FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Framework pro tvorbu diagnostického znalostního systému
Student:	Richard Werner
Vedoucí:	Ing. Magda Friedjungová
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

Student provede rešerši v oblasti znalostních (tj. expertních) systémů, zejména diagnostických. Na základě získaných teoretických znalostí student navrhne framework pro tvorbu znalostních systémů, který bude sloužit jako podpora pro výuku předmětu Znalostní systémy na FIT ČVUT (BI-ZNS). Framework se bude skládat ze šablon tříd a grafického rozhraní. Framework umožní implementaci jednotlivých slotů jako je reprezentace znalostí, inferenční mechanismus, neurčitost a vysvětlovací modul. Student návrh implementuje v jazyce Java a demonstruje jeho použití na vhodném problému.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Karel Klouda, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 19. února 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Framework pro tvorbu diagnostického znalostního systému

Richard Werner

Katedra aplikované matematiky
Vedoucí práce: Ing. Magda Friedjungová

14. května 2018

Poděkování

Chtěl bych poděkovat své vedoucí Ing. Magdě Friedjungové za trpělivost a cenné rady v průběhu vedení této práce. Také děkuji Ing. Zdeňku Balákovi za konzultace a rady týkající se zejména implementace.

Dále děkuji své rodině a přátelům za veškerou podporu během mého bakalářského studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. května 2018 .....

Odkaz na tuto práci

Werner, Richard. *Framework pro tvorbu diagnostického znalostního systému.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce řeší návrh a implementaci frameworku pro tvorbu znalostních diagnostických systémů. Pro framework je použit Java projekt s knihovnami JavaFX pro GUI. Vytvořené řešení poskytuje rozhraní pro znalostní systém, ve kterém lze jednotlivé moduly vytvořit nezávisle na sobě. Je tak zajištěno správné fungování systému s předem nespecifikovanou implementací. Pro většinu modulů jsou vypracovány ukázkové implementace, které je možné v systému využít, nebo se jimi inspirovat ve vlastní tvorbě. Účelem této práce je zkvalitnění výuky studentům bakalářského předmětu Znalostní systémy na FIT ČVUT a zjednodušení průběhu výuky samotným vyučujícím.

Klíčová slova framework, znalostní systém, Java, výuka, BI-ZNS

Abstract

This thesis deals with the design and implementation of a framework for creating knowledge-based diagnostic systems. The framework is designed as a Java project with the JavaFX libraries for the GUI. The solution provides an interface for a knowledge-based system in which the individual modules can be implemented independently on each other. This ensures the proper functioning of the system with a previously unspecified implementation. There are implementations for most of the modules, which may be used in the system or serve as an inspiration. The goal of this thesis is to improve the way students are taught at BI-ZNS lectures at FIT CTU in Prague and to simplify the teaching process for the teachers themselves.

Keywords framework, knowledge-based system, Java, lectures, BI-ZNS

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Znalostní systémy jsou takové systémy, které poskytují rady, rozhodnutí a doporučují řešení v konkrétních situacích. Jejich způsob výpočtu a podávání výsledků je oproti běžným programům velice konstruktivní a problémy řeší z několika úhlů pohledu experta. V ideálním případě je znalostní systém schopný své kroky k vyvození právě takového výsledku vypsat a odůvodnit.

Výsledek práce je určen studentům FIT ČVUT v Praze jako učební pomůcka na cvičeních předmětu Znalostní systémy. Slouží jako šablona pro vytváření znalostních (tj. expertních) systémů a je možné si v ní vyzkoušet tvorbu základních stavebních kamenů, ze kterých se znalostní systém skládá.

Jelikož jsem tímto předmětem sám prošel, motivací pro výběr tohoto tématu bylo mít možnost sjednotit a zefektivnit způsob, jakým se studenti učí o znalostních systémech. Práce pomůže nejen studentům, ale i samotným vyučujícím, kterým zjednodušší průběh celého semestru.

V této práci se tedy zabývám návrhem a implementací frameworku a GUI pro tvorbu znalostních systémů, pro které využívám jazyk Java. Ten je vhodný pro účely výuky studentů FIT ČVUT a nebude příliš náročné navázat na tento framework studentovou vlastní prací na cvičeních.

Práce se skládá ze dvou částí. První je teoretická část, která čtenáře seznámí s problematikou expertních systémů a to od obecných informací, přes rozebrání jednotlivých částí expertního systému, až po aplikaci a tvorbu. Druhá je praktická část, která se zabývá návrhem, implementací a testováním samotného frameworku. V práci budu pro pojem „znalostní systém“ využívat zkratku ES (tzn. expertní systém), která je v běžné praxi zakotvená.

Cíl práce

Cílem práce je návrh frameworku pro výuku předmětu BI-ZNS na FIT ČVUT. Ten bude sloužit jako šablona pro tvorbu expertního diagnostického systému. Po dohodě s vedoucím práce se bude framework skládat ze šablon tříd, které

umožní implementaci základních součástí expertního systému, jako jsou: báze znalostí, inferenční mechanismus, neurčitost, uživatelské rozhraní a vysvělovací činnost.

Podmínkou je, aby byl framework pochopitelný. Nesmí být příliš náročné pro studenty tuto šablonu uchopit a implementovat potřebné součásti. Z tohoto důvodu bude celá práce v jazyce Java, který je pro studenty dobře srozumitelný, protože mají ve stejném semestru předmět zaměřený na programování v Javě. Tento jazyk patří do seznamu dovedností absolventa oboru Znalostní inženýrství, pro který je předmět BI-ZNS povinný.

Dílčím cílem je veškerý zdrojový kód řádně zdokumentovat a poskytnout tak budoucím uživatelům tohoto frameworku dostatečnou oporu při využívání.

Část I

Teoretická část

Kapitola 1

Znalostní systémy

Expertní systémy jsou počítačové programy zpracovávající a interpretující informace od experta, které podávají konstruktivní výsledky na základě těchto znalostí a poznatků o dané problematice. Patří mezi jedny z nejrozšířenějších aplikací umělé inteligence. Jejich činnost spočívá v analýze, návrhu a diagnóze nejrůznějších problémů, k jejichž vyřešení jsou nutné rozsáhlé znalosti z mnoha oborů. Rozdíl oproti běžným počítačovým programům tedy spočívá v projevování právě těchto znalostí a vyhodnocování výsledků na základě mnoha různých faktorů a kritérií. V současnosti jsou používány v mnoha oblastech lidské činnosti, jako například: v technice, ve vědě, výrobě, v obchodu apod. [1, 2]

1.1 Charakteristika ES

Charakteristika ES se vyznačuje následujícími body [3]:

Oddělení BZ a IM Znalosti experta jsou uloženy v bázi znalostí odděleně od inferenčního mechanismu, aby bylo možné tento inferenční mechanismus nasadit na jinou, libovolnou bázi znalostí. Umožňuje to také tvořit tzv. *prázdné expertní systémy* (*expert system shells*), u kterých není předem dáno, s jakými informacemi bude inferenční mechanismus pracovat.

Neurčitost v bázi znalostí V bázi znalostí jsou zachyceny nejen přesné a jisté informace, ale také nejrůznější heuristiky, které expert při své práci sám využívá. V bázi se pak objevují výrazy, jako například „většinou“ nebo „často“, které je nutné nějakým způsobem kvantifikovat.

Neurčitost v datech Data o konkrétním případu mohou být zatížena neurčitostí, kterou mohou způsobit nepřesná měření, nejisté výsledky nebo subjektivní pohled (například odpověď „nejspíš ano“ na otázku, zda má pacient teplotu).

Dialogový režim Nejčastěji jsou expertní systémy konstruovány jako tzv. „konzultační systémy“. Probíhá dialog mezi uživatelem a počítačem způsobem „dotaz systému – odpověď uživatele“ podobně, jako s lidským expertem.

Vysvětlovací činnost V rámci zvýšení důvěry v rozhodovací činnost expertního systému by měl systém poskytovat vysvětlení svého uvažování. Vysvětlována je nejčastěji právě položená otázka, znalost relevatní k nějakému tvrzení nebo právě zkoumaná hypotéza.

Modularita a transparentnost BZ Pro účinnost expertního systému je nutná vysoká kvalita báze znalostí. Modularita umožňuje snadnou úpravu a aktualizaci báze znalostí a transparentnost její snadnou čitelnost a srozumitelnost. Samotná tvorba báze znalostí probíhá iterativně při konzultacích s expertem z dané oblasti problematiky, dokud chování expertního systému neodpovídá představám experta.

1.2 Klasifikace a aplikace typických ES

Pro následující podkapitolu bylo čerpáno z [2].

Rozmanité požadavky na řešení problémů vyžadují různé způsoby reprezentace znalostí a inference. Toto platí hlavně pokud chceme vytvořit expertní systém pomocí *nástroje* prázdný expertní systém. V ideálním případě stačí takový prázdný expertní systém „naplnit“ danými znalostmi, abychom vytvořili funkční systém. Taková tvorba ale může fungovat jen tehdy, kdy byl prázdný systém vytvořený pro stejnou třídu informací a znalostí jako ta, kterou chceme použít. Ale ani v takovém případě není garantované, že vše bude fungovat tak, jak si představujeme.

Následují definice základních typů expertních systémů, které mohou sloužit jako základní rozdělení pro prázdné ES a znalosti, které využívají:

Informativní systémy sbírají data od uživatele o daném případu pomocí dialogu. Tyto informace jsou ještě s pomocí znalostí samotného systému využity k předání speciálních informací uživateli. Systémy takového typu pokrývají rozsáhlý výčet aplikací, například: výběr nejvhodnější a nejkratší trasy při cestování, nejlepší nabídka dovolené, analýzy ekonomických indikátorů, nebo výběr mezi různými technickými procesy.

Diagnostické systémy jsou rozsáhlejší systémy, které vyžadují více (druhů) informací (ať už v bázi znalostí, nebo přímo od uživatele), a které se dají využít v medicíně či technice k diagnóze příčiny nějakého problému na základě naměřených hodnot, symptomů nebo chování a dalších relevantních informací. Rozdíl mezi *informativními* a *diagnostickými* systémy není na první pohled tak zřejmý. Základní charakteristika *diagnózy* oproti jednoduché *informaci* není jen v objemu pravidel a informací, ale

spíše v míře zastoupení neurčitosti/nejistoty. Z toho důvodu musí takové systémy obsahovat mechanismus (modul), který právě takovou nejistotu bude zpracovávat a vůbec zohledňovat.

