

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Vizuální reprezentace databázového schématu relační databáze

BAKALÁŘSKÁ PRÁCE

Peter Rozum

Brno, jaro 2011



Masarykova univerzita
Fakulta informatiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Datum: 20. 4. 2011

Student: Peter Rozum

Program: Informatika

Obor: Počítačové systémy a zpracování dat

Vedoucí práce: Mgr. Jan Géryk

Název práce: Vizualní reprezentace databázového schématu relační databáze

Zadání: Cílem bakalářské práce je vytvořit aplikaci pro vizuální reprezentaci relačního datového schématu. Vstupem pro aplikaci bude soubor ve formátu XML obsahující datové schéma. Ve výsledné prezentaci budou názorně zobrazeny entity schématu a vztahy mezi nimi. Text práce bude obsahovat přehled a porovnání algoritmů pro přehledné vykreslení grafů. Bude vybrán vhodný algoritmus a ten bude použit při tvorbě aplikace. Aplikace bude vytvořena v jazyce C++.

Základní literatura: E. F. Codd. A relational model of data for large shared data banks. Commun. ACM, 26:64–69, January 1983.

Helen Purchase. Which aesthetic has the greatest effect on human understanding? In Giuseppe DiBattista, editor, Graph Drawing, volume 1353 of Lecture Notes in Computer Science, pages 248–261. Springer Berlin / Heidelberg, 1997. 10.1007/3-540-63938-1_67.

R. Tamassia. On embedding a graph in the grid with the minimum number of bends. SIAM J. Comput., 16(3):421–444, 1987.

Souhlas se zadáním (podpis, datum)

Rozum 20.4.2011
student(ka)

Géryk
vedoucí
bakalářské práce

N?
vedoucí
odpovědného pracoviště

* KTP
* KPGD
* KPSK
* KIT
* Hodící se označte!

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Peter Rozum

Vedúci práce: Mgr. Jan Géryk

Podakovanie

Týmto chcem poďakovať Mgr. Janovi Gérykovi za pomoc a cenné rady potrebné pri tvorbe tejto bakalárskej práce. Ďalej chcem poďakovať svojej rodine a blízkym, ktorí ma podporovali pri písaní tejto práce, ako aj počas celého bakalárskeho štúdia.

Zhrnutie

Táto práca sa venuje automatizovanému vizuálnemu znázorňovaniu schémy relačných databáz. V práci je popísaná relačná schéma s možnosťami jej znázorňovania a sú tu popísané a porovnané spôsoby akými je možné schému vykresľovať. Ako výsledok tejto práce je poskytnutá aplikácia, ktorá je schopná vybraným spôsobom relačnú schému vizuálne znázorniť.

Kľúčové slová

Databázová schéma, ERD, vykresľovanie grafov, C++, Qt, OGDF.

Obsah

1	Úvod	1
2	Databázová schéma a jej vizuálna reprezentácia	2
2.1	Databázová schéma	2
2.1.1	Relačný model a alternatívy	2
2.1.2	Relačná databázová schéma	3
2.2	Entitne-relačné modelovanie	4
2.2.1	Súčasti ERM	5
2.2.2	Notácie ERD	6
2.3	Vizuálna reprezentácia databázovej schémy	7
2.3.1	Tvorba relačnej databázovej schémy	8
2.3.2	Vizualizácia vytvorenej relačnej databázovej schémy	8
3	Prehľadné vykresľovanie grafov	10
3.1	Estetika vykresľovania grafov	10
3.2	Kríženie hrán v grafe	11
3.3	Postupy pri vykresľovaní grafov	12
3.3.1	Hierarchické rozloženie	13
3.3.2	Ortogonálne rozloženie	15
3.4	Vhodné rozloženie pre vykresľovanie relačnej databázovej schémy	16
3.4.1	Experiment	17
3.4.2	Výber vhodného rozloženia	19
4	Aplikácia pre vizuálnu reprezentáciu schémy relačnej databázy	20
4.1	Vstupný formát pre aplikáciu	20
4.1.1	Definícia formátu	20
4.2	Externá funkcionálna využívaná aplikáciou	23
4.2.1	Framework pre automatizované vykresľovanie grafov	23
4.2.2	Grafické používateľské rozhranie	23
4.3	Funkcie aplikácie	24
4.3.1	Špecifikácia prípadov použitia	24
4.4	Grafické používateľské rozhranie aplikácie	26
4.5	Implementácia	27
4.5.1	Platforma a použitý software	27
4.5.2	Spracovanie vstupu	27
4.5.3	Tvorba a realizovanie vykresľovania reprezentácie schémy	29
4.5.4	Internacionalizácia a lokalizácia	31
4.6	Kompilácia a spúšťanie	31
5	Záver	33
	Literatúra	34
A	Obsah priloženého CD	36

1 Úvod

Databázy sú podstatnou časťou softvérových systémov a aplikácií. Správne navrhnutá databáza je schopná podporiť prácu s dátami v nej uloženými. Použitie databáz umožňuje pri vývoji softvéru sústrediť sa viac na riešenie zadaného problému. Je odbúraná nutnosť zaoberať sa problémami, ako napríklad určením presnej formy uloženia dát na disku, zabezpečovania efektívneho prístupu k dátam, alebo zabezpečením zálohovania týchto dát.

V súčasnosti sú v praxi stále populárne databázy založené na relačnom modeli. Dáta sú v tomto prípade ukladané do tabuliek. V prípade, že chce používateľ databázy získať nejaké dáta z databázy, sformuluje dopyt, ktorý typicky špecifikuje tabuľky a ich stĺpce, z ktorých sú dáta požadované. Dopyt môže špecifikovať aj obmedzenia pre filtrovanie dát. Na takto sformulovaný dopyt databáza odpovedá. Spája potrebné tabuľky, filtruje stĺpce, riadky a vo výsledku vráti používateľovi požadované dáta ako jednu tabuľku.

Pri tvorbe databázových dopytov je potrebné mať prehľad o tom ako sú tabuľky s dátami organizované a aké typy hodnôt obsahujú. Tieto informácie je možné znázorniť schémou zobrazujúcou tabuľky s typom dát, ktoré sú v nich ukladané a zobrazujúcou tiež vzťahy, ktoré medzi tabuľkami existujú. Takto znázornená schéma databázy sprehľadňuje orientáciu a uľahčuje používateľom tvorbu dopytov.

Databázová schéma je zvyčajne dostupná ako výsledok návrhu databázy. Ak však k navrhutej cieľovej databáze, ktorá už môže obsahovať značné množstvo náročným spôsobom zozbieraných dát, neexistujú žiadne podklady poskytujúce informácie o schéme, vzniká problém ako s takouto databázou pracovať. V tomto prípade je užitočné a žiadané aby bolo možné zo zadanej databázy schému spätne vytvoriť.

Schému relačnej databázy je možné znázorniť ako matematický graf obsahujúci vrcholy a hrany. Vrcholy v tomto prípade zastupujú tabuľky a hrany zastupujú vzťahy. Databázové systémy poskytujú možnosti na to ako z danej databázy získať informácie o jej schéme. Aby bolo možné tieto informácie vhodne zobraziť, je potrebné vyriešiť problém ako túto schému (graf) vykresliť. Automatické vykresľovanie grafov je náročná úloha ak je požadovaná istá miera prehľadnosti vykresleného grafu.

Po tejto úvodnej kapitole pokračuje táto práca kapitolou druhou, venujúcou sa relačnej databázovej schéme. Kapitola bližšie popisuje schému a jej obsah, metodiku pre tvorbu schémy a tiež možnú grafickú reprezentáciu schémy.

Kapitola tretia je venovaná spôsobom ako prehľadne vykresľovať grafy na plochu. Uvedené sú tu kritériá, ktoré sú predpokladom pre prehľadný graf a sú tu uvedené základné spôsoby, ktorými je možné graf vykresliť. Analyzované sú tu aj možnosti, ktorý spôsob je dobrý v konkrétnom prípade použiť. Popísané a porovnané sú tu aj algoritmy, ktorých výstupom je vykreslenie grafu požadovaným spôsobom.

V kapitole štvrtej je priestor venovaný vytvorenej aplikácii pre vizuálnu reprezentáciu relačnej databázovej schémy. Je tu popísané ako je aplikácia vytvorená, z čoho sa skladá, ako funguje a čo využíva pre svoju činnosť.

Záverečná kapitola zhŕňa a hodnotí poznatky uvedené v tejto práci. Venuje sa možnostiam pre využitie poznatkov tu uvedených pre ďalšiu prácu zaoberajúcu sa touto problematikou a popisuje možnosti pre rozšírenie vytvorenej aplikácie.

2 Databázová schéma a jej vizuálna reprezentácia

Databázy sa stali v oblasti informačných technológií nepochybne neoddeliteľnou súčasťou systémov. Potreba efektívne a spoľahlivo ukladať spracovávané dáta, alebo nimi akokoľvek manipulovať, je veľmi podstatná. Nároky na databázové systémy sú stále rastúce. Stabilita, robustnosť, efektivita, rýchlosť a množstvo ukladaných dát sú len niektoré z kritérií, ktoré formulujú požiadavky na tieto systémy.

Dosiahnuť splnenie definovaných požiadaviek sa stáva čoraz ťažšou úlohou. Databázové systémy sa stávajú vďaka svojmu vývoju stále viac a viac komplexnejšími a súčasne množstvo a rozsah dát, ktoré sú v týchto systémoch ukladané, si vyžaduje zvýšenú pozornosť.

Táto kapitola sa zaoberá databázovou schémou relačných databáz. Práve databázová schéma je rozhodujúca pre prípravu databázového systému na príjem dát, ich bezpečné uloženie a jeho schopnosť s nimi efektívne pracovať. Schéma je dôležitá aj pre používateľov databázy, ktorí si tiež potrebujú udržiavať prehľad o forme ukladaných dát.

V texte tejto kapitoly je vysvetlené, čo to databázová schéma je a aký je jej význam a využitie. Kapitola sa zaoberá aj procesom pri ktorom schéma vzniká, súvisiacou modelovacou resp. diagramatickou technikou a spôsobom ako schému graficky znázorniť.

2.1 Databázová schéma

Systém riadenia databáz (Database Management System, DBMS) je súhrn programov, ktoré pracujú nad databázou, umožňujú používateľom spravovať túto databázu a poskytujú im k nej prístup. V súčasnej dobe sú výrazne populárne DBMS, ktoré sú založené na relačnom modeli, ktorý definuje štruktúru dát pomocou matematických relácií. Tento model predstavil v roku 1970 Codd [4]. Systémy implementujúce tento model sú označované ako relačné DBMS (Relational DBMS, RDBMS). Medzi hlavných zástupcov RDBMS patria Oracle, MS SQL, DB2, MySql, PostgreSQL, Sysbase. Spôsob, akým je relačný model v známych RDBMS implementovaný však nie úplne obsahuje všetky vlastnosti definované v Coddovom relačnom modeli. Preto je možné stretnúť sa s označením takýchto „neúplných“ RDBMS ako pseudo RDBMS (PRDBMS).

2.1.1 Relačný model a alternatívy

Relačné DBMS nie sú jediným typom DBMS. Ako alternatívy k nim existujú napríklad hierarchické, sieťové, objektové a XML (eXtensible Markup Language, rozšíriteľný značkovací jazyk) DBMS.

Hierarchický model vytvára stromovú hierarchiu („otec - syn“), kde každá tabuľka môže mať niekoľko podradených tabuliek (môže mať niekoľko synov), ale každá tabuľka môže byť podradená iba jednej tabuľke (môže mať iba jedného otca). Táto stromová štruktúra znamená, že medzi dátami sa modeluje vzťah „jedna k mnoho“ (1:M).

Sieťový model dovoľuje hierarchiu, kde každá tabuľka môže mať viac podradených tabuliek a zároveň môže byť podriadená viacerým tabuľkám (rozšírenie hierarchického modelu).

Tieto modely (hierarchický a sieťový) prestali byť s príchodom relačného modelu pre potreby praxe atraktívne a ich využiteľnosť začala klesať. Význam mali hlavne v období pred relačným modelom, ktorý sa stal používanejším aj vďaka využívaniu hardvéru s vyšším výkonom, na ktorom relačný model značne vynikal nad hierarchickým a sieťovým modelom.

Objektový model využíva možnosti, ktoré prináša objektovo orientovaný vzor (Object Oriented Paradigm). Objektové RDBMS (Object RDBMS, ORDBMS) sú teda ideálne pri objektovom programovaní. Sú schopné využívať rovnaký typový systém ako aplikácie, ktoré ho využívajú a ukladať tak celé objekty. Objektový model je vhodný predovšetkým na ukladanie dát komplexných dátových typov, ako napríklad audio a video. Ukladať takéto dáta v relačnej databáze je typicky obmedzené.

XML Databázy ukladajú dáta vo formáte XML. Databázy, v ktorých je základnou jednotkou pre ukladanie dát XML súbor, sú označované ako natívne XML databázy (Native XML Database, NXD). Využívanie týchto databáz je žiadané hlavne v oblasti webových aplikácií.

2.1.2 Relačná databázová schéma

S RDBMS sa typicky komunikuje pomocou štruktúrovaného dopytovacieho jazyka (Structured Query Language, SQL), ktorý bol prvýkrát štandardizovaný americkým národným štandardovým inštitútom (American National Standards Institute, ANSI) v roku 1986. Podmnožiny tohto jazyka sú orientované na definíciu dát a na manipuláciu s nimi. Pre definíciu dát je to DDL (Data Definition Language) a DML (Data Manipulation Language) je určený pre manipuláciu s dátami.

V RDBMS sú dáta ukladané vo forme tabuliek, medzi ktorými existujú vzťahy (relácie). Systém si však potrebuje uchovávať dodatočné popisné informácie, ktoré súvisia s ukladaným obsahom. Tieto informácie (metadata) sú dôležité pre činnosť RDBMS a zároveň aj pre jeho používateľov.

Metadata, popisujúce štruktúru databázy, tvoria databázovú schému. Schéma by mala byť navrhovaná tak, aby ju počas existencie databázy nebolo nutné meniť a aby mohla zaručiť požadovanú vypovedaciu schopnosť databázy.

