom na náš projekt je Java skôr nevýhodou. Vie pristupovať k Java objektom a umožňuje použitie XML na defonovanie pravidiel. Založený je na deklaratívnom programovaní. Obsahuje aj prvky procedurálneho programovania.

## 2.2 Indexovanie a vyhľadávanie

Najjednoduchšou metódou pre vyhľadávanie slov a fráz v texte je sekvenčné prehľadávanie (MANU KONCHADY 2006). Táto metóda je však efektívna iba ak prehľadávaný dokument má veľkosť maximálne niekoľko megabytov. Pri väčších dokumentoch alebo väčšom množstve dokumentov je táto metóda nepraktická a pomalá.

Na zrýchlenie vyhľadávania slov v dokumente vytvoríme z dokumentu index. Index odzrkadľuje obsah dokumentu a má taký tvar, ktorý sa dá ľahko prehľadávať.

Ak je báza dokumentov statická, stačí vytvoriť index raz. Ak sa báza dokumentov v čase upravuje, je potrebné vytvárať index periodicky.

### 2.2.1 Inverzný index

Najbežnejšia metóda indexovanie je tzv. invertované indexovanie. Je to metóda pri ktorej sa postupne vykonávajú tieto kroky:

- dokument sa rozdelí na slová,
- vytvorí sa zoznam týchto slov,
- ku každému slovu sa priradí zoznam pozícií jeho výskytov v dokumente.

Pozície slov v takomto indexe môžu byť buď znakové alebo slovné.

Slovné pozície sú výhodnejšie pri vyhľadávaní frázy s približnou vzdialenosťou slov. Aby sme vedeli rozoznať v ktorom dokumente sa hľadané slovo nachádza na danej pozícii, musí mať pozícia slova priradený aj id dokumentu. Pokiaľ index neobsahuje slovnú pozíciu, trvá vyhľadávanie frázy oveľa dlhšie. Znakové pozície sú užitočné pri nachádzaní textu v dokumente. Inak je potrebné prehľadať celý dokument a nájsť hľadané slová. Je potrebné zvážiť, či potrebujeme obidva druhy pozícií a či sme ochotný zväčšiť tak výsledný indexový súbor.

Tretím parametrom môže byť číslo bloku. Dokumenty môžu byť delené na bloky s pevnou dĺžkou (10,64 alebo 256 tisíc znakov/slov) Použitím takýchto blokov sa veľkosť indexu znižuje.

### 2.2.2 Hľadanie v indexoch

Ako sme už spomenuli, cieľom indexovania je zvýšiť rýchlosť vyhľadávania v dokumente. Vyhľadávanie sa v zásade delí do troch základných krokov:

1. Rozdelenie hľadanej požiadavky na slová a vyhľadanie každého slova v indexe.
2. Získanie zoznamu dokumentov, v ktorých sa slovo nachádza.
3. Zkombinovanie zoznamov dokumentov podľa operátorov použitých v požiadavke.

V prvom kroku sa z požiadavky odstráni „stop slová“, to jest z, a, do, od, a podobne. Je možné, že ak sa požiadavka skladá iba z takýchto slov, po prvom kroku nemáme čo hľadať.

Po druhom kroku máme zoznamy dokumentov prislúchajúce jednotlivým slovám. Podľa operátorov v požiadavke tieto zoznamy skombinujeme a dostaneme výsledný zoznam hľadaných dokumentov.

### ***Príklad***

Hľadáme slová: **Porucha, Motor**

Zoznamy dokumentov po druhom kroku:

Porucha	21,68,125,348
Motor	25,68,97,125,336,412

Výsledný zoznam dokumentov by potom bol:

Pri použití operátora AND: 68,125

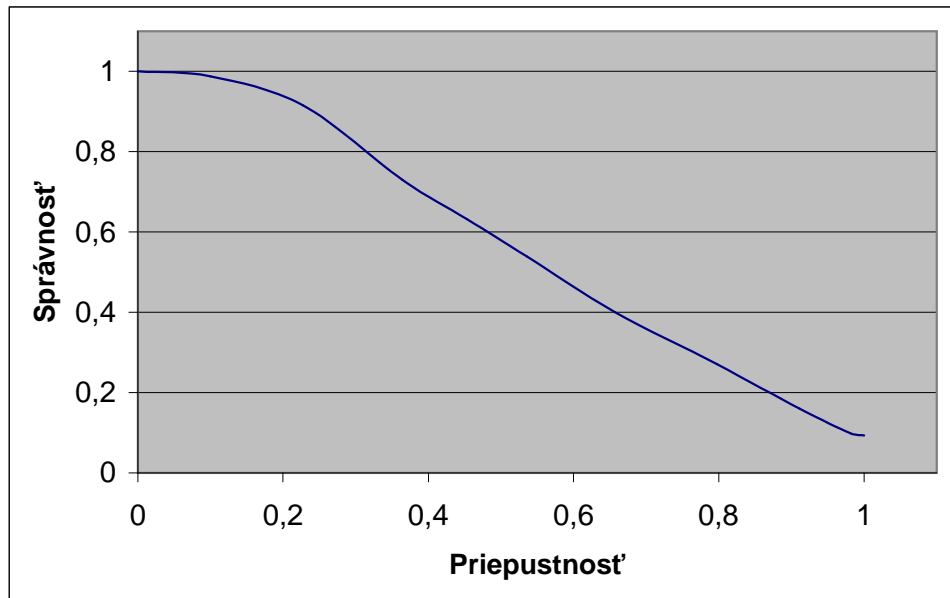
Pri použití operátora OR: 21,25,68,125,336,348,412

## **2.2.3 Správnosť a priepustnosť**

Toto sú dve základné metriky pre vyhľadávanie. Bližšie si ich popíšeme teraz.

**Správnosť** je pomer nájdených relevantných dokumentov ku všetkým nájdeným dokumentom.

**Priepustnosť** je pomer nájdených relevantných dokumentov ku všetkým relevantným dokumentom v báze dokumentov.



Obr. 7: Všeobecný graf závislosti správnosti od priepustnosti

**Príklad:**

Máme daný stav znázornený v tabuľke:

	Relevantné	Irelevantné	Spolu
Nájdene	40	80	120
Nenájdene	10	870	880
Spolu	50	950	1000

Správnosť je  $40/120 = 0,33$

Priepustnosť je  $40/50 = 0,8$

Ak by sme chceli zvýšiť správnosť z 0,33 na 0,5 upravili by sme podmienky vyhľadávania tak aby sa znížil počet nájdených dokumentov zo 120 na 60. Tým by sa znížil počet relevantných nájdených dokumentov zo 40 na 30.

Dostávame:

Správnosť  $30/60 = 0,5$

Priepustnosť  $30/50 = 0,6$

Pri tvorení podmienok vyhľadávania sa musíme rozhodnúť a určiť si pomer medzi správnosťou a priepustnosťou tak, aby sme sa nám nestalo že nenájdeme veľa relevantných dokumentov alebo nájdeme veľa irelevantných.

### 2.2.4 DotLucene – analýza vyhľadávacieho systému



DotLucene je fultextový indexovací a vyhľadávací systém (DAN LETECKY, 2006). Bol vyvinutý preportovaním projektu v Jave s názvom Jakarta Lucene pod platformu C# .Net. Je to open source projekt a je šírený pod licenciou (Apache Software License 2.0).

### *Hlavné rysy DotLucene*

- Použiteľnosť v ASP.NET, Win Forms alebo konzolových aplikáciách.
- Veľmi vysoký výkon.
- Klasifikované výsledky hľadania.
- Možnosť zvýrazniť hľadaný výraz vo výsledkoch hľadania.
- Prehľadávanie štruktúrovaných aj neštruktúrovaných dát.
- Vyhľadávanie v metadátach (podľa dátumu ,hľadanie vo voliteľných poliach indexu a iných).
- Veľkosť indexu je približne 30% z veľkosti indexovaného textu.
- Vie uchovávať indexované celé dokumenty.
- Čistý a jednoducho manažovateľný kód pod platformou .NET v jednej knižnici (244 KB).
- Veľmi priateľská licencia (Apache Software License 2.0).
- Lokalizovateľné (Angličtina, Čeština, Francúzština a iné).
- Rozšíriteľnosť (voľný zdrojový kód).
- Možnosť dopĺňať index (Pri pridávaní nových dokumentov nie je nutné vytvoriť celý index nanovo).
- Možnosť indexovať rôzne typy dokumentov. (Zaindexovať sa dá akýkoľvek dokument, ktorý sa dá previesť na text).
- Možnosť indexovať dokumenty z rôznych druhov zdrojov (web, databáza, lokálne uložené dokumenty).