Opravné systémy jsou rozšířením diagnostických systémů. Musí pracovat nejen se znalostmi potřebnými k detekování nějakého defektu, ale také s informacemi, jak tento defekt „opravit“. Jsou typicky využívány údržbáři, kteří mají na starost opravy složitějších systémů/mechanismů, případně pro automatizované opravy na vesmírných zařízeních, nebo v nebezpečných prostředích, jako jsou jaderné elektrárny. Běžnější využití si najdou v komerčním prostředí, například opravy automobilů, letadel, systémů dopravy, počítačů, nebo dalších mechanických zařízení.

Ladící systémy (debugging systems) jsou speciální podtřídou diagnostických a opravných systémů, které slouží k hledání a eliminaci chyb v softwarových systémech. Těchto systémů neexistuje příliš mnoho. Jedním příkladem je systém generující testy pro UNIXové implementace.

Interpretační systémy jsou využívány k analýze řeči nebo obrázků, vyhodnocování signálů a monitorování meřících technik. Interpretují vstupní data jako symbolický význam pro danou situaci/vstup. Pokud do této skupiny expertních sytémů budeme počítat systémy pro překlady přirozeného jazyka, bude tato skupina nejspíš tou nejstarší. Navíc jsou takové aplikace jedny z nejproblematičtějších, protože mají obrovské nároky na znalosti samotného systému.

Prediktivní systémy se snaží předvídat vývoj v (blízké) budoucnoti na základě dat z přítomných a minulých okolností a událostí. Prognóza může být založena na matematických, statistických, nebo „experimentálních“ pravidlech. Aplikaci si pak takové systémy najdou například v předpovědi počasí, sklizně, při ekonomických odhadech a také při předpovědi vývoje dopravy, spotřeby energií atd. Tyto systémy jsou pak specifické simulovaným, parametrickým a dynamickým modelem systému, v jakém mají předvídat daný vývoj. Parametry jsou dány právě zkoumaným případem. Síla (nebo slabost) takového systému je dána právě v modelu, který používá, protože „reálný svět“ je příliš komplexní na to, aby z něho šel vytvořit model. Musí být vždy do určité míry abstraktní, aby bylo možné tento model implementovat. Právě tato abstrakce je pak důvodem nepřesnosti systému.

Plánovací systémy se používají při plánování složitých úkonů, zejména ve vojenském a ekonomickém sektoru. Jsou pak také využívány ve vědě při plánování experimentů. Příkladem vědeckého využití je expertní systém *Molgen*, který se používá k přípravě molekulárních genetických experimentů.

Konfigurační systémy jsou speciálním případem plánovacích systémů. Umožňují technickou konstrukci a konfiguraci výrobků (např. výpočetní technika a instalace) z modulárních komponent, nebo uspořádání obvodů na desce plošných spojů. Tyto systémy jsou charakteristické pravidly v bázi znalostí, které udávají hranice a omezení možných výsledků. Důležitou částí jsou pak pravidla umožňující optimizaci v určitém směru, například minimalizace výrobních nákladů, času potřebného k výrobě, nebo rozměrů výsledného výrobku.

Monitorovací systémy porovnávají data od uživatele nebo z měřících technik s referenčními hodnotami a podávají informace o stavu daného procesu, o případných výkyvech oproti běžným hodnotám a v některých systémech také varování před nebezpečným vývojem. Podtřídou monitorovacích systémů jsou tzv. systémy *asistenční*, které sledují vstup nějakého běžného data-zpracujícího procesu (například textového procesoru) a v případě „potřeby“ dávají uživateli „inteligentní“ rady. Příkladem je kontrola pravopisu nebo napovídání slov.

Kontrolní systémy jsou rozšířením monitorovacích systémů. Tvoří uzavřený kontrolní cyklus, který automaticky provádí opravné akce v případě, že detekuje neoptimální stav.

Školící systémy jsou implementovatelné nad jakýmkoliv z výše zmíněných typů systémů. Jejich úkolem je simulovat uživateli (například studentovi nebo budoucímu expertovi v oboru) konkrétní problém založený na konkrétních datech a poté asistovat tomuto uživateli při řešení daného problému. V případě potřeby může být uživatel informován o případných dopadech jeho rozhodnutí. Typickým školícím systémem je systém „STEAMER“, vyvinutý BBN pro americké námořnictvo pro výcvik inženýrů v provozu lodních motorů. Takový systém si i ukládá současný stav znalostí jednotlivých uživatelů a podle toho jim poskytuje vhodné ukázky a vysvětlení v průběhu výuky.

Dále se ve své práci věnuji zejména diagnostickým systémům, na které jsou cvičení předmětu BI-ZNS zaměřena.

1.3 Struktura ES

V následující sekci budou probrány základní součásti ES a jejich funkce v rámci systému.

1.3.1 Báze znalostí

Veškeré znalosti experta o dané problematice jsou uloženy v bázi znalostí. Jsou v ní informace od velmi obecných, po úzce odborné. Speciální kategorií

jsou tzv. *soukromé znalosti* (také označovány jako heuristiky, či nejisté znalosti). Jde o exaktně nedokázané znalosti, které expert získává během jeho praxe. Tyto znalosti pomáhají řešit určité problémy, ale nezajišťují nalezení správného řešení. Právě tímto druhem znalostí se odlišují znalosti experta v oboru od běžného pracovníka (laika).

Nejnovější expertní systémy nepracují pouze s jednou bází znalostí, ale pracují s několika bázemi najedou – s tzv. *zdroji znalostí* [4]. Závěry z jednotlivých bází jsou zapisovány na společnou „tabuli“, tedy sdílenou datovou strukturu. Tato činnost je založena na reálné představě, že více expertů s různou specializací pracuje dohromady na jednom problému a každý z nich řeší problémy blízké jejich specializaci.

Vzhledem ke stále se vyvíjejícím znalostem expertů je nutné, aby byla báze znalostí dostatečně modulární, tedy aby ji bylo možné kdykoliv aktualizovat nebo doplnit. ES musí být také schopen využívat nejisté znalosti, kterým jsou v bázi znalostí přiřazeny různé váhy, stupně, nebo míry důvěry stejně tak, jako musí zvládat zpracovávat nejisté odpovědi uživatele (tedy nejisté záznamy v bázi dat).[4]

1.3.2 Reprezentace znalostí

Při reprezentaci znalostí je nejdůležitější vlastnost efektivní reprezentace. Požadavky na reprezentaci jsou [5]:

- dostatečně přirozená reprezentace pro danou oblast problematiky a její expresivita (způsoby vyjádření),
- umožnění aplikace efektivních deduktivních prostředků,
- rychlý přístup k bázi znalostí a k bázi dat.

1.3.2.1 Predikátová logika

Predikátová logika je velice dobře prozkoumaný systém pro reprezentaci a zpracování znalostí. Původně byla zkoumána pro potřeby matematiky, ale pro její univerzálnost se uplatnila v reprezentaci znalostí. Je vhodná pro studium odvoditelnosti (inference) mezi tvrzeními a z těchto důvodů slouží predikátová logika prvního řádu jako základ pro symbolické programovací jazyky (např. Prolog) [5].

Jazyk predikátové logiky obsahuje [3]:

Individuové proměnné: skupiny, do kterých spadají funkční symboly a konstanty.

Predikátové symboly: symboly, vymezující vztahy mezi jednotlivými individui.

Kvantifikátory: nejvýznamnější kvantifikátory jsou $\forall$ („pro všechny“) a $\exists$ („existuje alespoň jeden“).

Logické spojky: konjunkce, disjunkce, negace, implikace a ekvivalence.

Funkční symboly a konstanty: symboly, odkazující ke konkrétním individuím a sloužící k tvorbě jejich jmen.

Konkrétní jazyk pak obsahuje vždy alespoň jeden predikátový symbol. Konstanty s funkčními symboly nejsou povinné, jsou však dobrou pomůckou ke zlepšení srozumitelnosti a efektivitě inference. [5]

Příkladem výroku v predikátové logice může být například:

$$\forall x : (B(x) \wedge Z(x)) \implies R(x),$$

kde x je proměnná ovoce, predikátové symboly B, Z a R označují popořadě „je banán“, „je žlutý“ a „je zralý“. V překladu do přirozeného jazyka by pak tento výrok znamenal: „*Pro každé ovoce platí, že když je banánem a současně je žluté, pak je zralé.*“

1.3.2.2 Pravidla

Pravidla v současné době představují nejčastější způsob reprezentace znalostí. Ještě spolu s predikátovou logikou je tato reprezentace používána v *pravidlových* expertních systémech. Každé pravidlo má svoji *podmínkovou* a *důsledkovou* část. Pravidlem chápeme výraz typu

$$IF\ a\ THEN\ b,$$

kde část pravidla za IF se nazývá *antecedent* a část pravidla za $THEN$ se nazývá *konsekvent*. Levá strana pravidla je složena z individuálních podmínek a představuje tedy podmínkovou část pravidla. Pravá strana potom může obsahovat akce či závěry a představuje tedy důsledkovou část pravidla. Součástí zápisu pravidla může také být kontextová část, která vyjadřuje za jakých podmínek může být pravidlo použito. Pravidlo $IF - THEN$ je často zapisováno ve tvaru $E \rightarrow H$, kde E je předpoklad (evidence) a H je závěr (hypotéza). [6]

Pravidla bývají interpretovaná ve více podobách, z nichž nejčastější jsou následující:

- IF *situace* $THEN$ *akce*;
- IF *předpoklad* $THEN$ *závěr*;
- IF *podmínka* $THEN$ *důsledek A* $ELSE$ *důsledek B*.

