ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Modelovací nástroj pro ER konceptuální návrh databází

Petr Stejskal

Květen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Stejskal** Jméno: **Petr** Osobní číslo: **466225**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Modelovací nástroj pro ER konceptuální návrh databází

Název bakalářské práce anglicky:

Modeling Tool for ER Conceptual Design of Databases

Pokyny pro vypracování:

Databázové systémy jsou nepochybně jednou z neodmyslitelných oblastí informatiky. Přestože se dnes pozornost věnuje především moderním NoSQL databázovým řešením, relační databázové systémy stále mají své dominantní postavení. S intenzivnějším využíváním i jiných logických modelů dat než jen tradičního relačního však narůstá potřeba efektivního modelování databázových schémat na konceptuální, tedy platformě nezávislé vrstvě. Za tímto účelem lze použít například dobře známé jazyky ER nebo UML. Existující nástroje vycházející z těchto jazyků však modelování na konceptuální úrovni obvykle neumožňují.
Prvním krokem této bakalářské práce je seznámení se s jazykem ER, jeho jednotlivými notacemi a nabízenými konstrukty. Dále budou popsány a vzájemně porovnány existující nástroje umožňující modelování relačních databází. Na základě provedené analýzy, identifikovaných výhod a nevýhod těchto nástrojů a formulovaných požadavků bude navrženo nové řešení. To bude implementováno a experimentálně otestováno.

Seznam doporučené literatury:

POKORNÝ, J.; VALENTA, M.: Databázové systémy. Nakladatelství ČVUT, Praha, 2013. ISBN: 978-80-01-05212-9.
SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S.: Database System Concepts. 6th Edition. McGraw-Hill Professional, New York, 2011. ISBN 978-0-07-352332-3.
CHURCHER, C.: Beginning Database Design. Apress, 2007. ISBN 978-1-59059-769-9. DOI 10.1007/978-1-4302-0366-7.
BATINI, C. et al.: Conceptual Database Design: An Entity-Relationship Approach. Benjamin/Cummings Redwood City, CA, 1992. ISBN 978-0-8053-0244-8.
CHEN, P.: Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned. Software Pioneers, Springer, Berlin, Heidelberg, 2002.
TEOREY, T. J.; YANG, D.; FRY, J. P.: A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. ACM Computing Surveys (CSUR), 1986, 18.2: 197-222.
Creately.com. <https://creately.com/>.
Draw.io. <https://www.draw.io/>.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Martin Svoboda, Ph.D., MFF

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **26.01.2019** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **20.09.2020**

RNDr. Martin Svoboda, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studenta

Poděkování / Prohlášení

Rád bych poděkoval vedoucímu práce RNDr. Martinu Svobodovi, Ph.D., za spolupráci a veškeré užitečné rady a podněty poskytnuté během tvorby této práce.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 22. května 2020

.....

Abstrakt / Abstract

Tato práce se zabývá konceptuálním modelováním pomocí jazyka ER. Práce popisuje vyjadřovací prostředky jazyka ER v podobě dostupných konstruktů. Poskytuje ucelené a přehledné srovnání významných ER notací od různých autorů a uvádí i srovnání jejich vyjadřovací schopnosti. Součástí práce je vypracován přehled existujících nástrojů pro ER modelování. Druhá část práce se věnujeme implementaci vlastního řešení v podobě webové aplikace. Řešení využívá již existující knihovnu poskytující podporu pro vytváření editorů diagramů v rámci klientských aplikací. Stěžejním bodem implementace je nastudování cizí knihovny a s využitím této knihovny vytvoření editoru umožňujícího vytváření ER diagramů v notaci, která je až na drobnou grafickou změnu založena na notaci od autorů Atzeni, Ceri, Paraboschi a Torlone.

Klíčová slova: Konceptuální modelování, databázové systémy, ER

This thesis deals with the topic of conceptual modeling using the ER language. The work describes the expressive power of ER in terms of the available constructs. It provides a comprehensive and structured comparison of important ER notations proposed by various authors, as well as it also compares their expressive power. As one of the thesis parts, an overview of the existing ER modeling tools is provided. The second part of the thesis describes the implementation of our proposed solution in a form of a web application. The approach uses an existing library that provides support for creating diagram editors within client-side applications. The keypoint of the implementation was to study this third party library and using it to create an editor allowing us to create ER diagrams in the notation based on the notation by Atzeni, Ceri, Paraboschi, and Torlone, except for certain minor visual modifications.

Key words: Conceptual modeling, database systems, ER

Obsah /

Kapitola 1
Úvod

Modelování datové vrstvy je jednou ze základních fází procesu návrhu aplikací. O ukládání dat se starají databázové systémy, kde úkolem vývojáře je navrhnout schéma dané databáze. Modelování datové vrstvy je možné rozdělit podle stupně abstrakce. Jedním z nich je konceptuální modelování. Ačkoli relační databázové systémy mají v současnosti stále dominantní postavení, navzdory tomu do světla pozornosti pronikají nové koncepty ukládání dat, tzv. NoSQL databázové řešení. Hlavní silou konceptuálního modelování je možnost strukturovat data nezávisle na vybraném technologickém řešení. Mezi známé prostředky konceptuálního modelování patří jazyky ER a UML. UML v dnešní době dominuje v oblasti návrhu datové vrstvy, ačkoli jeho pohled je zatížen relačním modelem. V porovnání jazyk ER je méně rozšířeným prostředkem, nicméně jeho pohled není zatížen přístupem relačních databází. ER je navržen pro modelování objektů a skutečností reálného světa, a proto se hodí pro návrh konceptuální vrstvy.

Přestože výhody plynoucí z přístupu jazyka ER převažují, do dnešního dne neexistuje jednotný způsob zápisu. Tento fakt dává prostor pro vznik různých způsobů zápisu, které označujeme jako notace. Nejednotný pohled v oblasti ER pravděpodobně brání jeho rozvoji a ustálení jeho používání. Své zastání má však v akademickém prostředí. Je vhodným prostředkem pro výuku čistého nezatíženého pohledu na konceptuální modelování.

Naneštěstí situace ohledně dostupných nástrojů pro konceptuální návrh v jazyce ER není pozitivní. V dnešní době stále neexistují vhodné nástroje pro zachycení ER schématu prostřednictvím ER diagramu. Reálným důsledkem situace je vytváření diagramů pomocí tužky a papíru, vykreslování diagramů pomocí malovacích nástrojů poskytující práci s geometrickými tvary nebo se musíme spokojit s hrstkou řešení, které nám neumožní namodelovat to, co bychom potřebovali.

Cílem této práce je poskytnout ucelený pohled na ER a vytvořit vhodný nástroj pro návrh ER diagramů v rámci zvolené notace, která má dobrou vyjadřovací schopnost, přehlednost a výstižnost.

Tato práce čtenáře v úvodu seznámí s konceptuálním modelováním, ilustruje klíčové momenty historického vývoje a vlivy různých pohledů na problematiku v této oblasti a představí základní koncept entitně vztahového modelování. Tuto myšlenku následně rozvedeme do podrobného popisu prostředků, které nám jazyk ER poskytuje. U každého konstruktu se zastavíme a provedeme ucelené srovnání způsobu značení podle vybraných existujících notací. Také zhodnotíme, jakou vyjadřovací schopnost vzhledem k představeným konstruktům nám vybrané notace poskytují.

Na základě získaných zkušeností provedeme rozbor dostupných nástrojů pro záznam ER diagramů a vyhodnotíme jejich vhodnost pro modelování. Vzhledem k provedenému rozboru vytvoříme specifikaci vlastního řešení. Uvedeme požadavky na nové řešení, které rozpracujeme hlouběji pomocí případů užití, a také uvedeme, která data je nezbytné ukládat pro zachycení ER schématu prostřednictvím ER diagramu.

V rámci implementace popíšeme zvolené technologie řešení pro naši aplikaci a poskytneme základní popis jednoho existujícího frameworku pro tvorbu editorů diagramů.

Stěžejním bodem implementace bude integrace existujícího frameworku a funkcionality pro modelování ER digramů do výsledného řešení. Popíšeme, jak jsme jednotlivé konstrukty vizuálně navrhli, jakým způsobem jsou reprezentovány v grafovém konceptu knihovny a jaká úskalí jsme museli u některých prvků vyřešit.

V závěrečné části práce čtenáře seznámíme s ovládáním dostupných funkcionalit vytvořeného editoru a popíšeme, jakým způsobem je možné modelovat jednotlivé konstrukty v rámci tohoto řešení.

Jako součást práce popíšeme, jakým způsobem byla aplikace otestována a okrajově vysvětlíme některé pojmy spojené s oblastí testování a návrhu testů.

Na úplný závěr provedeme zhodnocení dosažených výsledků, přínos našeho řešení a zkusíme uvést některé myšlenky, jakým směrem by se aplikace mohla v budoucnu vydat.

Kapitola 2
Konceptuální modelování

Následující kapitola se bude zabývat problematikou konceptuálního modelování. V první části provedeme krátkou ilustraci doby, která vedla ke vzniku softwarového inženýrství, poté se podíváme na datové modelování jako disciplínu softwarového inženýrství. Po uvedení do problematiky se opět vrátíme do historie a odhalíme vznik entitně-vztahového (zkratkou ER) modelování. Závěrem si uvedeme, co je ER koncept a na jakém principu je založen.

2.1 Softwarové inženýrství a modelování

Pojem modelování ve smyslu vytváření modelů pro popis informačních systémů a programů se začíná objevovat spolu s disciplínou softwarového inženýrství. Počátky tohoto oboru sahají do šedesátých let minulého století. Toto období z pohledu pozdějšího vzniku softwarového inženýrství můžeme označit jako pionýrské. Programování, jak ho známe dnes, je v samotných počátcích. Pro představu na počátku šedesátých let začíná být používán jazyk COBOL a LISP (specifikován 1958), je popsán třídící algoritmus Quicksort, v roce 1963 vzniká ASCII tabulka. Začínají se formulovat první rané základy objektově orientovaného programování.

Výzkumné organizace, ale i obchodní společnosti si začínaly uvědomovat rostoucí význam počítačových systémů. Ty měly stále větší dopad na společnost v mnoha oblastech. Aby vývoj nových a stále větších systémů nebyl chaotický, při jeho vzniku měly napomáhat techniky, které měly pokrývat proces návrhu, vývoje a následné údržby systémů.

Právě na konferencích NATO SOFTWARE ENGINEERING CONFERENCE v letech 1968 a 1969 se ustálilo označení softwarové inženýrství, které pokrývá obory informatiky, inženýrství a managementu. Výstupem konferencí byly dokumenty, které popisovaly, jak by software měl být vyvíjen. Postupným vývojem byla terminologie kolem softwarového inženýrství standardizována v devadesátých letech[1].

2.2 Datové modelování

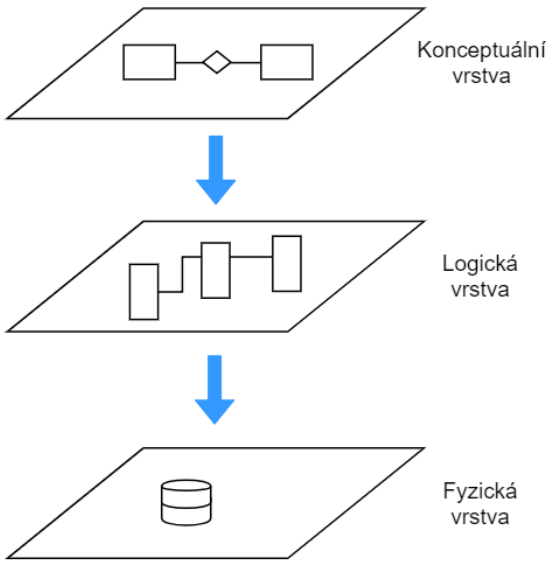
Jednou z disciplín softwarového inženýrství je i datové modelování. V tomto procesu definujeme a analyzujeme požadavky na strukturu dat v informačním systému. Výstupem je schéma, které popisuje strukturu a formát dat, vzájemné uspořádání datových prvků a vztahy mezi těmito prvky. Cílem tohoto modelování je zachytit tu část reálného světa, o které chceme uchovávat data v našem informačním systému.

Modelování v rámci vývoje software provádíme, abychom nejdříve promysleli, co náš systém bude obsahovat, a poté se soustředili pouze na implementaci na základě navrženého schématu. Dalším důvodem je, že tím zachycujeme stav našeho systému jako dokumentaci a v případě změn víme, co upravujeme a na co navazujeme. Posledním benefitem je, že pokud máme návrh hotový, můžeme implementační práci rozdělit mezi

více lidí či týmů. Ti mají pomocí schémat v podobě diagramů sdíleny základní informace, které potřebují znát mezi sebou a mohli tak pracovat relativně nezávisle. Uvádíme relativně, protože diagramy by měly být vždy jasně pochopitelné, ale někdy může docházet k různým výkladům stejné věci. Poté se jednotlivé osoby doptávají, zda specifikaci rozumí správně.

Proces datového modelování můžeme rozdělit do tří úrovní abstrakce. Tento způsob abstrakce dělí proces tvorby schéma datové vrstvy informačního systému tak, aby v každé fázi vývoje byla věnována pozornost jen určitému aspektu. Těmito aspekty jsou obsah, technologie a implementační specifika.

Můžeme si všimnout, že tyto aspekty tvoří přirozenou posloupnost, finální schéma tvoříme způsobem od shora dolů. Jedná se o hojně využívanou techniku, kdy postupujeme od nejobecnějšího pohledu na věc a postupně v každém kroku každou část rozpracováváme do hlubšího detailu. Zde se konkrétně v první fázi zabýváme tím, co se v systému bude nacházet. V informačních systémech jde především o převedení reálného světa do našeho schéma. Následně sestoupíme o úroveň níže a naše schéma začneme obohacovat o specifika, která plynou z námi zvolené technologie. Naše schéma přepracujeme, tak aby mohl být reprezentován zvolenou technologií. V poslední úrovni se již staráme o to, jak provedeme implementaci v rámci prostředků námi zvolené technologie. Následující obrázek 2.1 ilustruje existenci a souvislost vrstev abstrakce a proces prostupného vývoje.



Obrázek 2.1. Princip 3 vrstev abstrakce v procesu modelování databází

2.2.1 Konceptuální vrstva

Na této úrovni dochází k vymezení obsahu systému. Určujeme, jaké informace náš systém bude uchovávat většinou na základě požadavků týkajících se funkcionality. Identifikujeme, jaké prvky z reálného světa v datové vrstvě chceme popsat a jaké vztahy mezi těmito prvky panují. Při modelování reality se díky abstrakci na této vrstvě soustředíme pouze na obsahovou stránku věci. Nesmíme se ohlížet na technologické a implementační detaily, tvořené schéma jimi nesmí být zatíženo. Toto přináší zjednodušení někdy složité a obsáhlé reality.

Je podstatné na této úrovni uvést veškeré informace, které jsou v systému potřeba, další úrovně modelování již systém z hlediska vystupujících informací nerozšiřují, jen upravují schéma z technologického či implementačního pohledu.

2.2.2 Logická vrstva

Navazuje na předchozí úroveň modelování. Někdy se nazývá též technologická vrstva. Na této vrstvě začínáme zohledňovat, jak budeme data strukturovat. Je zřejmé, že budou data uchovávána jinak v relační databázi, objektově orientované či grafové... Schéma, které zde vzniká, je stále platformně nezávislé, ale už reflektuje, jak vymezený obsah z předešlého schéma na konceptuální vrstvě je organizován vzhledem ke zvolené struktuře dat.

Z toho můžeme říci, že jedno schéma na konceptuální vrstvě můžeme zpracovat do několika podob na logické vrstvě. Tyto popisy nebudou navzájem stejné, protože každý bude zatížen jinou strukturou dle volené technologie, ale jejich popis na konceptuální vrstvě je jednotný. Tedy systém bude schopen ukládat shodné informace.

2.2.3 Fyzická vrstva

Můžeme se setkat i s označením implementační vrstva. Jedná se o nejnižší úroveň abstrakce. V této fázi řešíme poslední věc, zohledňujeme námi zvolenou implementaci – konkrétní databázový systém od určitého vydavatele nebo implementační specifika programovacího jazyka. Tato vrstva určuje, čím je řešení realizováno.

2.3 Historický vývoj databázových systémů a konceptuálního modelování

V této práci se budeme nadále zabývat už pouze konceptuální vrstvou, kterou jsme si představili v rámci principu tří vrstev abstrakce. Konceptuální modelování je jedna z důležitých technik softwarového inženýrství, která se uplatňuje při návrhu a specifikaci systému.

Od poloviny šedesátých let docházelo k uvědomění, že mnoho projektů se zpožďovalo nebo překračovalo finanční rozpočty. Pro představu, jaké projekty v tomto období probíhaly, můžeme zmínit americkou agenturu NASA, která vysílala rakety k měsíci v rámci projektu Apollo. Přirozeně si uvědomujeme, že u projektů tohoto charakteru nesmělo docházet k chybám. Spolehlivost těchto systémů musela být ta nejvyšší. Důsledkem těchto uvědomění mezi akademiky pak byla již zmíněná konference o softwarovém inženýrství na konci šedesátých let.

Zároveň s rostoucí rolí návrhu a projektového řízení docházelo k vývoji i v oblasti databázových systémů. Ty staly na počátku zrodu, většina organizací spíše používala souborové systémy než databázové systémy pro ukládání dat.

V roce 1959 se konala konference CODASYL za účasti výrobců, uživatelů počítačů a amerického ministerstva obrany, jejíž výsledkem byl požadavek na univerzální databázový jazyk. Dřívější široké používání strojového kódu pro procesor se ukázalo nevhodné pro databázové úlohy. Nový jazyk měl umožňovat sestavení programů v minimálním čase s minimálním programovacím úsilím. Toho mělo být docíleno požadavkem, že jazyk se měl více zakládat na jazyce podobném angličtině místo strojovému kódu. Také mělo být možné provádět úplnou dokumentaci programů. Posledním požadavkem byla možnost programy převádět na novější typy počítačů. To bylo podstatné, protože se v této době programovalo na specifický hardware. O rok později byla publikována první verze jazyka COBOL. Jazyk se stal používaným a prodělal si další vývoj.

V následujícím desetiletí se vývoj databázových systémů upíral do různých směrů. Na trhu vznikalo několik řešení databázových systému, kde většinou každý používal svůj vlastní model a způsob modelování.

Kolem vývoje databázových systémů se pohyboval i Charles Bachman, uznávaná osoba v oblasti databází a softwarového inženýrství, který představil vlastní komerční databázový systém IDS (Integred data system). Za svou práci získal v roce 1973 Turingovu cenu. Kromě toho pracoval i na tom, jak modelovat strukturu databází. Na konci šedesátých let publikoval článek *Data Structure Diagrams*[2] pojednávající o konceptu datových entit a uvedl jeden z prvních diagramů datových struktur (Data structure diagram), který začal být obecně známý jako Bachmanův diagram. Ten se opíral o koncept síťového modelování. Je považován za předchůdce či ranou formu entitně relačních (ER) diagramů.

Dalším směrem vývoje byly hierarchické databáze. Představitelem byl databázový systém IMS (*Information Management System*) vyvinutý firmou IBM, který se používal v programu letů na měsíc Apollo. Pro návrh databází se používal hierarchický model.

V akademickém světě získal obrovský zájem relační model představený v roce 1970. Tento fakt podporuje, že na začátku sedmdesátých let akademický svět věnoval svou pozornost především tomuto modelu místo ostatním. Jeden z hlavních důvodů byl, že mnoho akademiků mělo obtíže pochopit dlouhé a nezáživné manuály komerčních databázových systémů. Článek popisující relační modelování[3] byl naopak sepsán stručně a odborně. Jeho autor Edgar Frank Codd tím odstartoval v té době revoluci a publikace položila základ pro relační databáze a i pro pozdější vznik dnes celosvětově známého a používaného dotazovací jazyka SQL.

2.4 Představení konceptu ER modelování

Roztříštěnost řešení se stala tématem počátku sedmdesátých let. Obchodníci se software volali po nutném sjednocení různých souborových a databázových formátů na trhu a vnesení více sémantiky do datových modelů. Organizace zase zdůrazňovaly potřebu jednotné metodiky pro návrh databázi v dostupných databázových systémech na komerčním trhu a také začlenění sémantiky. Oba světy firem a prodejců směřovaly stejným směrem k požadavku sjednocení.

Do této situace přichází Peter Pin-Shan Chen, původem z Taiwanu, kde vystudoval elektrotechniku a získal pozvání na postgraduální studium na Harwardu v oblasti počítačových věd. Vědomosti, které tam získal v oblasti elektrotechnicky, počítačových věd a aplikované matematiky byly zásadní pro jeho budoucí práci, uvádí ve svém retrospektivním článku o období vzniku ER modelování[4].

Po získání titulu Ph.D. v roce 1973 začal pracovat pro HoneyWell Information System, kde se podílel na vývoji nové generace počítačových systémů, které měli být založeny na architektuře distribuovaných systémů. Tým, jehož byl součástí, se skládal ze známých počítačových expertů včetně Charlese Bachmana. Pro zajímavost lze uvést, že většina členů týmu byla alespoň o dvacet let starší než Peter Chen. Již během tohoto angažmá si začínal pro sebe formulovat koncept ER modelování. O jeho myšlenkách se prozatím nikomu nezmínil. O rok později byl projekt přerušen a Chen zanedlouho začal pracovat jako asistent profesora na škole managementu Sloan pod MIT. Zde začal formulovat svoje myšlenky do článku. Později jako profesor na škole vyučující management měl příležitosti se setkávat se zástupci organizací a tedy i se základnou uživatelů. Jeho práce tak byla ovlivněna současnou potřebou jednotné metodiky pro strukturování souborů a návrh databází. Svou výslednou práci pak publikoval v roce 1976[5] jako článek v časopise *ACM Transactions on Database Systems*.

Vydání článku způsobilo velké množství reakcí. Pozitivní i negativní. Mezi největší kritiky patřil Edgar Frank Codd, autor relačního modelu. V návaznosti na vydání článků sepsal kritiku, ve které vyjádřil nesouhlas s novým konceptem. Ironicky však o několik let později, podle Petera Chena[4], Codd zahrnul některé rysy konceptu ER ve svém novém relačním modelu nazvaném RM/T.

V době publikace probíhala válka mezi zastánci relační a síťového modelu. Uprostřed toho mladý Chen publikoval svou práci s názvem obsahující výraz „jednotný datový model“. Toto lidově řečeno přililo olej do ohně. Dostal několik doporučení v dobrém úmyslu, ať se radši zabývá prací a publikací článku týkajícího se normálních forem jako většina ostatních té doby. Nicméně Chen věřil, že může přispět v oblasti konceptuálního návrhu. Z pohledu současnosti se mu to povedlo, neboť první článek o ER patří mezi hojně citované články.

Z počátku počet příznivců ER modelu nebyl velký, postupem času počet rostl. Mimo akademický svět i průmyslové firmy či vládní organizace začaly vidět potenciál v ER modelování. V roce 1979 byla uskutečněna první konference týkající se ER. Předpokládala se účast asi 50 účastníků, nakonec se sešlo okolo 250 až 300 lidí[4]. Konference o ER se odehrává každoročně až dodnes. Jedná se o uznávanou konferenci v oblasti konceptuálního modelování, jsou zde prezentovány novinky ve výzkumu a aplikaci.

2.5 Následující vlivy a vývoj ER

V následujících osmdesátých a devadesátých letech probíhala popularizace ER. Peter Chen přivedl na svět inovativní nápad, jak strukturovat data. Současně s rozšiřováním osobních počítačů, různé firmy pracovaly na vývoji a zavádění CASE nástrojů (Computer Aided Software Engineering, volně přeloženo počítačem podporované softwarové inženýrství). Nástroje měly napomáhat při životním cyklu programu – návrh, vývoj, údržba. Jedním z prvotních myšlenek byla, že programy nebudou programovat programátoři, ale jejich návrh by probíhal v těchto nástrojích modelováním pomocí diagramů a výsledný kód by se pak vygeneroval. Tato myšlenka se přímo neuchytila, ale CASE nástroje nyní hojně vypomáhají jako podpůrný prostředek ve vývojovém cyklu programů.

Samotný nápad generování kódu se nevytratil, například v programu Enterprise Architecture, který je jedním z používaných CASE nástrojů, můžeme vytvořit model tříd v UML[6] a pak vygenerovat kód výsledných tříd třeba v jazyce JAVA. Funkcionalita tříd se pak musí dopsat. Tento princip se dá použít i při návrhu grafického rozhraní programu GUI. Okno programu si navrhneme v rámci editoru a na pozadí se generuje výsledný kód. Podobných příkladů bychom mohli uvést mnohem více.

Některé CASE programy převzaly myšlenky ER konceptu a začlenily je do svých nástrojů pro konceptuální modelování. Uvádíme myšlenky, protože původní ER si procházel úpravy a inovacemi ze stran různých autorů. Tyto úpravy neprobíhaly nejen pro potřeby programů, ale probíhaly v rámci potřeb různých projektů, specifických systémů. Pro některé skupiny lidí byl původní ER nedostatečný a pro doplnění „funkcionality“ si do notace začaly přidávat své vlastní konstrukty či upravovaly formu zápisu pro lepší výstižnost diagramů.

V souvislosti s tímto děním začínáme u ER rozlišovat různé **notace**. Za původní notaci považujeme Chenovu. Také se můžeme sekat s pojmy vylepšený či rozšířený ER, v angličtině *Enhanced* nebo *Extended* ER, shodně označováno zkratkou EER. Nakonec se hodí uvést, ačkoliv je ER modelování v dnešní době využívané, není standardizováno napříč státy nějakou ISO normou. Mimo to, ale neexistuje ani jednotně formulovaná

představa o ER mezi širší skupinou uživatelů. Možná i z tohoto důvodu byl historicky dán prostor, a stále je, pro různé individuální úpravy.

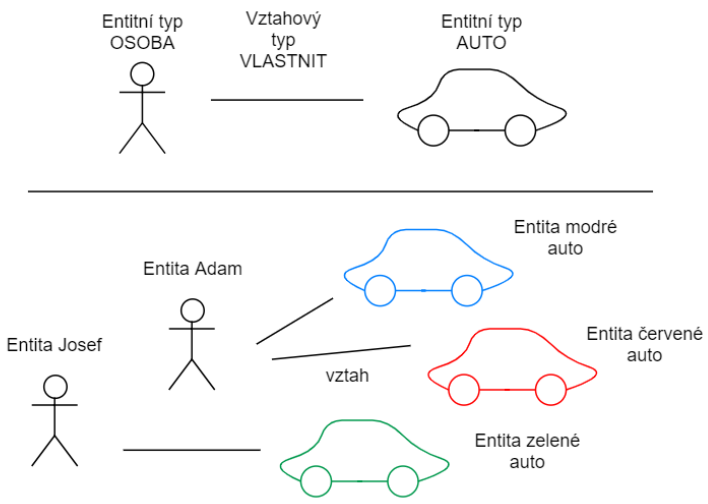
Entitně relační modelování se stalo široce známým nástrojem pro návrh konceptuální vrstvy databází. Dalším hojně používaným, dokonce i známějším prostředkem je UML (*Unified Modeling Language*, volně přeloženo jednotný modelovací jazyk). Oproti ER má UML mnohem širší možnost použití, jedná se o grafický jazyk používaný pro tvorbu různých typů diagramů v oblasti specifikace, návrhu a dokumentace systémů. Nutno podotknout, že UML svůj vývoj dovedlo do podoby standardu[6]. Nicméně na okraj můžeme uvést, že podle Petera Chena má UML své kořeny právě v ER modelu.

2.6 Co je ER koncept

Vytvoření konceptu entitně vztahového modelu bylo důležitým krokem v oblasti modelování informačních systémů a softwarového inženýrství. Myšlenka měla vliv na později vznikající metodiky a přinesla nový směr pohledu na věc. Nepochybně ovlivnila dnešní podobu procesu návrhu datové vrstvy.

V následující části pro jednoduchou představu zkusíme nastínit, s čím myšlenka ER modelu přichází. Pro zjednodušení nebudeme zacházet do hlubokého detailu, všechny prvky ER a způsob značení si podrobně popíšeme v následující kapitole.

Jak již název napovídá, model nebo z něho vycházející schéma je založeno na tzv. entitách a vztazích mezi nimi. V reálném světě jsme schopni jednoznačně identifikovat samostatné objekty. Ty nazýváme entitami, některé množiny entit můžeme popsat podle společné předlohy, kterou nazveme entitním typem. Jako jednoduchý příklad můžeme uvést *osobu* jako entitní typ. Pokud bychom se omezili, že každé jméno může mít pouze jeden člověk na světě, potom jednotlivé osoby jsme schopni unikátně rozeznat podle jména. Pak Josef a Adam jsou dvě rozlišitelné entity. Do naší představy ještě přidáme další entitní typ *auto*, které bude mít jedinou vlastnost barvu a opět každou barvu bude moci nést pouze jedno auto. Řekněme, že v naší představě může osoba vlastnit auto. Tuto skutečnost vyjádříme pomocí vztahového typu *vlastnit*. Díky tomu nám mohou vznikat jednotlivé vztahy mezi osobami a auty. Naši představu vizualizuje následující obrázek 2.2.



Obrázek 2.2. Diagram znázorňující myšlenku entit a vztahů. Ilustrace vlastnictví aut.

Konkrétně znázorňuje situaci, že Adam vlastní červené a modré auto. Josef vlastní zelené. Uvedený koncept využívá přirozeného pohledu na svět a identifikace entit a vztahů. Nicméně tento pohled je založen na Teorii množin a relacích. V době představení ER existovaly tři hlavní modely. Relační založen na matematických relacích, který je schopen dobře popisovat organizaci dat, ale může ztrácet některé sémantické informace o reálném světě. Vazby mezi jednotlivými tabulkami nejsou explicitně vidět. Síťový model pohlíží na reálný svět v rámci entit a vztahů, ale je obtížné dosáhnout vysokého stupně nezávislosti dat. Poslední množinový model je založen na Teorii množin. Diagram, ale nebyl výstižný v tom, co jednotlivá data popisují. Pokud se v diagramu objevila hodnota „5“, nebylo jasné, co číslo popisuje. Peter Chen „propojil“ tyto tři modely do jednoho, každý z nich lze pomocí ER vyjádřit. Ve svém článku[5] popsal, jak jednotlivé modely převést do jeho modelu ER. Proto tehdejší označení jednotný datový model nebylo mylné.