### *Hlavné funkcie DotLucene*

Nad knižnicou DotLucene boli vytvorené jednoduché nástroje na sprístupnenie poskytovaných funkcií (DAN LETECKY, 2006). V ďalšej časti sú opísané niektoré z nich.

### *Indexovanie*

Na báze DotLucene bol vytvorený jednoduchý indexovací nástroj. Je to jednoduchá konzolová aplikácia konfigurovateľná pomocou XML súboru. V tomto konfiguračnom súbore môžeme nastaviť tieto parametre:

- **Meno a lokácia výsledného indexu.**

```
<index name="index_A" indexFolderUrl="C:\Documents and Settings\LuceneIndexer\TestIndexA">
```

- **Indexované atribúty**

```
<field name="Nazov" isStored="true" isIndexed="true" isTokenised="false" />
```

Jednotlivé parametre majú nasledujúci význam:

- **isStored** - určuje, či bude daný parameter ukladaný do indexu.(Kedykoľvek sa dá k nemu prístupíť. Je to vhodné pre kratšie texty ako autor alebo názov.)
- **isIndexed** - určuje, či sa bude dať podľa tohto atribútu vyhľadávať.
- **isTokenized** - nastavením toho parametra sa indexovaná časť pred indexovaním rozdelí na slová.

- **Zdroj dát** (Ako zdroj sa v tomto jednoduchom programe využíva SQL Server. XML hodnota pre tento parameter je napríklad takáto.)

```
<sqlClient connectionString="Data Source=(local);Initial
Catalog=Junk;Trusted_Connection=True">SELECT * FROM
Content</sqlClient>
```

Implementácia metódy pre získanie dát z takto uvedeného zdroja je nasledovná:

```
private DataTable GetData
{
    get
    {
        //Get Data using SQL
        string selectCommandText = XmlNode.SelectSingleNode(
"selectCommandText").InnerText;
        string connectionString = XmlNode.SelectSingleNode(
"selectCommandText/@connectionString").Value;
        SqlDataAdapter da = new SqlDataAdapter(selectCommandText,
new SqlConnection(connectionString));
        DataTable dt = new DataTable();
        da.Fill(dt);
        return dt;
    }
}
```

Pre naše potreby nebude problematické vytvoriť metódy pre získavanie zdroja z lokálneho súborového systému alebo internetovej adresy.

Výsledný konfiguračný XML súbor potom vypadá napríklad takto:

```
<?xml version="1.0" encoding="utf-8"?>
<indexConfiguration>
  <index name="TaskA" indexFolderUrl="C:\Documents and
Settings\abi\Desktop\LuceneIndexer\LuceneIndexer\TestIndex">
    <sqlClient connectionString="Data Source=(local);Initial
Catalog=Junk;Trusted_Connection=True">SELECT * FROM
Content</sqlClient>
    <fields>
      <field name="ID" isStored="true" isIndexed="true"
isTokenised="false" />
      <field name="Title" isStored="true" isIndexed="true"
isTokenised="false" />
      <field name="ContentText" isStored="true" isIndexed="true"
isTokenised="true" />
    </fields>
  </index>
</indexConfiguration>
```

```

</fields>
</index>
</indexConfiguration>

```

### Vyhľadávacia syntax (DOTLUCENE, 2006)

Požiadavka	Príklad	Poznámka
Jedno slovo	dokument	Prehľadá dokument, či obsahuje slovo „dokument“ v defaultnom poli
Fráza	"dôležitý dokument"	Prehľadá dokument, či obsahuje frázu „dokument“ v defaultnom poli
V poliach	názov: dokument	Prehľadá dokument, či obsahuje slovo „dokument“ v poli „názov“
Náhradné znaky	dokument	Hľadanie s jedným náhradným znakom. Vyhľadá sa "dokument" and "dokiment" ale nie "dokooment".
	dokument*	Hľadanie s viacerými náhradnými znakmi. Vyhľadá sa "dokument" aj "dokumentacia".
Neurčité hľadanie	dokument~	Hľadanie na základe podobného hláskovania
	dokument~0.9	Hľadanie na základe podobného hláskovania. 0.9 je požadovaná podobnosť (prednastavené: 0.5)
Hľadanie so vzdialenosťou	"dôležitý dokument"~5	Hľadané slová vo fráze musia byť do určitej maximálnej vzdialenosti od seba. V príklade je to 5 slov
Rozsahové hľadanie	autor: {Einstein TO Newton }	Hľadá sa v poli autor od mena Einstein po meno Newton (abecedne zoradené)
	datum: {20050101 TO 20050201 }	Hľadá sa v poli datum medzi 20050101 až 20050201
Relevancia	dôležitý ^4 dokument	Určí sa faktor dôležitosti pre dané slovo vo fráze. Prednastavené je 1
	"dôležitý dokument"^4 "vyhľadávací systém"	Určí sa faktor dôležitosti pre frázu.
OR operátor	dôležitý dokument	"OR" je prednastavený operátor
	dôležitý OR dokument	Pole musí obsahovať „dôležitý“ alebo „dokument“
AND operátor	dôležitý AND dokument	Pole musí obsahovať slovo „dôležitý“ aj „dokument“
+ operátor	dôležitý +dokument	Pole musí obsahovať „dôležitý“ a môže obsahovať „dokument“
NOT/-operátor	-dôležitý dokument	Pole musí obsahovať „dokument“ ale nie „dôležitý“
Zoskupovanie	(dôležitý OR pekný) AND dokument	Na zoskupovanie sa používajú zátvorky
	Autor:(Einstein OR Newton)	Zátvorky sa dajú použiť aj pri definovaných poliach

### Obmedzenia

Query	Examples	Notes
-------	----------	-------

Náhradné znaky na začiatku slova	?okument, *okument	Vznikne Lucene.Net.QueryParsers.ParseException.
Stop slová	a, the, and	Tieto slová nie su indexované
Špeciálne znaky: + - &&    ! ( ) { } [ ] ^ " ~ * ? : \	\+, \:	Je nutné použiť spätné lomítko

### *Spracovanie iných ako textových formátov*

DotLucene vie priamo spracovávať iba textové formáty a ak chceme spracovávať iné typy súborov, musíme ich previesť do textového formátu (DOTLUCENE, 2006). Sú dostupné viaceré filtre ktoré dokážu previesť iné ako textové formáty na textové.

- Dokumenty MS Office (DOC, PPT, XLS, XML ...)
  - iFilter [microsoft.com]
- Dokumenty PDF
  - Adobe PDF IFilter
  - iTextSharp
  - PDFBox - vyžaduje:
    - § PDFBox-0.7.2.dll
    - § IKVM.GNU.Classpath
- Dokument RTF
  - Najjednoduchšie je využiť komponentu **RichTextBox** z referencie System.Windows.Forms

Sú voľne dostupné zdrojové kódy systémov vytvorených nad DotLucene ktoré spracúvajú všetky spomenuté typy súborov a môžeme ich použiť pre naše účely. Sú to napríklad:

- Desktop Search Application: Part 1 [codeproject.com]
- Seekafire Server 1.5 [seekafire.org]

### *Záver*

Ku knižnici DotLucene je dostupná kompletná API dokumentácia vo formáte CHM. Na záver už hádam iba uvedieme tvrdenie produkčného tímu DotLucene: „Existujú tri rýchlosti vyhľadávania: Rýchle, Super rýchle a DotLucene“

## 2.2.5 The Gnome Data Mine

### *Opis nástroja*

Gnome Data Mine Tools (GDM) je kolekcia nástrojov určených na data – mining. Nie je to teda vyhľadávací nástroj ako napríklad Lucene. GDM je voľne dostupné a je poskytované pod licenciou GNU GPL. Adresa je:

<http://www.togaware.com/datamining/gdatamine/gnome-datamine-tools.tar.gz>

Archív obsahuje konzolové data – mining aplikácie a GUI nastavbu nad nich. Prostredie je určené pre operačný systém Linux (odporúčaná distribúcia je Debian GNU / Linux). GUI nadstavba využíva jazyk Python a pre svoju funkčnosť vyžaduje desktopové prostredie Gnome.

