VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

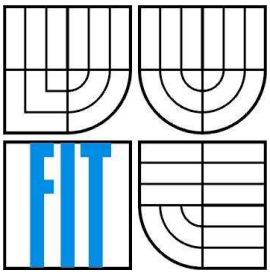
AUTOMATICKÉ ROZVRŽENÍ DIAGRAMŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Lukáš JEZNÝ
AUTHOR

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATICKÉ ROZVRŽENÍ DIAGRAMŮ

DIAGRAM AUTO-LAYOUT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Lukáš JEZNÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. Adam HEROUT, Ph.D.
SUPERVISOR

BRNO 2007

Abstrakt

Tato práce se zabývá automatickým rozvržením diagramů. V teoretické části jsou představeny možnosti, algoritmy a metriky pro automatické rozvržení diagramů. V praktické části jsou pak navrženy algoritmy pro automatické rozvržení diagramů organizační struktury a modelu procesů.

Klíčová slova

Diagram, automatické rozvržení, graf, kreslení grafu, kreslení stromu, kreslení hierarchie

Abstract

Automatic layouts for diagram drawing is described in this paper. Major methods, algorithms, metrics for automatic layouts are introduced in theretical part. Practical part of this work was developing algorithms for automatic layouts of organizational structures and business process model diagrams.

Keywords

Diagram, automatic layout, graph, graph drawing, tree graph drawing, hierarchical drawing

Bc. Lukáš Jezný: Automatické rozvržení diagramů, diplomová práce, Brno, FIT VUT v Brně, 2008.

Automatické rozvržení diagramů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Adama Herouta
Další informace mi poskytli Michal Hrabák, Petr Müller.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Jezný
15.5.2008

Poděkování

Odbornou pomoc mi poskytli Adam Herout a za společnost ISDIM s.r.o, Michal Hrabák, Petr Müller, kterým za to děkuji.

Obsah

Úvod

Již od dětství nás lákají knihy obrázkové více než knihy bez obrázků. V textu je sice uvedeno větší množství faktů, ale právě ilustrace přináší celkový náhled nad informací.

Stejně tak u většiny moderních aplikací, které se nám snaží popsat určitá fakta, zjišťujeme, že nejrychlejším a nejjednodušším výrazovým prostředkem jsou diagramy. Mnoho aplikací spatřilo světlo světa až právě zavedení podpory vizualizace pomocí diagramů. Vzpomeňme například CASE nástroje, které hojně využívají právě popisu pomocí diagramů.

Problém nastává jak správně zobrazit diagram, tak aby byl co nejpřehlednější a poskytoval správný náhled na strukturu. Většinu složitějších diagramů musí kreslit člověk, nicméně některé struktury lze kreslit automaticky.

Tato práce se zabývá automatickým rozvržením diagramů. V první části této práce se zaměřím na popis struktury grafu, budou uvedeny základy teorie grafů potřebné k následující práci.

V další části se ve zkratce zaměříme na dostupný software pro kreslení diagramů a to na software pro ruční i automatické kreslení.

Dále popíšeme metriky pro měření výsledné kvality zobrazení diagramů. Uvedeme typy základních algoritmů pro hledání nejlepšího rozložení v závislosti na uvedených metrikách. Uvedené algoritmy nelze aplikovat na jakékoliv diagramy, proto se podíváme i na automatické rozložení speciálních případů.

Krátce popíšeme aplikaci ATTIS.PM a analýzu problému, pro které bude v praktické části navrženo řešení automatického rozložení pro diagramy modelování procesů a pro vizualizaci organizační struktury.

V praktická část se zaměří na konkrétní algoritmy pro automatické rozvržení diagramů, a jejich implementace do programu ATTIS.PM a zhodnocení dosažených výsledků.

Tato práce navazuje na semestrální projekt řešený Lukášem Jezným ve školním roce 2007/2008 s názvem Automatické rozvržení diagramů, ve které byly položeny teoretické základy této diplomové práce.

