VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

DIAGNOSTICKÝ EXPERTNÍ SYSTÉM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE TOMÁŠ MERTLÍK
AUTHOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Tomáš Mertlík	***ID:***	112052
Ročník:	3	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Diagnostický expertní systém

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta bude prostudovat literaturu a získat znalosti v oblasti expertních systémů, zejména diagnostických expertních systémů. Na základě získaných teoretických znalostí student navrhne obecný diagnostický expertní systém, který umožní řešit úlohy z oblasti strojírenství, průmyslu, lékařství, výpočetní techniky apod. Student návrh implementuje a otestuje na vhodném problému. Implementace bude provedena v programovacím jazyce JAVA.

DOPORUČENÁ LITERATURA:

[1] JIRSÍK, V.; HRÁČEK, P. Neuronové sítě, expertní systémy a rozpoznávání řeči: elektronický text AMT008. 2002. 107 s.
[2] MAŘÍK, V.; ZDRÁHAL, Z. a kol. Expertní systémy: principy, realizace, využití. Praha ČSVTS, 1984. 215 s.
[3] JACKSON, P. Introduction to expert systems : International computer science series. Addison-Wesley Pub. Co., 1986. 1st edition. Michiganská univerzita : [s.n.], 1986. 246 s. Digitalizováno 14. říjen 2006. ISBN 0201142236, 97802.

Termín zadání:	7.2.2011	***Termín odevzdání:***	2.6.2011

Vedoucí práce: Ing. Jan Karásek

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce obsahuje informace ohledně expertních systémů. Expertní systém je počítačový program, který má za úkol poskytovat expertní rady, rozhodnutí nebo doporučit řešení v konkrétní situaci. V této práci jsou definované čtyři verze expertních systémů, které se v praxi objevují. Popisují se tu základní, přídavné a doplňkové složky expertních systémů. Praktická část této práce se zabývá diagnostikou problému se spouštěním počítače pomocí expertního systému a vytvoření aplikace ve vývojovém prostředí JAVA ECLIPSE.

KLÍČOVÁ SLOVA

expertní systémy, umělá inteligence, neurčitost, inference, řídící mechanizmus, báze znalostí, báze dat, reprezentace znalostí, tvorba expertního systému, JAVA, ECLIPSE

ABSTRACT

This paper refers to expert systems. Expert system is a computer program dedicated to giving an expert advice, decisions or recommending the best solution in a particular situation. This paper mentions four types of expert system which are commonly used in practice. Basic, additional and complementary components of expert systems are described. The practical part of this paper refers to the diagnostic problem of computer start based on expert system implemented in JAVA programming language and ECLIPSE IDE.

KEYWORDS

expert systems, artificial intelligence (AI), indeterminateness, inference, control mechanism, knowledge base, data base, knowledge representation, creation expert system, JAVA, ECLIPSE

MERTLÍK, Tomáš *Diagnostický expertní systém*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 74 s. Vedoucí práce byl Ing. Jan Karásek

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Diagnostický expertní systém“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

Poděkování

Děkuji vedoucímu mé bakalářské práce Ing. Janu Karáskovi za účinnou, metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne ..... Podpis autora .....

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Klasické programovací jazyky umí pracovat jen s daty ve formě čísel, znaků apod. a neumí logicky odvozovat výsledky. Programy jsou vždy určené jen pro jednu věc a neumí dělat nic jiného, než na co jsou naprogramované. Oproti tomu člověk není naprogramován a tak může logicky uvažovat, učit se novým věcem a vyhodnocovat své závěry pomocí mnoha různých faktorů. Proto byla snaha o naprogramování inteligentního programu, který dokáže rozhodovat podle zadaných kritérií. Jedním z výsledků výzkumů o oblasti myšlení je ***umělá inteligence*** (UI), která se zaobírá vývojem strojů a programů vykazujících inteligentní chování[1]. Samotné ***expertní systémy*** (ES) jsou součástí UI, kde zaujímají velmi významnou pozici, protože zahrnují návrh komplexních systémů, které se v praxi používají. Existuje celá řada ES, které řeší různé oblasti složitých situací v praxi.

Výsledek této práce je zaměřen na program ES diagnostikující problém při zapínání ***osobního počítače*** (PC). Je mnoho problémů, proč počítač nestartuje správně, jako jsou problémy s napájením, špatné nastavení, poškozené komponenty počítače a mnoho dalších. Proto bylo dobré tento problém zpracovat do ES. Tím pádem je bude moci použít mnoho lidí a nemusí se zbytečně odkazovat na odbornou pomoc, když budou schopni si některé chyby opravit sami s použitím tohoto programu.

Tato práce se dělí na dvě hlavní kapitoly.

První kapitola bude teoreticky popisovat expertní systémy. V první části této kapitoly jsou popisovány základy a úvod do problematiky expertních systémů. Druhá část se zabývá výhodami a nevýhodami ES. V další části je uvedena jejich stručná historie od vzniku do současnosti. V následující části se definují hlavní rysy expertních systémů. Další tři teoretické části popisují základní, přídavné a doplňkové složky expertních systémů. Ty specifikují co bude ES obsahovat a jakým způsobem bude vyhodnocovat a prezentovat svoje výsledky a předpoklady. Poslední teoretická část popisuje čtyři základní typy expertních systémů.

Druhá kapitola bude praktická část této práce. Ta se bude zabývat bližším popisem problému se zapínáním osobního počítače a naprogramování expertního systému ve vývojovém prostředí JAVA ECLIPSE. Jelikož je zadání bakalářské práce vytvoření prázdného expertního systému, musí se nejdříve vytvořit program, který vytvoří databázi. Tuto vytvořenou databázi potom bude ES používat pro svou funkci.

[1] *Definice pojmu „inteligentní chování“ je stále předmětem diskuse, nejčastěji se jako etalon inteligence užívá lidský rozum.*[23]

1 EXPERTNÍ SYSTÉMY

1.1 Úvod do expertních systémů

Ještě než dojde k bližšímu seznámení s ES, tak bude stručně popsána problematika znalostních systémů a jejich rozdíly s ES. Jsou to programy, umožňující řešit problémy produktivními postupy na základě znalostí nebo poznatků, kterými jsou tyto systémy vybaveny. Všeobecně se pod pojmem znalostní systém chápe počítačový systém, který zpracovává poznatky, čímž projevuje znalost problematiky, které se tyto poznatky dotýkají.

Vedle toho ES je programový systém, který využívá vhodně reprezentované poznatky specialistů k řešení komplikovaných problémů, jenž obvykle vyžadují expertízu. Kromě řešení problémů se od ES (podobně jako od specialistů) požaduje i schopnost svá rozhodnutí a řešení zdůvodnit, tedy vysvětlit důvody, které ho k navrhovaným řešením problémů přivedly. Expertní a znalostní systémy tudíž nejsou synonyma. Platí, že každý ES je také znalostním systémem. Znalostní systém však nemusí mít všechny rysy ES (nejčastěji jim chybí schopnost vysvětlit svá rozhodnutí). Znalostní a expertní systémy tedy představují možnost, jak uchovávat nejen všeobecné teoretické znalosti, ale i vědomosti získané zkušenostmi v praxi. Znalostní systém je realizovaný poměrně rozsáhlou soustavou kooperujících programů. Jednotlivé programové celky této soustavy tvoří prvky funkčně vymezených a svou strukturou i posláním odlišných modulů, které společně se vzájemnými vazbami tvoří architekturu znalostního systému [14] [4].

Začátky prvních prototypů ES se objevily již v 60.tých letech 20. století a jejich komerční využití se datuje od počátku 80. let 20. století. V dnešní době se ES velice rozvíjejí. Expertní systémy podle prof. Edwarda Feigenbauma [5] ze Stanford University v USA, který patří mezi první pracovníky v oboru ES, jsou definovány takto:

„Jsou to počítačové programy, simulující rozhodovací činnost experta při řešení složitých úloh a využívající vhodně zakódovaných, explicitně vyjádřených speciálních znalostí, převzatých od experta, s cílem dosáhnout ve zvolené problémové oblasti kvality rozhodování na úrovni experta.“

To znamená, že je snaha udělat takový počítačový program, který bude uvažovat jako lidský expert v praxi. Tedy jednoduše se vezmou od experta z dané problematiky jeho dlouholeté zkušenosti a dají se do ES, který bude rozhodovat, jako kdyby úlohu řešil lidský expert. Bohužel tomu tak není, protože je to stále jen program a neumí nic jiného, než co do něho napíšeme. Neumí totiž hodnotit různé výjimky, které mohou nastat v reálné praxi, sice se jim hodně přibližuje, ale nejde do ES dát

všechnu moudrost experta, jen rozhodování podmínek co mohlo a nemohlo nastat. Tudíž programování takového ES je velice složitá věc a v této práci se bude probírat hned po rozboru ES.

V textu se bude často objevovat slovo expert. Bude lepší toto slovo blíže definovat. Znamená to člověka, který je specializován v určité oblasti znalostí a tyto jeho specializované schopnosti nejsou známé nebo dostupné ostatním lidem. Expert je schopen řešit problémy, které většina lidí není schopna řešit, nebo je umí řešit lépe. Právě proto ES obsahují znalosti od expertů, ale také i z knih, časopisů a jiných lidí.[18]

1.2 Výhody a nevýhody ES

Jeho hlavní výhody:

- Základní výhodou ES je zvýšená dostupnost znalostí „uveřejněných“ v odladěné bázi znalostí. Zatímco expert může provádět v jednom momentě pouze jedinou konzultaci, ES může být nainstalován na mnoha počítačích, a tak své znalosti poskytovat více lidem současně [8].
- Může pracovat 24 hod denně 7 dní v týdnu (nemusí si brát dovolenou, nedá výpověď, neonemocní). Funguje vždy, kdy je ho zapotřebí.
- Poskytuje stále stejné výsledky, neovlivňuje ho únava, časový stres či jiné faktory. Pokud je v ES zabudovaný dobrý algoritmus, dává jednoznačné odpovědi.
- U některých ES se dokonce v praxi ukázala větší spolehlivost než u expertů. V případě, že se více expertů nemůže dohodnout na postupu, může expertní systém nabídnout alternativní řešení [7].
- ES lze nasadit i do prostředí, kde je pobyt pro člověka rizikový (chemická výroba, jaderný reaktor). Systém snímá přes čidla různé hodnoty a rozhoduje o dalším postupu, aniž by se v daném prostředí musel pohybovat člověk [8].
- ES umožňují udržování a rozšiřování odborných znalostí. Jediný odborník vytvoří znalostní bázi, která je k dispozici ostatním uživatelům [7].
- ES umožňují využívat odborné znalosti na vzdálených místech (kdekoliv na světě).
- ES mohou využívat všichni. Pro jejich používání nemusí člověk znát podrobně danou problematiku. Často ES obsahuje i vysvětlovací mechanizmus, kde zdůvodňuje své závěry.
- Cena za pořízení ES bude zpravidla podstatně nižší než placené konzultace experta. Do jednoho systému lze také zakomponovat znalosti více expertů současně. Kombinovaná znalost více expertů může převýšit znalosti jednoho experta [8].

Jeho nevýhody jsou následující: [24], [10]

- Lidský expert má vlastnosti, které do ES nelze vůbec promítnout např. lidský rozum, intuici, schopnost rozpoznat velmi vzácné výjimky a okamžitě se jim přizpůsobit
- Na lidském rozhodování se podílí i další znalosti a schopnosti, které s problémem na první pohled přímo nesouvisí např. všeobecný přehled, lidská životní zkušenost a moudrost, zdravý úsudek, vtip apod.
- Znalosti a dovednosti člověka se s časem a získanými zkušenostmi vyvíjejí. Expertní nebo znalostní systém bude bez lidského zásahu i za dvacet let navrhovat stejné postupy.

1.3 Historie

Tato kapitola bude pojednávat o historii expertních systémů od vzniku do současnosti. Samozřejmě jich existovalo a existuje mnohem více než je zde uvedeno, ale není úkolem této práce popsat celou historii expertních systémů. Je jich skutečně mnoho, obzvlášť v dnešní době, kde jsou expertní systémy ve velkém rozvoji. Proto zde jsou uvedeny jen ty, co napomohly celkovému vývoji a rozvoji expertních systémů. Tato kapitola byla čerpána z [21], [4], [6].

- rok 1971 - **DENDRAL**
  - Napomáhal při identifikaci chemických sloučenin na základě spektrografických dat a při odvozování struktur těchto sloučenin.
  - Vytvořen dříve, než došlo k zavedení pojmu ES.
- první polovina 70. let 20. století 1.– **MYCIN**
  - Klasický ES.
  - Systém využívaný v oblasti medicíny - k rychlému určení bakteriální infekce ze snadno dostupných informací.
  - Pomáhal při výběru vhodné léčby antibiotiky.
  - Významný přínos pro oblast umělé inteligence - metody použité v systému MYCIN se dodnes používají v řadě ES.
- první polovina 70. let 20. století 2.– **PROSPECTOR**
  - Používán v geologii.
  - Při prvním testování se podařilo tomuto systému odhalit ložisko molybdenu v hodnotě sta milionu dolarů.
  - Pro umělou inteligenci je tento systém přínosem zejména svým způsobem zpracování neurčité informace, který se dodnes v různých modifikacích používá.
- 80. leta 20. století - **EMYCIN (KAS)**

  - Tzv. prázdné ES, do kterých stačilo přidat databázi znalostí
  - Použité metody zpracování informací systémů MYCIN a PROSPECTOR.
- rok 1983 - **PUFF**
  - Slouží k odhalení příčin problémů dýchacích cest.
  - Následník systému MYCIN.
  - Dalšími systémy založené na systému MYCIN jsou např. systém ONCOCIN, který napomáhá určovat průběh léčby pacienta na onkologickém oddělení. Systém CLOT, který rozpoznává problémy srážlivosti krve. Systém DART, který byl určen k diagnostice závad počítačů.
- rok 1984 - **INTERNIST**
  - Významný ES.
  - cíl: Pokrýt svými znalostmi celou oblast interní medicíny.
  - Dodnes je považován za jeden z nejobsáhlejších ES.
  - Využívá se dodnes.
- A mnoho dalších

1.4 Hlavní rysy ES

Expertní systémy se charakterizují podle Berky [2] následujícími rysy:

1. **Oddělení znalostí a mechanismu pro jejich využívání.** Znalosti experta jsou uloženy v bázi znalostí odděleně od inferenčního (odvozovacího) mechanismu, na rozdíl od konvenčních programů, kde jsou znalosti experta pevně zakódovány v jednotlivých instrukcích programu, které se aplikují v předem stanoveném pořadí. To umožňuje vytvářet problémově nezávislé (prázdné) ES, ve kterých jeden inferenční mechanismus může pracovat s různými bázemi znalostí.
2. **Neurčitost v bázi znalostí.** V bázi znalostí jsou uloženy nejen exaktně dokázané znalosti, ale i zkušenosti experta, které se mu osvědčily v jeho praxi. Mohou se zde pak objevit pojmy jako „většinou“, „často“, „nevím“ apod.. Tyto pojmy je nutno kvantifikovat.
3. **Neurčitost v datech.** Konkrétní data o řešeném případu bývají zatížena neurčitostí způsobenou nepřesně určenými hodnotami nebo subjektivním pohledem uživatele.
4. **Vysvětlovací činnost.** ES by měl poskytnout vysvětlení svého uvažování. Počítačový ES musí být nejen schopen vést s uživatelem dialog ve formě otázka-odpověď, nýbrž musí být schopen vysvětlit a zdůvodnit dílčí závěry i položit vhodný doplňující dotaz stejně tak, jako být připraven zaměřit své úsilí na řešení podproblémů specifikovaných uživatelem v průběhu konzultace.

5. **Dialogový režim.** ES jsou obvykle vytvářeny jako tzv. konzultační systémy. Uživatel se systémem komunikuje způsobem „dotaz systému – odpověď uživatele“ (podobně by komunikoval i s lidským expertem). Z hlediska vnějšího chování je na expertní systémy kladena řada požadavků, odvíjejících se z představy, že expertní systém nahrazuje odborníka,poskytujícího konzultaci laikovi či méně zkušenému odborníkovi.
6. **Modularita a transparentnost báze znalostí.** Pro účinnost ES je rozhodující báze znalostí. Znalosti experta nemají statický charakter, nýbrž se postupně vyvíjejí a rozrůstají. Modularita umožňuje snadnou aktualizaci báze znalostí. Báze znalostí musí být též transparentní (tedy čitelná a srozumitelná) jak pro experta (aby ji mohl upravovat a rozšiřovat), tak i pro další odborné pracovníky, kteří z ní mohou čerpat potřebné znalosti. Vytváření báze znalostí probíhá interaktivně konzultacemi experta z dané problémové oblasti s odborníkem na tvorbu bází (znalostním inženýrem). Báze znalostí je tak postupně upravována, až chování ES odpovídá představám experta.

1.5 Základní složky

Každý ES se skládá ze tří hlavních složek, která tvoří minimální strukturu celého ES. Jsou to:

- řídící mechanizmus,
- báze znalostí,
- báze dat.

Základní princip ES je již uveden na obrázku 1.1. Spočívá v oddělení řídícího mechanizmu od báze dat a báze znalostí. Vzájemné oddělení báze znalostí a báze dat nemusí platit pro všechny ES.

Obr. 1.1: Základní složky ES [13].

V následujících kapitolách budou detailněji popsány základní složky ES.

1.5.1 Báze dat

První doplněk je báze dat. Báze je tvořena pasivními údajovými strukturami. Báze znalostí obsahuje symbolickou reprezentaci všeobecně platných poznatků a dané problémové oblasti. Báze dat uchovává symbolickou reprezentaci konkrétních faktů (údajů) souvisejících s právě řešeným problémem. Odpovídající data pak používá rozhodovací mechanismus v průběhu svojí činnosti [13].

Obsah báze dat se v průběhu odvozování zpravidla značně mění. Nejčastěji se doplňuje, tj. vytváří se v ní nové položky. Dochází tam i k rušení a modifikaci již stávajících položek. Některé položky vznikají při počáteční specifikaci problému, jiné v průběhu řešení problému. Výskyt nebo obsah některých položek v bázi dat může mít jen podmíněnou platnost. Jsou to položky odpovídající údajům, které při řešení daného problému za daných okolností není možno ani přímo získat. Vznikají „odhadem“ jako očekávané a předpokládané údaje. Platnost nebo pravdivost je přípustná do té doby dokud se nedostanou do rozporu s údaji odpovídající realitě nebo ohraničujícím podmínkám daným pro řešený problém [8].

1.5.2 Báze znalostí

V každém ES je nevyhnutelnou součástí k řídícímu mechanizmu báze znalostí. Ke každému řídícímu (inferenčnímu) mechanizmu se může připojit více bází znalostí, které se odlišují obsahově. Proto stačí jen jeden řídící mechanizmus a různé báze znalostí, které popisují jiné problematiky a ES se využije ve více účelech. Báze znalostí je v podstatě to samé co deklarativní program. Obsah báze znalostí je vytvořený údajovou strukturou, která nevykonává žádné instrukce. Zahrnuje znalosti experta, které jsou potřebné k řešení daného problému. Tato báze je koncepčně podobná databázi [15].

Typy reprezentace poznatků, definované v [13], které najdeme v ES:

- **kauzální** - odpovídají známé a zdůvodnitelné souvislosti mezi dvěma entitami (objekty), které jsou a nebo můžou být ve vztahu příčiny a následku,
- **asociativní** - odpovídají obvyklým, pozorovatelným či pravděpodobným, ne však zdůvodnitelným a ani nevyhnutelným vztahem mezi dvojicemi entit,
- **taxonomické** - odpovídají známým souvislostem mezi entitami, které vystihují vztah všeobecného k speciálnímu,
- **časové** - odpovídají takovým souvislostem mezi entitami, které na základě jejich výskytu nebo jejich změn v určitých časových okamžicích či intervalech podmiňují okamžité nebo časově následované výskyty. Tím umožňují uvažovat o jevech minulých a budoucích,

- **prostorové** - odpovídají prostorovým vztahům mezi entitami (například vztahem sousedství celků, nacházení se uvnitř nebo venku, poloha a orientace v prostoru, směr pohybu),
- **modelové** - odpovídají známým, či předpokládaným dynamickým souvislostem zpravidla mezi větším počtem určitým způsobem převzatých entit, které se navzájem ovlivňují a tím určují správné řízení nimi vytvořeného systému,
- **kontextové** - odpovídají různým podmínkám, za kterých se uplatňují určité souvislosti mezi entitami (vyjadřují například okolnosti, od kterých závisí, zda se souvislost uplatní a nebo neuplatní).