Schéma obsahuje objekty, ktoré sú určené pre ukladanie dát alebo sú schopné vykonávať operácie nad ukladanými dátami. Typickým objektom schémy je napríklad:

tabuľka – obsahuje riadky a stĺpce s dátami,

pohľad – virtuálna tabuľka, ktorá vznikne vykonaním daného SQL dopytu nad jednou alebo viacerými tabuľkami,

spúšťač (trigger) – používateľom zadefinované príkazy, ktoré sa automaticky spúšťajú ako odozva na požadovanú udalosť,

synonymum – alternatívne meno pre objekt schémy.

Tabuľka je základná jednotka pre úschovu dát v databáze a ako taká sa skladá z riadkov. Riadky tabuľky sa skladajú zo stĺpcov, ktoré majú pevne stanovený dátový typ.

Okrem typov, ktoré bližšie špecifikujú aký druh dát je povolené v tabuľke uchovávať, je možné špecifikovať ďalšie integritné obmedzenia. Napríklad v systéme Oracle existuje päť druhov integritných obmedzení [14]. Patria medzi ne:

- nulové hodnoty (null values) – povolenie nulových hodnôt umožňuje ponechať určité bunky tabuľky nevyplnené,
- unikátne hodnoty (unique values) – obmedzenie, ktoré zabezpečuje kontrolu, že sa v určitom stĺpci tabuľky nevyskytnú dvakrát rovnaké hodnoty,
- primárne kľúče (primary keys) – vymedzenie stĺpcov tabuľky, ktoré majú byť určené k tomu aby jednoznačne identifikovali záznam, ktorého sú súčasťou,
- cudzie kľúče (foreign keys) – vymedzenie stĺpcov tabuľky, ktoré jednoznačne identifikujú záznam inej tabuľky,
- komplexná kontrola integrity (complex integrity check) – používateľom zadefinované pravidlo aplikované na stĺpec alebo stĺpce, ktoré sa využíva pri manipulácii s dátami a povoľuje alebo zamietá túto manipuláciu.

V RDBMS je schéma ukladaná zvyčajne formou dátového slovníka. Dátový slovník je prístupný aj používateľom, ktorí vďaka nemu majú možnosť zistiť štruktúru databázy. Implementovaný môže byť ako súhrn špeciálnych systémových tabuliek/pohľadov, ktoré sú používateľovi prístupné spravidla len na čítanie. Akákoľvek úprava týchto systémových štruktúr je vyhradená pre vlastný RDBMS, ktorý schému automaticky upravuje na základe používateľom zadáných DDL výrazov. Týmto spôsobom je RDBMS sčasti schopný zaručiť integritu a konzistenciu ním spravovaných databáz.

2.2 Entitne-relačné modelovanie

Entitne-relačný model (ERM) bol predstavený v roku 1976 ako model schopný zahrnúť dôležité informácie o reálnom svete [1]. Využitie tohto modelu je výrazné hlavne pri entitne-relačnom modelovaní databáz. Pri tvorbe databáz je nutné popísať potrebné dáta, ktoré má byť vytváraná databáza schopná ukladať. Rovnako je potrebné vedieť určiť organizáciu týchto dát.

Dôvodom pre využitie tohto modelu je potreba jednotného pohľadu na dáta ukladané v databáze, pretože je schopný vhodne reflektovať štruktúru týchto dát. Zároveň je tento model vhodný aj ako model uľahčujúci komunikáciu medzi osobami pracujúcimi na jednom projekte, kde uľahčuje porozumenie problematiky skúmanej oblasti a odstránenie nejednoznačností.

Nejednoznačnosť alebo nepresnosť môže vzniknúť ako dôsledok rozdielov v používaní prirodzeného jazyka rôznymi skupinami osôb z rôznych odborov, ale aj nezjednotenými definíciami používaných pojmov a nedokonalou znalosťou problematiky. Uvedeným, ale aj ďalším problémom je možné predísť, vďaka modelovacej technike ako ERM, pretože poskytuje možnosti zjednotenia rôznych pohľadov na problémovú oblasť do jedného pohľadu a prístupu, ktorý je jednoznačný a dobre zrozumiteľný.

2.2.1 Súčasti ERM

Pri tvorbe entitne-relačného modelu sa spravidla postupuje spôsobom zhora nadol. Znamená to, že ide o snahu zadaný problém (skúmaný celok) postupne zjemňovať až dovtedy, kým nie je dostatočne jemne dekomponovaný.

Modelovanie začína identifikovaním dôležitých dát (entít) a vzťahov, ktoré medzi týmito dátami existujú. Následne sa identifikujú atribúty entít a vzťahov a popisujú sa rôzne integritné obmedzenia. Do modelu sa takto pridávajú viaceré súčasti zvyšujúce jeho vypovedaciu hodnotu. V nasledujúcom texte sú hlavné súčasti modelu stručne popísané.

Entita

Pod výrazom (silná) entita chápeme niečo, čo je možné jednoznačne identifikovať. Príkladom entity môže byť nejaký objekt vyskytujúci sa v reálnom svete, udalosť alebo pojem. Množinou entít rovnakého druhu je entitná množina resp. entitný typ.

Slabá entita je špeciálny typ entity. Jej existencia je závislá na existencii inej entity (alebo entít). Jednoznačne identifikovať takúto entitu je možné len ak vieme identifikovať aj entitu (alebo entity), na ktorej je slabá entita závislá.

Vzťah

Vzťah je asociácia, ktorá existuje medzi dvoma alebo viacerými prvkami entitných množín (participantov). Vo vzťahu môžu figurovať entity z ľubovoľných entitných množín, teda aj dve entity z jednej entitnej množiny.

Stupeň vzťahu je počet entít v danom vzťahu. Vzťah medzi dvoma entitami je označovaný aj ako binárny. Vzťah vyššieho stupňa ako 2 je možné pretransformovať na vzťahy stupňa 2 [6, 15], aj keď nie vždy je možné presne zachovať pôvodnú sémantiku vzťahu. Entity ktoré vzniknú pri tejto transformácii sa nazývajú asociatívne entity.

Atribút

Vlastnosť ktorou daná entita disponuje je atribút. Množina atribútov reprezentuje entitu. Všetky entity jednej entitnej množiny majú rovnakú množinu atribútov. Možné hodnoty atribútov sú dané doménami atribútov.

Ak na základe atribútu alebo skupiny atribútov dokážeme jednoznačne identifikovať prvok entitnej množiny, hovoríme, že atribút resp. skupina atribútov tvorí kandidátny kľúč v entitnej množine. Primárny kľúč je jeden vybraný kandidátny kľúč danej entitnej množiny určený na jednoznačnú identifikáciu entít.

Multiplicita

Vzťahy je možné bližšie popísať multiplicitou, ktorá sa skladá z dvoch zložiek. Sú to kardinalita a participácia. Kardinalita určuje maximálny možný počet vzťahov, ktorých sa môže daná entita zúčastniť. Participácia určuje, či sa daného vzťahu musí zúčastniť každá entita z entitnej množiny alebo len niektoré. Hovorí sa aj o povinnej, resp. nep povinnej participácii.

Generalizácia/špecializácia

Generalizácia/špecializácia alebo aj dedičnosť, reprezentuje istú formu vzťahu medzi entitnými množinami. Vzniká rozdelenie na nadtriedu a podtriedu. Podtrieda vlastností nadtriedy rozširuje, pričom vlastnosti nadtriedy sú v podtriede zachované.

2.2.2 Notácie ERD

Počas entitne-relačného modelovania vzniká ERD (Entity-Relationship Diagram, entitne-relačný diagram). Pre zobrazenie ERD na plochu existuje viacero notácií. Autori [18] zhrnuli a porovnali desať najpoužívanejších notácií, ktoré sú často využívané v literatúre a CASE (Computer-aided software engineering, počítačom podporované softwarové inžinierstvo) nástrojoch.

Ako je ukázané v [18], nie všetky používané notácie majú rovnakú vypovedaciu schopnosť. Je možné nájsť niekoľko odlišností. Chýbať môže možnosť zobraziť vzťahy vyššieho stupňa ako 2, zobraziť atribúty vzťahov, plne zobraziť generalizáciu/špecializáciu alebo zobraziť cudzie kľúče entít.

Problémovým sa môže stať aj konvertovanie diagramu medzi vybranými notáciami. Je nutné venovať pozornosť rozdielom a v prípade potreby rozšíriť danú notáciu tak, aby bola konverzia čo najpresnejšia.

Nasledujúci text sa venuje spôsobom, ktoré sú notáciami používané a popisuje, aké sú možnosti zobrazovania komponent na ERD. Komponenty ktorých notácie sú tu popísané sú obmedzené na entitné množiny, atribúty, vzťahy a multiplicitu.

Notácia entitných množín

Entitná množina sa na diagrame zvyčajne znázorňuje ako obdĺžnik (s hranatými alebo zaoblenými okrajmi) obsahujúci podstatné meno, reprezentujúce názov entitnej množiny. V prípade, že znázorňovaná entita je slabá, vykresľovaný obdĺžnik je obvykle zdvojený.

Notácia atribútov

V prípade atribútov sa používajú dva spôsoby. Prvý, pôvodný spôsob uvedený v [1], zobrazuje atribúty ako horizontálne elipsy obsahujúce meno atribútu, spojené čiarou s entitou, ktorej patria. Ak je atribút primárnym kľúčom, je jeho názov v elipse podčiarknutý.

Druhý spôsob notácie zobrazuje atribúty ako položky v obdĺžniku entity. Primárne kľúče sú zvyčajne označované „PK“ alebo „#“, cudzie kľúče „FK“.

Notácia vzťahov

Pre znázornenie vzťahu je používaný diamant, cez ktorý sú participanti spojení čiarami. Názov vzťahu je obsiahnutý v diamante. Ďalší spôsob notácie vzťahu je jednoduché spojenie participantov čiarou a názov vzťahu je uvedený ako popis pri čiare.

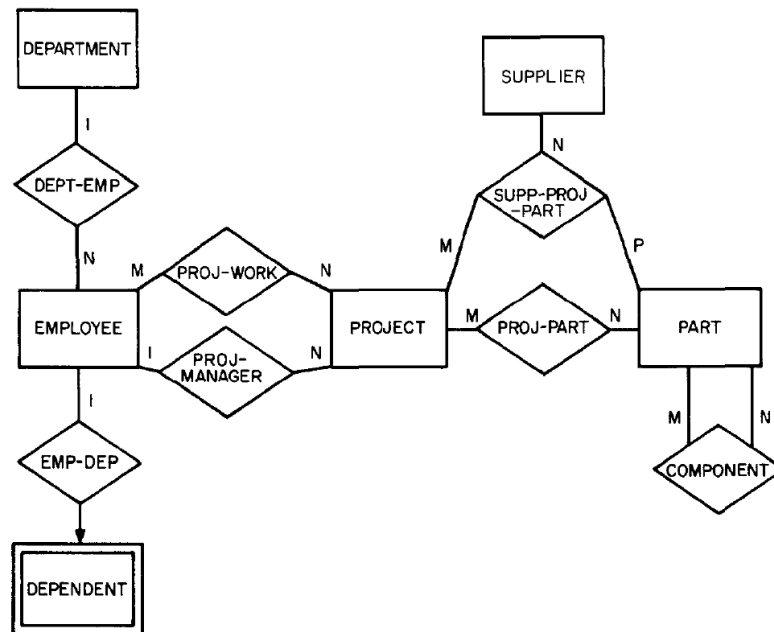
Pri notovaní vzťahu čiarou je dobré si uvedomiť, že takýto spôsob znemožňuje notovanie iných vzťahov ako binárnych. Problémom sa stáva aj notovanie atribútov týchto vzťahov. V „diamantovom“ prípade je atribút notovaný ako horizontálna elipsa pripojená k diamantu čiarou.

Problém so vzťahmi vyšších stupňov a atribútmi je možné riešiť zavedením novej umelej entitnej množiny, ktorá bude plniť úlohu sprostredkovateľa. Tento sprostredkovateľ bude v binárnom vzťahu so všetkými participantmi a bude obsahovať atribúty pôvodného vzťahu.

Notácia multiplicity

Multiplicita má tiež viacero spôsobov notovania. Notovanie môže byť numerické, kedy sa kardinalita a participácia vyjadruje ako dvojica čísel alebo graficky, ako to je napríklad pri použití notácie vraních nôh (crows feet notation).

Podstatné je aj to, kde na diagrame je multiplicita značená. Rozlišujú sa dva spôsoby, „Look Across“ a „Look Here“. V prípade spôsobu „Look Here“ je multiplicita týkajúca sa danej entity značená pri mieste, kde sa táto entita napája ku vzťahu. Pri použití „Look Across“ je táto multiplicita značená na opačnom konci vzťahu, teda tam, kde sa nachádza druhá entita, s ktorou je tá prvá vo vzťahu.



Obr. 2.1: Ukážka ERD znázorneného chanovou notáciou. [1]

2.3 Vizualná reprezentácia databázovej schémy

Ak požadujeme, aby bolo možné zorientovať sa v danej databázovej schéme rýchlo a bez veľkej námahy, je potrebné vedieť túto schému vhodne reprezentovať. V nasledujúcom texte je výklad zameraný na zobrazovanie databázovej schémy pomocou ERD. Popisuje sa tu forma, v akej sa diagram vyskytuje a rozdiely v notácii oproti typickej forme ERD modelujúcom skúmanú oblasť.

2.3.1 Tvorba relačnej databázovej schémy

Pri tvorbe relačnej databázovej schémy používa dizajnér zvyčajne diagramatickú techniku ERD a postupne vytvára a dopĺňa potrebné diagramy. V tomto procese sú vytvorené spravidla tri modely skúmanej oblasti.

Prvý vytvorený model je konceptuálny model, ktorý je tvorený ako ERD v plnej forme. Má predstavovať úplnú a presnú reprezentáciu požiadaviek na databázu. Tento model je navrhovaný bez akýchkoľvek úvah o konkrétnej implementácii.