Abychom základ entitně relační konceptu popsali trochu odborněji a předvedli si, že stojí na silných matematických základech, uvedeme formální vyjádření entit a vztahů pomocí Teorie množin v následujícím obrázku 2.3.

Popis v množinové teorii		
Entita	e	**Vztahový typ vyjádřen jako matematický vztah**
Entitní typ (množina)	E; e ∈ E	**na entitním typu**
Hodnota	v	
Množina všech možných hodnot	V; v ∈ V	$R = \{r_1, r_2, \ldots, r_n\}$
Vztah	r	$r_i = [e_{i1}, e_{i2}, \ldots, e_{in}], e_{i1} \in E_1, e_{i2} \in E_2, \ldots e_{in} \in E_n$
Vztahový typ (množina)	R; r ∈ R	

Obrázek 2.3. Formální definice Entitně relačního konceptu [4]. Přeloženo z originálu.

Nyní tento formální popis propojíme s naší jednoduchou představou o vlastnictví aut osobami a vysvětlíme si, co přesně jednotlivé množiny znamenají.

Začneme s množinou E, tato množina nám představuje entitní typ. To je například naše OSOBA. Do množiny jednoho entitního typu nám spadají různé entity e, které jsou od sebe navzájem rozlišitelné, ale všechny je lze charakterizovat daným entitním typem. V příkladu entitami jsou osoby konkrétně označené jako *Josef* a *Adam*.

Množina všech možných hodnot V, je pojmenovaný typ informace, například KŘESTNÍ JMÉNO, PŘÍJMENÍ, VĚK... Poté hodnota v je unikátní hodnota v této množině, například Martin či Honza pro množinu hodnot KŘESTNÍ JMÉNO. Spolu s tímto zmíníme pojem atribut. Je to pojmenované zobrazení z entitního typu nebo vztahového typu do jedné či více množin hodnot. V případě zobrazení do jedné množiny, můžeme atribut pojmenovat JMÉNO a přiřadit ho k entitnímu typu OSOBA, zobrazení nám ukazuje pouze na množinu hodnot KŘESTNÍ JMÉNO. Příkladem zobrazení do více množin (konkrétněji myslíme do produktu kartézského součinu několika množin hodnot) je atribut CELÉ JMÉNO, které ukazuje do množiny hodnot KŘESTNÍ JMÉNO a PŘÍJMENÍ. K jednomu entitnímu typu může být přiřazeno více atributů. V našem příkladu entitní typ OSOBA obsahuje atribut JMÉNO, kde zobrazení ukazuje na množinu KŘESTNÍ JMÉNO, jehož obsahem jsou hodnoty JOSEF a ADAM.

Vztahový typ R je pojmenovaná množina jednotlivých vztahů r. Vztah r je pak n-tice jednotlivých entit daných entitních typů. Můžeme uvést poznámku, že nás nesmí překvapit případ, kdy n-tici tvoří jen entity, které naleží jednomu entitnímu typu. Příkladem u entitního typu OSOBA je vztah MANŽELSTVÍ, je to dvojice, kde vystupují dvě entity, které jsou shodným entitním typem OSOBA. Někdy se můžeme setkat

s označením tohoto vztahu jako rekurzivní. V naší ukázce nám vystupuje jediný vztah VLASTNIT, kde se jedná o dvojici mezi entitním typem OSOBA A AUTO.

Kapitola 3
Konstrukty a notace ER

V následující kapitole se blíže seznámíme s tím, jak vytvářet entitně vztahový diagram (ERD). Postupně budeme procházet jednotlivé konstrukty neboli prvky, které můžeme pro modelování využít. Napříč každým konstruktem vysvětlíme jeho význam a uvedeme možné použití. Kromě toho si ukážeme, jak se graficky značí v základní Chenově notaci a v dalších možných zápisech. Závěrem si provedeme shrnutí, jaké prvky obsahují jednotlivé notace.

V následujícím textu se budeme zabývat jen určitým výběrem notací. Představíme si notace, které respektují původní myšlenku ER a liší se zejména v grafickém vyjádření skutečnosti. Budou nás zajímat následující notace označované podle jejich autorů:

- a) Chen - originální z roku 1976 [5, 7–8]
- b) Teorey [9]
- c) Elmasri a Navathe [10]
- d) Korth a Silberschatz [11]
- e) McFadden a Hoffer [12]
- f) Batini, Ceri a Navathe [8]
- g) Atzeni, Ceri, Paraboschi a Torlone [13]

3.1 Entitní typ

Základním kamenem schématu je entitní typ. Je to každý rozlišitelný typ objektů, o kterém chceme uchovávat informace. Příkladem může být člověk, dům, místnost, zboží, dopis, atd. Jednotlivé instance tohoto typu objektu, které pak popisují konkrétní data, jsou entity. Ty se ve schématu neobjevují a jsou zastoupeny hromadně pouze entitním typem. Všechny námi sledované notace entitní typ shodně značí pomocí obdélníka. Uvnitř se nachází text pojmenovávající entitní typ. Jako název volíme obvykle podstatné jméno charakterizující objekt, jedinečné napříč diagramem. Způsob značení zachycuje obrázek 3.1.

3.2 Atribut

Atribut je další konstrukt pevně spjatý s existencí entitního typu. Je to vlastnost, kterou každá instance entitního typu nabývá. Toto je nutná podmínka, protože podle našeho pojetí modelování na konceptuální vrstvě neuvažujeme existenci **NULL** hodnoty. Hodnota informace může být pro každou instanci úplně odlišná, či pro vybrané nebo všechny stejná. Příkladem atributu pro entitní typ osoba může být jméno, příjmení, datum narození, bydliště, atd. Nesmíme zapomenout, že atribut může být přiřazen i vztahovému typu.

Rozlišujeme několik typů atributů, které nám říkají dodatečnou informaci o vlastnosti dat, které atribut představuje. Atribut může být – jednohodnotový, vícehodnotový, identifikující, složený, odvozený. Některé typy lze navzájem kombinovat.

Notace	Způsob spojení slov
a) Chen	pomlčka
b) Teorey	pomlčka
c) Elmasri a Navathe	podtržítko
d) Korth a Silberschatz	podtržítko
e) McFadden a Hoffer	podtržítko
f) Batini, Ceri a Navathe	bez mezer, první písmena velká
g) Atzeni, Ceri, Paraboschi a Torlone	bez mezer, první písmena velká

Tabulka 3.1. Tabulka srovnávající způsob psaní víceslovného názvu

3.2.1 Název atributu

Jako název atributu může být použito jedno či více slov. V případě víceslovného pojmenování se v jednotlivých notacích liší způsob zápisu. Srovnání poskytuje tabulka 3.1.

3.2.2 Jednohodnotový atribut

Jak již název vypovídá, atribut nabývá pouze jedné hodnoty. Je to nejběžnější typ atributu používaný v ER diagramech. Příkladem pro entitní typ osoba je atribut jméno. Může nabývat hodnot `Adam`, `Honza`, `Klára`, atd.

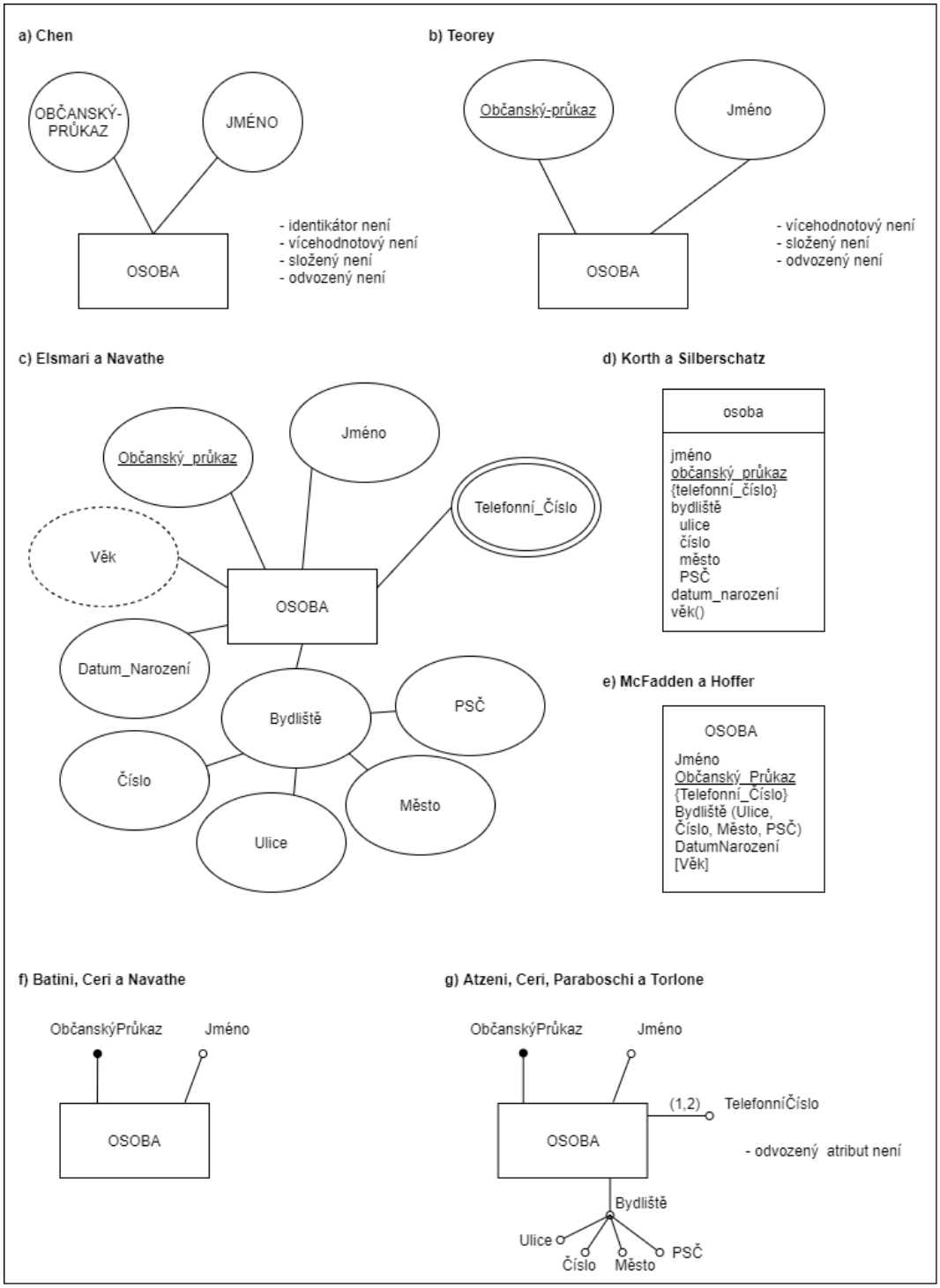
V následujícím přehledu můžeme porovnat způsob značení v jednotlivých notacích, příklad grafického vyjádření najdeme v obrázku 3.1:

- **a) Chen** – Kolečko s názvem atributu uvnitř. Atribut je spojen čárou s entitním nebo vztahovým typem ke kterému patří.
- **b) Teorey** – Elipsa s názvem atributu uvnitř. Atribut je spojen s entitním nebo vztahový typem rovnou čarou.
- **c) Elmasri a Navathe** – Značení shodné s předchozí notací b)
- **d) Korth a Silberschatz** – Entitní typ je rozdělen na dvě části. V horní se nachází název. Ve druhé spodní části se nachází atributy vypsané jako seznam pod sebou. Jednohodnotový atribut nemá speciální značení – pouze text. Jak značit atributy pro vztahový typ se zabývá podkapitola 3.3.3.
- **e) McFadden a Hoffer** – Atributy jsou umístěny uvnitř obdélníka znázorňující entitní typ přímo pod názvem jako seznam. Atribut je zastoupen pouze textem. O tom jak modelovat vztah s atributy se zabývá podkapitola 3.3.3.
- **f) Batini, Ceri a Navathe** – Malý kroužek spojený čarou s entitním typem. Název umístěný mimo kroužek někde poblíž.
- **g) Atzeni, Ceri, Paraboschi a Torlone** – Značení shodné s předchozí notací f).

3.2.3 Vícehodnotový atribut

Vícehodnotový atribut je atribut, který může nabývat více hodnot. Přesněji řečeno relace, kterou atribut představuje, je zobrazení na více hodnot ze stejné množiny hodnot. Příkladem u entitního typu osoba může být telefonní číslo. Pokud chceme evidovat o osobě méně či více jak jeden telefon nebo nějaké rozmezí, můžeme atribut telefonní číslo označit jako vícehodnotový.

Některé notace umožňují přesně specifikovat omezení na počet těchto hodnot. To se skládá ze dvou složek – účast parciální (značeno 0), nebo totální (značeno 1) a pak omezení na maximální počet hodnot (přesný počet nebo `N`). Můžeme vytvořit například `0..N`, `1..N`, `1..2`, nebo `0..1` – tím vytvoříme nepovinnou hodnotu.



Obrázek 3.1. Značení konstruktů v rámci sledovaných notací. Jako demonstrativní příklad je uveden entitní typ osoba, který obsahuje jednohodnotový atribut jméno, identifikující atribut občanský průkaz (myšleno jeho číslo), vícehodnotový atribut telefonní číslo s případným omezením totální účasti a horní hranice 2, tedy rozmezí 1-2. Jako složený atribut vystupuje bydliště skládající se z názvu ulice a čísla, města a poštovního směrovacího čísla (PSČ). Pro odvozený atribut byl zvolen věk, který je vypočítán na základě data narození.

V případě, že konstrukt neexistuje, není prvek modelován a je doplněna poznámka.

Další notace tento konstrukt neumožňují. Náhrada tohoto konstruktu, pokud chybí, je však možná. Z telefonního čísla vytvoříme entitní typ a mezi osobou a telefonním čísle vytvoříme vztahový typ.

V následujícím přehledu můžeme vidět, jak jednotlivé notace tento konstrukt značí, pokud jej umožňují:

- **a) Chen** – Značení neexistuje.
- **b) Teorey** – Značení neexistuje.
- **c) Elmasri a Navathe** – Dvojitá elipsa, název atributu uvnitř.
- **d) Korth a Silberschatz** – V seznamu atributů uveden ve složených závorkách.
- **e) McFadden a Hoffer** – V seznamu atributů uveden ve složených závorkách.
- **f) Batini, Ceri a Navathe** – V dostupných materiálech značení nedohledáno.
- **g) Atzeni, Ceri, Paraboschi a Torlone** – Malý kroužek s názvem atributu poblíž. U čáry spojující entitní typ s vícehodnotovým atributem je uvedeno v kulatých závorkách omezení. Dolní a horní mez je oddělena čárkou. Např. `(1, 2)`.

Ukázku grafického znázornění nalezneme v obrázku 3.1.

3.2.4 Složený atribut

Některé notace umožňují použití složeného atributu. Je to pojmenování pro určitou skupinu atributů. Příkladem osoba může mít složený atribut bydliště, který je složen z dalších atributů – město, ulice, číslo, poštovní směrovací číslo. Pokud tento konstrukt daná notace neumožňuje, s osobou můžeme atribut město a další spojit přímo bez použití skládání. Složené atributy lze do sebe vnořovat. V následujícím přehledu je popsán způsob značení. Grafická ukázka je obsažena v obrázku 3.1.

- **a) Chen** – Značení neexistuje.
- **b) Teorey** – Značení neexistuje.
- **c) Elmasri a Navathe** – Elipsa obsahující uvnitř název. Čarou připojen k entitnímu typu. Atributy skládající složený atribut jsou značeny podle jejich typu a jsou spojeny čarou připojeny k tomuto atributu.
- **d) Korth a Silberschatz** – Složený atribut se uvádí do seznamu atributů v entitě. Skládané atributy jsou uvedeny pod složeným atributem a v seznamu odsazeny.
- **e) McFadden a Hoffer** – Atribut se nachází v seznamu. Nejdříve je uveden jeho název a v kulaté závorce atributy, které ho tvoří.
- **f) Batini, Ceri a Navathe** – V dostupných materiálech značení nedohledáno.
- **g) Atzeni, Ceri, Paraboschi a Torlone** – Složený atribut představuje malý kroužek s názvem poblíž a spojen čarou k entitnímu typu. Jeho složky jsou připojeny čarou a značeny podle typu.

3.2.5 Identifikující atribut, identifikátor

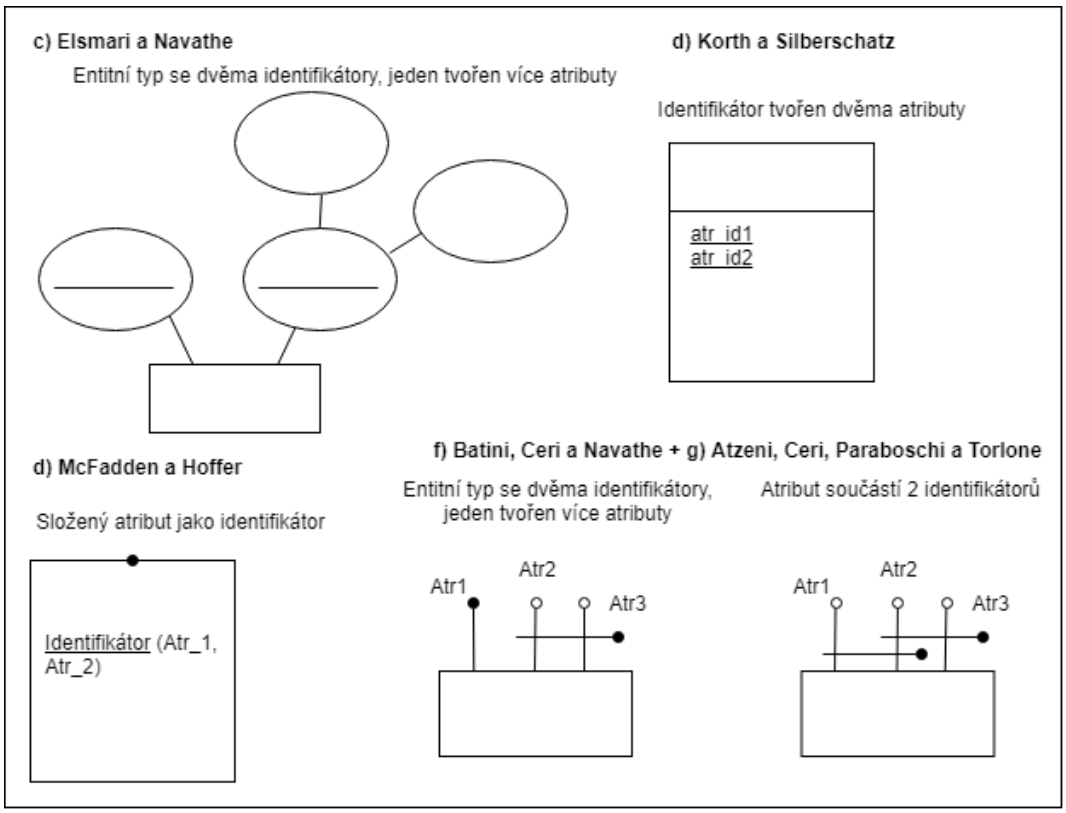
S tímto je spojen další pojem identifikátor. Je to minimální množina atributů identifikující konkrétní entitu. Nejčastěji najdeme případy, kdy identifikátor je pouze jeden identifikující atribut. U naší osoby třeba číslo občanského průkazu.

Ale identifikátor může tvořit i několik atributů, které tvoří identifikátor díky jejich kombinaci. Příkladem, chceme popsat vlaková nástupiště na nádražích. Každé nádraží má svůj unikátní název a více nástupišť. Náš entitní typ nástupiště bude obsahovat atributy název nádraží a číslo nástupiště. Pokud chceme identifikovat konkrétní nástupiště, název nádraží nám nestačí – má více nástupišť. Pomocí samotného čísla zase nerozeznáme nástupiště v různých stanicích. Pouze kombinací těchto dvou atributů nám vznikne jedinečný identifikátor pro každé nástupiště.

Někdy se nám může stát, že nejsme schopni identifikovat pro entitní typ identifikátor. V takovém případě si můžeme dovolit přidat umělý identifikátor, který je schopen určit každou entitu. Tento přístup se v praxi hojně využívá, přestože existuje i jiný identifikátor. V konceptuálním modelování je doporučeno se tomuto přístupu vyhnout a pokusit se najít přirozený identifikátor a umělý použít v krajní nouzi.

Entitní typ nemusí mít pouze jeden identifikátor. Například entitní typ osoba obsahující atribut email a telefon, můžou být oba atributy nezávislými identifikátory. Za podmínky, že informační systém dovoluje pouze jedinečné hodnoty. Některé notace umožňují existenci více identifikátorů vyznačit, ostatní umožňují pouze jeden identifikátor, potom musíme zvolit jeden ze dvou.

Následující přehled popisu přístupy značení v námi sledovaných notacích. Příklad značení jednohodnotového identifikátoru zobrazuje obrázek 3.1.



Obrázek 3.2. Schématická ilustrace značení pokročilejších případů identifikátorů.

- **a) Chen** – Značení pro identifikátor neexistuje.
- **b) Teorey** – Pro označení identifikujícího atributu se používá podtržení názvu atributu. Takto označené atributy tvoří dohromady jeden společný identifikátor pro daný entitní typ. Autor doporučuje, někdy části složeného identifikátoru modelovat pomocí slabého entitního typu, který si představíme v podkapitole 3.4.
- **c) Elmasri a Navathe** – Tato notace pracuje s označením klíčový atribut, což lze ztotožnit s pojmem identifikátor. Značení je opět stejné, pomocí podtržení textu názvu atributu. V případě, že identifikátor je tvořen jednou hodnotou, je za klíčový atribut označen tento atribut. V případě identifikátoru skládajícího se z více atributů, je nutné z hodnot vytvořit složený atribut a ten označit za klíčový atribut. Tato notace je schopna vyznačit skutečnost existence několika identifikátorů pro jednu entitu –

každý podtržený atribut je identifikátorem, viz. obrázek 3.2. Bohužel nejsme schopni vyznačit, že jeden atribut by byl součástí několika identifikátorů.

- **d) Korth a Silberschatz** – Umožňuje vyznačit pouze jeden identifikátor pro entitu, označován jako primární klíč. Atributy patřící do množiny identifikátoru jsou vyznačeny podtržením názvu atributu. Situaci skupiny atributů jako jeden identifikátor nalezneme v obrázku 3.2.
- **e) McFadden a Hoffer** – Identifikátor se uvádí na začátku seznamu atributů. Název identifikátoru je podtržen. Pokud se skládá z více atributů, značí se jako složený atribut, kde název je podtržen čarou. Např. `Nádraží_ID(Název, Číslo_Nástupiště)`, schématicky viz. obrázek 3.2.
- **f) Batini, Ceri a Navathe** – Pro značení jednoatributového identifikátoru se užívá vyplněné kolečka představující atribut. V případě víceatributového identifikátoru, jsou účastnící atributy spojeny kolmou čarou v místě spojnice atributu a entity a na konci čáry je umístěno vyplněné kolečko. Notace umožňuje pro jednu entitu vyznačit více identifikátorů. Také je možné znázornit fakt, že jeden atribut je součástí více atributů. Tyto speciální situace ilustruje obrázek 3.2.
- **g) Atzeni, Ceri, Paraboschi a Torlone** – Značení shodné s předchozí notací f).

3.2.6 Odvozený atribut

Jako poslední typ atributu vystupuje odvozený atribut. Značení existuje jen pro určité notace. Jedná se o znázornění, že informace je odvozena či vypočítaná z jiných atributů. Tímto můžeme vyjádřit, že pojmenovaná vlastnost je nám známa, ale není explicitně o entitě uložena. Typickým příkladem může být věk u entitního typu osoba. Pokud obsahuje atribut datum narození, můžeme věk osoby dopočítat. V přehledu níže uvádíme notace obsahující značení pro tento konstrukt, grafickou ukázku najdeme v obrázku 3.1:

- **c) Elmasri a Navathe** – Odvozený atribut je znázorněn čárkovanou elipsou s názvem uvnitř.
- **d) Korth a Silberschatz** – Uveden v seznamu atributů jako název a za ním prázdné kulaté závorky, např. `věk()`.
- **e) McFadden a Hoffer** – Atribut se nachází v seznamu jako název uzavřený do hranatých závorek, např. `[věk]`.

3.3 Vztah

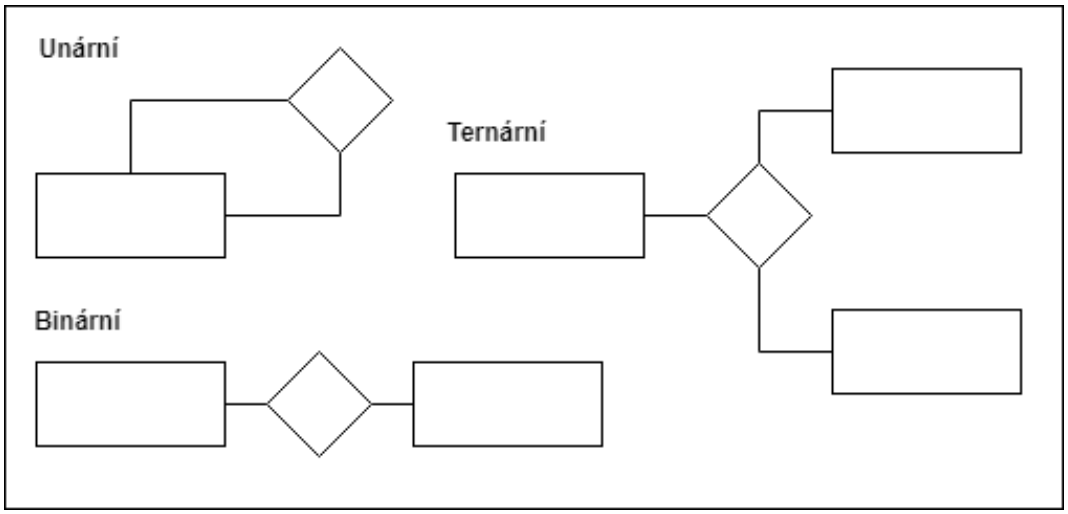
V minulé kapitole jsme již odhalili pojem vztah mezi entitami. V rámci ER jsme si vztah zavedli jako n-tici nad entitními typy jejíž složky jsou pak jednotlivé entity. Vztahy jsou reprezentovány pomocí vztahového typu, který představuje celou množinu n-tic. Je to paralela s entitním typem, který také představuje množinu konkrétních entit.

Rozlišujeme typy vztahů na unární, binární, ternární, n-arní. Unární vztah bývá často označován i jako rekurzivní. Je to vztah, který entitní typ tvoří sám se sebou. Příkladem je vztah nadřízený/podřízený. Ve vztahu vystupují dvě entity, ale stejného entitního typu, zde například zaměstnanec. Binární vztah vzniká mezi dvěma entitními typy, ternární mezi třemi, n-arní mezi počtem `n` entitních typů. Nejčastěji se setkáme s binárním vztahem. Ternární i unární vztah bývá také využíván, ale méně. Více-arní vztahy se již v praxi nepoužívají a nemodelují se snadno, nicméně nechme na paměti, že podle množinové definice je jejich existence možná.

Pro úplnost ještě dodáme, že pro dva vybrané entitní typy můžeme vytvořit více než jeden binární vztah. Samé platí pro unární i ternární vztahy.

Vztah je vždy pojmenován. Používá se označení charakterizující typ vztahu. Vzpomeňme si příklad vlastnictví aut z minulé kapitoly 2.6. Jako entitní typy vystupovaly `auto, osoba` a vytvářeli jsme mezi nimi binární vztah auto. Tento vztah jsme věcně pojmenovali `vlastnit`.

Ve všech notacích až na výjimku se entitní typ značí jako kosočtverec. Kosočtverec je čarou připojen k entitním typům účastnícím se ve vztahu. Obrázek 3.3 schématicky zachycuje zápis jednotlivých typů vztahů. Teoreyho notace uvádí název vztahu mimo kosočtverec. Ostatní uvádí název uvnitř. Příklad konkrétního značení nalezneme na příkladu v obrázku 3.7.



Obrázek 3.3. Značení jednotlivých typů vztahů

Zmíněnou výjimkou ve značení je notace McFaddena a Hoffera. Zde je vztah realizován pouze čarou mezi přímo mezi entitními typy bez prostředníka. Vztah lze pojmenovat, název nejčastěji umístíme doprostřed nad spojnici. Tento zápis přináší určitá omezení. Jedním z nich je, že jsme schopni vytvořit pouze unární a binární vztah.

Nyní už víme, jak zaznamenat existenci vztahu mezi entitními typy. Přirozeně cítíme, že ještě potřebujeme u vztahu vyznačit, jaká je účast entit na vztahu, to řeší kardinality vztahu.

3.3.1 Kardinality vztahu a integritní omezení

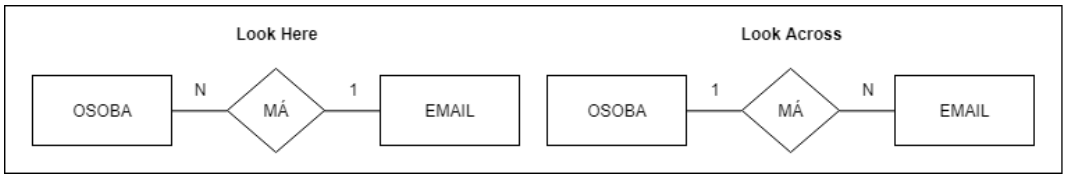
Integritní omezení jsou určitá omezení na data, v našem případě na vztahy mezi entitami, aby v databázi byla zachována integrita – koherentní stav. Integritní omezení na vztahy budeme skládat ze dvou složek – kardinalita a účast ve vztahu.

