Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

# DIPLOMOVÁ PRÁCE

Informační systém s neznámými hodnotami

2022

Bc. Petr Bobek

Vedoucí práce:
Mgr. Jan Laštovička, Ph.D.

Studijní program: Aplikovaná informatika,
Specializace: Vývoj software

**Bibliografické údaje**

| | |
|---|---|
| Autor: | Bc. Petr Bobek |
| Název práce: | Informační systém s neznámými hodnotami |
| Typ práce: | diplomová práce |
| Pracoviště: | Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci |
| Rok obhajoby: | 2022 |
| Studijní program: | Aplikovaná informatika, Specializace: Vývoj software |
| Vedoucí práce: | Mgr. Jan Laštovička, Ph.D. |
| Počet stran: | 46 |
| Přílohy: | 1 CD/DVD |
| Jazyk práce: | český |

**Bibliographic info**

| | |
|---|---|
| Author: | Bc. Petr Bobek |
| Title: | Information system with unknown values |
| Thesis type: | master thesis |
| Department: | Department of Computer Science, Faculty of Science, Palacký University Olomouc |
| Year of defense: | 2022 |
| Study program: | Applied Computer Science, Specialization: Software Development |
| Supervisor: | Mgr. Jan Laštovička, Ph.D. |
| Page count: | 46 |
| Supplements: | 1 CD/DVD |
| Thesis language: | Czech |

## Anotace

*V této diplomové práci představuji návrh a implementaci informačního systému s neznámými daty. Systém byl vytvořen za pomocí vývojového prostředí Visual Studio v jazyce C#. Vytvořený informační systém umožňuje spravovat a uchovávat data, včetně dat neznámých a nad těmito daty vyhodnocovat dotazy uživatele formou filtrace.*

## Synopsis

*The information system with unknown data represent in this master thesis was created with development environment Visual Studio and C# language. This information system enables it's users store and manage his data, include unknown data. Information system enables evaluate user's queries over these data.*

**Klíčová slova:** Neznámá hodnota, ontologie, informační systém, modální logika

**Keywords:** Unknown value, ontology, information system, modal logic

Tímto bych chtěl poděkovat vedoucímu mé diplomové práce Mgr. Janu Laštovičkovi, Ph.D. za cenné připomínky a vedení při tvorbě této práce. Dále bych chtěl poděkovat rodině a přátelům za podporu v průběhu studia.

*Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.*

datum odevzdání práce podpis autora

# Obsah


**1 Úvod** **8**

**2 Ontologie** **9**
2.1 Původ a vývoj . . . 9
2.2 Základní členění . . . 9
2.2.1 Entita . . . 11
2.2.2 Třída . . . 11
2.2.3 Atributy . . . 12
2.2.4 Vztahy . . . 12
2.3 Ontologie pro vývoj informačních systémů . . . 13
2.3.1 Fáze vývoje . . . 13
2.3.2 Vazby v systému . . . 14
2.3.3 Informační systém . . . 14

**3 Modální logika** **16**
3.1 Kripkeho modely . . . 16
3.2 Kripkeho sémantika . . . 18

**4 Existující řešení** **19**
4.1 Formální konceptuální analýza . . . 19
4.1.1 Neúplná data . . . 19
4.2 Neúplné uživatelské modely v doporučovacích systémech . . . 20
4.3 Vícehodnotový neúplný informační systém . . . 21

**5 Model aplikace** **22**
5.1 Základní popis . . . 22
5.1.1 Entita . . . 23
5.1.2 Kategorie . . . 23
5.1.3 Vlastnost . . . 24
5.1.4 Vazba . . . 25

**6 Implementace** **26**
6.1 Použité technologie . . . 26
6.1.1 C# . . . 26
6.1.2 XML . . . 26
6.1.3 XAML . . . 26
6.1.4 WPF . . . 27
6.1.5 CSV . . . 27
6.2 Princip vazeb v systému . . . 28
6.2.1 Struktura vazeb . . . 28
6.2.2 Práce s vazbami . . . 29
6.3 Práce se soubory . . . 30
6.4 Rozdělení souborů . . . 30

6.4.1 Adresáře kategorií . . . . . 30
6.4.2 Soubory entit . . . . . 31
6.5 Soubory s vazbami . . . . . 32
6.6 Neznámé proměnné . . . . . 33
6.6.1 Tvorba neznámé entity . . . . . 33
6.6.2 Přidání vazby k neznámé entitě . . . . . 34
6.7 Filtrační funkce systému . . . . . 34
6.7.1 Filtrace . . . . . 34
6.7.2 Dedukce . . . . . 37

**7 Uživatelská příručka** **38**
7.1 Instalace a spuštění . . . . . 38
7.2 Hlavní okno a menu . . . . . 38
7.3 Vazby . . . . . 38
7.4 Okno se seznamem entit a správa entit . . . . . 38
7.5 Neznámá entita a dedukce . . . . . 39
7.6 Filtrace . . . . . 40
7.7 Náhled z praxe . . . . . 41

**Závěr** **42**

**Conclusions** **43**

**A Obsah přiloženého CD/DVD** **44**

**Literatura** **45**

# Seznam obrázků


1 Externí, konceptuální a interní schéma . . . . . 11
2 Hierarchie . . . . . 12
3 Kripkeho model s propozicemi p a q . . . . . 17
4 Nekompletní kontext $K_i$ . . . . . 20
5 Schéma entity . . . . . 23
6 Schéma vlastnosti . . . . . 24
7 Schéma vazeb v modelovém systému . . . . . 25
8 Struktura slovníku vazeb . . . . . 28
9 Okno se seznamem entit . . . . . 39
10 Neznámá entita a její dedukce . . . . . 40
11 Filtrace . . . . . 40


# Seznam tabulek


1 Tabulka vlastností . . . . . 24

# 1 Úvod

S rostoucím rozvojem technologií a modernizací se stále více lidí, skupin, firem a podniků dostává do situace, kdy potřebují uchovávat a analyzovat data v elektronické podobě. Získávání a zpracování informací z těchto dat je dnes pro uživatele velmi obtížným úkolem z důvodu časté nedostupnosti úplných informací ze zdroje, nebo poškození či ztrátě dat během jejich přenosu. Díky tomu se uživatelé dostávají stále častěji do kontaktu s neúplnými či neznámými daty. I malé procento chybějících údajů může způsobit vážné problémy při analýze, což vede k vyvození nesprávných závěrů a nedokonalým znalostem. Existuje mnoho technik jak překonat nedokonalé znalosti a spravovat data s neúplnými položkami, ale žádná není absolutně lepší než ostatní.

Ve většině informačních systémů obvykle chybí některé hodnoty z důvodu nedostupnosti dat nebo deformaci po jejich zpracování. Tyto chybějící hodnoty minimalizují kvalitu a množství klasifikačních pravidel generovaných systémem dolování dat. Tyto hodnoty mohou ovlivnit procento pokrytí a počet vygenerovaných pravidel a vést k obtížím při získávání užitečných informací ze souborů dat. Jedná se o běžný problém existující od počátku zpracování dat v různých aplikacích.

Vhodným nástrojem při řešení a návrhu systémů s neznámými či neúplnými hodnotami se ukázala být ontologie. Hlavní podstatou ontologie je podrobný popis jednotlivých funkcí a objektů, definice pojmů a vztahů a především určení vztahů mezi jednotlivými objekty. To vše za účelem zvýšení efektivnosti systémů a maximalizace výkonu a funkčnosti.

Možností, jak zvýšit přesnost a efektivnost ontologie, spočívá buď v přijetí modální logiky, která umožňuje vyjádřit omezení napříč světy, nebo jen v zvěcnění světů jako běžných objektů domény. Modální logika ve spojení s ontologií obohacuje systém o další operátory, které objektům dodávají „možnost“ či „nutnost“ výskytu.

Cílem práce je vyvinout informační systém, který bude schopen zpracovávat neznámé hodnoty. Tyto neznámé hodnoty nebudou v systému nijak odstraňovány ani transformovány na hodnoty jiné, ale bude se s nimi zacházet stejně jako s běžnými prvky systému. Hlavní schopností systému bude vyhodnocovat dotazy uživatele nad uloženými daty. Toto dotazování bude především formou filtrace. Součástí výsledku systému budou i neznámé hodnoty vytvořené uživatelem.

# 2 Ontologie

Ontologie je v informatice explicitní a formalizovaný popis určité problematiky. Je to formální a deklarativní prezentace, která obsahuje glosář (definici pojmů) a tezaurus (definice vztahů mezi jednotlivými pojmy). Jedná se o slovník, který slouží k uchovávání a předávání znalostí týkající se určité problematiky.

## 2.1 Původ a vývoj

Tento pojem pochází z oblasti filozofie, která se zabývá studiem bytí nebo existence. Zatímco v minulosti se ontologie omezovala spíše na filosofickou sféru, nyní získává specifickou roli ve sféře informačních technologií. [1] První použití termínu ontologie v literatuře z oblasti počítačové a informační vědy se objevuje již v roce 1967 v práci S. H. Mealyho o základech datového modelování. [2]

Ontologie má v informatické oblasti vícero názvů, jako je například „konceptuální modelování“ v návrhu databází, „modely tříd“ v objektově orientovaném návrhu softwaru, nebo „modely aplikačních domén“ v softwarovém inženýrství. Vzhledem k rozšiřujícímu se počtu oblastí využívající ontologii dochází k jejímu neustálému rozkvětu, jaký nepředpokládali ani původní tvůrci. Zprvu se mělo jednat především o popisnou funkci a definici systémů, nicméně postupem času se z ontologických metod stal pomocník nejen při vývoji softwarů či práci s databázemi, ale i při spoustě dalších informatických oborů. Napomáhá dosáhnout automatizace, a to především díky vyššímu stupni přesnosti popisu svých postupů. [3]

Dnes se ontologie využívá v umělé inteligenci, počítačové lingvistice, softwarovém a systémovém inženýrství a teorii databází jako datový model reprezentující určitou znalost nebo její část. Jedná se o způsob, jak ukázat vlastnosti předmětné oblasti a jejich vzájemné vztahy pomocí definic pojmů a kategorií, které předmět reprezentují.

## 2.2 Základní členění

Systémové ontologie se řídí několika základními požadavky:

- Koherentnost: jednotlivé moduly aspektů tvoří logický a skutečně integrální celek
- Komplexnost: pokrytí všech relevantních otázek a úplnost celku
- Konzistentnost: modely aspektů jsou bez rozporů či nesrovnalostí
- Stručnost: v modelech nejsou obsaženy žádné nadbytečné záležitosti a celek je kompaktní a výstižný

Datový analytik by si měl uvědomovat potřebu deklarativních reprezentací, které by měly mít co největší obecnost, aby se maximalizovala možnost opakovaného použití. Zároveň však tyto reprezentace musí co nejpřesněji odpovídat

věcem a procesům, které mají reprezentovat. Začíná si tedy klást otázky jako například: co je to objekt/proces/atribut/vztah? To znamená, že analytik musí vědět nejen o konkrétních symbolických objektech (zákazníci, platby, dluhy), které eviduje ve své databázi, ale také o objektech, vlastnostech a vztazích obecně a také o obecných typech procesů, do kterých mohou být objekty, vlastnosti a vztahy zapojeny. [2]

Souhrnně lze říci, že ontologický model ukazuje pouze podstatu systému a jeho hlubokou strukturu izolovanou od všech realizací a implementačních otázek.