Ďalší vytváraný model je logický model. Tvorba tohto modelu je založená na výslednom konceptuálnom modeli. Logický model odráža skutočnosť ako sú dáta organizované s ohľadom na relačný model dát. To znamená, že sa obsah konceptuálneho modelu transformuje do formy tabuliek a relácií. Entitné množiny sú mapované na tabuľky a ich atribúty na stĺpce tabuliek. Prebieha normalizácia tabuliek, ktorej účelom je odbúranie redundancie a anomálií, ktoré môžu vznikať pri manipulovaní s dátami. Normalizácia upravuje tabuľky, atribúty, relácie a vznikajú nové tabuľky a relácie. Logický model tiež zahŕňa aj integritné obmedzenia.

Po vytvorení logického modelu nastáva fáza vytvorenia fyzického modelu. Tento model slúži ako popis konkrétnej implementácie databázovej schémy na cieľový RDBMS. Fyzický model je závislý na cieľovom RDBMS. Pri jeho tvorbe je nutné dobre poznať všetky funkcie a obmedzenia cieľového RDBMS a využiť ich pre optimalizáciu výslednej schémy. Pri optimalizácii môže dôjsť napríklad k potrebe denormalizovať niektoré časti logického modelu pre dosiahnutie zvýšenia výkonnosti.

2.3.2 Vizualizácia vytvorenej relačnej databázovej schémy

V prípade, že neexistuje dokumentácia k databázovej schéme alebo je zastaralá, vzniká potreba diagram schémy vytvoriť. Takto vygenerovaná vizualizácia schémy môže výrazne napomáhať používateľom databázy.

Informácie potrebné k spätnému vytvoreniu diagramu schémy sú uložené v dátovom slovníku databázového systému. Odtiaľ je potrebné zistiť, aké tabuľky sa v databáze nachádzajú, aké dáta obsahujú a ako sú tieto tabuľky prepojené cudzími kľúčmi. Spôsob ako získať tieto informácie o schéme je závislý na konkrétnom databázovom systéme.

Po získaní potrebných informácií, je potrebné nájsť vhodný postup ako ich znázorniť na plochu. Je možné využiť zjednodušenú formu ERD, kde tabuľky budú entity, čiže späťne vytvoriť niečo ako logický model databázy. Nie je však možné vytvoriť presný a úplný logický model v takej forme ako bol pôvodne navrhnutý, pretože pri prechode od logického modelu k modelu fyzickému je vykonávaná optimalizácia a prípadne aj denormalizácia.

V relačnej databáze je vzťah uložený tak, že jedna participujúca tabuľka (entitná množina) obsahuje stĺpec pre cudzí kľúč. Cudzí kľúč je primárny kľúč druhej participujúcej tabuľky. Takto je možné modelovať vzťah, keď je na jednej strane vzťahu jeden záznam a na strane druhej niekoľko záznamov (vzťah „jeden k mnohu“ – 1:M). Tabuľka obsahujúca cudzí kľúč je na strane „mnoho“ a tabuľka pre ktorú je cudzí kľúč kľúčom primárnym je na strane „jeden“. Jednej inštancie vzťahu sa zúčastňujú záznamy z prvej tabuľky s rovnakým cudzím kľúčom a záznam s vyhovujúcim primárnym kľúčom z tabuľky druhej.

Ostatné typy vzťahov ako 1:1 a M:M, prípadne aj s požadovaným obmedzením na povinnosť resp. nepovinnosť vzťahu, sú v relačnej databáze obvykle tiež ukladané vo forme ako by išlo o vzťah 1:M (v prípade M:M aj s umelo zavedenou asociačnou tabuľkou) a aby boli dodržané vlastnosti týchto vzťahov, sú použité kontrolné opatrenia, ako napríklad databázový trigger.

Pri vytváraní ERD z databázovej schémy, je teda možné zjednodušiť notáciu, ktorá je používaná pre vzťahy. Forma reprezentácie vzťahu 1:M v relačnej databáze naznačuje, že tabuľky na seba navzájom ukazujú. O vzťahu teda môžeme povedať, že je istým spôsobom orientovaný. Tento fakt je možné pri notácii zohľadniť takým spôsobom, že vzťahy budú znázornené ako orientované hrany smerujúce od tabuľky s cudzím kľúčom k tabuľke na ktorú tento cudzí kľúč ukazuje. Takýmto spôsobom vznikne z databázovej schémy orientovaný graf.

Hlavný problém pri automatizovanom vytváraní diagramu schémy je určenie rozloženia entít, ktoré schéma obsahuje. Tu sa vytvára priestor pre aplikovanie algoritmov pre vykresľovanie grafov. Je potrebné zvoliť vhodný spôsob vykreslenia výsledného diagramu tak, aby bol dobre použiteľný, dostatočne prehľadný a názorný.

3 Prehľadné vykresľovanie grafov

Graf je v matematike a informatike algebraická štruktúra, ktorá pozostáva z kolekcie vrcholov (uzlov) a kolekcie hrán. Hrany spájajú vrcholy. Graf G je teda dvojica (V, E) , kde V je množina obsahujúca vrcholy a E je množina obsahujúca hrany. Hrany v množine E sú dvojice (e, f) , kde e a f sú vrcholy z množiny V ($e, f \in V$). Graf môže byť použitý na reprezentáciu istej vzťahovej štruktúry, napríklad aj v predošlej kapitole popísanej, databázovej schémy, kde tabuľky sú vrcholy a relácie sú hrany.

Vykresľovanie grafov (graph drawing, graph layout) do roviny je odnož teórie grafov, zaoberajúca sa možnosťami geometrickej reprezentácie grafov do roviny. Poznatky o vykresľovaní grafov do roviny sú využívané napríklad pri vykresľovaní schém, navrhovaní VLSI (Very Large Scale Integration, veľmi veľká integrácia) obvodov, kartografii, bioinformatike, sociálnych sieťach a všeobecne všade tam, kde je potrebné vizualizovať informácie.

Nakreslenie grafu je zobrazenie, v ktorom sa mapujú vrcholy grafu na body v rovine a vytváranie kriviek spájajúcich body, na ktoré sú mapované koncové vrcholy hrán grafu. Existuje niekoľko algoritmov, ktoré sú určené na generovanie rôznych typov nakreslení grafov.

Graf je možné do roviny nakresliť nekonečne veľa spôsobmi. Pri vytváraní nakreslenia je dôležité dbať aj na isté estetické požiadavky.

3.1 Estetika vykresľovania grafov

Pri dizajne algoritmov produkujúcich nakreslenie je dobré si uvedomiť, že to ako bude výsledok vyzeráť značne ovplyvňuje aj schopnosť grafu poskytovať informácie prijímateľovi zrozumiteľne tak, aby sa dali zapamätať a prakticky využívať. Preto je užitočné vedieť určiť estetické požiadavky, ktorých splnenie vedie ku kvalitnejšiemu nakresleniu.

Typicky sa za esteticky významné ukazovatele pri kreslení grafov považuje:

- počet krížení hrán grafu,
- počet ohybov, ktoré sa na hrane grafu vyskytujú,
- rovnomerné rozloženie vrcholov,
- veľkosť uhlov medzi hranami grafu,
- ortogonalita nákresu, mapovanie nákresu na ortogonálnu mriežku,
- symetria vrcholov grafu,
- veľkosť nákresu grafu.

Optimalizovať naraz viaceré kritériá v plnej miere je vo všeobecnosti pri vykresľovaní grafu často nemožné. Napríklad dodržanie minimálneho počtu krížení hrán v grafe často vylučuje možnosť maximálnej symetrie grafu. Ako demonštrácia tohto faktu môže poslúžiť úplný graf K_4 , obr. 3.1.



Obr. 3.1: Kríženie hrán verzus symetria na grafe K_4

Bolo vykonaných niekoľko experimentov zameraných na estetickú stránku vykresľovania grafov [16, 17]. Cieľom týchto experimentov bolo zistiť a poskytnúť prehľad o tom, ktoré estetické kritériá sú dôležité a posúdiť ich podľa toho ako sú schopné napomáhať ľuďom v porozumení grafom znázorňovanej štruktúry. Takýto prehľad o estetických kritériách je dôležitý pre tvorcov vykresľovacích algoritmov a systémov, aby vedeli, načo je dobré klásť dôraz a ktoré vlastnosti uprednostňovať pri vykresľovaní grafov.

Účastníkom experimentu uvedenom v [17] bolo predložených niekoľko nákresov grafov a položené otázky týkajúce sa konkrétneho vykreslenia. Sledované hypotézy, ktoré mal experiment potvrdiť boli:

- Zvyšujúci sa počet ohybov hrán v nakreslení grafu znižuje pochopiteľnosť grafu.
- Zvyšujúci sa počet krížení hrán v nakreslení grafu znižuje pochopiteľnosť grafu.
- Zvyšovaním lokálnej symetrie v nakreslení grafu sa zvyšuje pochopiteľnosť grafu.

Uvedené hypotézy sa teda zameriavali na tri kritériá, a to na minimalizáciu ohybov hrán, minimalizáciu krížení hrán a na maximalizáciu symetrie v grafe.

Prvé dve hypotézy týkajúce sa ohybov hrán a počtu krížení hrán boli experimentom potvrdené. Hypotéza týkajúca sa symetrie nebola ani potvrdená, ale ani vyvrátená. Ako dôvod neúspechu pri testovaní tejto hypotézy bola uvedená aj pravdepodobne nevhodná voľba metriky pre symetriu. Zvolená metrika pravdepodobne vhodne neodrážala ľudskú schopnosť vnímania symetrie.

Ďalší vykonaný experiment [16] bol doplnený o sledované estetické kritériá. Išlo o maximalizáciu najmenších uhlov medzi hranami vychádzajúcimi zo spoločného uzla a maximalizáciu ortogonalitu grafu. Použitá bola aj lepšia metrika pre symetriu. Cieľom experimentu bolo poskytnúť poradie, ktoré reflektuje prioritu jednotlivých kritérií.

Výsledky experimentu ukázali, že množstvo krížení hrán grafu má výrazne najväčší vplyv na porozumenie štruktúry znázornenej grafom. Za počtom krížení nasledovala minimalizácia počtu ohybov hrán a maximalizácia symetrie grafu, pričom maximalizácia minimálnych uhlov a maximalizácia ortogonalit mali stanovenú najmenšiu prioritu.

3.2 Kríženie hrán v grafe

V texte zaoberajúcom sa estetickými kritériami pre vykreslenie grafov bol počet krížení hrán v grafe uvedený ako najpodstatnejšie kritérium. Vygenerovať nakreslenie grafu obsahujúce minimálny počet krížení hrán však nie je vôbec jednoduché.

Počet krížení hrán grafu alebo priesečníkové číslo grafu (graph crossing number) je jedným z predmetov skúmania teórie grafov.

Problém minimálneho priesečníkového čísla bol prvýkrát popísaný Paulom Turánom v roku 1977 na základe vlastných skúseností počas práce v tehelni [24]. Robotníci v tehelni tlačili po koľajniciach vozíky plné tehál od pecí k skladom. Problém nastával v miestach kde sa koľajnice križovali. Stávalo sa, že vozíky pri prechode cez miesto kríženia podskočili a niekoľko tehál z vozíka vypadlo. Pece a sklady prepojené koľajnicami je možné znázorniť ako úplný bipartitný graf. Turán sa zamyslel, ako je možné zistiť, koľko krížení je v takomto grafe nevyhnutných.

Priesečníkové číslo grafu je v súčasnosti dôležitá veličina. Jej aplikácia je dôležitá okrem vizualizácie diagramatických schém a všeobecne pri automatickom vykresľovaní grafov, hlavne v oblasti tvorby elektronických obvodov VLSI.

Zistiť priesečníkové číslo je vo všeobecnosti NP ťažký problém [10]. Tento problém pravdepodobne ostáva NP ťažký aj v prípade, že sa obmedzí len na istú triedu grafov. Napríklad pre triedu kubických grafov (grafy s vrcholmi stupňa 3) bolo ukázané, že problém je stále NP ťažký [12].

Neexistuje efektívny algoritmus pre výpočet priesečníkového čísla. Existujú však algoritmy, ktoré sú pre zadané pevné k a graf G schopné určiť, či je možné nakresliť G s priesečníkovým číslom rovným k (fixed parameter trackable). Výpočtová náročnosť algoritmov pre výpočet priesečníkového čísla viedla k vzniku algoritmov založených na aproximáciách a heuristikách.

3.3 Postupy pri vykresľovaní grafov

Pri hľadaní spôsobov vytvárania nakreslenia grafov, sú najpodstatnejšie tie, ktoré sú automatizované a dokážu byť deterministické, efektívne a produkujúce požadovaný typ rozloženia grafu. Existuje viacero typov rozložení. Každé rozloženie zdôrazňuje iné charakteristiky grafu.

Grafy sú typicky vykresľované niekoľkými spôsobmi. Najjednoduchšie je, ak je graf vykresľovaný s hranami ako rovnými úsečkami alebo oblúkmi. Vrcholy sú na plochu mapované buď náhodne alebo podľa istého zvoleného vzoru.

Zvolený spôsob mapovania vrcholov môže byť napríklad aj taký, že mapuje vrcholy na jednu alebo viac kružníc, prípadne iných geometrických útvarov. Ďalší spôsob mapovania je mapovanie na ortogonálnu mriežku, kde sú zároveň vykresľované aj hrany grafu ortogonálnym štýlom ako horizontálne a vertikálne úsečky.

V prípade, že sú známe nakreslenia úplných grafov, spĺňajúce požadované estetické charakteristiky, je možné použiť toto nakreslenie a postupne uberať hrany ktoré vykresľovaný graf neobsahuje. Tento spôsob je však obmedzený, pretože znalosť optimálneho rozloženia úplných grafov so zvyšujúcim sa počtom vrcholov klesá, takže je rozumné takto vykresľovať len grafy s malým počtom vrcholov.

Nasledujúca časť sa venuje najpodstatnejšej skupine spôsobov vykresľovania grafov. Je to hierarchický spôsob, ktorý upriamuje pozornosť na hierarchiu existujúcu v grafe, pričom ponúka možnosť relatívne dobre minimalizovať počet krížení hrán grafu a ortogonálny spôsob, ktorý mapuje vrcholy do mriežky a umožňuje jednotným a prehľadným spôsobom reprezentovať hrany.