Kardinalita určuje, jaký je vzájemný výskyt entit ve vztahu. Rozlišujeme tři případy `1:1, 1:N, M:N`. V prvním případě objekt může být ve vztahu s jedním a naopak. Ve druhém jeden objekt může být ve vztahu s více než s jedním, z druhé strany je objekt ve vztahu právě s jedním. Třetí případ ilustruje, že obě strany vztahu mohou mít vztah s libovolným počtem objektů. Kardinalita určuje pomyslnou horní hranici pro vztah.

Druhou složkou omezení účast ve vztahu. Pomyslně nám určuje spodní hranici vztahu. Účast může být plná nebo částečná (totální, parciální). Při totální účast se daný entitní typ musí účastnit vztahu, výskyt minimálně jedenkrát. Parciální účast znamená, že účast ve vztahu je dobrovolná. Tedy vztah může, ale nemusí existovat.

Jak značit kardinality a účast je u všech notací jiné. Informace se zaznamenává graficky v oblasti spojnice vztahového atributu a entitního typu nebo nad spojnicí. Další

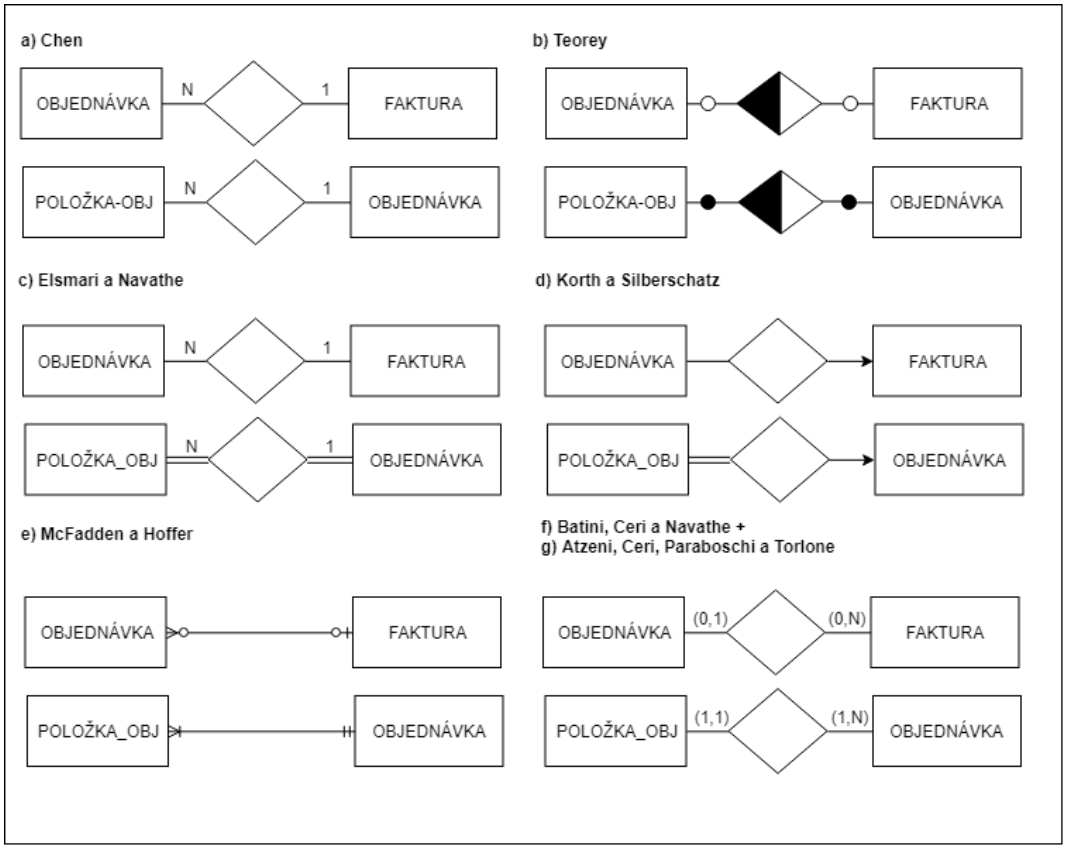
různorodostí je, na jakou stranu vztahu zapisujeme informaci o účasti či kardinalitě. Zda čteme ve směru od entitního typu, nebo se informace nachází za vztahovým typem na druhé straně. České ekvivalenty by nebyly výstižné, využijeme pojmenování směru čtení v angličtině (Look Here, Look Across). Rozdíl směru čtení ilustruje obrázek 3.4 na vztahu 1:N. Osoby a emailové adresy, jedna osoba může mít více emailových adres. Emailová adresa je vlastněna pouze jednou osobou.



Obrázek 3.4. Ilustrace čtení omezení na vztah.

V následující části se budeme zabývat konkrétním značením pro naše notace. Popis můžeme spojit s obrázkem 3.5, který ukazuje možné použití na dvou jednoduchých příkladech.

- **a) Chen** – Originální Chenova notace se čte směrem Look Across, neobsahuje informaci o účasti, značí se kardinalita – 1:1, 1:N, M:N (písmena by měla být odlišná). Později notace byla upravena a účast doplněna. Totální participaci značí plný kroužek na spojnici mezi entitním typem a vztahem. Pro částečnou prázdné kolečko.
- **b) Teorey** – Čtení probíhá způsobem Look Across. Účast se značí prázdným kolečkem na spojnici entitního a vztahového typu pro částečnou. Pro plnou účast kolečko je vyplněno (značení není nutné uvádět, pokud není, bere se jako by bylo). Kardinalita se značí do vztahového typu. Ten je rozdělen na půlku. Strana vztahu kardinality N je vyplněna, strana kardinality 1 je prázdná.
- **c) Elmasri a Navathe** – Pro vyznačení kardinalit se využívá textu uvedeného nad spojnicí(1, N). V případě kardinalit N:M by se písmena měla lišit. Čtení kardinality probíhá Look Across. Pro čtení účasti probíhá naopak, tedy Look Here. Totální účast se značí nahrazením spojnice dvojitou čárou. Parciální je vyznačena jednoduchou čarou.
- **d) Korth a Silberschatz** – Kardinalita 1 je značena jednoduchou čarou, kde na konci u entitního typu je šipka. Pro kardinalitu N se používá jednoduchá čára. Pokud chceme zdůraznit totální účast pro kardinalitu N, použijeme dvojitou čáru. Čtení a zápis probíhá směrem Look Across.
- **e) McFadden a Hoffer** – Notace využívá pohledu Look Across. Symboly označující účast a kardinalitu se značí na kraj spojnice u entitního typu. Kardinalitu N tvoří tzv. kuří nožka (z anglického označení Crow's Foot) a navazuje se na entitní typ. Pro kardinalitu 1 na spojnici vyznačíme kolmou čárku kousek od konce. Účast totální se značí čárkou kolmou na spojnici. Částečnou účast značíme malým kroužkem na spojnici. V případě kardinality N se účast navazuje na kuří nožku. Jinak se nechá mezera.
- **f) Batini, Ceri a Navathe** – Pro značení kardinalit se využívá formy nazvaném min-max. V kulatých závorkách nad spojnící jsou uvedeny dvě čísla oddělena čárkou. První značí účast (0 nebo 1), druhé kardinalitu (1,N). Kombinace vypadá např. (0,N). Čtení probíhá směrem Look Here.
- **g) Atzeni, Ceri, Paraboschi a Torlone** – Značení shodné s notací f).



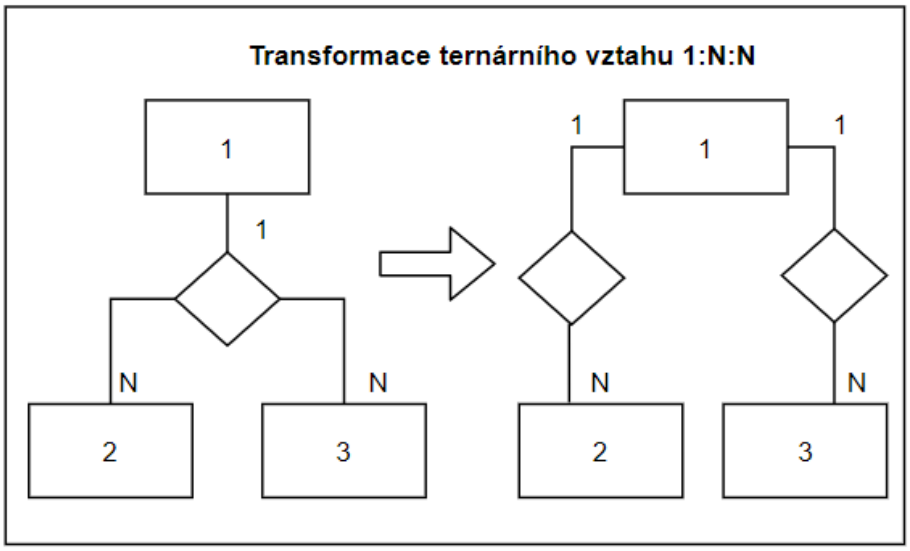
Obrázek 3.5. Ukázka integritních omezení na dvou příkladech. Faktura a objednávka ve vztahu – faktura může být vystavena na `0..N` objednávek, na objednávku může nebo nemusí být vystavena faktura (`0..1`). Pro příklad objednávka a položka objednávky platí, že objednávka musí mít `1..N` položek. Položka objednávky musí mít objednávku (`1..1`).

3.3.2 Ternární vztah – kardinality

Ternární vztah nebývá tak často používán jako binární vztah, nicméně v praxi se lze s ním setkat. Způsob značení se trochu liší. Zejména z hlediska, jak chápat přidělení kardinalit a také kam je uvést. Změna spočívá v tom, že omezení přidělujeme ve vztahu jedné entity vůči dvojici zbývajících entitních typů. Z předešlé části popisující značení kardinalit, zejména pro binární a unární vztah, jsme mohli dojít k závěru, že většina notací používá směr čtení Look Across. Ten nyní nemůžeme použít, jelikož není kam kardinalitu umístit. Lze použít pouze zápis Look Here. Na okraj uvedeme, že v případě ternárního vztahu kardinalit 1:N:N (po dvojicích, horní mez kardinality), lze transformovat do dvou binárních vztahů. Entitní typ, ke kterému v ternárním vztahu připadala 1 dvojice, nyní bude mít dva vztahy, každý s kardinalitou 1, zbylé entitní typy s ním budou mít kardinalitu N (viz obr. 3.6).

V následujícím přehledu shrneme, jak notace ke značení přistupují.

- **a) Chen** – Značení probíhá Look Here pro ostatní dvojice. V případě N kardinalit používáme odlišná písmena.
- **b) Teorey** – Pro značení používá Look Here. To znamená, že participace je značena na spojnici vztahového typu a entitního typu pro ostatní dvojice. Vztahový atribut je rozdělen na 4 trojúhelníky. Kardinalita vzhledem ke dvojici je umístěna v části směrem k jednomu entitnímu vztahu.



Obrázek 3.6. Myšlenka transformace ternárního vztahu do dvou binárních.

- **c) Elmasri a Navathe** – Značení stejné, jen zapisujeme na stranu Look Here. Arternativně můžeme použít formu zápisu min-max pro kardinalitu a účast. Autoři doporučují vyhnout se použití ternárního vztahu, pokud je to možné.
- **d) Korth a Silberschatz** – Stejné značení jako pro binární vztah. Pro kardinalitu N čára, pro 1 čára s šipkou. Pro dvojici se uvádí naproti přes vztahový atribut, tedy zápis a čtení Look Here.
- **e) McFadden a Hoffer** – Není třeba řešit, ternární vztah neexistuje, vše se modeluje pomocí binárního.
- **f) Batini, Ceri a Navathe** – Standardně pro vztahy používá Look Here. Značí se ve tvaru min-max.
- **g) Atzeni, Ceri, Paraboschi a Torlone** – Značení shodné s notací f).

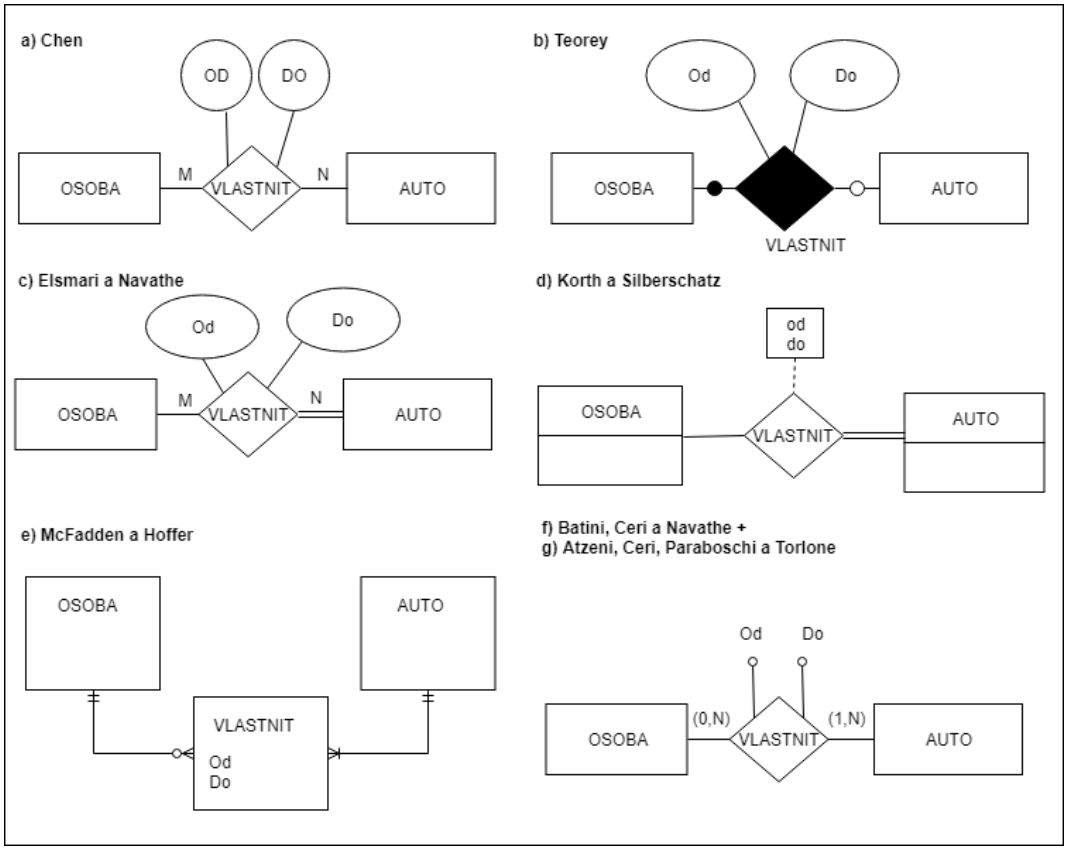
3.3.3 Vztah s atributy

Pojem atribut již známe. Stejně jako přiřazujeme atribut entitnímu typu, tak můžeme atribut přiřadit i vztahovému typu. Z uvedených typů atributů v kapitole 3.2 můžeme přiřadit všechny typy, kromě označení identifikujícího atributu. Ten u vztahového typu nemá význam, protože vztah nepotřebujeme identifikovat. Vztahovému typu můžeme přiřadit jednohodnotový, vícehodnotový, složený nebo odvozený atribut.

Jelikož notace McFaddena a Hoffera nemá značení pro vztahový typ, proto není možné vztahu přiřadit atributy. Vzniklá situace se řeší vytvořením nového entitního typu s atributy, který představuje vztah a ten je spojen vztahem s entitními typy, které by jinak ve vztahu vystupovali. Příklady značení pro jednotlivé notace si demonstrujeme v obrázku 3.7 na příkladu vlastnictví aut, kde vztah vlastnit bude obsahovat atributy `od, do`, které budou představovat dobu vlastnictví. U kardinalit předpokládáme vztah N:N. Osoba může vlastnit jakýkoliv počet aut, auto by mělo být vlastněno aspoň jednou. Atributy entit zanedbáváme. Notace Korth a Silberschatz své atributy má umístěné v rámečku poblíž vztahového typu a připojené čárkovanou čarou.

3.4 Slabý entitní typ

Slabý entitní typ je speciální případ entitního typu, jehož existence je závislá na jednom či více vztazích s entitními typy. Závislost je tvořena díky tomu, že slabý entitní typ



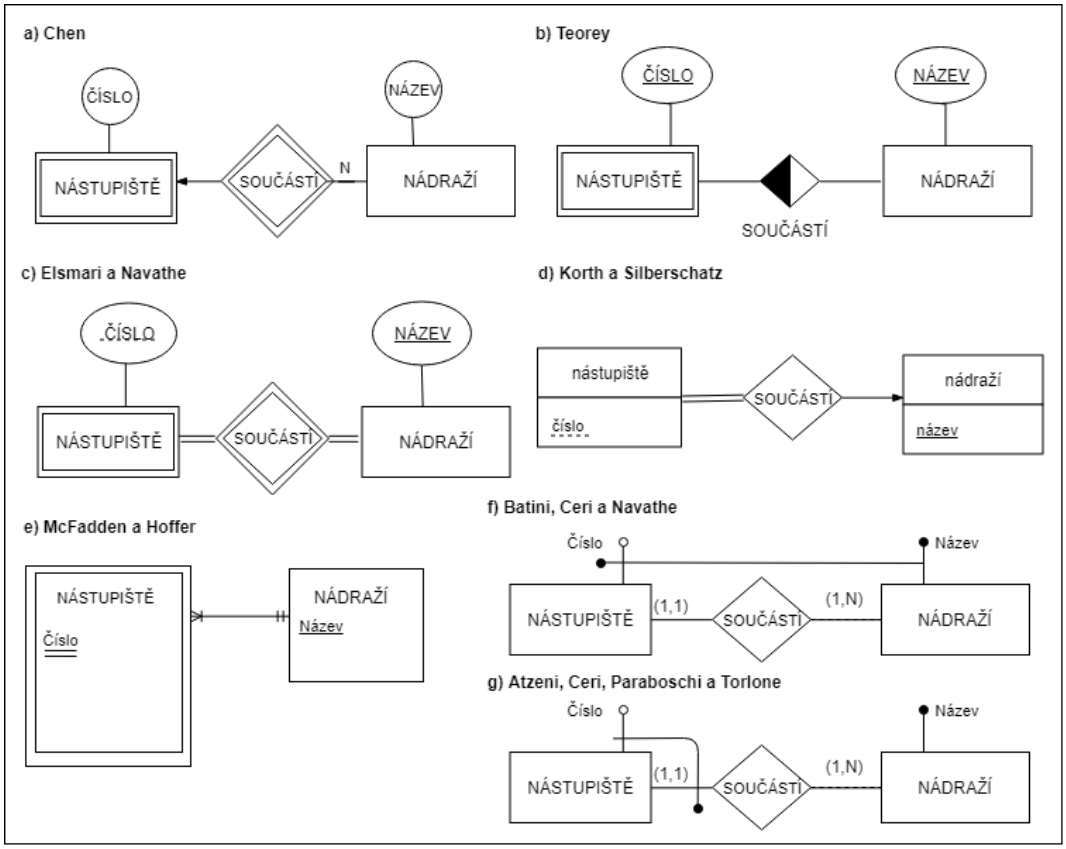
Obrázek 3.7. Příklad vztahu s atributy na vlastnictví aut osobami. Atributy entitních typů jsou zanedbány.

není možné identifikovat pomocí jeho dostupných atributů. Je možná pouze částečná identifikace. Úplná identifikace probíhá kombinací částečného identifikátoru slabého entitního typu a identifikátorů entitních typů, na nichž je závislý. Kardinalita spolu identifikujícího entitního typu vůči slabému entitnímu typu ve vztahu musí být 1, účast musí být totální. Tímto lze konkrétní instanci slabého entitního typu rozlišit.

Jako demonstrativní příklad budeme chtít modelovat nádraží a jejich nástupiště. Entitní typ nádraží bude identifikován podle názvu, který je jedinečný. Dalším vystupujícím entitním typem je nástupiště, u něho budeme chtít rozlišovat číslo nástupiště. Přirozeně vzniká vztah mezi entitami a cítíme, že existuje přímá závislost nástupiště na nádraží. Bez vztahu konkrétního nástupiště na nádraží, nejsme schopni nástupiště podle čísla rozlišit. Pokud k číslu nástupiště přidáme referenci konkrétního nádraží, jsme schopni všechny nástupiště rozlišit.

Obrázek 3.8 zobrazuje náš příklad vymodelovaný pomocí sledovaných notací. Nyní se podíváme, jak popis pomocí notací probíhá.

- **a) Chen** – Slabý entitní typ je znázorněn pomocí obdélníka s dvojitou čárou. Identifikující vztahový atribut je také zakreslen dvojitou čarou. Od identifikujícího vztahového typu je vedena šipka ke slabému entitnímu typu. Identifikující atributy nejsou nijak vyznačeny, protože notace pro ně nemá značení.
- **b) Teorey** – Slabý entitní typ je znázorněn dvojitým rámem obdélníka. Podtržené atributy slabého entitního typu tvoří dohromady částečný identifikátor. Identifikující vztah není nijak vyznačen.



Obrázek 3.8. Způsoby značení slabého entitního typu na příkladu nádraží s nástupišti.

- **c) Elmasri a Navathe** – Slabý entitní typ je znázorněn dvojitým rámem obdélníka. Částečný identifikátor je znázorněn čárkovaně. V případě kombinace několika atributů, musí být tento identifikátor složený. Spolu identifikující vztahový typ je označen dvojitým rámem.
- **d) Korth a Silberschatz** – Slabý entitní typ není speciálně označen. Atributy podílející se na částečném identifikátoru jsou podtrženy čárkovaně. Značení pro určení identifikujícího vztahu není zavedeno.
- **e) McFadden a Hoffer** – Slabý entitní typ je znázorněn dvojitým rámem obdélníka. Částečný identifikátor je označen dvojitým podtržením. V případě identifikátorů z více atributů, musí být složen. Identifikující vztah není rozlišen.
- **f) Batini, Ceri a Navathe** – Slabý entitní typ není explicitně označen. Nicméně, slabý entitní typ má skupinu atributů tvořící částečný identifikátor přeškrtnutou a spojenou s identifikátorem spolu identifikujícího entitního typu. Na konci spojnice u slabého entitního typu je malé plné kolečko značící, že následující atributy tvoří identifikátor.
- **g) Atzeni, Ceri, Paraboschi a Torlone** – Identifikátor slabého entitního typu je tvořen spojením atributů (tvořící částečnou identifikaci) spolu se vztahy, které spolu dohromady tvoří identifikátor. Na jednom konci spojnice je umístěno malé plné kolečko.

3.5 Generalizace / specializace

Posledním zbývajícím konstruktem je specializace či generalizaci. V původní myšlence Petera Chena tento prvek neobjevíme. S postupem vývoje objektově orientovaného programování začal být do ER od různých autorů doplňován koncept dědičnosti a později

ustálen. Koncept pracuje se vztahem is-a mezi objekty a existencí podtříd a super tříd. Generalizace je proces, kdy slučujeme společné vlastnosti jednotlivých objektů (podtříd) do jedné super třídy. Specializace je obrácený proces, rozšiřujeme vlastnosti objektu (super třídy) a tvoříme specializované podtřídy. Super třída je tedy nadřazená přiřazeným podtřídám a ty ze svého předka dědí. Atributy, které přiřadíme předkovi, má i jeho potomek. Ten může potom své atributy jakkoliv rozšířit.

Jako příklad nadtřídy si uvedeme vozidlo. Řekněme, že tato entita bude obsahovat atribut počet kol a registrační značku. Jeho potomky budou autobus, nákladní auto. Obě nové entity přebírají atribut počet kol, registrační značka. U těchto dvou vozidel budeme chtít nějaké informace odlišit, například u autobusu budeme chtít znát přepravní kapacitu jako počet osob. U nákladního vozidla nás bude zajímat počet palet, které je možné naložit do přepravního prostoru. Tyto dvě informace nejsou slučitelné, proto musí existovat dvě samostatné entity. Vyčlenění společných atributů do předka přináší zjednodušení celého schématu. Vztahy, které by jinak byly duplicitní s oběma entitními typy, je možné vytvořit pouze s jejich předkem – vozidlo. Zároveň je povoleno, že potomek může mít své další vlastní vztahy s jinými entitními typy nad rámec svého předka. Z hlediska identifikátorů, potomci jsou identifikováni na základě identifikátoru předka, zároveň potomek může mít i svůj vlastní identifikátor.

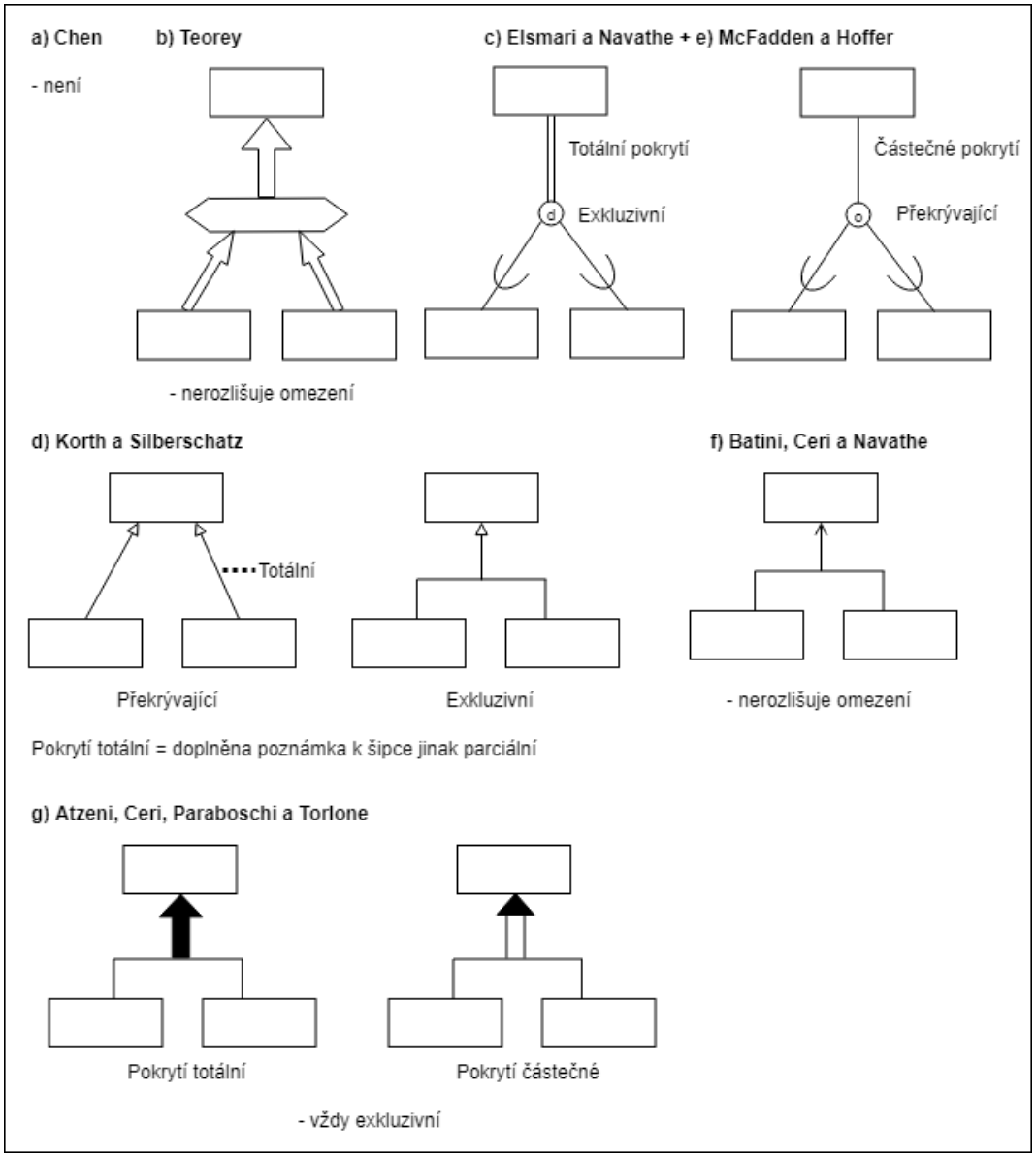
U popisu této dědičnosti řešíme dva pojmy týkající omezení v rámci existence entit.

- **Pokrytí** – Rozlišujeme totální a parciální (kompletní, částečná). Je vynucení specializace entity. V případě totální, každý objekt musí být objektem nějaké podtřídy (samostatná instance nadtřídy neexistuje). Pokud je parciální, objekty mohou být v roli nadtřídy i podtřídy. Příkladem totálního pokrytí může být vozidlo a jako podtřídy všechny možné typy vozidel (autobus, osobní auto, nákladní...). Jako parciální pokrytí můžeme uvést osobní auto, podtřídy sedan, offroad. Pokud entita nespadá do jedné ze skupin, je to stále osobní auto.
- **Překrytí** – Rozlišujeme totální a parciální (exkluzivní a překrývající). Pokud je vztah exklusivní, instance může být členem pouze jedné podtřídy. V opačném případě entita může být členem více podtříd. Předchozí příklad rozdělení vozidel do všech skupin je příklad exkluzivního překrytí. Vozidlo se může nacházet pouze v jedné kategorii. Jako překrývající můžeme označit pro osoby a jejich podtřídy nadřízený, podřízený. Jedna entita může být zároveň šéfem, ale i podřízeným.

Kombinací vlastností nám vznikají 4 možnosti omezení na instance entit v rámci super tříd/podtříd.

Na obrázku 3.9 máme ilustrovány způsoby značení, které si nyní popíšeme pro jednotlivé notace.