Způsob reprezentace poznatků se musí zvolit při tvorbě ES. Dále to již nelze měnit. Odpovídající úvahy musí být těsně propojené s předpokládanými vlastnostmi řídícího mechanizmu, případě i dalších procedurálních složek systému.

Je-li vytvořen prázdný ES, který ještě neobsahuje znalosti experta v dané problematice, stačí do báze znalostí zadat poznatky od experta s již předem zadanou reprezentací poznatků. Báze znalostí obsahuje znalosti z určité domény a specifické znalosti o řešení problémů v této doméně.

Znalosti mohou být nejrůznějšího charakteru [4]:
- od nejobecnějších znalostí k úzce odborným znalostem,
- od exaktně prokázaných znalostí až k nejistým heuristikám,
- od jednoduchých znalostí až po metaznalosti (tj. znalosti o znalostech).

Specialitou jsou soukromé znalosti (též označovány jako heuristiky či nejisté znalosti). Jde o exaktně nedokázané znalosti, které expert získává postupně v průběhu praxe a o nichž ví, že mu pomáhají při řešení určitých problémů. Tyto nejisté znalosti však nezaručují nalezení správného řešení. Heuristiky odlišují znalosti experta od znalostí průměrného pracovníka (resp. laika) [3].

Vedle své hlavní úlohy (obecný systém pravidel pro řešení problému) může být báze znalostí využita k výuce nebo k získávání informací (znalostí) z oboru, na který je báze znalostí orientována [3].

Pro reprezentaci znalostí se nejčastěji využívají tyto prostředky [4]:
- pravidla,
- matematická logika,
- rámce a scénáře,
- rozhodovací stromy,
- sématické sítě,
- objekty.

1.5.3 Řídící mechanizmus

Řídicí nebo-li inferenční (odvozovací) mechanismus je programový modul, který předem udává strategii využívání znalostí z báze znalostí, zprostředkovává komunikaci mezi bází znalostí a bází dat (resp. bází znalostí a uživatelem ES). Existuje mnoho technik na prohledávání stavového prostoru, který může být vyjádřen různými způsoby. Tato kapitola bude popisovat tři podstatná hlediska pohledu na řídící mechanizmus. Jedná se o vnější (uživatelský), vnitřní (funkční) a realizační pohled.

Z vnějšího pohledu na ES jsou podstatné funkce rozhodovacího mechanismu, které uživatel vnímá při práci s expertním systémem. To je například deduktivní nebo induktivní usuzování, zahrnování, analogie nebo odhad, heuristické usuzování, zaměření či odpoutání pozornosti, usuzování za přítomnosti nejistot a neurčitosti. Při vnitřním pohledu se musíme zaměřit na funkce rozhodovacího mechanismu, které souvisí s teoretickými principy jeho konstrukce. Je to například teorie grafů, síťové závislosti, syntaktická analýza, matematická lingvistika a logika apod. [13], [11].

Jako poslední je uvedený realizační pohled na rozhodovací mechanizmus. To jsou funkce, které jsou sledované přes programovací prostředí a programovací jazyky. Základní funkce rozhodovacího mechanismu jsou realizovatelné dvěma různými metodami nebo jejich kombinací. Jsou to metody zpětný a přímý chod [13].

Přímý chod řešení problému vyplývá z určitého objemu známých dat, které jsou potřebné odvodit. Je potřebné odvodit co z nich vyplývá a k jakým důsledkům je možné na jejich základě dospět. Počáteční údaje charakterizují počáteční stav řešeného problému, které postačí na jeho vyřešení [4].

Zpětný chod řešení problémů spočívá v nacházení vhodného a efektivního způsobu dosahování určitého dopředu stanoveného cíle. Příkladem této metody je potvrzení nebo vyvrácení uvažované hypotézy: *jsem nemocný, benzín nehoří, bolí mě noha.* Zodpovězení dané otázky: *Je venku teplo?* a nebo odvození konstrukce požadovaného zařízení: *návrh počítače s požadovanými parametry* [4].

Metody inference (odvozování) [4]:

- **Indukce** - usuzování z jednotlivého na obecné nebo přesněji řečeno jde o poznání, které vychází z empiricky zjištěných faktů a dospívá k obecným závěrům.
- **Intuice** - obtížně vysvětlitelný způsob usuzování, jehož závěry jsou možná založeny na nevědomém rozpoznání nějakého vzoru. Tento typ usuzování zatím nebyl v ES implementován a snad by se k němu mohlo blížit usuzování neuronových sítí.
- **Dedukce** - logické usuzování, při němž závěry musejí vyplývat z předpokladů.
- **Generování a testování** - Asi nejznámější metoda z programového hlediska

kterou využívá řada lidí. Nazývá se to také jako metoda pokus-omyl,

- **Abdukce** - usuzování směřující ze správného závěru k předpokladům, které jej mohly způsobit.
- **Analogie** - odvozování závěru na základě podobnosti situace vzhledem k jiné známé situaci.
- **Heuristika** - je to metoda, pomocí které se řeší problémy neobvyklým způsobem. Využívá se především jako technika, při které se často velice rychle najde velké množství řešení, z nichž je vybráno to nejlepší možné. Jsou to hrubé odhady převážně na základě zkušeností.
- **Standardní inference** - usuzování na základě obecných znalostí, pokud chybějí znalosti specifické.

Jednou z nejdůležitějších schopností ES je zpracování neurčitosti. Neurčitost se může vyskytovat v bázi znalostí tak i v bázi dat.

Zdroje neurčitosti jsou:

- nejisté znalosti,
- nepřesnost zadaní,
- nekompletnost,
- nekonzistence dat.

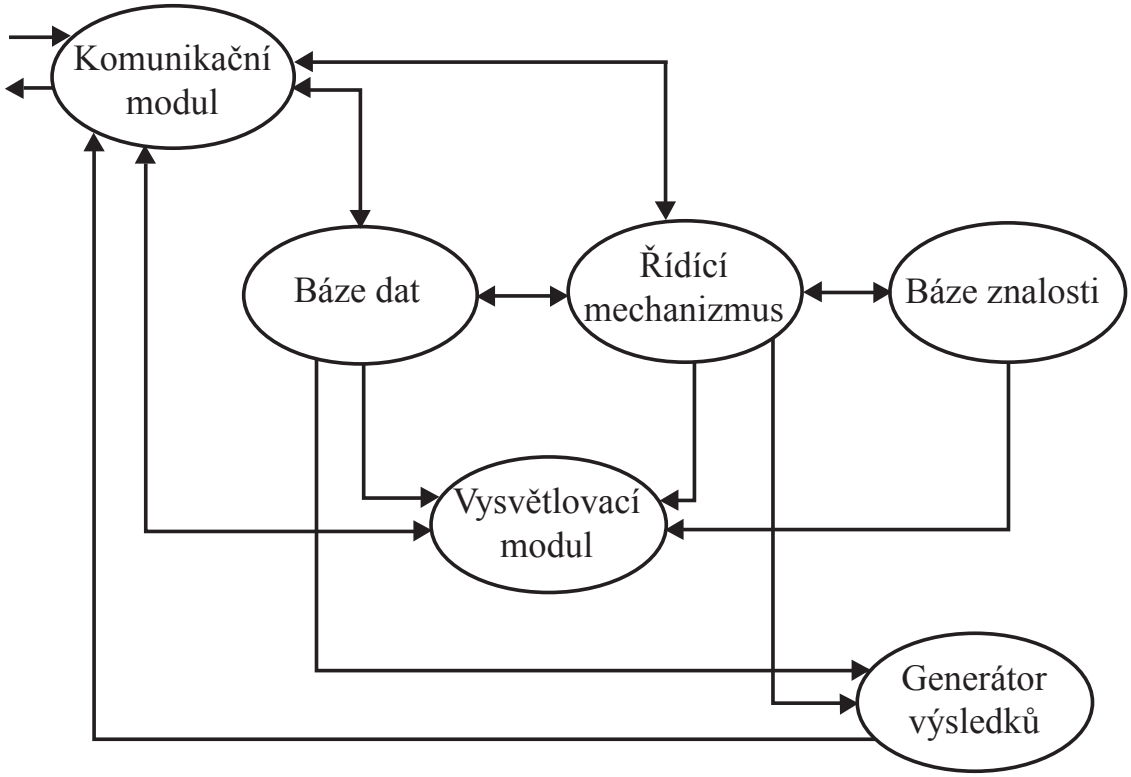
1.6 Přídavné složky

ES se nemusí skládat pouze ze tří základních částí (báze dat, báze znalostí, řídící mechanismus) popsaných v předchozích kapitolách. Jsou to části, které musí být v každém ES. Naproti tomu v této kapitole budou popsány další přídavné složky, které se nemusí vždy v ES vyskytovat. Záleží tak na tvůrci ES, zda zahrne do programu přídavné složky.

Na následujícím obrázku 1.2 jsou znázorněny vazby mezi základními a přídavnými složkami expertního systému.

Rozeznáváme tři samostatné celky přídavných složek [13] :

- **komunikační modul** - zabezpečuje komunikaci mezi expertním systémem a uživatelem,
- **vysvětlovací modul** - vysvětluje stav a průběh řešeného problému,
- **generátor výsledků** - sestavuje průběžné výsledky do celku v požadovaném a srozumitelném tvaru, bez nadbytečných informací.



Obr. 1.2: Základní a přídavné složky expertních systémů [13].

1.6.1 Komunikační modul

Tento modul je důležitý pro toho, kdo potřebuje radu experta nebo jeho službu. Cílem využívání tohoto modulu je pohodlné používání ES a srozumitelnou komunikaci (interakci) mezi uživatelem a ES.

Jeho hlavní funkce jsou [13]:

- realizace dialogu v průběhu odvozování. Předkládání dotazů uživateli při shromažďování potřebných dat, načítání jeho odpovědí, kontrola jejich správnosti a vypisování chybových hlášek,
- inicializace a ukončení činnosti expertního systému jako celku a také inicializace činností některých jeho modulů,
- obsloužení povelů (příkazů a požadavků) uživatele.

Při inicializaci ES se specifikují služby, které od něho uživatel očekává. Nejčastěji to je řešení problému nebo vytváření či modifikování báze znalostí. Některé ES umožňují po inicializaci svojí odvozovací činnosti specifikovat rozsah interakce požadované uživatelem a jednak způsob, rozsah a detailnost odvozování v závislosti od funkčnosti systému. To znamená zadání faktů do báze dat a určení přesného cíle řešení včetně případných ohraničení pro každý výsledek. Tímto je myšleno, že by

měl mít uživatel možnost si zvolit vlastní ohraničení a tím ovlivnit postupy řešení daného problému i související dialog se systémem [11].

Formy a rozsah dialogu může být podmíněný více faktory. Interaktivní schopnosti komunikačního modulu mohou být vytvořené způsobem, který připouští činnost komunikačního modulu ve více režimech. Měl by například připustit volbu jestli je uživatel v dané problematice začátečník nebo zkušený odborník. Zkušený odborník nepotřebuje tak rozsáhlé dialogy jako začátečník. Stačí mu pouze heslovitý a stručný dialog. Zatímco začátečník může vyžadovat mnoho navazujících a vysvětlujících pokynů, aby pochopil danou problematiku a mohl pracovat s ES [4].

1.6.2 Vysvětlovací modul

Pokud je možnost komunikovat s expertem v dané problematice, tak od něj očekáváme, že svoje rozhodnutí dokáže vysvětlit a zdůvodnit. Funkce vysvětlovacího modulu má za úkol právě vysvětlování a zdůvodňování stavu a průběhu řešení problému. Dále zdůvodňuje jednotlivé řešící kroky a taky dosažené výsledky. Při výsledku by mělo být patřičné vysvětlení proč si ES tento výsledek zvolil jako konečný. Není však podmínkou, aby byl vysvětlovací modul v každém ES. V některých ES je dokonce nežádoucí.

V nejjednodušším provedení by měl vysvětlovací modul být schopný odpovídat na dva typy nejčastějších otázek [20].

- **Proč**
  - požaduje od uživatele daný fakt?
  - volí určitý způsob odvozování?
  - odvozuje určitý fakt?
- **Jak**
  - dospěl k odvození daného faktu?
  - dospěl k danému způsobu odvozování?
  - dospěl k danému výsledku?

V případě typu otázky „*proč?*“, vysvětlovací modul bude vysvětlovat, co se děje a nebo se předpokládá, že se bude dít v navazujícím odvozovacím procesu.
V případě typu otázky „*jak?*“, vysvětlovací modul vysvětluje jak dospěl k určitému závěru.

1.6.3 Generátor výsledků

Při řešení problému se člověk nevyhne potřebě vykonávat alternativní kroky a postupy řešení. Mnohé z nich se časem ukážou jako zbytečné, nevhodné nebo přímo

špatné. Když pak řešitel po vyřešení problému oznamuje výsledek a to, jak k němu dospěl, reprodukuje většinou jenom ohraničenou část svého postupu, a sice tu, která vedla k cíli. O ostatních, chybných nebo zavádějících krocích, o neúspěšných pokusech a neopodstatněných předpokladech hovoří v běžných situacích jen zřídka. Jeho výklad vzbuzuje představu postupu vedoucího víceméně přímočaře k výsledku [20].

Podobným způsobem by měl předkládat výsledek i expertní systém. Buď z výsledků jako takových, nebo z vysvětlení poskytnutého vysvětlovacím modulem by měl bezprostředně vyplynout úspěšný postup od počátečního do koncového stavu řešení. Expertní systém by měl oznamovat postup a dosažené výsledky přímočaře, bez nadbytečných informacích o odvozovacích „obchůzkách“, které jsou, vzhledem k běžnému užívání systému nepotřebné, popřípadě i zavádějící. Úlohou generátora výsledků je zabezpečit takovou prezentaci výsledků řešení problémů. Z faktů, zjištěných odvozováním nebo kladením otázek začleňuje generátor výsledků do závěrečné zprávy jen ty, které bezprostředně souvisí s dosaženým výsledkem a s tou částí odvození, která k němu vedla. Uspořádává vybrané fakta do odůvodněných a obsahově vzájemně souvisejících celků, nepoužívá ty, které jsou nadbytečné, zhodnocuje mezivýsledky v kontextu celého postupu řešení a začleňuje je do uspořádaného a srozumitelného celku [20].

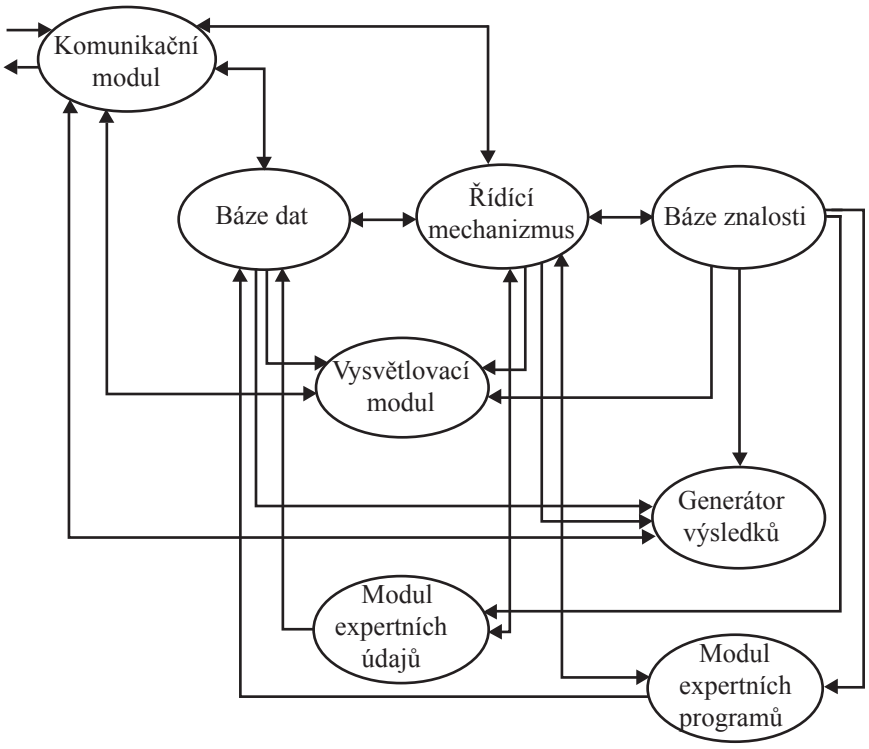
Generátor výsledků se nejvíce uplatní v plánovacích ES, které budou detailněji popsány v kapitole 1.8.2.

Generátor výsledků poskytuje a umožňuje [13]:

- zabezpečuje prezentaci výsledků řešených problémů,
- vybrané fakty uspořádá do odvozených a obsahově vzájemně souvisejících celků a požije při vyhodnocení jen ty z nich, které nejsou nadbytečné,
- jednotlivé částečné výsledky zhodnocuje v kontextu celého řešícího postupu a integruje je do uspořádaného a srozumitelného celku,
- při tvorbě báze znalostí je potřebné reprezentovat i znalosti, které slouží na „inteligentní“ činnost generátoru výsledků.

1.7 Doplňkové složky ES

V ES se dále vyskytují další doplňkové složky. Je to modul expertních programů a modul expertních údajů. Začlenění je naznačeno na obrázku 1.3.



Obr. 1.3: Základní, přídavné a doplňkové složky ES [13].

1.7.1 Modul expertních údajů

Když ES pracuje s externími údaji, tak rozhodovací mechanismus vyžaduje určitý fakt na probíhající odvozování. Rozhodovací mechanismus se tento fakt může pokusit získat. Prostředkem pro získání požadovaného faktu může být modul expertních údajů. Základní úkol modulu expertních údajů je sledovat situace ve kterých rozhodovací mechanizmus požaduje údaje nebo fakty. Jestli tato situace nastane, modul expertních údajů přebere řízení a zabezpečí prohledávání expertních údajů. Pokud nalezne požadovaný fakt, umožní ho vložit do báze dat a předá řízení nazpět rozhodovacímu mechanizmu. V případě, že modul požadovaný fakt nenajde, musí se tato situace ošetřit [13].

1.7.2 Modul expertních programů

Modul expertních programů zajišťuje interakci ES s programy ze svého okolí. Vzájemná interakce může být zavedena z obou stran, jak expertním programem, tak ES.

ES ji může vyvolat, když požadovaný fakt může získat aktivací procedury [8]:

- zabezpečující naplnění registrů požadovanými daty,
- zabezpečující prohledávání registrů, do kterých se vkládají aktuální údaje,
- zabezpečující vykonání určité části řešení problémů.

Interakce se může vyvolat expertním programem aktivující ES když [8]:

- interpretace údajů vyžaduje produktivní řešící postupy pomocí znalosti experta,
- je potřebné rozhodnout na základě znalostí, který z dostupných programů je pro daná kritéria nejvhodnější pro další zpracování zadaných údajů.

1.8 Typy ES

Z hlediska charakteru řešených úloh lze existující ES rozdělit do čtyř skupin:

1. **diagnostické**,
2. **plánovací**,
3. **hybridní**,
4. **prázdné**.