3.3.1 Hierarchické rozloženie

Jedným z možných typov rozloženia, podľa ktorého sa dá graf zobrazíť je také, ktoré reprezentuje hierarchiu vrcholov alebo nejaký prirodzený tok informácií. V procese tvorby rozloženia sa vrcholy grafu rozdelia na niekoľko skupín. Každá skupina reprezentuje jednu vrstvu v hierarchii. Vrstvy sa následne usporiadajú a vrcholy sa prepoja hranami. Takýmto spôsobom vznikne hierarchia, kde je možné vidieť „nadradené“ a „podradené“ vrcholy.

Sugiyama algoritmus

Populárny algoritmus pre vykresľovanie grafu ako hierarchickej štruktúry je prezentovaný v [19]. Zaužívané meno pre tento algoritmus je „sugiyama“ na základe mena jedného z jeho autorov. Vstupom pre tento algoritmus je orientovaný acyklický graf (hrany majú určený počiatočný a koncový vrchol a graf neobsahuje kružnice). Algoritmus pracuje v štyroch fázach.

Prvá fáza je prípravná a definuje sa v nej adekvátne hierarchia grafu, čo znamená, že sa tu definujú jednotlivé vrstvy a priradujú sa na ne vrcholy. Priradovanie prebieha tak, aby hrany medzi vrcholmi neprechádzali cez viacero vrstiev. V prípade, že je prechod hrany cez viac vrstiev nevyhnutný, na hranu sa vložia „umelé“ vrcholy v miestach, kde hrana pretína jednotlivé vrstvy.

V druhej fáze prebieha optimalizácia počtu krížení hrán. Pre optimalizáciu sa využíva barycentrická metóda. Barycentrum grafu je vrchol s najmenšou vzdialenosťou k ostatným vrcholom grafu. Takýchto vrcholov môže byť v grafe niekoľko. Pre každé dve susediace vrstvy sa pomocou počítania barycentier hľadá permutácia vrcholov, ktorá dovoľuje najmenšie množstvo krížení hrán medzi týmito dvoma vrstvami.

Tretia fáza upravuje dĺžku hrán medzi vrstvami. Dĺžka sa upravuje tak, že sa presúvajú vrcholy na vrstvách. Presúvanie je vykonávané v niekoľkých iteráciách. Následne vo fáze štvrtej sa vykonáva vykreslenie tohto grafu.

Výpočtová zložitosť druhej fázy je $O(n^2)$. Zložitosť tretej fázy závisí od počtu vykonaných iterácií. Zložitosť jednej iterácie je $O(n^2)$, čiže dohromady to je $O(kn^2)$ kde k je počet vykonaných iterácií.

Vzostupná planarizácia

Vzostupná planarizácia [2, 3] je spôsob ako získať nakreslenie grafu s minimálnym počtom krížujúcich sa hrán. Postup je, rovnako ako v prípade sugiyama algoritmu, aplikovateľný na orientovaný acyklický graf. Takéto nakreslenie má počiatočný vrchol v spodnej časti nákresu a hrany z neho vychádzajú smerom nahor. Výsledné nakreslenie je možné považovať za hierarchické, avšak to, čo má vplyv na vytvorenie výslednej hierarchie nie sú dopredu dané obmedzenia, ako to môže byť pri prvej fáze vykonávanej algoritmom sugiyama.

Hierarchia je určovaná počas tvorby nakreslenia zároveň s minimalizáciou krížení. Spojenie tvorby hierarchie a minimalizácie krížení hrán dovoľuje lepšie aproximovať minimálny počet krížení, keďže hierarchiu je možné meniť v záujme minimalizácie počtu krížení.

Pri optimalizovaní počtu krížení sa využíva výpočet vzostupne rovinného podgrafu. Vzostupne rovinný podgraf je taký podgraf daného grafu, ktorý spĺňa podmienku, že je rovinný a je možné ho nakresliť vzostupno planárne. Ideálne je, ak každá ďalšia hrana, ktorá by bola do tohto podgrafu pridaná, túto podmienku poruší, teda aby tento podgraf bol najväčší možný. Hrany, ktoré tvoria rozdiel daného grafu a jeho podgrafu sú pri tvorbe nakreslenia vykresľované takým spôsobom, ktorý zabezpečí čo najmenšie množstvo krížení.

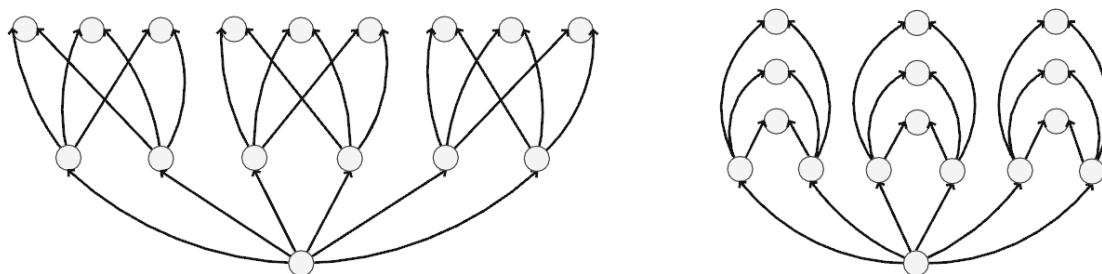
Vo všeobecnosti je vzostupná planarizácia zložená z dvoch fáz. V prvej fáze je vytvorený vzostupne rovinný podgraf. Druhá fáza je určená na doplnenie vytvoreného podgrafu o hrany pôvodného grafu, ktoré ešte neobsahuje, pričom sa sústreďuje na minimalizovanie počtu vzniknutých krížení. Výslednú reprezentáciu zadaného grafu je vhodné následne upraviť. Úprava spočíva v zadefinovaní vrstiev, na ktoré sa vrcholy namapujú a upravajú sa vzdialenosti a miesta ohybov hrán.

Porovnanie sugiyama algoritmu a vzostupnej planarizácie

Experiment prezentovaný v [3] porovnával vzostupnú planarizáciu a sugiyama algoritmus. V experimente bol zohľadnený počet krížení hrán vo výsledných nakresleniach, výška (počet použitých vrstiev) a šírka (maximálny počet vrcholov na jednej vrstve) vygenerovaných nakreslení, ale aj čas, ktorý tieto postupy potrebovali na vygenerovanie výsledku. Výsledky tohto experimentu sú zhrnuté v nasledujúcom texte.

V experimente bolo ukázané, že v minimalizovaní krížení hrán má vzostupná planarizácia jednoznačne lepšie výsledky. Vzostupná planarizácia je teda vhodná pre použitie v prípadoch, že počet krížiacych sa hrán je hlavným kritériom pre vygenerované nakreslenie. Algoritmus sugiyama je vhodné použiť v prípadoch, kedy je účelné mať možnosť vopred predurčiť istú hierarchiu, prípadne jej časť, keďže tento algoritmus najprv zadefinuje vrstvy, priradí k nim vrcholy a toto rozdelenie ostáva v ďalšom spracovaní nemenné.

Porovnávanie výšky (počtu použitých vrstiev) a šírky (maximálneho počtu vrcholov na jednej vrstve) vygenerovaných nakreslení ukázalo, že výška nakreslení vygenerovaných sugiyama algoritmom bola menšia ako pri vzostupnej planarizácii. Dôvodom je lepšie využívanie vrstiev pri minimalizovaní počtu krížení hrán vzostupnou planarizáciou. Výška grafu má zároveň vplyv aj na šírku grafu. Čím viac vrstiev je použitých, tým menej vrcholov je nutné uložiť na jednu vrstvu.



Obr. 3.2: Príklad kedy lepšie určenie vrstiev zabezpečí lepší výsledok v krížení hrán. [2]

V oblasti časovej náročnosti bolo ukázané, že algoritmus sugiyama dokáže vygenerovať nakreslenie v kratšom čase ako vzostupná planarizácia. Čas potrebný pre vygenerovanie nakreslenia grafu na 100 vrchoch bol s použitím algoritmu sugiyama na procesore Intel Pentium M 1,7 GHz a 1 GB RAM pod 0,5 sekundy. Použitie vzostupnej planarizácie, ktorá je všeobecne pomalšia, prinieslo stále prijateľný čas behu pod 1,5 sekundy.

3.3.2 Ortogonálne rozloženie

Ortogonálne rozloženie je také rozloženie, ktoré reprezentuje graf v rovine bez potreby zdôrazniť hierarchiu grafu alebo tok v grafe. Hrany grafu sú v tomto prípade reprezentované horizontálnymi a vertikálnymi čiarami. Pri použití tejto formy rozloženia je potrebné viac brať do úvahy počet ohybov hrán vznikajúcich v miestach, kde hrany menia smer.

Topology-shape-metrics

Algoritmy generujúce ortogonálne rozloženie grafu sú založené na tzv. topology-shape-metrics [20] (topológia-tvar-metrika) prístupe. To rozdeľuje algoritmy na tri fázy.

V prvej fáze prebieha planarizácia zadaného grafu. Znamená to, že sa vytvára nakreslenie, ktoré nedovoľuje hranám aby sa pretínali. Ak je kríženie hrán nevyhnutné, do grafu sa vkladajú v miestach kríženia umelé vrcholy stupňa 4. Výsledná rovinná reprezentácia grafu je vstupom pre ďalšiu fázu.

Druhou fázou je ortogonalizácia, v priebehu ktorej sa planárna reprezentácia tvaruje tak, aby mohla byť vo výsledku namapovaná na ortogonálnu mriežku. Znamená to, že sa upravujú súradnice vrcholov a upravuje sa tvar hrán do podoby horizontálnych a vertikálnych segmentov.

Ortogonalizovaná planárna reprezentácia grafu následne vstupuje do tretej fázy, v ktorej sa upravuje jej metrika. Pozícia vrcholov, dĺžka hrán a miesta ohybov sú zosúladené tak, aby výsledný náčrt mal čo najmenšie odchýlky a vzdialenosti na ňom boli jednotné.

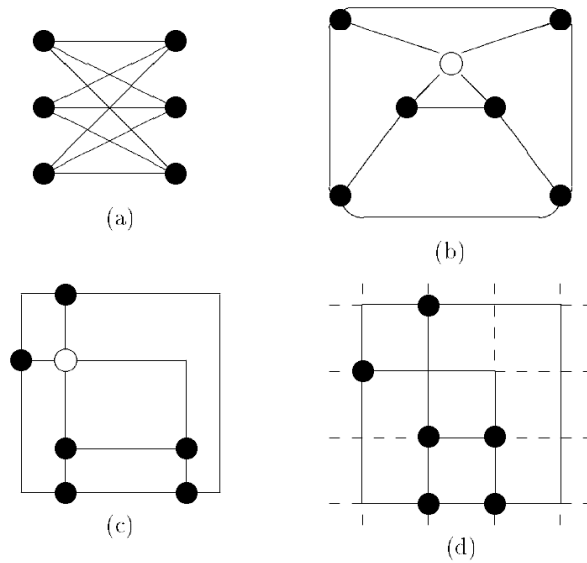
Celý vyššie popísaný proces pre graf $K_{3,3}$ je znázornený na obr. 3.3. Vrchol znázornený bielou farbou v častiach (b) a (c) je umelý, pridaný kvôli odstráneniu kríženia hrán.

Planarizácia

Vykreslenie grafu do roviny je možné v prípade, že zadaný graf je planárny. Podľa Kuratowského vety graf môže byť planárny len v prípade, že žiaden jeho podgraf neobsahuje izomorfné delenie grafu K_5 (úplný graf na piatich vrchoch) alebo $K_{3,3}$ (úplný bipartitný graf s troma a troma vrcholmi).

Ďalší vzťah aplikovateľný na planárne grafy je Eulerova implikácia, ktorá hovorí, že ak je graf G rovinný, tak potom $|V(G)| - |E(G)| + s = 2$, kde s je počet stien (oblastí), ktoré hrany grafu ohraničujú.

Pri planarizácii sa hľadá maximálny rovinný podgraf zadaného grafu. Tento podgraf je možné nájsť v lineárnom čase [9]. Do tohto podgrafu je potrebné späť vložiť chýbajúce hrany spôsobom, ktorý zaručí, že bude minimálny počet pretnutých hrán. Takto je možné hrany do grafu vkladať tiež v lineárnom čase [11] s pomocou SPQR-tree (SPQR-strom) štruktúry [8].



Obr. 3.3: Prístup topology-shape-metrics. (a) Vstupný graf. (b) Planarizácia. (c) Ortogonalizácia. (d) Upravenie rozloženia. [7]

Ortogonalizácia

Najpodstatnejšia časť generovania ortogonálneho rozloženia grafu je práve ortogonalizácia. V tejto fáze sa zortogonalizuje planarizovaná reprezentácia daného grafu a minimalizuje sa počet ohybov v tejto reprezentácii. Minimalizovanie ohybov je založené na metóde toku v sieťach [13] pre zaručenie minimálneho počtu ohybov hrán. Časová náročnosť procesu je $O(n^2 \log n)$.

Tento proces je aplikovateľný na grafy s vrcholmi stupňa maximálne 4, no existujú spôsoby ako toto obmedzenie obísť. Jedným z nich je nahradzovanie vrcholov vyšších stupňov viacerými vrcholmi stupňa maximálne 4 a vo výsledku ich spätne nahradiť pôvodným vrcholom.

3.4 Vhodné rozloženie pre vykresľovanie relačnej databázovej schémy

Táto časť sa zaoberá výberom spôsobu, ktorý je vhodný na generovanie vizuálnej reprezentácie relačnej databázovej schémy. Je nutné aby reprezentácia schémy bola prehľadná a ľahko porozumiteľná. Tiež je potrebné dosiahnuť aby sa adresát v schéme ľahko orientoval a nenarážal na prekážky brániace v plynulom študovaní schémy.

V časti 2.3.2 je uvedené, že relačnú databázovú schému je možné vizuálne znázorniť pomocou zjednodušenej verzie ERD, kde jednotlivé tabuľky schémy sú mapované na entitné množiny. Takýto ERD sa potom stáva orientovaným grafom.