Ťažiskom GDM sú konzolové aplikácie a tieto sú dostupné aj pre operačný systém Windows. Tie je možné stiahnuť z adries:

<http://fuzzy.cs.uni-magdeburg.de/~borgelt/src/apriori.exe>  
<http://fuzzy.cs.uni-magdeburg.de/~borgelt/src/winbayes.zip>  
<http://fuzzy.cs.uni-magdeburg.de/~borgelt/src/wintable.zip>  
<http://fuzzy.cs.uni-magdeburg.de/~borgelt/src/windtree.zip>

Posledné zdrojové kódy sú z 2. 3. 2004.

### **Zoznam nástrojov**

GDM obsahuje tri hlavné nástroje, tie sú:

- Apriori Association Rules – hľadanie asociačných pravidiel pomocou Apriori algoritmu
- Bayes Classifier – naivné Bayes klasifikovanie
- Decision Trees – vytváranie rozhodovacieho a regresného strom

### **Apriori Association Rules**

Prvý z opisovaných nástrojov vytvára asociačné pravidlá na základe algoritmu Apriori. Asociačné pravidlo je silný nástroj na hľadanie spojitostí v dátach. Používa sa v prípadoch, kedy na základe doteraz spracovaných dát potrebujeme predpovedať, že či a s akou pravdepodobnosťou sa ďalšie dáta dajú zaradiť do nejakej z množín klasifikovaných dát. Hlavným problémom asociačného pravidla je veľké množstvo možných pravidiel. Už len pri tisícke dát sú milióny možných priraďovacích pravidiel. Preto je potrebné použiť efektívny algoritmus, ktorý obmedzí prehľadávaný priestor a prezrie podstatné vzťahy s tým, že tie dôležité nevynechá. Jeden z takých algoritmov je práve algoritmus Apriori.

### **Príklad**

Vstup obsahuje položky oddelené medzerami. Jednotlivé záznamy sú na samostatných riadkoch. Tento príklad uvádza položky (tovar v nákupnom košíku) reprezentované číslami.

```
1 2 3
1 4 5
2 3 4
1 2 3 4
2 3
1 2 4
4 5
1 2 3 4
```

```
3 4 5
1 2 3
```

Príkaz spracúvajúci vstupné dáta a vytvárajúci asociačné pravidlá:

```
apriori.exe test1.tab test1.tab.out
```

Výstup je v tvare  $X \leftarrow Y_1 Y_2 \dots (P_1\%, P_2\%)$ .

Položka  $X$  reprezentuje položku, ktorej pravdepodobnosť výskytu v zázname je  $P_2$  v prípade, že záznam obsahuje položky  $Y_2$ .  $P_1$  vyjadruje percentuálny podiel takýchto prípadov vo vstupnej množine.

```
4 <- 5 (30.0, 100.0)
2 <- 1 (60.0, 83.3)
3 <- 2 (70.0, 85.7)
2 <- 3 (70.0, 85.7)
4 <- 5 1 (10.0, 100.0)
4 <- 5 3 (10.0, 100.0)
3 <- 1 2 (50.0, 80.0)
2 <- 1 3 (40.0, 100.0)
2 <- 1 3 4 (20.0, 100.0)
```

### ***Bayes Classifier***

Druhým z opisovaných nástrojov je nástroj na klasifikovanie dát do tried na základe naivného Bayesovho klasifikátora.

### ***Príklad***

Vstup obsahuje položky oddelené medzerami. Jednotlivé záznamy sú na samostatných riadkoch. Za triedu sa považuje posledný stĺpec vstupu. Jednotlivé stĺpce môžu byť pomenované – mená musia byť prvým riadkom vstupu. Príklad uvádza, aký druh lieku použiť na základe informácií o krvnom tlaku, veku a pohlaví.

Sex	Age	Blood_pressure	Drug
male	20	normal	A
female	73	normal	B
female	37	high	A
male	33	low	B
female	48	high	A
male	29	normal	A
female	52	normal	B
male	42	low	B
male	61	normal	B
female	30	normal	A
female	26	low	B
male	54	high	A

Pred tým, ako je možné vykonať klasifikáciu dát do tried, je potrebné určiť domény jednotlivých vstupov. To sa uskutočňuje samostatným príkazom:

```
dom -a drug.tab drug.dom
```

Výstupom je určenie domény pre každý stĺpec buď vymenovaním alebo (ako aj v tomto príklade) určením štandardného typu (ZZ reprezentuje množinu čísel).

```
/*-----
----
domains
-----
--*/
dom(Sex) = { female, male };
dom(Age) = ZZ;
dom(Blood_pressure) = { high, low, normal };
dom(Drug) = { A, B };
```

Samotné klasifikovanie sa uskutoční spustením príkazu:

```
bci -sa drug.dom drug.tab drug.nbc
```

Výstupná klasifikácia obsahuje absolútnu početnosť jednotlivých tried vo vstupnom súbore. Za ním nasleduje určenie pravdepodobnosti na základe určitého pravidla – napr. aká je pravdepodobnosť, že použitý liek má byť A, keď poznám vek. Pre symbolické parametre je použitá tabuľka výskytu – prípad závislosti od krvného tlaku. Pre numerické parametre je použité normálne rozdelenie pravdepodobnosti v tvare  $N(u, \sigma^2)$  [n], kde  $u$  je stredná hodnota,  $\sigma$  variácia a  $n$  počet pravidiel, na základe ktorých bola pravdepodobnosť vypočítaná.

```
/*-----
----
naive Bayes classifier
-----
--*/
nbc(Drug) = {
  prob(Drug) = {
    A: 6,
    B: 6 };
  prob(Age|Drug) = {
    A: N(36.3333, 161.867) [6],
    B: N(47.8333, 310.967) [6] };
  prob(Blood_pressure|Drug) = {
    A: { high: 3, low: 0, normal: 3 },
    B: { high: 0, low: 3, normal: 3 } };
};
```

### **Decision Trees**

Posledným nástrojom je nástroj na vytváranie rozhodovacích stromov zo vstupných dát.

#### **Príklad**

Vstup je rovnakého tvaru ako v prípade Bayes klasifikátora:

Sex	Age	Blood_pressure	Drug
male	20	normal	A
female	73	normal	B
female	37	high	A
male	33	low	B

```
female 48  high      A
male   29  normal    A
female 52  normal    B
male   42  low       B
male   61  normal    B
female 30  normal    A
female 26  low       B
male   54  high      A
```

Pred tým, ako je možné vytvoriť rozhodovací strom, je potrebné určiť domény jednotlivých vstupov. To sa uskutočňuje samostatným príkazom:

```
dom -a drug.tab drug.dom
```

Výstupom je určenie domény pre každý stĺpec buď vymenovaním alebo (ako aj v tomto príklade) určením štandardného typu (ZZ reprezentuje množinu čísel).

```
/*-----
----
domains
-----
--*/
dom(Sex) = { female, male };
dom(Age) = ZZ;
dom(Blood_pressure) = { high, low, normal };
dom(Drug) = { A, B };
```

Príkaz na vytvorenie rozhodovacieho stromu je:

```
dti -a drug.dom drug.tab drug.dt
```