1 Úvod do teorie grafů

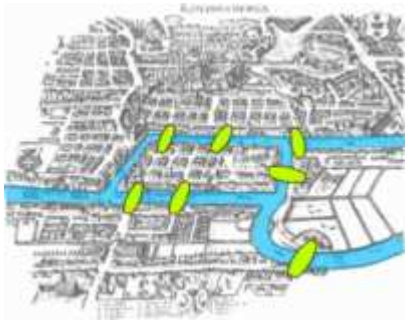
Automatické rozložení diagramů je v podstatě analýza grafu, který je vykreslován.

V matematice a informatice je teorie grafů názvem pro studium grafů: matematických struktur sloužící pro modelování vztahů objektů z dané množiny. Graf je množinou vrcholů (uzlů) a hran spojujících tyto uzly. Graf může být neorientovaný, což znamená, že se nerozeznává orientace spojujících hran, nebo může být orientovaný a mezi vrcholy je orientovaná hrana.

Teorie grafů je velmi obsáhlá, proto zde budou uvedeny jen části, které souvisí s automatickým rozvržením.

1.1 Historie

Za první zmínku o teorii grafů se dá považovat práce Leonharda Eulera zvaná Sedm mostů Královce publikované roku 1736. Řešil úlohu, jak projít přes sedm mostů v Königsbergu (každý z nich právě jednou) a vrátit se do výchozího místa. To v moderní teorii odpovídá pojmu eulerovský graf.



1-1 Mosty města Královce

V roce 1845 publikoval Gustav Kirchhoff zákony, které platí v elektrických obvodech a slouží k výpočtu napětí a proudu v jednotlivých větvích obvodu. V teorii grafů našly své uplatnění při studiu tzv. toků v sítích.

V roce 1852 předložil Francis Guthrie takzvaný problém čtyř barev - tedy otázku, zda je možné obarvit libovolnou mapu pomocí nejvýše čtyř barev tak, aby každé dvě sousední země (které mají společnou hranici delší než jediný bod) měly odlišnou barvu. Byl vyřešen až o více než sto let později, přičemž pro jeho řešení bylo zavedeno mnoho zásadních konceptů teorie grafů (viz rovinný graf).

1.2 Kreslení grafů

Grafy (diagramy) se obvykle reprezentují body pro zobrazení vrcholů a křivkami pro zobrazení hran mezi spojovanými vrcholy. Šipky pak znázorňují orientaci vazeb v orientovaných grafech. Poznamenejme, že grafická reprezentace (rozvržení, styl kreslení) by neměla být zaměněna se samotným grafem (abstrakce, negrafická struktura). V abstrakci záleží jen na vrcholech a na tom jak jsou propojeny hranami, ale to jak budou nakresleny může pomoci k lepšímu pochopení tohoto grafu.

1.3 Datové struktury

Je mnoho způsobů jak v počítači reprezentovat graf. Použitá datová struktura se často odvíjí od použitého algoritmu ke zpracování grafů. Teoreticky lze rozlišit mezi zápisem v seznamu nebo pomocí matice, prakticky se ale používá většinou kombinace obou. Seznamy jsou často preferovány pro ukládání řídkých grafů (počet hran << počet uzlů). Matice na druhou stranu poskytují rychlejší přístup pro zpracování, ale využívají velké množství paměti.

Následující seznam uvádí nejběžnější reprezentace grafů

- Matice sousednosti
- Matice vzdáleností
- Laplaceova matice
- Matice incidence
- Seznam sousedů

1.4 Problémy v teorii grafů

Pro zajímavost budou uvedeny některé problémy teorie grafů.