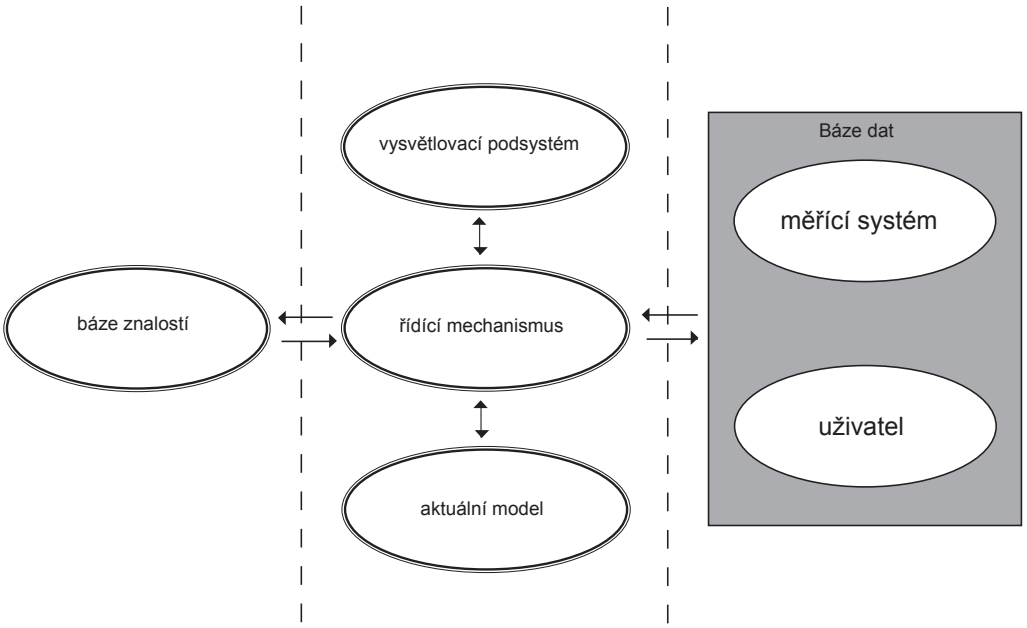
Architektura ES vychází ze základní struktury, která je vždy tvořena bází znalostí na jedné straně a inferenčním mechanismem na straně druhé. Další rysy jejich architektury jsou dány jeho typem, tj. určením pro řešení buď diagnostických nebo plánovacích úloh.

Efektivita expertního systému je rozhodujícím způsobem ovlivňována kvalitou báze znalostí. Tvorba báze znalostí je obvykle dlouhodobým procesem získávání znalostí od experta a jejich kódování do tvaru akceptovatelného příslušným inferenčním mechanismem [12].

1.8.1 Diagnostické ES

Úlohou diagnostických ES je provádět efektivní interpretaci dat s cílem určit, která z hypotéz z předem stanovené konečné množiny cílových hypotéz o chování zkoumaného systému nejlépe koresponduje s reálnými daty konkrétního případu. Nejtypičtější příklad je stanovení diagnózy pacienta na základě jeho subjektivních potíží. [19]. Architektura diagnostického ES je znázorněna na následujícím obrázku (obr. 1.4).

Jádrem řídícího systému je řídící mechanizmus, který pomocí znalostí a báze dat upřesňuje po každé odpovědi aktuální model řešeného problému. Řídicí mechanismus odpovídá za výběr dotazu, po jehož zodpovězení dokáže zpřesnit aktuální model a také za úpravu aktuálního modelu po obdržení odpovědi. Aktuální model



Obr. 1.4: Blokové schéma diagnostického ES [19].

reprezentuje současný stav řešeného problému, tzn. je množinou všech právě platných poznatků a faktů o konzultovaném problému. Aktuální model se může měnit dvojím způsobem [22]:

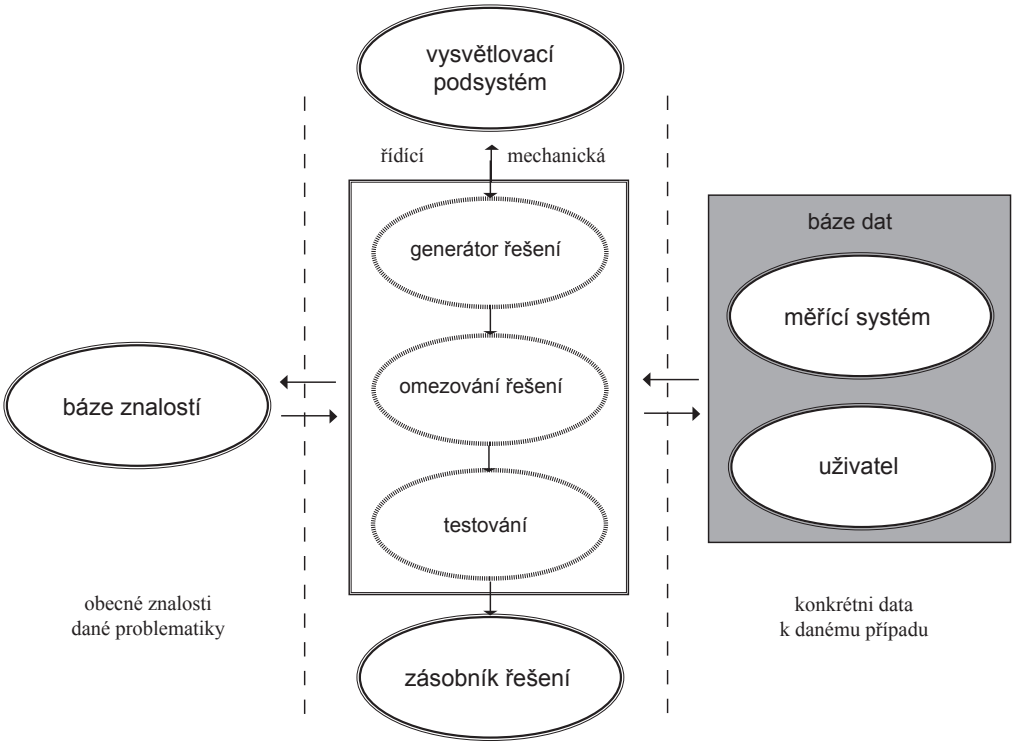
- přidáním nového údaje do databáze,
- odvození nového údaje z aktuálního modelu.

Vysvětlovací podsystém je uživatelsky významnou částí ES, která poskytuje informace o konkrétním postupu, jímž bylo dosaženo závěru.

1.8.2 Plánovací ES

Plánovací ES jsou určeny pro řešení takových úloh, kdy je znám stav počátečního objektu a cíl řešení a systém má s využitím dat a zadání nalézt optimální posloupnost povolených kroků, kterými lze cíle dosáhnout. Na obrázku 1.5 je znázorněné blokové schéma plánovacího ES.

Základní částí plánovacího ES je generátor možných řešení, který automaticky kombinuje posloupnost operátorů a vytváří tak kombinace posloupností operátorů (varianty řešení). Dále je testována shoda řešení s daty z báze dat. Výsledkem je seznam přípustných řešení, z nichž každé je ohodnoceno stupněm kvality. S rostoucím počtem operátorů velmi rychle narůstá počet možných kombinací. Data o konkrétním případu resp. znalosti experta tento nárůst generovaných řešení výrazně omezují. Výsledkem činnosti plánovacího ES je nabídka ohodnocených přípustných řešení, z nichž si uživatel podle svých kritérií vybere jednu dominantní [19], [10].



Obr. 1.5: Blokové schéma plánovacího ES [19].

Řídicí mechanismus s využitím báze znalostí a báze dat [3]:

- ovlivňuje výběr přípustných operátorů,
- omezuje generativní schopnost generátoru použitím znalostí z báze znalostí např. apriorním zamítnutím některých dílčích posloupností kroků,
- řídí testování shody vygenerovaných řešení s daty z báze dat a tím utváření zásobníku potenciálních řešení, včetně ohodnocení jejich vhodnosti.

1.8.3 **Hybridní** ES

Hybridní ES se vyznačují kombinovanou architekturou, tj. kombinují plánovaní a diagnostiku. Příkladem může být inteligentní expertní systém pro výuku, který diagnostikuje znalosti studenta a podle toho plánuje jeho další vzdělávání nebo monitorovací systémy, kdy je diagnostika poruchy v okamžiku detekce poruchy střídána podsystémem pro plánování zásahu [19], [22].

1.8.4 **Prázdné** ES

Důležitým nástrojem jsou tzv. prázdné ES. Diagnostické či plánovací mohou být bez báze znalostí a bez báze dat. Doplněním báze znalostí k prázdnému systému vzniká orientace na řešení příslušné problematiky. Doplněním báze dat je vždy řešen

konkrétní problém. Idea prázdných systémů vychází ze skutečnosti, že základní a zcela univerzální částí ES je jeho inferenční mechanismus.

Inferenční mechanizmus může operovat na bázi znalostí, které mají sice společnou architekturu, ale mohou být věcně (problémově) orientovány do různých oblastí. Prázdný ES je tak vybaven řídícím mechanismem a všemi ostatními komponentami, pouze báze znalostí je prázdná. Naplněním báze znalostí expertem se z něj stane expertní systém k podpoře rozhodování v oblasti, na kterou jsou znalosti zaměřeny. Prázdné ES se podařilo vyvinout pouze pro řešení diagnostických úloh (s tímto druhem ES se bude tato práce nadále zabývat) [9].

2 PRAKTICKÁ ČÁST

Tato kapitola se zabývá popisem problému se startem PC a následné vytvoření expertního systému v Java Eclipse na toto téma. Součástí této kapitoly je také popis a ovládání programů *Expertní systém* a *Editace databáze*. Jsou zde probrány problémy se startem počítače, průchod otázkami, které bude program využívat pro výsledné řešení situace, jejich tvoření a přidání do vytvořeného programu. Je zde probrána i logika řídícího mechanizmu, který využívá vytvořený ES.

Praktická část je rozdělena na tři hlavní části, popis problému se startem PC, návrh a vytvoření databáze (báze znalostí a báze dat) pomocí programu *Editace databáze*, kterou bude následně využívat naprogramovaný prázdný ES (program *Expertní systém*). Tím pádem ES může být zaměřený na více problémů, pokud bude vytvořených více databází. Není tedy tento ES omezen jen na jednu databázi tím, že jsou data uložená v souboru a ne přímo vložená do řídícího kódu, kde by byla těžká modifikace dat a ES by byl naprogramován jen pro jeden daný problém.

2.1 Popis problému

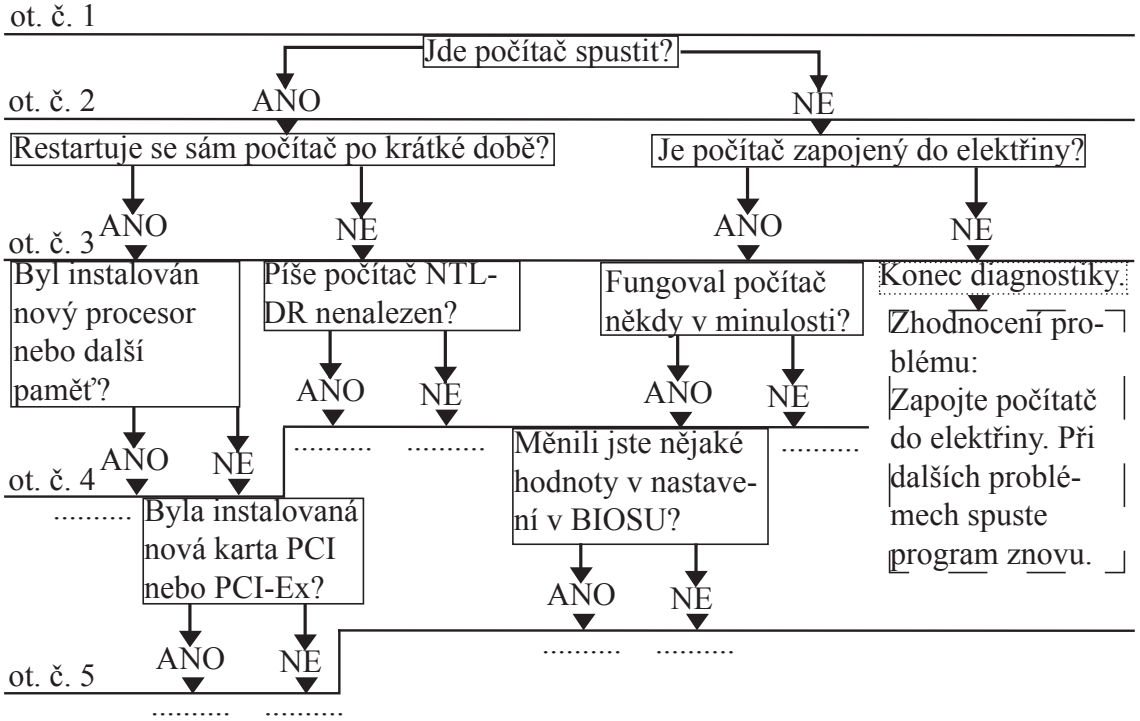
Tato část bude sloužit k bližšímu popisu problémům se startem počítače. Určitě se již každý setkal s problémem proč počítač nenaběhl správně. Většina lidí to řeší tak, že hned počítač odnesou do odborného servisu a zbytečně zaměstnávají lidi, kteří mají většinou dost práce se závažnějšími problémy. Často jde o chyby, které jsou opraveny během pár minut a hlavně je schopen opravit tyto chyby i člověk, který počítačům moc nemusí rozumět. Nebude se zbytečně obracet na odborný servis a tím ušetří čas i peníze, pokud by daný výrobek nebyl už v záruce, nebo by v servisu neprokázali závadu na přístroji. V horším případě lidi počítač ihned vyhodí a shání se po novém. Tato varianta je celkem nesmyslná, protože s počítačem může být hodně problémů a nemusí jít o tak závažný problém, že by počítač už nikdy nešel rozběhnout. Pro výrobce ***hardwaru*** (HW) by toto řešení problému bylo nejlepší. Hodně lidí si toto řešení nemůže dovolit, proto mají snahu opravit počítač vlastní silou bez vydání zbytečných financí na opravu. Tyto problémy řeší vytvořený ES v této práci. Je snaha pomoc lidem, kteří řeší problém se špatným spouštěním počítače.

Při bližším pohledu na daný problém se startem počítače se začíná jednoduchou otázkou: „Jde počítač spustit?“. Velice jednoduchá otázka díky své odpovědi umožní diagnostiku problému rozdělit na dvě hlavní větve.

První větev se bude zabývat diagnostikou problému, že daný počítač lze spustit, avšak dělá problémy až po samostatném spuštění. Zde problémy mohou nastat na mnoha místech jako jsou např. špatné nastavení ***vstupně výstupního systému*** (BIOS), poškození zavaděče ***operačního systému*** (OS), chybný celý OS, poškození

pevného disku (HDD) a mnoho dalších. K dalšímu větvení k řešení problému následuje otázka: „Restartuje[1] se sám počítač po krátké době?“. Tato otázka opět rozdělí problémy na další dvě větve podle zacykleného restartu počítače. Pokud uživatel ES odpoví *„ANO“* jde např. o špatné nastavení BIOSu nebo o některou vadnou komponentu.

Druhá hlavní větev se zabývá diagnostikou problému pokud daný počítač nelze ani spustit. Následující otázka pro uživatele ES je také velice jednoduchá: „Je PC zapojené do elektřiny?“. Tato otázka se klasicky uvádí na první místo, pokud nejde počítač spustit. Tím se zjistí jestli počítač má přívod napájení a pokud ne, tak je diagnostika u konce. Některé lidi tato varianta ani nemusí napadnout. Mohlo se např. pohnout s PC a tím mohl vypadnout konektor pro napájení počítače. Následující otázka, „Fungoval počítač někdy v minulosti?“, vystihuje situaci jestli vůbec někdy v minulosti fungoval PC, nebo je to jeho první spuštění. Pomocí této otázky se může diagnostika problému větvit dále. Pokud počítač nikdy v minulosti nefungoval znamená to např. poničený nebo špatně zapojený HW, popř. nekompatibilní HW.



Obr. 2.1: Náznak řešení problému př spuštění počítače.

Tento způsob řešení problému ukázaný na obrázku 2.1 je reprezentován pomocí pravidel (*rules*). Programy používající tuto logiku řízení se nazývají pravidlové. Odtud je název pro pravidlové ES. Ty používají znalosti založené na konkrétních pra-

[1]Restart znamená, že dojde k vypnutí a opětovnému spuštění.

vidlech definovaných v databázi, kterou pak následně využívá rozhodovací mechanizmus v ES. Podobný způsob reprezentace databáze a řídícího mechanizmu je založen na tzv. rozhodovacích stromech. V této bakalářské práci je použit hybrid těchto dvou mechanizmů.

Znalosti v pravidlových systémech jsou reprezentované pomocí pravidel, která mohou mít například takovéto tvary:

- **if** předpoklad **else** závěr,
- **if** podmínka **else** závěr **and** akce,
- **if** situace **else** akce,
- **if** podmínka důsledek 1 **else** důsledek 2.

Obecná podmínka `if` se definuje takto:

if (podmínka) {*tělo1*} **else** {*tělo2*}.

Tato podmínka zní v překladu následovně. Jestliže platí podmínka následovaná za příkazem `if` (podmínka = 1 nebo `true`[2]), tak udělej příkazy definované ve složených závorkách pod jménem *tělo1*. Pokud ovšem bude podmínka vyhodnocena jako nepravda (podmínka = 0 nebo `false`) skočí na řádek kde je příkaz `else` a vykoná definované příkazy ve složených závorkách pod jménem *tělo2*. Příklad pravidel pro diagnostiku problému se startem počítače vypadá následovně:

`if` (Jde počítač spustit? Odpověď = NE){*tělo1* se přeskočí}
`else` {skoč na následující otázku v rozhodování a zapamatuj si odpověď}
`if` (Je počítač zapojený do elektřiny? Odpověď = NE){*tělo1* se přeskočí}
`else` {Jde o konec diagnostiky. Vypiš výsledek: „Zapojte počítač do elektřiny. Po setrvání problému se spuštěním počítače spusťte program znovu“}.

Jiný příklad:

`if` (Startuje auto? Odpověď = NE) {*tělo1* se přeskočí }
`else` {`if` (Svítí světla? Odpověď = NE){*tělo1* se přeskočí }
{*tělo2*: Výsledek je vybitá baterie.}}.

V tomto příkladě je ukázáno, že lze podmínky do sebe libovolně vnořit. Ovšem pak závisí jak jsou do sebe podmínky vnořené a jak se budou postupně vyhodnocovat. Musí se dbát na správný zápis vnořených podmínek.

Většina znalostních systémů je založena na pravidlech, nebo kombinuje pravidla s jiným způsobem reprezentace. Pravidlové systémy se od klasických logických systémů odlišují nemonotonním uvažováním a možností zpracování neurčitosti.

[2]logika pravda (true), nepravda (false)

V této bakalářské práci budou definované jen podmínky pro odpovědi „ano“ a „ne“. S reprezentací neurčitostí se tato práce nezabývá, protože je použit hybridní mechanizmus řídící logiky pomocí rozhodovacích stromů a řídících pravidel.

2.2 Popis programu *Editace databáze*

2.2.1 Návrh databáze

Ještě než bude popsáno uživatelské rozhraní pro vytváření databáze, je nutné si naznačit jakým způsobem se budou ukládat a vyhledávat v databázi záznamy (otázky spolu s ostatními hodnotami pro správnou funkci ES). Detailněji bude popsáno ukládání databáze do souboru a načítání databáze ze souboru v kapitole 2.2.3.

Každá otázka je spolu s pomocnými hodnotami uložená na jednom řádku v daném souboru. Toto nemusí vždy platit. Jeden záznam může být libovolně uložený na více řádcích v souboru. Důležité je to, že každý záznam končí ukončovacími znaky `<konec radku>`. Hodnoty záznamu (číslo otázky, index, text otázky, nápověda, výsledek diagnózy, konec a obrázek) jsou mezi sebou odděleny oddělovacími znaky ␣`<nextItem>`␣, kde znak ␣ zvýrazňuje použití mezery. Tím je docílena jednoznačná struktura databáze, která se ze souboru načte a posléze se použije. Hodnoty jednotlivých prvků jsou probrány v kapitole 2.2.6. Nejpodstatnější je index pro jednotlivý záznam. Ten bude zaznamenávat průchod jednotlivými větvemi databáze z obrázku 2.1. Tímto indexem jde jednoduše určit na jaké otázce se právě uživatel nachází a jaké otázky mohou být v pořadí následující. Pokud je dodržena podmínka jednoznačnosti indexu, lze jednoduše vyhledávat záznamy z databáze prostým porovnáváním hledaného indexu s indexy všech záznamů databáze.