Výstup programu je strom, ktorý je vetvený na základe pravidiel, ktoré môžu byť vnárané. Pravidlá majú vylepšenú formu príkazu switch ako ho poznáme z jazyka C. Z výsledku programu sa dajú vyčítať informácie o strome. Tento konkrétny rozhodovací strom pracuje s tromi pravidlami, jeho hĺbka je tri a počet uzlov šesť. Bol odvodený zo vstupu, ktorý obsahoval 12 záznamov.

```
/*-----
----
decision tree
-----
--*/
dtree(Drug) =
{ (Blood_pressure)
  high  :{ A: 3 },
  low   :{ B: 3 },
  normal:{ (Age|41)
            <:{ A: 3 },
            >:{ B: 3 } } };

/*-----
----
number of attributes: 3
```



```
tree height      : 3
number of nodes  : 6
number of tuples : 12
```

---\* /

### **MLC++**

MLC++ je knižnica tried C++ určená na učenie sa s učiteľom. Poskytujú sa s ňou aj MLC++ nástroje. Do verzie 1.3.x bola knižnica vyvíjaná Stanfordskou univerzitou. Táto verzia je stále poskytovaná pod hlavičkou SGI. Verzie 2.0 a vyššie obsahujú vylepšenia oproti pôvodnej verzii a sú vyvíjané pod záštitou SGI.

MLC++ poskytuje všeobecné algoritmy učenia sa, ktoré môžu byť využité koncovými používateľmi, analytikmi, profesionálmi alebo na výskumné účely. Hlavným účelom je poskytnúť široké spektrum nástrojov na data – mining, urýchliť vývoj nových algoritmov, zvýšiť spoľahlivosť softvéru, poskytnúť nástroje na porovnávanie a zobraziť informácie vizuálne. S knižnicou sa poskytujú utility, ktoré knižnicu využívajú a je možné ich využiť. Originálne zdrojové kódy sú voľne k dispozícii, upravené a vylepšené zdrojové kódy vyvíjané pod záštitou SGI sú naproti tomu použiteľné iba na výskumné účely.

Adresa MLC++: <http://www.sgi.com/tech/mlc/>

Posledné zdrojové kódy sú z 21. 12. 1997.

### **Zhodnotenie**

MLC++ obsahuje oproti GDM podstatne viac implementovaných algoritmov. Napriek tomu GDM veľmi dobre plní svoju funkciu a zároveň je stále rozširovaný. GDM je dostupný pod licenciou GNU GPL, oproti tomu MLC++ je použiteľný iba na výskumné účely, jeho komerčné využitie je teda otáznave.

Hoci sa nejedná o vyhľadávacie systémy, charakter týchto nástrojov nevyklučuje, že by sa mohli použiť v inej časti modelu systému.

### 3. Predbežná špecifikácia požiadaviek

V tejto kapitole definujeme požiadavky na vytváraný informačný systém. Pri ich definovaní vychádzame zo zadania projektu ako aj z požiadaviek prostredia, v ktorom bude výsledný softvér používaný. Požiadavky sme z hľadiska ich charakteru rozdelili na funkcionálne, požiadavky na prevádzku systému a požiadavky z hľadiska ďalšieho vývoja systému. (Tab. 1 **Chyba! Nenašiel sa žiaden zdroj odkazov.**)

Funkcionálne požiadavky	Požiadavky na prevádzku systému	Požiadavky z hľ. ďalšej implementácie
<p>Vyhľadávanie informácií z rôznych zdrojov – typov dokumentov a zoradovanie podľa priority na základe rôznych kritérií:</p> <p>klúčové slová</p> <p>príslušnosť k projektu</p> <p>predošlé výbery, vyhľadávania</p> <p>predošlá činnosť používateľa a iné</p>	<p>Užívateľské rozhranie programu – na platforme .NET</p>	<p>Rozšírenie systém o moduly transformácie dokumentov do metamodelu.</p>
	<p>Interaktívna komunikácia s rôznymi textovými editormi (najmä s MS Word)</p>	<p>Prepojenie s archívom dokumentov (MS SourceSafe, CVS, ...)</p>
	<p>Jednoduchosť, minimálny potrebný zásah používateľa</p>	

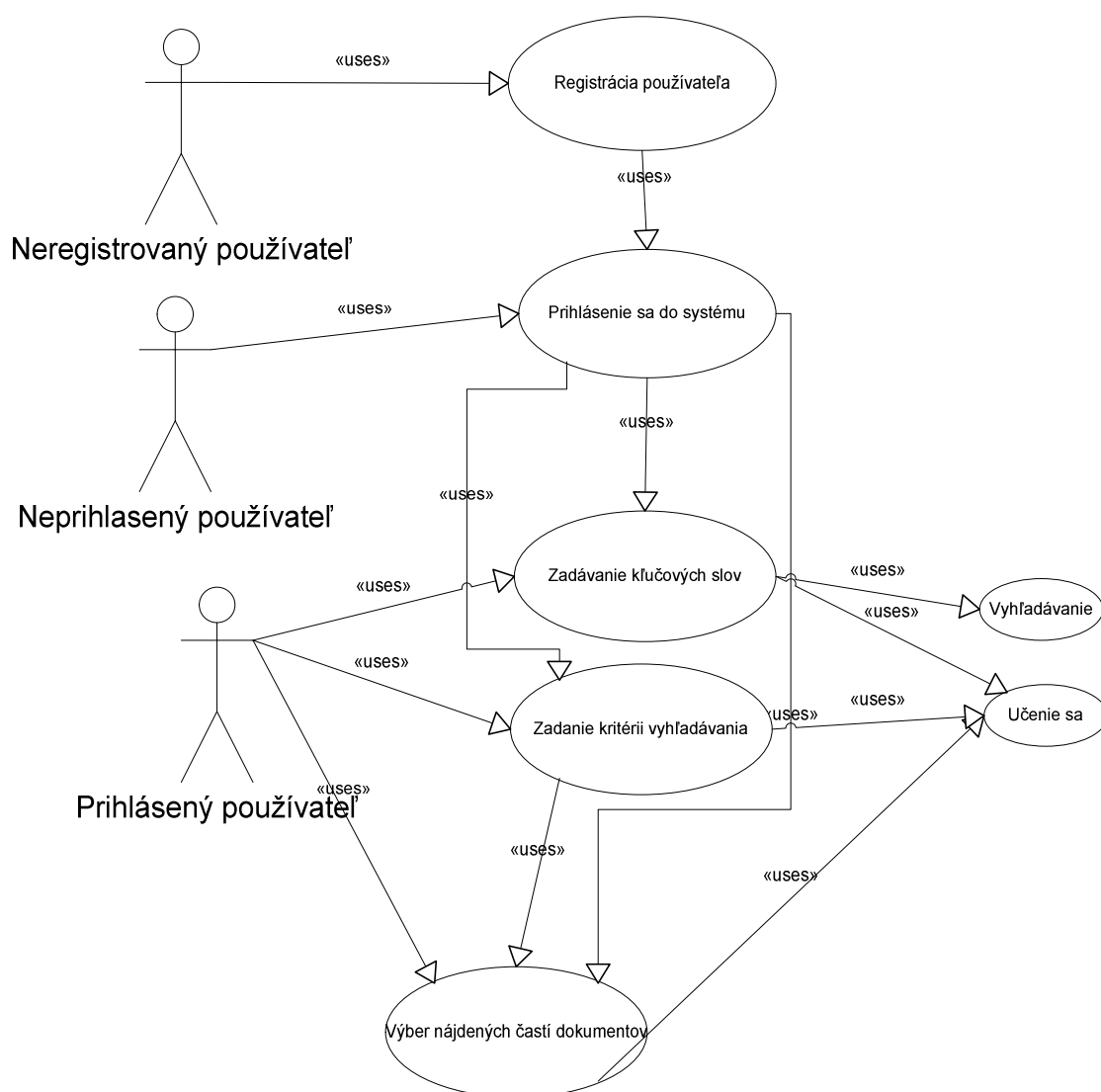
Tab. 2: Funkcionálne a nefunkcionálne požiadavky na systém.