Časť 3.1 oboznamuje o tom, že je dobré dodržiavať pri vykresľovaní grafov isté estetické kritériá z ktorých pravdepodobne najpodstatnejšie je dosiahnutie minimálneho počtu krížení hrán grafu.

Predstavené boli dva spôsoby možného vytvorenia vizuálnej reprezentácie grafu. Oba minimalizovali počet krížení hrán grafu iným spôsobom. Pri hierarchických rozloženiach je to určenie vrstiev, na ktoré sú vrcholy namapované, resp. sú medzi nimi presúvané a ich permutácia na vrstvách. Ortogonálny spôsob najprv vytvára rovinný podgraf, do ktorého následne vhodným spôsobom vkladá chýbajúce hrany.

3.4.1 Experiment

Za účelom porovnania vyššie uvedených prístupov pre vykresľovanie grafov bol vykonaný experiment porovnávajúci tieto prístupy s ohľadom na počet krížení hrán, ktoré boli vo výslednom rozložení prítomné a zároveň s ohľadom aj na čas, ktorý bol potrebný pre výpočet týchto rozložení.

Testovacie dáta pre experiment boli orientované grafy, s počtom vrcholov 10 až 100¹. Grafy s uvedeným počtom vrcholov sú najčastejšie sa vyskytujúci grafmi v praxi [7]. Celkový počet testovaných grafov bol 1277.

Implementácie algoritmov, použitých v experimente, boli z voľne dostupného frameworku OGDF [23].

Experiment bol vykonaný na Linuxovom systéme Ubuntu 10.10 s 512MB RAM, virtualizovanom s využitím softvéru VirtualBox verzie 4.0.4 r70112 v systéme Windows 7 Professional x64 s 2GB RAM a Intel Core2 Duo CPU s taktom 2,27 GHz.

Výsledky experimentu pre počty krížení hrán testovaných grafov sú zhrnuté v grafe na obr. 3.4 a dosiahnuté časové výsledky sú uvedené v grafe na obr. 3.5. Pre grafy s rovnakým počtom uzlov bola výsledná hodnota určená ako aritmetický priemer výsledkov týchto grafov.

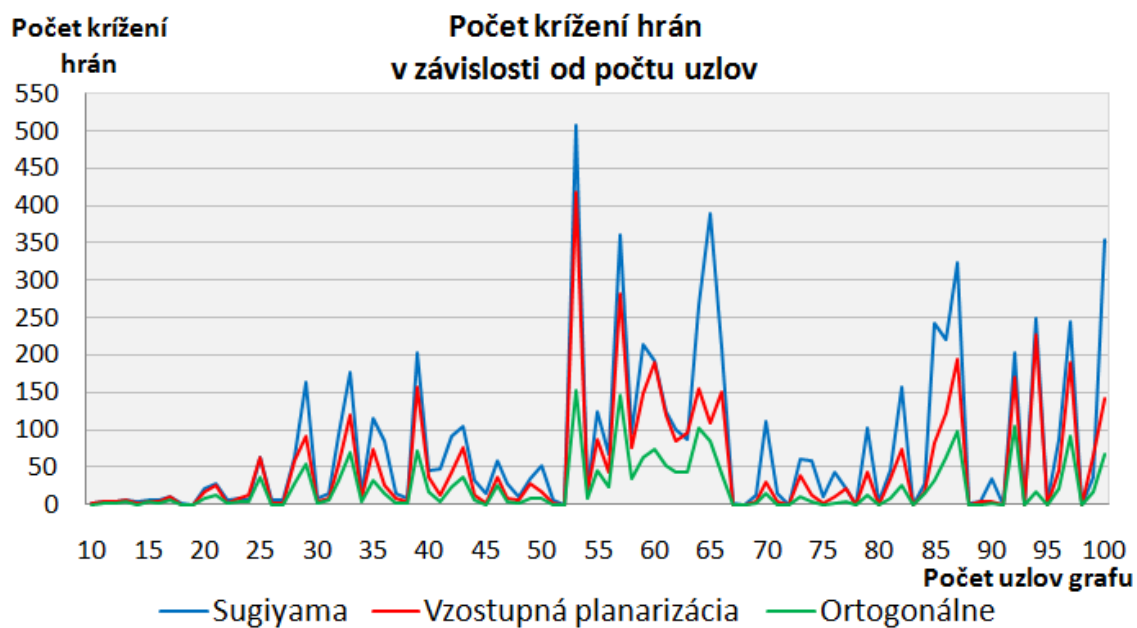
Z výsledkov experimentu v oblasti časovej náročnosti sa dá usúdiť, že algoritmus sugiyama dosahuje najlepšie výsledky. Potom nasleduje vzostupná planarizácia a generovanie ortogonálneho rozloženia sa ukázalo ako časovo najviac náročné. Priemerné a stredné hodnoty času potrebného pre výpočet sú uvedené v tabuľke 3.1.

	priemer [ms]	medián [ms]
Sugiyama	38	23
Vzostupná planarizácia	131	64
Ortogonálne	582	91

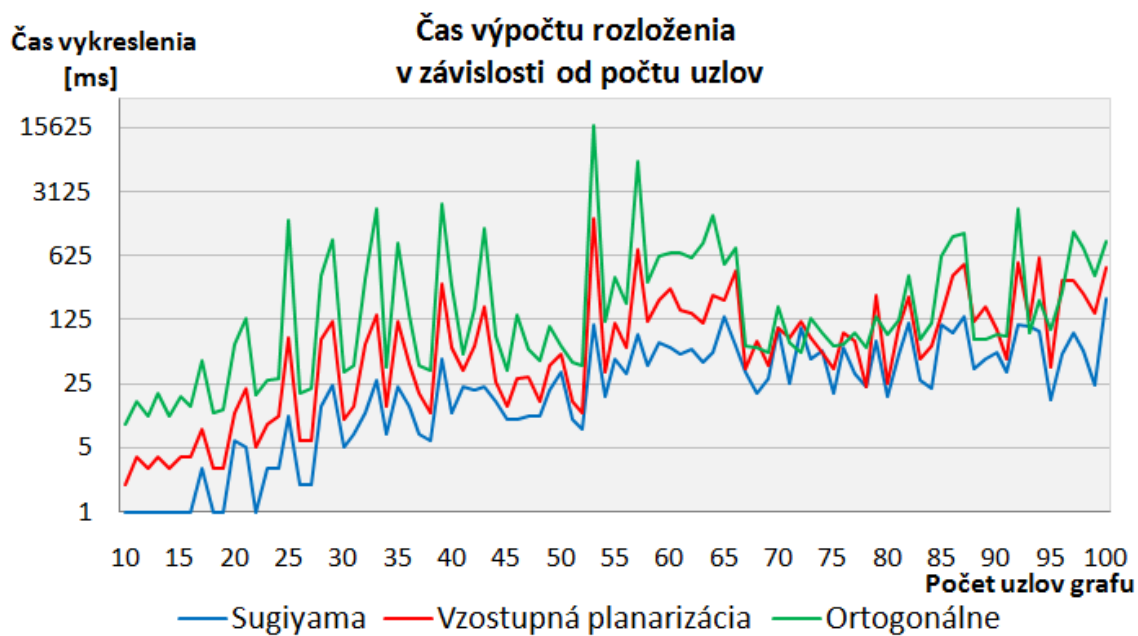
Tabuľka 3.1: Priemerné a stredné hodnoty času potrebného pre výpočet.

V oblasti počtu krížení hrán grafu, je zrejmé, že ortogonálny spôsob, je schopný generovať rozloženie s najnižším počtom krížení hrán. Počet krížení hrán v rozloženiach vygenerovaných algoritmom sugiyama a vzostupnou planarizáciou je nižší pri vzostupnej planarizácii, čo je zároveň konzistentné s výsledkami experimentu uvedenom v [3] a popísanom v časti 3.3.1. Priemerné a stredné hodnoty počtov krížení sú uvedené v tabuľke 3.2.

1. Zbierka grafov použitá pre experiment je voľne dostupná na url <http://graphdrawing.org/download/north-graphml.tgz>



Obr. 3.4: Počet krížení hrán v závislosti od počtu uzlov.



Obr. 3.5: Čas výpočtu rozloženia v závislosti od počtu uzlov.

	priemer	medián
Sugiyama	78	32
Vzostupná planarizácia	50	16
Ortogonalne	23	6

Tabuľka 3.2: Priemerné a stredné hodnoty počtov krížení.

3.4.2 Výber vhodného rozloženia

Na základe vyššie uvedeného experimentu je možné vyvodiť záver, že ortogonálny spôsob vykresľovania grafu je najvhodnejší pre vykresľovanie relačnej databázovej schémy. Nielen že najlepšie aproximuje minimálny počet krížení hrán, ale vygenerované rozloženie poskytuje aj jednotný spôsob pre vykreslenie každej tabuľky schémy.

Vďaka ortogonálnemu spôsobu rozloženia sú tabuľky pri pohľade na vygenerovanú schému rovnocenné. Jednotný spôsob napájania hrán relácií na tabuľky zlepšuje prehľadnosť a zjednodušuje orientáciu v schéme. Minimalizovanie ohybov hrán v ortogonalizačnej fáze algoritmu a jednotnosť ich uhlov prispieva k prehľadnosti a vo výslednej schéme sa preto ľahšie sledujú relácie medzi tabuľkami.

Generovanie ortogonálneho rozloženia dokáže aj dobre optimalizovať počet krížení hrán relácií v schéme, vďaka čomu spĺňa veľmi dôležité estetické kritérium o krížiacich sa hranách v grafe.

Hierarchické rozloženia schémy môžu byť tiež užitočné. Hierarchickým spôsobom vykreslená schéma zvýrazňuje „tok“ relácií a pri pohľade na schému je možné skôr určiť, na ktoré tabuľky daná tabuľka odkazuje. Rozloženie však mapuje tabuľky na horizontálne vrstvy, čo bráni dôkladnej minimalizácii počtu krížení relácií schémy a diktuje istý vzor pre výsledné vykreslenie.

Ďalším obmedzením hierarchických rozložení je to, že sa vytvárajú z acyklických grafov. Databázová schéma však môže reprezentovať aj cyklický graf. Cyklický graf je však možné pred spustením algoritmu pretransformovať na acyklický preorientovaním hrán uzatvárajúcich cyklus a po vykonaní algoritmu vrátiť orientáciu týchto hrán do pôvodného stavu. Výsledný graf po takejto transformácii však už nebude mať jednotný smer „toku“.

Pri hodnotení rýchlosti behu jednotlivých algoritmov sa dá usúdiť, že z pohľadu rýchlosti je najvhodnejšie schému generovať sugiyama algoritmom. Generovanie ortogonálneho nakreslenia má preukázateľne horšie výsledky, avšak dĺžka generovania je s ohľadom na výkon súčasného hardvéru stále prijateľná.

4 Aplikácia pre vizuálnu reprezentáciu schémy relačnej databázy

Táto kapitola sa zaoberá popisom pre účely tejto práce vytvorenej aplikácie pre osobné počítače pre vizuálnu reprezentáciu relačnej databázovej schémy. Aplikácia realizuje vykreslenie tejto schémy formou zjednodušeného entitne-relačného diagramu, kde tabuľky sú reprezentované ako entity diagramu a relácie medzi tabuľkami sú reprezentované ako relácie medzi entitami diagramu formou šípok, ktoré určujú, na ktoré tabuľky daná tabuľka odkazuje.

Kapitola obsahuje časť venovanú vstupnému formátu, ktorý dokáže aplikácii poskytnúť informácie o schéme, potrebné pre jej vykreslenie. Ďalej kapitola obsahuje popis frameworku, použitého pre generovanie vizuálnej reprezentácie schémy a popis vytvorenej aplikácie.

4.1 Vstupný formát pre aplikáciu

V praxi sa používa viacero RDBMS. Každý takýto systém má svoje vlastné špecifiká, ktoré robia tieto systémy navzájom nekompatibilnými. Obvykle však tieto systémy poskytujú službu exportu databázy. Je teda možné exportovať schému, ako aj dáta, zo spravovanej databázy. Formát takéhoto exportu však nemusí byť u každého RDBMS totožný.

Pre elimináciu problémov s formátom, podľa ktorého je relačná databázová schéma exportovaná, bol pre potreby vytvorenej aplikácie navrhnutý vlastný XML formát, ktorý je schopný reflektovať:

- tabuľky schémy,
- stĺpce tabuliek,
- typ stĺpca tabuľky,
- vzťahy medzi tabuľkami.

Formát je schopný uchovávať aj informácie o integritných obmedzeniach pre tabuľky schémy, a to:

- primárne kľúče tabuliek,
- cudzie kľúče tabuliek,
- unikátne hodnoty,
- nulové hodnoty.

4.1.1 Definícia formátu

Schéma vstupného XML formátu je definovaná pomocou XSD (XML Schema Definition, definícia XML schémy), ktorá definuje množinu pravidiel. Tieto pravidlá musí XML dokument spĺňať aby mohol byť považovaný za platný.

Kód 4.1 predstavuje definíciu pre XML formát nesúci definíciu relačnej databázovej schémy.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- simple attributes declaration -->
  <xs:attribute name="id" type="xs:integer"/>
  <xs:attribute name="name" type="xs:string"/>
  <xs:attribute name="description" type="xs:string"/>
  <xs:attribute name="type" type="xs:string"/>
  <!-- elements declaration -->
  <xs:element name="constraint">
    <xs:complexType>
      <xs:attribute name="type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="primary"/>
            <xs:enumeration value="foreign"/>
            <xs:enumeration value="unique"/>
            <xs:enumeration value="null"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="references" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="column">
    <xs:complexType>
      <xs:sequence minOccurs="0">
        <xs:element ref="constraint" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute ref="name" use="required"/>
      <xs:attribute ref="description"/>
      <xs:attribute ref="type" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="table">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="column" maxOccurs="unbounded" minOccurs="1"/>
      </xs:sequence>
      <xs:attribute ref="id" use="required"/>
      <xs:attribute ref="name" use="required"/>
    </xs:complexType>
  </xs:element>
  <!-- root element -->
  <xs:element name="schema">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="table" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="name" use="required"/>
      <xs:attribute ref="description"/>
    </xs:complexType>
    <xs:unique name="uniqueTableId">
```

```

    <xs:selector xpath="table" />
    <xs:field xpath="@id" />
  </xs:unique>
  <xs:unique name="uniqueTableName">
    <xs:selector xpath="table" />
    <xs:field xpath="@name" />
  </xs:unique>
</xs:element>
</xs:schema>

```

Kód 4.1: XML schéma pre relačnú databázovú schému.

Obsah ukážkového XML dokumentu, spĺňajúceho požiadavky definované XML schémou (kód 4.1) je poskytnutý v kóde 4.2.