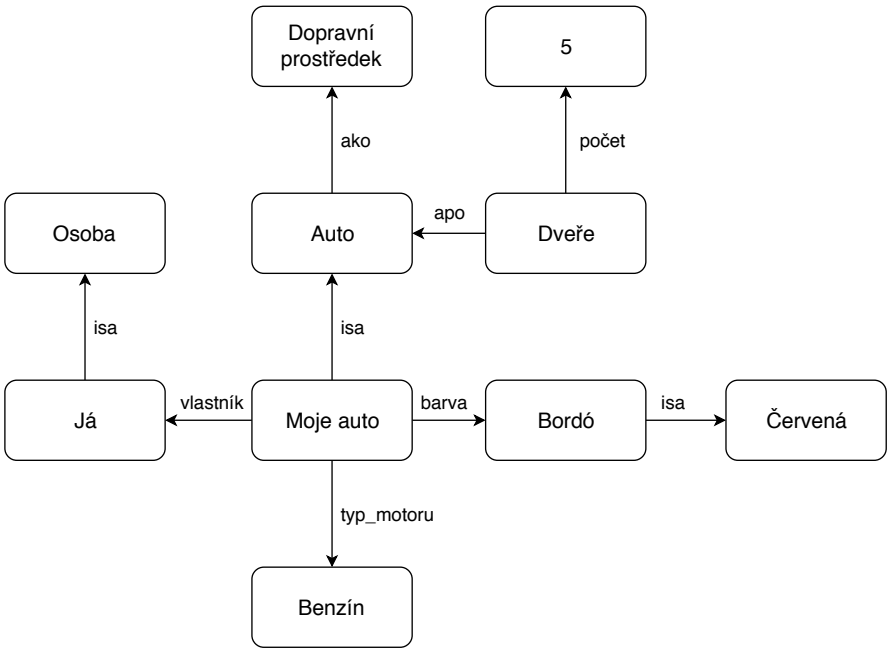
V prvním případě je pravidlo interpretováno *procedurálně*, v druhém pak *deklarativně*. Procedurální způsob chápání pravidla (jestliže nastane určitá situace, systém provede akci) je typický spíše pro plánovací expertní systémy. Naopak deklarativní způsob (jestliže je splněn určitý předpoklad, systém učiní příslušný závěr) odpovídá spíše diagnostickým systémům. Ve třetím případě došlo k rozšíření pravidla o $ELSE$ část. [6]

1.3.2.3 Sémantické sítě

„Sémantické sítě byly vyvinuty koncem 60. let. V roce 1968 použil tento pojem ve své disertační práci americký vědec Quillian k reprezentaci anglických slov. Sémantické sítě původně sloužily k vyjádření významu různých výrazů v přirozeném jazyce, postupem času se však staly obecnějším grafickým nástrojem pro reprezentaci znalostí.“([6], Reprezentace znalostí; Sémantické sítě)

Na sémantickou síť lze pohlížet jako na *orientovaný graf* tvořený *uzly* a *hranami*, kde uzly reprezentují jednotlivé objekty a hrany pak vztahy (relace) mezi těmito objekty. Velmi časté vztahy jsou $IS\text{-}AN\text{-}INSTANCE\text{-}OF$ (ISA), $A\text{-}KIND\text{-}OF$ (AKO), nebo $A\text{-}PART\text{-}OF$ (APO). Vztah ISA slouží k vyjádření, že konkrétní objekt je instancí určité třídy. Vztah AKO vyjadřuje, že daná třída je podtřídou jiné třídy. Poslední vztah APO říká, že určitá třída objektů je složena z více částí. [3]

Na obrázku 1.1 je ukázka, jak by mohla vypadat jednoduchá sémantická síť, kde jsou použity všechny tři druhy relací zmíněné výše.



Obrázek 1.1: Ukázka jednoduché sémantické sítě

1.3.3 Báze dat

Při řešení konkrétního problému je potřeba do systému ukládat zjištěná data. Tato data se ukládají do báze dat, tedy do množiny údajů k danému případu, která se pak dosazuje do báze znalostí k vyhodnocování výsledku. Konkrétní data jsou získávána v dialogovém režimu s uživatelem. Systém se zde pokouší dotazovat se co nejefektivněji, analyzovat odpovědi uživatele a na základě těchto informací zkonstruovat závěr, nebo navrhnout řešení. [4]

1.3.4 Inferenční mechanismus

Inferenční mechanismus je programový modul, který předem udává, jakým způsobem bude využívat data z báze znalostí a jakým způsobem bude zprostředkovávat komunikaci mezi bází znalostí a bází dat.

Podle [7, 3] jsou typy inference, ze kterých si většina z nich našla uplatnění v expertních systémech, tyto:

Dedukce: logické usuzování, při kterém musí závěr plynout z předpokladů (modus ponens, modus tollens).

Indukce: zobecnění konkrétních případů.

Abdukce: usuzování ze závěru k předpokladům, které mohly tento závěr splnit.

Heuristiky: pravidla nebo odhady, které jsou založeny na zkušenosti.

Analogie: odvození závěru na základě podobnosti s jinou situací/problémem.

Defaultní inference: při absenci speciálních znalostí se usuzuje z obecných znalostí.

Nemonotonní: předcházející znalosti se mohou upravovat na základě nově získaných poznatků.

Generování a testování: generují se výsledky a testuje se, zda sedí na danou situaci (metoda pokusu a omylu).

A	B	$A \implies B$
0	0	1
0	1	1
1	0	0
1	1	1

Tabulka 1.1: Implikace [8]

První tři typy (dedukce, indukce a abdukce) vycházejí z pojmu implikace z výrokové logiky (tabulka 1.1). Zde bych rád definoval, že v kontextu logiky znamená číslice „0“ nepravdu a číslice „1“ pravdu.

U *dedukce* předpokládáme, že platí implikace (tedy pravidlo) a předpoklad, z čehož se dá jednoduše odvodit, že platí závěr (modus ponens). Analogicky při předpokladu, že platí implikace a neplatí závěr, můžeme jednoznačně odvodit nepravdivost předpokladu (modus tollens – obě pravidla jsou znázorněna v tabulce 1.2). Dedukce je tedy způsob usuzování, který zachovává pravdu (truth preserving reasoning).

$A \implies B$	$A \implies B$
A	B
A	$\neg B$
modus ponens	modus tollens

Tabulka 1.2: Dedukce [8]

Při *abdukci* předpokládáme pravdivost implikace a závěru. Z tabulky 1.1 je vidět, že předpoklad může a nemusí platit, takže se lze jen domnívat, že může platit. Někdy je abdukce označována za nejlepší vysvětlení v rámci pozorovaných faktů. Abdukce tedy zachovává nepravdu (falsity preserving reasoning). Když budeme předpokládat, že platí implikace a neplatí závěr, můžeme jednoznačně říct, že neplatí předpoklad.

Indukce z opakovaného současného výskytu A a B vyvozuje, že je mezi touto dvojicí implikace. Je to metoda využívaná pro strojové učení a automatizované získávání znalostí z dat.

Heuristika je označení pro druh informací používaných v expertních systémech. Jsou založené na zkušenostech experta a na zobecnění situací, ve kterých se expert musel rozhodovat.

Analogie se používá například při *případovém usuzování* (Case-Based Reasoning). Znalosti nemají podobu obecných pravidel získaných od experta, ale jsou tvořeny souborem dříve vyřešených (typických) případů. Má-li systém poskytnout doporučení, projde svoji knihovnu dříve řešených případů a hledá ten nejpodobnější tomu aktuálnímu. Výhodou je snadnější vývoj aplikace, jelikož není nutné náročné získávání pravidel od experta, ale stačí „pouze“ dostatek reprezentativních případů.

Usuzování za použití *defaultů* (default reasoning) bývá spíše doplněním usuzování na základě pravidel. Využívá se, když není pro danou situaci využitelné žádné pravidlo. Například v Česku se dá předpokládat, že chřipka bude mnohem častější onemocnění, než žlutá zimnice. Tedy bez jakýchkoliv otázek na pacienta je možné předpokládat, že má spíše chřipku, než žlutou zimnici a pokud nám nezodpoví žádné otázky, pošleme ho s diagnózou chřipky domů.

Nemonotónní usuzování je založeno na skutečnosti, že předchozí získané informace je možné „zahodit“ pro nově získané znalosti.

Při *generování a testování* se opakovaně generuje nové „náhodné“ řešení a testuje se, zda vyhovuje všem předpokladům. Pokud se najde takové, které vyhovuje, cyklus končí a systém předloží vygenerované řešení.

1.3.4.1 Přímé řetězení

U tzv. „odvozování řízeném daty“ se začíná známými fakty (daty), od kterých se postupuje přes aplikovatelná pravidla na danou situaci, až k závěrům. Při přímém řetězení se tedy postupuje bází znalostí od předpokladů pravidel k jejich závěrům. Tento způsob odvozování je vhodný pro plánovací typy expertních úloh. [6]

Podle [9] spočívá přímé řetězení v opakování následujících kroků:

Porovnání (matching): Pravidla z báze znalostí jsou porovnána se známými fakty, aby se zjistilo, která pravidla mají splněné předpoklady.

Řešení konfliktu (conflict resolution): Z množiny pravidel se splněnými předpoklady se vybírá pravidlo podle priority a v případě konfliktu podle nějaké strategie. Příklady strategie pro řešení konfliktů:

- Strategie hledání do hloubky (depth search), kde jsou preferována pravidla, používající aktuálnější data – tedy data, která se v bázi znalostí vyskytují kratší dobu.
- Strategie hledání do šířky (breath strategy), při níž jsou naopak preferována pravidla používající starší data.
- Strategie složitosti, resp. specifičnosti (complexity strategy), kde jsou preferována pravidla mající více podmínek.
- Strategie jednoduchosti, která zase naopak preferuje jednodušší pravidla – tedy ta s méně podmínkami.

Provedení (execution): Provede se pravidlo vybrané v předchozím kroku. Důsledkem provedení může být přidání pravidla do báze dat, odebrání pravidla z báze dat, nebo třeba přídání pravidla do báze znalostí atp. Obvykle je při tomto postupu uplatňována podmínka, že každé pravidlo může být „aktivováno“ pouze jednou pro stejnou množinu dat (faktů).

Vhodnými aplikacemi pro dopředné (přímé) řetězení jsou:

- Monitorování a diagnostika řídících systémů pro řízení procesů v reálném čase, kde jsou data kontinuálně přidávána a měněna a kde existuje málo dopředu daných vztahů mezi vstupními daty a závěry.

- Problémy zahrnující syntézu (navrhování, konfigurace, plánování, rozvrhování, ...). V těchto aplikacích existuje mnoho možných řešení a pravidla proto vyjadřují znalosti jako obecné vzory. Přesné vztahy (*inferenční řetězce*) proto nemohou být předem přesně určeny a musejí být použity systémy pro porovnávání se vzorem.

1.3.4.2 Zpětné řetězení

Inference řízená cíli (tedy zpětné řetězení) je založena na hledání informací podporujících určitý závěr. Při tomto druhu inference se začíná u cíle, který chceme odvodit, tudíž je báze znalostí procházena směrem zpětně od cílů k předpokladům. Pořadí vyhodnocování může být zase stejně jako u přímého řetězení dáno prioritami nebo pořadím v bázi znalostí. Zpětné řetězení je používáno spíše u diagnostických typů úloh, které mají malý počet cílových hypotéz. [6]

Dle [9] dochází při zpětném řetězení k opakování následujících kroků:

1. Vytvoření a naplnění zásobníku všemi koncovými cíli.

2. Shromáždění všech pravidel schopných splnit cíl na vrcholu zásobníku. Pokud je zásobník prázdný, inference je ukončena.