Datový model ontologie obsahuje čtyři základní typy prvků, které budou dále podrobněji popsány:

- Entita
- Třída
- Atribut
- Vazba

Součástí datového modelu mohou být další doplňující prvky:

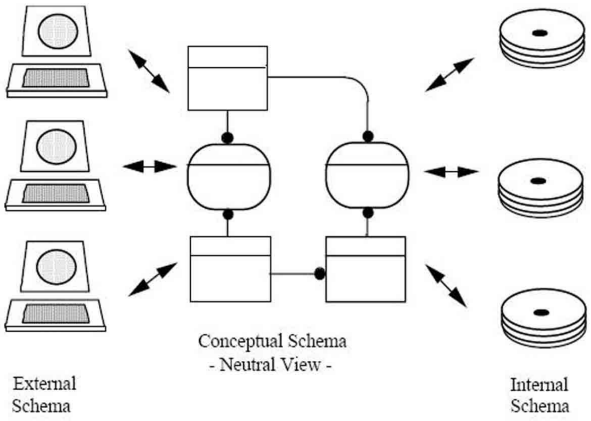
- Omezení
- Pravidla
- Axiomy
- Události

Modelování dat a informačních systémů se od 70. let rozvíjí podle tzv. přístupu tří diagramů. [4] Modelování dat je rozděleno do tří diagramů (Obr. 1):

- Externí diagram (prezentační schéma) definuje pohled na data z pohledu koncového uživatele nebo určité množiny programů.
- Konceptuální diagram zobrazuje pohled na data z perspektivy návrháře systému, nezávisle na uživatelích a způsobu uložení dat.
- Interní diagram (implementační schéma) zobrazuje organizaci dat v rámci aplikace a je závislý na implementaci, jedná se o fyzický způsob uložení dat a procedur.

Hlavní cíl rozdělení návrhu na několik úrovní je nezávislost návrhu na implementaci.

Pro tvorbu konceptuálních diagramů se využívalo několik modelů. Nejprve logické modely (relační a síťové), jež byli považovány za neintuitivní. Dále potom sémantický model [5] a ER diagram [6] používaný dodnes. Tyto dva modely předpokládají určitý pohled na svět (případně informační systém, aplikaci aj.) a hlavním předpokladem je, že strukturní aspekty lze vyjádřit pomocí pojmů entita a vztah.




Obrázek 1: Externí, konceptuální a interní schéma

### 2.2.1 Entita

Entity, jinak nazývané objekty nebo instance, jsou základními složkami ontologie. Entity mohou zahrnovat buďto konkrétní objekty jako jsou lidé, zvířata, automobily, nebo i abstraktní entity jako jsou čísla, slova, události. Obecně vzato ontologie nemusí zahrnovat žádné entity, ale jedním z hlavních účelů ontologie je poskytnout prostředek pro klasifikaci entit.

### 2.2.2 Třída

Třídy, neboli typy, kategorie či druhy, jsou abstraktní skupiny nebo kolekce objektů. Mohou obsahovat entity, jiné třídy nebo kombinace obou. Mezi třídy patří například:

- *Osoba*, třída všech lidí
- *Molekula*, třída všech molekul
- *Auto*, třída všech aut
- *Číslo*, třída všech čísel

Třídy ontologie mohou být dvojího typu: extensionální a intensionální. Třída je extensionální v případně, že je charakterizována pouze svým členstvím. Přesněji řečeno je třída C extensionální jen tehdy, když jakákoli třída C' má přesně stejné členy jako C. V tom případě jsou C' a C totožné. Pokud tato podmínka není splněna, pak je třída intensionální. Extensionální třídy jsou matematicky lépe pochopeny a jsou méně problematické, ale zato neumožňují jemné rozdíly mezi třídami, jež jsou v ontologii často potřeba.

Další vlastností tříd může být subsumpce. To znamená, že dochází ke vzniku podtříd a nadtříd. Subsumpce je využívána ke tvorbě hierarchie tříd, ve které je zahrnuta maximální obecná třída. [7] Klíčovým důsledkem subsumpčního vztahu

je dědičnost vlastností z rodičovské třídy na podřízenou. Platí tedy, že to co je pravdivé pro nadřazenou třídu, je nezbytně pravdivé i pro všechny její třídy podřízené. Rozlišujeme jednoduchou dědičnost (jeden rodič) a vícenásobnou dědičnost (více rodičů).

### 2.2.3 Atributy

Atributy slouží k popisu objektů. Každý atribut má minimálně název a hodnotu a slouží k ukládání informací charakteristických pro objekt, ke kterému je připojen. Například objekt Ford má atributy:

- *Jméno*: Ford Explorer
- *Počet dveří*: 4
- *Motor*: {4.0L, 4.6L}

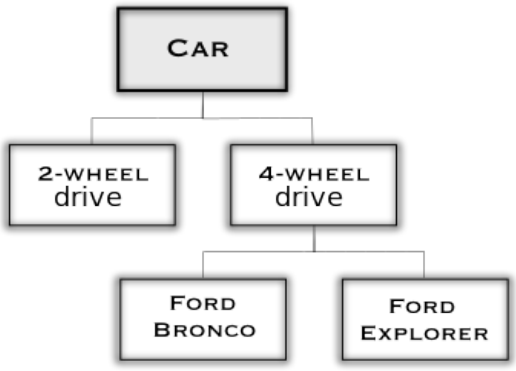
Hodnota atributu může být komplexní datový typ. V tomto příkladu je hodnota atributu s názvem *Motor* seznam hodnot, nikoli pouze jedna hodnota.

### 2.2.4 Vztahy

Jedná se o jednosměrné nebo obousměrné propojení dvou entit. Dá se říci, že vazba je speciálním typem atributu, jehož hodnota je jiná entita. Například, pokud ontologický systém obsahuje entity Ford Bronco a Ford Explorer, pak by Ford Bronco měl atribut nástupce:

- *Nástupce*: Ford Explorer

To nám říká, že Explorer je model, který nahradil Bronco. Velká část síly ontologií pochází ze schopnosti popsat tyto vztahy. Sada vztahů společně popisuje sémantiku domény.




Obrázek 2: Hierarchie

Existuje několik typů vztahů a mezi základní patří vztah subsumpce, který umožňuje definovat a klasifikovat objekty podle tříd. Použitím tohoto vztahu

dochází ke vzniku podtříd/nadtříd a stromové struktury jasně znázorňují souvislost mezi jednotlivými objekty. Příkladem může být, že třída Ford Explorer je podtřídou Ford s pohonem 4 kol a ta je zase podtřídou automobilu. [7]

Dalším typem často užívaného vztahu je mereologie, která představuje, jak se objekty spojují a vytvářejí složené objekty. Například, pokud zavedeme pojem volant tak můžeme říct, že volant je součástí automobilu Ford Explorer, protože volant je vždy jednou z komponent automobilu. Zatímco subsumpci lze graficky vyjádřit pomocí jednoduché stromové struktury, u mereologie toto možné není, protože objekty se mohou objevit pod více než jedním rodičem nebo větví. Z toho důvodu se pro znázornění používá řízený acyklický graf. [8]

Kromě výše zmíněných vztahů lze použít i další typy relací, které dál zpřesní sémantiku. Takové vztahy jsou často specifické pro danou doménu a využívají se k zodpovězení konkrétních otázek. Například pokud u automobilu definujeme v jakém státě byl vyroben a kde se daný stát nachází, můžeme získat odpověď na otázku: jaké automobily se v daném státě vyrábí?

## 2.3 Ontologie pro vývoj informačních systémů

Vývoj systémů je komplexní činnost, která zahrnuje např. pochopení kontextu použití a potřeby změn, získání a stanovení funkčních a nefunkčních požadavků, výběr nejlepšího návrhu a technické realizace, zajištění pokrytí aspektů lidské interakce a IT bezpečnosti a integraci systémů v kontextu použití. [9]

Ontologie je definována jako explicitní specifikace konceptualizace a slouží k definování pojmů a jejich vztahů. Z toho vyplývá, že koncepty, které jsou předmětem zájmu v dané oblasti (např. vývoj systémů), mohou být uspořádány jako ontologie. Ontologii lze použít ke stanovení a sdělení významu konceptualizovaných pojmů, aby bylo dosaženo srozumitelnosti mezi zúčastněnými subjekty. Dále by měla být rozšiřitelná, což znamená, že zúčastněné strany by měly mít možnost definovat a zahrnout nové pojmy a koncepty. Rozšiřitelnost je zvláště důležitá při vývoji systémů, kde má terminologie tendenci rychle růst a někdy se stává zahlcující. [10]

### 2.3.1 Fáze vývoje

Koncepčně lze vývoj systémů chápat jako sekvenční proces, v němž každá činnost vytváří informace, které jsou vstupem pro další činnost.

V ontologii jsou hlavní pojmy týkající se systémů jsou rozděleny do čtyř fází (oblastí):

- Obecný koncept
- Koncept popisu
- Koncept realizace
- Koncept vzhledu

Fáze obecného konceptu zahrnuje tři dílčí větve - systémová struktura, systémová produkce a systémová role. První větev vykazuje pojmy, které popisují strukturu systému a systém samotný na vysokých abstrakčních a logických úrovních. Příkladem je model nebo architektura systému. Systémová produkce představuje termíny související s procesem vzniku systému, jako je vývoj systému, jeho proces a jednotlivé fáze vývoje. Role systému se týká osob zapojených do procesu vývoje a zahrnuje pojmy zúčastněné strany, aktér, uživatel či zákazník.

Koncept popisu popisuje systém z různých hledisek v různých fázích vývoje včetně prohlášení, potřeby, požadavků, návrhu a řešení.

Koncept realizace zahrnuje činnosti prováděné v průběhu procesu vývoje systému. Tyto činnosti zahrnují analýzu kontextu, analýzu zúčastněných stran, analýzu potřeb, inženýrství požadavků, návrh, implementaci, nasazení, ověření a validaci.

Koncept vzhledu se týká vnějšího vzhledu systémů, tedy toho, jak systémy vypadají a chovají se z pohledu zúčastněných stran, například uživatelů. Mezi tyto koncepty patří funkce, vlastnosti, schopnosti, kapacita, účinek a služba. [10]

Základními pojmy ontologie pro vývoj informačního systému jsou systém a vývoj systémů (patřící do fáze obecného konceptu). Systém existuje v systémovém kontextu a je tvořen systémovými komponentami. Návrh a architektura jsou modely systému, přičemž návrh může mít různou úroveň podrobnosti, zatímco architektura představuje celkovou strukturu systému.

### 2.3.2 Vazby v systému

Systémy lze popsat podle toho, jak se jeví při pohledu zvenčí. Systémy poskytují funkce, které mají definované účinky. Právě tyto účinky poskytují užitek pro zúčastněné strany. Systémy mají také vlastnosti, které usnadňují interakci těchto účinků a umožňují různou míru možnosti jejich využití za různých okolností. Některé z vlastností úzce souvisejí s tím, jak a kdy mohou systémy poskytovat své funkce. Proto jsou to právě funkce a vlastnosti systému, které určují jeho možnosti. Z toho vyplývá, že systém má schopnost poskytovat funkce v určitém rozsahu a za určitých okolností. Schopnosti systémů mohou být viditelné a přístupné prostřednictvím služeb. Služba tedy poskytuje rozhraní pro přístup ke schopnostem systémů.

### 2.3.3 Informační systém

Informační systémy jsou širokou skupinou počítačových softwarových systémů, které poskytují informace na podporu rozhodování v organizaci. Jinými slovy, informační systém vytváří, zpracovává, uchovává a generuje informace, které pomáhají jednotlivcům činit smysluplná rozhodnutí.

Ontologie popisující informační systémy mohou být použity k vytvoření lepších modelovacích nástrojů. Ontologie informačních systémů jsou většinou zamě-

řeny na teorii. Tento typ ontologií Smith [2] nazval ontologií založenou na realitě (R-Ontology).

Ontologie popisující informace, jež lze použít v informačních systémech, vedou ke zlepšení fungování informačních systémů. Fonesca je definuje jako „pragmaticky orientované ontologie“, které vznikají spojením filozofického přístupu s pragmatickými cíli. Patří k epistemologickým ontologiím (E-Ontology), jak je navrhl např. Smith. [2]

Každý informační systém (IS) má svou vlastní ontologii, protože přisuzuje význam používaným symbolům podle určitého pohledu na svět. Explicitní ontologie může hrát v informačním systému specifickou roli. Je to role řídící všechny aspekty a všechny komponenty IS, lze tedy mluvit o IS řízených ontologií. IS se skládá z komponent tří typů: aplikačních programů, informačních zdrojů (databáze a báze znalostí) a uživatelské rozhraní. Tyto komponenty jsou integrovány tak, aby splňovali konkrétní účel. Každá z komponent může využívat ontologii svým vlastním způsobem. [7]

Možná ne tak zřejmé, ale přesto velmi důležité je použití ontologie ve spojení s komponentou uživatelského rozhraní. Ontologie obsahují sémantické informace o omezeních kladených na třídy a vztahy používané k modelování dané domény a úlohy.