Základnou funkciou navrhovaného systému je uľahčenie práce používateľom systému pri tvorbe rôznych typov dokumentácií. Systém umožní používateľom vyhľadávať v archíve dokumentov na základe zvolených kritérií. Cieľom je odhadnúť, akú kapitolu, resp. akej téme sa ide používateľ venovať. Vyhľadaním podobných dokumentov a ponúknutím relevantných odsekov z archívu, systém ušetrí používateľovi značnú časť práce tým, že ponúkne už vytvorené formulácie viet, odseky, kapitoly atď.

### 3.1 Prípady použitia

Z predchádzajúcich kapitol si je možné urobiť predstavu o náročnosti a rozsiahlosti problematiky. Vzhľadom na to, že jedným z našich primárnych cieľov je vytvorenie produktu, ktorý bude spĺňať požiadavky spoločnosti Gratex, sme veľkú väčšinu času venovali analýze problémovej oblasti. Náš projekt nestavia na žiadnej predchádzajúcej práci a chceme vytvoriť podmienky pre úspešne pokračovanie iného tímu na nami začatej práci. Prípady použitia je preto vhodné chápať ako veľmi predbežný pohľad na systém, ktoré sa po ujasnení sporných otázok adekvátne rozšíria.

Na obrázku (Obr. 8) je znázornený diagram prípadov použitia, ktorý poskytuje prehľadnú informáciu o poskytovanej funkcionalite vo vzťahu k jednotlivým hráčom.



Obr. 8: Diagram prípadov použitia.

#### 3.1.1 Identifikácia hráčov (používateľov systému)

**Neregistrovaný používateľ** – používateľ, ktorého identita je v systéme neznáma, nebol ešte v systéme registrovaný a systém ho nepozná. Takýto používateľ nemôže vyhľadávať v znalostiach.

**Neprihlásený používateľ** – používateľ, ktorý je v systéme registrovaný – systém pozná jeho identitu. Neprihlásený používateľ nemôže v systéme vyhľadávať. Môže sa doň len prihlásiť.

**Prihlásený používateľ** – používateľ, ktorý je v systéme registrovaný a je v systéme práve prihlásený. Takýto používateľ môže v systéme vyhľadávať. Systém pozná jeho identitu a vie sa podľa toho prispôbiť svoju ďalšiu činnosť.

### 3.1.2 Prípady použitia

#### a) Registrácia používateľa

<b>Identifikátor</b>	TR01	
<b>Názov</b>	Registrácia používateľa	
<b>Opis</b>	Predávajúci objednal tovar na sklad, ktorý ešte v databáze nie je evidovaný	
<b>Priorita</b>	3 = nízka	<b>Frekvencia</b>   Týždenne jednotky nových užívateľov
<b>Vstup. podm.</b>	Práve registrovaný používateľ v systéme nebol doposiaľ registrovaný.	
<b>Výstup. podm.</b>	Používateľ je zaregistrovaný.	
<b>Používatelia</b>	Neregistrovaný používateľ	
<b>Základná postupnosť</b>	<b>Krok</b>	<b>Činnosť</b>
	1	Systém zobrazí formulár na registráciu používateľa.
	2	Predávajúci zadá osobné údaje a rolu (funkciu) vo firme, v ktorej je systém nasadený.
	3	Systém skontroluje, či sa daný používateľ už systéme nenachádza.
	4	Systém uloží informáciu o novom používateľovi do databázy
	5	Systém zobrazí informáciu o uložení do databázy
<b>Alternatívna postupnosť</b>	<b>Krok</b>	<b>Činnosť</b>
	3.a	Ak sa používateľ nachádza v databáze, bude vyzvaný na opätovné zadanie jedinečných osobných údajov. Pokračuje krokom 2.

#### b) Prihlásenie do systému

<b>Identifikátor</b>	TR02	
<b>Názov</b>	Prihlásenie do systému	
<b>Opis</b>	Registrovaný používateľ sa prihlasuje do systému.	
<b>Priorita</b>	2 = stredná	<b>Frekvencia</b>   Týždenne rádovo stovky prihlásení.
<b>Vstup. podm.</b>	Práve prihlasovaný používateľ je v systéme registrovaný.	
<b>Výstup. podm.</b>	Používateľ je prihlásený.	
<b>Používatelia</b>	Registrovaný používateľ	
<b>Základná postupnosť</b>	<b>Krok</b>	<b>Činnosť</b>
	1	Systém zobrazí formulár na prihlásenie používateľa.
	2	Používateľ zadá prihlasovacie údaje a rolu (funkciu) vo firme, v ktorej je systém nasadený.
	3	Systém skontroluje, či sa daný používateľ už systéme nenachádza a overí správnosť prihlasovacích údajov.
	4	Systém uloží informáciu o prihlásení.
	5	Systém zobrazí formulár pre vyhľadávanie.

Alternatívna postupnosť	Krok	Činnosť
	3.a	Ak používateľ zadá nesprávne prihlasovacie údaje, pokračuje krokom 2.

*c) Zadanie kritérií pre vyhľadávanie*

<b>Identifikátor</b>	TR03		
<b>Názov</b>	<b>Zadanie kritérií pre vyhľadávanie</b>		
<b>Opis</b>	Prihlásený používateľ zadáva kritéria, podľa ktorých chce vyhľadávať.		
<b>Priorita</b>	1 = vysoká	<b>Frekvencia</b>	Týždenne rádovo tisícky vyhľadávaní.
<b>Vstup. podm.</b>	Používateľ musí byť v systéme prihlásený.		
<b>Výstup. podm.</b>	Nie je.		
<b>Používatelia</b>	Prihlásený používateľ		
<b>Základná postupnosť</b>	<b>Krok</b>	<b>Činnosť</b>	
	1	Systém zobrazí formulár pre vyhľadávanie.	
	2	Predávajúci zadá kritéria, podľa ktorých chce vyhľadávať. (typ projektu, jeho rola, ...)	
	3	Systém požiadavku zaznamená a spustí vyhľadávanie.	
<b>Alternatívna postupnosť</b>	<b>Krok</b>	<b>Činnosť</b>	
	2.a	Ak používateľ nezadá kritéria, systém požiadavku nezaznamená a bude vyhľadávať bez ohľadu na kritéria.	

*d) Zadanie kľúčových slov*

<b>Identifikátor</b>	TR04		
<b>Názov</b>	<b>Zadanie kľúčových slov</b>		
<b>Opis</b>	Prihlásený používateľ zadáva kľúčové slová, podľa ktorých chce vyhľadávať.		
<b>Priorita</b>	1 = vysoká	<b>Frekvencia</b>	Týždenne rádovo tisícky vyhľadávaní.
<b>Vstup. podm.</b>	Používateľ musí byť v systéme prihlásený.		
<b>Výstup. podm.</b>	Používateľ zadal kritéria resp. kľúčové slová pre vyhľadanie.		
<b>Používatelia</b>	Prihlásený používateľ		
<b>Základná postupnosť</b>	<b>Krok</b>	<b>Činnosť</b>	
	1	Systém zobrazí formulár pre vyhľadávanie.	
	2	Predávajúci zadá kľúčové slová, podľa ktorých chce vyhľadávať.	
	3	Systém požiadavku zaznamená. (učí sa).	
	4	Systém vyhľadá relevantné časti dokumentov.	
<b>Alternatívna postupnosť</b>	<b>Krok</b>	<b>Činnosť</b>	
	2.a	Ak používateľ nezadá kľúčové slová, systém požiadavku nezaznamená a ostáva v kroku 2.	

*e) Výber nájdených častí dokumentov.*

<b>Identifikátor</b>	TR05		
<b>Názov</b>	<b>Výber nájdených častí dokumentov.</b>		
<b>Opis</b>	Prihlásený používateľ po zadaní kľúčových slov, vyberá z ponúknutých nájdených relevantných častí dokumentov.		
<b>Priorita</b>	1 = vysoká	<b>Frekvencia</b>	Týždenne rádovo tisícky vyhľadávaní.
<b>Vstup. podm.</b>	Používateľ musí byť v systéme prihlásený. Používateľ zadal kritéria a kľúčové slová pre vyhľadávanie.		