3. Zkoumání pravidel a platnosti jejich předpokladů z předchozího kroku.

   a) Pokud jsou splněny všechny předpoklady, odvodí se závěr (použije se pravidlo) a samotné pravidlo se dočasně odloží. Jestliže zkoumaný cíl byl koncový, pak je odstraněn ze zásobníku a vracíme se ke kroku 2. Pokud to byl pouze podcíl, je odstraněn ze zásobníku a je zpracováváno předchozí pravidlo, které bylo odloženo.

   b) Jestliže fakta nalezená v bázi dat (faktů) nesplňují předpoklady pravidla, je zkoumání pravidla ukončeno.

   c) Pokud pro některý předpoklad chybí informace v bázi dat, zjišťuje se, zda se tato informace nedá z nějakého pravidla odvodit. Pokud ano, vloží se do zásobníku jako podcíl, zkoumané pravidlo se dočasně odloží a pokračuje se krokem 2. V opačném případě se informace zjistí od uživatele a pokračuje se krokem 3 a) zkoumáním dalšího předpokladu.

4. Jestliže pomocí žádného ze zkoumaných pravidel nebylo možné odvodit cíl, pak tento cíl zůstává neurčen. Je odstraněn ze zásobníku a pokračuje se krokem 2.

1.3.5 Vysvětlovací modul

Máme-li možnost s expertem při řešení problému komunikovat, očekáváme od něho, že své rozhodnutí a postup bude schopen zdůvodnit a vysvětlit. Funkce vysvětlovacího modulu má tedy za úkol vysvětlit rozhodnutí expertního systému a průběh tohoto rozhodování na místo člověka. Tento modul není podmínkou každého expertního systému, někdy je dokonce nežádoucí.[10]

Podle [10] by v tom nejjednodušším provedení měl být vysvětlovací modul schopen odpovídat na dva následující typy dotazů:

- Proč
  - požaduje danou informaci od uživatele?
  - volí daný způsob odvozování?
  - odvozuje daný fakt?
- Jak
  - dospěl k odvození určitého faktu?
  - dospěl k určitému způsobu odvozování?
  - dospěl k určitému výsledku?

V případě typu otázky „Proč?“ bude vysvětlovací modul vysvětlovat, co se děje, nebo co se předpokládá, že se bude dít v následujícím procesu inference.

V případě typu otázky „Jak?“ bude vysvětlovací modul vysvětlovat, jak dospěl k určitému závěru.

1.4 Tvorba ES

Tvorbou expertních systémů se zabývá vědní obor *znalostní inženýrství*. Jeho základním úkolem je získání znalostí potřebných k řešení problémů z dané oblasti, což zahrnuje nejen vytvoření metod pro získávání a modelování znalostí, ale také nalezení vhodného inferenčního mechanismu. Protože jsou tyto znalosti často špatně definovány a navíc zatíženy neurčitostí, je vývoj ES doprovázen četnými potížemi.

Tvorba ES je složitý proces dělící se do několika vývojových fází. Jedním z prvních a nejtěžších rozhodnutí je, zda se vůbec do velice časově (a finančně) náročného vývoje pustit. [6]

Podle [2] by se každý návrh softwaru s umělou inteligencí měl řídit následujícími metodologickými kroky:

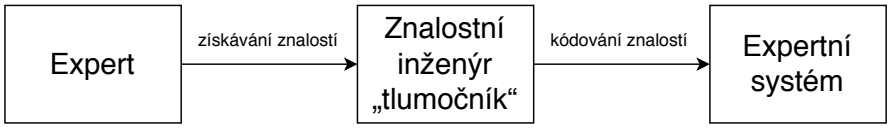
1. identifikace problému,
2. výběr nebo implementace vhodné reprezentace (znalostí),

3. snaha o nalezení přirozených omezení, zákonů a heuristik, pomocí kterých se budou řešit dané problémy,

4. formulace procedur řešení problémů,

5. ověření a testování daného řešení.

Celý proces tvorby ES vyžaduje spolupracující tým odborníků a velké množství času. Procesu získávání znalostí a tvorby báze znalostí se účastní hlavně experti ze zvolené oblasti problematiky, znalostní inženýři a koncoví uživatelé. Významné postavení zde mají právě znalostní inženýři. Jejich prvním úkolem je posoudit účelnost a vhodnost ES ještě před začátkem vlastní tvorby. Musí se tedy jednat o osobu s analytickými a komunikačními schopnostmi, s velkým nadhledem a rozvinutou schopností proniknout do podstaty věci. Práce znalostního inženýra může připomínat práci tlumočníka. Stejně jako tlumočník umožňuje komunikaci mezi dvěma stranami, umožňuje znalostní inženýr komunikaci mezi expertem a bází znalostí. Také je zapojen do procesu získávání znalostí, při kterém pomáhá expertovi strukturovat znalosti a následně je zodpovědný za kódování znalostí do báze znalostí (proces znázorněn na obrázku 1.2). Poté musí také ověřit, zda systém simuluje převzaté znalosti správně. [6]



Obrázek 1.2: Znalostní inženýr jako „tlumočník“ [6]

1.5 Rešerše blízkých řešení

Vzhledem k výukové povaze této práce, a tím pádem specifičtějším nárokům na provedení, není možné žádné nasazení tohoto frameworku do jakékoliv jiné praxe, než výuky na FIT ČVUT, nebo jiné škole. Z tohoto důvodu je porovnání se současným řešením problému obtížné.

Nejblíž tomuto problému jsou tzv. „prázdné expertní systémy“ (například o nich mluví [3], 3. část skript), které nemají předem určené, jakou problémovou oblast znalostí budou řešit. Mají pouze implementovaný způsob, jak informace z báze znalostí zíkávají (tedy inferenční mechanismus), ale tuto bázi znalostí mají prázdnou. Využitelné pro výuku by tedy byly pouze v naplnění báze znalostí, ale student by si nevyzkoušel implementaci žádné jiné části. Příkladem prázdného expertního systému je FEL Expert [11], který se používá pro výuku na škole FEL ČVUT, ale ten jen dovoluje plnění báze znalostí pravidly a pouze

do určité míry úpravu inferenčního mechanismu. To je pro účely této práce nedostatečné.

Z analýzy vyplývá, že nelze navázat na dostupnou práci a je nutné navrhnout vlastní způsob řešení.

Část II

Praktická část

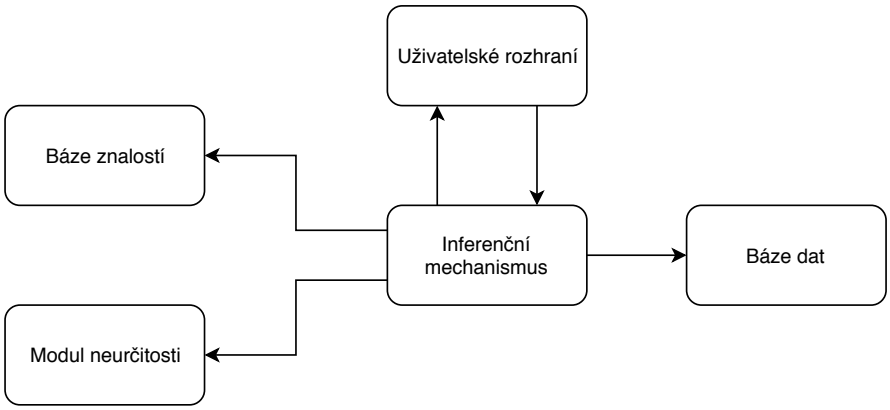
Kapitola 2

Návrh

Celý framework je navržen jako pět navzájem komunikujících modulů. Pro každý z modulů je připraveno rozhraní (interface nebo abstraktní třída), které určuje základní vstupy a výstupy každého z nich. Díky těmto rozhraním je zajištěna modularita celého systému a tento systém bude tudíž fungovat správně se kterýmikoliv implementacemi jednotlivých rozhraní.

Ve frameworku jsou připraveny rozhraní následujících modulů:

- Báze znalostí (BZ)
- Báze dat
- Inferenční mechanismus (IM)
- Modul neurčitosti
- Uživatelské rozhraní (UI)



Obrázek 2.1: Komunikace modulů

Na obrázku 2.1 je znázorněna komunikace mezi moduly. Inferenční mechanismus se chová jako mozek celého systému a moduly báze znalostí, neurčitosti a báze dat využívá pouze jako datové struktury. Modul IM a UI jsou implementovány podle návrhového vzoru Observer a Observable. Komunikace a možnosti jednotlivých modulů jsou podrobněji popsány v kapitole 3.

2.1 Modul báze znalostí

Modul báze znalostí má za úkol uchovávat zakódované informace o dané problematice. Současně musí zvládat tyto informace načítat a parsovat ze souboru, protože student bude vyplňovat BZ pomocí textového souboru v přesně dané formě. Tento způsob „plnění“ BZ bude pro studenta nejpohodlnější, jelikož psát pravidla pomocí kódu přímo v projektu by při rozsáhlejší BZ bylo velice nepřehledné. Tyto soubory pro jednotlivé formy BZ jsou součástí projektu a je možné je upravovat v rámci IDE bez nutnosti otevírat externí textový editor. Formy, které báze znalostí podporuje jsou:

- predikátová logika,
- pravidla,
- sémantická síť.

Dále je nutné, aby byla BZ schopna odpovídat, zda je konkrétní výrok pravdivý, aby byl na základě toho IM schopen usuzovat. Poslední podmínkou pro fungování IM je, aby bylo možné získat seznam *predikátů* (vlastností, na které se bude IM dotazovat uživatele), *závěrů* (tedy cílů, ke kterým systém směřuje) a pro zjednodušení i seznam predikátů souvisejících s daným závěrem (tedy predikáty, které mají v kontextu daného závěru smysl – takové, které pro daný závěr platí).

2.2 Modul báze dat

Úloha báze dat spočívá v ukládání informací o daném případu, tedy v evidování odpovědí uživatele. Jedinými funkcemi tohoto modulu jsou: uložení nové odpovědi, vrácení dané odpovědi a vrácení (případně chronologicky seřazeného) seznamu všech odpovědí pro vysvětlovací činnost.

2.3 Modul inferenčního mechanismu

Jelikož je inferenční mechanismus mozkem celého systému, je v něm naimplementovaný algoritmus, který využívá všechny ostatní moduly ke své práci. Využívá tedy funkce ostatních modulů nezávisle na jejich implementaci ke

komunikaci s uživatelem a k získávání nebo ukládání informací o dané problematice. Nabízejí se dvě implementace tohoto modulu: dopředné řetězení a zpětné řetězení.