- **a) Chen** – Značení neexistuje.
- **b) Teorey** – Mezi entitní typy představující super třídu a podtřídy je umístěn splácnutý šestiúhelník, který představuje is-a vztah. Uvnitř může být uvedeno pojmenování, které charakterizuje hierarchii. Od podtříd jsou připojeny šipky a pak směřuje šipka k super třídě. Notace nerozlišuje omezení.
- **c) Elmasri a Navathe** – Doprostřed mezi entitní typy je umístěno kolečko, uvnitř je uvedeno písmeno `d` (disjoint), které představuje exkluzivní překrytí nebo písmeno `o` (overlapping) představující možnost překrývání. Podtřídy jsou ke kolečku připojeny čarou, na které se nachází půlkruh otevřený směrem ke kolečku a nadtřídě. Kolečko je spojeno s nadtřídou, tato spojnice graficky znázorňuje pokrytí. Dvojitá čára – totální, jednoduchá – parciální pokrytí. Notace umožňuje znázornit všechny typy omezení a jejich kombinace.



Obrázek 3.9. Ilustrace značení generalizace/specializace v rámci jednotlivých notací. Z trojice horní entitní typ představuje super třídu, další dva spodní jsou jeho podtřídou. Kde existuje rozlišení omezeních, je vyznačeno.

- **d) Korth a Silberschatz** – Notace rozlišuje způsob značení překrytí na základě toho, zda jsou podtřídy spojeny či ne. Spojené třídy tvoří hierarchii s exkluzivním překrytím. Pro překrývající dědičnost jsou podtřídy připojeny samostatně pomocí šipky. Pro vyjádření totálního pokrytí je k šipce vedoucí od podtřídy nebo spojnice podtříd doplněna poznámka – `totální`. Jinak se hierarchie považuje za parciální.
- **e) McFadden a Hoffer** – Shodné značení s notací c).
- **f) Batini, Ceri a Navathe** – Jednotlivé podtřídy jsou spojeny a od spojnice vede šipka k super třídě. Omezení pokrytí a překrytí nerozlišujeme.
- **g) Atzeni, Ceri, Paraboschi a Torlone** – Notace je stavěna z předpokladu existence všech čtyř kombinací omezení. Nicméně dle autorů je možné překrytí vždy vhodnou tvorbou hierarchie vytvořit. Tedy hierarchie představuje vždy překrytí exkluzivní. Podtřídy

jsou spojeny čarou, od které vede šipka k nadtřídě. Pokud je vyplněna, značí totální pokrytí. Pro případ částečného pokrytí, je vyplněn jen konec šipky.

3.6 Dostupnost konstruktů v jednotlivých notací

V rámci popisu jednotlivých konstruktů jsme si uváděli, pokud konstrukt v notaci není dostupný. Pro úplnost si provedeme srovnání dostupnosti na jednom místě. V následující tabulce na obrázku 3.10 máme vyznačeno, zda je konstrukt v notaci obsažen. Tímto naše porovnání notací uzavřeme.

3.7 Další modely označované jako ER notace

Okrajem na závěr si ještě uvedeme některé další modely, na které můžeme při krátkém hledání po různých notacích narazit a jsou označovány jako ER. Označení není zcela chybné, protože daný způsob diagramu zobrazuje entity a vztahy mezi nimi. Nicméně modely, buď nemají s myšlenkou Petera Chena nic společného, nebo se jí pouze inspirovali. Tyto modely jsou trochu jinak myšlenkově postaveny, nejedná se jen o jiný způsob značení stejné věci, ale o trochu jiný koncept.

Uvedeme jednu celkem zásadní odlišnost, kterou většina těchto modelů má a to, že vztahy nemohou mít atributy. Toto omezení se nemusí zdát jako zásadní problém. Situaci umíme vyřešit. Pokud chceme vymodelovat vztah, který má určitou vlastnost, musíme ho znázornit jako entitu a vytvořit vztahy se členy, které vstupují do původního vztahu. Tento způsob by nám měl připomínat relační model, které se využívá v relačních databázích, můžeme si všimnout přístupu tvoření takzvaných mezitabulek. Vzpomeňme si na proces modelování ve třech úrovních abstrakce. Pokud se chceme dívat na ER jako koncept, který by nám měl umožňovat modelovat na konceptuální vrstvě, tak nám tento způsob přístupu již modeluje na logické vrstvě, která zohledňuje určité strukturování dat. V oblasti modelování dat můžeme narazit na následující notace, někdy pojmenované pouze po svých autorech:

Charles Bachman – Můžeme se setkat s pojmenováním i Bachmanův diagram. Je označován za ranou formu ER a vznikl před publikací ER konceptu Petera Chena, které této práce využil.

Richard Barker – Notace vznikla v roce 1981 a byla využita v CASE nástroji firmy Oracle.

Information Engineering – Původním autorem je Clive Finkelstein, práci později více popularizoval James Martin. Častěji označována jako *Crow's Foot* notace podle grafického zápisu kardinalit, které připomínají kuří nožku.

IDEF1X – Značení vyvinuté pod vedením Amerických vzdušných sil v rámci programu ICAM (*Integrated Computer-Aided Manufacturing* volně přeloženo integrovaná počítačem podporovaná výroba). Na práci se podílel Peter Chen.

UML – Uvádíme z důvodu, že se jedná o současný populární nástroj pro modelování dat.

	a) Chen	b) Teorey	c) Elsmari a Navathe	d) Korth a Silberschatz	e) McFadden a Hoffer	f) Batini, Ceri a Navathe	g) Atzeni, Ceri, Paraboschi a Torlone
Entitní typ	ANO	ANO	ANO	ANO	ANO	ANO	ANO
Jednohodnotový atribut	ANO	ANO	ANO	ANO	ANO	ANO	ANO
Vícehodnotový atribut	X	X	ANO	ANO	ANO	?	ANO
Složený atribut	X	X	ANO	ANO	ANO	?	ANO
Odvozený atribut	X	X	ANO	ANO	ANO	?	ANO
Identifikující atribut	X	ANO	ANO	ANO	ANO	ANO	ANO
Identifikátor tvořený více atributy	X	X	ANO	ANO	ANO	ANO	ANO
Více identifikátorů na jednom entitním typu	X	X	ANO	X	X	ANO	ANO
Atribut součást více identifikátorů	X	X	X	X	X	ANO	ANO
Vztah	ANO	ANO	ANO	ANO	ANO	ANO	ANO
Vztahový typ	ANO	ANO	ANO	ANO	X	ANO	ANO
Vztahový typ s atributy	ANO	ANO	ANO	ANO	X	ANO	ANO
Ternární vztah	ANO	ANO	ANO	ANO	X	ANO	ANO
Slabý entitní typ	ANO	ANO	ANO	ANO	ANO	ANO	ANO
Označení identifikujícího vztahu slabého entitního typu	ANO	X	ANO	ANO	X	ANO	ANO
Generalizace / Specializace	X	ANO	ANO	ANO	ANO	ANO	ANO
Rozlišení omezení hierarchie	X	X	ANO	ANO	ANO	X	ANO

Obrázek 3.10. Tabulka zobrazující výskyt konstruktu v jednotlivých notacích. Řádky představují konstrukt, sloupce jednotlivé notace. ANO – vyskytuje se, X – značení neexistuje, ? – existence není potvrzena dle dostupné literatury

Kapitola 4
Existující řešení

V minulé kapitole jsme si představili, jaké prostředky můžeme použít pro vyjádření ER digramu. Tyto vědomosti nyní budeme moci uplatnit v následujícím rozboru dostupných programů na trhu, které je možné využít pro vytváření ER modelu.

Hlavním cílem této kapitoly bude představit si již existující řešení, uvedené aplikace se pokusíme podrobně rozebrat. Na začátku provedeme krátké seznámení s danou aplikací z hlediska její dostupnosti pro uživatele, platformové závislosti a použité technologie. Dále se zaměříme, jakou z uvedených ER notací řešení používá a jestli umožňuje vytvořit všechny konstrukty, které nám daná notace nabízí. Kromě zaměření na to, jaký model můžeme vytvořit, budeme také zkoumat celkovou funkcionalitu aplikace a její ergonomii při práci.

Na závěr si provedeme shrnutí všech základních rysů uvedených aplikací a pokusíme se říci, které řešení je vhodné použít.

4.1 Rozdělení řešení podle reprezentace modelu

Nejdříve než začneme s rozborem jednotlivých aplikací, zavedeme si jednoduché rozdělení, podle kterého aplikace budeme rozlišovat. Toto rozdělení není založeno na žádné odborné literatuře, bude se jednat jen o naši čistě pracovní terminologii.

Kritérium, na kterém bude toto rozdělení založeno, je skutečnost, zda při práci na tvorbě diagramu vytváříme schéma, které je následně graficky reprezentováno jako ER diagram. Pokud do schématu chceme přidat entitní typ, vložíme do schématu konstrukt, kterému lze nastavit vlastnosti a chování, které entitní typ může mít. Zároveň prostředí chápe logiku konstruktů a jejich vztahů mezi sebou. Například nedovolí vytvářet kombinace konstruktů, které nedávají smysl, nevytváří validní schéma a jejich modelování by nejlépe nemělo být umožněno. Nesnažíme se pomocí grafických prvků vytvářet diagram, který nám zobrazuje schéma.

Na základě tohoto kritéria uvedeme dvě možnosti, jak budeme aplikace dělit:

- **modelovací** – Budeme označovat nástroj, který splňuje výše uvedené kritérium, tj. aplikace obsahuje logiku pro tvorbu ER modelu, v aplikaci je vytvářeno schéma, které je následně reprezentováno jako diagram.
- **kreslící** – Budeme označovat nástroj, který umožňuje „nakreslit“ diagram, ale jeho tvorba probíhá pomocí nástrojů, které nejsou určeny primárně pro tvorbu ER konstruktů. Program konstrukty ER nemusí obsahovat, ale pomocí jeho jiných prostředků, jsme schopni tyto prvky napodobit. Vytváříme z objektů – obdélník, čára, elipsa, kolečko, text, atd. V případě, že chceme provést změnu ve schématu (kardinality, typ atributu), neměníme hodnoty v nastavení (můžeme tolerovat výměnu objektu), ale musíme provést ruční opravu diagramu.

4.2 Demonstrativní příklad

Po zavedení našeho rozdělení, kterým můžeme aplikace dělit na dva hlavní typy, ještě potřebujeme uvést nějaký jednoduchý příklad, na kterém bychom mohli demonstrovat použití aplikace. Následující příklad nebude obsahovat všechny možné konstrukty. Vybereme pouze základní, protože potřebuje ukázat, jak základní značení v aplikaci vypadá a demonstrovat, jakou notaci můžeme vytvořit. Nicméně v každém programu se i zaměříme, jestli má aplikace nějaké omezení na použití konstruktů.

Máme situaci, že potřebujeme namodelovat zjednodušený systém umožňující evidovat vozidla, osoby jako vlastníky, jejich řidičské průkazy a získané skupiny oprávnění s platností. Entitní typ vozidlo bude obsahovat identifikátor VIN kód a atribut název pro uvedení o jaké vozidlo se jedná (např. značka + typ). Vozidlo bude možné specializovat do dvou podtříd – osobní a nákladní. Osobní auto bude moci ukládat informaci o přepravní kapacitě osob. Nákladní bude ukládat informaci, kolik palet umožňuje přepravit. Hierarchie bude částečná a exkluzivní. Dalším entitním typem bude osoba. O ní chceme evidovat jméno – křestní a příjmení, číslo občanského průkazu (bude identifikátorem), datum narození. Vozidlo a osoba budou mezi sebou tvořit vztah vlastnit, které bude obsahovat atributy od a do. Vozidlo může mít 1..N vlastníků. Osoba může vlastnit 0..N vozidel. Dalším entitním typem bude řidičské oprávnění, o něm evidujeme číslo, které slouží jako identifikátor. Ve vztahu osoby a řidičského oprávnění, osoba může mít 0..1 řidičských průkazů, řidičské oprávnění má právě jednoho vlastníka. Posledním entitním typem je skupina oprávnění, která bude modelována slabým entitním typem, jejím částečným identifikátorem je platnost od, do a skupina oprávnění (A, B, C, atd.). Identifikující vztah s řidičským oprávněním bude takový, že řidičskému oprávnění lze přiřadit 1..N skupin, entita skupiny připadá jednomu řidičskému oprávnění.

Nyní máme zavedeno vše potřebné pro naše zkoumání, proto můžeme přejít k samotnému rozboru jednotlivých řešení.

4.3 ERDPlus

První aplikaci, kterou si představíme je řešení pojmenované ERDPlus [14]. Jedná se o webovou aplikace běžící v prohlížeči (HTML, CSS, JS). Díky tomu nejsme omezeni na operační systém. Vydavatel umožňuje použít aplikaci ve dvou režimech. Za prvé je to režim s uživatelským účtem a za druhé režim bez registrace.

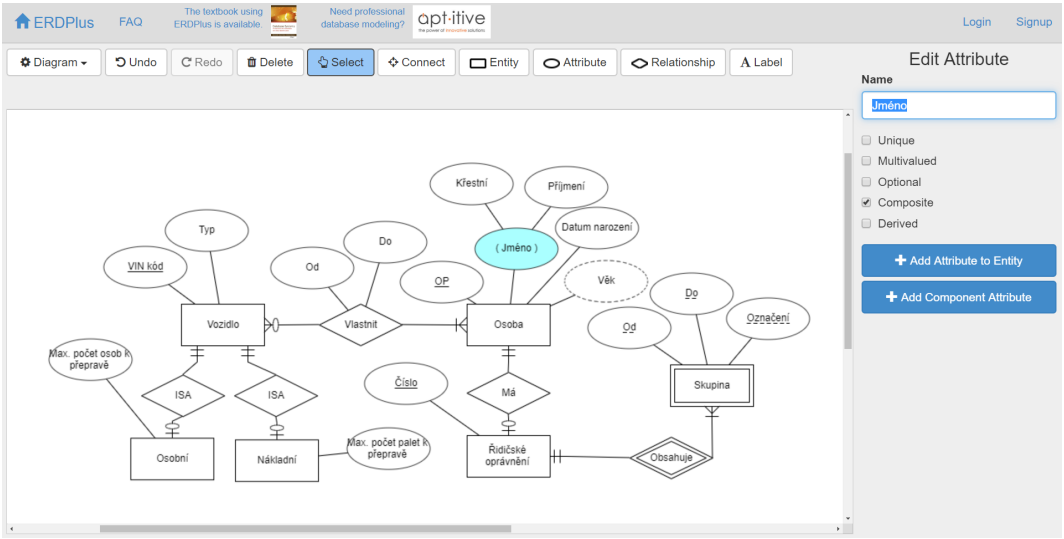
Pokud zvolíme variantu bez registrace, nejsme omezeni na dostupné funkce pro modelování. Svou práci si můžeme uložit do počítače a později opět načíst. V případě registrace můžeme využít uložení projektů na server vydavatele a můžeme k nim přistupovat přes účet odkudkoliv. Další dodatečnou výhodou může být export diagramu do SQL, což je funkcionalita, kterou pro náš účel nepotřebujeme, jen může usnadnit práci při pozdější implementaci návrhu.

Nyní se podíváme na modelovací část aplikace. Hned při první interakci si můžeme všimnout, že aplikaci podle našeho rozdělení lze označit jako modelovací na základě toho, že v případě umístění objektu do plátna máme po pravé straně programu, kde upravujeme vlastnosti objektu a ty se vykreslují.

- **entitní typ** – umožňuje nastavit typ jako normální, slabý, asociativní
- **vztahový typ** – může být nastaven jako normální či identifikující. Další nastavení se týká účasti ve vztahu a kardinalit – povinný, nepovinný, nespecifikovaný a kardinalit – jedna, mnoho, nespecifikováno.

- **atribut** – lze nastavit jako normální, unikátní, částečně unikátní, volitelný, složený, odvozený
- **text** – vložení textového řetězce do plátna

Z hlediska notace není možné označit jednu používanou notaci. Vztahové a entitní typy, atributy přesně odpovídají značení notace Elsmari a Navathe. Neodpovídá záznam kardinalit a účasti ve vztahu. Zde se používá značení tzv. kuří nožky, které například využívá notace McFaddena a Hoffera. Bohužel v programu se nepodařilo najít prostředky pro tvorbu hierarchie. Pokud chceme použít, musíme dědičnost modelovat pomocí vztahu nazvaný jako isa mezi nadtřídou a podtřídou. Dále není možné vytvořit ternární vztahový typ.



Obrázek 4.1. Ukázka práce s programem ERDPlus, diagram znázorňující náš příklad.

Z dojmů při tvorbě ukázky (viz. obr. 4.1) můžeme říci, že se s aplikací pracuje velmi pohodlně. Oceníme snadné vytváření schéma díky možnosti nastavit vlastnosti objektům v postranním panelu. Dobře funguje i přidávání atributů pomocí tlačítka, které vytvoří nad entitním typem atribut a provede automatické spojení. Stačí uvést název a nastavit typ atributu. V aplikaci se opravdu soustředíme na tvorbu schématu a vykreslování dělá program za nás.

Závěrem lze říci, že aplikace je jednoduchá a povedená. Bohužel musíme vytknout absenci ISA hierarchie a chybějící možnost vytvořit ternární vztah. Pro tvorbu základních schémat je aplikace dobrou volbou. V případě potřeby vytvářet náročnější schéma, můžeme být omezení absencí těchto konstruktů. Především při potřebě vytvořit kvalitní a dobře čitelný návrh.

4.4 Creately

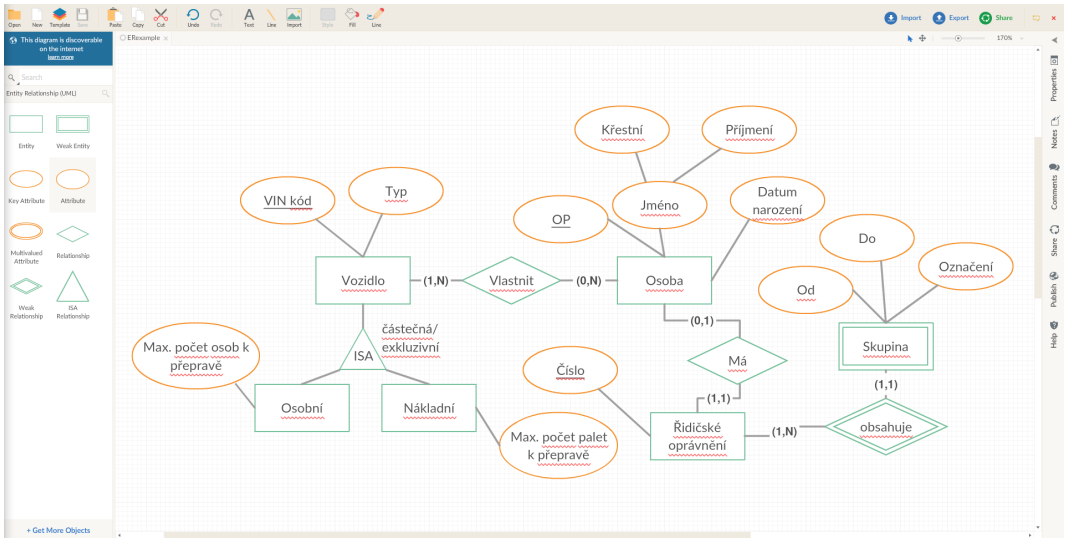
Další aplikaci, kterou si představíme je Creately [15]. Toto řešení není zaměřeno jen na tvorbu ER diagramů, ale umožňuje vytvářet širokou škálu typů diagramů. Od různých business diagramů (flowchart, mind map, wireframe ...) až po technické diagramy (UML, sekvenční, diagram tříd, diagramy nasazení ...).

Vydavatel své řešení podporuje napříč běžnými platformami, aplikace je dostupná pro desktop zařízení (systémy windows, linux, Mac OS X), mobilní zařízení (systémy

Android, iOS). Pokud nechceme software instalovat, je možné použít online verzi skrze webový prohlížeč. Aplikace také podporuje použití pro skupinu uživatelů (týmové použití), kteří mohou sdílet svoji práci mezi sebou.

Vzhledem k široké funkcionalitě a podpory na různých platformách nás nepřekvapí, že plně dostupné řešení je placeno. Nebudeme zacházet do velkých detailů ohledně placených verzí, spíše se zaměříme, jak můžeme software použít bezplatně. Vydavatel umožňuje bezplatné použití takzvané veřejné verze. Uživateli stačí pouze provést registraci a může aplikaci začít používat. V tomto režimu je uživatel limitován na 5 veřejných diagramů. Veřejný diagram znamená, že diagram, který chce uživatel začít vytvářet a mít uložený pod svým účtem, je veřejně dostupný pro ostatní. Pokud uživatel nechce mít svůj diagram veřejný, musí přejít na placenou verzi. Dále neplacená verze je omezena na využití některých importů a exportů.

Pro ukázku použití aplikace můžeme použít demo verzi ve webovém prohlížeči [16]. Pomocí této verze se pokusíme vytvořit diagram pro náš příklad.



Obrázek 4.2. Ukázka práce s programem Creately, diagram znázorňující náš příklad.

Podobu programu i s naším příkladem si můžeme prohlédnout na obrázku 4.2. Zprvu si všimneme v levém panelu, že program umožňuje použít následující objekty – entitní typ, slabý entitní typ, atribut, klíčový atribut, vícehodnotový atribut, vztahový typ, slabý vztahový typ, ISA hierarchii. Pokud si zkusíme spojit dva entitní typy přes vztahový typ, zjistíme, že program podporuje vlastní označení kardinalit pomocí textu i grafickou Crow's Footovu notaci. V našem případě jsme zvolili formu min-max se směrem čtení Look Here. Dostupné konstrukty odpovídají notaci Elsmari a Navathe. Značení kardinalit si můžeme zvolit. Jako isa hierarchii můžeme použít trojúhelník, který spojíme s nadtřídou a podtřídami. Poznámkou můžeme doplnit omezení.

Po první chvíli používání lze tento nástroj označit jako kreslící. Ačkoliv nástroj obsahuje objekty pojmenované podle ER terminologie, tedy nesetkáme se s označením, obdélník, kosočtverec, trojúhelník, tak při procesu tvorby cítíme, že kreslíme obrázek, který představuje diagram.

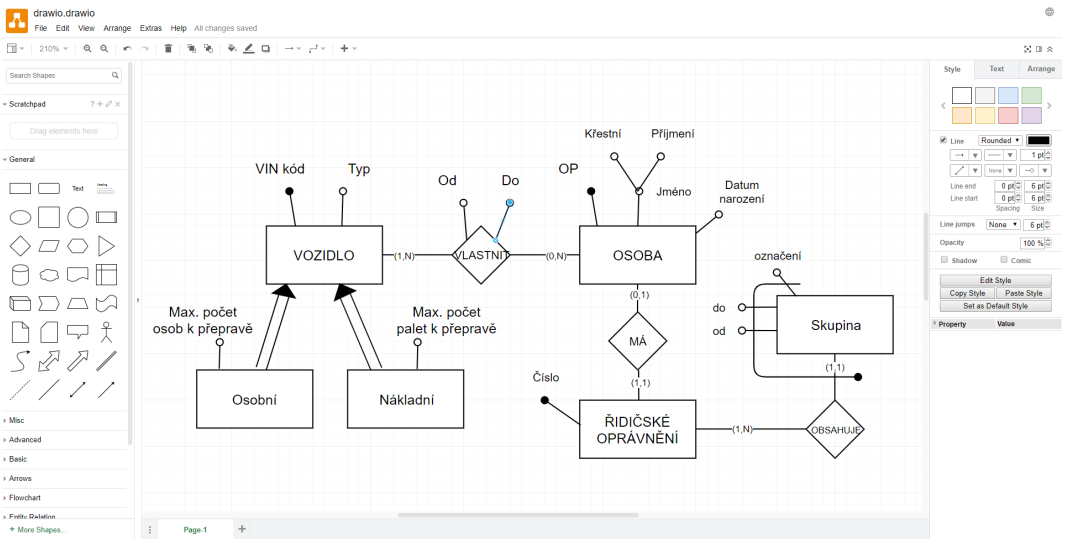
Pozitivně musíme ohodnotit, že vydavatelé nezapomněli na aspoň nějakou formu ISA hierarchii. Dále je možné vytvořit ternární vztahový typ. Bohužel postrádáme možnost naznačit, na čem závisí existence slabého vztahového typu.

Z hlediska funkcionality se s programem pracuje relativně dobře a program obsahuje všechny potřebné funkce. Vytvořený graf můžeme uložit do souboru, exportovat jako obrázek, do PDF nebo SVG.

Zhodnocením na závěr pro malé náčrty diagramů můžeme program využít. Pro pokročilejší diagramy použití programu není příliš vhodné kvůli absenci některých prvků a zároveň vytváření diagramu je celkem pracné. V případě změny není možné lehce předělat – např. u atributů změnit typ. Také jsme limitováni o ukládání projektu a jeho zpětného otevření (jen uložení veřejně v rámci účtu). Dostupná verze opravdu slouží jen pro ukázkové použití.

4.5 Draw.io

Draw.io [17] je dalším zástupcem webové aplikace. Jedná se o bezplatný nástroj pro malování různého charakteru. Široká dostupnost je zaručena díky použití webových technologií a spouštění v prohlížeči. Program disponuje širokou variabilitou pro veškeré objekty. Lze nastavovat barvy, tloušťky, typy čar a jejich zakončení a mnohem víc. Veškeré diagramy z předešlé kapitoly byly tvořeny právě touto aplikací. Uživateli je umožněno ukládání a načítání svých projektů, jak ze svého počítače, tak třeba i z cloudových úložišť – Google Drive, Onedrive, ... Projekt je možné exportovat i do obrázku PNG, JPG, vektorové grafiky SVG nebo jako HTML a mnohé další. Aplikace uživateli nebrání ji používat.



Obrázek 4.3. Ukázka práce s programem Draw.io, diagram znázorňující náš příklad.

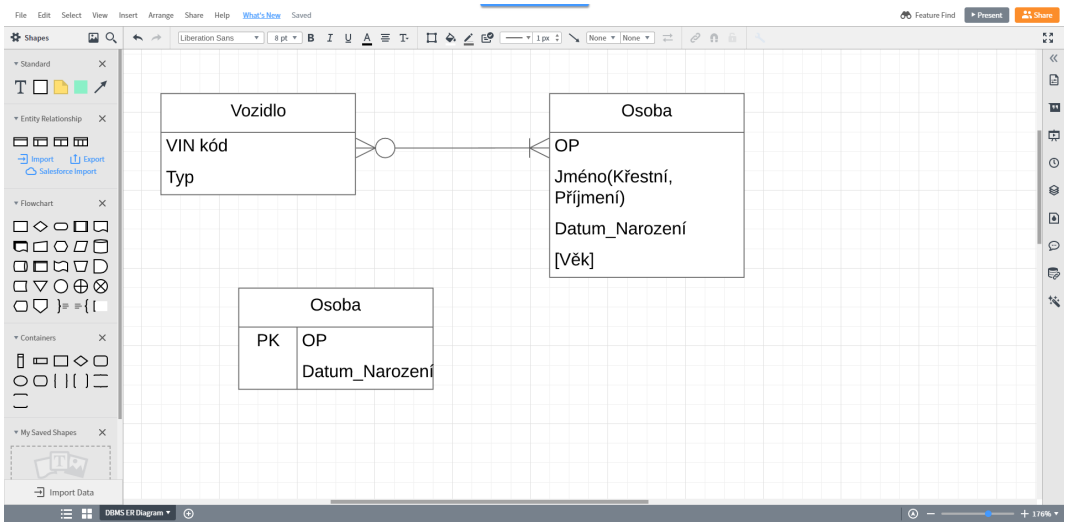
Aplikace obsahuje velkou řadu základních objektů, tak zároveň má knihovnu nazvanou ER. Dostupné objekty v knihovně propojují různé notace. Entity notaci Kortha a Silberschatze, ale jsou dostupné i klasické obdélníkové. Pro vztahy a kardinality je dostupné značení kuří nožky typické pro notaci McFaddena a Hoffera. Dostupné konstrukty nemají možnost použití všech typů atributů, které jsou v původní notaci dostupné. Nejedná se o ucelenou knihovnu odpovídající přesně notaci, proto jejímu použití je lepší se vyhnout. V programu si vyzkoušíme značení podle notace Atzeni, Ceri, Paraboschi a Torlone (viz obr. 4.3). Všechny jejich konstrukty je možné modelovat. Pro atributy můžeme využít čáry s nastavením zakončení. Pro vztahové a entitní typy máme

dostupné geometrické tvary. Isa hierarchii si musíme šikovně vytvořit pomocí trojúhelníku a široké dvojité čáry. S trochou zručnosti lze vytvořit pěkný diagram.

Nástroj určitě označíme za kreslící. Nabízenou podporu pro ER nemá kvalitní (knihovna objektů ER), nicméně jiné prostředky pro vytvoření diagramu je možné najít a kvalitně použít. Na okraj zmíníme, že nástroj dovoluje vytvoření knihovny svých vlastních prvků. Automaticky se nabízí přístup vytvoření svých vlastních předpřipravených prvků pro snadnější použití. Přestože diagram se musí kreslit, jeho vyhotovení je celkem rychlé díky dobře navrženému UI a řadě podpůrných funkcí. V případě, že postrádáme nástroj pro modelování ER, může být tento program náhradou.

4.6 LucidChart

LucidChart [18] je webovým řešením. Nástroj podle prezentace na webu vydavatele dovoluje široké množství tvorby diagramů včetně ER. Jedná se o placenou aplikaci, která umožňuje týdenní bezplatnou verzi na vyzkoušení. Řešení využívá i řada známých společností jako – Google, Adobe, Amazon, ... Ukládání svých projektů v bezplatné verzi pouze v rámci vytvořeného účtu. Diagram lze bez omezení vyexportovat do obrázkových formátů.



Obrázek 4.4. Práce s programem LucidChart. Ukázka entitních typů, vztahu, atributů.