<b>Výstup. podm.</b>	Systém ponúkol relevantné časti dokumentov.	
<b>Používatelia</b>	Prihlásený používateľ	
<b>Základná postupnosť</b>	<b>Krok</b>	<b>Činnosť</b>
	1	Systém zobrazí formulár pre vyhľadávanie.
	2	Vo formulári zobrazí názvy dokumentov a ich relevantné odseky, ktoré sa zhodujú so zadanými kritériami pre vyhľadávanie.
	3	Používateľ vyberá z ponúknutých dokumentov.
	4	Systém sleduje jeho činnosť a učí sa..
<b>Alternatívna postupnosť</b>	<b>Krok</b>	<b>Činnosť</b>
	2.a	Ak systém nezobrazí žiadne relevantné časti dokumentov, vyzve používateľa o zmenu kritérií vyhľadávania. Pokračuje prípadom použitia označeným identifikátorom: TR03

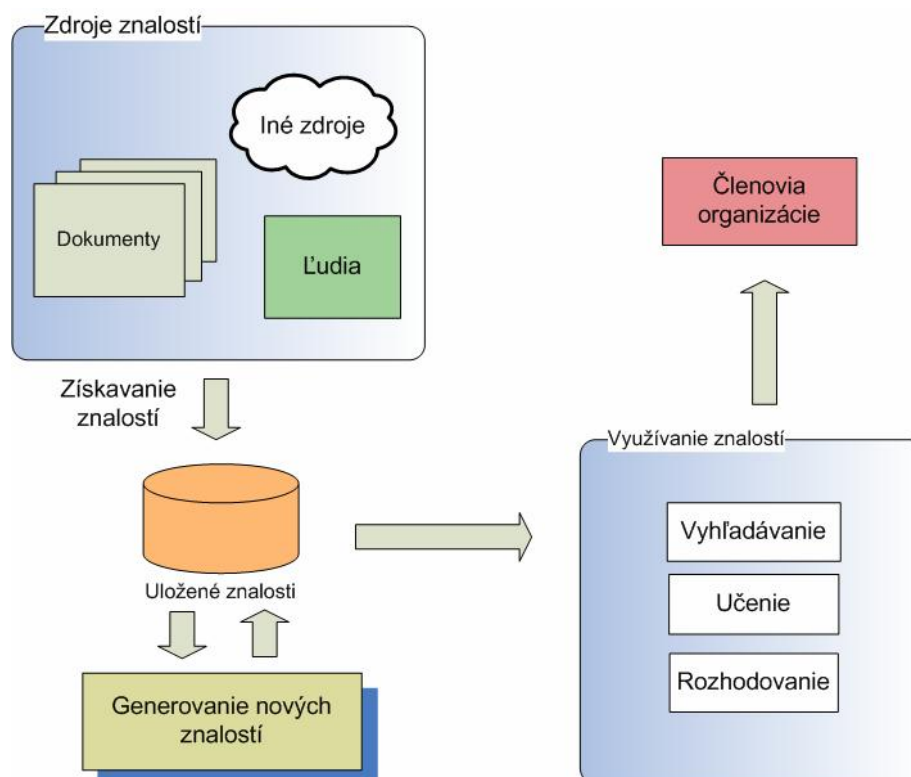
## 4. Návrh

### 4.1 Hrubý návrh systému

Predbežné fragmenty návrhu uvedieme v tejto kapitole, pričom dôraz kladieme na konkrétny príklad použitia. Po ujasnení problémových oblastí v priebehu zjemňovania návrhu, bude slúžiť tento príklad ako základ pre vytvorenie scenárov použitia.

#### 4.1.1 Fázy manažmentu znalostí

Na manažment znalostí sa môžeme pozeráť ako na skupinu procesov znázornených na obrázku (Obr. 9). Náš projekt je orientovaný na tvorbu dokumentácie. Preto primárnym zdrojom znalostí sú vytvorené a v budúcnosti vytvárané dokumenty. Ľudia dodávajú systému dôležitú spätnú väzbu pre budúce zdokonaľovanie produkovaných výsledkov.



Obr. 9: Rozdelenie manažmentu znalostí na skupiny procesov

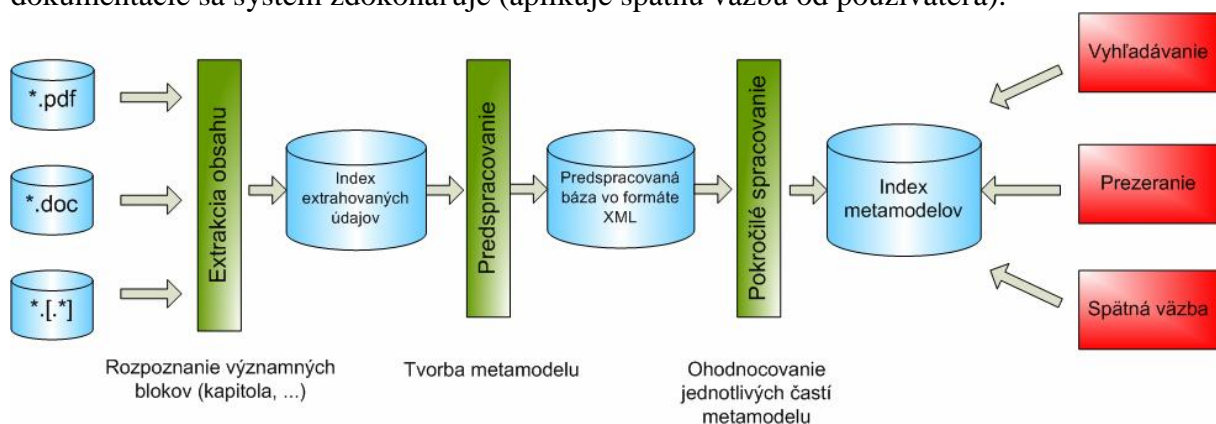
#### 4.1.2 Vyhľadávanie a metamodel

Celý proces je podporovaný vyhľadávaním v dokumentoch. Každý dokument rešpektuje metamodel, čo môžeme chápať ako súhrn pravidiel – šablónu pre vytváranie dokumentov. Každý dokument pozostáva z blokov, kapitol, resp. odsekov teda má určitú štruktúru, resp. hierarchiu. Použitím rôznych modulov pre rôzne typy dokumentov (doc, pdf, rtf a iné) bude možné tieto bloky v dokumente vyhľadať a reprezentovať ich pomocou metaznačiek vo

vytvorenom tzv. metadokumente. O každom takomto bloku sa budú uchovávať metainformácie ako napríklad:

- koľkokrát bol daný blok použitý,
- v akej súvislosti bol použitý (téma dokumentu, odseku),
- akým používateľom bol použitý (rola používateľa v tíme, funkcia),
- s akými kľúčovými slovami bol vyhľadávaný a následne použitý,
- s akými inými odsekmi bol naraz použitý,
- v akom projekte (téma projektu – bezpečnosť, financie, sieťová komunikácia...) sa doposiaľ použil,
- či ho bolo potrebné upravovať a aké percento bolo upravené v novom dokumente oproti pôvodnému zneniu bloku,
- pozícia bloku v rámci dokumentu a budúca pozícia v novom dokumente,
- susedné bloky.