2.4 Modul neurčitosti

Stejně jako báze znalostí, i modul neurčitosti načítá data ze souboru. Nejistota ze strany experta, tedy nejistota v BZ, se určuje pro každou kombinaci predikátu a závěru zvlášť. Tudíž modul nejistoty vrací hodnotu pro každou dvojici z množiny uspořádaných dvojic (predikát, závěr).

2.5 Modul uživatelského rozhraní

Od každé implementace uživatelského rozhraní se očekává, že se bude schopna uživatele dotázat na určitou informaci (na predikát) a že bude umět nějakým způsobem vypsat výsledky ES. Dále je nutné, aby bylo možné z UI zastavit a popřípadě i spustit inferenci. Posledním požadavkem je schopnost vysvětlovací činnosti, tedy aby bylo UI schopné uživateli interpretovat vysvětlení z inferenčního mechanismu.

Kapitola 3

Implementace

Tato kapitola se zabývá konkrétními implementacemi jednotlivých rozhraní a způsoby jejich využití.

3.1 Pomocné datové struktury

V této sekci představím pomocné datové struktury, které jsou využívány většinou modulů a slouží převážně ke zpřehlednění kódu a tím i ke zjednodušení orientace v něm.

3.1.1 Element interface a jeho implementace

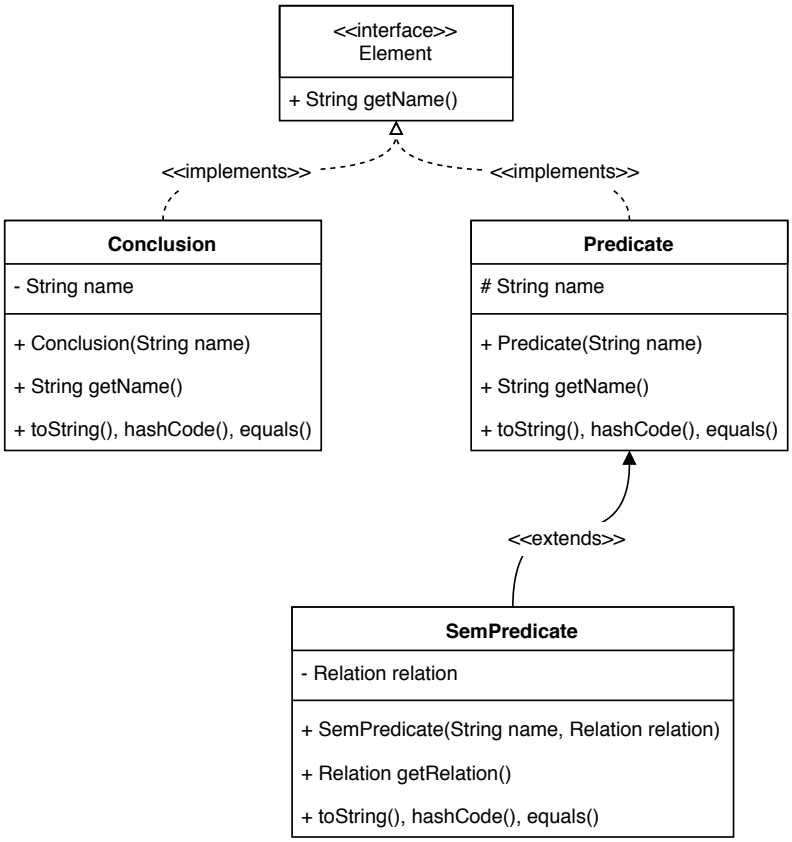
Element je interface, který implementují třídy Conclusion a Predicate (popř. SemPredicate pro sémantickou síť), které slouží k reprezentaci predikátů a závěrů skrze celý systém. Způsob dědění a implementace tříd je znázorněna na obrázku 3.1.

Element je tedy interface, který má pouze členskou proměnnou `String name`, která uchovává jméno predikátu nebo závěru.

Conclusion je první z implementací a slouží k ukládání závěrů (cílů). Je to malá struktura, která umí oproti nadtřídě vracet své jméno.

Predicate slouží k ukládání predikátů a její implementace je prakticky totožná se třídou `Conclusion`.

SemPredicate pak reprezentuje predikát pro sémantickou síť a rozšiřuje třídu `Predicate` o proměnnou typu `Relation`, kterou popisuji v sekci 3.1.2.



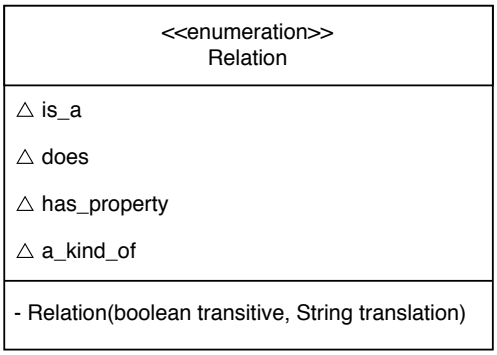
Obrázek 3.1: Element a jeho implementace

3.1.2 Relation enum

Výčtový typ Relation (viz obrázek 3.2) je pomocná „třída“ pro sémantickou síť, ale jelikož je využívána v dalších pomocných strukturách (viz sekce 3.1.1 – Element), popíši ji zde.

Tento výčtový typ udává typy relací, které může sémantická síť zpracovávat. Název každého prvku je úzce spjatý s názvem přímo v sémantické síti (ale case-insensitive – nezáleží na velikosti písmen), takže se v souboru pro bázi znalostí typu sémantická síť musí dodržet název. Dále má každý z prvků dvě vlastnosti:

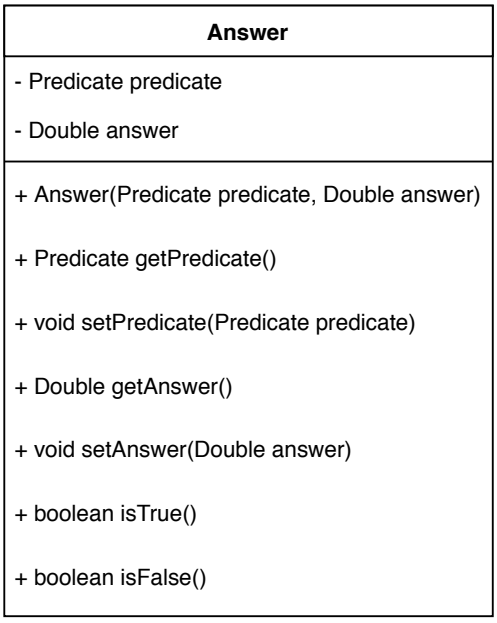
- `boolean transitive`, která udává, zda je daná relace tranzitivní, tedy jestli se přes tuto relaci (hranu v teorii grafů) přenáší vlastnosti do dalšího uzlu.
- `String translation` je překlad relace do přirozeného jazyka. Slouží při komunikaci s uživatelem tak, aby otázka, ve které tato relace figuruje, byla srozumitelná.



Obrázek 3.2: Výčtový typ Relation

Na studentovi při jeho práci bude, aby si případné další relace do této třídy doplnil podle jeho potřeb a určil jim tranzitivitu a jejich překlad. Na obrázku 3.2 jsou čtyři ukázkové prvky i s konstruktorem, který má právě dva zmíněné atributy.

3.1.3 Třída Answer



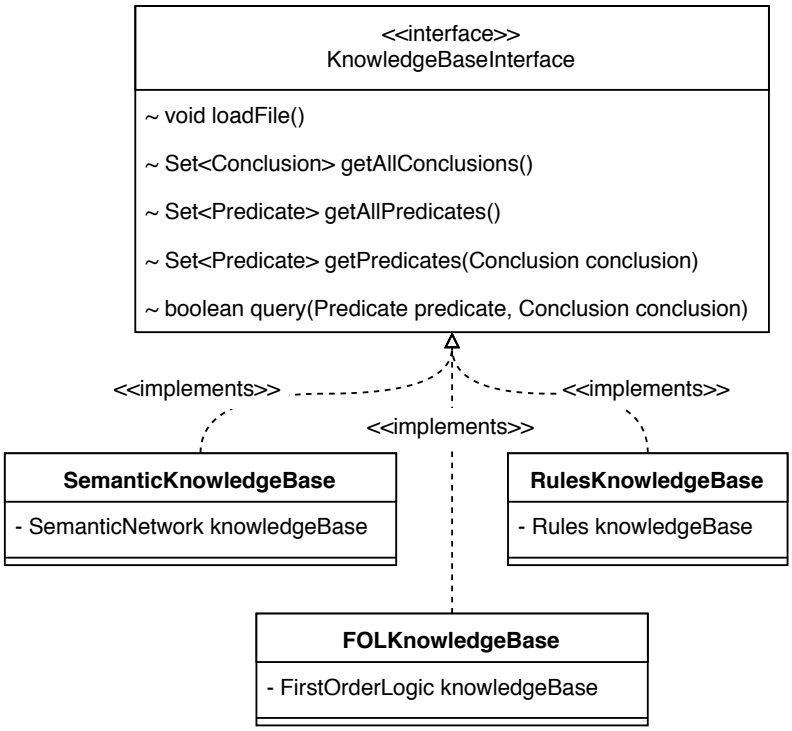
Obrázek 3.3: Třída Answer

Třída Answer (viz obrázek 3.3) je struktura ukládající odpovědi od uživatele a je využívána zejména v bázi dat. Skládá se ze dvou proměnných a to:

- `Predicate predicate`, která říká, ke kterému predikátu se uživatel vyjádřil,
- `Double answer`, která drží hodnotu jeho odpovědi (tedy desetinné číslo v rozmezí od 0 do 1, kde 0 znamená NE a 1 ANO).

Třída má metody `isTrue()` a `isFalse()`, které vrací `boolean` podle hodnoty proměnné `answer`. Pokud je hodnota proměnné 1, `isTrue()` vrací `true`, pokud je 0, `isFalse()` vrací `true`. V ostatních případech vrací obě metody `false`.

3.2 Rozhraní báze znalostí



Obrázek 3.4: Rozhraní BZ a jeho implementace

Jak již bylo naznačeno v sekci 2.1, každá z implementací BZ musí mít implementováno pět metod (viz obrázek 3.4) a to:

- `void loadFile()`, která podle typu BZ načte jeden ze souborů, který slouží k plnění BZ a podle něj vytvoří tuto bázi. Relativní cesta k souborům vůči projektu je „src; expertsystemfw; KnowledgeBase; Files“. Tato metoda je u všech implementací BZ použita v konstruktoru, takže se nemusí explicitně volat při vytváření instance.