```

<?xml version="1.0"?>
<schema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="defaultNamespace schema.xsd"
  name="Library"
  description="demonstration schema">
  <table id="0" name="Books">
    <column name="id" type="INTEGER">
      <constraint type="primary" />
    </column>
    <column name="title" type="VARCHAR2(15)" />
    <column name="year_of_release" type="SMALLINT" />
    <column name="ammount" type="SMALLINT" />
  </table>
  <table id="1" name="Authors">
    <column name="id" type="INTEGER">
      <constraint type="primary" />
    </column>
    <column name="name" type="VARCHAR2(15)" />
  </table>
  <table id="3" name="Works">
    <column name="author" type="INTEGER">
      <constraint type="primary" />
      <constraint type="foreign" references="1" />
    </column>
    <column name="book" type="INTEGER">
      <constraint type="primary" />
      <constraint type="foreign" references="0" />
    </column>
  </table>
</schema>

```

Kód 4.2: Jednoduchá demonštračná schéma

4.2 Externá funkcionálna využívaná aplikáciou

4.2.1 Framework pre automatizované vykresľovanie grafov

Algoritmy generujúce rozloženia grafov sú často veľmi zložité a vyžadujú vysokú úroveň znalostí v oblasti teórie grafov. Implementácia týchto algoritmov nie je tiež triviálnou záležitosťou a vytvára veľký priestor pre výskyt chýb rôzneho druhu. Existujú však knižnice resp. frameworky, ktoré poskytujú implementácie viacerých vykresľovacích algoritmov, ktoré je možné ľahko použiť.

Medzi známe knižnice a frameworky umožňujúce generovanie nakreslení grafov patria LEDA, GDTToolkit, GoVisual a OGDF (Open Graph Drawing Framework).

LEDA, GDTToolkit a GoVisual sú komerčne licencované. LEDA a GDTToolkit sú určené pre C++. GoVisual je dostupný pre C++, Java, .NET a COM. OGDF je dostupný voľne a licencovaný pod GPL. Tento framework je určený pre programovací jazyk C++ a získal aj niekoľko ocenení [22], čo uisťuje o jeho kvalite.

Pre použitie vo vytvorenej aplikácii bol zvolený OGDF, a to hlavne vďaka jeho voľnej licencií, dostupnosti pre platformy Linux aj Windows a aktívnemu diskusnému fóru na jeho domácich webových stránkach.

Algoritmy pre vygenerovanie rozložení grafov v OGDF sú schopné spoľahlivo vypočítať všetky potrebné súradnice pre vrcholy a ohyby hrán. Framework však nie je schopný vlastného vykresľovania. Vygenerované rozloženia je však schopný exportovať do textového formátu GML.

Vykresľovanie grafov uložených vo formáte GML dokáže realizovať jednoduchý nástroj gml2pic [21]. Tento nástroj je založený na OGDF a dokáže načítať graf z GML a uložiť jeho grafickú podobu do rôznych bitmapových (PNG, JPEG, TIFF) a vektorových (SVG, PDF, EPS) formátov.

Nástroj gml2pic je licencovaný pod GPL a je dostupný spolu so zdrojovými kódmi v jazyku C++. Vo vytvorenej aplikácii sú zdrojové kódy tohto nástroja využité pre realizáciu nakreslenia schémy.

4.2.2 Grafické používateľské rozhranie

Aplikácie pre osobné počítače zvyčajne poskytujú grafické používateľské rozhranie (Graphical User Interface, GUI) pomocou ktorého používateľ s aplikáciou komunikuje. Keďže je framework pre prácu s grafmi, vybraný pre aplikáciu, písaný v jazyku C++, bolo nutné vybrať vhodný nástroj pre vytvorenie GUI, ktorý by sa dal použiť v kombinácii s použitým programovacím jazykom.

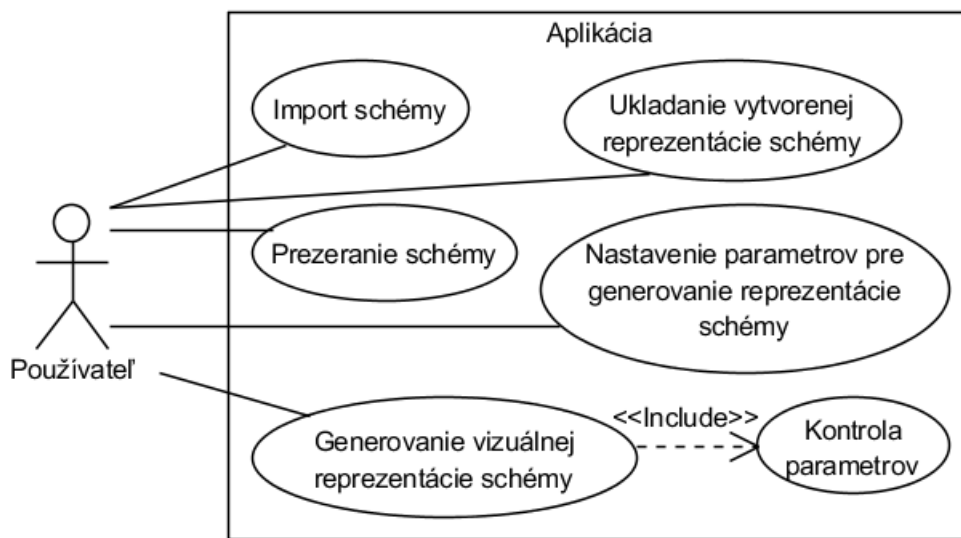
Qt je platformovo nezávislý framework pre aplikácie a tvorbu GUI, určený pre jazyk C++ [5]. Tento framework zároveň obsahuje veľké množstvo užitočných tried pre vývoj aplikácií.

Vytvorená aplikácia používa pre GUI prostredie Qt verzie 4.7.0 a je v nej využitý napríklad aj XML parser spolu s XML Schema validátorom, ktoré sú dostupné práve v rámci Qt. Aplikácia taktiež využíva Qt funkcionálnu pre prácu so súborami a grafickými formátmi.

4.3 Funkcie aplikácie

Aplikácia svojmu používateľovi umožňuje vygenerovanie vizuálnej reprezentácie relačnej databázovej schémy, na základe vstupného XML súboru platného podľa XML schémy definovanej v kóde 4.1. Používateľ má možnosť definovať niektoré parametre pre generovací algoritmus a pre vizuálnu podobu vygenerovanej reprezentácie poskytnutej schémy.

Jednotlivé prípady použitia aplikácie sú zhrnuté v diagrame prípadov použitia, uvedenom na obr. 4.1.



Obr. 4.1: Diagram prípadov použitia.

4.3.1 Špecifikácia prípadov použitia

Import schémy

Prípado použitia začína, keď používateľ aplikácii zadá súbor so schémou. Aplikácia overí, či štruktúra súboru vyhovuje požiadavkám definovaným XML schémou 4.1. V prípade, že zadaný súbor schéme nevyhovuje, aplikácia zobrazí používateľovi chybové hlásenie o zlom formáte vstupného súboru. V prípade, že zadaný súbor schéme vyhovuje, aplikácia zobrazí informácie o schéme, a to názov schémy, počet tabuliek a počet relácií.

Nastavenie parametrov pre generovanie reprezentácie schémy

Prípado použitia začína, keď používateľ úspešne zadá aplikácii súbor so schémou a následne spresňuje parametre pre algoritmy generujúce grafickú reprezentáciu schémy.

Používateľ si tu vyberá jeden z dvoch spôsobov rozloženia. Prvý spôsob je ortogonálny a druhý spôsob je hierarchický. Predvolená a odporúčaná voľba je ortogonálny spôsob.

V prípade zvolenia ortogonálneho spôsobu rozloženia, môže používateľ špecifikovať minimálnu požadovanú dĺžku hrán relácií. Predvolená hodnota je 50 pixelov.

V prípade zvolenia hierarchického spôsobu rozloženia, môže používateľ špecifikovať minimálnu vzdialenosť medzi vrstvami rozloženia. Predvolená hodnota je 30 pixelov. Ďalej môže používateľ špecifikovať minimálnu vzdialenosť tabuliek susediacich na vrstvách. Predvolená hodnota je 30 pixelov.

Používateľ môže špecifikovať aj farby použité pre vykreslenie tabuliek a relácií vo formáte „#RRGGBB“. Predvolená farba pre vykresľovanie tabuliek je „#FFFFFF“, teda biela a pre relácie „#000000“, teda čierna.

Posledné parametre, ktoré môže používateľ špecifikovať sú požiadavky či sa majú vo vykreslenej schéme zobraziť aj názvy stĺpcov tabuliek a názvy ich typov. Predvolené hodnoty sú „áno“.

Kontrola parametrov

Prípád použitia je spustený vrámci prípadu použitia „Generovanie vizuálnej reprezentácie schémy“. Tento prípad použitia slúži pre kontrolu parametrov pre algoritmy generujúce vizuálnu reprezentáciu zadanej schémy. V prípade, že je niektorý parameter chybné zadaný, aplikácia používateľovi zobrazí chybové hlásenie, ktoré ho na chybný parameter upozorní.

Generovanie vizuálnej reprezentácie schémy

Prípád použitia začína, keď používateľ spustí generovanie reprezentácie schémy. Na začiatku tohto prípadu použitia je spustený najprv prípad použitia „Kontrola parametrov“. V prípade, že sú všetky zadané parametre akceptovateľné, aplikácia spustí požadovaný algoritmus pre generovanie reprezentácie a zobrazí vygenerovanú reprezentáciu schémy.

Prezeranie schémy

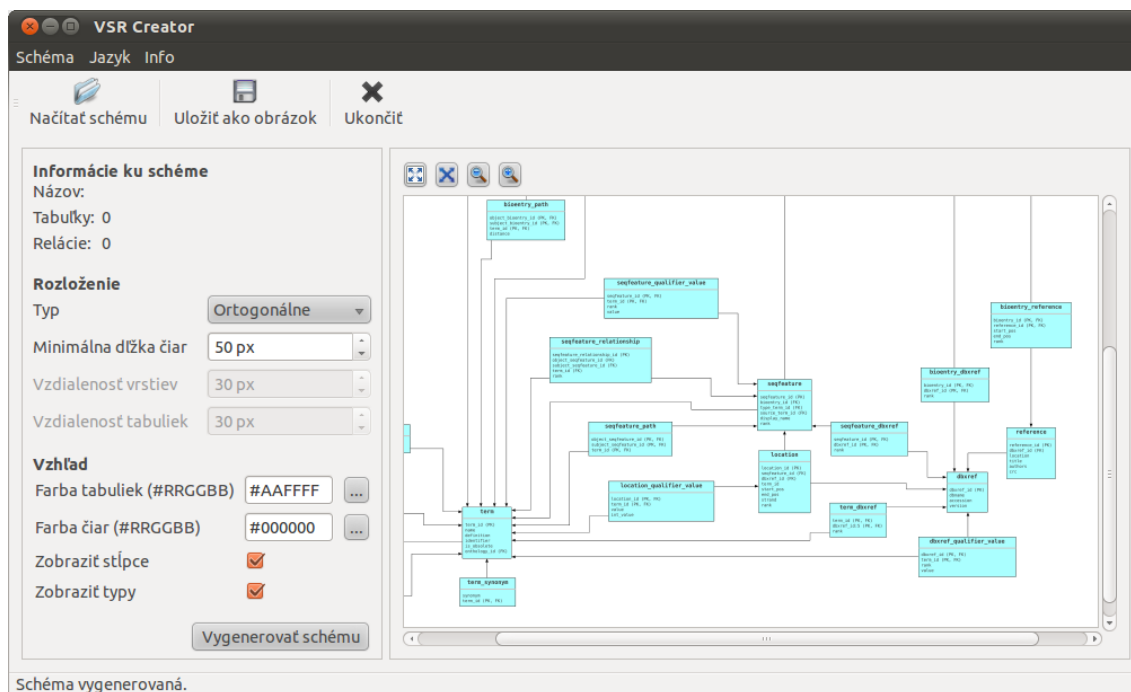
Prípád použitia začína po vygenerovaní reprezentácie schémy. Aplikácia umožní používateľovi prezeranie vygenerovanej reprezentácie. Počas prezerania má používateľ k dispozícii funkcie na „priblíženie a oddialenie“ reprezentácie, funkciu pre nastavenie pôvodnej veľkosti reprezentácie a funkciu pre nastavenie veľkosti reprezentácie tak, aby celá vyplnila oblasť určenú pre zobrazovanie.

Ukladanie vytvorenej reprezentácie schémy

Prípád použitia začína, keď používateľ spustí uloženie vygenerovanej reprezentácie zadanej schémy. Aplikácia skontroluje, či existuje vygenerovaná reprezentácia. V prípade, že neexistuje žiadna vygenerovaná reprezentácia, aplikácia zobrazí používateľovi chybové hlásenie s informáciou o nemožnosti uloženia reprezentácie schémy, pretože žiadna nebola vytvorená. V prípade, že vygenerovaná reprezentácia existuje, aplikácia požiada používateľa aby špecifikoval výstupný súbor. Ak je špecifikovaný výstupný súbor, aplikácia do tohto súboru uloží vygenerovanú reprezentáciu schémy v bitmapovom formáte PNG.

4.4 Grafické používateľské rozhranie aplikácie

Táto časť popisuje GUI aplikácie, ktoré je vytvorené s využitím Qt frameworku. Ukážkový snímok spustenej aplikácie zobrazuje obr. 4.2.



Obr. 4.2: Snímok spustenej aplikácie.

Používateľské rozhranie pozostáva z jedného hlavného okna, ktoré obsahuje všetky potrebné ovládacie prvky pre celú komunikáciu používateľa s aplikáciou. Okno obsahuje lištu s hlavným menu, ktoré obsahuje odkazy na najzákladnejšie funkcie aplikácie.

Hlavné menu, pod zložkou „Schéma“, obsahuje spúšťače akcií pre načítanie novej schémy („Načítať schému“), uloženie vygenerovanej reprezentácie schémy („Uložiť ako obrázok“) a odkaz pre ukončenie aplikácie („Ukončiť“). Pod zložkou „Jazyk“ sa nachádzajú spúšťače akcií pre zmenu jazyka aplikácie (možnosti „Slovensky“ a „Anglicky“). Nakoniec menu zložky „Info“ obsahuje spúšťač „O aplikácii“ spúšťajúci dialógové okno obsahujúce stručné informácie o aplikácii.

Pod hlavným menu sa nachádza lišta so spúšťačmi vybraných akcií, a to akcie pre načítanie novej schémy, uloženie vygenerovanej reprezentácie a ukončenie aplikácie.

Obsah hlavného okna je delený vertikálne na dve hlavné časti. Ľavá časť obsahuje tri skupiny prvkov a tlačidlo pre spustenie generovania reprezentácie schémy. Prvá skupina je určená na zobrazovanie informácií o aktuálne načítanej schéme. Druhá skupina obsahuje prvky pre nastavovanie typu algoritmu pre generovanie reprezentácie schémy a jeho parametrov. Tretia skupina obsahuje prvky pre nastavovanie farieb tabuliek a relácií a prvky pre nastavenie možnosti či sa vo výslednej reprezentácii majú zobrazovať aj stĺpce tabuliek a ich typy.

Pravá časť obsahu hlavného okna je určená pre realizovanie zobrazenia vygenerovanej reprezentácie schémy. V hornej časti tejto oblasti sa nachádzajú tlačidlá pre manipuláciu veľkosti vykreslenej schémy a vo zvyšku oblasti sa zobrazuje vygenerovaná schéma.

4.5 Implementácia

4.5.1 Platforma a použitý software

Aplikácia bola implementovaná pre platformu GNU Linux. Táto platforma bola zvolená pre jej dostupnosť vďaka voľnej licencií GPL, ale aj preto, že poskytuje prehľadné a zrozumiteľné prostredie pre vývoj aplikácií takéhoto rozsahu. Pre realizovanie implementácie aplikácie bola zvolená linuxová distribúcia Ubuntu 10.10 vďaka jej používateľskej prívetivosti.

Pre editáciu a správu zdrojových kódov bolo použité integrované vývojové prostredie (Integrated Development Environment, IDE) Qt Creator 2.0.1. Toto IDE je vhodné pre použitie pri vývoji aplikácií založených na Qt frameworku. Jedným z hlavných praktických dôvodov rozhodnutia bola možnosť využiť vstavaný dizajnér pre vizuálny návrh komponent GUI.

Informácie o projekte (potrebné zdrojové a hlavičkové súbory, externé závislosti a podobne) ukladá Qt Creator do projektového súboru (súbor s príponou „.pro“). Tento súbor je podstatný pre generovanie súboru nazývaného „makefile“, potrebného pre skompilovanie a linkovanie výsledného spustiteľného súboru.

Pre vytvorenie „makefile“ z projektového súboru sa používa nástroj QMake dodávaný spolu s Qt SDK (Software Development Kit, sada nástrojov pre vývoj softvéru). Verzia QMake dodávaná s použitou verziou Qt SDK je 2.01a.

Kompilácia a linkovanie aplikácie na základe vygenerovaného „makefile“ je vykonávaná nástrojom GNU Make vo verzii 3.81. Tento nástroj riadi generovanie binárnych súborov podľa cieľov definovaných v „makefile“.

Keďže aplikácia je písaná v programovacom jazyku C++, pre jej vytvorenie je potrebný aj kompilátor tohto jazyka. Zvolený bol kompilátor G++ dostupný v rámci GCC (GNU Compiler Collection, GNU kolekcia kompilátorov) vo verzii 4.4.5.

Použitá verzia OGDF pre prácu s grafmi a tvorbu ich rozložení bola 2010.10 (Sassafras) a verzia vykresľovacieho nástroja gml2pic, 1.1.

Pre lokalizovanie aplikácie do slovenčiny a angličtiny bola použitá prekladateľská aplikácia Qt Linguist a pomocné nástroje lupdate a lrelease. Všetky tieto nástroje boli vo verzii 4.7.0.

Všetok vyššie spomenutý softvér použitý pri implementácii vytvorenej aplikácie je voľne dostupný zo základných repozitárov distribúcie Ubuntu 10.10.

4.5.2 Spracovanie vstupu

Spracovanie a validácia XML

Aplikácia získava vstupné údaje pomocou XML formátu definovanom v časti 4.1. Tento formát je spravidla spracovávaný XML parsermi. Existujú dva základné typy XML parserov, ktorými sú SAX a DOM.

SAX parsery (udalosťami riadené parsery) prechádzajú XML dokument a vyvolávajú udalosti podľa toho na čo v dokumente narazia. Sú vhodné v prípade, že je potrebné sústrediť sa na rýchlosť spracovávania a nie je nutné pamätať si kontext dokumentu. V prípade, že je potrebné navigovať sa v dokumente, stáva sa použitie SAX parserov náročnejším.

DOM parsery prechádzajú XML dokument a jeho obsah si pamätajú pomocou stromovej štruktúry. Vďaka tejto lokálne udržiavanej štruktúre je možné s DOM parsermi pohodlnejšie pracovať, keďže je možné sa v štruktúre navigovať, upravovať ju a následne exportovať do súboru. Pre takúto manipuláciu je potrebné spracovať zdrojový XML súbor len jedenkrát, avšak je nutné mať k dispozícii viac operačnej pamäte, do ktorej je potrebné vytvorenú stromovú štruktúru uložiť.

Aplikácia potrebuje ešte pred spracovaním vstupu vedieť skontrolovať, či používateľom zadany súbor s databázovou schémou naozaj vyhovuje požiadavkám a či je možné na základe jeho obsahu schému vygenerovať.

Vstupný XML súbor musí byť v prvom rade dobre formátovaný (well formed). To znamená že použité otváracie značky použitých XML elementov sú párované s uzatvárajúcimi značkami a elementy sú do seba adekvátne vnárané takým spôsobom, že sa otváracie a uzatváracie značky neprekrývajú.

Podstatnejšie kritérium, ktoré musí vstupný súbor spĺňať je to, či je jeho obsah platný podľa požadovanej definície XML schémy. Ak dokument platný nie je, svedčí to o tom, že neobsahuje všetky aplikáciou požadované informácie potrebné pre generovanie schémy a aplikácia preto nemôže zaručiť správne vygenerovanie schémy.

Pre kontrolu dobrého formátovania a platnosti dokumentu sa používajú XML validátory, ktoré sú schopné rozhodnúť, či XML dokument potrebné požiadavky spĺňa.

Riešenie v aplikácii

Pre spracovanie vstupného XML súboru je v aplikácii použitý XML parser typu DOM. Použitá implementácia tohto parsera je z Qt frameworku implementovaná QDom triedami z modulu Xml. Tento parser bol vybraný pre jeho dostupnosť v rámci Qt frameworku a jednoduchosť použitia. Zároveň poskytuje potenciál pre možné rozšírenie aplikácie, ktoré by mohlo poskytovať funkcie pre editáciu zobrazovanej schémy.

Funkcionalita, ktorá vykonáva spracovávanie vstupu pre aplikáciu je implementovaná v triede `SchemaParser`. Prototypy verejných metód poskytnutých touto triedou sú uvedené v kóde 4.3.