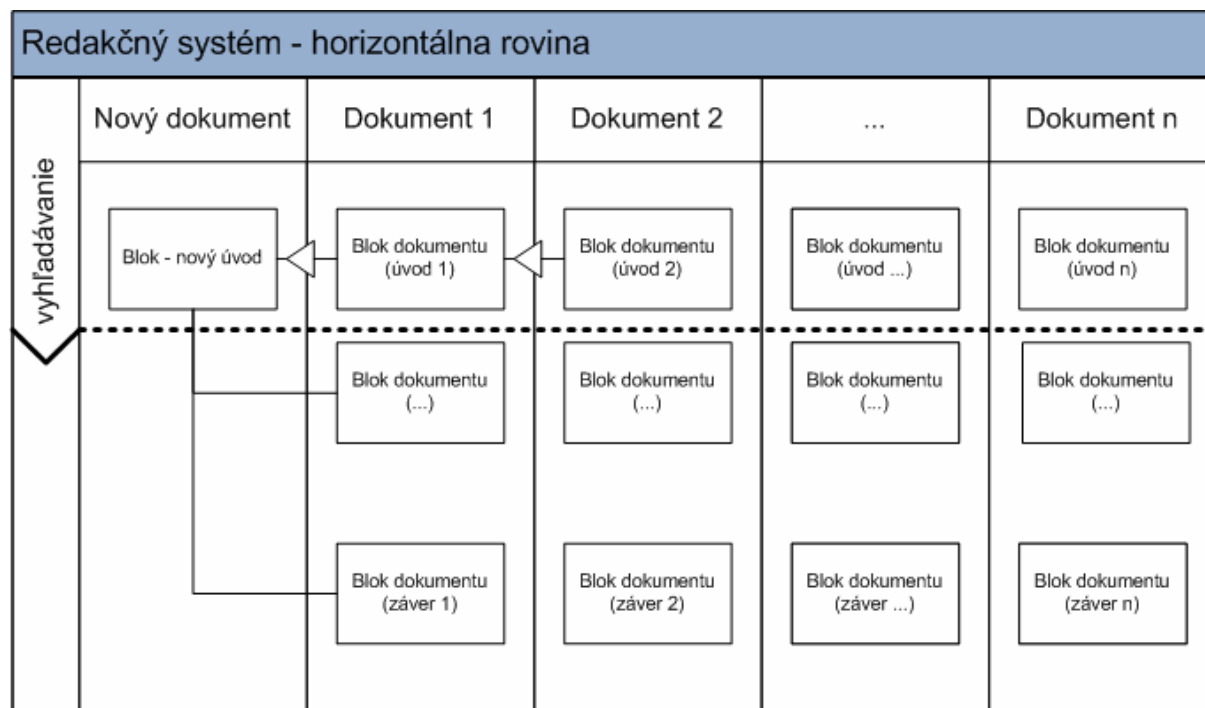
Metainformácie pomôžu určiť prioritu daného bloku pri vyhľadávaní a možno ich získať jednak inferečnými mechanizmami v doposiaľ získaných znalostiach, a jednak pozorovaním činnosti používateľa. Vymenovaný zoznam metainformácií nie je konečný a bude sa na základe získaných skúseností rozširovať. Náš projekt počíta s prácou nad dokumentáciou, ktorá rešpektuje v maximálnej miere metamodel. Pre úplnosť uvádzame na obrázku (Obr. 10) aj podporný proces transformácie dokumentácie do metamodelu spolu s procesmi využívania a zlepšovania bázy znalostí. Využívaním indexu metamodelov pre tvorbu dokumentácie sa systém zdokonaľuje (aplikuje spätnú väzbu od používateľa).



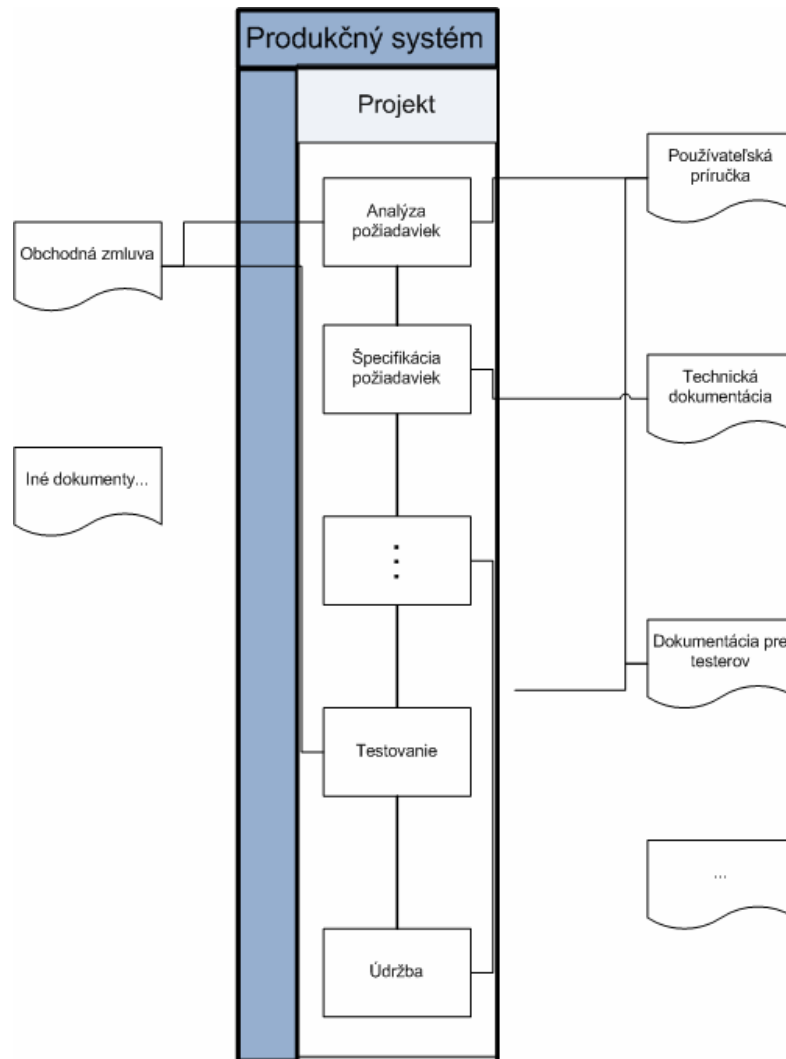
**Obr. 10: Proces transformácie dokumentácií do metamodelu**

Metainformácie budú tvoriť metadokument. Ten bude zostavený na základe tzv. metamodelu popisujúceho všeobecnú štruktúru konkrétneho metadokumentu. Opisovaný systém je takzvaný redakčný systém – pracuje nad viacerými dokumentmi v horizontálnej rovine (Obr. 11). No nad týmto metamodelom s podobnými metainformáciami je možné vytvoriť aj iný typ systému – napr. produkčný systém, ktorý bude v rámci jedného projektu ponúkať rôzne pohľady naň (Obr. 12). Navrhovaný systém by mal byť teda rozšíriteľný a využiteľný aj v inej oblasti.





Obr. 11: Redakčný systém – vyhľadávanie v archíve dokumentov



Obr. 12: Produkčný systém – generovanie dokumentov na základe projektu. (príklad iného možného použitia metamodelu)

### 4.1.3 Príklad fungovania navrhovaného systému

Fungovanie systému popíšeme na príklade práce zo systémom ktorého databáza známych dokumentov obsahuje len dva dokumenty: technickú dokumentáciu systému Alfa a používateľskú dokumentáciu systému Beta. Dokumenty sú v databáze známych dokumentov uchovávané v štruktúrovanom formáte. Telo každého dokumentu obsahuje názov dokumentu, typ dokumentu a konečne postupnosť kapitol aj s ich telom. Práve štruktúrovanosť uchovávaných dokumentov uľahčuje analýzu ich jednotlivých častí (kapitol) dokumentu.

V našom príklade budeme analyzovať kapitolu úvod, preto v opise ciel dokumentov od ostatných kapitol abstrahujeme. Nasleduje popis dokumentov, ktoré tvoria databázu známych dokumentov.

#### **Dokument k systému Alfa**

**Názov:**

Technická dokumentácia systému Alfa

**Typ:**

Technická dokumentácia

**Úvod:**

Predkladaný dokument obsahuje technickú dokumentáciu systému Alfa, ktorý vytvára back-endové riešenie systému Izmund, slúžiaceho na konsolidáciu bankových platieb.