- `Set<Conclusion> getAllConclusions()` vrací množinu všech závěrů z dané BZ ve formě struktur `Conclusion`.
- `Set<Predicate> getAllPredicates()` vrací množinu všech predikátů z dané BZ ve formě struktur `Predicate`.
- `Set<Predicate> getPredicates(Conclusion conslusion)` zase vrací množinu predikátů, tentokrát ale pouze těch, které jsou pravdivé pro zadaný závěr `conclusion`.
- `boolean query(Predicate p, Conclusion c)` vrací `boolean` podle toho, zda je zadaný predikát pravdivý pro daný závěr. Pokud ano, vrací `true`, jinak `false`.

Pro toto rozhraní jsou již ve frameworku implementovány tři typy báze znalostí a to: sémantická síť, predikátová logika a pravidla. Na studentovi při výuce bude, aby správně vyplnil soubory, ze kterých se jednotlivé BZ vytváří.

V následujících částech představím jednotlivé druhy BZ a jejich použití. Vnitřní implementace by standardně neměly být měněny, a proto bude jejich popis méně obsáhlý. Důležitými sekcemi je formát souborů, které implementace BZ využívají.

3.2.1 Sémantická síť

Sémantická síť je vnitřně reprezentována jako orientovaný graf, ve kterém uzly reprezentují predikáty a závěry. Hrany pak jednotlivé relace mezi nimi. Závěr je každý uzel, který nemá žádné vstupní hrany (je tedy *zdroj* v teorii grafů) a ostatní jsou predikáty. Třída `SemanticNetwork`, která reprezentuje orientovaný graf, využívá dvě třídy a to:

- `public class Node` pro reprezentaci uzlů a
- `public class Edge` pro reprezentaci hran.

Třída `Node` v sobě nese název uzlu a seznam hran, které z něho vycházejí. Metody této třídy pak umožňují vrátit seznam hran a také seznam uzlů, na které daný uzel „ukazuje“, tedy s jakými uzly je v relaci.

Třída `Edge` nese informaci o tom, z jakého uzlu vychází, do jakého uzlu vede a jakým typem relace je. Typ relace je určen pomocí výčtového typu `Relation`, který je popsán v sekci 3.1.2.

Součástí třídy jsou metody pro zpracovávání (parsování) souboru, ze kterého sémantická síť načítá uzly a hrany. Formát tohoto souboru je popsán v následující sekci.

3.2.1.1 Vytváření báze ze souboru

Sémantická báze znalostí se vytváří ze souboru *SemanticNetworkKnowledgeBase*. Soubor se nachází v balíčku *expertsystemfw.KnowledgeBase.Files* a má předepsaný formát, který vypadá následovně:

```
#comment
edge_from, relation_name, edge_to
edge_from; relation_name; edge_to
```

a kde platí, že:

edge_from je název uzlu, ze kterého hrana vychází,

relation_name je název relace mezi uzly (sekce 3.2) a

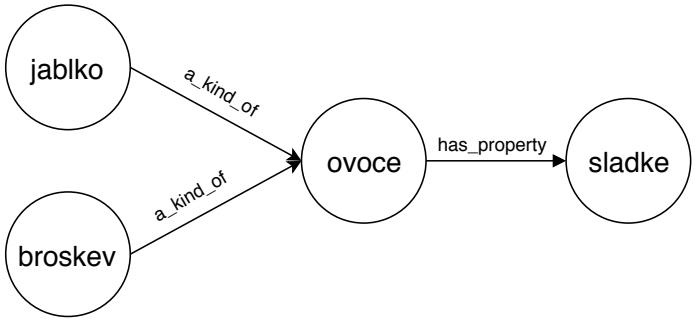
edge_to je uzel, do kterého hrana směřuje.

Jako oddělovač mezi řetězci funguje čárka (,) nebo středník (;). Mezery na začátku a na konci řádku jsou ignorovány a mezery kolem řetězců též. Prázdné řádky jsou ignorovány. Řádky začínající znakem „#“ slouží jako komentáře a jsou také ignorovány. Dále platí, že parser musí detekovat právě tři řetězce při rozdělení řádku podle jednoho z oddělovačů, tudíž jeden řádek musí vždy obsahovat právě dva stejné oddělovače.

S těmito pravidly by se z následujícího souboru:

```
#Ukazkovy soubor
jablko, a_kind_of, ovoce
broskev, a_kind_of, ovoce
ovoce, has_property, sladke
```

vytvořila sémantická síť v podobě obrázku 3.5.



Obrázek 3.5: Příklad sémantické sítě z ukázkového souboru

3.2.2 Predikátová logika

Vnitřní reprezentace predikátové logiky je založena na externích knihovnách od Tweety [12]. Pro reprezentaci jazyka slouží třída `FolBeliefSet`.

K dotazování se na bázi znalostí slouží třída `FolTheoremProver`, která využívá externí binární soubor k vyhodnocování tvrzení. Při spouštění projektu z OS Windows i z UNIX-based OS se využívá program „Eprover“ [13]. Windows varianta pak navíc ke své práci potřebuje dynamicky linkovanou knihovnu (DLL) *cygwin1.dll* [14], která je součástí projektu.

Pro tvorbu BZ ze souboru je pak využívána třída `FolParser`.

3.2.2.1 Vytváření báze ze souboru

BZ pro predikátovou logiku se vytváří ze souboru *FOLKnowledgeBase*, který se nachází v balíčku *expertsystemfw.KnowledgeBase.Files*. Jeho formát je podle BNF (Backusova-Naurova forma) následovný:

```
KB ::== SORTSDEC DECLAR FORMULAS

DECLAR ::== (PREDDEC)*

SORTSDEC ::==
( SORTNAME "=" "{" (CONSTANTNAME ("," CONSTANTNAME)*)? "}" "\n" )*

PREDDEC ::==
"type" "(" PREDICATENAME ("(" SORTNAME ("," SORTNAME)* ")")? ")" "\n"

FORMULAS ::== ( "\n" FORMULA)*

FORMULA ::== ATOM | "forall" VARIABLENAME ":" "(" FORMULA ")" |
"exists" VARIABLENAME ":" "(" FORMULA ")" |
"(" FORMULA ")" | FORMULA "&&" FORMULA |
FORMULA "||" FORMULA | "!" FORMULA | "+" | "-"

ATOM ::== PREDICATENAME ("(" TERM ("," TERM)* ")")?

TERM ::== VARIABLENAME | CONSTANTNAME
```

kde `SORTNAME` (název skupiny konstant), `PREDICATENAME` (název predikátu), `CONSTANTNAME` (název konstanty) a `VARIABLENAME` (název proměnné) jsou alfanumerické řetězce začínající písmenem a KB je startovním symbolem. Tento BNF formát souboru byl převzat z Tweety technické API dokumentace ([15], net.sf.tweety.logics.fol.parser; FolParser) a byla z něj odebrána definice funktorů, které nemají v bázi znalostí tohoto frameworku upotřebení.

Pro lepší pochopení, jak se podle výše uvedeného předpisu báze vytváří, uvedu ukázkový soubor:

```
Plody={jablko, broskev, okurka}

type (sladke(Plody))
type (zdrave(Plody))
type (barva_zelena(Plody))
type (barva_cervena(Plody))
type (ovoce(Plody))
type (zelenina(Plody))

forall Plody: (zdrave(Plody))
barva_zelena(okurka)
barva_zelena(jablko)
barva_cervena(jablko)
ovoce(broskev)
ovoce(jablko)
zelenina(okurka)
```

Soubor se dělí na tři části:

1. Definice skupin konstant, které se v dalších částech používají jako proměnné. Pro tyto skupiny platí, že jedna konstanta nemůže být ve dvou skupinách najednou, tedy skupiny musí být disjunktní.

2. Definice predikátů, které budou využívány v tvrzeních. V tomto návrhu frameworku budou správně fungovat pouze unární predikáty, takže nejdou definovat vztahy mezi jednotlivými konstantami.

3. Seznam tvrzení, která platí o jednotlivých konstantách a proměnných. Dají se využít predikáty, kvantifikátory (`forall` a `exists`), negace (`!`), konjunkce (`&&`), disjunkce (`||`), popř. tautologie (`+`) a kontradikce (`-`).

Je potřeba si uvědomit, že systém je v současném návrhu schopný se dotazovat uživatele pouze na jednoduché predikáty (s výjimkou sémantické sítě, kde je v potaz brána relace, což je ale v pojetí sém. sítě jediná možnost), tudíž nemají smysl ne-unární predikáty v bázi znalostí (binární, ternární atp.).

3.2.3 Pravidla

Vnitřní reprezentace pravidel také z části spoléhá na knihovny Tweety [12], díky kterým tentokrát ukládá jednotlivé výroky a dotazuje se na jejich pravděpodobnost. K reprezentaci formulí jsou využity dvě knihovní třídy `Formula` a `PropositionalFormula`. Pro vytvoření formule z řetězce je pak využita třída `PlParser`.

3.2.3.1 Vytváření báze ze souboru

BZ pro pravidla se vytváří ze souboru *RulesKnowledgeBase*, který se nachází v balíčku *expertsystemfw.KnowledgeBase.Files*. Formát každého řádku souboru pro tuto BZ vypadá následovně:

```
IF conclusion THEN FORMULA
```

kde na každém řádku musí být slova „IF“ a „THEN“, která musí být oddělena bílým znakem od závěru a formule. `conclusion` je název závěru a `FORMULA` je pak podle BNF:

```
FORMULA ::==
PROPOSITION | "(" FORMULA ")" | FORMULA "&&" FORMULA |
FORMULA "||" FORMULA | "!" FORMULA | "+" | "-"
```

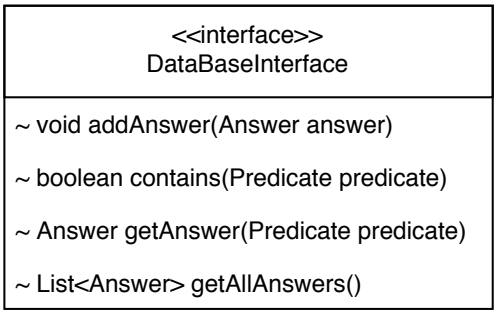
a kde `PROPOSITION` je řetězec všech znaků kromě `|,&,!,(,)` a bílých znaků, který má význam predikátu.