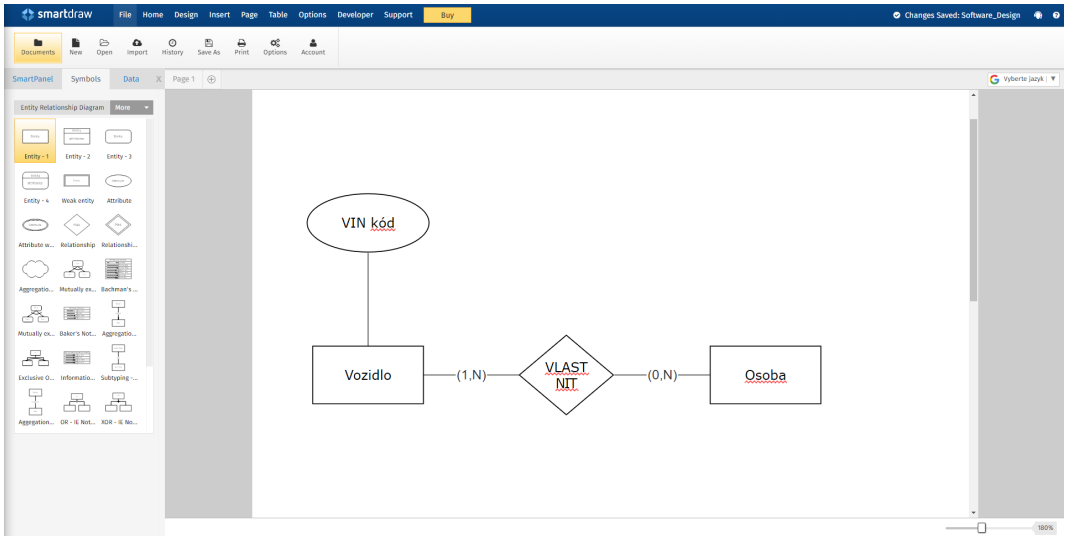
V programu entitní typy jsou zastoupeny obdélníkem rozděleným na dvě částí – název a atributy (viz. obr. 4.4), kde je možné vytvářet seznam atributů. Tuto reprezentaci popisuje notace Kortha a Silberschatze. Formát popisu atributů si uživatel může zvolit, pro atributy můžeme použít i popisu McFaddena a Hoffera.

Nicméně ve značení identifikátorů narazíme na pomyslný problém. Není možné podtrhávat text identifikujících atributů, jak bychom čekali. Pro identifikátory se může použít další sloupeček (viz samostatná entita na obrázku 4.4), do kterého je možné uvést PK (primární klíč), FK (cizí klíč). Práce s klíčí není to, co bychom u modelování na konceptuální vrstvě očekávali. Tímto přístupem začínáme vytvářet pouze relační model. Vztahy jsou tvořeny pomocí značení Crow's Foot (kuří nožka). Program označíme spíše jako kreslící s podporou ER podle představy vydavatele.

Aplikaci více nebudeme zkoumat, protože neodpovídá ER modelování, jak bychom chtěli.

4.7 SmartDraw

SmartDraw [19] je webovým řešením nebo desktopovým řešením pro operační systém Windows. Opětovně se jedná o nástroj s možností vytváření diagramů různých typů. SmartDraw je placená aplikace, umožňují 7 denní zkušební verzi. Nástroj obsahuje šablonu pro ER diagram. Na první pohled aplikace obsahuje různorodé konstrukty – entitní typ, slabý entitní typ, vztahové typy, značení pro ISA hierarchii. Konstrukty však neodpovídají jedné konkrétní notaci. Značení kardinalit vztahů probíhá pomocí textu na spojnici. Nástroj můžeme označit jako modelovací. Jednoduchou ukázku můžeme vidět na obrázku 4.5.



Obrázek 4.5. Práce s programem SmartDraw. Ukázka entitních typů, vztahu, atributu.

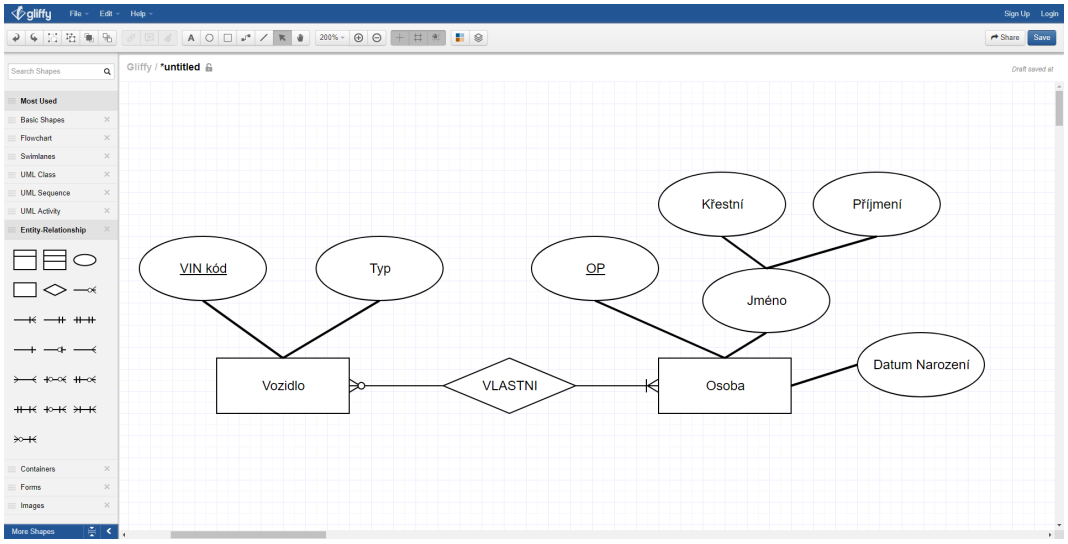
Způsob vytváření diagramů není zcela uživatelsky pohodlný. Objekty je možné umístit na přesně určené pozice v mřížce, problematické je i spojení objektů čarou. Ačkoliv aplikace je po technické stránce schopna vytvořit ER diagram, ovládání aplikace uživatele přesvědčuje, že SmartDraw není ideální volbou.

4.8 Gliffy

Zástupcem webového řešení je i nástroj Gliffy [20]. Jedná se o placenou aplikaci s možností použití omezené bezplatné verze určené pro vyzkoušení. Nástroj zahrnuje podporu pro modelování ER. Opět se jedná o kombinaci konstruktů značených podle různých notací. Mezi dostupnými objekty jsou entitní typ (více provedení vzhledu), atribut, vztahový typ a spojení různých kardinalit. Můžeme vidět na postranním panelu na obrázku 4.6. Pokud využíváme připravené konstrukty pro ER, nástroj se chová spíše jako modelovací. S programem se pracuje snadno, při ovládání pomáhají podpůrné funkce, které uživateli usnadňují vytváření diagramu.

Z hlediska nabízených konstruktů je nabídka užší než bychom si přáli – postrádáme ISA hierarchii a modelování slabého entitního typu. Slabý entitní typ by bylo možné pomocí dalších jednoduchých objektů doplnit, ale poté už pouze nemodelujeme, ale také kreslíme.

Gliffy můžeme zhodnotit jako jednoduchý, téměř modelovací nástroj. Překážkou je omezená nabídka konstruktů, která neumožňuje vytváření složitějších schémat.

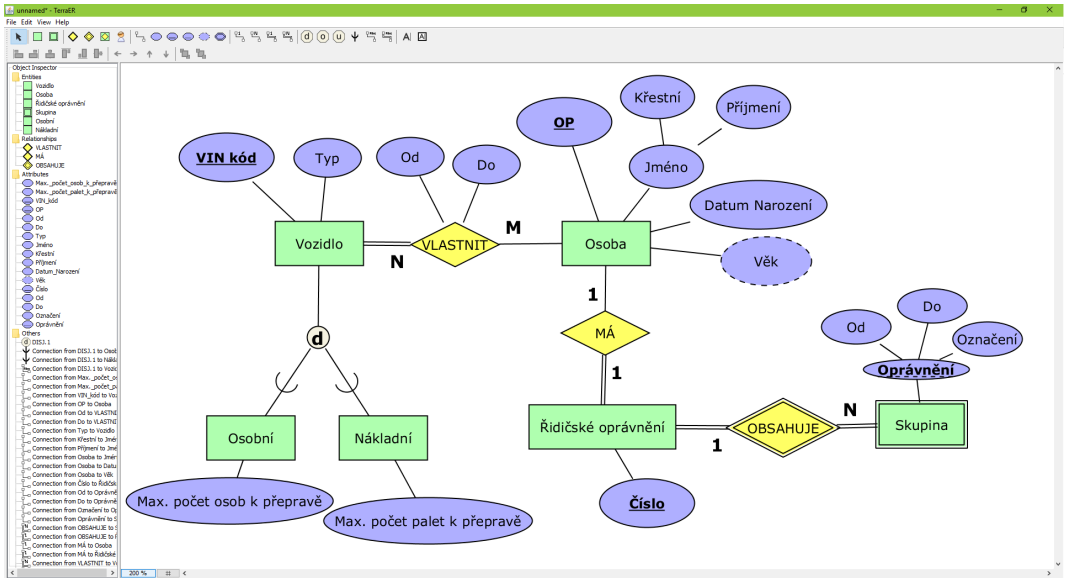


Obrázek 4.6. Práce s programem Gliffy. Ukázka entitních typů, vztahu, atributů.

4.9 Terra ER

Jako poslední ukázku si uvedeme Terra ER [21]. Terra ER je akademický nástroj pro modelování ER diagramů, zejména používaný na univerzitách v Brazílii dle stránek vydavatele. Program je volně dostupný. Jedná se o aplikaci v jazyce JAVA [22] distribuovanou jako spustitelný soubor JAR, možné použití na počítačích s nainstalovaným JVM (Java Virtual Machine) [23]. Námi testovaná verze programu je 3.13.

Aplikace je vytvořena účelně viz. obrázek 4.7. V horní části se nachází panel s konstrukty a nástroje pro posun a zarovnání objektů. V levé části panel obsahující seznam objektů, v hlavní části se nachází plátno, kde probíhá vlastní tvorba diagramu. Nástroj můžeme bez váhání označit za modelovací.



Obrázek 4.7. Práce s programem Terra ER. Ukázka modelování našeho příkladu.

Používané vykreslování objektů odpovídá notaci Elsmari a Navathe a jsou dostupné veškeré konstrukty této notace. Práce s programem je rychlá a bezproblémová. Ukládání

a načítání projektu je možné ve formátu xml. Exportovat diagram do obrázku je možné pouze ve formátu PNG.

Některé chování by mohly být vylepšeno – u kopírování objektu, nový objekt překrývá původní nebo při pozicování objektu by mohlo být naznačeno zarovnávání s ostatními objekty. Tyto nedostatky nijak nebrání v používání. Terra ER splňuje očekávání na dostatečný modelovací ER nástroj.

4.10 Závěr

V této kapitole jsme se seznámili s nejdostupnějšími nástroji, které je možné využít pro modelování ER diagramu. Většina uvedených nástrojů deklaruje možnost tvorby ER diagramů, bohužel následně jsme odhalili, že neobsahují všechny dostupné prostředky, které nám dnes ER dává.

Jako nejpovedenější aplikaci můžeme hodnotit Terra ER, která poskytuje kompletní sadu konstruktů v rámci používané notace a je úplným modelovacím nástrojem. Dalším povedeným nástrojem byl ERDPlus, který obsahoval nedostatky v konstruktech, nicméně modelování s ním bylo možné. Jako třetí nejlepší uvedeme Draw.io, které umožňuje zakreslit vše možné. Námi testovanou notaci bylo celkem snadné vytvořit, i když při vytváření jsme se už soustředili na grafickou prezentaci místo pouze tvorby vlastního schématu.

Kapitola 5
Specifikace

V předchozí kapitole jsme se zabývali analýzou dostupných řešení pro vytváření ER diagramů. Nabídka dostupných nástrojů není malá, ale při detailnějším zkoumání jsme zjistili, že většina nástrojů neodpovídá nárokům, které pro tento typ nástroje očekáváme.

V případě modelovacího nástroje **TerraER**, který byl vyhodnocen prozatím jako nejvhodnější, byla zvolena ER notace **Elsmari a Navathe**, která podporuje téměř všechny konstrukty, nicméně notace **Atzeni, Ceri, Paraboschi a Torlone** nám umožňuje vyjádřit o trochu více. Podporované konstrukty si můžeme připomenout z provedeného porovnání na obr. 3.10 z kapitoly 3. Nástroj podporující tuto notaci jsme při naši analýze neobjevili.

Notaci **Atzeni, Ceri, Paraboschi a Torlone** zmiňujeme záměrně. V drobně graficky pozměněné podobě se používá pro výuku konceptuálního modelování v rámci předmětu Databázové systémy na Fakultě elektrotechnické pod Českým vysokým učením technickým v Praze. Pro potřeby výuky však neexistuje vhodný nástroj pro digitální záznam ER schémat a tvorbu ER diagramů. Našim cílem bude pokusit se vyvinout aplikaci, která by umožnila digitální tvorbu ER schéma a vytvářela diagram v notaci používané v rámci výuky.

Jako cílovou skupinou aplikace budou studenti a vyučující tohoto předmětu. Aplikace bude implementovat notaci používanou v rámci výuky, jejíž grafickou reprezentaci zmíníme v rámci požadavků na aplikaci. V následující části si uvedeme specifikaci na tuto aplikaci obsahující požadavky, případy užití a předběžnou představu datového modelu.

5.1 Požadavky

Sběr požadavků [24] je jedna z disciplín softwarového inženýrství. Požadavky nám popisují očekávání od systému. Zpravidla se vytváří mezi zákazníkem a dodavatelem. Slouží jako podklad pro hlubší analýzu a vývoj aplikace, ale také pro vyhodnocení, zda je splněno zadání při předání vypracovaného díla. Požadavky tvoří tzv. katalog požadavků. Jednotlivé požadavky se skládají z názvu, popisu, očíslování – označeny identifikátorem, aby bylo možné se na ně odvolávat. Dále můžeme evidovat, kdo požadavek vznesl, jeho prioritu a další informace. Jeden požadavek by měl vytvářet jedno splnitelné „očekávání“.

Nyní si uvedeme požadavky na naši aplikaci, které rozdělíme do dvou skupin – funkční a nefunkční.

5.1.1 Funkční požadavky

Funkční požadavky popisují, co aplikace bude uživatelům umožňovat – akce, úkony, aktivity. Funkční požadavky se používají jako podklad pro vytváření případů užití (use case). V našem případě je budeme označovat ve formátu FP-X, kde X je číslo požadavku.

Editor

FP-1 **Ukládání** – Aktuální projekt bude možné uložit do souboru.

FP-2 **Načítání** – Systém umožní načíst ze souboru projekt vytvořený tímto systémem.

FP-3 **Export do obrázku** – Systém umožní vyexportovat diagram do obrázku. Před exportem systém umožní nastavit okraj, který se doplní okolo diagramu v obrázkovém výstupu.

FP-4 **Nastavení velikosti konstruktů** – Systém umožní měnit výchozí nastavení konstant velikosti konstruktů používaných při vytvářených nových prvků.

FP-5 **Lišta nástrojů** – Systém bude obsahovat lištu nástrojů pro práci a nastavení editoru, pro obsluhu plátna a prvků v plátně.

FP-6 **Plátno** – Systém bude obsahovat plátno, které bude zobrazovat diagram.

FP-7 **Paleta konstruktů** – Systém bude obsahovat paletu konstruktů, ze které bude možné přidávat konstrukty do plátna.

FP-8 **Panel nastavení konstruktu** – Systém bude obsahovat nastavení, ve kterém bude možné měnit vlastnosti konstruktu.

FP-9 **Obnovení výchozího nastavení** – Systém umožní resetovat hodnoty ukládané v nastavení ovlivňující chování editoru do výchozích hodnot.

FP-10 **Náhled na diagram** – Systém umožní zobrazit okno, které zobrazí náhled diagramu a umožní navigaci po rozsáhlejším diagramu.

FP-11 **Nápověda** – Systém umožní zobrazit okno s nápovědou obsahující základní popis aplikace a její funkčnosti, popis ovládání.

FP-12 **Status lišta** – Systém bude obsahovat lištu pro výpis stavových informací aplikace.

FP-13 **Pozadí plátna** – Systém umožní vypnout pozadí plátna do bílé barvy.

FP-14 **Historie úprav** – Systém umožní pracovat s historií, vracet provedené úpravy, nebo opět provést vrácenou úpravu.

Operace s plátnem

FP-15 **Posouvání se po diagramu** – Systém umožní posouvat se po diagramu stisknutím pravého tlačítka a posunu myši.

FP-16 **Přiblížení** – Systém bude umožňovat pracovat s přiblížením – funkce přiblížit a oddálit.

FP-17 **Přiblížení kolečkem myši** – Systém umožní ovládat přiblížení diagramu pomocí pohybu kolečka.

Operace s prvky plátna

FP-18 **Označení** – Systém umožní označit prvek v plátně pro další operace.

FP-19 **Posouvání** – Systém umožní posouvat prvky po plátně.

FP-20 **Posouvání, pomocné zarovnání** – Systém při posouvání prvku zobrazí pomocné čáry k ostatním prvkům pro možné zarovnávání.

FP-21 **Zarovnání** – Systém umožní zarovnat vybrané prvky plátna podle osy.

FP-22 **Změna velikosti** – Systém umožní měnit velikosti prvků v plátně.

FP-23 **Posun v hloubce plátna** – Systém umožní posouvat prvky v hladině do popředí a do pozadí.

FP-24 **Kopírování a vložení** – Systém umožní kopírovat vybrané prvky a následně vkládat do diagramu.

FP-25 **Mazání** – Vybrané prvky umožní systém smazat.

FP-26 **Mazání klávesovou zkratkou** – Systém umožní smazat vybraný prvek i pomocí klávesy `delete`.

FP-27 **Kopírování a vložení, zkratky** – Kopírování a vložení bude možné provést i přes klávesové zkratky `ctrl+c`, `ctrl+v`.

FP-28 **Editace popisků přímo v plátně** – Systém umožní provádět změny názvu dvojitým kliknutím na popisek obsahující název.

FP-29 **Mřížka** – Systém umožní nastavit posouvání prvků po mřížce s volbou velikosti kroku. Prvky diagramu pak bude možné posunout do bodů určené mřížkou.

Operace s prvky plátna přestavující konstrukty

FP-30 **Kopírování smysluplných celků** – Systém bude provádět kopírování pouze logických celků diagramu, tak aby nedocházelo k nevalidnímu schématu.

FP-31 **Konzistence při mazání** – Při mazání vybraného prvku nebo vybraných prvků, systém zajistí, že se smažou i další nutné části schéma tak, aby existovalo konzistentní schéma.

FP-32 **Zarovnání popisků** – Systém umožní u vybraných konstruktů (kde dává smysl) upravit pozici popisku.

FP-33 **Editace vlastností konstruktu** – Systém po označení prvku v plátně umožní editaci vlastností konstruktu v panelu nastavení konstruktu, v případě, že prvek má nějaké informace k editaci. Tyto změny se projeví do diagramu.

FP-34 **Úprava vedení čar vztahů** – Systém umožní měnit vedení čar mezi entitním typem a vztahovým typem.

FP-35 **Navigační body čar vztahů** – Systém umožní přidávat a odebírat navigační body pro vedení čáry mezi entitním typem a vztahovým typem.

Požadavky na práci s ER diagramem, ER schématem

FP-36 **Vytvoření entitního typu** – Systém umožní vytvoření entitního typu.

FP-37 **Vytvoření jednoduchého atributu** – Systém umožní pro konkrétní entitní nebo vztahový typ přiřadit atribut.

FP-38 **Vytvoření jednoatributového identifikátoru** – Systém umožní pro konkrétní entitní typ přiřadit jednoatributový identifikátor.

FP-39 **Vytvoření vícehodnotového atributu** – Systém umožní pro konkrétní entitní nebo vztahový typ přiřadit vícehodnotový atribut.

FP-40 **Vytvoření víceatributového identifikátoru** – Systém umožní pro konkrétní entitní typ vytvořit vícehodnotový identifikátor.

FP-41 **Vytvoření složeného atributu** – Systém umožní vytvoření složeného atributu pro konkrétní entitní nebo vztahový typ.

FP-42 **Vytvoření rekurzivního vztahového typu** – Systém umožní pro konkrétní entitní typ vytvořit rekurzivní vztahový typ.

FP-43 **Vytvoření binární vztahového typu** – Systém umožní pro dva konkrétní entitní typy vytvořit binární vztahový typ.

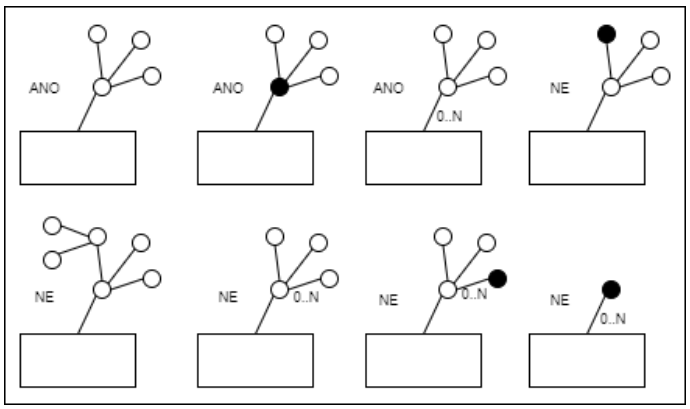
FP-44 **Vytvoření ternárního vztahového typu** – Systém umožní pro tři konkrétní entitní typy vytvořit ternární vztahový typ.

FP-45 **Vytvoření hierarchie** – Systém umožní vytvořit mezi konkrétními entitními typy hierarchii, kde jeden entitní typ je v roli nadtřídy a jeden nebo více entitních typů v roli podtřídy.

FP-46 **Vytvoření slabého entitního typu** – Systém umožní zaznamenat slabý entitní typ, umožní vytvořit složený identifikátor s účastí jednoho a více atributů

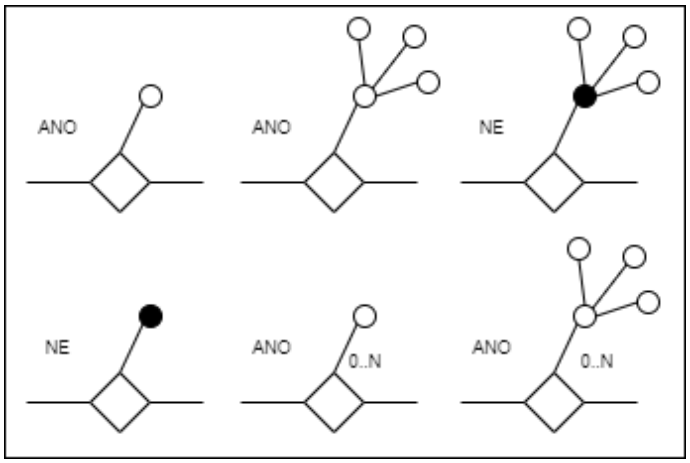
a jednoho a více vztahový typů s kardinalitou `1..1` ve směru od slabého entitního typu ke vztahovému typu.

FP-47 **Kombinace druhů atributů nad entitním typem** – Systém umožní či zabrání vytvoření kombinací atributů nad entitním typem. Umožní vytvoření složeného atributu, složeného atributu jako identifikátor, složeného atributu jako vícehodnotový. Zabrání situacím – atribut součástí složeného atributu jako identifikátorem, skládání složených atributů do sebe, vícehodonotový atribut součástí složeného atributu, atribut součástí složeného atributu jako vícehodnotový a identifikátor, vícehodnotový atribut jako identifikátor. Zmíněné situace dle uvedeného pořadí shrnuje obrázek 5.1.



Obrázek 5.1. Přípustné případy a kombinace atributů pro entitní typ

FP-48 **Kombinace druhů atributů ve složeném atributu nad vztahovým typem** – Systém umožní či zabraní vytvoření kombinací atributů nad vztahovým typem. Umožní přiřadit jednoduchý atribut, složený atribut. Zamezí vytvoření složeného atributu jako identifikátor, jednoduchému atributu jako identifikátor. Umožní přiřadit vícehodnotový atribut a složený atribut jako vícehodnotový. Zmíněné situace dle uvedeného pořadí znázorňuje obrázek 5.2.



Obrázek 5.2. Přípustné případy a kombinace atributů pro vztahový typ

FP-49 **Editace názvu konstruktu** – Systém umožní nastavit pojmenování entitního typ, vztahového typu, atributu.

FP-50 **Editace pojmenování účasti ve vztahu** – Systém umožní pojmenovat účast entitního typu ve vztahu.

FP-51 **Editace omezení vícehodnotového atributu** – Systém umožní nastavit hranice omezení pro vícehodnotový atribut. Pro dolní mez hodnoty – `0`, `1`. Pro horní mez hodnoty – `1`, `N`.

FP-52 **Editace kardinality vztahového typu** – Systém umožní měnit nastavení kardinalit vztahového typu. Pro dolní mez hodnoty – `0`, `1`. Pro horní mez hodnoty – `1`, `N`.

FP-53 **Editace pokrytí hierarchie** – Systém umožní měnit nastavení pokrytí pro hierarchii z výběru hodnot – úplné, částečné.

FP-54 **Editace překrytí hierarchie** – Systém umožní měnit nastavení překrytí pro hierarchii z výběru hodnot – exkluzivní, překrývající.

5.1.2 Nefunkční požadavky

Nefunkční požadavky uvádí nároky na aplikaci, které se netýkají přímo funkčnosti systému pro uživatele, ale zabývají se dalšími aspekty aplikace, jako je výkon, spolehlivost, zabezpečení, rozšiřitelnost, nároky na použité technologie, dodržení designu a další. Nefunkční požadavky budeme označovat ve formátu NP-X, kde X je číslo požadavku.

NP-1 **Podpora uživatelů** – Systém bude možné používat na různých operačních systémech – Windows[1], Linux[2].

NP-2 **Formát souboru projektu** – Systém bude ukládat diagram do čitelného souboru ve formátu `.xml`.

NP-3 **Formát exportovaného obrázku** – Systém bude exportovat obrázek do formátu `.png`.

NP-4 **Automatické ukládání práce** – Aplikace bude ukládat průběžně aktuální stav rozdělaného projektu, tak aby nedocházelo ke ztrátě rozdělané práce v případě pádu aplikace.

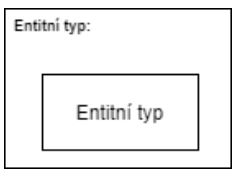
NP-5 **Automatické ukládání nastavení** – Systém při úpravě nastavení toto nastavení uloží pro příští použití aplikace.

NP-6 **Rozšiřitelnost** – Aplikace by měla být koncipována tak, aby bylo možné poskytované funkcionality bez komplikací rozšířit.

5.1.3 Požadavky na podobu notace

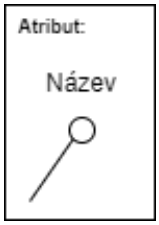
Jako samostatnou část si uvedeme požadavky na podobu notace, kterou naše aplikace bude implementovat. Označíme ve formátu DP-X, kde X je číslo požadavku.

DP-1 **Entitní typ vzhled** – Aplikace bude graficky reprezentovat entitní typ dle obrázku 5.3. Obdélník s názvem entitního typu uprostřed.



Obrázek 5.3. Grafická reprezentace entitního typu

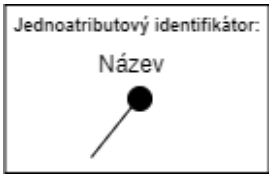
[1] https://www.microsoft.com/en-us/windows/
[2] https://help.ubuntu.com/



Obrázek 5.4. Grafická reprezentace atributu

DP-2 **Jednoduchý atribut vzhled** – Aplikace bude graficky reprezentovat atribut dle obrázku 5.4. Kolečko s uvedeným názvem atributu v jeho blízkosti, které je spojeno s entitním či vztahový typem čárou.

DP-3 **Jednoatributový identifikátor vzhled** – Aplikace bude graficky reprezentovat jednoatributový identifikátor dle obrázku 5.5. Vzhled je shodný s jednoduchým atributem, kolečko je vyplněno barvou.

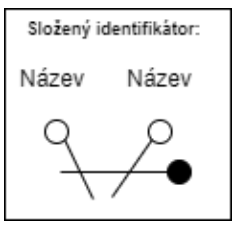


Obrázek 5.5. Grafická reprezentace jednoatributového identifikátoru

DP-4 **Vícehodnotový atribut vzhled** – Aplikace bude graficky reprezentovat vícehodnotový atribut dle obrázku 5.6. Vzhled je shodný s jednoduchým atributem, v oblasti konce čáry směrem k entitnímu nebo vztahovému typu je uvedena kardinalita.

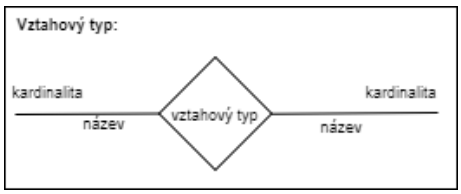
Obrázek 5.6. Grafická reprezentace vícehodnotového atributu

DP-5 **Víceatributový identifikátor vzhled** – Aplikace bude graficky reprezentovat víceatributový identifikátor dle obrázku 5.7. Bude reprezentovat jako čáru, která kříží účastnící se atributy, na jednom z konců čáry se nachází vyplněné kolečko.



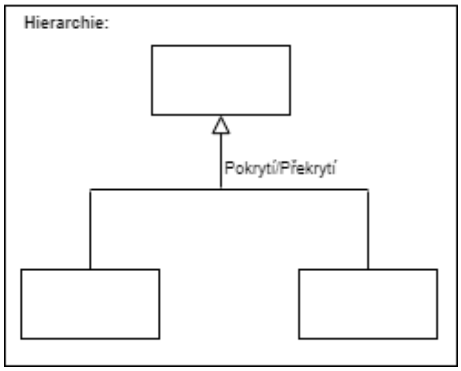
Obrázek 5.7. Grafická reprezentace víceatributového identifikátoru

DP-6 **Vztahový typ vzhled** – Aplikace bude graficky reprezentovat vztahový typ dle obrázku 5.8. Vztahový typ bude reprezentován kosodélníkem, název vztahového typu uveden uvnitř. K entitním typům povedou čáry. V oblasti středu čáry se bude nacházet místo pro popisek pro pojmenování účasti. V oblasti konce čáry směrem k entitnímu typu se bude nacházet popisek s kardinalitou.