Ontologie může mít na IS vliv ve dvou rozměrech:

- časový rozměr
- strukturální rozměr

Časový rozměr se týká toho, zda se ontologie používá ve fázi vývoje nebo ve fázi běhu. Strukturální rozměr se týká konkrétního způsobu, jakým může ontologie ovlivnit hlavní komponenty IS. V době vývoje může ontologie hrát důležitou roli ve fázi analýzy požadavků a konceptuálního modelování s cílem podpořit analýzu neformálních specifikací v přirozeném jazyce. [11]

Za běhu může ontologie v rámci uživatelského rozhraní hrát především roli, která umožňuje, aby se na ni uživatel dotazoval a procházel ji. V tomto případě uživatel o ontologii ví a používá ji (tj. dotazuje se na ni) v rámci běžného používání IS. Tímto způsobem může uživatel procházet ontologii, aby lépe porozuměl slovní zásobě používané IS, a mohl tak formulovat dotazy na požadované úrovni konkrétnosti. [7]

Ve fázi návrhu informačních systémů mohou být ontologie užitečné při vytváření konceptuálních modelů informačního systému, které přisuzují obsah a význam shromažďovaným informacím. Ontologie také slouží jako společný sdílený slovník pro více návrhářů informačních systémů. Znalosti z domény tak mohou být opakovaně využívány. [12]

# 3 Modální logika

Modální logika je rozšířením tradiční výrokové logiky. Ke známým výrokovým operátorům ($\neg$, $\vee$, $\wedge$, $\Longrightarrow$, $\Longleftrightarrow$) přidává operátory možnosti $\Diamond$ („je možné, že“) a nutnosti $\Box$ („je nutné, že“). Zde $\Box$p znamená, že výrok $p$ je nutně pravdivý, a $\Diamond$p znamená, že výrok $p$ je možná pravdivý.

V posledních padesáti letech se modální logika stala velmi užitečnou v různých oblastech informatiky jako je umělá inteligence, teorie databází, distribuované systémy, verifikace programů a teorie kryptografie. Například výroková dynamická logika se dobře hodí k ověřování správnosti složitých a integrovaných systémů.

Přizpůsobitelnost modální logiky mnoha oblastem informatiky vyplývá z flexibility významů modálních operátorů. Základní modální operátory „nutnosti“ a „možnosti“ mohou nabývat různých významů v závislosti na aplikaci a typu modální logiky, na kterou jsou aplikovány. U procesů v distribuovaném systému chceme mít jistotu, že pokud proces „ví“, že $\varphi$ je pravdivé, pak $\varphi$ je skutečně pravdivé. Tato pravidla povedou k různým variantám modální logiky. Každou variantu označujeme jako samotnou modální logiku (multimodální, epistemická, doxastická nebo temporální modální logika). [13]

Modální logika se skládá ze syntaktických a sémantických informací, které popisují modální formule. Nejjednodušší a nejzákladnější modální formule jsou propozice. Propozice popisují fakta o určitém světě nebo stavu, například „nebe je dnes zatažené“. Nechť $\Phi = \{p_1, p_2, p_3\}$ je neprázdná množina propozic. Modální formule je buď prvkem $\Phi$, nebo má tvar $\varphi \wedge \psi$, $\neg\varphi$ nebo $\Box\varphi$, kde $\varphi$ a $\psi$ jsou modální formule.

Velikost modální formule $\varphi$, zkráceně size($\varphi$), je počet symbolů v $\varphi$. Každý symbol je členem množiny $\Phi \cup \{\wedge, \neg, \Box, (,)\}$. Kódování formule $\varphi$ je $\varphi$ s každou propozicí nahrazenou jejím binárním kódováním. Délka $\varphi$, zapsána jako $\mid \varphi \mid$, je délka kódování $\varphi$. [13]

## 3.1 Kripkeho modely

Abychom mohli matematicky uvažovat o modalitě, potřebujeme sémantiku, která by popsala modální operátory. K tomu používáme takzvanou Kripkeho sémantiku, známou také jako sémantika možných světů, ve které můžeme uvažovat o tom, jaké „světy“ jsou možné. Než se však dostaneme k podrobnostem Kripkeho sémantiky, musíme se nejprve podívat na Kripkeho modely. Tyto Kripkeho modely slouží ke specifikaci toho, jak spolu světy souvisejí, a to nám umožní uvažovat o nutnosti a možnosti propozic.

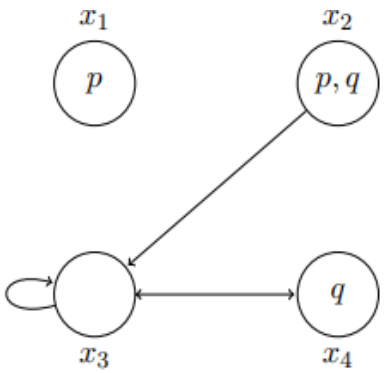
**Definice 1 (kripkeho model)**
*Kripkeho model M je trojice (W, R, L) skládající se z:*

- *množiny světů W*
- *relace přístupnosti $R \subseteq W \times W$ nad světy*
- *označovací funkce $L : W \to P$, která mapuje každý svět na množinu atomických propozic*

Relace přístupnosti R spojuje světy dohromady. Například $xRy$ znamená, že svět $y$ je přístupný ze světa $x$, což znamená, že je možné přejít ze světa $x$ do světa $y$. Nutno poznamenat, že $xRy$ je pouze alternativní zápis pro $(x, y) \in R$.

Funkce označování L slouží k určení, které atomické propozice (tj. pojmenované proměnné, tedy propozice bez operátorů) platí v každém světě. Například $p \in L(x)$ říká, že atomická propozice $p$ platí ve světě $x$. [14]

Každá modální struktura je znázorněna pomocí směrovaného grafu. Každý uzel je pojmenován podle příslušného stavu ve $W$. Hrana z uzlu $w$ do uzlu $w'$ znamená, že (w, w')∈ R. V každém uzlu jsou znaky symbolizující, které propozice jsou v daném stavu pravdivé.




Obrázek 3: Kripkeho model s propozicemi p a q

V tomto modelu máme čtyři světy W = $\{x_1, x_2, x_3, x_4\}$ označeny sadou atomických propozic, které v těchto světech platí. Například ve světě $x_1$ platí atomická propozice $p$, ale neplatí $q$, takže L($x_1$) = {p}. Může se zdát, že svět $x_3$ nemá označení, ale to znamená, že zde neplatí žádná z atomických propozic, takže L($x_3$) = $\emptyset$. Ačkoli máme pro každou kombinaci atomických propozic jeden svět, není to pravidlem. Můžete mít více světů označených stejnou sadou atomických propozic a můžete mít kombinace atomických propozic, které se nevyskytují v žádném světě.

Relace přístupnosti R je na Obr.3 znázorněna pomocí šipek. Jak je vidět na světě $x_3$, je možné, že svět je přístupný sám ze sebe. V tomto ukázkovém modelu šipky představují R = $\{(x_2, x_3), (x_3, x_3), (x_3, x_4), (x_4, x_3)\}$

**Definice 2 (splnitelnost)**

*Kripkeho model $M = (W, R, L)$ splňuje formuli $\varphi$, jestli každý svět $x \in W$ splňuje $\varphi$.*

**Definice 3 (platnost)**

*Formule $\varphi$ je platná, pokud ji splňuje každá možná formule Kripkeho modelu.*

## 3.2 Kripkeho sémantika

**Definice 4 (kripkeho sémantika)**

*Kripkeho sémantika definuje relaci splnitelnosti $x \models \varphi$ nad Kripkeho modelem $M = (W, R, L)$, který se používá pro vyhodnocení, zda propozice $\varphi$ platí či neplatí ve světě $x \in W$.*

*Pro libovolnou atomickou propozici $a$ platí $x \models a$ tehdy a jen tehdy, když $a \in L(x)$.* [15]

Sémantika modální logiky vychází ze sémantiky možných světů. Sémantika možných světů sleduje různé světy neboli stavy, které jsou případně pravdivé poté, co v jednom světě dojde ke změně. Například existuje mnoho možností, co by mohlo být zítra, ale pouze jedna z těchto možností je realizována.

# 4 Existující řešení

Existuje několik prací, které se zabývají zpracováním neúplných informací, aby bylo možné získat informace z neúplné datové sady. Některé z nich jsou založeny na hrubých množinách, fuzzy množinách, statistických metodách, bayesovských klasifikátorech, podobnostech atd. Doposud však nedošlo k žádnému výsledku implementace těchto metod kvůli nedostatkům v jejich metodikách nebo pro zpracování velkých souborů dat.

Růst objemu dat a počtu existujících databází značně převyšuje schopnost člověka tato data analyzovat, což vytváří potřebu i příležitost získávat znalosti z databází. Například lékařské databáze nahromadily velké množství informací o pacientech a jejich zdravotním stavu. Vztahy a vzorce v těchto datech by mohly poskytnout nové lékařské znalosti. Analýza lékařských dat se často zabývá zpracováním neúplných znalostí, správou nekonzistentních částí informací a manipulací s různými úrovněmi reprezentace dat.

## 4.1 Formální konceptuální analýza

Formální konceptuální analýza (FKA) popisuje vztah mezi množinou objektů a množinou atributů. Je to metoda pro reprezentaci znalostí, správu znalostí a analýzu dat. Slouží k vyhledávání a vizualizaci všech pojmů a závislostí ve vstupních tabulkových datech.

Ve FKA se prvky jednoho typu nazývají formální objekty, prvky druhého typu se nazývají formální atributy. Vstupní data pro metodu FKA jsou reprezentována ve formě tabulky, takže každý řádek představuje objekt z oblasti zájmu a každý sloupec představuje jeden z definovaných atributů. Vstupní prvky matice mohou nabývat pouze logických hodnot, tj. každý objekt buď má, nebo nemá určitý atribut.

Základním pojmem ve FKA je formální kontext, který představuje vstupní data. Metody FKA pracují s jednoduchými binárními daty, která se skládají z množiny objektů, množiny atributů a vztahem určujícím, zda daný objekt má daný atribut.

### 4.1.1 Neúplná data

Formální konceptuální analýza zpracovává úplná data. To znamená, že pokud chceme zpracovávat neúplná data, musíme k nim přistupovat zvláštním způsobem. Jedním z těchto způsobů je jejich předzpracování. Nejjednodušší možností je odstranění neúplných záznamů (objektů/atributů) a zachování pouze úplných dat, která umíme zpracovat. Odstraněním neúplných záznamů ale přijdeme o informace, které tato data nesla. [16]

V praxi rozlišujeme dva typy kontextů: kontexty úplné a neúplné, jinak nazývané také tří-hodnotové kontexty. Neúplný kontext vypadá jako běžný vícehodnotový kontext $K_i = (\mathrm{G}, \mathrm{M}, (W_m)_{m \in M}, \mathrm{J})$, kde $V_m = \{+, -, ?\}$ pro všechny $m \in M$, kde pro každý vícehodnotový atribut $m \in M$ existuje neprázdná množina $V_m$

možných hodnot atributu. Znaky {+,-,?} symbolizují 3 možné vstupy v tabulce formálních kontextů. Písmeno $J$ je ternární relace $J \subseteq G \times M \times \{+, -, ?\}$, kterou lze považovat za graf mapování množiny $G \times M$ všech dvojic objektů a atributů do množiny {+,-,?} možných hodnot.

Interpretace je následující:
$(g, m+) \subseteq J$: je známo, že objekt $g$ má atribut
$(g, m-) \subseteq J$: je známo, že objekt $g$ nemá atribut
$(g, m?) \subseteq J$: není známo, zda objekt $g$ má atribut

| $\mathbb{K}_u$ | $a$ | $b$ | $c$ |
|---|---|---|---|
| $g$ | + | + | + |
| $h$ | ? | − | + |
| $k$ | − | + | ? |

Obrázek 4: Nekompletní kontext $K_i$

Na neúplné kontexty lze aplikovat proces škálování. Tímto procesem dochází k transformaci tříhodnotového kontextu na jednohodnotový, na který lze pak aplikovat běžné FKA metody. V procesu škálování je každá položka *J(g,m)* nahrazena - v řádku objektu $g$ - objektem řádku *J(g,m)* ve škálovacím kontextu *Sm*. Škálování také umožňuje zrekonstruovat původní neúplný kontext. [17]

## 4.2 Neúplné uživatelské modely v doporučovacích systémech

V práci [18] byla představena myšlenka, že atributy lze modelovat pomocí stromu konceptů a že strom konceptů lze použít k odhadu chybějících hodnot. Tuto myšlenku lze rozšířit na hodnoty domény, protože mezi nimi obvykle také existuje nějaký vztah. Například pokud máte rádi akční filmy, pak budete mít pravděpodobně rádi i dobrodružné filmy, protože akční a dobrodružné spolu úzce souvisejí. Tyto informace jsou reprezentovány pomocí atributového stromu konceptů, kde jsou doménové hodnoty modelovány pomocí konceptů. Hodnota konceptu představuje uživatelovu užitnou hodnotu a vztahy mezi nimi jsou vztahy typu *is-a*.