Jelikož je takto zadaný formát na první pohled relativně složitý, opět uvedu ukázkový soubor:

```
IF broskev THEN sladke && zdrave && ovoce
IF okurka  THEN barva_zelena && zdrave && zelenina && !barva_cervena
IF jablko  THEN (sladke || kysele) && zdrave && ovoce
```

Oproti definici pravidel (sekce 1.3.2.2) je změněné pořadí předpokladu a závěru. Pokud by předpoklad byl v první části pravidla, pak by byla náročná orientace ve dlouhém souboru, pokud by nebylo na první pohled vidět, pro jaký závěr je dané pravidlo určeno.

3.3 Rozhraní báze dat



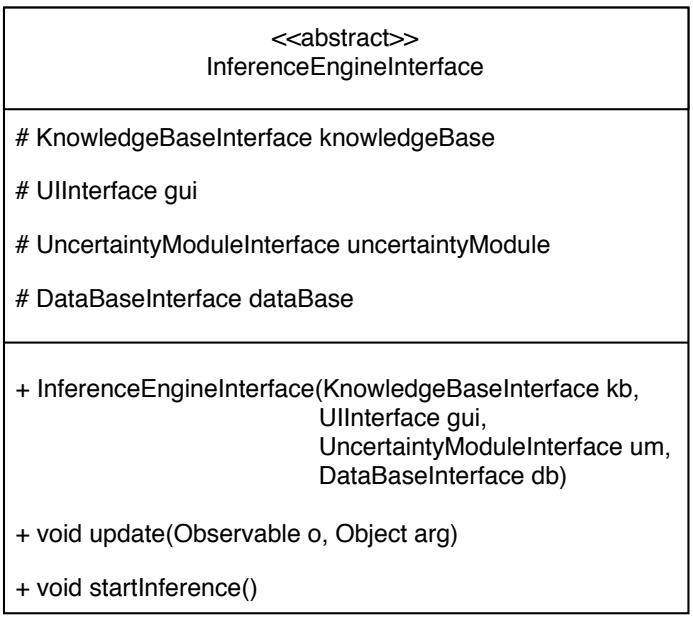
Obrázek 3.6: Rozhraní báze dat

Báze dat má jednoduché rozhraní, které pracuje převážně se třídou `Answer` (sekce 3.1.3). Podle obrázku 3.6 je potřeba implementovat čtyři metody a to:

- `void addAnswer(Answer answer)`, která přidává nový objekt `Answer`, tedy uživatelovu odpověď, do báze dat.
- `boolean contains(Predicate predicate)` vrací `boolean` podle toho, jestli báze dat obsahuje odpověď na daný predikát.
- `Answer getAnswer(Predicate predicate)` vrací objekt typu `Answer`, tedy uživatelovu odpověď na daný predikát.
- `List<Answer> getAllAnswers()` vrací (seřazený) seznam odpovědí uživatele, který je využitelný pro vysvětlovací činnost.

Očekává se, že student implementuje tento interface pomocí zvolené struktury (např. kolekce atp.) a bude při tom zachovávat nějaké řazení (např. chronologické).

3.4 Rozhraní inferenčního mechanismu



Obrázek 3.7: Rozhraní inferenčního mechanismu

Rozhraní inferenčního mechanismu (obrázek 3.7) je navrženo jako abstraktní třída implementující interface `java.util.Observer` a spolu s UI tvoří dvojici podle návrhového vzoru Observer/Observable. Fungování tohoto vzoru je popsáno pro třídu UI v sekci 3.6. Toto rozhraní má pouze dvě metody, u kterých se očekává, že budou v implementacích přepsány a to:

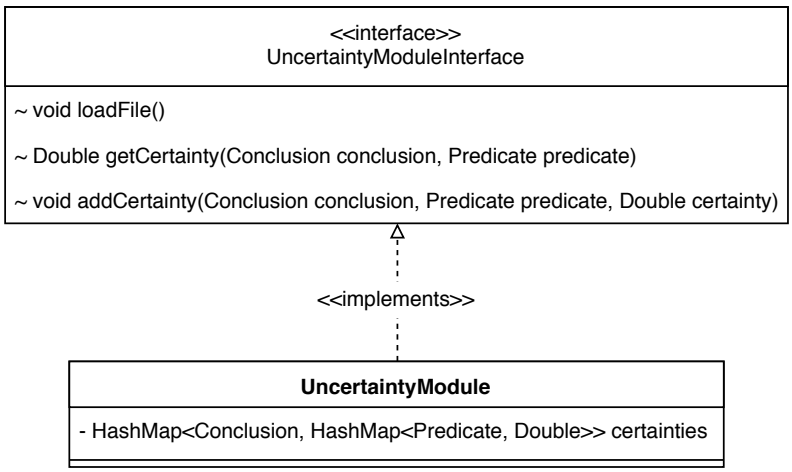
- `void update(Observable o, Object arg)`, což je metoda z rozhraní Observer, která se spustí ve chvíli, kdy UI zavolá jednu z metod notifikací. Očekává se, že podle druhu notifikace (z výčtového typu Notification) se v implementaci IM pošle do UI správný objekt, například pomocí metody `explainWhy()`.

- `void startInference()` slouží jako hlavní cyklus programu, který ke své práci využívá všechny moduly. Z tohoto cyklu se IM dotazuje uživatele, čte bázi znalostí, ukládá odpovědi a řeší neurčitost. Chování záleží na konkrétní implementaci a tedy na studentovi.

Členské proměnné této třídy jsou právě čtyři reference na zbývající moduly, které ke své práci IM využívá. To jsou tedy BZ, UI, modul nejistoty a báze dat.

Dvě hlavní implementace, které se očekávají, jsou dopředné řetězení a zpětné řetězení. Nic ale nebrání vlastní tvořivosti studenta.

3.5 Rozhraní modulu neurčitosti



Obrázek 3.8: Rozhraní modulu neurčitosti

Modul nejistoty je další rozhraní, které má připravenou jednu implementaci (viz obrázek 3.8) pomocí vnořené mapy. Stejně jako implementace báze znalostí, i modul nejistoty načítá data ze souboru. Tento soubor se jmenuje *Uncertainty* v balíčku *expertsystemfw.KnowledgeBase.Files*. Metody, které je nutné implementovat, jsou:

- `void loadFile()`, která načte soubor a uloží jednotlivé nejistoty do datové struktury.

- `Double getCertainty(Conclusion c, Predicate p)` navrací `Double` s hodnotou nejistoty (resp. jistoty) pro danou kombinaci predikátu a závěru.
- `void addCertainty(Conclusion c, Predicate p, Double uc)`, která přidá novou hodnotu nejistoty pro zadanou kombinaci predikátu a závěru.

3.5.1 Formát souboru

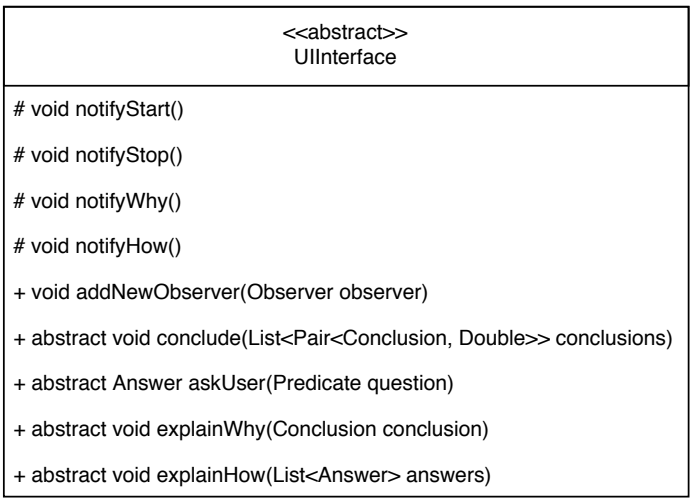
Formát každé řádky souboru se svou strukturou podobá souboru pro tvorbu sémantické sítě a vypadá následovně:

```
conclusion, predicate, certainty
conclusion; predicate; certainty
```

kde `conclusion` je název závěru, `predicate` je název predikátu a `certainty` je desetinné číslo od 0 do 1, které udává míru jistoty dané kombinace závěru a predikátu.

Modul je implementován tak, že pro kombinaci závěru a predikátu, která v něm není uložena, vrací 1. To znamená, že není potřeba do souboru vyplňovat všechny kombinace z báze znalostí, ale jen ty, které mají sníženou míru jistoty.

3.6 Uživatelské rozhraní



Obrázek 3.9: Rozhraní UI

Rozhraní UI (obrázek 3.9) je navrženo jako abstraktní třída, která má implementované metody pro komunikaci s IM pomocí návrhového vzoru Observer/Observable. Třída, která implementuje `java.util.Observer`, jíž je

v tomto případě IM, „pozoruje“ změny ve třídě UI, která rozšiřuje (dědí) `Observable` a při změně je zavolána metoda `update()` ze třídy IM. Změna ve třídě UI je indikována metodami `notifyStart()`, `notifyStop()`, atd. Poslední implementovanou metodou je `addNewObserver()`, která se musí zavolat před startem IM a do implementace UI se pomocí této metody vloží reference na Observer, kterou je právě IM.

Metody, které je nutné implementovat, jsou:

- `void conclude(List<Pair<Conclusion, Double>> c)`, která očekává seznam dvojic závěrů a desetinných čísel, která udávají s jakou jistotou systém uznal tento závěr za platný. Používá se na konci běhu IM.
- `Answer askUser(Predicate predicate)` se má uživatele dotázat na daný predikát a vrátit objekt typu Answer s danou odpovědí a predikátem.
- `void explainWhy(Conclusion conclusion)` je metoda, kterou IM zavolá z metody `update()`, když se uživatel dotáže „Proč?“. Tento proces je spuštěn z UI metodou `notifyWhy()`, když UI detekuje dotaz (ať už z terminálu, nebo kliknutí tlačítka v GUI implementaci). Tato metoda a celý proces jsou součástí vysvětlovací činnosti.
- `void explainHow()` má totožnou stavbu jako metoda `notifyWhy`, jen reaguje na dotaz typu „Jak?“. Tato metoda a proces jsou též součástí vysvětlovací činnosti.

Kapitola 4

Testování

V následující kapitole popíši způsoby, jakými byl framework testován a výsledky tohoto testování. Testování probíhalo ve většině případů pomocí JUnit testů, aby bylo možné simulovat jednotlivé případy, které mohou v praxi nastat. Samostatně nebyl testován modul neurčitosti a báze dat. Tato dvě rozhraní byla testována v rámci IM, který všechny jejich metody využíval, a které byly v případě potřeby odladěny v tomto procesu.