Ukázka sestavení indexu:

1. **„“**: první záznam musí mít index prázdný. Tímto řádkem se bude začínat v databázi a je v něm uložená první otázka, popř. další informace. Tento záznam bude uložen na prvním řádku v souboru.
2. **A**: tento záznam bude uložen hned za prvním záznamem po ukončovacích znacích záznamu (`<konec radku>`). Bude obsahovat záznam co se má dělat, pokud uživatel odpoví na první otázku „ano“.
3. **N**: záznam pro odpověď na první otázku „ne“.
4. **NA**: záznam obsahující informace, pokud se na první otázku odpovědělo „ne“ a na druhou „ano“,
5. **NN**: první otázka „ne“, druhá otázka „ne“.
6. **AA**: první otázka „ano“, druhá otázka „ano“.
7. **AN**: první otázka „ano“, druhá otázka „ne“.

Tímto způsobem lze projít jednotlivé větve rozhodování a přitom si pamatovat všechny indexy již zodpovězených otázek. Bude možnost se v programu vracet na předchozí otázky až k první otázce v databázi. Vždy se bude vyhledávat jen jeden záznam v databázi, na kterém budou uložené potřebné hodnoty (otázka, nápověda, výsledek atd.). Z ukázky sestavení indexu je zřejmé, že index bude růst směrem doprava podle odpovědí na otázky „ano“ nebo „ne“ a bude nabývat pouze hodnot „A“ a „N“. S rostoucím počtem otázek to může být celkem nepřehledné avšak tento způsob pohybu ve stromu otázek byl zvolen jako nejlepší co se týče složitosti algoritmu pro hledání otázek a rychlosti vyhledání následujících otázek. Při editaci databáze se tímto způsobem nemusí dodržovat žádné pořadí záznamu při ukládání. Program ovšem musí hlídat, aby uživatel mohl zadat jen jednoznačný index pro každý záznam.

Nutno ještě dodat k indexu, že záznamy v databázi mohou být libovolně uložené. Nemusí mít pevnou strukturu podle rostoucího indexu např. třetí záznam může být klidně až na konci celé databáze. Jedinou výjimku tvoří první záznam, který má prázdný index a musí být na začátku databáze. Tím se také bude následně začínat v expertním systému.

2.2.2 Použitý model databáze

Než budou probrány jednotlivé komponenty z programu, bude dobré se stručně seznámit s modelem databáze a datovými typy, které jsouv něm použity.

Nejprve musí být vytvořen datový model, který bude zastupovat všechny sloupce z databáze. Bude zde použito sedm datových proměnných do kterých se budou ukládat různé hodnoty. Aby bylo jednoduché oddělení těchto prvků od sebe, byla použita struktura obsahující všechny prvky vystupující pod třídou `Item`. Struktura je speciální datový typ obsahující více proměnných, které můžou nabývat různými hodnotami.

Příklad naprogramování třídy `Item` obsahující strukturu prvků.

```
public class Item {
  private String c_otazky;      //číslo otázky v pořadí
  private String index;         //index v databázi
  private String text_otazky;   //text otázky
  private String napoveda;      //nápověd k otázce
  private String vysledek;      //výsledek diagnostiky
  private boolean konec;        //jestli je konec diagnostiky
  private String obrazek;       //cesta k obrázku ze složky MEDIA
}
```

Z kódu je zřejmé, že většina prvků bude typu `String`[3] a jeden typu `boolean`[4]. U čísla otázky je typ `String` použit kvůli tomu, že pokud by to byl datový prvek typu `int`[5] musela by být konverze z typu `int` na typ `String` při ukládání databáze, zobrazování hodnot do tabulky a zobrazování hodnot do editačního okénka. Zpětná konverze by potom musela být u načítání databáze a při ukládání zadaných hodnot do datového prvku.

V kódu je použit příkaz `private`. Ten říká, že se nesmí k datovým prvkům přistupovat přímo jen v rámci dané třídy. Proto je nutné vytvořit tzv. *kontruktor* s metodami *Getters* a *Setters*, které nám zajistí přístup k datovým prvkům. Naopak veřejný přístup se definuje příkazem `public`.

Konstruktor je zvláštní metoda volaná při vytváření nové instance dané třídy. Má stejný název jako třída a nevrací žádnou hodnotu (nemá typ, ani void). V této databázi je použita na přidání nového záznamu.

Metoda *Getters* je metoda pro čtení dané proměnné. Vrací tedy hodnotu dané proměnné. Zatímco metoda *Setters* slouží pro zápis do proměnné.

[3]Datový typ do kterého lze zapsat všechny znaky, včetně tzv. bílých znaků (mezery, tabelátory, nové řádky atd.)

[4]Tento typ může nabývat pouze dvou hodnot,„true“(pravda) a „false“(nepravda). Tento datový typ je použit na signalizaci konce diagnostiky

[5]Typ který nabývá pouze celočíselnými čísly v omezeném rozsahu.

Příklad naprogramování *kontruktoru* s metodami *Getters* a *Setters* v třídě `Item`.

```
public Item(String c_otazky, String index, String text_otazky,
            String napoveda, String vysledek, boolean konec,
            String obrazek) {
  this.c_otazky = c_otazky;
  this.index = index;
  this.text_otazky = text_otazky;
  this.napoveda = napoveda;
  this.vysledek = vysledek;
  this.konec = konec;
  this.obrazek = obrazek;
}//konstruktor třídy Item

public String getC_otazky() { //metoda Getters pro čtení otázky
  return c_otazky;            //vrací hodnotu čísla otázky
}
public void setC_otazky(String c_otazky) { //metoda Setters na
  this.c_otazky = c_otazky;                //ukládání otázky
}
public boolean isKonec() {      //speciální typ metody Getters
  return konec;                 //vrací true nebo false
}
public void setKonec(boolean konec) { //nastaví true nebo false
  this.konec = konec;
}
```

V příkladu je uveden pojem `this`. Tento pojem ukazuje na proměnné ze struktury v třídě `Item`. Speciální případ metody *Getters* nastává u datového typu `boolean`, kde nevrací znaky, ale jen logické hodnoty „true“ nebo „false“.

Tento datový model stačí na ukládání všech sloupců [6], ale pouze v rámci jednoho záznamu (řádku). Musí se tedy naprogramovat ještě datový typ pole, který bude obsahovat tuto strukturu, aby bylo možné ukládat i více záznamů této struktury. Bude to taková matice, která bude mít na řádky datový typ pole `List` a na sloupce bude mít datový typ struktury `Item`. Popis jednotlivých proměnných ve struktuře `Item` je v kapitole 2.2.6.

[6] V příkladu není datový model kompletní pro celou databázi. Celý výpis by zde byl zbytečný, protože má zbytek proměnných stejné metody *Getters* a *Setters* jako proměnná `c_otazky`.

Příklad naprogramování třídy `Databaze`, která bude obsahovat jednotlivé záznamy.

```
public class Databaze {
  private List <Item> data;      //vytvoření databáze

  public Databaze() {            //konstruktor,
    data=new ArrayList<Item>();  //který vytvoří pole ArrayList
  }
  public boolean add(Item e) {   //přidá strukturu do pole,
    return data.add(e);          //čili přidá další záznam
  }
  public Item get(int index) {   //vrací jednotlivý záznam
    return data.get(index);      //(strukturu) na určitém indexu
  }
  public int size() {            //vrací velikost datábaze
    return data.size();
  }
  public void del(int index) {   //maže řádek daný indexem
    data.remove(index);
  }
  public void delAll() {         //smaže celou databázi
    for (int i = databaze.size(); i > 0; i--)
    data.del(i-1);  //i-1 proto, že metoda size() vrací velikost
  }                 //databáze od 1 a index databáze začíná od 0
  public boolean isEmpty() {  //vrací jestli je databáze prázdná
    return data.isEmpty();    //pokud ano vrátí tato funkce true
  }
}
```

Kontejner typu `List` má výhodu v tom, že pokud je smazán určitý záznam v databázi, tak se celé pole přeskládá, aby se zaplnila mezera po vymazaném řádku. Má řadu dalších výhod jako je například, že stejný prvek může být v kontejneru vícekrát a má určenu jednoznačnou polohu indexem [7]. Další výhody jsou: rychlost přidávání, odebírání a prohledávání záznamů v kontejneru. Největší výhoda tkví v tom, že je to pole s proměnnými rozměry. To znamená, že se nemusí zadávat počáteční velikost pole při vytváření[8]. Pole se automaticky zvětšuje při přidání hodnoty a zmenšuje při odebrání hodnoty.

[7] Tento index je pouze celočíselný a začíná od 0.

[8] Např. v programovacím jazyku C/C++ se musela předem zadat velikost pole nebo se pro pole musel složitě dynamicky alokovat paměťový prostor.

U metody `delAll` je použit cyklus `for`. Obecný zápis tohoto cyklu je následující:

`for(počáteční hodnota; podmínka cyklu; inkrementace/dekrementace)`,

kde počáteční podmínka v tomto případě je `i = databaze.size()`, podmínka cyklu `i > 0` vyjadřuje ať se cyklus opakuje dokud nesmaže všechny data z databáze, dále je použita dekrementace[9] o jedničku. Obecně místo inkrementace[10]/dekrementace se může použít jakákoliv matematická rovnice např. `i = i +3`, kde v každém cyklu se zvýší proměnná `i` o 3.

2.2.3 Ukládání databáze do souboru a načítání databáze ze souboru (třída *Databaze*)

Zde jsou vysvětleny algoritmy ukládání databáze do souboru a načítání databáze ze souboru.

Příklad ukládání databáze do souboru pomocí `BufferedWriter` ze třídy `Databaze`.

```
public void saveItems(String fileName) {
  BufferedWriter bufferedWriter = null;
    try {
    bufferedWriter = new BufferedWriter(new FileWriter(fileName));
      for (int index = 0; index < databaze.size(); index ++){
        bufferedWriter.write(data.get(index).getC_otazky());
        bufferedWriter.write(" <nextItem> "); //oddělení hodnot
        .
        .
        if(databaze.get(index).isKonec())//pro konec false or true
          bufferedWriter.write("true");
        else
          bufferedWriter.write("false");
        bufferedWriter.write(" <nextItem> ");
        bufferedWriter.write(databaze.get(index).getObrazek());
        bufferedWriter.write(" <nextItem> ");
        bufferedWriter.write("<konec radku>"); //konce záznamu
        bufferedWriter.newLine(); //nový řádek
      }
    }catch(FileNotFoundException ex) {
      System.err.println("Nelze nalézt soubor.");
```

[9]Dekrementace znamená snížení hodnoty o jedničku.
[10]Inkrementace znamená zvýšení hodnoty o jedničku.

```
        return;
      }catch(IOException ex) {
        System.out.println("Chyba při ukládání do souboru");
        return;
    }finally{
        try {
          if(bufferedWriter != null) {
            bufferedWriter.flush();
            bufferedWriter.close();
          }
        }catch(IOException ex) {
           System.out.println("Chyba při ukládání do souboru");
           return;
         }
      }
  }
```

Nejdříve se funkci `saveItems` musí předat název a cesta k souboru, do kterého se budou data ukládat. Soubor se otevře pro zápis vyvoláním třídy `FileWriter`, do které se předá název a cesta k souboru. Dále se vytvoří proměnná `bufferedWriter` ze třídy `BufferedWriter`. Třída `BufferedWriter` byla v použita, protože podporuje znakové ukládání do souboru a urychluje zpracování souboru používáním své vyrovnávací paměti (*buffer*). Pomocí metod *Getters* se z databáze vybírají konkrétní data pro uložení, které se následně metodou `write` zapisují do souboru. Zajímavé je zapisování hodnoty `boolean` do souboru pomocí podmínky `if`, která rozhodne jestli se do souboru zapíše „true“ nebo „false“. Na tomto algoritmu je patrné to, že každá hodnota ze struktury `Item` je oddělena posloupností znaků ␣`<nextItem>`␣ (kde znak ␣ je mezera) a na konci záznamu posloupností znaků `<konec radku>`. Aby se uložily postupně všechny záznamy z databáze musí být celé ukládání struktury v cyklu `for`, který projede postupně všechny záznamy.

Jakmile jsou uložena všechna data z databáze musí se uvolnit vyrovnávací paměť pomocí metody `flush()` a pomocí metody `close()` se zajistí zavření souboru.

Celý tento algoritmus je uzavřen do příkazu `try{}`, který zachytává výjimky, které mohou při provádění algoritmu nastat. Jedná se např. o výjimky: soubor nebyl nalezen `FileNotFoundException`, chyba vstupně-výstupní operace `IOException` a další. Příkaz `catch{}` potom tyto výjimky může zpracovat např. vypsáním chybové hlášky jako v tomto případě, nebo programově obsloužit výjimku. Příkaz `return` ukončuje provádění funkce. Je to v podstatě příkaz na vracení hodnot, ale tato funkce nic nevrací (typ `void`), proto se pomocí tohoto příkazu ukončí funkce.

Načítání dat ze souboru je trochu složitější než ukládání. Při ukládání se jednotlivé hodnoty zapisují postupně do souboru, ale zpětné načítání těchto dat ze souboru už není tak jednoduché, protože se musí data správně načíst do databáze jinak by program nefungoval správně.

Příklad načítání databáze ze souboru pomocí `BufferedReader` ze třídy `Databaze`.

```
public void loadItems(String fileName) {
  String radek;
  String zaznam = "";
  BufferedReader bufferedReader = null;
  try {
     bufferedReader = new BufferedReader(new FileReader(fileName));
      while ((radek=bufferedReader.readLine())!=null) {
        zaznam = zaznam + radek;
        if(!zaznam.endsWith("<konec radku>")){
          zaznam = zaznam + "\r\n";
          continue;
        }
        String [] items=zaznam.split(" <nextItem> ");
        zaznam = "";
        boolean konec;
         if(items[5].equals("true")){
           konec=true;
         }else{
           konec=false;
         }
        this.add(new Item(items[0], items[1], items[2], items[3],
                 items[4], konec , items[6]));
      }
    bufferedReader.close();
  }catch(FileNotFoundException e) {
     System.err.println("Nelze nalézt soubor.");
     return;
  }catch(IOException e) {
     System.out.println("Chyba při čtení ze souboru");
     return;
  }
}
```

Při načítání databáze se začíná stejně jako při ukládání. Nejprve se do metody `loadItems` musí předat název a cesta k souboru ze kterého se budou data číst. Následně se vytvoří dvě proměnné typu `String` na ukládání načtených dat, které budou fungovat jako vyrovnávací paměť pro jednotlivé záznamy. V dalším kroku se vytvoří proměnná `bufferedReader` ze třídy `BufferedReader`, která společně s třídou `FileReader` otevře soubor pro čtení.

Na načítání dat ze souboru je použit cyklus `while`, který má obecný zápis:

`while (podmínka) {tělo cyklu}`,

kde podmínka je v tomto případě `data2=bufferedReader.readLine())!=null`. Tato podmínka se skládá z více příkazů. Nejdříve se provede příkaz pomocí metody `readLine()`, který načte jeden řádek ze souboru a uloží ho do proměnné `radek`. Další příkaz kontroluje jestli už není na konci souboru, porovnáním `radek != null`. Pokud je proměnná `radek` prázdná, tak se podmínka vyhodnotí jako nepravdivá a načítání dat končí. Touto podmínkou je zaručeno načtení celého obsahu až do konce souboru.

Hned po podmínce `while` je příkaz `zaznam = zaznam + radek`. Ten ukládá data ze souboru do proměnné `zaznam` po řádcích (kvůli možnosti existence záznamu, který zabírá v souboru více než jeden řádek). Kompletní záznam se detekuje podmínkou `!zaznam.endsWith("<konec radku>")`, která ho pustí dále ke zpracování. Pokud v proměnné není načten celý záznam (záznam má více řádků a metoda `readLine()` čte pouze po řádku v souboru), detekuje se to v podmínce, která se vyhodnotí jako „true“ a začne vykonávat příkazy v těle podmínky. Zde už je jisté, že záznam je na více řádcích, tak se k němu přidají znaky `\r\n`, které data v databázi odřádkují a přesunou kurzor na začátek řádku. Další příkaz v podmínce je `continue`, ten vyvolává znovu vyhodnocení podmínky. Jednoduše řečeno vyvolá načítání dalšího řádku v databázi. Toto se provádí dokud není v proměnné `zaznam` načten kompletní záznam.

Po detekci kompletního záznamu pokračuje proměnná `zaznam` k dalšímu zpracování. Musí se celý záznam rozdělit na 7 hodnot určených ze struktury `Item`. Při ukládání byly použity oddělovací znaky `␣<nextItem>␣` (kde znak ␣ je mezera), pro oddělení proměnných struktury `Item`. Proto pro načítání se použijí ty samé znaky podle kterých se bude záznam rozdělovat do proměnné `items` (proměnná typu pole `String`ů) pomocí metody `split("␣<nextItem>␣")`, která rozdělí záznam na sedm částí. Po rozdělení záznamu do proměnné `items`, se musí proměnná `zaznam` vyprázdnit, aby bylo možno načítat další data.

Ve funkci `loadItems` je opět podmínka, která rozhoduje, podle přijatých dat u proměnné `konec`, jestli je „true“ nebo „false“. Po rozhodnutí podmínky se může již rozdělený záznam přidat do databáze metodou `this.add()`, kde `this` ukazuje

na databázi do které se ukládají záznamy (třída `Databaze`). Zde je ukázka použití *konstruktoru* ze třídy `Item`. Pomocí klíčového slova `new` se vytvoří jeden záznam do databáze a může se pokračovat s načítáním dalších záznamů až do konce souboru. Pokud je načítání dat u konce, je nutné soubor uzavřít metodou `close()`, která soubor bezpečně uzavře.

Celý tento algoritmus musí být opět uzavřen do příkazu `try()`, na zachycení výjimek v provádění tohoto algoritmu. Příkaz `catch()` potom tyto výjimky může zpracovat.

2.2.4 Uživatelské rozhraní programu (Gui)

V předchozích kapitolách byl probrán návrh a naprogramování databáze, způsoby ukládání databáze do souboru a načítání databáze ze souboru. Tato část se věnuje popisu ***grafického uživatelského rozhraní*** (Gui) pro vytvoření databáze. Jsou zde popsány všechny použité komponenty, jejich vlastnosti a funkce, se kterými se v programu pracuje. Na jednodušší práci s komponenty byl použit plugin[11] *Jigloo*, který používá SWT a Swing knihovny (Návod na stažení a instalaci pluginu *Jigloo* do programu Eclipse je v příloze A.1). Celé uživatelské rozhraní je zobrazeno na obrázku 2.2, kde je vidět rozvržení jednotlivých komponent v programu. Každá komponenta má na tomto obrázku své číslo, postupně zde budou probrány všechny. Některé komponenty, pro jejich složitost, jsou detailněji popsány v samostatných kapitolách.

Popis jednotlivých komponent z obrázku 2.2:

1. **menu** - komponenta `JMenuBar` (popsáno v kapitole 2.2.5),
2. **tabulka** - komponenta `JTable` (popsáno v kapitole 2.2.6),
3. **tlačítka** - komponenta `JTable` (popsáno v kapitole 2.2.7),
4. **co se právě edituje** – komponenta `JLabel`. Zde se vypisují informace co uživatel právě edituje (např. na obrázku je označena buňka tabulky ve sloupci nápověda), aby uživatel neztratil přehled co zrovna edituje. Pokud danou buňku nelze editovat, zde se vypíše hlášení proč to nelze,
5. **editační okénko** - komponenta `JEditorPane` (popsáno v kapitole 2.2.8),
6. **informační výpisy** - komponenta `JLabel`. Zde uživatel vidí, jaká buňka je vybrána v tabulce, čili její příslušný řádek a sloupec. Pro přehled je tam i výpis kolik je záznamů v dané databázi,

[11] Zásuvný modul neboli plugin, také plug-in (neologismus vytvořený z anglického slovesa to plug in – zasunout) je software, který nepracuje samostatně, ale jako doplňkový modul jiné aplikace a rozšiřuje tak její funkčnost.

7. **cesta k obrázku** - komponenta `JLabel`. Výpis názvu obrázku příslušného záznamu ze složky *MEDIA*,
8. **obrázek** - komponenta `JPanel` + třída `Zobrazovac`. Vykreslení obrázku probíhá pomocí třídy `Zobrazovac`, která přijme cestu k obrázku příslušného záznamu a následně ho zobrazí pomocí metody `paint`. Pokud je načten obrázek, je možné ho zvětšit kliknutím na něj. Tím se otevře nové okno, kde se vykreslí zvětšený obrázek. Okno se uzavře opět kliknutím na obrázek nebo kliknutím na křížek vlevo nahoře u okna.