Podobnost konceptů je převládající formou ontologického uvažování. Většina technik používá k odhadu podobnosti vzdálenost mezi koncepty ve stromu konceptů nebo v podobné grafické ontologii a platí, že čím menší je vzdálenost mezi dvěma koncepty, tím více si jsou podobnější.

V grafu jsou atributy reprezentovány úrovněmi stromu a různé koncepty nabývají v hloubkách stromů hodnot daných atributů.

Pokud tedy víme, že dva koncepty jsou spojeny s kořenovým konceptem, pak můžeme odvodit, že nemají nic společného, a funkce podobnosti vrátí nulu. V mnoha situacích představují různé atributy stejný koncept nebo mají podobný význam. Model v citované práci využívá tohoto vzoru tím, že seskupuje podobné atributy do množiny, která se nazývá složený atribut.

Výsledkem experimentu byl systém pro doporučování filmů.

## 4.3 Vícehodnotový neúplný informační systém

V práci [19] je pojednáváno o tom, jak reprezentovat a zpracovávat ontologii založenou na granulárním počítání z vícehodnotového neúplného informačního systému. Granulární výpočty jsou přirozeným způsobem řešení lidských problémů a může být velkým přínosem při reprezentaci a konstrukci ontologie.

Hlavními cíli tohoto článku je vytvořit granulární model pro reprezentaci přibližných pojmů a doménových ontologií a představit algoritmy pro automatické generování doménových ontologií na základě granulárních výpočtů z vícehodnotových neúplných informačních systémů.

V článku je definován koncept doménového granula a je navržen model a algoritmus pro sestavení doménové ontologie. Granulární mřížka doménového konceptu je granulární abstrakcí znalostí o informačních systémech. Realizuje specializaci a generování znalostí o informačních systémech. Pro demonstraci platnosti algoritmu je použit reálný příklad. Výsledky simulace ukazují, že algoritmus je efektivní a proveditelný.

# 5 Model aplikace

V teoretické části práce je popsán nově vytvořený informační systém (IS) umožňující pracovat s neznámými hodnotami.

Při návrhu a zpracování IS jsem vycházel z ontologie. S její pomocí byl IS rozdělen na jednotlivé prvky, které byly jednoznačně definovány a popsány. Byly také ustanoveny vztahy mezi těmito prvky. Mezi použité prvky patří entita, vazba, třída a atribut. Všechny jsou aktivně využívány v systému (resp. informačním systému), který si uživatel sám vytvoří, pojmenuje a definuje si veškeré zmíněné prvky. Takových systémů lze mít v rámci našeho IS hned několik, přičemž každý z nich pracuje zvlášť a samostatně, bez prolínání prvků či vložených dat. Zjednodušeně řečeno lze říci, že se jedná o informační systém, umožňující vytvářet a spravovat další informační systémy.

V rámci teoretické práce budou modely, algoritmy a návrhy prezentovány na modelovém příkladu informačního systému pro zdravotnickou laboratoř. Systém je schopen navrhovat výsledek dotazovaných diagnóz na základě vložených dat, především vazeb mezi entitami.

## 5.1 Základní popis

Informační systém byl cílen především na malé firmy, podniky a obecně na drobné zájmové skupiny. Mezi základní vlastnosti systému patří:

- **univerzálnost**: vzhledem k cílové skupině je systém vytvořen co nejuniverzálněji. V systému si může uživatel navolit libovolné prvky, vlastnosti a vazby a vše si vytvořit podle svého.
- **jednoduchost**: v rámci systému nebyly vyvíjeny žádné nadbytečné funkcionality z důvodu, aby byl snadno pochopitelný a využitelný pro široké spektrum skupin. Veškeré funkce i grafická stránka byla pojata tak, aby byla snadno a rychle uchopitelná a co nejvíce usnadňovala a urychlovala práci s daty.
- **ukládání dat**: součástí systému je souborová databáze, ve které jsou uchovávány veškeré zadané informace a data, s nimiž uživatel pracuje.
- **neznámá hodnota**: hlavní funkcí systému je práce s neznámými hodnotami, které si uživatel sám definuje. Neznámou hodnotou může být kterýkoli prvek v souborové databázi. Neznámá hodnota má definovaný svůj rozsah (číslo, textový řetězec) a v rámci tohoto rozsahu s ní pak systém pracuje.
- **dotazování**: systém je schopen uživateli poskytnout odezvu na jednoduché dotazy nad uloženými daty. Výsledkem může být jistá hodnota, nebo hodnota možná za určitých podmínek.

### 5.1.1 Entita

Entita je základním stavebním kamenem systému. Jedná o elementární proměnné, které si uživatel sám vytváří. Jak bylo zmíněno dříve (kap. 2.1), může se jednat o objekty konkrétní nebo abstraktní. Ukázkou konkrétní entity může být například *leucinóza*, jaktožto jedno z onemocnění v našem vzorovém IS.

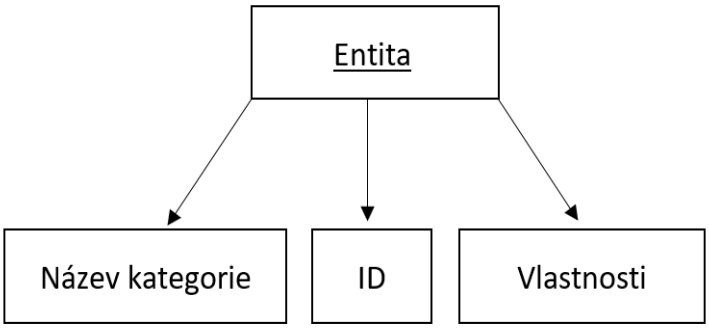
Každá entita obsahuje:

- *název kategorie*
- *vlastnosti*
- *ID*

Název kategorie slouží k rozdělení entit do skupin stejného charakteru.

Vlastnosti popisují a charakterizují jednotlivé entity. Každá entita může mít libovolný počet vlastností.

ID je unikátní identifikátor generovaný pro každou entitu. Ten zajišťuje, že každá entita je unikátní a nezaměnitelná s jinou v případě shody ve vlastnostech a názvu. Jedná se speciální typ vlastnosti.




Obrázek 5: Schéma entity

### 5.1.2 Kategorie

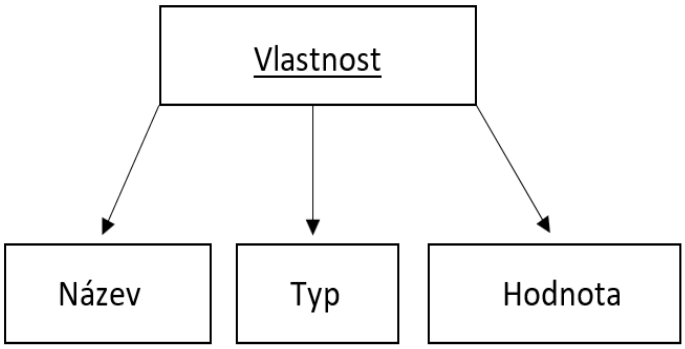
Z ontologické terminologie je kategorie rovna třídě. V našem systému jsme zvolili pojem *Kategorie* z důvodu korespondence s programovým kódem, ve kterém se předpisu pro objekt říká *Třída*, a mohlo by docházet k záměně pojmů. Hlavním účelem kategorie je sumarizace entit do jednoho seznamu v závislosti na názvu kategorie, jenž je uvedena u jednotlivých entit. Kategorie entit jsou uplatňovány zejména při filtraci a také vykreslení uživatelského rozhraní.

Příkladem kategorie je *diagnóza*. Seznamem této kategorie je soubor onemocnění spadající do dané kategorie.

### 5.1.3 Vlastnost

Vlastnost umožňuje popsat jednotlivé entity. Entita sama o sobě není nijak charakterizována, jedná se pouze o proměnnou s jedinečným ID a přiřazením. Pomocí vlastností entitám přidělujeme jednoznačné popisy, definice a charakteristiky.

Každá vlastnost je definována pomocí tří parametrů: název, typ, a hodnota.




Obrázek 6: Schéma vlastnosti

Název umožňuje pojmenovat danou vlastnost. Typ udává, v jakém formátu je daná vlastnost zobrazována. V systému jsou využity základní typy: textový řetězec a číslo. Hodnota je konkrétní vyjádření dané vlastnosti.

U každé vlastnosti lze navolit, zda se uživateli zobrazí defaultní název, resp. ID entity, nebo název zvolený uživatelem.

Příklady vlastností jsou uvedeny v tabulce:

| | Název | Typ | Hodnota |
|---|---|---|---|
| Příklad 1 | Název | Textový řetězec | Leucinóza |
| Příklad 2 | OMIM | Celé číslo | 248600 |
| Příklad 3 | Klasifikace | Textový řetězec | Porucha metabolismu aminokyselin |

Tabulka 1: Tabulka vlastností

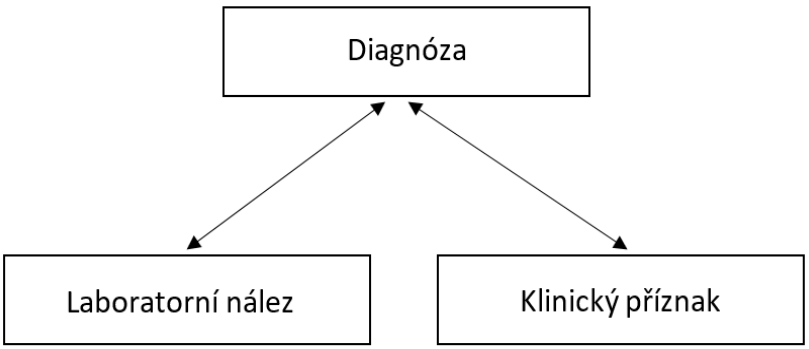
### 5.1.4 Vazba

Vazbu mezi entitami lze nazvat jako relace a vyjadřuje skutečnost, kolik výskytů jedné entity může vstoupit do vztahu s kolika výskyty druhé entity. Existují tři typy vazeb vztahů *1:1*, *1:N*, *N:M*. V našem systému je využívána relace N:M, především z hlediska univerzálnosti systému. Tato relace říká, že několik entit z jedné kategorie může mít vazbu k několika entitám z kategorie jiné.

V ontologické terminologii je definován pojem vztah, který je v systému roven pojmu vazba. Jedná se o speciální typ vlastnosti, jejíž hodnotou je jiná entita.

Vazby jsou v systému vytvářeny mezi proměnnými stejného typu. Může být tedy vytvořena vazba mezi dvěma kategoriemi nebo dvěma entitami, ale nelze vytvořit vazbu mezi kategorií a entitou. Provázání vazeb mezi proměnnými je nastaveno tak, že nejprve je vytvořena vazba mezi určenými kategoriemi a poté při vzniku konkrétních entit je možno provázat entity vzájemně mezi sebou.

Pokud bychom to uvedli na příkladu modelového zdravotnického systému, v němž máme kategorie *diagnóza*, *klinický příznak* a *laboratorní nález*, pak by provázání mezi kategoriemi vypadalo následovně:




Obrázek 7: Schéma vazeb v modelovém systému

Následně při tvorbě entit daných kategorií bude možné mezi sebou provázat již konkrétní entity. Například je možné vytvořit vazbu mezi diagnózou *leucinóza* a laboratorním nálezem *křeče*.