*[Nasledujú ďalšie kapitoly dokumentu]*

**Dokument k systému Beta****Názov:**

Používateľská dokumentácia systému Beta

**Typ:**

Používateľská dokumentácia

**Úvod:**

Dokument opisuje hlavné možnosti použitia systému Beta, ktorý je vnútrofiremným knowledge base systémom. V ďalších kapitolách je popísaný krok po kroku postup, ktorý slúži používateľovi aplikácie ako návod na použitie systému Beta.

*[Nasledujú ďalšie kapitoly dokumentu]*

**Popis práce zo systémom****Vstupná podmienka:**

Báza znalostí znalostného systému obsahuje základné doposiaľ neupravené poznatky, čo znamená že systém do tejto chvíle nebol používaný.

**Použitie systému:**

- Predpokladajme že používateľ zvolí vytvorenie nového dokumentu a ako typ dokumentu zvolí technickú dokumentáciu.
- Používateľ zvolí písanie kapitoly Úvod a napíše: „dokument knowledge base návod“. Systém počas písania používateľa automaticky prehľadáva databázu známych dokumentov a snaží sa nájsť dokumenty, ktorých kapitola úvod bude používateľovi najviac vyhovovať.
- Systém spustí ohodnocovanie príslušných kapitol dokumentov obsiahnutých v báze znalostí. Ohodnocovanie sa deje v dvoch fázach. Najskôr ohodnotí kapitoly všetkých dokumentov vyhľadávací systém. Čím viac kľúčových slov zadaných používateľom kapitola má, tým väčšie hodnotenie dostane. Dokumenty ktorých kapitoly majú najlepšie hodnotenie sú hodnotené znalostným systémom. Tento posudzuje rôzne aspekty dokumentov a podľa nich im prideluje hodnotenia. Dokumenty sú

používateľovi predkladané v poradí podľa súčtu oboch hodnotení, kedy dokumentu s najväčším hodnotením je používateľovi najviac odporúčaný.

- V našom prípade vyhľadávací systém ohodnotí lepšie dokument systému Beta, lebo sa v jeho kapitole Úvod nachádza viac kľúčových slov. Dokument o systéme Alfa zaradí vyhľadávací systém na druhé miesto.
- V ďalšom kroku dokumenty ohodnotí znalostný systém. Tento zohľadní fakt, že typ dokumentu, ktorý píše používateľ, je rovnaký ako typ dokumentu o systéme Alfa. Zvýši preto hodnotenie dokumentu o systéme alfa.
- Hodnotenia dokumentu o systéme Alfa je však aj po zvýšení stále menšie ako hodnotenie dokumentu o systéme Beta a v zozname odporúčaných dokumentov je na prvom mieste prezentovaný opis systému Beta. Pri popise vlastností dokumentácie systému Alfa je však zahrnutý fakt, že má rovnaký typ ako vytváraný dokument a je ponúknutá možnosť práve kvôli tejto skutočnosti použiť časť Úvod s opisu systému Alfa.
- Ak je táto možnosť používateľom zvolená, systém v báze znalostí zvýši mieru ohodnocovania dokumentov na základe ich typov.

Pri ďalšom použití systému bude znalostný systém vo vyššej miere ohodnocovať dokumenty ktoré majú rovnaký typ ako vytváraný dokument.

Ak by sa vyššie opísaný postup zopakoval opäť dokument systému Alfa by mal finálne lepšie ohodnotenie než pri prvom použití systému. Toto však neznamená že by mal už pri druhom použití lepšie ohodnotenie než opis systému Beta (miera učenia závisí na koeficiente rýchlosti učenia systému, ktorý bude stanovený pri doladovaní). Je však zaručené, že pri dostatočnom počte opakovania opísaného postupu bude dokument Alfa zaradovaný na prvé miesto v rebríčku odporúčaných dokumentov a to aj navzdory faktu, že vo svojom tele obsahuje menej kľúčových slov zadaných používateľom než systém Beta.

Opísaný učiaci proces sa zakladá na myšlienke zvyšovania relevantnosti tých súvislostí, pre ktoré sa používatelia pri výbere dokumentu rozhodujú. Tento algoritmus učenia bude aplikovaný na všetky súvislosti, ktoré bude znalostný systém pri posudzovaní relevantnosti kapitoly dokumentu zvažovať.

### **Ďalšie požiadavky:**

Systém dokáže vyhľadávať v dokumentoch v dokumentoch vo vlastnom metaformáte. Táto funkcionlita bude ďalej rozšíriteľná pomocou modulov na možnosť vyhľadávania vo formátoch .doc, .pdf a iných.

Systém bude samostatnou aplikáciou, ktorá bude v budúcnosti prepojitelná s GKO – (Gratex Knowledge Office).

Ďalej sa dôraz kladie aj na jednoduchosť užívateľského rozhrania. Celý proces by mal vyžadovať len minimum interakcie s používateľom, ale zistiť a naučiť sa z nej maximum. Systém by mal vedieť vyhľadávať v archívoch rôznych typov ako napr. MS SourceSafe resp. CVS. Navrhnutý softvér (užívateľské rozhranie) bude implementovaný na platforme .NET.

## **Prílohy**

## A. Použitá literatúra

- BECHHOFER, Sean et al.: OWL Web Ontology Language Reference, W3C Recommendation 10.2.2004. W3C. 2004. <http://www.w3.org/TR/2004/REC-owl-ref-20040210/> (15.11.2006)
- DAVENPORT T.H., PRUSAK L.: 1998. Working Knowledge – How Organizations Manage What They Know, Boston, Harvard Business School Press.1998 ISBN 0-87584-655-6
- DOTLUCENE, 2006, DotLucene - Resources for Lucene.Net <http://www.codeproject.com/csharp/desktopsearch1.asp> (6.11.2006)
- DUFFNER, S.: A Knowledge Portal for Multi-Project Management. Karlsbad, 2000. 90 s. Diplomová práca na Univerzite Karlshure. Vedúci práce: prof. Dr. Studer.
- KELEMEN J.,KUBÍK A., LENHARČÍK I., MIKULECKÝ P.: Tvorba expertních systému v prostředí Clips. Praha: Grada, 1999. 247 s. ISBN 80-7169-501-7
- KONCHADY M.: Text Mining Application Programing, Boston: Charles Rive Media 2006, 405 s. ISBN 1-58450-460-9
- LETECKY D., 2006, Full-Text Search: Fast, Damn Fast and DotLucene, <http://www.codeproject.com/useritems/damnfastsearch.asp> (6.11.2006)
- LETECKY D., 2006, Desktop Search Application: Part 1, <http://www.codeproject.com/csharp/desktopsearch1.asp> (6.11.2006)
- NONAKA, I. and TAKEUCHI, I.: The Knowledge Creating Company. How Japanese Companies Create the Dynamics of Innovation. Oxford University Press, New York, NY/Oxford. 1995.
- PARALIČ J.: Manažment znalostí – podklady k prednáškam, <http://www.tuke.sk/paralicj/mz.html> (15.11.2006)
- PROBST G., RAUB S., ROMHARDT K.: Wissen managen. Gabler Verlag, Wiesbaden, 1997.
- PROTÉGÉ – znalostný systém. Stanford Medical Informatics. 2006. <http://protege.stanford.edu/> (15.11.2006)
- RILEY G.: Clips. A tool for building expert systems. <http://www.ghg.net/clips/CLIPS.html> (15.11.2006)
- SENGE, P.: The Fifth Discipline – The Art and Practice of the Learning Organization, New York: Doubleday, 1990.
- SMITH, Michael K. – WELTY, Chris – MCGUINNESS, Deborah L.: OWL Web Ontology Language Guide, W3C Recommendation 10.2.2004. W3C. 2004. <http://www.w3.org/TR/2004/REC-owl-guide-20040210/> (15.11.2006)
- TOGAWARE: The Gnome Data Mine. <http://www.togaware.com/datamining/gdatamine/> (14. 11. 2006)
- SGI - MLC++: Home Page. <http://www.sgi.com/tech/mlc/> (14. 11. 2006)
- TRUNECEK J: 2001. Znalosti v roce 2001. Moderní řízení č.11/2001 s. 39-43