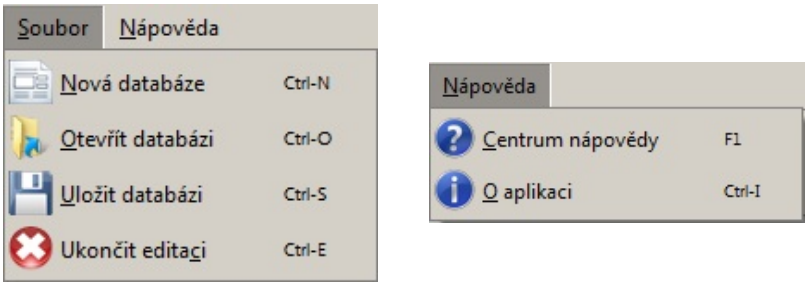
Obr. 2.2: Ukázka vzhledu uživatelského rozhraní pro vytvoření databáze

2.2.5 Popis menu (komponenta *JMenuBar*)

U složitějších aplikací máme potřebu nabídnout uživateli program členit. Vzniká potřeba dávat uživateli možnosti, které by na ovládacích panelech jenom zbytečně zabírali místo. V takovém případě můžeme jednoduše vytvořit menu programu, ve kterém členíme uživatelské volby a nastavení. Pro takovéto menu z obrázku 2.3 musí být vytvořeny následující tři komponenty:

- komponenta `JMenuBar` – tato komponenta sama o sobě žádné menu nevytváří, ale vytvoří pouze `MenuBar`, do kterého se samotné menu vkládá,

- komponenty `JMenu` – komponenty tohoto typu se přidávají do `JMenuBar` (standardně se vkládají vedle sebe od levého horního rohu). V tomto případě nabídka `Soubor` a `Nápověda`. Kliknutím se rozevře další nabídka směrem dolů. Lze jim přiřadit tzv. „horkou klávesu“ `LALT`[12] + libovolná klávesa[13]. Značí se podtržením písmene v textu `JMenu`, např. `Soubor` má klávesovou zkratku `LALT + S`, proto je podtržené písmenko S. Tuto vlastnost může mít většina komponent. Další vlastnosti jsou např. velikost a font písma.
- komponenty `JMenuItem` – tyto komponenty se přidávají do komponent `jMenu`. Jsou to tzv. „klikací“ položky menu, pod kterými se vyvolá událost na kliknutí. Tím se začnou provádět přiřazené funkce. Jdou jim přiřadit klávesové zkratky, horké klávesy, ikony, velikost písma, font písma a mnoho dalších vlastností.
- další – menu může pokračovat dále ve větvení, nabídka se bude standardně rozevírat z komponenty `JMenuItem` směrem doleva.



a) b)

Obr. 2.3: Menu programu *Editace databáze* (komponenta `JMenu` + komponenty `JMenuItem`): a) menu soubor), b) menu nápověda.

V následujících částech jsou probrány jednotlivé funkce položek ze menu programu *Editace databáze*.

Funkce při kliknutí na položku `Nová databáze`

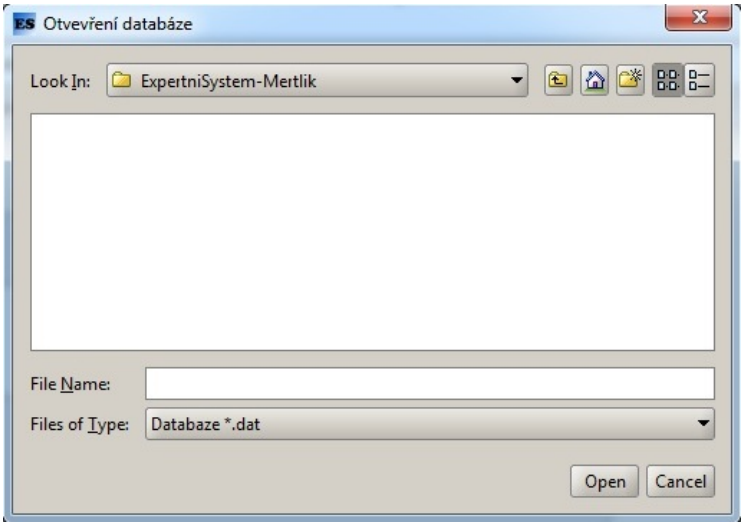
Při kliknutí na tuto položku v menu se otevře varovné okno, které varuje uživatele před smazáním aktuální databáze. Pokud uživatel potvrdí smazání (kliknutí na tlačítko `Yes`), smaže se celá databáze použitím metody `delAll()`. Po smazaní databáze se musí aktualizovat tabulka použitím metody `updateUI()`. Po aktualizaci tabulky je ještě nutné nastavit program do výchozích hodnot metodou `defaultniHodnoty()`, která zajistí např. skrytí některých tlačítek, vymazání textu z komponent `JLabel`, nastavení pozadí editačního okénka na šedou atd.

[12] LALT znamená klávesu levý ALT.

[13] Pokud bude vybrána klávesa, která se v textu komponenty JMenu nebude vyskytovat, tak nebude nic podtrženo, ale je možno takovouto klávesu přiřadit.

Funkce při kliknutí na položku `Otevřít databázi`

Funkce této položky, z menu, je zajistit správné otevření databáze. Po kliknutí se otevře okno na obrázku 2.4 (komponenta `JFileChooser`). Skrze tuto komponentu si lze vybrat soubor z nějakého umístění na počítači. Podmínka je, aby soubor měl příponu .dat[14], jinak soubor nepůjde otevřít a načítání se ukončí varovnou hláškou o chybě. Po vybrání souboru a kliknutí na tlačítko `Open`, `JFileChooser` vrátí název souboru metodou `getSelectedFile()` a metodou `getAbsolutePath()` vrátí cestu k souboru. Nyní se smaže databáze metodou `delAll()` a načte se nová databáze metodou `loadItems()` (metoda je popsána v kapitole 2.2.3), které se musí předat název a cesta k souboru, který se má načíst. Po načtení dat se vyvolá aktualizace tabulky metodou `updateUI()` a vypíše se informace, že databáze byla úspěšně načtena. Pokud se při načítání souboru stane někde chyba, načítání se ukončí a vypíše se chybová hláška proč došlo k chybě.



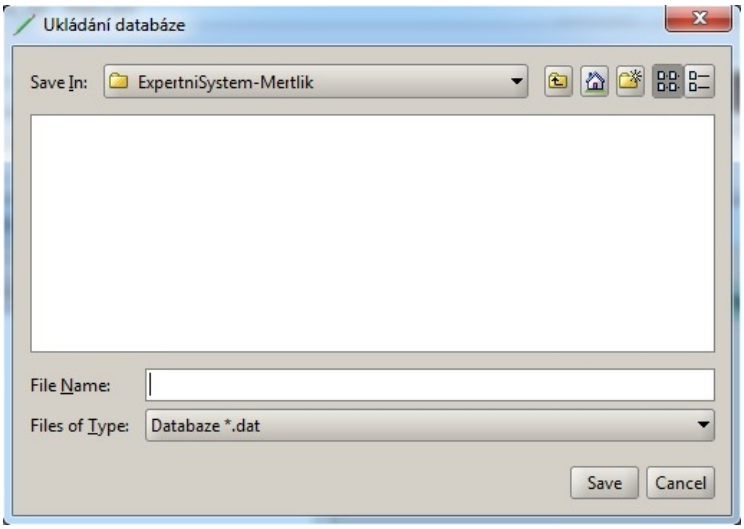
Obr. 2.4: Výběr pro načtení databáze (komponenta JFileChooser)

Funkce při kliknutí na položku `Uložit databázi`

Tato funkce spočívá v ukládání databáze do vybraného souboru. Po kliknutí se otevře okno (obr. 2.5), kde se zadá název souboru a jeho umístění v paměti počítače. Pomocí komponenty `JFileChooser` a jeho metody `getSelectedFile()` se zjistí jaké jméno souboru bylo zadáno. U jména se kontroluje jestli končí příponou .dat (podmínka `if (!cesta.endsWith(".dat"))`). Pokud zadaný soubor nekončí příponou

[14]Přípona která udává, že soubor byl vytvořen programem Editace databáze.

.dat automaticky se mu přidá (příkaz `cesta = cesta + ".dat"`). Dále se ještě zjišťuje umístění, kde se má soubor vytvořit pomocí metody `getAbsolutePath()`. Pokud program zjistí, že daný soubor již existuje, vypíše varovné hlášení, že se uživatel chystá soubor přepsat. Naopak pokud soubor neexistoval, tak se automaticky vytvoří na zadaném umístění.



Obr. 2.5: Výběr pro uložení databáze (komponenta JFileChooser)

Důležité je před uložením databáze zkontrolovat správně zadané uživatelské hodnoty, aby mohl správně fungovat program *Expertní systém*, který bude používat tuto databázi. Hlavní kontrola je kladena na sloupec *Index*, kde se smí zadávat pouze hodnoty „A“ nebo „N“ a nesmí být nikdy prázdný (výjimka nastává u prvního indexu, který musí být vždycky prázdný[15]). Ostatní kontroly nejsou tak důležité. Většina je vyřešena tím, že se uživateli zamezí editace sloupce. Sloupce *č. otázky*, *Nápověda* a *Obrázek* se nemusí kontrolovat. Číslo otázky se generuje v závislosti na délce zadaného indexu a nápověda s obrázkem je u záznamu zadaná nebo ne. Sloupec *Text otázky* se kontroluje v případě, pokud není zadán konec diagnostiky ve sloupci *Konec* a sloupec *Výsledek* se kontroluje pouze pokud je zadán konec diagnostiky. Kontrola těchto dvou sloupců spočívá v tom, že musí být něco zadáno. Více o hodnotách databáze je v kapitole 2.2.6.

Pokud při kontrole databáze nenastala žádná chyba, může se přejít k samotnému ukládání do souboru funkcí `saveItems()` (algoritmus ukládání je popsán v kapitole 2.2.3). Ovšem pokud nastala nějaká chyba při kontrole, databáze se neuloží a vypíše se varovné okno z nalezenými chybami. Jestliže i samotné ukládání do souboru skončí bez chyby, vypíše se informační okno, že byla databáze úspěšně uložena.

[15]Uživateli se zamezí editace prvního indexu v databázi, ale i přesto se radši kontroluje.

Příklad kontroly sloupce *Index*.

```
String indexS = databazeTabulky.get(index).getIndex();
  if(index == 0){                                          //1.
    if(!indexS.isEmpty()){
      hlaska = hlaska + "Na " + (index +1) + ". řádku nemůže
               být zadán index!\n";
    }
  }
  else{                                                    //2.
    if(indexS.isEmpty()){
      hlaska = hlaska + "Na " + (index +1) + ". řádku není
               zadán index!\n";
    }else{                                                 //3.
      int indexCHc = indexS.length();
      char indexCH []=  indexS.toCharArray();
      int err = 0;
        for (int i = 0; i < indexCHc; i++){
          if(!((indexCH[i] == 'A') || (indexCH[i] == 'N'))){
            hlaska = hlaska + "Na " + (index +1) + ". řádku je
                     zadán špatný index!\n";
            err ++;
            break;
          }
        }
        if(err == 0){                                      //4.
          int cetnost = 0;
            for(int l = index; l < databazeTabulky.size(); l++)
              if(databazeTabulky.get(index).getIndex()
                 .equals(databazeTabulky.get(l).getIndex())){
                cetnost++;
                if(cetnost > 1)
                  hlaska = hlaska + "Na " + (index +1) + ". řádku
                           se vyskytl konflikt s " + (l+1) + ".
                           řádkem.\n";
              }
        }
    }
  }
```

Jak je vidět na příkladě, kontrola sloupce *index* už je trochu složitější. Kontrola začíná tím, že se do proměnné `indexS` typu `String` uloží hodnota ze sloupce *Index* příslušného záznamu. Není zde uveden cyklus `for`, který projíždí postupně celou databázi. Ostatně je zde uveden číselný index záznamu (proměnná `index` typu `int`, která se navyšuje cyklem `for`), který ukazuje na konkrétní záznam od prvního do posledního. Je zde také vidět proměnná `hlaska` (typ `String`), která je deklarována v jiné části algoritmu. Vypisují se do ní postupně chybové hlášení a pak se všechny najednou uživateli vypíšou.

Popis kontroly je rozdělen do čtyř částí:

1. **Kontrola prvního záznamu** – pomocí podmínky `if(index == 0)` se kontroluje jestli se jedná o první záznam z databáze. Pokud ano, provede se kontrola zda je první index prázdný podmínkou `if`, pomocí metody `isEmpty()`. Pokud není první index prázdný, uloží se do `hlasky` chybová hláška a přejde se na další záznam.
2. **Ostatní záznamy, kontrola zadání** – v této části se kontrolují ostatní záznamy kromě prvního. V tomto případě se zde opět pomocí metody `isEmpty()` kontroluje, zda je vůbec nějaká hodnota zadaná. Pokud není zadaná žádná hodnota, tak se do proměnné `hlaska` uloží chybové hlášení a přejde se na další záznam.
3. **Kontrola znaků „A“ a „N“** – v této části se kontrolují jednotlivé znaky. Vytvoří se proměnná `indexCHc` typu `int`, do které se uloží délka zadaného indexu. Dále se vytvoří pole `indexCH` typu `char`[16], do kterého se uloží řetězec z proměnné `indexS` metodou `toCharArray()`. Proměnná `err` typu `int` je pro detekci chyby v indexu. Postupně se projede index znak po znaku cyklem `for` a kontroluje se zda je zadán znak „A“ nebo znak „N“ podmínkou `if(!((indexCH[i]=='A')||(indexCH[i]=='N')))`, kde příkaz `||` znamená klíčové slovo „nebo“. Pokud se nebude jednat v indexu o znak „A“ nebo o znak „N“ podmínka se vyhodnotí jako pravdivá. Začnou se tedy provádět příkazy definované v podmínce. Do proměnné `hlaska` se uloží chybové hlášení, inkrementací (++) se navýší proměnná `err` na hodnotu 1 a příkazem `break` se nuceně ukončí cyklus `for` (pokud to najde chybu, tak je zbytečné projíždět zbytek indexu).
4. **Kontrola četnosti výskytu stejného indexu** – jak již několikrát bylo řečeno, index musí být v celé databázi jedinečný. Nesmí se tedy vyskytnout

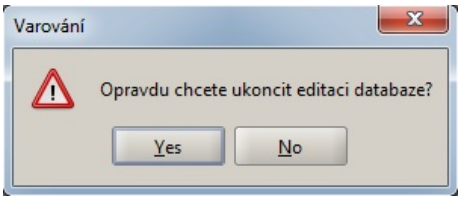
[16]Tento typ je podobný typu STRING až na to, že typ STRING ukládá text jako celek (řetězec znaků), zatímco typ CHAR ukládá text po jednotlivých znacích (pole znaků). Pole typu CHAR je výhodné v tom, že lze vybírat určité znaky podle celočíselného indexu. Číslování opět začíná od nuly.

stejné indexy u různých záznamů. Pokud index prošel předchozími kontrolami bezchybně (není prázdný a obsahuje pouze znaky „A“ a „N“) musí se zkontrolovat ještě jeho četnost v databázi. V algoritmu je inicializovaná proměnná `err`, která naznačuje, že pokud nastala chyba v kontrole znaků, nemusí se prohledávat jeho četnost. Pokud ovšem chyba nenastala proměnná `err` bude mít hodnotu 0 a podmínka `if(err == 0)` ho pustí do kontroly četnosti výskytu. Četnost výskytu se provádí pomocí cyklu `for`, který projede databázi od kontrolovaného záznamu do konce databáze[17] a pomocí podmínky `if` se porovnávají indexy s kontrolovaným indexem. Pokud se vyskytne kontrolovaný index v databázi vícekrát než jednou[18] inkrementuje se proměnná `cetnost` a v dalším kroku se podmínka `if(cetnost > 1)` vyhodnotí jako pravdivá. V těle podmínky se uloží do proměnné `hlaska` chybové hlášení a pokračuje se s prohledáváním databáze, zda není více konfliktů s kontrolovaným indexem.

Na příkladu je vidět, že podmínky jsou do sebe vnořené. Pokud se najde chyba v kontrolovaném indexu, dále se v kontrole zbytečně nepokračuje. Index musí projít všemi čtyřmi kontrolami bezchybně, aby byl vyhodnocen jako vyhovující.

Funkce při kliknutí na položku `Ukončit editaci`

Tato položka z menu vyvolá otevření varovného okna (obr. 2.6), které varuje uživatele před ukončením programu. Pokud uživatel klikne na tlačítko `Yes` ukončí se program systémovou metodou `System.exit(NORMAL)`, kde hodnota `NORMAL` značí normální ukončení programu. Lze nastavit různé hodnoty ukončení, např. ukončení programu s chybu.



Obr. 2.6: informační okno ukončení programu

Funkce při kliknutí na položku `Centrum nápovědy`

Při kliknutí na tuto položku se otevře okno (obr. 2.7), kde se zobrazí stručná nápověda na ovládání programu Editace databáze. Nápověda je tzv. „klikací“ HTML, která se načte do komponenty `JEditorPane` pomocí metody `setPage(helpURL)`,

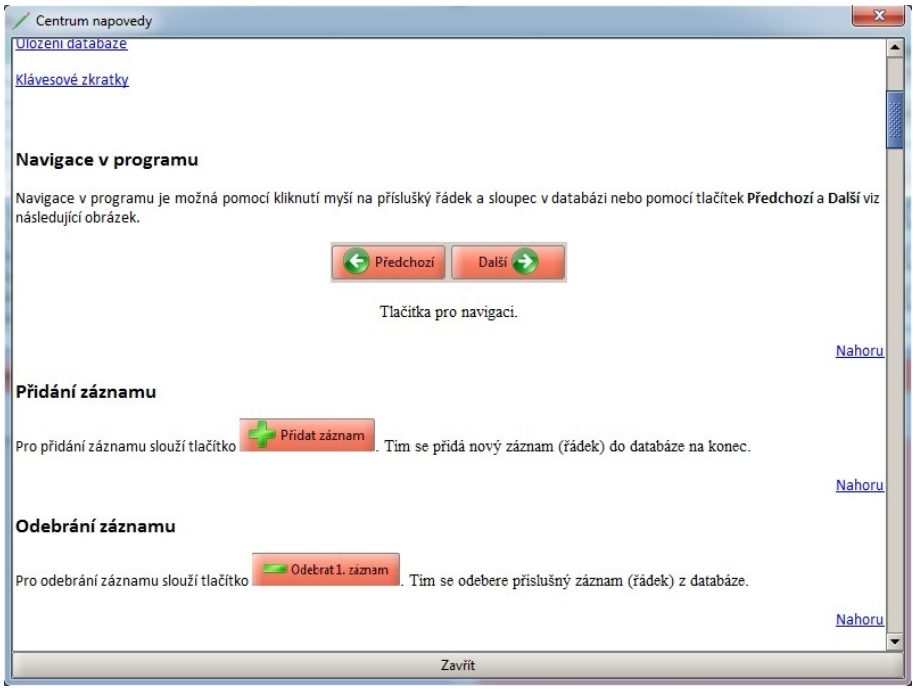
[17] kvůli výpisu chybové hlášky, aby se vypsal jeden konflikt pouze jednou hláškou
[18] zaznamená sám sebe

kde proměnná `helpURL` je typu `URL` a ukazuje na HTML stránku pro nápovědu, která je ve složce *help*. Pokud soubor s nápovědou nebude k dispozici, vypíše se varovné okno, že program nemohl najít soubor s nápovědou.

Aby správně fungovaly klikací odkazy v komponentě `JEditorPane`, je nutné zakázat editaci komponenty pomocí metody `setEditable(false)` a nastavit obsah na typ HTML metodou `setContentType("text/html")`. Klikací odkazy (anglicky hyperlink) se v komponentě `JEditorPane` musejí zapnout pomocí události `addHyperlinkListener`, která obslouží kliknutí na odkaz. Funkce `hyperlinkUpdate` zjistí na jaký odkaz uživatel klikl a co se má po kliknutí odehrát (např. odkaz na jinou stránku nebo jinou část stejné stránky). Potom se opět pomocí metody `setPage` vypíše obsah.