Jak je vidět na Obr. 9, vazby v systému jsou v relaci symetrie. Formálně definováno:

**Definice 5 (relace symetrie)**

$\forall a,b \in X,\ aRb \Rightarrow bRa$, kde množina X je množinou všech proměnných (kategorií a entit), a $a$, $b$ jsou konkrétní proměnné.

To znamená, že pokud uživatel vytvoří vazbu z *leucinóza* na *křeče*, vznikne automaticky vazba ve směru opačném.

# 6 Implementace

Předchozí kapitola se věnovala spíše teoretické stránce modelu a jeho popisu. Tato kapitola je zaměřena na popis použitých technologií a především samotnou implementaci systému. Systém byl vyvíjen pro operační systém Windows. Budou zde rozebrány a popsány hlavní funkce systému a jejich řešení. Mezi hlavní funkce patří filtrace, s níž souvisí zavedení a práce s vazbami, které jsou uloženy v souborech a načítány z nich.

V předchozí kapitole (4.1.) byl popsán přístup k datům neúplným a jejich následnému zúplnění a transformaci. V našem IS se práce s daty liší a to v tom, že chybějící informace nejsou do databáze nijak doplňovány ani ničím nahrazovány, ale pracuje se s nimi jako s neznámou proměnnou.

## 6.1 Použité technologie

Kapitola obsahuje stručný popis technologií použitých k vývoji informačního systému. Jako vývojové prostředí bylo použito Visual Studio 2019 a programovacím jazykem je C#.

### 6.1.1 C#

Jazyk C# byl vyvinut firmou Microsoft zároveň s vývojovým prostředím .NET a vychází z programovacího jazyka C++/C a v mnoha ohledech se velmi blíží jazyku Java. Jazyk C# je čistě objektově orientovaný a nepodporuje vícenásobnou dědičnost. Správa paměti u C# je automatická – o korektní uvolňování zdrojů aplikace se stará garbage collector. Kromě členských dat a metod přidává také vlastnosti a události. O překladačích jazyka C# platí, že jsou case sensitive.

### 6.1.2 XML

XML je zkratka z anglického eXtensible Markup Language, rozšiřitelný značkovací jazyk. Ve skutečnosti se ovšem jedná o nadřazený značkovací jazyk, v jehož rámci si je možné vytvářet vlastní jazyky. Neobsahuje žádné konkrétní značky a můžeme si tedy vymyslet vlastní, což umožňuje velice dobře definovat přesnou strukturu každého XML dokumentu podle potřeby. V dnešní době se používá především pro snadnou výměnu informací a komunikaci nezávislou na konkrétní aplikaci. Hlavní výhodou XML oproti jiným formátům je jeho nezávislost, standardizace, poměrně malá velikost a podporování národních kódování či jednoduchý převod na jiné formáty. [20]

### 6.1.3 XAML

XAML je zkratka pro Extensible pplication Markup Language. Je to deklarativní jazyk založený na XML. Primárně je určen k návrhu uživatelského prostředí pro technologie WPF a Silverlight, ale můžeme jej použít téměř na cokoli chceme a

není vázán pouze na WPF, ačkoli je v této formě nejpoužívanější. XAML slouží k usnadnění zápisu a ve WPF primárně slouží sestavení objektového stromu elementů uživatelského prostředí (XAML totiž pracuje s objektovým modelem .NET). [21]

### 6.1.4 WPF

Technologie Windows Presentation Foundation (WPF) je rozhraní pro návrh a zobrazování uživatelského prostředí. Jedná se o nástupce starší technologie Windows Forms (WinForms). WPF je součástí .NET frameworku od verze 3.0 a umožňuje tvorbu rozsáhlých formulářových aplikací. Nabízí spoustu hotových komponent jako jsou pole, tlačítka či popisky a díky tomu lze formulář snadno poskládat, případně si můžeme tyto komponenty vytvořit sami a vlastní. [22]

### 6.1.5 CSV

Formát CSV (comma separated value, hodnoty oddělené separátorem) se již delší dobu používá pro výměnu a převod dat mezi různými tabulkovými procesory. Díky jednoduchosti, nenáročnosti a čitelnosti i bez specializovaného souboru se tento formát používá i pro výměnu informací mezi systémy. [23]

Soubor ve formátu CSV obsahuje řádky, ve kterých jsou jednotlivé položky odděleny separátorem (čárka, středník, tabulátor, ...). Hodnoty položek mohou být zapsány v uvozovkách v případě, že v rámci jedné hodnoty chceme zapsat více hodnot oddělených čárkami (ta by totiž mohla být brána jako separátor).

I přes svou jednoduchost nebyl tento formát nikdy zdokumentován. Přesto jsou zavedeny jisté specifikace se společnými pravidly:

- každý záznam je umístěn na samostatném řádku ohraničeném řádkem
- poslední záznam v souboru může, ale nemusí mít koncový řádek break
- jako první řádek souboru se může objevit nepovinný řádek záhlaví ve stejném formátu jako řádky běžných záznamů

## 6.2 Princip vazeb v systému

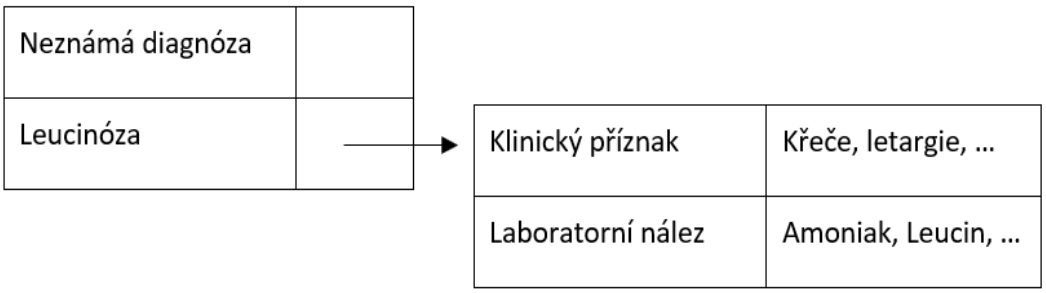
Úloha vazeb v systému je velmi klíčová a uplatňuje se zejména při filtrační a deduktivní funkci systému.

Prozatím je v systému uveden pouze jeden typ vazby, a to vazba typu „mít“, nebo jinak řečeno „vyskytuje se“. Z důvodu široké univerzálnosti systému není možné přesně pojmenovat vazbu mezi entitami a v rámci různých systémů se její význam může lišit. Jedná se o obecnou vazbu mezi dvěma entitami, jejíž pojmenování nemá v systému žádný význam ani funkci. Opakem prvního typu vazby v systému je její vyloučení, resp. zavedení, že dvě entity spolu rozhodně nesouvisejí.

### 6.2.1 Struktura vazeb

Pro práci s vazbami jsem se snažil najít co nejefektivnější způsob jejich ukládání a zpracování. Nakonec jsem zvolil práci se slovníkem, což je generická kolekce ukládající páry ve tvaru klíč-hodnota v nespecifikovaném pořadí. Obecně platí, že klíče musejí být jedinečné a nesmí mít hodnotu *null*, naopak hodnoty mohou být nulové a nebo i dokonce duplicitní. Ve slovníku můžeme ukládat pouze stejné typy prvků.

V našem systému je využita kombinace dvou slovníků (Obr. 8). První slovník je generický (dále označován jako *generický slovník*) a klíčem může být jakýkoli datový typ. V našem případě je to textový řetězec s entitou, k níž je vytvářena vazba. Hodnota prvního slovníku je druhý slovník (dále označován jako *slovník vazeb*). V něm je vždy klíčem název kategorie ve tvaru textového řetězce a hodnotou je seznam entit s již existující vazbou na danou entitu. V případě vzniku nové vazby je jednoduše seznam existujících vazeb obohacen o vazbu další.




Obrázek 8: Struktura slovníku vazeb

Jinou možností by bylo uložit veškeré již existující vazby rovnou do hodnoty generického slovníku, bez tvorby další dvojice klíč a hodnota. V takovém případě by se ale zvýšila prohledávací doba systému, protože by musel procházet stejný seznam vícekrát místo toho, aby rovnou hledal v kategorizovaných seznamech.

### 6.2.2 Práce s vazbami

V systému je k dispozici třída *BindingTableManager*, s jejíž pomocí jsou naimplementovány všechny důležité funkce pro práci s vazbami a jejich soubory. Součástí třídy jsou 2 slovníky, jejichž struktura byla popsána výše. Jeden z nich je pro vazby existující, druhý pro vazby vylučující. Struktura obou slovníků je stejná.

Hlavní náplní třídy *BindingTableManager* je práce s těmito slovníky. Mezi ztěžejní funkce třídy patří:

- *ukládání vazeb*
- *odstraňování vazeb*
- *hledání v tabulkách vazeb podle zadaného ID entity* (Zdrojový kód 1)

```
1  public List<EntityTuple> GetBindingValuesById(string id)
2  {
3     Entity entity = null;
4     foreach (var i in entityValues)
5        if (i.Key.InstanceId == id)
6           entity = i.Key;
7
8     if (entity != null)
9        return entityValues[entity];
10    return null;
11 }
```

Zdrojový kód 1: C#

Funkce vrací hodnotu *EntityTuple*, což je reprezentace slovníku vazeb. *entityValues* je proměnná představující generický slovník. Funkce vyhledá v generickém slovníku entitu (klíč) odpovídající zadanému ID. Pokud je entita nalezena, tak vrátí hodnotu odpovídající nalezenému klíči generického slovníku. Výsledkem je seznam všech existujících / vyloučených vazeb daného klíče.

Hledání v tabulkách vazeb podle zadaného ID entity se využívá v systému velmi často. Jednou z možností je při filtraci, která bude popsána dále, nebo při zobrazování detailu entity, zadávání nových vazeb či jejich úpravě.

## 6.3 Práce se soubory

Databáze systému je uchována ve formě souborů. Soubory jsou rozděleny do patřičných skupin pro zvýšení efektivity vyhledávací funkce systému.

Hned po vzniku, resp. vytvoření, systému dochází ke tvorbě řady souborů. Soubory jsou dvojího typu. Prvním typem jsou soubory kódující vazební tabulky, druhým typem jsou soubory entit.

Stejně jako celý systém je i formát souborů pojat jednoduchou, srozumitelnou a univerzální formou. Cílem bylo umožnit uživateli do souborů přímo zasahovat a upravovat je. Díky tomu má možnost pomocí souborů nahrát data do systému, aniž by je tam musel ručně vyplňovat, nebo si takto dokonce vytvořit celý systém.

## 6.4 Rozdělení souborů

Prvním typem souboru jsou soubory popisující jednotlivé entity uloženy v adresáři kategorií, které jsou obsaženy v hlavním adresáři celého systému. Všechny jsou vedeny ve formátu XML.

V systému je zavedena jednoduchá hierarchie: systém -> kategorie -> entita. Stejným způsobem jsou vedeny i cesty přes soubory a adresáře, od nejvýše postaveného po nejnižší (od systému po entitu).

### 6.4.1 Adresáře kategorií

Adresáře kategorií jsou v hierarchii druhé v pořadí. Jedná se o adresář obsahující soubory s entitami, které budou podrobněji popsány dále, a soubor *define*. Ten vzniká ihned po vytvoření kategorie a obsahuje následující vlastnosti:

- *ID*
- *Unknown*
- *Vlastnosti entit dané kategorie*

Vlastnost *unknown* je speciálním typem vlastnosti typu *boolean*. Ta uvádí, zda je entita dané kategorie neznámá či známá.

Vlastnosti entit dané kategorie jsou všechny ostatní vlastnosti, které si uživatel sám definuje. Po jejich definování jsou uloženy do tohoto souboru a jsou uplatňovány na všechny entity stejné kategorie bez výjimky. Pokud je vytvořena nová vlastnost k entitě, dojde k přenosu této vlastnosti i na ostatní, již dříve vytvořené entity, a bude mít prázdnou hodnotu. Stejně tak pokud je některá z vlastností změněna, soubor *define* zajistí, že se změna projeví globálně v rámci kategorie.