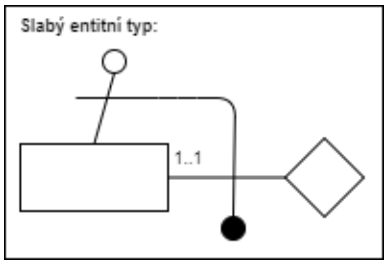
Obrázek 5.8. Grafická reprezentace vztahového typu

DP-7 **Hierarchie vzhled** – Aplikace bude graficky reprezentovat hierarchii dle obrázku 5.9. Od entitních typů představující podtřídy hierarchie povedou čáry do jednoho bodu, od tohoto bodu povede čára s šipkou k entitnímu typu představující nadtřídu. V oblasti styku čar bude umístěn popisek, zobrazují informaci o typu hierarchie - pokrytí a překrytí.



Obrázek 5.9. Grafická reprezentace hierarchie

DP-8 **Slabý entitní typ vzhled** – Aplikace bude graficky reprezentovat slabý entitní typ dle obrázku 5.10. Existence slabého typu bude znázorněna čárou vedoucí přes atributy a čáry vztahu od vztahového typu, které tvoří identifikátor pro slabý entitní typ. Na jednom z konců čáry bude vyplněné kolečko.



Obrázek 5.10. Grafická reprezentace slabého entitního typu

DP-9 **Formát výpisu kardinality** – Kardinalita bude v diagramu uváděna ve formátu `min..max`, např. `0..1`.

DP-10 **Jazyk prostředí** – Prostředí systému bude lokalizováno do anglického jazyka.

5.2 Možné způsoby řešení aplikace

Na základě nefunkčního požadavku NP-1 na platformovou nezávislost nyní zvolíme typ řešení pro naši aplikaci. Na zvážení se nabízejí dvě možná řešení.

První možností je vytvořit JAVA [22] aplikaci pro desktop. Pokud má uživatel nainstalovaný JVM (Java Virtual Machine) [23], což je virtuální stroj pro spouštění JAVA programů, tak máme zajištěno, že uživatel může aplikaci spustit. Instalace JVM je možná pro operační systémy Windows i Linux.

Druhým řešení je vytvoření webové aplikace. Zde nemusíme řešit instalaci speciálních programů, protože téměř vždy je uživatel vybaven internetovým prohlížečem. Uživatel zobrazí aplikaci ve svém prohlížeči jednoduše přes URL (Uniform Resource Locator) [25], kde je aplikace hostována. Drobnější komplikací, na kterou se pak vývojář musí zaměřit, je podpora různých druhů prohlížečů. V našem případě bychom se zaměřili na podporu Google Chrome[1] a Mozilla Firefox[2], které patří mezi nejpoužívanější internetové prohlížeče.

Pro naši aplikaci jako řešení zvolíme webovou aplikaci. Webové aplikace jsou v současné době moderní a nekladou na uživatele nárok na instalaci různých programů do svého počítače. Postačí používat vydavatelem aplikace podporovaný prohlížeč a mít přístup k internetu.

Následující části analýzy budou předpokládat již zvolené řešení aplikace.

5.3 Případy užití

Model případů užití zachycuje funkcionality systému a práva přístupu uživatelů systému k těmto funkcionalitám. Případy užití se prezentují pomocí diagramu případů užití (use case diagram), který zobrazuje hranice systému, jeho funkcionality, uživatele v rolích. Pomocí vazeb mezi rolí a funkcionalitou je vyjádřeno, že účastník má k funkcionalitě přístup. V diagramu se může objevit i vazba mezi jednotlivými případy užití. Existují dva případy – vazba `include` (jeden případ užití zahrnuje jiný) a `extend` (jeden případ užití rozšiřuje chování jiného).

Každý případ užití by měl reprezentovat triviální akci jednoho účastníka. Případ užití může být hlouběji specifikován názvem, stručným popisem a popisem vnitřního běhu. Tento popis nazýváme scénář, který popisuje, jak systém funguje spolu s uživatelem a kromě hlavního scénáře může popisovat i různé alternativy.

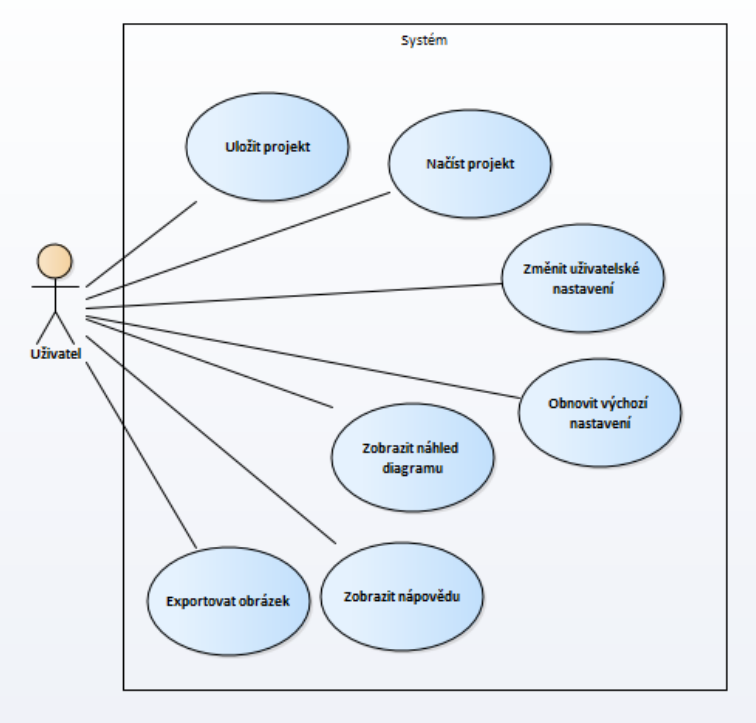
Případy užití zachycují pouze funkční požadavky. Modelovaní těchto případů užití je vhodné pro rozpracování detailu jednotlivých funkcionalit a získání hlubší představy o tom, jak má systém fungovat. Dále usnadňují odhad pracnosti řešení a lze je použít jako poklad pro testování.

Případy užití pro naši aplikaci jsme rozdělili do tří skupin a uvedené funkcionality doplnili scénářem. Scénář obsahuje kroky a u některých je uveden i vstup, což je požadavek na stav systému před zahájením scénáře.

5.3.1 Editor

Případy užití editoru (nazveme tak část aplikace, která vytváří prostředí okolo plátna) nalezneme na obrázku 5.11. Detailní popis jednotlivých scénářů následuje.

[1] Google Chome – https://www.google.com/intl/cs/chrome/
[2] Mozilla Firefox – https://www.mozilla.org/en-US/firefox/new/



Obrázek 5.11. Diagram případů užití editoru.

Uložit projekt

Kroky:

1. Uživatel v menu editoru zvolí funkci uložit.
2. Systém zobrazí okno obsahující textový vstup pro zadání názvu souboru s výchozí hodnotou ER, tlačítko pro provedení akce uložení.
3. Uživatel vyplní název souboru a stiskne tlačítko uložit.
4. Systém provede vytvoření souboru a vyvolá stažení souboru.

Načíst projekt

Kroky:

1. Uživatel v menu editoru zvolí funkci načíst.
2. Systém zobrazí okno obsahující vstup pro výběr souboru na disku uživatele a tlačítko pro provedení akce načtení.
3. Uživatel vybere soubor na disku a stiskne tlačítko načíst.
4. Systém zobrazí okno s žádostí o potvrzení, zda provést akci s upozorněním, že hrozí ztráta neuložených změn.
5. Uživatel potvrdí, že systém má provést načtení.
6. Systém načte data o modelu ze souboru.

Změnit uživatelské nastavení

Kroky:

1. Uživatel v menu editoru zvolí funkci nastavení.
2. Systém zobrazí okno obsahující vstupy pro úpravu jednotlivých vlastností s aktuálními hodnotami a tlačítko pro obnovení nastavení.
3. Uživatel provede změnu hodnoty u některého vstupu.

4. Systém při opuštění prvku vstupu provede změnu nastavení. Pro textový vstup provede uložení i po stisknutí klávesy `enter`.

Obnovit výchozí nastavení

Kroky:

1. Uživatel v menu editoru zvolí funkci nastavení.
2. Systém zobrazí okno obsahující vstupy pro úpravu nastavení jednotlivých vlastností s aktuálními hodnotami a tlačítko pro obnovení nastavení.
3. Uživatel stiskne tlačítko obnovit nastavení.
4. Systém se dotáže, zda opravdu provést obnovu nastavení.
5. Uživatel potvrdí, nebo zruší.
6. V případě potvrzení systém obnoví nastavení do výchozích hodnot.

Exportovat obrázek

Kroky:

1. Uživatel v menu editoru zvolí funkci exportovat.
2. Systém zobrazí okno zobrazující vstup pro zadání měřítka rozlišení fotky vůči diagramu, vstup pro velikost okraje navíc, tlačítko pro provedení exportu do formátu png.
3. Uživatel vyplní údaje a provede akci exportovat do png.
4. Systém provede převod diagramu do png a vyvolá stažení souboru s výchozím názvem ER.

Zobrazit náhled diagramu

Kroky:

1. Uživatel v menu editoru zvolí funkci náhled.
2. Systém zobrazí okno zobrazující celý diagram s vyznačením aktuální zobrazené části diagramu.

Zobrazit náhled diagramu

Kroky:

1. Uživatel v menu editoru zvolí funkci náhled.
2. Systém zobrazí okno zobrazující celý diagram s vyznačením aktuální zobrazené části diagramu.

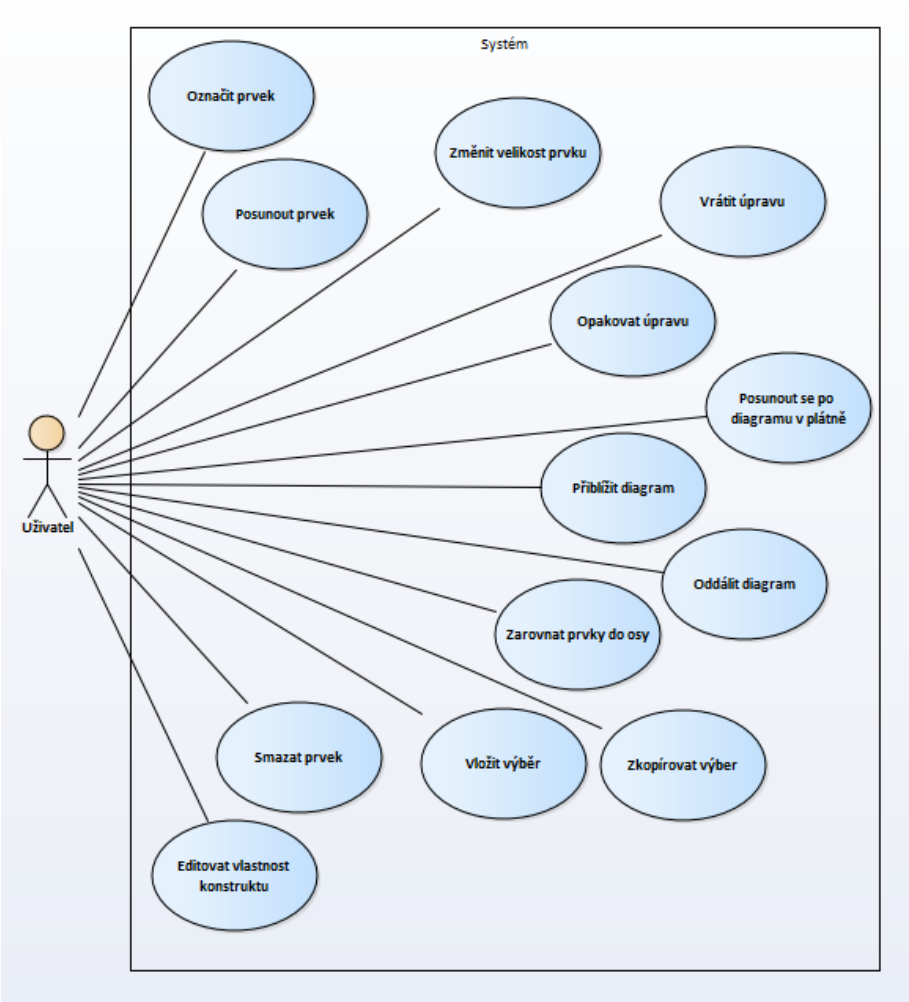
Zobrazit nápovědu

Kroky:

1. Uživatel v menu editoru zvolí funkci nápověda.
2. Systém zobrazí okno zobrazující stránku nápovědy aplikace.

5.3.2 Interakce s diagramem

Případy užití popisující práci s plátnem a diagramem v něm nalezneme na obrázku 5.12. Popis jednotlivých scénářů následuje.



Obrázek 5.12. Diagram případů užití – interakce s plátnem / diagramem.

Označit prvek

Kroky:

1. Uživatel klikne na prvek diagramu.
2. Systém zvýrazní označený prvek.

Posunout prvek

Vstup: Označený prvek nebo prvky.
Kroky:

1. Uživatel stiskne levé tlačítko v místě označeného prvku, drží levé tlačítko stisknuté a posun vytváří pohybem myši.
2. Systém vykresluje a naznačuje posun vybrané oblasti, při uvolnění tlačítka provede posun na pozici.

Změnit velikost prvku

Vstup: Označený prvek.
Kroky:

1. Uživatel namíří myš do kraje prvku a stiskne levé tlačítko myši, drží tlačítko stisknuté a posunem myši mění velikost prvku.

2. Systém naznačuje výsledek změny, při uvolnění tlačítka provede změnu.

Vrátit úpravu

Kroky:

1. Uživatel klikne na tlačítko vrátit úpravu.
2. Systém provede návrat v historii změn, pokud nějaká úprava předchází.

Opakovat úpravu

Kroky:

1. Uživatel klikne na tlačítko opakovat úpravu.
2. Systém provede posun v historii změn, pokud nějaká úprava následuje.

Posunout se po diagramu v plátně

Kroky:

1. Uživatel klikne pravým tlačítkem myši v oblasti diagramu a drží stisknutou klávesu, posunem myši vytváří směr posunu.
2. Systém překresluje diagram do plátna podle změny pozice.

Přiblížit diagram

Kroky:

1. Uživatel klikne na tlačítko přiblížení.
2. Systém zvětší přiblížení a vykreslí diagram.

Oddálit diagram

Kroky:

1. Uživatel klikne na tlačítko oddálit.
2. Systém oddálí přiblížení a vykreslí diagram.

Zarovnat prvek podle osy

Vstup: Označené prvky.
Kroky:

1. Uživatel klikne na ikonu představující požadující typ zarovnání podle osy.
2. Systém zarovná prvky dle vybrané osy.

Zkopírovat výběr

Vstup: Označený prvek nebo prvky.
Kroky:

1. Uživatel stiskne zkratku `ctrl + c`.
2. Systém si uloží označené prvky do paměti. Uloží si jen celky z výběru diagramu, které tvoří mezi sebou validní schéma.

Vložit výběr

Vstup: Uložený výběr v paměti.
Kroky:

1. Uživatel stiskne zkratku `ctrl + v`.

2. Systém přidá do diagramu výběr uložený v paměti s určitým posunem, aby se přidaný výběr nepřekrýval s původními prvky.

Smazat prvek

Vstup: Označený prvek nebo prvky.
Kroky:

1. Uživatel stiskne klávesu `delete`.
2. Systém odstraní vybrané prvky. Zároveň odstraní všechny části diagramu tak, aby existovalo validní schéma a diagram ho správně promítal.

Editovat vlastnosti konstruktu

Vstup: Označený konstrukt, konstrukt má vlastnosti k editaci.
Kroky:

1. Při změně výběru konstruktu systém vykreslí do panelu nastavení konstruktu menu pro editaci vlastností. Pro entitní typ – změna názvu, pro vztahový typ – změna názvu, pro čáru účasti ve vztahu – nastavení kardinalit, pro atribut – změna názvu, nastavení vícehodnotového atributu, zarovnání popisku, pro hierarchii - nastavení pokrytí a překrytí, zarovnání popisku.
2. Uživatel upraví vlastnost pro daný konstrukt. Pro vstup výběru změní vybranou hodnotu, pro textový vstup změní hodnotu a opustí prvek nebo stiskne klávesu `enter`.
3. Systém uloží změněnou hodnotu pro daný prvek a vyvolá překreslení diagramu, aby zobrazoval aktuální změněnou hodnotu.

5.3.3 Vytváření nových konstruktů

Případy užití týkající se vytváření nových konstruktů nalezneme na obrázku 5.13. Popis scénářů následuje.

Vytvořit entitní typ

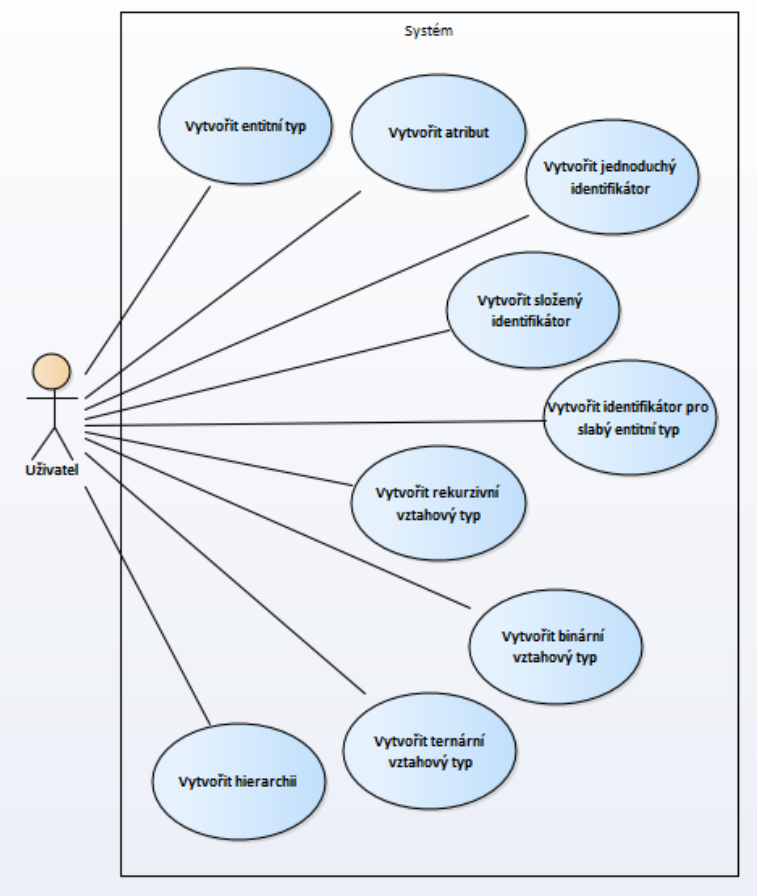
Kroky:

1. Uživatel klikne na ikonu v paletě konstruktů představující entitní typ a táhne do požadovaného místa v diagramu.
2. Systém v požadovaném místě vytvoří entitní typ.

Vytvořit atribut

Kroky:

1. Uživatel klikne na ikonu v paletě konstruktů představující atribut, táhne a pustí nad prvkem, kterému lze atribut přidat – entitní typ, vztahový typ, jiný atribut (kořen složeného atributu nebo atribut, který může vytvořit složený atribut)
2. Systém v určeném místě od vybraného prvku podle nastavených konstant vytvoří atribut.



Obrázek 5.13. Diagram případů užití – vytváření konstruktů.

Vytvořit jednoduchý identifikátor

Vstup: Označený prvek představující atribut, který se může stát identifikátorem, nebo je jednoduchým identifikátorem.

Kroky:

1. Uživatel klikne na ikonu pro změnu mezi atributem a jednoduchým identifikátorem.
2. Systém vybraný atribut změní na identifikátor, nebo jednoduchý identifikátor změní na jednoduchý atribut.

Vytvořit složený identifikátor

Vstup: Označené prvky představující atributy, které mohou tvořit složený identifikátor.

Kroky:

1. Uživatel klikne na ikonu pro vytvoření složeného identifikátoru.
2. Systém vytvoří složený identifikátor ze zvolených atributů.

Vytvořit identifikátor pro slabý entitní typ

Vstup: Označené prvky představující atributy a vztahy, které mohou tvořit složený identifikátor pro slabý entitní typ. Nejméně jeden atribut a jeden vztah s kardinalitou `1..1`

Kroky:

1. Uživatel klikne na ikonu pro vytvoření složeného identifikátoru pro slabý entitní typ.
2. Systém vytvoří identifikátor ze zvolených atributů a vztahů.

Vytvořit rekurzivní vztahový typ

Vstup: Označený prvek představující entitní typ.

Kroky:

1. Uživatel klikne na ikonu pro vytvoření rekurzivního vztahového typu.
2. Systém vytvoří vztahový typ v určeném místě od vybraného entitního typu podle nastavených konstant. Výchozí kardinalita vazeb `0..N`.

Vytvořit binární vztahový typ

Vstup: Označené dva prvky představující entitní typy.

Kroky:

1. Uživatel klikne na ikonu pro vytvoření binárního vztahového typu.
2. Systém vytvoří vztahový typ na středu mezi vybranými entitními typy. Výchozí kardinalita vazeb `0..N`.

Vytvořit ternární vztahový typ

Vstup: Označené tři prvky představující entitní typy.

Kroky:

1. Uživatel klikne na ikonu pro vytvoření ternárního vztahového typu.
2. Systém vytvoří vztahový typ na středu mezi třemi vybranými entitními typy. Výchozí kardinalita vazeb `0..N`.

Vytvořit hierarchii

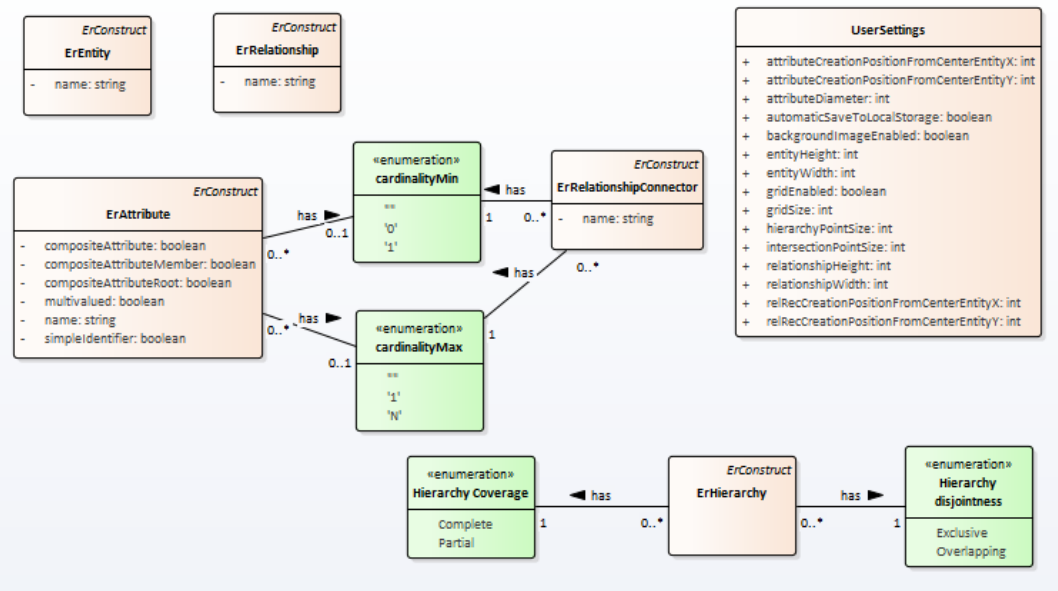
Vstup: Označený jeden a více prvků představující entitní typ.

Kroky:

1. Uživatel klikne na ikonu pro vytvoření hierarchie.
2. Systém změní zabarvení ikony a čeká na výběr dalšího prvku představující entitní typ.
3. Uživatel vybere prvek představující entitní typ.
4. Systém vytvoří hierarchii, jako podtřídy budou nastaveny zvolené entitní typy před stiskem ikony, jako nadtřída bude použit entitní typ vybraný následně.

5.4 Datový model

Jako poslední část specifikace si uvedeme datovou vrstvu aplikace. Data, které budeme ukládat, najdeme na obrázku 5.14 a jsou zachycena pomocí diagramu tríd v UML. Prozatím tu máme uvedené vybrané samostatné třídy s jejich atributy – třídu



Obrázek 5.14. Diagram datového modelu

`UserSettings` představující uživatelské nastavení aplikace, ostatní třídy budou mít na starost ukládaní dat o konstruktech.

Tento popis zatím použijeme jako dočasný podklad pro představu, jaká data budou v aplikaci figurovat. Samotnou podobu v jakých objektech a jak objekty budou mezi sebou provázány, necháme až na fázi implementace. Naše aplikace bude využívat určitou knihovnu, která podporuje vytváření grafů a diagramů. Tato knihovna má svůj vlastní způsob, jak data o prvcích diagramu uchovávat. Podrobnější vysvětlení způsobu uchovávání dat uvedeme v rámci následující kapitoly 6.

5.5 Závěr specifikace

V této kapitole jsme si uvedli základní analýzu vlastního řešení na ER modelovací nástroj. Nejdříve jsme vznesli požadavky na naše řešení. Pomocí digramu případů užití jsme uvedli funkcionality systému. Díky textovým scénářům případů užití jsme upřesnili některé funkcionality aplikace o určité detaily, které požadavky nezmiňují. Nakonec jsme na základě požadavků sestavili datový model aplikace, ve kterém jsme identifikovali, jaká data budou v aplikaci figurovat.

Kapitola 6
Implementace

Tuto kapitolu budeme věnovat vytvořenému řešení. Nejdříve zmíníme použité technologie, následně se seznámíme s knihovnou pro vytváření grafů, kterou naše řešení využívá, a nastíníme, jak jsme knihovnu integrovali do našeho řešení. Závěrem si popíšeme, jak je aplikace strukturována.

6.1 Technologie

V předešlé kapitole jsme diskutovali, jaké řešení pro náš typ aplikace zvolit. Rozhodli jsme se, že budeme vytvářet webovou aplikaci. Naše řešení je tedy založeno na trojici technologií – HTML, CSS, Javascript. HTML [26] je značkovací jazyk pro tvorbu webových stránek. CSS [27] neboli kaskádové styly se používají pro definici stylů, které určují, jak se mají HTML elementy stránky zobrazit. Důležitou složkou je Javascript [28–29], který umožňuje provádět dynamické změny ve stránce. Webová aplikace se liší od webové stránky díky použití Javascriptu, který umožňuje vytvářet aplikace srovnatelné jako počítačové programy s tím, že aplikace běží ve webovém prohlížeči. Toto dělá webovou aplikaci plně multiplatformní. Zároveň také kód aplikace běží na straně klienta, místo toho, aby byly posílány dotazy na server, kde jsou zpracovány a vyhodnoceny a uživateli se vrací zpět odpovědi.

Na začátek musíme podotknout, že Javascript nemá nic společného s jazykem Java, jak se na první pohled může zdát. V době vzniku byl jazyk Java populární a autoři do názvu vložili slova `java` z marketingových důvodů, aby se začal více používat.

Javascript je interpretovaný jazyk – překládán a vyhodnocován za běhu (pomocí interpretu), založen na objektovém a funkcionálním modelu – využívá takzvané prototypové dědičnosti. Tím se logika objektů liší například od jazyka Java, který používá třídní objektový model. Javascript také umožňuje do proměnné uložit funkci, často jazykem využívaný koncept. Díky tomu můžeme jako argument volané funkce vložit jinou funkci. Javascript je i událostmi řízený – na různé události je možné vytvořit posluchač, který v době vzniku události vykoná funkci předanou při jeho registraci.

Námi vytvořená aplikace je napsána v čistém Javascriptu – řešení nevyužívá žádné frameworky pro stavbu javascriptových aplikací. V řešení využíváme tyto tři následující knihovny:

- **JQuery** [30] – knihovna pro interakci Javascriptu s HTML, využita velmi okrajově
- **mxGraph** [31] – knihovna pro tvorbu klientských aplikací pracujících s diagramy, naše řešení silně využívá tuto knihovnu
- **saveSvgAsPng** [32] – knihovna pro generování obrázků ve formátu PNG[1] z SVG[2] grafiky, použita pro export diagramu v aplikaci do obrázku

[1] Portable Network Graphics - https://www.w3.org/TR/2003/REC-PNG-20031110/
[2] Scalable Vector Graphics - https://www.w3.org/TR/SVG2/

6.2 Knihovna mxGraph

Knihovna mxGrah je opensourcové řešení pro tvorbu editorů diagramů v rámci klientských aplikací. Knihovnu můžeme označit za framework pro stavbu aplikací tohoto typu. V kapitole týkající se existujících řešení jsme uváděli aplikaci `Draw.io`. Tato aplikace používá jako své jádro tuto knihovnu a je rozšířením či nadstavbou této knihovny. Tvůrce aplikace Draw.io je i tvůrcem této knihovny.

Tento framework poskytuje základní podporu funkcí a chování, které bychom očekávali – vykreslení prvků do plátna, interakci s prvky plátna, historie, možnost vytvořit si různé nabídky, přiblížení, oddálení, zarovnání, kopírování, vkládání a mnoho dalšího. V základu se jedná o schopnou kostru editoru. Úkolem uživatele je sestavit si z předpřipravených komponent vlastní editor. A v případě potřeb výchozí chování si upravit nebo rozšířit.