Příklad zapnutí klikacích odkazů v komponentě `JEditorPane`.

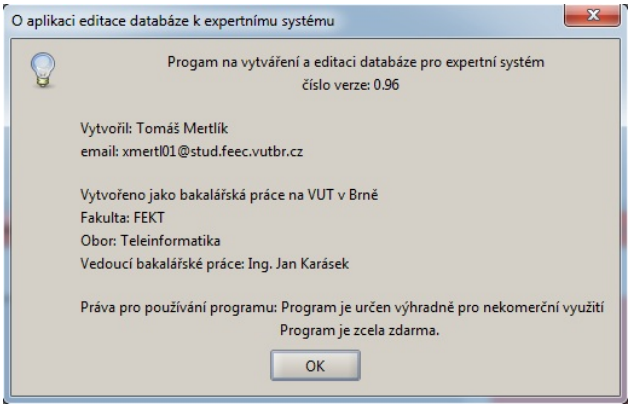
```
EP_napoveda.addHyperlinkListener(new HyperlinkListener(){
  public void hyperlinkUpdate(HyperlinkEvent r){
  if(r.getEventType() == HyperlinkEvent.EventType.ACTIVATED)
    EP_napoveda.setPage(r.getURL());
  }
});
```



Obr. 2.7: Okno pro nápovědu (komponenty JPanel, JButton, JEditorPane.)

Funkce při kliknutí na položku `O aplikaci`

Po kliknutí na tuto položku z menu se objeví informační okno s informacemi o programu (obr. 2.8).



Obr. 2.8: informační okno o aplikaci

2.2.6 Popis tabulky (komponenta `JTable`) a jednotlivých hodnot ze třídy `Item`

Tato část se zabývá popisem proměnných ze třídy `Item` (kapitola 2.2.2) reprezentovaných v tabulce na obrázku 2.2. Dále je zde uveden popis a funkce tabulkového modelu, který je zapotřebí vytvořit pro správnou synchronizaci databáze a tabulky.

Následující skupiny popisují jednotlivé proměnné ze třídy `Item` (záhlaví tabulky).

- **Číslo otázky** - udává číslo otázky v databázi, která bude položena uživateli. Toto číslo se generuje v závislosti na délce indexu a přičítá k tomu jednička [19]. Výjimkou je první index, který nelze editovat a je automaticky nastaven na prázdnou hodnotu, tudíž číslo otázky bude 1.
- **Index** - index byl popsán v kapitole 2.2.1. Index je nutno vždy zadat, výjimku ovšem tvoří první záznam, kde nelze index editovat a tudíž zůstává prázdný.
- **Text otázky** - zadání otázky pro uživatele, která se mu bude zobrazovat a on na ní bude moct odpovídat. Pokud není určen konec diagnostiky ve sloupci *konec*, musí být otázka zadána. Pokud je ovšem zadán konec diagnostiky, nesmí se otázka zadávat (program nepovolí zadání otázky, popř. pokud je vyplněna, tak ji vymaže).

[19] Přičítání jedničky je kvůli tomu, aby číslo otázky začínalo od jedné. Jak již bylo napsáno, tak je to v závislosti na délce indexu. První index má nulovou délku (prázdný index) a proto se přičítá jednička ke každé délce indexu. Délka indexu se zjistí metodou `index.length()`, která vrátí hodnotu počtu znaků v daném indexu.

- **Nápověda** - pokud je uživatel odborník v daném oboru nebude potřebovat rozsáhlé nápovědy a stačí mu pouze jasně položená otázka. Pro laiky v oboru bude lepší mít nápovědu k otázce. Nemusí to být zadáno u všech otázek. Nápověda tvoří pomoc při odpovídání na položenou otázku. Pokud si uživatel není jistý jednoznačností otázky, nebo pokud se objevují v otázce pojmy, které běžný uživatel počítače nemusí znát, může si zobrazit nápovědu k dané otázce (pokud je k dispozici). Nápověda se nemusí zadávat, ale je dobré ji zadat pro lepší pochopení otázky.
- **Výsledek** - slouží pro výslednou diagnostiku problému popř. stručný popis jak odstranit diagnostikovaný problém. Tato hodnota může být editována pouze tehdy, když je zadán konec diagnostiky ve sloupci *konec*.
- **Konec** - zde je zaškrtávací políčko, které se edituje přímo v tabulce. Hodnota zaškrtnutí (*true, pravda*) signalizuje konec diagnostiky a na příslušném záznamu musí být zadán výsledek diagnostiky, který se uživateli vypíše.
- **Obrázek** - tento sloupec obsahuje cestu k obrázku, který se bude uživateli v programu *Expertní systém* zobrazovat. Uživatel může vybrat obrázek z nějakého umístění na počítači do databáze. Z důvodu přenositelnosti databáze s obrázky se automaticky nakopíruje vybraný obrázek do složky *MEDIA* [20], která se potom bude přenášet s programem a tím se zaručí přenositelnost celého programu. Z této složky databáze bude používat obrázky k daným záznamům.

Příklad modelu tabulky z programu *Editace databáze* (třída `MyTableModel`) je uveden v příloze A.2. Na tomto příkladu jsou uvedeny nejdůležitější metody co musí tabulkový model obsahovat. Tento model definuje chování tabulky a zacházení s daty, které obhospodařuje. Pro názvy hlaviček sloupců (nultý řádek tabulky) slouží proměnná `columHeader`, která je typu pole `String`. Dále je vidět, vytvoření databáze pomocí konstruktoru `MyTableModel()`. Pro vykreslení tabulky a práci s daty si sama tabulka volá metody jako jsou:

- `getColumnCount()` - tato metoda vrací počet sloupců. Spočítá si kolik hodnot je zadaných v proměnné `columnHeader`,
- `getRowCount()` - vrací počet záznamů (počet řádků) v databázi. Implicitně po spuštění je tato hodnota nastavena na nulu (prázdná databáze), ale s přibývajícím počtem záznamů se toto číslo zvětšuje,
- `getColumnName(int columnIndex)` - tuto metodu si tabulka volá pro vykreslení hlavičky sloupců. Metoda vrací název hlavičky na hledané pozici, kterou metodě předáme (proměnná `columnIdex`),

[20] Složka určena pro obrázky k databázi. Pokud není složka vytvořena, vytvoří se automaticky sama, aby se do ní mohly ukládat obrázky potřebné k databázi

- `getColumnClass(int columnIdex)` - zde si tabulka zjistí jakého typu jsou sloupce, které obsahuje. V tomto případě je všechno typu `String`, kromě pátého sloupce (sloupec konec), který je typu `boolean` (hodnoty true a false),
- `isCellEditable(int rowIndex, int columnIndex)` - pomocí této metody si tabulka nastaví editaci buněk přímo v tabulce. V tomto případě je opět vše zakázáno, kromě pátého sloupce (sloupec konec). Proměnné `rowIndex` a `columnIdex` ukazují na příslušný řádek a sloupec v tabulce. Zde se neprovádí editace hodnot přímo v tabulce, ale pomocí editačního okénka (kap. 2.2.8),
- `getValueAt(int rowIndex, int columnIndex)` - tato metoda spolupracuje s databází a vypisuje hodnoty do příslušné buňky v tabulce pomocí proměnných `rowIdex` a `columnIndex`, které ukazují na určitou buňku,
- `setValueAt(Object aValue, int rowIndex, int columnIndex)` - tato metoda slouží pro ukládání hodnot z tabulky přímo do databáze. Proměnná `aVaule` typu `Object` slouží k přenosu dat z tabulky do databáze z příslušného řádku a sloupce.

2.2.7 Popis a funkce tlačítek (komponenty JButton)

V této části budou probrány funkce jednotlivých tlačítek (komponenty *JButton*) z obrázku 2.2, které se nachází hned pod tabulkou.

Popis tlačítek `Přidat záznam` a `Odebrat záznam`

- **přidat záznam** (obr. 2.9 a) – funkce tohoto tlačítka spočívá v přidání prázdného záznamu na konec databáze metodou `add(new Item( "", "", "", "" , "", false , ""))`. Toto tlačítko je vždy viditelné. Po přidání prázdného záznamu do databáze se musí aktualizovat tabulka metodou `updateUI()`.
- **odebrat záznam** (obr. 2.9 b) – toto tlačítko po kliknutí vymaže vybraný záznam z databáze pomocí metody `del(vybranyRadek)`. Vybraný záznam je modře zvýrazněný v tabulce. Toto tlačítko mění svůj popisek podle vybraného čísla záznamu. Na obrázku 2.9 b je vidět, že je vybrán první záznam z databáze = první řádek). Po kliknutí na tlačítko se vyvolá rozhodovací podmínka `if`, kde se rozhodne o tom jestli je vybrán první záznam nebo kterýkoliv jiný. Při mazání prvního záznamu se uživatele zeptá jestli ho chce opravdu vymazat a to z důvodu, že se z druhého řádku, který nahradí ten první [21], odebere index (index prvního záznamu musí být prázdný). Zde se opět musí provést aktualizace tabulky po odebrání záznamu z databáze metodou `updateUI()`.

[21] Po vymazání určitého záznamu se celá databáze přeskládá, aby se vyplnilo volné místo po vymazaném záznamu.

U tohoto tlačítka je zajímavé, že po vymazání je program uměle nastaven do defaultních hodnot[22]. Je to nastavené z toho důvodu, aby uživatel nevymazal celou databázi pouhým klikáním na toto tlačítko, ale aby si vždy musel vybrat záznam, který se má vymazat.

a) b)

Obr. 2.9: První dvě tlačítka: a) přidání záznamu na konec, b) odebrání příslušného záznamu

Popis tlačítek `Předchozí` a `Další`

- **předchozí** (obr. 2.10 a) – po kliknutí se vybere buňka [23], nalevo od buňky, která byla označena. Z tohoto popisu vyplývá, že se postupně vybírají hodnoty ze všech záznamů směrem od konce na začátek databáze. Toto tlačítko je klikací pokud je před vybranou buňkou buňka jiná. Pokud narazí na začátek databáze, tak nepůjde tlačítko zmáčknout. V tabulce probíhá výběr pomocí metody `changeSelection(radek, sloupec)`.
- **další** (obr. 2.10 b) – toto tlačítko má opačnou funkci než tlačítko `Předchozí`. Posunuje výběr buňky od začátku do konce databáze, tedy vybírá se buňka napravo od buňky, která byla označena v tabulce. Na tlačítko jde kliknout pokud není vybraná buňka na konci databáze. V opačném případě nelze klikat. Výběr buňky v tabulce se opět mění metodou `changeSelection(radek, sloupec)`.

a) b)

Obr. 2.10: Druhá dvě tlačítka: a) předchozí buňka, b) další buňka

Popis tlačítek `Přidej obrázek`, `Smaž obrázek` a `Změň obrázek`

Pro funkci těchto tlačítek musí být vždy vybrán příslušný záznam v tabulce na který se budou aplikovat funkce těchto tlačítek. Pokud není žádný záznam vybrán, tlačítka zešednou a nebude možno je zmáčknout.

- **přidej obrázek** (obr. 2.11 a) – k tomuto tlačítku je přidružena funkce přidání obrázku k příslušnému záznamu. Po kliknutí se otevře okno (obr. 2.12), kde si

[22] hodnoty po spuštění programu
[23] Buňka je výraz pro vybraní řádek a vybraný sloupec v tabulce

uživatel vybere nějaký obrázek, který bude u záznamu použit. Jelikož mohou být vybrané obrázky kdekoliv uložené, musí se zajistit přenositelnost databáze. Proto je nutné obrázky, které jsou v databázi použité, nakopírovat do speciální složky. Tato složka se potom musí pro zobrazení obrázků přenášet s databází. Jako nejrozumnější bylo tuto složku pojmenovat *MEDIA*. Pokud tato složka není u programu vytvořena (ověření podmínkou `media.exists()`), vytvoří se sama pomocí `media.mkdir()`, kde proměnná `media` je typu `File`[24] a nese v sobě název složky *MEDIA*[25]. Po vytvoření této složky se může přistoupit ke kopírování. Na zkopírování obrázku byla vytvořena funkce `copyFile()`, která přijme soubor pro zkopírování a vrací zda operace byla provedena úspěšně či nikoliv. Po neúspěšném kopírování se zobrazí chybová hláška. Toto může nastat u obrázků, které uživatel nemá právo číst nebo se může jednat o poškozený obrázek. Před samostatným kopírováním se testuje zda již složka *MEDIA* neobsahuje soubor se stejným názvem jako má kopírovaný obrázek. Pokud se vyskytuje, zeptá se uživatele zda-li ho má přepsat. Pokud uživatel zamítne přepsání souboru, funkce kopírování se ukončí a uživatel může pokračovat v práci. Pokud ale povolí přepsání, nebo pokud se ve složce *MEDIA* nevyskytuje soubor se shodným názvem, může se začít kopírovat. Nejprve se musí vytvořit soubor s názvem kopírovaného obrázku ve složce *MEDIA* pomocí metody `createNewFile()`. Po vytvoření se začne kopírovat obrázek bit po bitu (tzv. se vytvoří jeho bitová kopie). Celá tato operace musí být uzavřená do příkazu `try`, aby zachytával výjimky[26] a do příkazu `catch`, který tyto výjimky zpracuje.

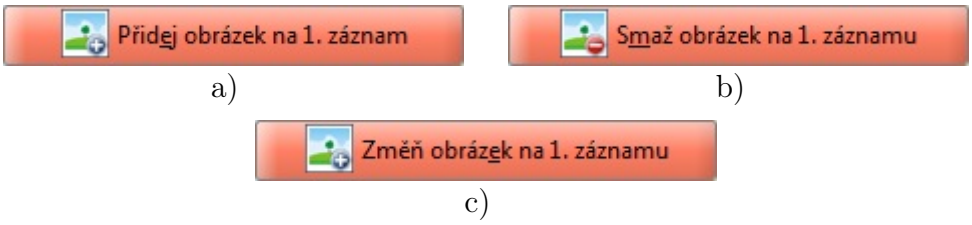
- **smaž obrázek** (obr. 2.11 b) – jak již název tohoto tlačítka napovídá, bude jeho funkce mazání obrázku z vybraného záznamu. Před vymazáním se kontroluje jestli je v databázi obrázek použit vícekrát (u různých záznamů) prostým porovnáváním názvu obrázku z příslušného záznamu s celou databází. Pokud se nalezne jen jeden výskyt v databázi (sám záznam) zeptá se program uživatele, zda chce obrázek smazat i ze složky *MEDIA*, když už nebude v databázi použit. Mazání souboru proběhne pomocí metody `soubor.delete()`, kde proměnná soubor je typu `File` a odkazuje na obrázek ve složce *MEDIA*. Pokud se ovšem najde více výskytů použití obrázku v databázi, ponechá se obrázek ve složce *MEDIA*, aby ho mohly využívat ostatní záznamy ke kterým je přidán.
- **změň obrázek** (obr. 2.11 c) – pokud už na vybraném záznamu je přiřa-

[24]Speciální třída pro práci se soubory a adresáři na počítači.

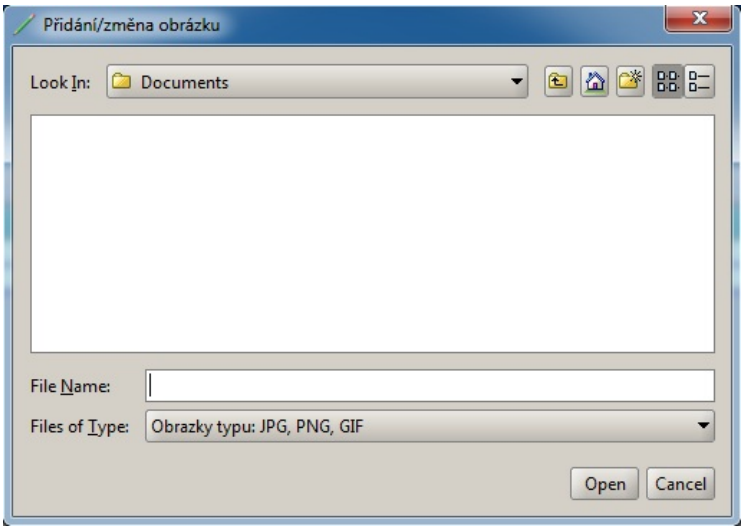
[25]tzv. virtuální odkaz, nejdříve se vytvoří odkaz a pak se testuje zda existuje na paměťovém zařízení.

[26]Jedná se např. o to, že v průběhu kopírování se znemožní kopírování např. odpojením zdrojového souboru.

zen obrázek, tak se tlačítko `přidej obrázek` změní na `změň obrázek`. Toto tlačítko spojuje funkce předešlých dvou tlačítek. Po kliknutí se prověří jestli je vícekrát obrázek v databázi použit. Pokud ano, nechá se. Pokud ne, zeptá se uživatele a popřípadě se smaže. Následně začne stejný postup jako u tlačítka `přidej obrázek`.



Obr. 2.11: Poslední sada tlačítek: a) přidání obrázku do záznamu, b) odebrání obrázku ze záznamu, c) změna již přidaného obrázku v záznamu



Obr. 2.12: Výběr obrázku k záznamu

2.2.8 Popis editačního okénka (komponenta *JEditorPane*)

Touto komponentou lze vepisovat do databáze data (text). Na obrázku 2.13 lze vidět, že editační okénko podporuje i základní formátování textu. Pokud by se text nevešel do tohoto okénka, automaticky se objeví na pravém boku okénka scrollbar[27]. Opět jako předchozí komponenty, je editační okénko závislé na označení buňky v tabulce. Editace příslušné buňky je omezena několika podmínkami. Pokud buňku nelze editovat, vypíše se hláška proč tomu tak je, editační okénko zešedne a zamezí se vepisování hodnot do něj. Jestliže buňku lze editovat, vypíše se jaká buňka je právě označena (editována), editační okénko zbělá a povolí se vepisování hodnot.

Editaci nelze provádět:

1. pokud není označena žádná buňka v tabulce,
2. pokud je označena buňka ze sloupce číslo otázky (číslo otázky se automaticky generuje v závislosti na délce indexu příslušného záznamu),
3. pokud je označena buňka indexu v prvním záznamu (index prvního záznamu musí být prázdný),
4. pokud je zadán konec editace v příslušném záznamu (sloupec konec v tabulce), zamezí se editace textu otázky v tomto záznamu. Pokud tam bylo něco vyplněno, vymaže se to,
5. pokud není zadán konec editace v příslušném záznamu (sloupec konec v tabulce), zamezí se editace výsledku diagnostiky. Pokud tam bylo něco vyplněno, vymaže se to,
6. pokud je označena buňka ze sloupce konec, nelze měnit její hodnotu editačním okénkem. Editace probíhá v tabulce kliknutím na zaškrtávací políčko,
7. pokud je označena buňka sloupce obrázek. Do tohoto sloupce nelze přímo vepisovat odkaz na obrázek. Přidávání odkazů se dělá pomocí tlačítek.

Omezení editace buněk je velice důležitá vlastnost. Díky tomuto omezení se zaručí správná funkce databáze v programu *Expertní systém*. Pokud by nebylo omezení aplikováno, mohl by uživatel zapsat jakoukoliv hodnotu do jakékoliv buňky. To by mohlo vést ke špatné funkčnosti databáze. Např. kdyby uživatel zadal číslo otázky neodpovídající pravému číslu otázky, vypisovalo by se potom špatné číslo otázky v programu *Expertní systém*. Toto by ještě neovlivnilo funkčnost, ale u obrázku by toto mohlo vyvolat řadu komplikací. Pokud používá záznam obrázek, musí být u něj uvedena celá cesta k obrázku ze složky *MEDIA*. Omezení editace bylo navrženo, aby bylo co nejlehčí naprogramování programu *Expertní systém* pro ovládání vytvořené databáze.

[27]Speciální komponenta pomocí níž se lze pohybovat v objektu, pokud překročí jeho velikost.

Zkontrolujte správné zapojení uvnitř oočítače. Můžete podle manuálu ke své základní desce nebo dejte počítač do odborného servisu. Kontrola zapojení se hlavně týká:
1) Napájení základní desky, které vede od zdroje.
2) Správné osazení procesoru a pamětí na základní desku.
3) Správné zapojení tlačítka na vypnutí na počítačové skříni. Viz manuál k základní desce.