```
void readFromFile(QFile& file);
void parseXMLToGraphAttributes(GraphAttributes& schemaGraphAttributes,
                               Graph& schemaGraph,
                               bool columns, bool types);
int numberOfTables() const;
int numberOfRelations() const;
QString schemaName() const;
```

Kód 4.3: Prototypy verejných metód triedy `SchemaParser`

Podstatnou metódou triedy `SchemaParser` je `parseXMLToGraphAttributes()`, ktorá transformuje informácie zo vstupného XML dokumentu, do podoby grafu a napĺňa objekt s atribútmi grafu. Graf a jeho atribúty sú metóde poskytované referenciou. Bližšie informácie k týmto štruktúram poskytuje nasledujúca časť 4.5.3. Metóda dostáva aj dva parametre typu `bool`, ktoré metóde určujú, či sa má zaoberať spracovávaním stĺpcov tabuliek a ich typov podľa toho či je požadované ich zobrazenie alebo nie.

Qt framework poskytuje aj implementáciu XML validátora schopného potvrdiť platnosť dokumentu podľa danej XML schémy. Táto implementácia je dostupná z Qt modulu `XmlPatterns` a v aplikácii ju využíva trieda `SchemaValidator`, ktorá obsahuje verejnú statickú metódu pre validovanie súboru. Prototyp tejto metódy je uvedený v kóde 4.4

```
static bool isValid(QFile& schemaFile);
```

Kód 4.4: Prototypy statickej verejnej metódy triedy `SchemaValidator`

4.5.3 Tvorba a realizovanie vykresľovania reprezentácie schémy

Základné rysy OGDF

Pre generovanie rozloženia schémy bol použitý framework OGDF, ktorý poskytuje potrebné algoritmy a dátové štruktúry pre prácu s grafmi.

Framework poskytuje triedu `Graph` pre reprezentáciu grafu a jeho hrán a vrcholov. Táto trieda neuchováva žiadne informácie o rozložení grafu (pozícia, veľkosť alebo tvar vrcholov). Pre účely ukladania týchto informácií slúži trieda `GraphAttributes`, ktorá si zároveň ukladá aj referenciu na graf, ktorého informácie obsahuje. Dôležitá metóda tejto triedy je `constGraph()`, ktorej návratová hodnota je práve konštantná referencia na tento zdrojový graf.

Jednotlivé typy rozložení grafov, ktoré tento framework obsahuje sa skladajú z modulov, ktoré poskytujú algoritmy pre jednotlivé fázy tvorby rozloženia. Tieto moduly sú triedy implementujúce požadované rozhranie (typicky `LayoutModule`, podmoduly môžu mať vlastné rozhranie s ďalšími požiadavkami), ktoré predpisuje implementovanie metódy `call(GraphAttributes&)` s parametrom typu `GraphAttributes`, ktorý obsahuje potrebné informácie o grafe. Prítomnosť týchto rozhraní umožňuje programátorom použiť pre niektoré fázy algoritmov vlastné implementácie, čo je výhoda hlavne pri optimalizácii a vývoji nových modulov

Riešenie v aplikácii

Aplikácia využíva dva typy rozložení, a to hierarchické a ortogonálne, ktoré boli popísané v časti 3.3. Pre hierarchické rozloženie bol použitý algoritmus sugiyama, implementovaný v triede `SugiyamaLayout`. Tento algoritmus pozostáva z troch fáz (presnejšie zo štyroch, ale keďže OGDF nevykonáva vlastnú realizáciu vykreslenia, čo je štvrtá fáza, algoritmus je zložený z troch fáz). Pre každú z týchto fáz je použitý modul, ktorý ju vykonáva.

Modul zvolený pre prvú fázu sugiyama algoritmu (vytvorenie hierarchie vrcholov) je `RankingModule`. Pre druhú fázu je použitý modul `BarycenterHeuristic` a tretia fáza je vykonávaná modulom `FastHierarchyLayout`. Tretí modul umožňuje nastaviť

niekoľko parametrov. Aplikácia pre tento modul definuje minimálnu vzdialenosť vrstiev a minimálnu vzdialenosť vrcholov na jednej vrstve.

Proces tvorby ortogonálneho rozloženia je o čosi zložitejší. V planarizačnej fáze sa pre tvorbu maximálneho planárneho podgrafu využíva modul `FastPlanarSubgraph` a pre nasledovné vkladanie odstránených hrán modul `VariableEmbeddingInserter`. Modul, podľa ktorého je vytvorená výsledná planarizovaná reprezentácia zadaného grafu je `EmbedderMinDepthMaxFaceLayers`. Ortogonalizácia tejto reprezentácie je vykonávaná modulom `OrthoLayout`. Tento modul dostáva ako parameter minimálnu dĺžku hrán.

Moduly pre jednotlivé fázy oboch algoritmov boli volené tak, aby odrážali poznatky uvedené v časti 3.3. Požadovanú funkcionality zaobahuje trieda `LayoutCreator`, ktorá ako rozhranie poskytuje dve statické verejné metódy. Prototypy týchto metód sú uvedené v kóde 4.5.

```
static void applyOrthoLayout(Graph& G, GraphAttributes& GA,
                           int separation);
static void applyHierarchyLayout(Graph& G, GraphAttributes& GA,
                                int layerDistance, int tableDistance);
```

Kód 4.5: Prototypy statických verejných metód triedy `LayoutCreator`.

Trieda `LayoutCreator` je však len zaobalením funkcionality požadovanej od OGDF. Znamená to, že trieda určí súradnice pre jednotlivé uzly a nevykonáva vykresľovanie. Na tieto účely je využitá funkcionality poskytnutá voľne dostupným nástrojom `gml2pic`. Jadrom tohto nástroja je funkcia `draw()`, ktorá realizuje vykresľovanie zadaného grafu. Jej prototyp je uvedený v kóde 4.6.