4.1 Testování báze znalostí

Testování bází znalostí probíhalo ve dvou fázích. V první fázi byla navržena BZ pro identifikaci problémů s počítačem. Tato báze byla za účelem otestování navržena shodně pro všechny tři implementace. To umožnilo ověřit, zda metody z tohoto rozhraní vrací totožné výsledky pro každou z implementací. Ukázka těchto testů je ve výpisu kódu 1. Stejným způsobem bylo ověřeno, zda metoda `getAllPredicates()` vrací stejné hodnoty pro všechny tři báze.

```
KnowledgeBaseInterface folKB = new FOLKnowledgeBase();
KnowledgeBaseInterface rulKB = new RulesKnowledgeBase();
KnowledgeBaseInterface semKB = new SemanticKnowledgeBase();

assertEquals(true,
    folKB.getAllConclusions().equals(rulKB.getAllConclusions()));
assertEquals(true,
    rulKB.getAllConclusions().equals(semKB.getAllConclusions()));
```

Výpis kódu 1: Ukázka JUnit testů BZ – getAllConclusions()

Podobným způsobem byla testována metoda `getPredicates(Conclusion)`, která má jako parametr závěr. Zde byly testovány všechny kombinace závěrů a bází znalostí způsobem, který je ukázán ve výpisu kódu 2. Pro toto porovnání

musely být predikáty sémantické sítě „standadizovány“, aby v něm byl pouze predikát bez relace.

```
folKB.getAllConclusions().forEach(c -> {
    assertEquals(true,
        folKB.getPredicates(c).equals(rulKB.getPredicates(c)));
    semKB.getPredicates(c).forEach(p -> {
        standardSemKBPredicate.add(new Predicate(p.getName()));
    });
    assertEquals(true,
        rulKB.getPredicates(c).equals(standardSemKBPredicate));
    standardSemKBPredicate.clear();
});
```

Výpis kódu 2: Ukázka JUnit testů BZ – getPredicate(Conclusion)

Poslední test byl zaměřen na metodu `query(Predicate, Conclusion)`, u které bylo důležité, aby vracela stejné odpovědi na stejné otázky do různých BZ. Test je ukázán ve výpisu kódu 3. Byly tak ověřeny všechny kombinace predikátů a závěrů, na které by se uživatel mohl dotázat.

```
folKB.getAllConclusions().forEach(c -> {
    semKB.getAllPredicates().forEach(p -> {
        assertEquals(folKB.query(p, c), rulKB.query(p, c));
        assertEquals(rulKB.query(p, c), semKB.query(p, c));
    });
});
```

Výpis kódu 3: Ukázka JUnit testů BZ – query(Predicate, Conclusion)

Druhá fáze testování BZ spočívala v testech funkcionalit typických pro jednotlivé implementace BZ. Na takové případy byly vytvářeny jen drobné BZ, které vždy obsahovaly testovaný případ. Například pro sémantickou síť bylo nutné ověřit, zda správně funguje dotazování se na transitivní a ne-transitivní relace, nebo že systém nedovolí vytvořit mezi dvěma uzly více než jednu relaci.

Pro predikátovou logiku bylo ověřováno, zda správně fungují kvantifikátory a logické spojky a dotazování se na BZ s výroky, ve kterých byly použity.

Pravidlová BZ byla testována na správnou funkčnost logických spojek a negací.

U všech tří implementací bylo samozřejmě testováno i samotné načítání souborů, ze kterých čerpají data.

4.2 Testování inferenčního mechanismu

Pro otestování IM byly vytvořeny dvě implementace a to dopředné řetězení a zpětné řetězení. Oba IM využívaly všech dostupných modulů ke své práci a rozhraní těchto modulů (obzvlášť BZ) byla dostačující k veškeré činnosti, která se od expertního systému očekává.

Stručný popis implementací:

Dopředné řetězení bylo navrženo jako cyklus, který postupně procházel všechny predikáty a dotazoval se na ně uživatele. Při odpovědi 0 (absolutně ne) na jeden z predikátů byly odstraněny všechny závěry, které tento predikát měly jako vlastnost se stoprocentní jistotou ze strany BZ (tedy experta). Všechny odpovědi byly ukládány do báze dat. Pokud na konci cyklu zbyly nějaké závěry, spočítala se pro ně pomocí báze dat (nejistota uživatele) a pomocí modulu nejistoty (nejistota experta) procentuální přesnost výsledku a byly přes modul UI vypsány uživateli. Pokud žádné nezbyly, byla vypsána hláška, že systém nenašel vhodné řešení specifikovaného problému.

Zpětné řetězení bylo implementováno pomocí zásobníků. Do zásobníku pro závěry byly vloženy všechny závěry z BZ. Potom dokud nebyl zásobník prázdný, vzal se první prvek, expandoval se na predikáty, které musely být pro daný cíl splněny a vložily se do zásobníku pro predikáty. Poté dokud nebyl tento zásobník prázdný se systém dotazoval uživatele a pokud byly splněny všechny predikáty, cíl byl uložen jako splněný. Pokud jeden z nich splněn nebyl, zásobník predikátů byl vyprázdněn a pokračovalo se dalším cílem. Ostatní funkcionality byly řešeny velice podobně, jako u dopředného řetězení.

Za zmínku stojí, že v rámci těchto testů se ověřila funkčnost návrhového vzoru Observer/Observable mezi IM a UI, která fungovala podle očekávání.

4.3 Testování uživatelského rozhraní

Toto rozhraní bylo z velké části testováno za běhu spolu s IM. Metody tohoto rozhraní jsou poměrně jednoduché a „přímočaré“, tudíž nebylo nutné hledat speciální krajní případy, jako například v bázi znalostí.

Pro potřeby testování bylo implementováno UI formou konzolového GUI. Je to nejjednodušší způsob GUI, při kterém se ale díky jeho jednoduchosti dají odhalit nedostatky a rychle testovat jeho funkčnost.

Jak již bylo zmíněno v testování IM, byla otestována komunikace mezi UI a IM, která je navržena pomocí Observer/Observable.

Závěr

V práci jsem se zabýval analýzou, návrhem a implementací frameworku pro tvorbu expertního systému. Cílem bylo navrhnout šablonu, která bude sloužit jako pomůcka při výuce BI-ZNS na FIT ČVUT, ve které bude možné implementovat jednotlivé části expertního systému.

Tento framework se skládá ze šablon tříd (interface nebo abstraktní třída), čímž je zajištěno fungování celého systému nezávisle na konkrétních implementacích, a proto je možné jednotlivé implementace libovolně kombinovat.

Bylo kladeno za cíl vytvořit celý projekt v jazyce Java. Projekt je v tomto jazyce vytvořen a je tak po stránce dovedností dostupný budoucím studentům předmětu BI-ZNS.

Posledním cílem bylo GUI, které je řešeno pomocí rozhraní a je možné ho tedy implementovat více způsoby. Například v rámci testování bylo implementováno jako konzolová aplikace, ve které uživatel psal své odpovědi.

Framework byl otestován a byla potvrzena modularita systému. Veškerý kód je řádně zdokumentován, aby měli budoucí uživatelé tohoto frameworku dostatečnou oporu při práci.

Zadání práce tak bylo splněno.

Literatura

[1] FIKAR, J.: *Expertní systém pro volbu vhodné metody využití odpadů.* Diplomová práce, Brno: Vysoké učení technické, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2011, vedoucí RNDr. Jiří Dvořák CSc. Dostupné z: `https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=42059`

[2] SCHNUPP, P.; Huu, C. T. N.; BERNHARD, L. W.: *Expert systems lab course.* Springer Science & Business Media, 2012, ISBN 978-3-642-74305-4.

[3] BERKA, P.: Expertní systémy. [online skripta], 1998, [cit. 2018-04-15]. Dostupné z: `http://sorry.vse.cz/~berka/4IZ229/`

[4] CELBOVÁ, I.: Úvod do problematiky expertních systémů. [online], 1999, [cit. 2018-04-15]. Dostupné z: `https://ikaros.cz/uvod-do-problematiky-expertnich-systemu`

[5] VERBÍK, J.: *Metody reprezentace znalostí.* Bakalářská práce, Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008, vedoucí práce doc. Ing. Václav Jirsík. Dostupné z: `https://core.ac.uk/download/pdf/30279603.pdf`

[6] FARUZEL, P.: Webový průvodce světem expertních systémů. [online], 2007, [cit. 2018-04-25]. Dostupné z: `http://faruzel.borec.cz/200.html`

[7] GIARRATANO, J. C.; RILEY, G.: *Expert systems: principles and programming.* Brooks/Cole Publishing Co., 1989, ISBN 0878353356.

[8] Kateřina TRLIFAJOVÁ, D. V.: *Matematická logika.* V Praze: České vysoké učení technické, 2013, ISBN 978-80-01-05342-3.

[9] DVOŘÁK, J.: Expertní Systémy. [online], 2004, [cit. 2018-04-25]. Dostupné z: `http://www.uai.fme.vutbr.cz/~jdvorak/Opory/ExpertniSystemy.pdf`

[10] MERTLÍK, T.: *Diagnostický expertní systém.* Bakalářská práce, Brno: Vysoké učení technické, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011, vedoucí Ing. Jan Karásek. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=42142

[11] Expertní systém. *FEL ČVUT: CourseWare Wiki*, 02 2018, [online]. Dostupné z: https://cw.fel.cvut.cz/wiki/courses/a5m33izs/cviceni/09

[12] THIMM, M.: Tweety. [online], 2018. Dostupné z: http://tweetyproject.org/index.html

[13] Schulz, P. D. S.: The E Theorem Prover. [online], 2018. Dostupné z: http://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html

[14] Cygwin authors: Cygwin. [online], 2018. Dostupné z: https://www.cygwin.com/

[15] THIMM, M.: Tweety Technical API 1.10. [online], 2018. Dostupné z: http://tweetyproject.org/api/1.10/index.html

Příloha **A**

Seznam použitých zkratek

API Application programming interface (rozhranní pro programování aplikací)

BI-ZNS Bakalářský předmět Znalostní systémy

BNF Backusova-Naurova forma

BZ Báze znalostí

DLL Dynamic-link library (dynamicky linkovaná knihovna)

ES Expertní systém

FIT ČVUT Fakulta informačních technologií, České vysoké učení technické v Praze

GUI Graphical user interface (grafické uživatelské rozhraní)

IDE Integrated development environment (vývojové prostředí)

IM Inferenční mechanismus

UI User interface (uživatelské rozhraní)

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `ExpertSystemFW` ..... adresář s projektem s ukázkami implementací
  - `ExpertSystemFW_blank` ..... adresář s projektem bez implementací
- `text` ..... text práce
  - `BP_Werner_Richard_2018.pdf` ..... text práce ve formátu PDF