Tyto zapojené komponenty stačí k tomu, aby se počítač spustil. Popřípadě zkuste vytáhnout a zandat zpět komponenty ze základní desky.

Pokud potíže přetrvají vyhledejte odbornou pomoc k Vašemu počítači.

Obr. 2.13: Editační okénko

2.3 Popis programu *Expertní systém*

Jak již bylo napsáno v kapitole 2, praktická část se dělí na dvě části. První část byla vytvoření programu *Editace databáze*, který byl popsán v kapitole 2.2. Tato kapitola se zabývá popisem programu *Expertní systém*, který používá databázi vytvořenou v programu *Editace databáze*. Jsou zde probrány algoritmy procházení otázek, popis uživatelského rozhraní, funkce tlačítek a funkce menu.

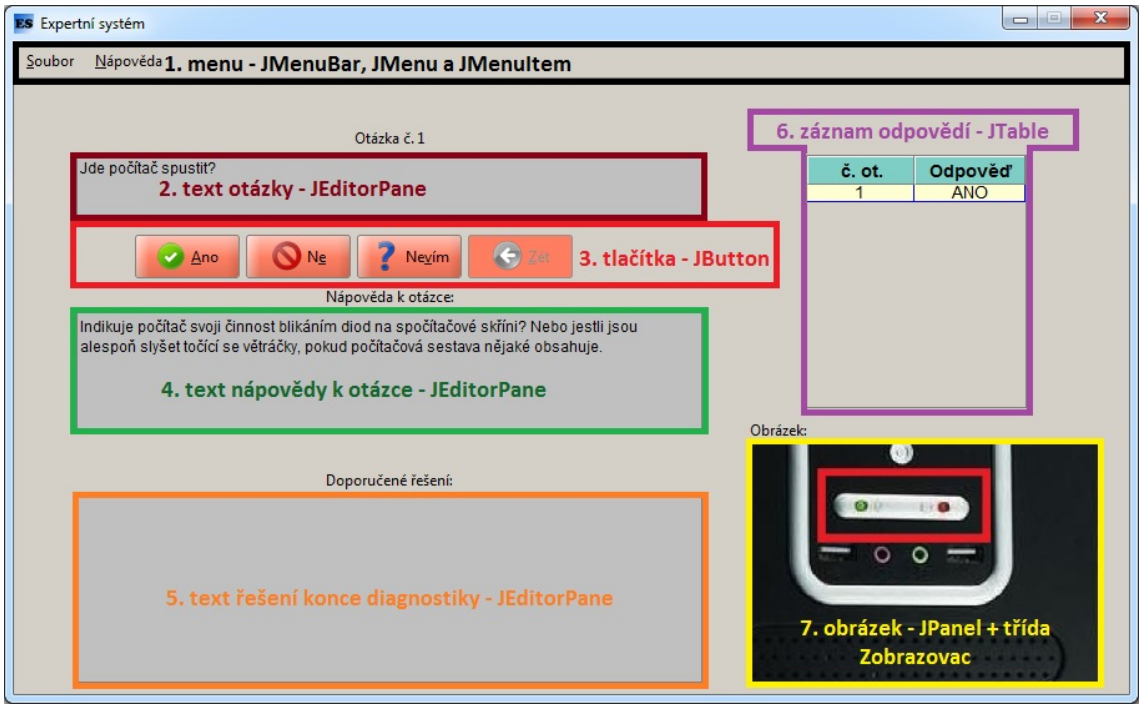
Při vytváření programu *Expertní systém* byla zhodnocena nejen jeho funkčnost, ale i pokus vytvořit uživatelské rozhraní takovým způsobem, aby bylo velmi snadno ovladatelné. Aby spolu programy mohly pracovat, musí se zachovat stejný datový model, který je popsán v kapitole 2.2.1. Jelikož je velice promyšlený návrh databáze, kterou oba programy společně používají, je jednodušší programování obou programů. Obzvlášť programu *Expertní systém*, který databázi jen čte a tudíž nemusí kontrolovat souvislosti v databázi. Proto je jeho programový kód podstatně jednodušší. Vyhledá záznam podle předem daných, napevno zabudovaných, pravidel a zobrazí jeho potřebné části.

Výběr otázky spočívá pomocí indexu, který je popsaný v kapitole 2.2.1. Vybere se tedy vždy jen jeden záznam, který je zrovna potřebný a použijí se hodnoty z tohoto záznamu do programu. V každém indexu je uchováno kde se přesně záznam nachází. Tudíž je jednoduché dohledat záznamy další a předchozí, které byly už použity.

2.3.1 Uživatelské rozhraní (GUI)

Na obrázku 2.14 je znázorněno uživatelské rozhraní programu *Editace databáze*. Jelikož tento program nemá napevno zabudovanou bázi dat a bázi znalostí (tzv. prázdný ES), musí se načíst databáze, se kterou bude následně program pracovat. Uživatel si

načte databázi potřebnou k diagnostice svého problému, v našem případě problém se spuštěním PC, následně se začíná prvním záznamem z databáze, který má prázdný index (první otázka). Postupným odpovídáním na otázky se dojde k závěrnému určení výsledku popř. řešení konkrétního problému.

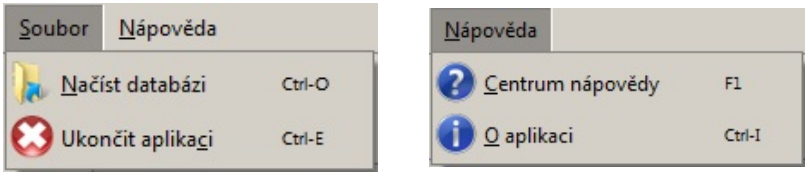


Obr. 2.14: Ukázka vzhledu uživatelského rozhraní programu *Expertní systém*

Popis jednotlivých komponent z obrázku 2.14:

1. **menu** – komponenta `JMenuBar`. Toto menu je stejné jako v programu *Editace databáze* až na pár drobností. Není zde zapotřebí ukládat a zakládat novou databázi. Ostatní položky menu, vyobrazené na obrázku 2.15, jsou funkčně stejné jako v programu *Editace databáze*. Vytvoření a funkce menu je popsáno v kapitole 2.2.5,
2. **text otázky** – komponenta `JEditorPane`. Zde se uživateli zobrazí aktuální otázka na kterou bude odpovídat. Otázky se postupně budou měnit podle průchodu databází v závislosti na indexu záznamu. Pokud je diagnostika již u konce, toto okno se vyprázdní. Nad tímto oknem je komponenta `JLabel`, ve které se vypisuje číslo otázky (na obrázku je znázorněna první otázka).
3. **tlačítka** – komponenty `JButton` (popsáno v kapitole 2.3.2),
4. **text nápovědy k otázce** – komponenta `JEditorPane`. V tomto okně se upřesňuje otázka, nebo některé pojmy z otázky. Toto okno je zobrazeno pouze tehdy, když si uživatel vyžádá pomocnou nápovědu kliknutím na tlačítko `Nevím`,

5. **text řešení konce diagnostiky** – komponenta `JEditorPane`. Tato část slouží k výpisu výsledků diagnostiky a popř. ke stručnému popisu řešení problému,
6. **záznam odpovědí** – komponenta `JTable`. Tato tabulka zaznamenává odpovědi na otázky, které již uživatel zadal,
7. **obrázek** – komponenta `JPanel` + třída `Zobrazovac`. V tomto panelu se zobrazují pomocné obrázky, pokud jsou ovšem k dispozici u daného záznamu. Při kliknutí na panel se obrázek zvětší, při dalším kliknutí se opět zmenší. Funkce je naprosto stejná jako u programu *Editace databáze* a je detailněji popsána v kapitole 2.2.4.



a) b)

Obr. 2.15: Menu programu *Expertní systém* (komponenta JMenu + komponenty JMenuItem): a) menu soubor), b) menu nápověda.

2.3.2 Popis a funkce tlačítek (komponenty *JButton*)

Tlačítka jsou rozdělena do dvou skupin. První skupina souvisí s odpověďmi na zadané otázky. Tyto tlačítka mají podobný algoritmus hledání následujícího záznamu, který bude vzápětí popsán. Druhá dvojce tlačítek obsahuje přídavné funkce, pro zjednodušení práce s programem.

Popis tlačítek `Ano` a `Ne`

- **Ano** (obr. 2.16 a) – funkce tohoto tlačítka spočívá v odpovědění na zadanou otázku „ano“. Toto tlačítko je viditelné jen pokud je možno odpovídat na otázku (tzn. pokud není diagnostika u konce).
- **Ne** (obr. 2.16 b) – toto tlačítko má stejnou funkci jako předchozí až na to, že se na otázku odpoví „ne“.

Jelikož jsou funkce těchto tlačítek skoro stejné, stačí popsat jen jednu funkci a v té druhé budou jen nepatrné změny, které jsou uvedeny v popisu.

a) b)

Obr. 2.16: První dvě tlačítka: a) Pro odpověď na otázku „ano“, b) Pro odpověď na otázku „ne“

Ukázka algoritmu pro hledání potřebného záznamu při odpovědění na otázku „ano“:

```
String novyIndex = predchoziIndex + "A";
  for(int i = 1; i < databaze.size(); i++){
    if(databaze.get(i).getIndex().equals(novyIndex)){
      index = i;
      break;
    }
  }
  int c_otazky=Integer.parseInt(databaze.get(index)
                                .getC_otazky()) - 1;
  databazeTabulky.add(new ItemTabulka( Integer
                        .toString(c_otazky),"ANO"));
  T_tabulka.updateUI();
  ukazData();
  B_zpet.setEnabled(true);
```

V příkladu jsou vidět čtyři proměnné, které zde nejsou deklarované. Tyto proměnné jsou deklarované jako globální, aby s nimi mohlo pracovat více funkcí, bez nutnosti předávání hodnot proměnných. Proměnná `predchoziIndex` je typu `String` a obsahuje index předchozího záznamu z databáze[28]. Proměnná `databaze` je deklarovaná pomocí třídy `Databaze` a slouží jako globální databáze všech načtených záznamů. Proměnná `index` je typu `int` a obsahuje celočíselný odkaz na určitý záznam z databáze. Poslední globální proměnná `databazeTabulky` je deklarována pomocí třídy `DatabazeTabulky`, která slouží jako databáze k tabulce pro uložení čísla otázek a odpovědí uživatele.

Nejdříve se musí nakopírovat `predchoziIndex` do proměnné `novyIndex` spolu s odpovědí. V tomto případě je zde znak „A“, který zastupuje odpověď na otázku „ano“[29]. Po vytvoření nového indexu (odkaz na hledaný záznam) se projíždí celá databáze od druhého záznamu[30] do posledního pomocí cyklu `for`. Při projíždění

[28] Po načtení databáze je inicializována na prázdný index = ukazatel na první záznam.

[29] V případě odpovědi „ne“ by zde byl znak „N“.

[30] Číslování pole sice začíná od nuly, ale zde není potřeba hledat v prvním záznamu. Na ten se lze dostat jen pomocí tlačítka `Zpět`.

databáze se hledá shoda s novým indexem prostým porovnáváním metodou `equals`. Pokud se najde shoda, do proměnné `index` se uloží ukazatel na patřičný záznam z databáze a příkazem `break` se ukončí hledání. Jako další krok následuje zjištění čísla otázky u hledaného záznamu a jeho převod z typu `String` na typ `int`. Převod je nutný, aby se mohla od čísla otázky odečíst jednička. Odečítání je kvůli tomu, že se do tabulky zaznamená vždy předchozí otázka a odpověď, než na kterou je aktuálně uživatel tázán. To souvisí i s koncem diagnostiky. Pokud nastane konec, tak se objeví číslo otázky, na kterou odpověděl uživatel jako poslední. Metodou `add()` se uloží číslo otázky a odpověď [31] na ní do databáze tabulky. Aby se tabulka aktualizovala musí se vyvolat metoda `updateUI()`. Vytvořenou funkcí `ukazData()` se vyvolají potřebné operace pro zobrazení vyhledaného záznamu. Zde se také kontroluje, jestli je diagnostika již u konce pomocí hodnoty `konec` ze záznamu. Jestli je u konce, zobrazí text konce diagnostiky a skryje tlačítka `Ano`, `Ne` a `Nevím`. Pokud může diagnostika pokračovat nechají se tlačítka aktivní. Také se testuje jestli je u vyhledaného záznamu zadaná nápověda k otázce pomocí metody `isEmpty()`. Jestliže je zadaná nápověda, zaktivuje se tlačítko `Nevím`. V opačném případě se tlačítko deaktivuje. Testování podléhá i hodnota `obrazek` ze záznamu, ve které se ukládají odkazy na konkrétní obrázky ze složky *MEDIA*. Pokud není hodnota prázdná, vykreslí se obrázek, na který hodnota ukazuje pomocí, vyvolané funkce `nahledNaObrazek()`. Poslední příkaz v ukázce je pro aktivaci tlačítka `Zpět`, které umožňuje skok na předchozí otázku.

Popis tlačítek `Nevím` a `Zpět`

- **Nevím** (obr. 2.17 a) – jestli uživatel neví, co se přesně myslí otázkou nebo nezná nějaký použitý pojem v otázce, může kliknout na toto tlačítko pro upřesnění pojmů z otázky. Ovšem to lze jen tehdy, jestli je nápověda v daném záznamu k dispozici. Jestliže není zadaná nápověda v záznamu, tlačítko bude neaktivní a nelze na něj kliknout.
- **Zpět** (obr. 2.17 b) – tímto tlačítkem se lze posunout na předchozí záznam, který již byl zobrazen. Tlačítko je vždy aktivní, až na výjimku prvního záznamu, kde se není kam vracet.

a) b)

Obr. 2.17: Druhé dvě tlačítka (komponenty JBUTTON): a) Nevím), b) Zpět

Ukázka algoritmu hledání předchozího záznamu:

[31] V případě odpovědi „ano“ se uloží hodnota `ANO` a naopak u odpovědi „ne“ se uloží hodnota `NE` do tabulky.

```
char indexCH []=  predchoziIndex.toCharArray();
String novyIndex = String.copyValueOf(indexCH, 0,
                      predchoziIndex.length()-1 );
  for(int i = 0; i < databaze.size(); i++){
    if(databaze.get(i).getIndex().equals(novyIndex)){
      index = i;
      break;
    }
  }
databazeTabulky.del(databazeTabulky.size()-1);
T_tabulka.updateUI();
ukazData();
B_ano.setEnabled(true);
B_ne.setEnabled(true);
```

U této ukázky jsou nejdůležitější první dva řádky. První příkaz vytvoří proměnnou `indexCH` typu pole `char`, do kterého se pomocí metody `toCharArray()` překopíruje hodnota z proměnné `predchoziIndex`. Tím, že se tato hodnota překopíruje do pole, rozdělí se znaky do jednotlivých buněk v poli. Pomocí tohoto naplněného pole a metody `copyValueOf()`, lze vytvořit nový index. Tato metoda kopíruje do proměnné `novyIndex` (typ `String`) jednotlivé znaky předchozího indexu od prvního do předposledního. Tím máme zaručeno, že se vytvoří nový index na předchozí záznam. Hledání záznamu v databázi probíhá stejně jako u tlačítek `Ano` a `Ne`. Po vyhledání záznamu s odpovídajícím indexem se vymaže poslední záznam z tabulky (metodou `del()`)[32], který obsahuje číslo otázky a odpověď na předchozí otázku. Následně se aktualizuje tabulka metodou `updateUI()` a ukáží se příslušná data vyvoláním funkce `ukazData()` (tato funkce je již popsaná u tlačítka `Ano`). Jelikož se uživatel vrátil na předchozí záznam, kde bude určitě zadaná otázka, aktivují se tlačítka na odpovědi metodou `setEnabled(true)`.

[32]Číslování databáze začíná od nuly, ale metoda `size()` vrací velikost databáze (nula znamená prázdnou databázi). Proto musí být tato hodnota snížena od jedničku.

3 ZÁVĚR

Prvním úkolem této bakalářské práce bylo se seznámit s problematikou expertních systémů a zpracovat literární rešerši o expertních systémech. V textu byly popsány výhody a nevýhody, stručná historie, základní složky, přídavné složky a doplňkové složky expertních systémů. Dále byly uvedeny 4 stěžejní typy expertních systémů, z nichž jsou nejpoužívanější diagnostické a prázdné expertní systémy.

Hlavním cílem této bakalářské práce bylo vytvoření a odladění expertního systému diagnostikující problém v určitém oboru. Po dohodě s vedoucím této bakalářské práce byl vybrán jako diagnostický problém z oblasti výpočetní techniky, konkrétně diagnostika problému se startem počítače. Toto téma bylo dobré zpracovat, protože hodně lidí se často setkává s nějakým problémem u startu počítače. Po důkladném promyšlení bylo zjištěno, že nejlepší řídící technikou pro tento systém bude použít hybridní databázi, která je založena na pravidlech a rozhodovacích stromech. Databáze je tedy založena na pravidlech, která specifikují co se v konkrétním případě bude přesně dělat. Funkce rozhodovacího stromu spočívá v umístění otázky v rámci databáze.

První část návrhu bylo vytvořit tzv. prázdný expertní systém. To je systém, který nemá pevně zabudovanou databázi znalostí ve svém programovém kódu. Nejprve musel být vytvořen program *Editace databáze*, který bude danou databázi vytvářet. Konkrétní návrh byl propracován, aby spolu programy spolupracovali bezchybně, a aby bylo možno vytvořit i jinou databázi než je problém se startem počítače. Byl navrhnut systém pro hledání konkrétních záznamů z databáze. Každý záznam má pevně stanoveny zadané hodnoty, které pak následně expertní systém používá. Databáze obsahuje tzv. „strom problémů“ a ten popisuje všechny problémy postupně od základní otázky, „Jde počítač spustit?“, po složitější otázky. Po ukončení diagnostiky je uvedeno stručné vysvětlení jak mohlo dojít k danému problému, jak mu příště předejít a pokud to jde, tak jak ho odstranit. Na uživatelské rozhraní obou programů pro komunikaci s uživatelem byl kladen důraz, aby byl srozumitelný pro všechny uživatele. Součástí programů je i nápověda, která popisuje jak se s daným rozhraním zachází.

LITERATURA

[1] ANDREW, D. M. *The definitions of progranimg languages* Cambridge University Press, New York, 1980, ISBN 0521226317.

[2] BERKA, P.; a kol. *Expertní systémy: skripta* Praha: VŠE, 1998. 160 s.

[3] CELBOVÁ, I. *Úvod do problematiky expertních systémů.* Ikaros [online]. 1999, Roč. 3,, č. 8 , [cit. 2011-03-07]. Dostupné z URL: *http://www.ikaros.cz/node/*393.

[4] DVOŘÁK, J. *Expertní systémy: skripta* Brno: FSI VUT v Brně, 2004. 92 s. [online]. Dostupné z URL: *http://www.uai.fme.vutbr.cz/~jdvorak/opory/expertnisystemy.pdf*.

[5] FEIGENBAUM, E. A. *Some challenges and grand challenges for computational intelligence.* Journal of the ACM 2003 50 (1): 32-40. doi:10.1145/602382.602400.

[6] JACKSON, P. *Introduction to expert systems : International computer science series.* Addison-Wesley Pub. Co., 1986. 1st edition. Michiganská univerzita : [s.n.], 1986. 246 s. Digitalizováno 14. říjen 2006. ISBN 0201142236, 97802.

[7] JIRSÍK, V.; HRÁČEK, P. *Neuronové sítě, expertní systémy a rozpoznávání řeči: elektronický text* AMT008. 2002. 107 s.

[8] JIRSÍK, V.; VALENTA, J. *Expertní systémy v praxi* AT&P journal PLUS [online] 2005 [cit. 2011-03-07]. Dostupné z URL: *http://www.atpjournal.sk/atpplus/archiv/*2005_7/*PDF/plus*05_08.*pdf*.

[9] KALUŽA, R. *Expertní systémy – úvod.* Kat. IT, Internet. [s.l.] : Http://owebu.bloger.cz/, 7. 5. 2006 [cit. 2011-03-7]. Dostupné z URL: http://radovan.bloger.cz/IT-internet/expertni-systemy—uvod.