Knihovna je rozdělena do 8 balíčků – Editor, Handler, IO, Layout, Model, Shape, Util, View. Z těchto balíčku zmíníme některé důležité třídy:

- **mxEditor** – třída představující editor, zastřešuje chování celého frameworku
- **mxGraph** – hlavní třída umožňující práci s grafem, je fasádou pro některé funkce, které jsou poskytovány jinou třídou (vybrané funkce jiných tříd se dají volat přímo z této třídy)
- **mxGraphModel** – třída představující model grafu
- **mxGraphView** – třída starající se o vizuální část grafu
- **mxCell** – je třída, která představuje prvek grafu, pro následující text budeme používat i označení buňka
- **mxGeometry** – třída představující geometrii (pozice, šířka, výška,...) , každá buňka obsahuje svoji geometrii mxGeometry
- **mxCellState** – stav buňky, informace potřebné pro vizuální část diagramu dané buňky, obsahuje referenci na konkrétní mxCell
- **mxConstans** – třída obsahující konstanty například pro styly – názvy vlastností a jejich možností, další nezbytné konstanty

Uvedené třídy jsou mezi sebou provázány následujícím způsobem: Editor (mxEditor) v sobě obsahuje graf (mxGraph). Ten v sobě obsahuje model (mxGraphModel), který registruje všechny buňky (mxCell) grafu a pohled (mxGraphView), který v sobě eviduje stavy jednotlivých buněk (mxCellState). Každá buňka grafu má v sobě místo určené pro ukládání dat vztahující se k této buňce, obsahem může být cokoliv – číslo, řetězec nebo i objekt. V rámci grafu registrujeme styly, jejichž obsah je definován pomocí vlastností a jejich možností v rámci třídy mxConstats. Tyto styly, popřípadě i samotné vlastnosti můžeme nastavit pro každou buňku. Na základě nastavení těchto vlastností je buňka graficky reprezentována (barva textu, pozadí, tvar, určité chování, pozice popisku a další ...)

Koncepce knihovny je uzpůsobena tak, že model grafu tvoří buňky dvou typů – **vrchol** nebo **hrana**. Datová vrstva grafu je zde navržena podle teorie grafů z matematiky. Komponenty knihovny mezi sebou fungují na základě událostí. Některé třídy jsou producenti různých událostí a určité komponenty knihovny se starají o zpracování těchto událostí. Na tyto události je možné přidat i vlastní posluchače – vytváří prostor pro přizpůsobení chování.

Začlenění objektů knihovny do vlastního řešení je snadné, stačí do hlavičky HTML stránky přidat následující kus kódu a všechny třídy knihovny jsou rázem dostupné v našem Javascriptovém kódu.

```
<script type="text/javascript">
   mxBasePath = 'cesta ke knihovny/javascript/src';
   mxResources = 'en';
</script>
```

Při prezentaci této knihovny můžeme nabýt pocitu, že vytvoření diagramu díky této knihovně je snadné. Bohužel dokumentace knihovny pro nového uživatele není příliš sdílná tak, jak bychom někdy chtěli. Očekávali bychom širší pojednání o konceptu knihovny, které části knihovny mezi sebou a jak interagují. Knihovna má svůj vlastní systém událostí, nicméně nikde není dokumentováno, v jakém pořadí k jednotlivým událostem dochází. Například po vykonání změny v diagramu, jaké části a funkce knihovny jsou vykonávány a v jakém pořadí vzhledem ke vznikajícím událostem – kdy probíhá přepočet informací ve stavu buňky nebo vyhodnocování a vykreslování prvku do plátna.

Pokud chceme tuto knihovnu rozšířit do vlastního širšího řešení, je již potřeba detailněji pochopit její fungování. Proniknutí do knihovny není nijak rychlé, jedná se o celkem komplexní řešení a jeho prostudování vyžaduje čas. Bohužel někdy i API specifikace funkcí není zcela přesná. Na základě zjištění během vývoje naší aplikace, popis některých dostupných funkcí nebyl dostatečný a muselo se vyhledávat přímo v kódu knihovny, jak daná funkce funguje a co je jejím efektem. Naštěstí kód celé knihovny je programátorovi dostupný.

V rámci seznámení se s knihovnou jsme vytvořili diagram tříd popisující existenci tříd v jednotlivých balíčcích a hlavně zachycující dědičnost tříd mezi sebou. Některé třídy jsou potomkem jiné a tato skutečnost má vliv na jejich použití (můžeme tak použít funkce třídy, z které dědíme), tato informace není většinou v dokumentaci u dané třídy uvedena. Skutečnost dědičnosti jsme zjišťovali a zaznamenali na základě studia dostupného kódu. Diagram kvůli jeho velikosti je přiložen v rámci přílohy A.

Tvůrci knihovny v rámci dokumentace implementovali i větší počet miniaplikací, které demonstrují některé možné funkcionality knihovny. Pokud programátor chce zjistit, jak dané funkcionality bylo dosaženo, musí si projít zdrojový kód dané miniaplikace.

Pro použití knihovny se předpokládá znalost Javascriptu a objektově orientovaného programování, knihovna není určena pro úplné začátečníky. Závěrem lze říci, že programátor dostatečně obeznámený s touto knihovnou je schopný snadněji vytvářet editory diagramů různého typu. Není třeba celou řadu funkcionalit a chování vytvářet od úplného začátku.

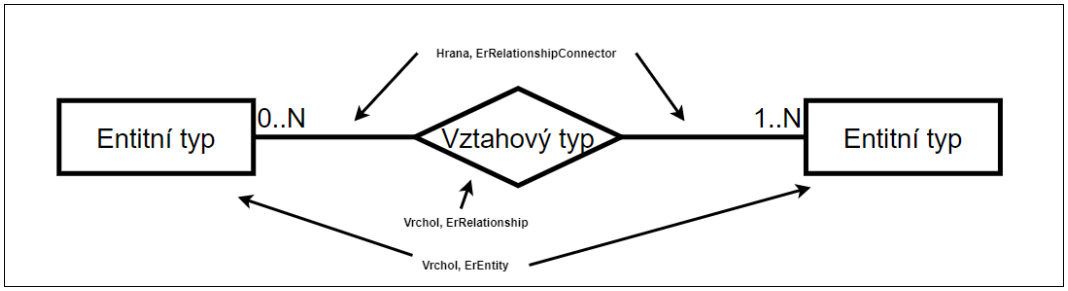
6.3 Integrace knihovny do řešení

Stěžejním bodem této práce bylo pochopení fungování knihovny a hledání způsobu, jak ji použít a přizpůsobit pro vytvoření editoru ER diagramů podle námi zvolené notace. V první řadě bylo potřeba vymyslet způsob, jak zaznamenat konstrukty pomocí dostupných prostředků knihovny – navrhnout styly, namapování konstruktu do buněk (vrcholy, hrany), ohlídat možná propojení mezi konstrukty a doplnit chování prvků a mezi prvky, které knihovna nenabízí.

Nyní si představíme základní myšlenky, jak jednotlivé konstrukty jsme v rámci logiky knihovny implementovali a jak knihovnu v některých zásadních případech bylo nutné rozšířit. Na začátek malá poznámka k terminologii, buňka diagramu umožňuje uložit vlastní datový objekt – do buňky budeme ukládat objekt třídy a buňky pak budeme nazývat *buňka typu „název dané třídy uložený v buňce“* nebo i pouze pomocí názvu třídy.

Entitní typ vytváříme jako vrchol grafu. Do buňky podle navrženého datového modelu (obrázek 5.14) vkládáme objekt třídy ErEntity, která ukládá název entitního typu.

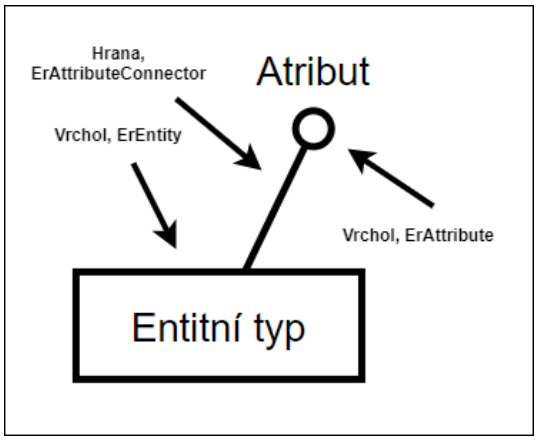
Vztahový typ namodelujeme pomocí dvou typů buněk. Prvním je buňka v roli vrcholu obsahující objekt ErRelationship, který ukládá název vztahového typu. Dalším typem je buňka v roli hrany obsahující objekt ErRelationshipConnector. Tuto hranu vytváříme mezi buňkou typu ErRelatinship a ErEntity. Buňka ErRelationshipConnector v sobě pro tuto vazbu ukládá pojmenování hrany a obě složky (spodní a horní omezení) kardinality vztahu. Výše popsané ještě schématicky shrnuje obrázek 6.1.



Obrázek 6.1. Způsob vykreslení vztahového typu v diagramu, popis účastnících se buněk – role, třída uložená v buňce

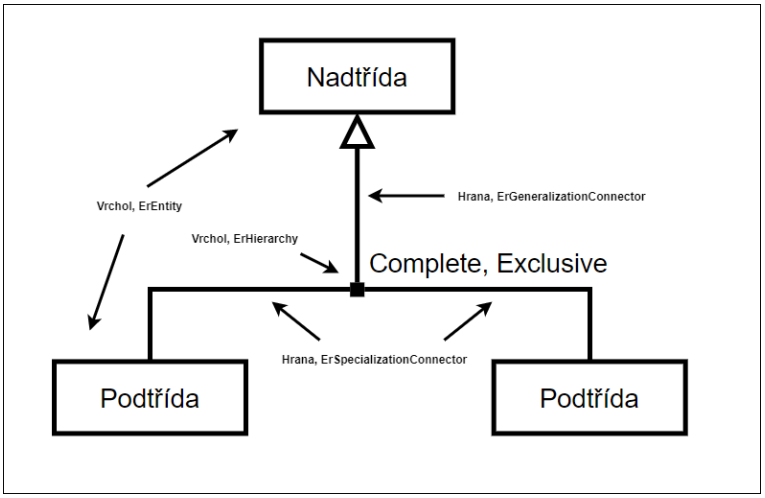
Tímto způsobem zaznamenání vztahu jsme schopni zaznamenat rekurzivní vztah, binární, ternární. Po implementování tohoto zobrazení, vazby mířící do buňky typu ErRelationship se shlukovaly do jednoho místa (výchozí chování zvoleného způsobu vedení čáry knihovny). Museli jsme chování upravit, aby čára vztahu vybrala vždy neobsazené místo ze čtyř vrcholů vztahového typu (tvar kosodélníku). Také jsme buňce ErRelationshipConnector museli doimplementovat druhý popisek pro zobrazení kardinality a vytvořit jeho automatické pozicování. Výchozí popisek se používá pro název účasti ve vztahu.

Atribut je navržen pomocí dvou buněk, první představuje samotný atribut – třída ErAttribute a ukládá v sobě informace o atributu (název, jednoduchý identifikátor, data ke složenému atributu...) opět dle návrhu třídy v datovém modelu (obrázek 5.14), zastává roli vrcholu. Druhým prvkem je buňka v roli hrany, jejímž obsahem je třída ErAttributeConnector, neukládá v sobě data potřebná k ER schématu. Tato buňka vytváří hranu mezi ErAttributem a druhým vrcholem, který může mít atribut – ErEntity, ErRelationship, nebo ErAttribute (v případě skládání do složeného atributu, v roli kořene). Na obrázku 6.2 máme popsán případ atributu entitního typu s označením jednotlivých buněk.



Obrázek 6.2. Způsob vykreslení atributu s jeho `vlastníkem` v diagramu, popis účastnících se buněk – role, třída uložená v buňce

Reprezentace **hierarchie** je tvořena díky třem typům buněk, první je hierarchie samotná jako vrchol se třídou ErHierarchy, objekt obsahuje informace o pokrytí a překrytí hierarchie. Dále pro vyjádření skutečnosti účasti v hierarchii existují dva druhy buněk – hrana ErGeneralizationConnector (generalizace), která reprezentuje spojení mezi hierarchií a nadtřídou. Z podstaty hierarchie tato hrana existuje právě jedna. Druhým typem je hrana ErSpecializationConnector (specializace), která vyjadřuje účast podtřídy v hierarchii. Obrázek 6.3 ukazuje zmíněné třídy v rámci příkladu hierarchie.

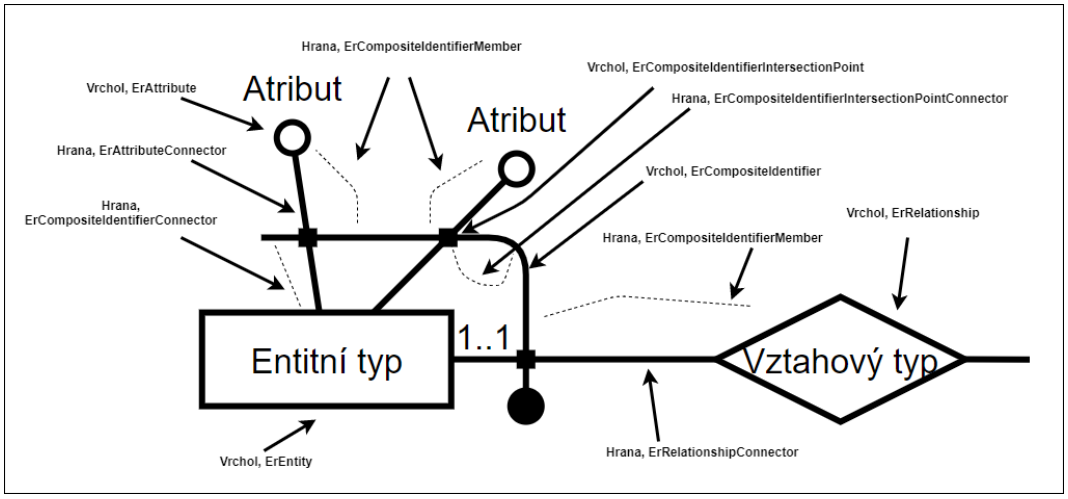


Obrázek 6.3. Způsob vykreslení hierarchie v diagramu, popis účastnících se buněk – role, třída uložená v buňce

Pro konstrukt hierarchie jsme museli vytvořit funkci, která se volá vždy po změně v diagramu a jejímž cílem je procházet hierarchie a opravovat vykreslení čar, které představují spojovací čáry. Z výchozího chování vedení čar docházelo k úseku překrývání čar (vykresleny přes sebe) – čára generalizace a specializace nebo všech čar specializací vedoucích z bodu pro hierarchii. Byla naimplementována oprava, která posouvá bod hierarchie (buňka typu ErHierarchy) do místa, aby toto překrývání nevznikalo.

Složený identifikátor identifikátor můžeme v diagramu potkat ve dvou podobách, případ několika atributů tvořící identifikátor, nebo „slabý“ identifikátor tvořený vztahy a atributy – entitní typ s tímto jedním nebo více „slabými“ identifikátory nazýváme slabý entitní typ. Tyto dvě situace jsou v diagramu zaznamenány stejným konceptem buněk. Buňka ErCompositeIdentifier je vrcholem a představuje složený identifikátor, zobrazena je pomocí čáry křížící vazby atributů a vztahů, na jednom z konců čáry je vyplněné kolečko. Další dva typy buněk v roli hrany – ErCompositeIdentifierConnector (vytváří spojení s ErEntity) a ErCompositeIdentifierMember (vytváří spojení s ErAtribute nebo ErRelationship) nejsou v diagramu viditelné, ale tvoří vazbu.

Poslední dva typy buněk tvoří pomocný zobrazovací prvek, který není uveden v notaci, ale vyznačuje účast atributu nebo vztahu v identifikátoru. Může nastat situace, že by automaticky vykreslený identifikátor pomocí algoritmu protínal vazbu atributu nebo vztahu, ale tento atribut nebo vztahový typ by nebyl účastníkem identifikátoru. Pro vyloučení omylu byl přidán čtvereček na místo styku vazby a čáry složeného identifikátoru, jako indikace, že je účastníkem identifikátoru. Tyto dva typy buněk nemají v ER schématu modelovací význam. Čtvereček je reprezentován buňkou typu ErCompositeIdentifierIntersectionPoint v roli vrcholu a nezobrazená hrana vytvářející spojení s identifikátorem jako ErCompositeIdentifierIntersectionPointConnector. Celý koncept buněk mapující složený identifikátor zachycuje obrázek 6.4.



Obrázek 6.4. Způsob vykreslení složeného identifikátoru v diagramu, popis účastnících se buněk – role, třída uložená v buňce. Nezobrazená hrana je reprezentována tečkovanou čárou.

Složený identifikátor je reprezentován pomocí čáry, která je určena body. V rámci řešení musela být naimplementována funkce, která vypočítává body identifikátoru na základě rozložení jeho členů v diagramu při každé změně diagramu. **Algoritmus pro výpočet bodů složeného identifikátoru** není zcela triviální, proto si popíšeme jeho princip.

1. V prvním kroku musíme identifikovat čáry, které budeme složeným identifikátorem křížit (pro lomené čáry vybíráme první úsek vedoucí od buňky entitního typu).
2. Každé vazbě musíme spočítat ortogonální (kolmou) vzdálenost čáry tj. jaká může být maximální ortogonální vzdálenost čáry složeného identifikátoru od entitního typu, vybereme tu nejkratší vzdálenost.
3. Musíme spočítat, v jaké vzdálenosti bude složený identifikátor vykreslován – v rozmezí do nejkratší vzdálenosti spočítané v předešlém kroku, určitým způsobem generujeme délku, aby se identifikátory nepřekrývaly (algoritmus tvoří rozptyl vzdáleností do určitého počtu identifikátorů nad entitním typem).
4. Spočítáme místa průniků mezi vazbou účastnících se prvků (atribut, vztahový typ) a slouženého identifikátoru v určené vzdálenosti z minulého kroku.
5. Získaná místa průniku musíme seřadit do nejkratší čáry – generujeme kombinace pořadí bodů a vybíráme variantu s nejkratší vzdáleností mezi body. Jelikož identifikátor vykreslujeme po obvodu obdélníka, do kalkulace musíme uvažovat vzdálenost vedoucí přes rohy, jinak by nemuselo docházet ke správnému seřazení bodů.
6. Seřazenou variantu bodů doplníme o další body, aby čára vedoucí přes roh obdélníka byla zakulacena.
7. Zjišťujeme směr čáry a kontrolujeme, aby body byly po směru hodinových ručiček. Pokud nejsou, přetočíme jejich pořadí. Složený identifikátor má na konci vyplněné kolečko, zachováváme body v jednom směru, aby nedocházelo vykreslování kolečka jednou na začátku čáry a po druhé na konci.
8. Na koncích dopočítáme přesah, aby čára identifikátoru za prvkem nějaký úsek pokračovala. Směr určujeme na základě vektoru posledních dvou bodů čáry.
9. Algoritmus upraví nastavení dotčených buněk – body složeného identifikátoru, pozice bodů označující účast v identifikátoru.

Na závěr uvedeme, že vytvořené ER schéma skrývající se v diagramu není tvořeno pomocí vlastního modelu, využívá struktury modelu knihovny a jednotlivé prvky schématu

mezi sebou neinteragují přímo, ale přes hrany mezi vrcholy modelu knihovny. Do každé buňky vkládáme objekt určité třídy nejen pro uložení některých dat, ale i abychom tak různé buňky při procházení modelu grafu mohli rozeznat. Pro identifikaci dané buňky využíváme reflexi třídy, v kódu používáme `instance of` a název třídy.

6.4 Koncepce řešení

Nyní si popíšeme strukturu aplikace a následně si uvedeme, jak je aplikace koncipována. Složka projektu obsahuje následující adresáře a soubory:

- **config** – složka obsahující konfigurační soubory pro editor
  - **keyhandler.xml** – konfigurační soubor obsahující definice klávesových zkratek a funkcí, které se mají vykonat
  - **popupmenu.xml** – konfigurační soubor popisující strukturu a funkce menu zobrazujícího se v plátně při kliknutí pravého tlačítka myši
- **css** – složka pro definice CSS stylů
  - **app.css** – CSS styly pro design stránky editoru
- **img** – složka pro obrázky použité v editoru
- **js** – složka pro javascriptové kódy
  - **app.js** – obsahuje hlavní funkci provádějící nastartování aplikace – volající funkce pro úpravy knihovny, vytvoření stylů, vytváření editoru...
  - **behaviour.js** – obsahuje funkce pro úpravu chování knihovny a nastavené vlastností komponent knihovny
  - **customObjectDecoder.js** – dekodér umožňující dekódování vlastních datových objektů v rámci buněk
  - **erConstructGraphBehaviourHandlers.js** – popisy obsluhovačů (handlerů) pro třídy modelu v případě vzniku vybraných událostí
  - **erModel.js** – obsahuje definice tříd představující datové objekty, které jsou využity v rámci buněk, funkce pro práci s buňkami diagramu tvořící model
  - **graphMakingHandler.js** – obsahuje funkce pro vytváření konstruktů v diagramu
  - **graphValidationHandler.js** – obsahuje funkce, které provádějí validaci diagramu po vykonání změny
  - **perimeter.js** – obsahuje implementaci chování projekce bodu na obvod prvku – využito pro napojení čar do vztahového typu ve 4 místech
  - **propertiesBar.js** – implementace panelu nastavení konstruktu
  - **sidebar.js** – implementace panelu pro vytváření konstruktů
  - **styles.js** – obsahuje definice stylů pro buňky diagramu
  - **testExample.js** – příklady diagramů pro účely testování a vývoje
  - **toolbar.js** – implementace lišty nástrojů
  - **userSettings.js** – objekty a metody týkající se uživatelského nastavení
  - **utils.js** – definice různých funkcí použitých napříč knihovnou
  - **windows.js** – obsahuje třídy pro okna vytvářená v aplikaci – okno pro načtení, uložení...
- **vendor** – složka pro zdrojové kódy knihoven třetích stran (mxGraph, JQuery, ...)
- **index.html** – HTML stránka webové aplikace
- **help.html** – HTML stránka obsahující nápovědu pro aplikaci, možnost jejího zobrazení v aplikaci

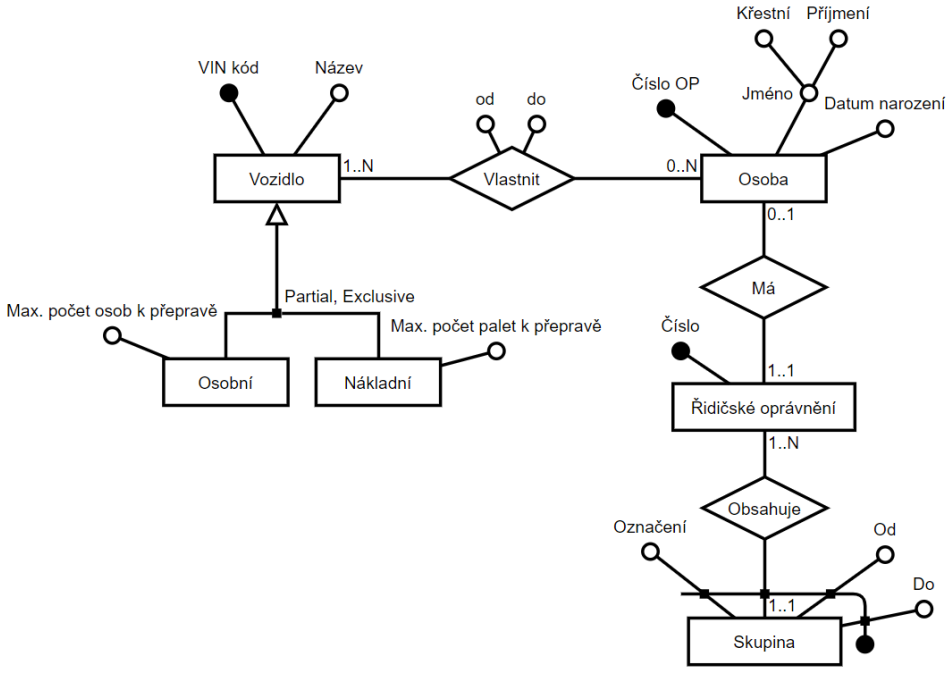
Aplikace je přístupná přes soubor `index.html`, zobrazením této stránky se načtou veškeré potřebné soubory a začne se vykonávat Javascriptový kód obsažen ve stránce – v rámci definice HTML stránky je obsažena funkce pro zavádění aplikace, která zahájí sestavení a přípravu celé aplikace.

Tato startovací funkce v první řadě vytvoří instanci editoru knihovny, předá knihovně oblasti stránky, které se stanou kontejnery pro části editoru (plátno, lišta nástrojů, atd.), dále z paměti prohlížeče načte uložená data v `local storage`[1] a následně zahájí nastavení součástí knihovny, doplnění funkcionalit editoru, doplnění prostředků a chování pro vytváření ER diagramu. Dodání funkcionalit do knihovny provádíme dvojím způsobem – výměnou implementace funkce obsažené v knihovně nebo přidáním posluchače na vznikající událost, které produkují různí producenti událostí v editoru.

Po nastartování aplikace je celé dění řízeno pomocí událostí a jejich zpracováním nebo vyvoláním definovaných funkcionalit.

6.5 Ukázka diagramu vytvořeného novým řešením

Abychom uzavřeli kapitolu zabývající se implementací, na závěr si ukážeme výstup diagramu z námi vytvořeného editoru. Na obrázku 6.5 je zachycen diagram systému evidujícího vozidla, osoby a řidičská oprávnění, který jsme modelovali v rámci rozboru existujících řešení a na kterém jsme demonstrovali použití existujících řešení. Zadání si můžeme připomenout v podkapitole 4.2.



Obrázek 6.5. Ukázka diagramu vytvořeného novým řešením

[1] https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

Kapitola 7
Manuál použití

V této části popíšeme základy práce s aplikací a jejího ovládání. Uvedeme si pár slov k instalaci a o možném spuštění na lokálním počítači.

7.1 Instalace

V rámci této práce jsme vytvořili webovou aplikaci, předpokládáme její nasazení na webovém serveru a její používání přes webový prohlížeč. Pro správce systému stačí obsah projektu aplikace nasadit na webový server a nakonfigurovat cestu k webové aplikaci. Následně stačí uživateli poskytnout URL, pod kterou je aplikace dohledatelná. Od běžného uživatele se očekává, že má nainstalovaný aplikací podporovaný webový prohlížeč a má k dispozici URL s umístěním aplikace. Tuto URL stačí zadat do adresního řádku prohlížeče a uživateli se aplikace spustí.

Podrobnostmi instalace webového serveru se zabývat nebudeme. Na okraj si ukážeme, jak aplikaci lze hostovat na svém počítači a spustit bez hlubší znalosti instalace a nastavení webových serverů. Na internetovém obchodu Chrome[1] je dostupná aplikace s názvem `Web Server for Chrome`[2]. Aplikaci je nutné nainstalovat a spustit, dostupnost je ověřena pro uživatele operačního systému Windows.

Po spuštění aplikace se objeví okno, které obsahuje možnost výběru složky, která bude hostována. Uživateli stačí na svém počítači vybrat umístění složky naší modelovací aplikace.

Po nastavení cesty ke složce aplikace stačí uživateli otevřít jakýkoliv prohlížeč na svém počítači a přejít na URL `http://127.0.0.1:8887/`(jako výchozí port je aplikací určen – 8887). Aplikace uživateli zobrazuje i odkaz, na který stačí kliknout.

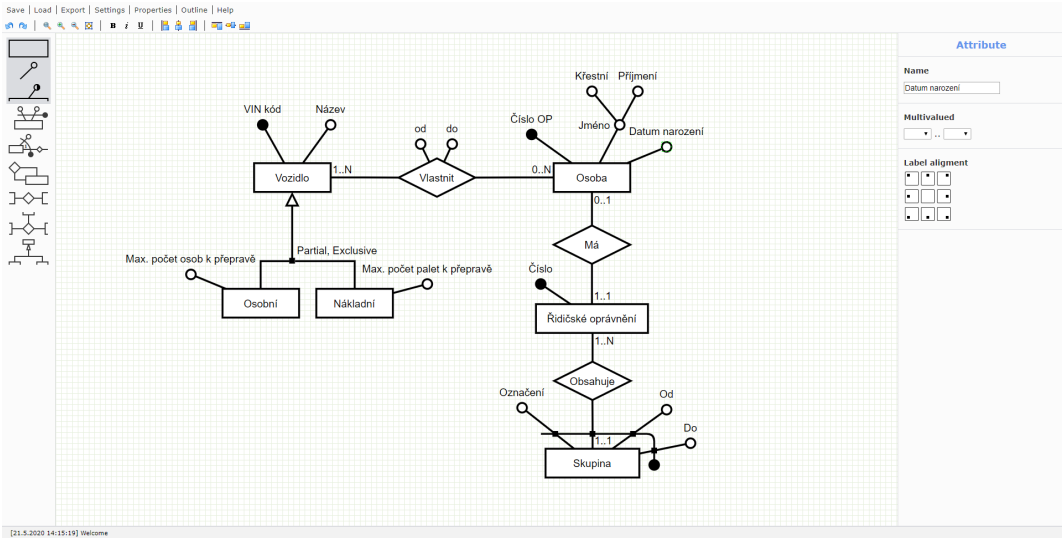
Webový server je spuštěn automaticky při spuštění aplikace (aplikace umožňuje jeho vypnutí a zapnutí). Uživatel si může volitelně změnit číslo přednastaveného portu, číslo portu poté musí správně uvést v URL, přes kterou aplikaci spouští.

7.2 Uživatelská příručka

Náhled na finální podobu aplikace si můžeme prohlédnout na obrázku 7.1. Editor se skládá z lišty nástrojů v horní části, panelu pro vytváření prvků diagramu v levé části, panel nastavení vlastností konstruktů v pravé části, informační lišty ve spodní části a plátna uprostřed.

Lišta nástrojů se skládá z dvou řádek. V první řadě jsou odkazy na okna editoru – okno pro uložení, načtení, nastavení, náhled diagramu, nápovědu a možnost skrýt panel nastavení vlastností konstruktu. V druhém řádku jsou uvedeny ikony pro práci s historií, ovládání zobrazení (přiblížení, oddálení), styly písma (možnost ztučnit písmo, podtrhnout, nastavit kurzívu) a zarovnání prvků podle osy.