Příklad XML souboru *define* pro kategorii diagnóza:

```
<?xml version="1.0" encoding="utf-8"?>
<Diagnóza>
  <Id type="text" view="False" />
  <Unknown type="text" view="False" />
  <Název type="text" view="True" />
  <Klasifikace type="text" view="False" />
  <OMIM type="integer" view="False" />
  <Léčba type="text" view="False" />
  <Zkratka type="text" view="False" />
  <Dědičnost type="text" view="False" />
</Diagnóza>
```

Zdrojový kód 2: C#

Každá vlastnost obsahuje atribut *type* a *view*. *type* je typ vlastnosti (textový řetězec, celé číslo) a *view* udává, zda hodnota dané vlastnosti bude zobrazena uživateli v seznamu entit.

### 6.4.2 Soubory entit

Soubor ke konkrétní entitě vzniká v okamžiku, když uživatel v systému danou entitu vytvoří. Druhou možností vzniku je, že si uživatel může soubor entity napsat sám a tak do systému nahrát data.

Při vzniku nové entity (Zdrojový kód 3) dojde k vygenerování unikátního ID entity a je určena speciální vlastnost *unknown* říkající, zda je daná proměnná neznámá. U známých proměnných je hodnota nastavena na *false*. Speciální vlastnosti (ID, unknown) nejsou pro uživatele viditelné.

Následně jsou k těmto dvěma speciálním vlastnostem přidány další vlastnosti entity (Zdrojový kód 3) definované v souboru *define* dané kategorie, které už jsou uživateli zobrazeny v systému. V souboru *define* jsou vyplněny pouze názvy vlastností, ale ne jejich hodnoty.

Seznam vlastností ze souboru *define* je reprezentován proměnnou *DefineProperties*. Hodnoty vlastností jsou uloženy v souborech entit a proto byla vytvořena další proměnná typu seznam *propertieValues*, která vyhledá a přiřadí hodnoty vlastností v souborech entit. Seznam entit je reprezentován proměnnou *Entities*.

Pokud vytvoříme entitu novou, budou mít hodnoty jejích vlastností zprvu nulovou hodnotu, kterou musí uživatel vyplnit. Při dalším zobrazení nebo úpravě entity se zobrazí vyplněné hodnoty díky proměnné *propertieValues*.

```
1  public void AddEntity(List<string> propertieValues)
2  {
3     string entityName = directoryManager.ActualDirectory;
4
5     string uniqueId = CreateUniqueId(entityName, Entities.Count);
6     Property idProperty = GetEmptyIdProperty();
7     idProperty.Value = uniqueId;
8     Property unknownProperty = GetUknownProperty();
9     unknownProperty.Value = "False";
10
11    Entity e = new Entity();
12    e.EntityName = entityName;
13    e.InstanceId = uniqueId;
14    e.Properties.Add(idProperty);
15    e.Properties.Add(unknownProperty);
16
17    for (int i = 0; i < propertieValues.Count; i++)
18    {
19       Property p = new Property();
20       p.Name = DefineProperties[i].Name;
21       p.Type = DefineProperties[i].Type;
22       p.View = DefineProperties[i].View;
23       p.Value = propertieValues[i];
24       e.Properties.Add(p);
25    }
26
27    directoryManager.AddEntityFile(e, uniqueId);
28    Entities.Add(e);
29 }
```

Zdrojový kód 3: C#

Soubory entit jsou stejně jako soubory *define* ve formátu XML. Jsou tedy zapsány i stejným způsobem (viz. zdrojový kód 2) s tím rozdílem, že jednotlivé hodnoty vlastností jsou vyplněny konkrétními údaji.

## 6.5 Soubory s vazbami

Jak vyplývá z názvu, soubory s vazbami obsahují soubory kódující vazby mezi entitami / kategoriemi. V systému jsou zavedeny 3 následující soubory:

- *SetBinding*
- *BindingTable*
- *NotBindingTable*

Soubor *SetBinding* zahrnuje seznam vytvořených vazeb mezi kategoriemi, soubor *BindingTable* obsahuje záznamy o existujících vazbách mezi entitami a

soubor *NotBindingTable* obsahuje soubor s vazbami vyloučenými. První zmíněný typ souboru je nutný k tomu, aby mohli vzniknout další dva soubory. V případě, že není vytvořena vazba mezi kategoriemi, nemůže dojít ke vzniku vazeb mezi entitami z těchto kategorií.

Soubory *BindingTable* a *NotBindingTable* využívají principu vazebních tabulek ale místo tabulek se zde pracuje se soubory. To znamená, že ukládají vazby mezi dvěmi různými entitami zapsanými ve dvou různých souborech entit.

Data jsou v těchto souborech zapsány ve formátu CSV. Využívám v nich oddělení jednotlivých hodnot pomocí separátoru, konkrétně středníku, přičemž oddělovaným hodnotami jsou ID entit. V případě souboru *SetBinding* jsou oddělovaným hodnotami ID kategorií. Vzhledem k jednoduchosti zápisu není nutné využívat popisné hlavičky a v souboru jsou uvedeny jen řádky s dvojicemi. ID entity je zapsáno formou: název kategorie a pořadového čísla, oddělené separátorem pomlčky.

Druhou možností by bylo generovat ID pomocí *GUID*. V systému jsem zvolil jednoduchou a pochopitelnou formu ID z toho důvodu, že v případě nevyplněného názvu entity se uživateli zobrazí ID, které by mělo být srozumitelné. Druhým důvodem je poskytnout uživateli možnost definovat data přímo v souborech.

## 6.6 Neznámé proměnné

Práce s neznámou hodnotou je centrálním aspektem celého systému, který s ní pracuje jako s neznámou proměnnou. Ta je řešena pomocí speciálního typu vlastnosti *unknown*, která je pro uživatele skryta. Je navolena pouze na začátku procesu vzniku entity, kdy uživatel danou entitu jako neznámou označí. Systém si pak tuto informaci uloží do souboru a dále s ní pracuje.

Neznámá proměnná je v systému považována za běžnou entitu, liší se pouze hodnotou speciální vlastnosti *unknown*, která je nastavena na *true*.

### 6.6.1 Tvorba neznámé entity

Neznámá proměnná je vytvořena na základě žádosti uživatele, který si tuto možnost navolí v systému. Proces vzniku neznámé entity je téměř identický jako vznik entity známé. Rozdíl spočívá v hodnotě speciální vlastnosti *unknown* a v tom, že v systému je pro tvorbu neznámé entity speciální funkce, která ihned vytvoří neznámou entitu se všemi vlastnostmi dané kategorie s nulovou hodnotou. Tyto vlastnosti lze upravovat teprve v rámci editace entity.

Dva rozdílné procesy pro vznik entit jsem zvolil z předpokladu, že znalosti hodnot vlastností u neznámých entit budou velmi omezené a do systému budou doplňovány průběžně.

### 6.6.2 Přidání vazby k neznámé entitě

Princip přidávání a práce s vazbami funguje u neznámé entity úplně stejně, jako je tomu u entit známých. Tvorba vazeb u neznámých entit má ale mnohem větší význam. Jsou totiž využívány při dedukční funkci systému, kdy systém sám na základě zadaných vazeb navrhuje výsledek neznámé entity z řady entit již známých a definovaných.

## 6.7 Filtrační funkce systému

Jednou z hlavních funkcí IS je vyhodnocovat dotazy, které jsou v systému řešeny formou filtrace. Při filtraci jsem vycházel z modální logiky, resp. jsem v systému využil jejích modálních operátorů: „je možné, že“ a „je nutné, že“. Výsledkem filtrace jsou dva druhy výstupů, první z nich je „nutným“ výsledkem dotazu, druhý „je možný“ za určitých podmínek. Nutnost výsledku je ovlivněna počtem vložených dat a platí, že více informací a dat vytvořený IS má, tím přesnější výsledek je schopen vyhodnotit.

V systému je uplatněna filtrace dvojího typu. První je klasická filtrace se samostatnou položkou v menu systému a druhá, pro odlišení od filtrace označována jako dedukce, se vyskytuje přímo v editačním okně neznámých entit.

### 6.7.1 Filtrace

Klasická filtrace je nastavena tak, že nejprve uživatel zvolí kategorii, kterou chce filtrovat. Následně má možnost filtraci provést podle dvou parametrů: podle vlastností nebo podle entit.

V případě filtrace podle vlastností dochází ke klasickému porovnání textových řetězců a vyhodnocení shody. Vyfiltrované entity jsou pouze ty, které zadanou vlastnost, nebo její část, splňují. Filtrace podle entit už je složitější a právě u ní se využívá modálních operátorů „je možné, že“ a „je nutné, že“.

Pokud byla zadána filtrace podle obou parametrů, entity i vlastnosti, pak je nejprve provedena filtrace podle entity, a mezivýsledek je následně specifikován filtrací podle vlastnosti. Pokud je prováděna filtrace podle jen jednoho parametru, provede se pouze postup pro příslušný parametr.

#### Filtrace podle entit

Na začátku je nutné zvolit entitu / entity, podle kterých mát být filtrace provedena. V případě pouze jedné entity je postup samozřejmě jednodušší. Systém projde generační slovník a hledá klíč roven filtrované entitě. Při nálezu takového klíče je výsledkem filtrace seznam entit uvedených ve slovníku vazeb příslušného klíče, bez nutnosti dalších kroků.

Pokud jsou ve filtraci zvoleny dvě a více entit, postup je vícekrokový a využívá se zde matematického průniku množin:

```
1  List<Entity> resultIds = new List<Entity>();
2  foreach (Entity instance in listOfFilteredEntities)
3  {
4    List<EntityTuple> listOfBounds = manager.GetBindingValues(
         instance.InstanceId);
5    EntityTuple bounds = listOfBounds.Where(i => i.Name ==
         categoryName).FirstOrDefault();
6
7    resultIds = Intersection(resultIds, bounds.Values);
8  }
```

Zdrojový kód 4: C#

Množina výsledných hodnot *resultIds* je na začátku prázdná a postupně se do ní ukládají filtrované výsledky. Funkce *GetBindingValues* prohledává generický slovník a hledá v něm entitu, podle níž je filtrováno. Poté se do proměnné *listOfBounds* uloží slovník vazeb vyhledané entity. Z tohoto slovníku jsou pak vybrány pouze ty vazby, které odpovídají kategorii zvolené na počátku filtrace. Následně se do proměnné *resultIds* vloží průnik hodnot z proměnné *resultIds* a nově nalezených hodnot. Tento postup se opakuje tolikrát, kolik je zadaných entit ve filtraci. Po každém cyklu se počet hodnot v proměnné *resultIds* sníží a upřesní právě pomocí průniků. Konečné hodnoty v této proměnné jsou zobrazeny uživateli jako hodnoty s „nutností“ výskytu. Mohou zde být i neznámé entity ale pouze v případě, že mají vazbu na všechny filtrované entity současně.

Výsledek filtrace vychází zásadně z vazeb mezi entitami, ať už existujícími nebo vyloučenými. Pokud entita nemá žádnou vazbu na jinou entitu, pak nemůže být v rámci filtrační funkce systému vybrána za výslednou hodnotu.

### Filtrace neznámé entity

V případě, že neznámé entity neobsahují všechny vazby na filtrované entity, pak získáváme druhý typ výsledku s „možností“ výskytu. Pro filtraci neznámých entit je samostatný kód (Zdrojový kód 5), který pracuje pouze s entitami mající hodnotu speciální vlastnosti *unknown* nastavenou na *true*.

```
1  List<Entity> bind = GetListOfFilteredEntities();
2  List<Entity> unknowns = GetUnknowns();
3
4  foreach (var i in unknowns)
5  {
6     countOfBind = 0;
7     List<EntityTuple> tuples = GetBindingValues(i.InstanceId);
8     foreach (var j in tuples)
9        foreach (var k in j.Values)
10          if (bind.Contains(k.InstanceId))
11             countOfBind++;
12
13    bool not = false;
14    tuples = GetNotBindingValues(i.InstanceId);
15    foreach (var j in tuples)
16       foreach (var k in j.Values)
17          if (bind.Contains(k.InstanceId))
18             not = true;
19
20    if (countOfBind != bind.Count && !not)
21       result.Add(i);
22 }
```

Zdrojový kód 5: C#

Systém postupně prochází všechny neznámé entity a pomocí metody *GetBindingValues* si načte jejich slovníky vazeb do proměnné *tuples*. Tuto proměnnou pak porovnává s proměnnou *bind*, což je seznam filtrovaných entit. Proměnná *countOfBinds* udává počet shodných entit, které byli při srovnání nalezeny. Pokud je počet nalezených entit stejný jako počet filtrovaných entit, pak se jedná o výsledek s „nutností výskytu“. Pokud je počet nalezených entit nižší než počet filtrovaných, jedná se o výsledek s „možností“ výskytu. Jinak řečeno tento výsledek bude „možný“ pouze za splnění určitých podmínek (vazeb).