[10] KOS, Z. *Expertní systémy : Rozhodovací procesy v životním prostředí* [online]. Praha : ČVUT v Praze - fakulta stavební, 15. 6. 2008 [cit. 2011-03-07]. Dostupné z URL: *http://geohydraulika.fsv.cvut.cz/on_line/rpz/*10−*Expertni_systemy.pdf*.

[11] MAŘÍK, V.; ZDRÁHAL, Z. a kol. *Expertní systémy: principy, realizace, využití.* Praha ČSVTS, 1984. 215 s.

[12] NEJEDLOVÁ, D. *Umělá inteligence: skripta* Liberecké informatické fórum 2008. 1. vyd. Liberec: Technická univerzita v Liberci, 2008. [online] [cit. 2011-03-07]. Dostupné z URL: *http://ilex.kin.tul.cz/ dana.nejedlova/multiedu/AIpredn.pdf*.

[13] NOVÁK, J. *Expertní systémy pro samostatné studium a jeho vyhodnocení - diplomová práce* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 104 s. Vedoucí diplomové práce doc. Ing. Václav Jirsík, CSc. [online]. *http : //www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id = 15034.*

[14] PINDUROVÁ, L.; ŠIMČÍK, A. *Vývoj znalostního systému na podporu rozhodování při konzervaci kovů vyvinutém na FPF Slezské univerzity v Opavě.* [online]. Slezská univerzita v Opavě : Filozoficko-přírodovědná fakulta. Verze 1.0. Opava : 2004 [cit. 2011-03-07]. Dostupné z URL: http://axpsu.fpf.slu.cz/ sim20uh/technologie/emedia/clanek1/me.htm.

[15] POSPÍŠIL, R. *Využití expertních systémů v marketingovém průzkumu.* Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2009. 106 s. Vedoucí diplomové práce doc. Ing. Vladimír Chalupský, CSc., MBA.

[16] PROVAZNÍK, I.; BARDOŇOVÁ, J. *Expertní systémy – praktická cvičení: skripta* Brno: Vysoké učení technické v Brně, 2000. 92 s. ISBN 80-214-1768.

[17] PROVAZNÍK, I.; KOZUMPLÍK, J. *Expertní systémy: skripta* Brno: Vysoké učení technické v Brně, 1999. 123s.

[18] PROVAZNÍK, I.; KOZUMPLÍK, J. *Úvod do medicínské informatiky: skripta* Brno: Vysoké učení technické v Brně, FEKT, 2002. 36s.[online] Dostupné z URL: *http : //files.gamepub.sk/ET1/uvod_do_medicinske_informatiky.pdf.*

[19] RANSDOLF, M. *Expert.* Edice č. 12. Brno : 2011 [online], 12.02.2011 [cit. 2011-03-07]. Ransdorf. Dostupné z URL: *http : //www.ransdorf.cz/Expert.html.*

[20] SVITEK, T. *Umělá inteligence.* [online] 2011, 2010-03-17 [cit. 2011-03-07]. Scribd - volné publikování dokumentů. Dostupné z URL: http://www.scribd.com/doc/50937571/31/Vysvětlovací-mechanismus.

[21] THON, M. *Expertní systémy : Historie a přehled expertních systémů* [online]. 2006 [cit. 2011-03-07]. Dostupné z URL: *http : //milost.wz.cz/umi/referat/index.html#historie.*

[22] THON, M. *Expertní systémy : Klasifikace expertních systémů* [online]. 2006 [cit. 2011-03-07]. Dostupné z URL: *http : //milost.wz.cz/umi/referat/klasifikace.html.*

[23] *Umělá inteligence. In Wikipedia : the free encyclopedia* [online].St. Petersburg (Florida) : Wikipedia Foundation, 10. 5. 2004, poslední modifikace v 18. 4. 2011 [cit. 2011-03-07]. Dostupné z URL:
$http://cs.wikipedia.org/wiki/Um\%C4\%9Bl\%C3\%A1_inteligence.$

[24] *Expertní systém. In Wikipedia : the free encyclopedia* [online].St. Petersburg (Florida) : Wikipedia Foundation, 17. 3. 2005, poslední modifikace v 15. 12. 2010 [cit. 2011-03-07]. Dostupné z URL:
$http://cs.wikipedia.org/wiki/Expertn\%C3\%AD_syst\%C3\%A9m.$

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

BD Báze dat – Data base

BIOS Základní vstupně–výstupní systém – Basic Input-Output System

BZ Báze znalostí – Knowledge base

ES Expertní systém – Expert system

Gui Grafické uživatelské rozhraní – Graphical user interface

HDD Pevný disk – Hard disk drive

HW Jednotlivé elektronické součásti – Hardware

OS Operační systém – Operating system

PC Osobní počítač – Personal computer

RM Řídící mechanizmus – Control mechanism

UI Umělá inteligence – Artificial intelligence (AI)

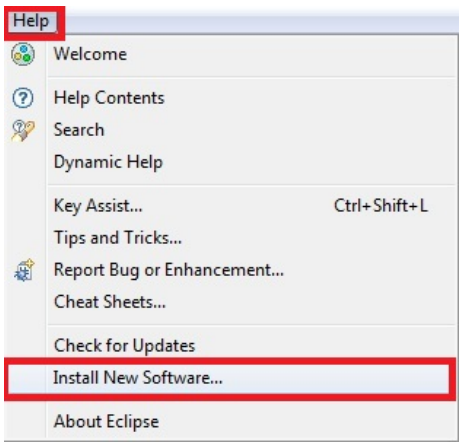
SEZNAM PŘÍLOH

A PŘÍLOHY

A.1 Návod na stažení a instalaci pluginu *Jigloo* do programu Eclipse

V prvé ředě je zapotřebí mít stažené a nainstalované vývojové prostředí Eclipse. Na adrese http://www.eclipse.org/downloads/ si lze tento program zdarma stáhnout. Na uvedené adrese je vidět více verzí programu Eclipse v poslední verzi Helios, ke dni 14. 4. 2011. Podle využití programu Eclipse ho lze stáhnout s již naimportovanými pluginy, jako jsou např: podpora C/C++, podpora PHP, prostředí Java EE a další. Výhoda programu Eclispe je v tom, že je rozšířený, je zadarmo a existuje na něj velké množství různých pluginů. Popis programu Eclipse není součástí této práce, více k popisu Eclipse je na adrese http://nb.vse.cz/ buchalc/eclipse.pdf.

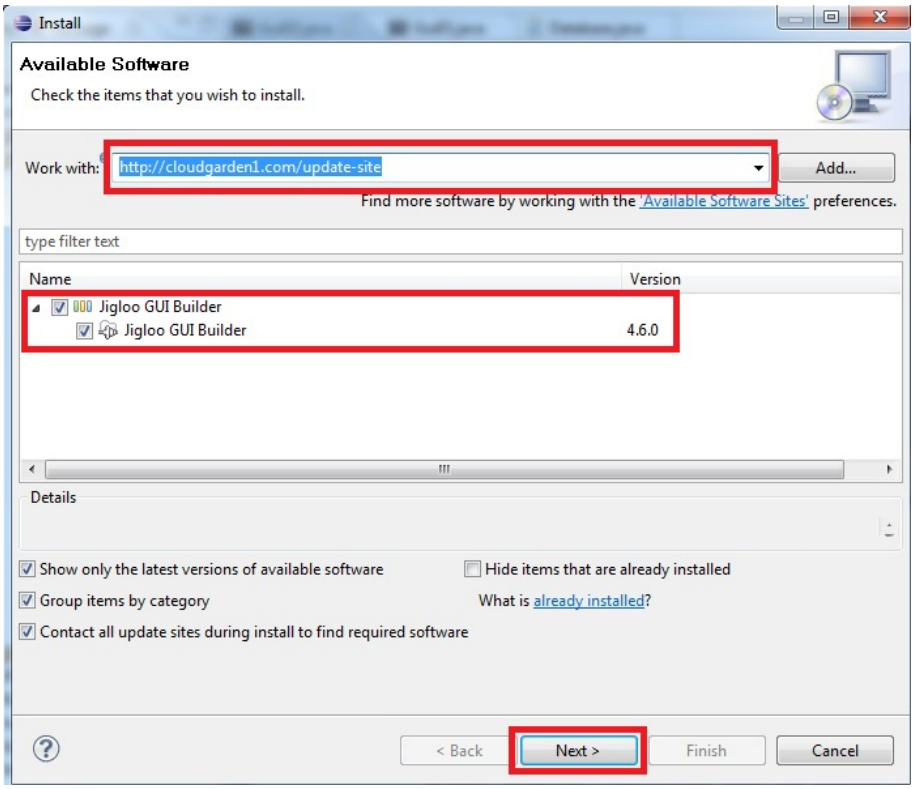
Stažení pluginu *Jigloo* provedeme pomocí nabídky v programu Eclipse. Kliknutím na menu `help` se rozevře nabídka. Vybráním položky `Install New Software...` podle obrázku A.1, se otevře nové okno, které je znázorněné na obr. A.2.



Obr. A.1: Menu Eclipse – instalování nového softwaru

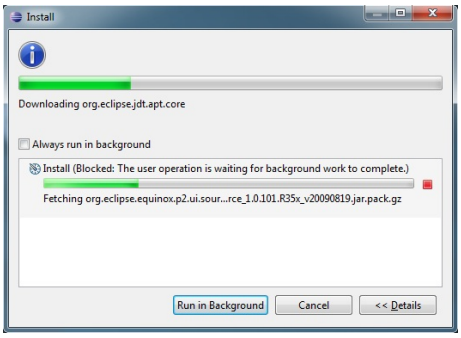
Další krok instalace spočívá v tom, že se zadá adresa pluginu *Jigloo* do editačního okna `Work with` (obr. A.2). Adresa je: `http://cloudgarden1.com/update-site`. Po zadání adresy, se uprostřed okna objeví nabídka s názvem `Jigloo GUI Builder`, která se musí pro instalaci zaškrtnout. Ostatní zaškrtávací políčka se nechají ve výchozích hodnotách. V následujícím kroku stačí přejít k instalaci tlačítkem `Next`. Objeví se okno `Install Details`, ve které se instalace potvrdí opět tlačítkem `Next`.

Ještě před samotným započetím instalace se musí potvrdit licence (obr A.4). Následným potvrzením licence a kliknutí na tlačítko `Finish` se začne plugin stahovat a posléze instalovat (obr. A.3). Instalace programu chvilku trvá. Po instalaci

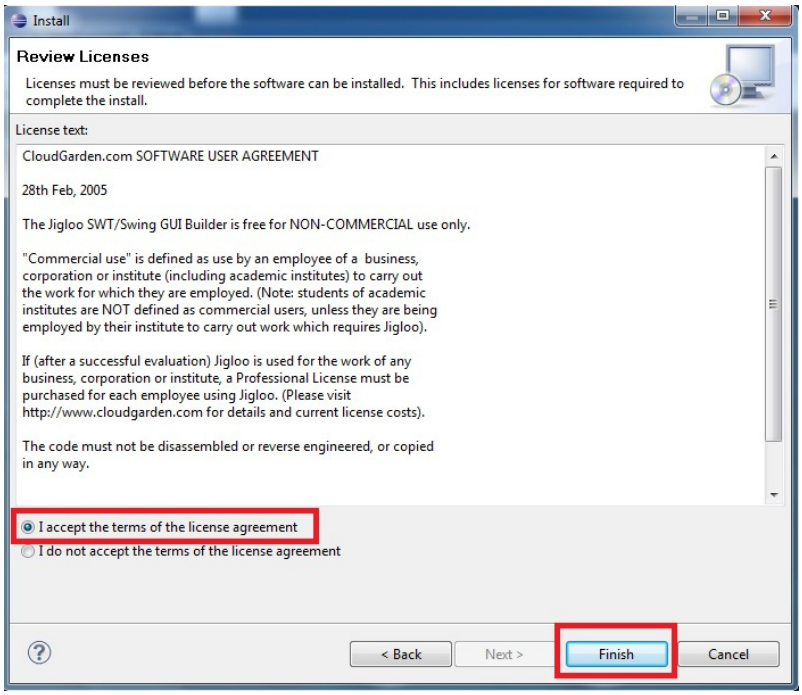


Obr. A.2: Instalační okno č. 1 – výběr pluginu

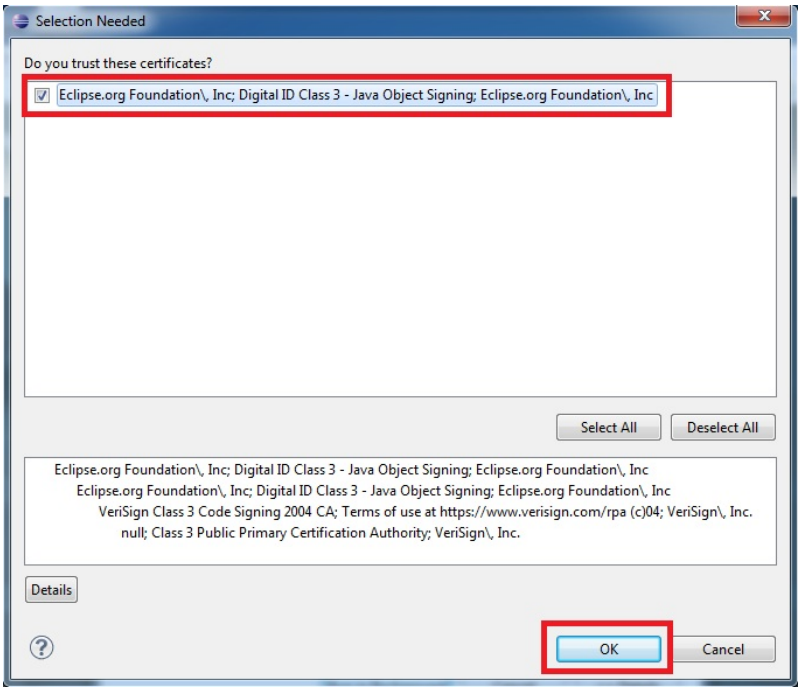
programu se objeví okno (obr. A.5), kde se musí potvrdit certifikát vydaný od Eclipse pro plugin *Jigloo*. Potvrzením vyskočí okno (obr. A.6), které upozorňuje, že se musí program Eclipse restartovat. Potvrzením tlačítka `Yes` se Eclipse restartuje a je připraven pro práci s pluginem *Jigloo*.



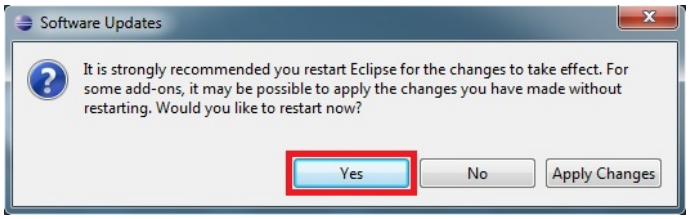
Obr. A.3: Instalační okno č. 3 – průběh instalace



Obr. A.4: Instalační okno č. 2 – potvrzení licence



Obr. A.5: Instalační okno č. 4 – potvrzení



Obr. A.6: Instalační okno č. 5 – restart

A.2 Použitý model tabulky (třída MyTableModel)

```
public class MyTableModel extends AbstractTableModel {
  private String [] columnHeader={"č. ot.", "Index",
  "Text otázky","Nápověda", "Výsledek", "Konec", "Obrázek"};
           //hlavička sloupců
private Databaze databazeTabulky;    //databáze
public MyTableModel() {              //konstruktor prázdný
}
public MyTableModel(Databaze databazeTabulky) {
  this.databazeTabulky=databazeTabulky;//přetěžování konstruktoru
}                                  //vytvoří databázi při volání
public int getColumnCount() {
  return columnHeader.length;    //vrací kolik je zadaných sloupců
}
public int getRowCount() {
  return databazeTabulky.size();   //vrací počet záznamů (řádků)
}
public String getColumnName(int columnIndex ) {
  return columnHeader[columnIndex];//vrací jméno hlavičky sloupce
}
public Class<?> getColumnClass(int columnIndex) {
  switch (columnIndex) {           //tato metoda říká tabulce
  case 5:return Boolean.class;     //typy proměnných ve sloupcích
  default:return String.class;
  }
}
public boolean isCellEditable(int rowIndex, int columnIndex) {
  switch (columnIndex) {        //toto tabulce říká jaké sloupce
  case 5:return true;           //jsou editovatelné
  default:return false;
```

```
    }
  }
  public Object getValueAt(int rowIndex, int columnIndex) {
    Item myItem = databazeTabulky.get(rowIndex);
    switch (columnIndex) {
      case 0:return myItem.getC_otazky();
      case 1:return myItem.getIndex();
      case 2:return myItem.getText_otazky();
      case 3:return myItem.getNapoveda();
      case 4:return myItem.getVysledek();
      case 5:return myItem.isKonec();
      case 6:return myItem.getObrazek();
      default:return null;
    }
  }//metoda na vykreslení hodnot z databáze do tabulky

  public void setValueAt(Object aValue, int rowIndex,
                         int columnIndex) {
    Item myItem=(Item)databazeTabulky.get(rowIndex);
    switch (columnIndex) {
      case 0:myItem.setC_otazky((aValue.toString()).trim());break;
      //funkce trim() odstraňuje úvodní a koncové bílé znaky
      case 1:myItem.setIndex(((
      aValue.toString()).trim()).toUpperCase());break;
      //funkce toUpperCase() převede všechno na velká písmena.
      //u indexu jsou požadovaná jen velká písmena
      case 2:myItem.setText_otazky((aValue.toString()).trim());break;
      case 3:myItem.setNapoveda((aValue.toString()).trim());break;
      case 4:myItem.setVysledek((aValue.toString()).trim());break;
      case 5:myItem.setKonec((Boolean)aValue);break;
      case 6:myItem.setObrazek((aValue.toString()).trim());break;
}
}//metoda na ukládání hodnot do databáze z tabulky
```

A.3 Použité klávesové zkratky v programech

Zkratka	Akce
CTRL + N	Nová databáze
CTRL + O	Otevření databáze
CTRL + S	Uložení databáze
F1	Nápověda
CTRL + I	O aplikaci
LALT + S	Menu soubor
LALT + N	Menu nápověda
LALT + P	Přidat záznam
LALT + O	Odebrat záznam
LALT + C	Předchozí buňka
LALT + D	Další buňka
LALT + E	Přidej obrázek
LALT + M	Smaž obrázek

Tab. A.1: Klávesové zkratky v programu *Editace databáze*

Zkratka	Akce
CTRL + O	Načíst databázi
CTRL + E (LALT F4)	Ukončení programu
F1	Nápověda
CTRL + I	O aplikaci
LALT + S	Menu soubor
LALT + A	Odpověď „ano“
LALT + E	Odpověď „ne“
LALT + V	Nevím
LALT + Z	Zpět

Tab. A.2: Klávesové zkratky v programu *Expertní systém*

A.4 Obsah přiloženého DvD

1. Složka **Dokumentace** obsahuje zdrojové soubory pro program LaTex.
2. Složka **Eclipse** obsahuje spustitelný program *Eclipse*, ve kterém je už připravený plugin *Jigloo*.
3. Složka **ExpertiSystem** obsahuje zdrojové soubory pro program Java *Eclipse*.
4. Složka **Java** obsahuje instalaci programu *Java*, který je nutný pro spuštění programů.
5. Složka **MiKTeX 2.9 Setup** obsahuje instalaci programu LaTex.
6. Složka **Program Spustitelný** obsahuje:
   - Složku **help**, ve které jsou uloženy HTML stránky na nápovědu k programům (z této složky čerpají programy nápovědu).
   - Složku **MEDIA**, která obsahuje potřebné obrázky pro databázi (programy ji potřebují pro správnou funkci databáze, pokud používají obrázky).
   - Soubor **diagnostika_StartPC.dat**, to je hlavní databáze pro program *Expertní systém*, která byla vytvořena v programu *Editace databáze*.
   - Soubor **Editace databáze**.jar. Je to spustitelný soubor programu *Editace databáze*.
   - Soubor Expertní systém.jar. Je to spustitelný soubor programu *Expertní systém*.
     - Pro tyto dva spustitelné soubory je nutné mít nainstalovaný program *Java* ze složky **JAVA**.
7. 7. Soubor **Diagnostický expertní systém.pdf** je dokumentace k bakalářské práci.