[1] https://chrome.google.com/webstore/category/extensions
[2] https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb



Obrázek 7.1. Náhled na celý editor

Při ukládání editor umožňuje uživateli nastavit název souboru před stažením. Název se uvádí bez formátových přípon. Jeho stažení pak probíhá do určené složky pro stažené soubory podle nastavení v prohlížeči. Při načítání projektu dochází ke ztrátě současného modelu. Je potřeba neuložené změny si nejdříve uložit, jinak dojde ke ztrátě. Při exportování obrázku je možné nastavit násobek rozlišení. Výchozí hodnota je 1, pro vyšší číslo je vytvářen obrázek ve vyšším rozlišení než původní originál.

Editor umožňuje měnit nastavení pro určité chování – umožňuje vypnout pozadí na bílou barvu, zapnout mřížku – při posunu prvku se pozice upravuje do míst po určené mřížce. V rámci nastavení je možné určit velikost kroku mřížky. Pokud je mřížka vypnuta a je nastaven nenulový krok mřížky, v případě vytváření nových konstruktů pozice nového prvku je neustále zarovnávána do mřížky. Pokud uživatel nechce používat mřížku, doporučujeme nastavit krok na hodnotu 0. Dále je možné nastavením ovlivňovat hodnoty pro vytváření nových prvků – velikosti prvků, výchozí vzdálenosti prvků. Nastavení je možné resetovat do výchozích hodnot.

Editor po každé změně ukládá diagram do paměti prohlížeče, pokud je v nastavení zapnuta funkce automatického ukládání. Při otevření aplikace je editorem zobrazena úprava posledního diagramu. Pokud máme editor otevřený ve více záložkách, do paměti prohlížeče se vždy uloží poslední úprava diagramu z aktuální záložky (úpravy z ostatních záložek nejsou pak nikde uloženy, záložky se dělí o jeden paměťový slot).

V editoru je možné použít následující zkratky, zároveň uvádíme některé užitečné způsoby ovládání.

- **šipka vlevo** – vybere předcházející prvek
- **šipka nahoru** – vybere rodiče prvku
- **šipka dolů** – vybere potomka prvku
- **šipka vpravo** – vybere následující prvek
- **delete** – provede smazání vybraných prvků
- **ctrl + a** – vybere veškerý obsah plátna
- **ctrl + x** – zkopíruje vybrané prvky a následně odstraní
- **ctrl + c** – zkopíruje vybrané prvky
- **ctrl + v** – vloží dříve zkopírovaný výběr prvků
- **ctrl + z** – vrátí provedenou změnu
- **ctrl + y** – zopakuje vrácenou změnu

- **+** – přiblíží diagram
- **-** – oddálí diagram
- **ctrl + označení buňky** – přidá prvek do výběru (u již vybraného prvku zruší výběr)
- **pravé tlačítko myši v plátně** – zobrazí vyskakovací menu nabízející úpravy, při vykonání nad prvkem nabídne operace, které lze s prvkem provést
- **pravé tlačítko myši a posun myši** – posouvání diagramu

Editor umožňuje pro čáry vedoucí ke vztahovému typu, atributu či hierarchie přidat nebo odebrat navigační bod v místě vyvolání vyskakovacího menu nad požadovaným prvkem.

Nyní si uvedeme, jak se vytváří jednotlivé konstrukty pomocí levého panelu pro vytváření konstruktů. Levý panel zvýrazňuje dostupné funkce pomocí šedé barvy na základě aktuálně vybraných prvků. Prvky do výběru přidáváme pomocí `ctrl`. Každá položka následujícího seznamu představuje ikonu panelu v pořadí od shora dolů.

- **Vytvoření entitního typu** – Najdeme v panelu ikonu představující entitní typ, ikonu uchopíme a táhneme do místa, kde chceme entitní typ vytvořit a pustíme. Entitní typ se v daném místě vytvoří.
- **Vytvoření atributu** – Najdeme v panelu ikonu představující atribut, ikonu uchopíme a táhneme do místa, kde se nachází prvek, kterému chceme atribut přiřadit, aktuálně vybraný prvek se zvýrazní modrým obrysem. Atribut můžeme přidat entitnímu typu, vztahovému typu a atributu, který se může přidáním atributu změnit na složený atribut nebo přidáním rozšířit o další složku (prvek složeného atributu odstraníme označením požadováného atributu a použitím klávesy `delete`).
- **Změna identifikátoru** – Třetí ikona panelu představuje funkci, která vybraný atribut změní na identifikátor, nebo pokud je již identifikátorem, změní na jednoduchý atribut. Pro vykonání je potřeba vybrat atribut (kolečko, které ho představuje). Funguje jen pro případy, které dávají smysl, indikováno zvýrazněním ikony.
- **Vytvoření složeného identifikátoru** – Pro vytvoření složeného identifikátoru musíme vybrat atributy na stejném entitním typu (kolečko, nebo čára k němu vedoucí). Po výběru stačí kliknout na ikonu představující složený identifikátor a ten je následně vytvořen.
- **Modelování slabého entitního typu** – Pro vytvoření identifikátoru značící slabý entitní typ je nutné nejdříve vybrat atributy a vztahového typu jednoho entitního typu (lze označit čáry atributu, kolečko atributu, vazby vztahového typu). Po výběru stačí kliknout na ikonu představující případ slabého entitního typu a následně je vytvořen složený identifikátor.
- **Vytvoření rekurzivního vztahového typu** – Provedeme výběr jednoho entitního typu a klikneme na ikonu představující rekurzivní vztahový typ. Proběhne vytvoření rekurzivního vztahového typ s dvojnásobnou účastí vybraného prvku.
- **Vytvoření binárního vztahového typu** – Provedeme výběr dvou entitních typů a klikneme na ikonu představující binární vztahový typ. Proběhne vytvoření binárního vztahového typu mezi vybranými entitními typy
- **Vytvoření ternárního vztahového typu** – Provedeme výběr tří entitních typů a klikneme na ikonu představující ternární vztahový typ. Proběhne vytvoření ternárního vztahového typu mezi třemi vybranými entitními typy.
- **Vytvoření hierarchie** – Nejdříve provedeme výběr entitních typů účastnících se v hierarchii jako podtřídy, následně klikneme na ikonu představující hierarchii, následně vybere jeden entitní typ (klikneme na něj), který bude v roli nadtřídy. Následně dochází k vytvoření hierarchie. Přerušit vytváření hierarchie při výběru nadtřídy lze opětovným kliknutím na ikonu nebo pomocí klávesy `esc`.

Další vlastnosti prvků diagramu modelujeme pomocí pravého panelu nastavení vlastností konstruktu. Jeho obsah se mění na základě vybraného prvku.

- **Změna názvu konstruktu** – Název lze změnit dvojitým kliknutím na popisek (editace v diagramu), nebo pomocí označení konstruktu a změny hodnoty `Name`, uložení probíhá pomocí klávesy `enter` nebo opuštěním textového vstupu.
- **Pojmenování účasti vazby ve vztahovém typu** – Pojmenování vazby probíhá dvojitým kliknutím na vazbu, vyvolá se úprava popisku v diagramu. Pozici lze posunout pomocí bílé tečky v diagramu, která se zobrazí při výběru čáry.
- **Vícehodnotový atribut** – Vytvořit vícehodnotový atribut je možné pomocí označení konkrétního atributu a změny jedné z hodnot kardinalit. Pro zrušení vícehodnotového atributu stačí jednu z hodnot kardinalit nastavit na prázdnou hodnotu.
- **Úprava kardinalit vztahového typu** – Kardinalitu vztahového typu lze upravit výběrem konkrétní vazby a změnou hodnoty v pravém panelu.
- **Úprava pokrytí a překrytí hierarchie** – Vlastnosti upravujeme po výběru hierarchie (výběr čtverečku styku vazeb nebo kliknutím na popisek obsahující vlastnosti hierarchie) v panelu nastavení.

Na závěr dodáme poznámku ke kopírování a vkládání. Při kopírování výběru v diagramu editor provádí kontrolu smysluplných celků diagramu a vkládá jen validní celky. Například pokud vybereme entitní typ s atributem, ale je označena jen čára k atributu bez atributu (kolečka není vybráno), po vložení se přidá pouze entitní typ, protože atribut nebyl validně označen. Vložení entitního typu a čáry by nedávalo smysl.

Kapitola 8
Testování

Testování je jednou z fází životního cyklu aplikace. Často této části z různých důvodů nebývá věnována dostatečná pozornost, jak by si zasloužila. I když situace se pořád zlepšuje. Nicméně je nepochybné, že testování software má své opodstatnění v rámci vývoje systému. V této části provedeme krátké uvedení do problematiky a ilustrujeme, jak aplikace byla testována.

8.1 Testování software

Obor testování má za úkol otestovat aplikaci a před uvedením do provozu najít defekty v systému tak, aby během jeho provozu nedocházelo k selháním. Existují různé techniky návrhů testů s různou mírou účinnosti i způsoby jejich vykonání. Vždy je potřeba zvážit, co testujeme a jaký test se hodí pro daný případ. Zároveň je potřeba zmínit, že testování všeho není možné – např. všechny kombinace vstupů (kromě případu několika triviálních vstupů). Spíše je potřeba určit si priority, jaké oblasti otestovat, které jsou rizikovější.

Testování, pokud má být efektivní, musí poměrně silně spolupracovat s ostatními částmi vývojového cyklu. Je zřejmé, že čím později odhalíme chybu v rámci vývoje, tím roste cena za její odhalení. Chyba se může vyskytovat i v prvotních fázích analýzy a její odhalení může proběhnout až při implementaci. Pak musí dojít ke změnám ve všech předešlých fázích – to pozdrží dobu vývoje a samozřejmě navýší cenu za vývoj. Jak testování zapojit do jednotlivých částí vývoje nám popisuje W-model [33], který lidem v oblasti testování je dobře známý. Tento model nám zdůrazňuje, že je důležité testovat už od raných fází vývoje.

8.2 Pojmy z oblasti testování

Uvedeme si několik základních pojmů z oblasti technik testování. Začneme pojmy `white box` (bílá krabička) a `black box` (černá krabička) testování. U první techniky při návrhu testu známe testovaný systém a testy tvoříme na základě znalosti vnitřní struktury systému. Při druhém přístupu `black box` se oprošťujeme od vnitřní znalosti systému, nezajímá nás jeho vnitřní implementace, při návrhu testu vycházíme ze specifikace nebo dokumentace funkcionalit systému.

Dalším pojmem je testovací scénář, je to jednoznačný předpis, který uvádí jaké kroky provést na testované aplikaci a jaký máme očekávat výsledek. Scénář by měl obsahovat určité náležitosti jako id testu, název, popis testu, vstupní podmínky, testovací data, očekávaný výsledek a jednotlivé kroky testu, u kterých může být uveden i výsledek daného kroku testu. Samozřejmě je možné uvádět i další informace, my zde uvádíme typickou základní kostru.

Existují postupy, jak vytvářet testy, podle toho, co potřebujeme otestovat. Pro návrh testů kombinace vstupů můžeme použít následující techniky:

- MCC – Multiple condition coverage

- MC/DC – Multiple Condition/Decision Coverage
- Pairwaise testing
- C/DC – Condition/Decision Coverage
- CC – Condition coverage
- DC – Decision coverage
- Náhodný výběr

Techniky jsme seřadili podle toho, jak velkou míru pokrytí nám daná technika poskytuje od nejvyšší po nejnižší. Jak jednotlivé techniky použít, uvádět nebudeme.

Dále můžeme navrhovat testy průchodu programem, nebo testy životního cyklu datových objektů. Pokud pro danou aplikaci máme vytvořeny zmíněné testy, pak je můžeme použít jako podklad pro sepsání testovacích scénářů.

Existují různé druhy provedení testů podle toho, co testujeme. Například jednotkové testy, funkční testy, integrační testy. Tyto uvedené testy řeší testování kódu. Na sepsání těchto testů existují různé frameworky, které umožní i automatizované spuštění vytvořených testů. Pro aplikace v jazyce JAVA, například framework JUnit[1]. Dále můžeme testovat uživatelské rozhraní, buď manuálně – tester podle scénáře provádí test nebo i automatizovaně – opět pomocí frameworku, který umožní specifikovat scénáře, které reagují s uživatelským rozhraním aplikace a provádí automatizované testy. Pro testování webových aplikací můžeme zmínit například nástroj Selenium[2].

8.3 Otestování aplikace

V případě naší aplikace, fáze implementace v rámci životního cyklu aplikace zabrala příliš dlouhou dobu a ve vymezeném čase nezbylo příliš času na důkladné testování. Z tohoto důvodu bylo zvoleno uživatelské testování nejdůležitějších funkcionalit systému. Provedené testy bychom možná mohli označit za `smoke testy` – testy, které mají ověřit, zda nejdůležitější části aplikace správně fungují.

Během testování a konzultace finální verze aplikace vyšlo najevo, že ER má ještě o trochu větší vyjadřovací schopnosti, než aplikace umožňuje. Během analýzy zůstalo opomenuto (z důvodu, že se nejedná o příliš časté případy), že by mělo být možné vytvořit libovolný n-arní vztah, včetně opakování účasti stejných entitních typů – vlastnost plyne z definice, že vztahový typ je n-tice entitních typů. Dále okolo skutečnosti slabého entitního typu je možné vytvořit identifikátor, který je určen i pouze samotným vztahem, bez přítomnosti atributu. Aplikace obsahuje omezení účasti minimálně jednoho vztahu a jednoho atributu.

Tyto vlastnosti byly v rámci analýzy požadavků přehlédnuty a v našem řešení se s nimi tak neuvažovalo. Nicméně aplikace je stavěna tak, že v budoucnu je možné jejich doplnění.

Nyní si představíme některé testovací scénáře sestavené pro testování aplikace. Scénáře jsou postaveny z větší části na základě dokumentace, částečně i podle znalosti systému – kombinací přístupu `black box` a `white box` testování.

Parametr	Obsah
ID testu:	001
Název testu:	Vytvoření diagramu
Shrnutí:	Uživatel provede vytvoření diagramu dle určeného postupu.
Doba testu:	7 minuty

[1] JUnit – https://junit.org/junit5/
[2] Selenium – https://www.selenium.dev/

Vstupní podmínky: Žádné

Kroky testu:

1. Uživatel vytvoří entitní typ 1
2. Uživatel přejmenuje entitní typ 1 pomocí panelu nastavení konstruktu
3. Uživatel vytvoří entitní typ 2
4. Uživatel přejmenuje entitní typ 2 pomocí dvojkliku na popisek názvu
5. Uživatel označí vytvořené entitní typy a vytvoří mezi nimi binární vztahový typ 1 pomocí ikony pro vytvoření binárního vztahového typu
6. Uživatel přejmenuje vztahový typ 1
7. Uživatel přidá atribut 1 entitnímu typu 1 a provede jeho přejmenování
8. Uživatel entitnímu typu 1 přidá atribut 2, provede přejmenování a změní atribut na identifikátor pomocí ikony pro změnu atributu na identifikátor
9. Uživatel vztahovému typu 1 opraví pomocí panelu nastavení konstruktu kardinality vztahového typu od entitního typu 1 na `1..N` a od entitního typu 2 na `0..N`
10. Uživatel vztahovému typu 1 přidá atributy 3 a 4
11. Uživatel entitnímu typu 2 přidá atribut 5 jako identifikátor a provede přejmenování
12. Uživatel vytvoří složený identifikátor, vytvoří atribut 6 a přejmenuje, poté přidá atributu 6 další dva atributy 7 a 8 a provede jejich přejmenování pozici.

Testovací data: Entitní typ 1 – auto
Entitní typ 2 – člověk
Vztahový typ 1 – vlastnit
Atribut 1 – barva
Atribut 2 – VIN kód
Atribut 3 – od
Atribut 4 – do
Atribut 5 – OP
Atribut 6 – jméno, atribut 7 – křestní, atribut 8 – příjmení

Očekávaný výsledek: Uživatel vytvoří diagram obsahující dva entitní typy (jeden obsahující atribut a identifikátor, druhý obsahující identifikátor a složený atribut) a binární vztahový typ se dvěma atributy.

Parametr	**Obsah**
ID testu:	002
Název testu:	Přidání hierarchie do diagramu
Shrnutí:	Uživatel do diagramu přidá hierarchii.
Doba testu:	5 minuty
Vstupní podmínky:	Diagram obsahující tři entitní typy

Kroky testu:

1. Uživatel označí dva entitní typy
2. Klikne na ikonu pro vytvoření hierarchie
3. Uživatel označí jeden entitní typ, který nebyl ještě označen

4. Uživatel označí vytvořenou hierarchii a pomocí panelu nastavení konstruktu nastaví pokrytí a překrytí

Testovací data: Pokrytí – částečné
Překrytí – překrývající
Očekávaný výsledek: Uživatel vytvoří hierarchii mezi třemi entitními typy. První dva vybrané entitní typy v roli podtříd, následně vybraný entitní typ v roli nadtřídy. Hierarchie má částečné pokrytí a překrývající překrytí.

Parametr	**Obsah**
ID testu:	003
Název testu:	Uložení a načtení diagramu
Shrnutí:	Uživatel nejdříve provede uložení projektu, poté provede v projektu změnu a načte původní projekt a zkontroluje, zda jeho načtení proběhlo v pořádku.
Doba testu:	5 minuty
Vstupní podmínky:	Vytvořený diagram obsahující entitní typy, nejméně jeden rekurzivní, binární a ternární vztahový typ, slabý entitní typ, složený identifikátor a hierarchii. Dále libovolné atributy – nejméně jeden klasický, jednoduchý identifikátor, složený atribut, vícehodnotový.
Kroky testu:	

1. Uživatel aktivuje okno uložení pomocí funkce uložit.
2. Vyplní do textového pole požadovaný název souboru
3. Uživatel provede uložení pomocí tlačítka uložit
4. Uživatel zavře okno uložení
5. Uživatel přidá do současného diagramu entitní typ
6. Uživatel klikne na přidaný entitní typ a v panelu nastavení konstruktu provede jeho přejmenování
7. Uživatel aktivuje okno načtení
8. Vybere stažený soubor z disku
9. Uživatel provede akci načíst a souhlasí, že změny budou ztraceny
10. Uživatel klikne na entitní typ – panel nástrojů zobrazí nastavení pro jméno
11. Uživatel klikne na atribut – panel nástrojů zobrazí nastavení pro jméno, omezení pro vícehodnotový atribut, zarovnání popisku
12. Uživatel klikne na vztahový typ – panel nástrojů zobrazí nastavení pro jméno
13. Uživatel klikne na vazbu vztahového typ – panel nástrojů zobrazí nastavení pro kardinality
14. Uživatel klikne na hierarchii – panel nástrojů zobrazí nastavení pro pokrytí a překrytí, zarovnání popisku

Testovací data: Název souboru – diagram
Název nového entitního typu – auto
Očekávaný výsledek: Systém vytvoří soubor s názvem `diagram.xml`, uživateli se podaří vytvořit entitní typ auto, uživatel načte uložený soubor

bez chyb, diagram načteného projektu vypadá stejně jako původní před zahájením testu. Při výběru prvků panel nástrojů vypisuje správné možnosti nastavení.

Parametr	**Obsah**
ID testu:	004
Název testu:	Export obrázku
Shrnutí:	Uživatel provede uložení obrázku.
Doba testu:	4 minuta
Vstupní podmínky:	Vytvořený diagram obsahující entitní typy, nejméně jeden rekurzivní, binární a ternární vztahový typ, slabý entitní typ, složený identifikátor a hierarchii. Dále libovolné atributy – nejméně jeden klasický, jednoduchý identifikátor, složený atribut, vícehodnotový.
Kroky testu:	1. Uživatel aktivuje okno pro exportování obrázku 2. Do pole pro měřítko zadá uvedenou hodnotu 3. Uživatel provede export pomocí funkce export png 4. Uživatel otevře uložený obrázek a zkontroluje
Testovací data:	Měřítko – 2
Očekávaný výsledek:	Systém vytvoří soubor s názvem `er.png`, vygenerovaný obrázek zachycuje diagram zachycený v programu.

Parametr	**Obsah**
ID testu:	005
Název testu:	Změna nastavení
Shrnutí:	Uživatel provede uvedené změny v nastavení a zkontroluje jejich efekt
Doba testu:	5 minuty
Vstupní podmínky:	Nastavení: zobrazit obrázek pozadí – zapnuté, šířka entitního typu – 80, výška entitního typu – 30
Kroky testu:	1. Uživatel vytvoří entitní typ 2. Uživatel zobrazí okno pro nastavení pomocí funkce nastavení v menu 3. Uživatel deaktivuje vlastnost zobrazit obrázek pozadí – pozadí diagramu by se mělo změnit na bílé 4. Uživatel změní v nastavení šířku entitního typu a výšky 5. Uživatel vytvoří druhý entitní typ – nově vytvořený entitní typ by měl být dvakrát širší a vyšší než první vytvořený.
Testovací data:	Entitní typ šířka – 160 Entitní typ výška – 60
Očekávaný výsledek:	Pozadí plátna je bílé, systém vytvoří 2 entitní typy, jeden je dvakrát větší než první vytvořený.

8.4 Shrnutí testování

V úvodní části kapitoly jsme se seznámili se základními pojmy z oboru testování software. Následně jsme uvedli, jak naše aplikace byla otestována a jak vypadají některé

konkrétní testovací scénáře pro otestování naší aplikace. Musíme konstatovat, že část testování by si zasloužila větší pozornost. Ve vymezeném čase nebylo možné například provést testování kódu (jednotkové, funkční, integrační testy) a funkčnost systému jsme museli ověřit pomocí testů, které většinou dokáží nalézt snadno objevitelné chyby systému.

Podle provedených testů zaměřujících se na průchod důležitých funkcionalit (uvedených i mnoha dalších) aplikace nevykazuje případy selhání. Toto však neznamená, že aplikace nemůže skrývat defekty, které například při málo pravděpodobných kombinací dat či operací mohou být aktivovány a zapříčinit selhání.

Kapitola 9
Závěr

Tato práce se v první řadě zabývala návrhem a vytvořením nástroje, který umožní modelování na konceptuální vrstvě pomocí jazyka ER. Úspěšně se nám podařilo implementovat editor, který umožňuje vytvářet ER diagramy v notaci, která na základě dostupného rozboru existujících řešení nebyla v žádném modelovacím nástroji prozatím implementována. Skutečnost, že tuto notaci žádné řešení nenabízí, není způsobená tím, že by se zvolená notace nepoužívala nebo byla bezvýznamná. Námi zvolená notace měla vyšší vyjadřovací schopnost oproti identifikovaným notacím v nalezených modelovacích nástrojích. Hlavním důvodem proč tato notace nebyla implementována, je pravděpodobně, že implementovat automatický způsob jejího vykreslení není triviální. Existující nástroje nabízejí notace, u kterých lze snadněji implementovat jejich vykreslení.

Námi zvolenou notaci nelze vyjádřit prostým vykreslením geometrických útvarů a vytvořením vazeb. Nad vykreslením prvků diagramu musí existovat určitá hlubší logika, která provádí výpočty pro vykreslení, návrh umístění komponent prvku.

Během řešení práce jsme se museli vypořádat s pochopením obsáhlé cizí knihovny, která nebyla zcela dobře dokumentována. Její popis nedostatečně ilustroval vnitřní fungování procesů v knihovně. Bylo tak ztíženo identifikovat místa, kde je potřeba upravit funkcionalitu tak, aby bylo možné vytvořit editor s požadovaným chováním a také aby prvky diagramu představující konstrukty byly vykresleny a chovaly se dle našich představ.

Při návrhu vykreslování prvků dostupnou knihovnou jsme museli najít řešení pro konstrukt hierarchie tak, aby mohl být reprezentován bodem hierarchie a vazbami k entitním typům. V některých případech byl problém s umístěním bodu hierarchie, v tomto významu řídícím bodem, který výchozí chování knihovny umístilo tak, že docházelo k překrývání vazeb vedoucích do tohoto bodu. Podařilo se navrhnout algoritmus, který překrývající čáry detekoval a posunul řídící bod tak, aby nedocházelo k překryvům souběžných čar.

Složitým úkolem bylo najít algoritmus pro výpočet řídicích bodů složeného identifikátoru tak, aby křížil prvky, které se ho účastní, naplánoval optimální zobrazení vzhledem k aktuálnímu rozmístění těchto prvků kolem entitního typu a předcházel možnému překryvu s jiným složeným identifikátorem u stejného entitního typu. Nalezení tohoto algoritmu se podařilo, navržené řešení je relativně robustní, ale určitý prostor pro nalezení lepšího řešení zde existuje. Možná problematika hledání optimálního umístění složeného identifikátoru by mohla být předmětem jiné samostatné práce.

Dalším významným dosaženým výsledkem práce je, že se nám podařilo poskytnout ucelené a přehledné srovnání významných ER notací od různých autorů, které v dostupných zdrojích v tak širokém pojetí nebylo nalezeno.

Výsledným produktem této práce je funkční a stabilní webová aplikace. Webové aplikace v posledních letech rostou na oblibě, jsou stále populárnější a postupně vytlačují programy v klasické podobě. Výsledkem je editor pro snadnější vytváření ER diagramů s přijatelnou vyjadřovací schopností. Zároveň tento editor najde své uplatnění jako mo-

delovací nástroj v rámci výuky konceptuálního modelování v jazyce ER na Fakultě elektrotechnické ČVUT.

Umíme si představit, že by vytvořený editor v budoucnu mohl uživateli nabídnout větší možnosti přizpůsobení diagramu – měnit barvy prvků, velikosti písma, styly písma a tak podobně. Také by mohlo být řešení rozšířeno o další automatické algoritmy, možnost ukládání diagramů v rámci uživatelského účtu a další vylepšení, které by ještě více napomáhali při práci a tím dosáhnout komplexnějšího modelovacího ER nástroje.

Literatura

[1] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*. Dec, 1990, s. 1-84. Dostupné na DOI 10.1109/IEEESTD.1990.101064.

[2] Bachman, Charles W. Data Structure Diagrams. *SIGMIS Database*. New York, NY, USA: ACM, jul, 1969, ročník 1, č. 2, s. 4–10. ISSN 0095-0033. Dostupné na DOI 10.1145/1017466.1017467.

[3] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*. New York, NY, USA: ACM, jun, 1970, ročník 13, č. 6, s. 377–387. ISSN 0001-0782. Dostupné na DOI 10.1145/362384.362685.

[4] Chen, Peter. *Entity-relationship modeling: historical events, future trends, and lessons learned*.

[5] Chen, Peter Pin-Shan. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*. ACM, 1976, ročník 1, č. 1, s. 9–36.

[6] OMG. *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1*.
`http://www.omg.org/spec/UML/2.4.1`.

[7] Hay, David C. A comparison of data modeling techniques. *Essential Strategies*. 1999, ročník 41, s. 43.

[8] Song, Il-Yeol, Mary Evans a E K. Park. A Comparative Analysis of Entity-Relationship Diagrams. *Journal of Computer and Software Engineering*. 01, 1995, ročník 3.

[9] Teorey, Toby J., Dongqing Yang a James P. Fry. A Logical Design Methodology for Relational Databases Using the Extended Entity-relationship Model. *ACM Comput. Surv.* New York, NY, USA: ACM, June, 1986, ročník 18, č. 2, s. 197–222. ISSN 0360-0300. Dostupné na DOI 10.1145/7474.7475.

[10] Elmasri, Ramez a Shamkant Navathe. *Fundamentals of Database Systems*. 6th vyd. USA: Addison-Wesley Publishing Company, 2010. ISBN 0136086209, 9780136086208.

[11] Silberschatz, Abraham, Henry Korth a S. Sudarshan. *Database Systems Concepts*. 5 vyd. New York, NY, USA: McGraw-Hill, Inc., 2006. ISBN 0072958863, 9780072958867.

[12] McFadden, Fred R., Mary B. Prescott a Jeffrey A. Hoffer. *Modern Database Management*. 5th vyd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998. ISBN 0805360549.

[13] Atzeni, Paolo, Stefano Ceri, Stefano Paraboschi a Riccardo Torlone. *Database Systems - Concepts, Languages and Architectures*. McGraw-Hill Book Company, 1999 . ISBN 0-07-709500-6.

[14] *ERDPlus*.
`https://erdplus.com/`.

[15] *Creately*.
https://creately.com/.

[16] *Creately demo app*.
https://creately.com/app/?login_type=demo.

[17] *Draw.io*.
https://www.draw.io/.

[18] *LucidChart*.
https://www.lucidchart.com/pages/tour/ER_diagram_tool.

[19] *Draw.io*.
https://cloud.smartdraw.com/.

[20] *Gliffy*.
https://go.gliffy.com/go/html5/launch.

[21] *TerraER*.
http://www.terraer.com.br/.

[22] *JAVA*.
https://docs.oracle.com/en/java/.

[23] *Java Virtual Machine*.
https://docs.oracle.com/javase/specs/jvms/se14/html/index.html.

[24] Richta, Karel a Jiří Sochor. *Softwarové inženýrství I*. ČVUT, 1996.

[25] Berners-Lee, Tim, Larry Masinter a Mark McCahill. *RFC1738: Uniform Resource Locators (URL)*.

[26] *HTML standard*.
https://www.w3.org/TR/html52/.

[27] *CSS*.
https://www.w3.org/Style/CSS/.

[28] *Javascript MDN web docs*.
https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[29] Žára, Ondřej. *JavaScript*. Albatros Media as, 2015.

[30] *Jquery*.
https://api.jquery.com/.

[31] *mxGraph*.
https://github.com/jgraph/mxgraph.

[32] *SaveSvgAsPng*.
https://www.npmjs.com/package/save-svg-as-png.

[33] Bureš, Miroslav, Miroslav Renda, Michal Doležel a others. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Grada Publishing as, 2016.

Příloha A
Elektronické přílohy

Součástí práce jsou přílohy uvedené v následujícím seznamu souborů a složek, které jsou odevzdány v rámci datového média.

- Složka app – obsahuje veškeré zdrojové kódy vytvořené aplikace
- ERD.xml – zdrojový soubor diagramu otevíratelný ve vytvořené aplikaci, diagram systému popsaný v podkapitole 4.2.
- mxGraphClassModel.png – diagram tříd v rámci balíčků knihihovny mxGraph
- thesis.pdf – text této práce