```
void draw(const ClusterGraphAttributes &cga, QPainter &painter,
          const Options &opt);
```

Kód 4.6: Prototypy funkcie nástroja `gml2pic` realizujúcej vykresľovanie grafu.

Funkcia preberá pri svojom volaní tri parametre. Prvý parameter je referencia na objekt triedy `ClusterGraphAttributes`, ktorá dedí z triedy `GraphAttributes`. Ako druhý parameter funkcia dostáva referenciu na objekt typu `QPainter`. Táto trieda Qt frameworku poskytuje rozhranie pre vykresľovanie na zariadenie, na ktoré je jej inštancia napojená. Tretí parameter je štruktúra, ktorá obsahuje potrebné informácie, ktoré táto funkcia potrebuje pre realizovanie kreslenia.

V prípade vytvorenej aplikácie, graf ktorý je funkciou `draw()` poskytovaný, reprezentuje zadanú schému a jej rozloženie. Druhý parameter, objekt typu `QPainter`, poskytovaný tejto funkcii je napojený na objekt triedy `QSvgGenerator`, ktorý uchováva vo formáte SVG to, čo naň bolo vykreslené. Takto vytvorené SVG je následne používateľovi zobrazené pomocou GUI.

Aplikácia poskytuje používateľovi aj možnosť uložiť vygenerovanú a zobrazenú reprezentáciu schémy ako obrázok vo formáte PNG. Táto funkcionality je implementovaná podobne ako vykresľovanie reprezentácie schémy. Objekt typu `QPainter` je však napojený na objekt typu `QImage` ktorý všeobecne reprezentuje obrázok. Po zavolaní funkcie

`draw()` je na tomto obrázku zavolaná metóda `save()`, ktorej prvým parametrom je používateľom zadaný názov súboru do ktorého sa má obrázok uložiť a druhým parametrom je formát tohto obrázka, ktorý je nastavený na PNG.

Pri ukladaní schémy ako obrázka, je nutné si uvedomiť, že v prípade veľkých schém (300 tabuliek a viac) zaberá vygenerovaný obrázok plochu typicky viac ako 7000x7000 px a jeho vytvorenie si vyžaduje viac operačnej pamäte (200 MB a viac). Trvanie tohto procesu je zároveň aj časovo náročnejšie, čo používateľ určite zaregistruje. Z týchto dôvodov je v prípade veľkej schémy používateľovi zobrazený informačný dialóg s otázkou či má aplikácia v akcii pokračovať. V prípade zápornej odpovede sa ukladanie preruší.

4.5.4 Internacionalizácia a lokalizácia

Internacionalizácia, je spôsob dizajnu softvérových aplikácií tak, aby bolo možné aplikovať metódy pre prispôbenie aplikácie viacerým jazykom a regiónom bez potreby zasahovania do kódu aplikácie. Lokalizácia aplikácie je proces prispôbovania internacionalizovaného softvéru pre použitie konkrétneho jazyka a špecifik regiónu. Qt framework poskytuje spôsob jednoduchého internacionalizovania a lokalizácie aplikácie.

Internacionalizácia textov vo vytvorenej aplikácii je realizovaná pomocou statickej metódy `tr()` triedy `QObject`, prípadne jej reimplementáciou v triedach, ktoré z tejto triedy dedia. Parametrom tejto metódy je textový reťazec, ku ktorému metóda vráti preklad. Táto metóda je použitá všade tam, kde sa používateľovi zobrazuje text.

Metóda `tr()` hľadá preklady reťazcov v súboroch s príponou „.qm“. Pre vygenerovanie týchto súborov je potrebné pomocou nástroja `lupdate` vygenerovať zo zdrojových kódov aplikácie xml súbory (prípona „.ts“) obsahujúce všetky reťazce, ktoré je potrebné lokalizovať. Pre lokalizáciu týchto reťazcov je vhodné použiť Qt Linguist, ktorý poskytuje prehľadné rozhranie pre prekladanie textov. Po preložení potrebných textov je potrebné pomocou nástroja `lrelease` vygenerovať „.qm“ súbory, ktoré môže aplikácia následne využívať.

Aplikácia pri spustení zisťuje, ktorý lokalizačný súbor sa má použiť na základe toho, v akom jazyku je lokalizovaný operačný systém, na ktorom je spustená. Ak aplikácia nemá k dispozícii preklady pre danú lokalizáciu, použije sa lokalizácia pre anglický jazyk. Používateľovi je pomocou hlavného menu GUI umožnené, aby túto voľbu zmenil a aplikácia následne všetky použité reťazce preloží do používateľom zvoleného jazyka.

4.6 Kompilácia a spúšťanie

Na priloženom CD (príloha A) sa nachádza archív obsahujúci zdrojové súbory vytvorenej aplikácie. Predtým ako je možné aplikáciu spustiť je nutné ju najprv skompilovať. Pre skompilovanie aplikácie je potrebné mať nainštalované tieto prerekvizity:

- **G++** – (testovaná verzia 4.4.5),
- **GNU Make** – (testovaná verzia 3.81),
- **Python** – verzia 2.6 alebo 2.7 (nie verzia 3 a vyššia),
- **Qt SDK** – verzia 4.7.0 alebo vyššia.

Po rozbalení archívu s aplikáciou je potrebné navigovať sa do adresáru s názvom `VisualSchemaRepresentationCreator`, v ktorom sa nachádza kompilačný skript s názvom „build“. Tento skript najprv skompiluje OGDF a po úspešnom ukončení kompilácie začne kompilovať vlastnú aplikáciu. Aplikácia sa kompiluje tak, že sa najprv pomocou `qmake` vytvorí „makefile“ a následne sa prostredníctvom GNU Make začne kompilácia.

Kroky potrebné pre skompilovanie aplikácie z príkazového riadku (BASH) sú zhrnuté v kóde 4.7.

```
$ tar xvfz VisualSchemaRepresentationCreator-linux-x86.tar.gz
$ cd VisualSchemaRepresentationCreator
$ ./build
```

Kód 4.7: Kroky potrebné pre skompilovanie aplikácie.

Po úspešnom priebehu kompilácie je v aktuálnom adresári vytvorený binárny súbor, ktorý je možné spustiť. V prípade, že kompilácia neprebehla úspešne je potrebné overiť, či sú nainštalované všetky vyššie spomenuté prerekvizity a cesta k nim je uvedená v systémovej premennej `$PATH`. Prípadné nedostatky je potrebné odstrániť a spustiť kompiláciu vyššie popísaným spôsobom ešte raz.

So zdrojovými súbormi aplikácie je v archíve pribalených aj niekoľko demonštračných schém nachádzajúcich sa v adresári `example_schemas`, ktoré môže používateľ použiť pre ukážku behu aplikácie.

5 Záver

Automatické vykresľovanie grafov do roviny sa stáva problémom v prípade, že je potrebné dodržiavať určité estetické kritériá. Význam dodržiavania týchto kritérií je dôležitý hlavne preto aby informácia uchovávaná grafom bola pre adresáta jednoducho pochopiteľná.

Medzi dôležité estetické kritériá patrí najmä minimalizácia počtu ohybov hrán, zvyšovanie symetrie grafu, ale najdôležitejším estetickým kritériom sa javí minimalizácia počtu krížení hrán grafu.

Databázovú schému relačnej databázy je možné graficky reprezentovať ako tabuľky, medzi ktorými sú orientované asociácie. Tento pohľad je možné zovšeobecniť a databázovú schému reprezentovať ako orientovaný graf, čo umožní pre vykresľovanie schémy použiť algoritmy pre vykresľovanie grafov.

Táto práca poskytuje popis dvoch prístupov pre vykresľovanie grafov. Výsledkom prvého prístupu je hierarchické rozloženie, kde vrcholy grafu sú vo výsledku mapované na horizontálne vrstvy. Druhý prístup sa najprv snaží vytvoriť rozloženie grafu s minimálnym počtom krížení hrán a následne toto rozloženie upraviť do požadovanej vizuálnej prívetivej podoby, napríklad, v tejto práci zdôrazňovanej ortogonálnej podoby.

Pre účely tejto práce bol vykonaný experiment, ktorého cieľom bolo porovnať výsledky algoritmov generujúcich rozloženia grafu. Porovnávané boli dva algoritmy generujúce hierarchické rozloženie (sugiyama algoritmus a vzostupná planarizácia) a jeden algoritmus generujúci ortogonálne rozloženie. Výsledky experimentu ukázali, že v oblasti krížení hrán najlepšie výsledky dosahoval práve ortogonalizačný algoritmus. V oblasti rýchlosti výpočtu sa však tento algoritmus ukázal v porovnaní s hierarchickými algoritmi ako menej výhodný. V prípade, že je požadovaný estetický nákras grafu, je vo všeobecnosti možné preferovať ortogonálny spôsob.

Pre riešenie problémov vizualizácie relačnej databázovej schémy bola ako súčasť tejto práce vytvorená aplikácia pre platformu GNU Linux. Pre eliminovanie problémov spojených s exportnými formátmi relačných databáz bol pre aplikáciu vytvorený vlastný intuitívny vstupný XML formát schopný niesť základné informácie o databázovej schéme.

Aplikácia poskytuje možnosť vykresliť databázovú schému dvoma spôsobmi, a to odporúčaným ortogonálnym spôsobom a hierarchickým spôsobom, ktorý je možné použiť v prípade, že ortogonálny spôsob vykreslí schému nevhodne, predovšetkým v prípadoch, keď je medzi tabuľkami schémy málo relácií a vygenerovaná reprezentácia obsahuje zbytočne „dlhé refazce“ v jednom smere, ktoré zväčšujú plochu nákrasu.

Práve problém „dlhých refazcov“ vymedzuje priestor pre ďalšie vylepšenia ortogonalizačného algoritmu, resp. pridanie ďalšej fázy algoritmu, ktorá by tieto anomálie redukovala a týmto by sa lepšie minimalizovala plocha ktorú nákras zaberá.

Ďalšie vylepšenie aplikácie môže byť podpora klastrov v schéme. Znamená to, že by bolo možné definovať skupiny tabuliek, ktorých asociácie majú obdobný význam a ktoré by boli vo výslednej grafickej reprezentácii agregované do vlastnej oblasti, čo zároveň umožní lepšie členenie schémy a zvýši sémantický rozhľad v schéme hlavne v prípadoch, keď schéma obsahuje veľké množstvo tabuliek.

Literatúra

- [1] Peter Pin-Shan Chen. The entity-relationship model – toward a unified view of data. *ACM Trans. Database Syst.*, 1:9–36, March 1976.
- [2] Markus Chimani, Carsten Gutwenger, Petra Mutzel, and Hoi-Ming Wong. Layer-free upward crossing minimization. *J. Exp. Algorithmics*, 15:2.2:2.1–2.2:2.27, March 2010.
- [3] Markus Chimani, Carsten Gutwenger, Petra Mutzel, and Hoi-Ming Wong. Upward planarization layout. In David Eppstein and Emden Gansner, editors, *Graph Drawing*, volume 5849 of *Lecture Notes in Computer Science*, pages 94–106. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-11805-0_11.
- [4] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 26:64–69, January 1983.
- [5] Nokia Corporation. Qt – A cross-platform application and UI framework. <http://qt.nokia.com/products>. [cit. 22.5.2011].
- [6] R. Dechter. Decomposing an n-ary relation into a tree of binary relations. In *Proceedings of the sixth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '87, pages 185–189, New York, NY, USA, 1987. ACM.
- [7] Giuseppe Di Battista, Ashim Garg, Giuseppe Liotta, Roberto Tamassia, Emanuele Tassinari, and Francesco Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7:303–325, April 1997.
- [8] Giuseppe Di Battista and Roberto Tamassia. On-line graph algorithms with SPQR-trees. In Michael Paterson, editor, *Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 598–611. Springer Berlin / Heidelberg, 1990. 10.1007/BFb0032061.
- [9] Hristo N. Djidjev. A linear-time algorithm for finding a maximal planar subgraph. *SIAM J. Discret. Math.*, 20:444–462, February 2006.
- [10] M.R. Garey and D.S. Johnson. Crossing number is np-complete. *SIAM J. on Algebraic and Discrete Methods*, 4:312–316.
- [11] Carsten Gutwenger, Petra Mutzel, and René Weiskircher. Inserting an edge into a planar graph. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, SODA '01, pages 246–255, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
- [12] Petr Hliněný. Crossing number is hard for cubic graphs. *J. Comb. Theory Ser. B*, 96:455–471, July 2006.
- [13] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46:787–832, November 1999.

- [14] Oracle. Data Integrity.
http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/data_int.htm. [cit. 22.5.2011].
- [15] Rafael Camps Paré. From ternary relationship to relational tables: a case against common beliefs. *SIGMOD Rec.*, 31:46–49, June 2002.
- [16] Helen Purchase. Which aesthetic has the greatest effect on human understanding? In Giuseppe DiBattista, editor, *Graph Drawing*, volume 1353 of *Lecture Notes in Computer Science*, pages 248–261. Springer Berlin / Heidelberg, 1997. 10.1007/3-540-63938-1_67.
- [17] Helen Purchase, Robert Cohen, and Murray James. Validating graph drawing aesthetics. In Franz Brandenburg, editor, *Graph Drawing*, volume 1027 of *Lecture Notes in Computer Science*, pages 435–446. Springer Berlin / Heidelberg, 1996. 10.1007/BFb0021827.
- [18] Il-Yeol Song, Mary Evans, and E.K. Park. A comparative analysis of entity-relationship diagrams. *Journal of Computer and Software Engineering*, 3:427–459, 1995.
- [19] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiro Toda. Methods for visual understanding of hierarchical system structures. *Systems, Man and Cybernetics, IEEE Transactions on*, 11(2):109–125, 1981.
- [20] Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16:421–444, June 1987.
- [21] OGDF Team. gml2pic.
<http://www.ogdf.net/doku.php/project:gml2pic>. [cit. 22.5.2011].
- [22] OGDF Team. News.
<http://www.ogdf.net/doku.php/news>. [cit. 22.5.2011].
- [23] OGDF Team. OGDF – Open Graph Drawing Framework.
<http://www.ogdf.net/doku.php/start>. [cit. 22.5.2011].
- [24] Paul Turán. A note of welcome. *Journal of Graph Theory*, 1:7–9, 1977.

A Obsah priloženého CD

Priložené CD obsahuje adresáre:

- aplikacia – obsahuje archív so zdrojovými súbormi potrebnými pre skompilovanie vytvorenej aplikácie,
- bakalarska_praca – obsahuje text tejto bakalárskej práce vo formáte PDF a jej zdrojové súbory,
- schemy – obsahuje ukážky reprezentácií schém vygenerovaných vytvorenou aplikáciou.