Podobný postup je uplatněn i na slovník vyloučených vazeb a metodu *GetNotBindingValues*. Nejprve je slovník uložen do proměnné *tuples* a poté se srovnávají entity ve slovníku a filtrované entity. V případě, že slovník obsahuje alespoň jednu shodnou entitu, pak je jisté, že neznámá entita nebude mít ani „možnost“ výskytu a ve výsledných hodnotách se vůbec nezobrazí.

### Filtrace podle vlastností

Na začátku je vytvořen seznam buď všech entit v systému, nebo všech entit z předchozí filtrace podle entit. Tento seznam je pak porovnáván s filtrovanými vlastnostmi. Může se jednat o celé hodnoty vlastností, nebo jen jejich část (ve formě textového řetězce nebo čísla). Pokud je zadána jen jedna vlastnost, jedná se o jednoprůchodový proces seznamem. V případě zadání více filtrovaných vlastností je proces vícekrokový a opět se zde uplatní průnik výsledků.

### 6.7.2 Dedukce

Kromě klasické filtrační funkce je schopen systém vykonávat i funkci dedukční. Stejně jako filtrace vychází z vazeb mezi entitami. Dedukce neslouží k vyhledávání výsledků, ale jedná se o schopnost systému definovat neznámé entity na základě vytvořených vazeb na ostatní entity.

Jak bylo zmíněno dříve, nově vytvořená entita má vlastnosti s nulovou hodnotou a nemá žádné vazby na další entity. V takovém případě systém jako výsledek nabídne seznam veškerý existujících entit, známých i neznámých. S postupným přidáváním existujících a vyloučených vazeb dochází k navýšení informací, se kterými systém může pracovat a jeho dedukční schopnost se zvyšuje. To znamená, že čím více vazeb je z neznámé entity vytvořeno, tím méně výsledků je v seznamu entit s možnou ekvivalencí na danou neznámou entitu.

Při hledání vazeb je využito slovníků vazeb existujících a vyloučených. V případě nálezu shody ve slovníku existujících vazeb je příslušná entita přidána do výsledného seznamu entit. Naopak při shodě ve slovníku vyloučených vazeb je příslušná entita ze seznamu vyloučena.

Dovolte mi uvést příklad dedukce na modelovém systému pro zdravotnickou laboratoř, kde je zavedena kategorie *Diagnóza*. Jejími entitami jsou jednotlivá onemocnění, které můžou být buďto známé, nebo neznámé. U neznámé entity bude uveden seznam s možnými známými onemocněními, které jsou spojeny vazbami na stejné klinické příznaky a laboratorní nálezy, jako u neznámé entity. Nejprve budou výsledkem všechny onemocnění v seznamu entit, ale s narůstajícím množstvím vazeb u neznámé entity se bude seznam možných onemocnění zmenšovat a zpřesňovat.

# 7 Uživatelská příručka

Uživatelská příručka slouží pro popis hlavních funkcí informačního systému a uživatel má možnost se zde zorientovat v systému. Může zde vyčíst základní funkce a jak se systémem pracovat. Mezi hlavní funkce systému patří tvorba databáze ve formě souborů a vyhodnocování dotazů nad příslušnými daty pomocí filtrace a dedukce.

## 7.1 Instalace a spuštění

Informační systém lze přímo spustit ze složky Application (soubor SystenUnknownValue.exe) v adresáři bin na CD/DVD, nebo si jej nainstalovat. Instalaci lze provést pomocí instalačního souboru Setup. Poté se na ploše objeví ikona se zástupcem. Jako první se otevře vstupní okno aplikace. To obsahuje seznam systémů, které náš informační systém zpracovává.

Na začátku je systém prázdný a neobsahuje žádný vytvořený systém ani data. Součástí CD s prací je i složka **data/**. Ta obsahuje složku *Nemocnice* se vzorovými daty pro systém. Pro použití těchto dat je nutné složku *Nemocnice* po spuštění aplikace přesunout do složky s daty aplikací na disku C uživatele. Do složky s daty aplikací se lze dostat zapsáním *%appdata%* do cesty průzkumníka a tam otevřít složku s názvem systému *UnknownSystem*.

Po zvolení již existujícího systému nebo vytvoření nového se uživateli zobrazí hlavní okno aplikace. Při tvorbě nového systému dochází ke vzniku řady souborů. Při rychlém spuštění nového systému se může objevit okno s hlášením, že tyto soubory se teprve tvoří a nelze je tedy načíst. Po odkliknutí hlášky je možné systém spustit.

## 7.2 Hlavní okno a menu

V levé části systému je menu rozděleno na dvě části. V horní části jsou nástroje a funkce systému, ve spodní je seznam kategorií s možností přidání nové kategorie. Samotné hlavní okno je prázdné, slouží pouze pro zobrazování položek z menu.

## 7.3 Vazby

Položka *Vazby* v menu zobrazí okno se seznamem vazeb mezi kategoriemi. Nejprve je nutné, aby uživatele vytvořil vazby mezi kategoriemi, aby mohl následně definovat vazby mezi entitami z daných kategorií.

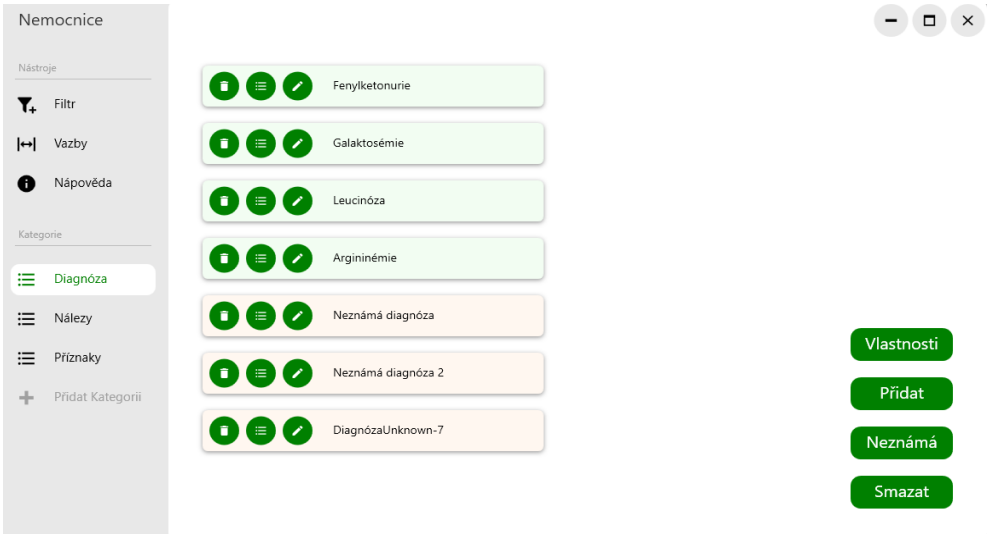
Vazby mezi kategoriemi lze definovat ve spodní části okna. Nelze navolit vazbu mezi stejnými kategoriemi.

## 7.4 Okno se seznamem entit a správa entit

Po zvolení položky *Kategorie* v menu se uživateli zobrazí okno se seznamem entit dané kategorie a možností spravovat jednotlivé entity.

V pravém spodním rohu se nachází skupina tlačítek pro manipulaci s entitami. Tlačítko *Vlastnosti* přesměruje uživatele na okno se seznamem vlastností a možností definování vlastností nových. Každá vlastnost má dva parametry: název a typ. Součástí okna vlastností je checkbox pro volbu, zda se název zadané vlastnosti bude zobrazovat v seznamu entit (jako součást popisu).

V systému lze vytvořit dva typy entit pomocí dvou různých tlačítek. Tlačítko *Přidat* slouží pro tvorbu známé entity a po kliknutí na něj se uživateli rovnou zobrazí okno, ve kterém může vyplnit hodnoty vlastností entity. Tlačítko *Neznámá* vygeneruje novou neznámou entitu, která je přidána do seznamu entit ve formě ID (kombinace názvu kategorie a pořadového čísla entity) a teprve po zobrazení jejího detailu je možné jí přidělit hodnoty vlastností a vazby.



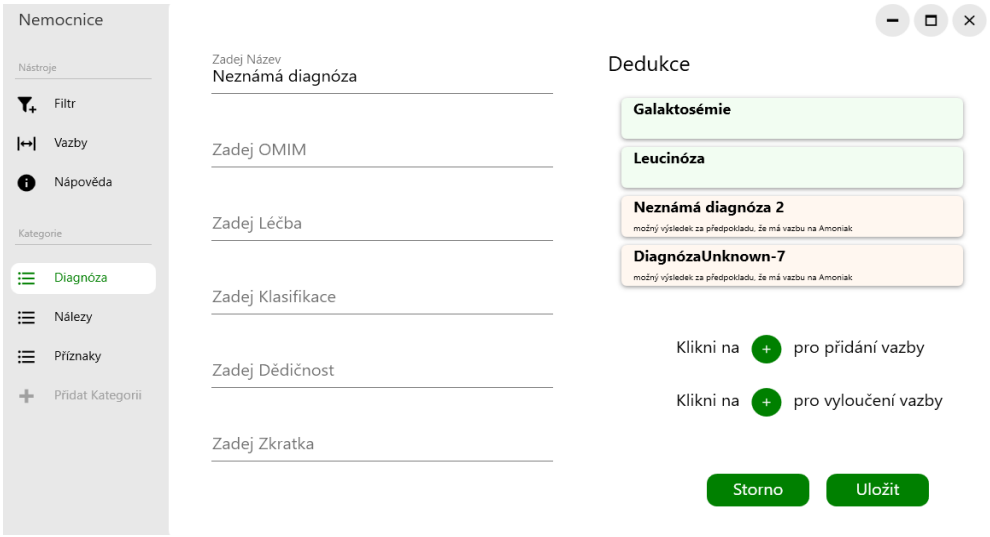

Obrázek 9: Okno se seznamem entit

Na Obr. 11 je vidět seznam 4 známých entit a 3 neznámých. Známé entity jsou napříč systémem zobrazeny na zeleném podkladu, neznámé entity na oranžovém. První dvě neznámé entity jsou už upraveny, poslední je nově vygenerovaná a ještě neupravená.

U jednotlivých entit v seznamu je ikona pro její smazání, zobrazení detailu a úpravu. Ikona pro úpravu umožňuje přidat k entitě vlastnosti (existující nebo vylučující) a úpravu hodnot vlastností.

## 7.5 Neznámá entita a dedukce

Po kliknutí na ikonu úprava u neznámé entity se uživateli zobrazí okno entity obsahující seznam vlastností, dedukční funkci systému a tlačítka pro přidání existujících či vyloučených vazeb. Dedukce (Obr. 10) je jednou z hlavních funkcí systému a zobrazuje uživateli možné výsledky zvolené neznámé entity ve formě seznamu. Seznam obsahuje již vytvořené známé i neznámé entity a je tvořen na základě vazeb neznámé entity. Platí, že čím více vazeb uživatel vytvoří a definuje, tím je dedukce přesnější a seznam obsahuje méně výsledků.



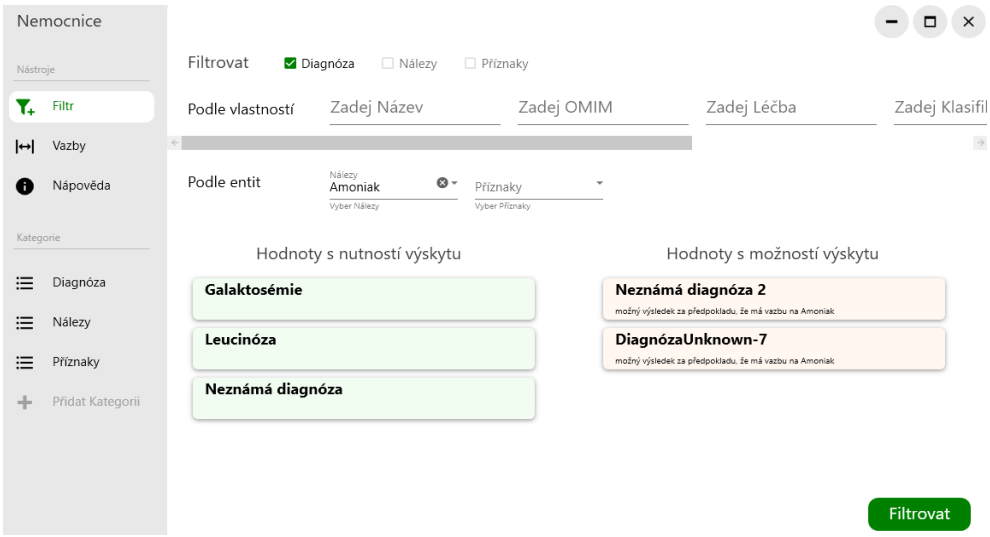

Obrázek 10: Neznámá entita a její dedukce

## 7.6 Filtrace

Druhou hlavní funkcí systému, kromě výše zmíněné dedukce, je filtrace. Ta dává uživateli možnost aplikovat na systém dotazy, které jsou vyhodnocovány na základě vazeb mezi entitami.

V horní části okna je zobrazen seznam kategorií ve formě checkboxů. Uživatel si zvolí kategorii, kterou chce vyhledávat a filtrovat.

Po zvolení kategorie se zobrazí 2 druhy filtrace: podle vlastností a entit.




Obrázek 11: Filtrace

Výsledkem filtrace jsou dva druhy výstupů: hodnoty s nutností výskytu a hodnoty s možností výskytu. Hodnoty s nutností výskytu jsou entity, které splňují všechny zadané parametry a mohou obsahovat známé a neznámé entity. Hodnoty

s možností výskytu obsahují pouze neznámé entity, které by mohli být hledanou hodnotou při splněný určitých podmínek (vazeb).

## 7.7 Náhled z praxe

Z důvodu univerzálnosti a jednoduchosti systému jsem chtěl zjistit, na jaké cílové skupiny a jaké všemožné využití by systém mohl mít. Se svou žádostí jsem uspěl ve zdravotnické laboratoři, kde byl systém vyzkoušen pracovníkem z praxe, odkud také pocházejí vzorová a modelová data. Níže uvádím zhodnocení a připomínky Mgr. Michaely Kotkové, pracovníka laboratoře dědičných metabolických poruch FN Olomouc.

Měla jsem příležitost vyzkoušet informační systém z neznámou hodnotou a aplikovat jej na konkrétní zaměření. Při popisu systému mě ihned napadlo využití ve zdravotnické laboratoři, zaměřené na diagnostiku. V naší laboratoři je cílem určit diagnózu pacientů. Diagnóza probíhá na základě klinických příznaků a laboratorních nálezů. Obě tyto komponenty je nutné spojit a teprve jejich kombinací je možné se dopátrat správné diagnózy. Ta je vždy na počátku neznámá a výslednou možností je kterákoli z existujících diagnóz, a teprve postupem času získáváme informace o laboratorních nálezech a klinice pacienta.

Informační systém s neznámou hodnotou umožňuje takové případy zpracovat a mít potřebné údaje pro stanovení diagnózy na jednom místě. V systému jsem měla uloženou sadu základních onemocnění spolu s neznámými diagnózami. Postupným přidáváním informací do systémů mi systém vyhodnocoval stále méně možných onemocnění daného případu.

V rámci dokončení případu a konečného zjištění diagnózy bych v systému uvítala možnost spárování neznámé diagnózy s některou ze známých, nebo její přetvoření do diagnózy nové. Vzhledem k množství onemocnění, které jsou předmětem naší laboratoře, se může stát, že nějakou diagnózu stanovíme poprvé. V takovém případě bych ji vytvořila z už nadefinované a vyplněné neznámé a nemusela bych ji vytvářet od začátku se všemi vlastnosti znovu.

V rámci efektivnější diagnostiky by bylo také užitečné mít možnost do filtrace zadat více než jeden příznak a nález.

Celkově systém shledávám do praxe užitečným a praktickým. V systému jsem se rychle zorientovala a práce sním byla pohodlná.

# Závěr

Cílem práce bylo vyvinout informační systém s neznámými hodnotami. Původně byl systém cílem na malé firmy, ale v průběhu vývoje se z něj stal velmi univerzální nástroj, který je možné použít pro široké spektrum zaměření a oblasti zájmů. Při návrhu systému bylo cíleno především na univerzálnost a jednoduchost systému.

Centrálním aspektem systému je práce s neznámou hodnotou. V systému je možné vytvořit různé entity, kategorie a vazby, přičemž kterákoli z entit může být neznámou hodnotou.

Vytvořený systém je schopen s neznámou hodnotou pracovat a vyhodnocovat dotazy na systém, především formou filtrace. Výsledky filtrace jsou zpracovány na základě vazeb mezi entitami, které vytváří v systému uživatel, nebo vlastností entit. Výsledkem dotazu jsou dva výstupy, jeden s nutností výskytu, druhý s možností výskytu. V obou typech výstupů se mohou objevovat známé i neznámé hodnoty.

Další funkcí systému je dedukce. Ta slouží k definování neznámých entit. Opět se zde využívá vytvořených vazeb mezi entitami a výsledek dedukce je zobrazen ve formě seznamu. Ten obsahuje entity (známé i neznámé), které by mohli být pravděpodobným výsledkem neznámé entity.

Data jsou do systému nahrávána ze souborů, ve kterých jsou uložené. Formát souboru je pojat jednoduchou a snadno pochopitelnou formou. To dává uživateli možnost do souborů zapisovat a tím je přímo editovat nebo dokonce i tvořit.

Na základě náhledu a experimentu z praxe vyplynulo několik užitečných návrhů pro rozšíření funkcionality systému. Jedním z nich je přetvoření neznámé entity na známou, například po úspěšné dedukci systému a jednoznačném určení neznámé entity. Další užitečnou úpravou je přidání více entit jedné kategorie do filtrace, čímž by se zvýšila efektivita a účinnost filtrace.

# Conclusions

The aim of the work was to develop an information system with unknown values. Initially, the system was aimed at small companies, but in the course of its development it has become a very universal tool that can be used for a wide range of focuses and areas of interest. In designing the system, the main focus was on the versatility and simplicity of the system.

The central aspect of the system is working with unknown value. Various entities, categories and relationships can be created in the system, and any of the entities can be an unknown value.

The created system is able to handle the unknown value and evaluate queries to the system, mainly in the form of filtering. The results of the filtering are processed based on the relationships between entities created in the system by the user or the properties of the entities. The result of the query is two outputs, one with the necessity of occurrence, the other with the possibility of occurrence. Both types of outputs can contain known and unknown values.

Another function of the system is deduction. This is used to define unknown entities. It uses the established relationships between entities and the result of the deduction is displayed in the form of a list. This contains the entities (known and unknown) that could be the probable result of the unknown entity.

Data is loaded into the system from the files in which it is stored. The file format is conceived in a simple and easy to understand form. This gives the user the option to write to the files and thus directly edit or even create them.

Based on preview and experimentation from practice, several useful suggestions for extending the functionality of the system have emerged. One of them is the transformation of an unknown entity into a known one, for example after successful deduction of the system and unambiguous identification of the unknown entity. Another useful modification is to add multiple entities of the same category to the filtering, which would increase the efficiency and effectiveness of the filtering.

# A Obsah přiloženého CD/DVD

Na samotném konci textu práce je uveden stručný popis obsahu přiloženého CD/DVD, tj. jeho závazné adresářové struktury, důležitých souborů apod.

**bin/**
: Obsahuje instalátor SETUP.MSI a adresář Application, jehož součástí je program SystemUnknownValue, spustitelný přímo z CD/DVD. Adresář obsahuje i všechny runtime knihovny a další soubory potřebné pro bezproblémový běh programu z CD/DVD. Systém byl vyvíjen pro operační systém Windows.

**doc/**
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

**src/**
: Kompletní zdrojové texty programu SystenUnknownValue se všemi potřebnými zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření spustitelných verzí programu.

**readme.txt**
: Instrukce pro instalaci a spuštění programu SystemUnknownValue, včetně všech požadavků pro jeho bezproblémový provoz.

Navíc CD/DVD obsahuje:

**data/**
: Ukázková a testovací data s názvem *Nemocnice* použitá v práci a pro potřeby testování práce při tvorbě posudků a obhajoby práce.

# Literatura


[1] Guarino N., 1995, Formal Ontology, Conceptual Analysis and Knowledge Representation, International Journal of Human-Computer Studies, 43, pp. 625–640.

[2] Smith B., 2003, Ontology. In Luciano Floridi (ed.), Blackwell Guide to the Philosophy of Computing and Information. Oxford: Blackwell, pp. 155-166.

[3] Guarino N. and Welty C., 2000, A Formal Ontology of Properties, in R. Dieng and O. Corby (eds.), Knowledge Engineering and Knowledge Management: Methods, Models and Tools. 12th International Conference, pp. 97–112.

[4] Jardine D.A., 1976, The ANSI/SPARC DBMS model. Proceedings of the second SHARE Working Conference on Data Base Management Systems, Montreal, Canada, April 26-30.

[5] Abrial J-R., 1974, Data semantics. Klimbie and Koffeman (eds.) Data Management Systems, North Holland.

[6] Chen P., 1976, The entity-relationship model: Towards a unified view of data. ACM Transactions on Database Systems 1(1).

[7] Guarino, N. (ed.), 1998, Formal Ontology in Information Systems, Amsterdam, IOS Press, pp. 3-15.

[8] Ciza T. 2018, Ontology in Information Science, pp. 5.

[9] I. Sommerville, 2001, Software engineering (Addison-Wesley, Harlow, UK

[10] Hallberg N., Ontology for Systems Development, International Journal of Software Engineering and Knowledge Engineering 24(03), pp. 329-345.

[11] Hoppenbrouwers, J., van der Vos, B., and Hoppenbrouwers, S., 1996, NL Structures and ConceptualModelling: the KISS case. In R. P. van de Riet, J. F. M. Burg and A. J. van der Vos (eds.), Applica-tions of Natural Language to Information Systems. IOS Press.

[12] V. Kabilan, 2007, Ontology for Information Systems (O4IS) Design Methodology, A Dissertation submitted to The Royal Institute of Technology.

[13] Lamber L., 2006, Modal logic in computer science. Thesis. Rochester Institute of Technology.

[14] Philipse M., 2021, Distributed Knowledge Proofs in the Coq Proof Assistant, Bachelor Thesis, Radboud University, Computing Science

[15] Kripke S., 1963, A semantical analysis of modal logic I: normal modal propositional calculi. In Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik, 9, pp. 67-96.

[16] Priss U., 2006, Formal Concept Analysis in Information Science, vol. 40, pp. 521-543.

[17] Burmeister, P., Holzer, R. (2000). On the Treatment of Incomplete Knowledge in Formal Concept Analysis.

[18] Schickel-Zuber, V., Faltings, B., 2006, Overcoming Incomplete User Models in Recommendation Systems Via an Ontology, Lecture Notes in Computer Science, vol 4198. Springer, Berlin, Heidelberg

[19] Li, Xiangjun et al., 2010, Ontology Building from Incomplete Information System Based on Granular Computing. 2010 IEEE International Conference on Granular Computing, pp. 292-296.

[20] KOSEK, Jiří. *XML pro každého.* Praha: Grada Publishing, 2000, 164 s. ISBN 8071698601.

[21] Jazyk XAML. Největší český web zaměřený na .NET framework [online]. Copyright © 2019 [cit. 30.06.2019]. Dostupné z: https://www.dotnetportal.cz/clanek/198/Jazyk-XAML

[22] Úvod do Windows Presentation Foundation (WPF). Největší český web zaměřený na .NET framework [online]. Copyright © 2019 [cit. 30.06.2019]. Dostupné z: https://www.dotnetportal.cz/clanek/196/Uvod-do-WPF

[23] Repici, J., HOW-TO: The Comma Separated Value (CSV) File Format, 2004. Dostupné z: http://www.creativyst.com/Doc/Articles/CSV/CSV01.htm