1.4.1 Problém čtyř barev

Problém čtyř barev či také věta o čtyřech barvách je (již kladně vyřešený) problém z teorie grafů, který zní: „Stačí čtyři barvy na obarvení libovolné politické mapy tak, aby žádné dva sousedící státy nebyly obarveny stejnou barvou?“ (Za sousední státy jsou považovány takové, že mají společnou hraniční čáru, tj. nesousedí spolu jen v jednom bodě.) Obecněji se lze tázat na minimální potřebný počet barev, lze však poměrně snadno dokázat, že pět barev postačuje. Oproti tomu tvrzení, že čtyři barvy stačí, dlouhou dobu odolávalo všem pokusům o důkaz, nikdo však také nebyl schopen nalézt mapu, která by ho vyvrátila.

Větu dokázali až roku 1976 američtí matematici Kenneth Appel a Wolfgang Haken tím, že pomocí počítačového programu vymodelovali 1936 možných konfigurací, dokázali, že tyto

konfigurace pokrývají všechny možnosti, a u každé z nich ukázali, že pro její obarvení čtyři barvy stačí (k tomu potřeboval 1200 hodin procesorového času). Tento důkaz však velká část matematiků odmítá akceptovat, protože ho žádný matematik není schopen přímo zkontrolovat. (Od té doby byl důkaz mnohokrát nezávisle zopakován a zjednodušen dalšími matematiky pomocí jiných programů, ale „hezký“ důkaz vhodný pro člověka nalezen nebyl.)

1.4.2 Problém obchodního cestujícího

Laická formulace: Existuje n měst, mezi nimi silnice o známých délkách. Úkolem je najít nejkratší možnou trasu, procházející všemi městy a vracející se nazpět do výchozího města.

Matematická formulace používající pojmosloví teorie grafů: Jak v daném ohodnoceném úplném grafu efektivně najít nejkratší hamiltonovskou kružnici?

Problém nespočívá ani tak ve stanovení libovolného postupu nalezení nejkratší cesty – jeden takový postup je totiž skoro samozřejmý: stačí jednoduše prohledat všechny možné uzavřené cesty mezi danými městy a vybrat nejkratší z nich. Obtíž však je, že s rostoucím počtem měst (či uzlů grafu) počet možných cest velice rychle narůstá, a tím se doba potřebná k propočtu hrubou silou na počítačích stává zcela neúnosnou už při několika málo desítkách uzlů. Klíčová obtíž je tedy v nalezení časově efektivního algoritmu hledání nejkratších cest.

1.5 Aplikace

Struktury, které mohou být popsány grafem jsou všudypřítomné a mnoho problémů kolem nás může být grafem reprezentováno.

Odkazy na webových stránkách mohou být reprezentovány orientovaným grafem. Uzly jsou jednotlivé dostupné webové stránky a hrany jsou odkazy mezi jednotlivými stránkami. Podobný přístup může být použit v biologii, návrhu počítačových čipů apod. Vývoj algoritmu pro zpracování grafů je hlavním zájmem počítačového oboru.

Struktura grafu může být rozšířena přiřazením váhy k jednotlivým hranám. Grafy s váhami, nebo váhované grafy, jsou používány k popisu struktur, ve kterých párující spoj reprezentuje nějakou numerickou hodnotu. Například v silniční síti mohou tyto váhy reprezentovat vzdálenosti jednotlivých měst. Orientované váhované grafy se nazývají sítě.

Sítě mají velký praktický význam v teorii grafů, například pro analýzu silniční dopravy.

Teorie grafů je také součástí studia v chemii a fyzice. Trojrozměrné grafy se používají například k simulací atomové struktury materiálu.

Dále se používá teorie v sociologii. Známé jsou analýzy sociálních sítí.

1.6 Planární grafy

Pro studium automatického rozložení diagramu má velký význam určení, zda je graf planární.

Rovinný graf (též planární graf) je graf, pro který existuje takové rovinné nakreslení, že se žádné dvě hrany nekříží.

1.6.1 Kuratowského věta

Graf G je rovinný právě tehdy, není-li žádný jeho podgraf izomorfní dělení grafu K5 ani K3,3. (K5 označuje úplný graf na pěti vrcholech, K3,3 pak úplný bipartitní graf.)

1.6.2 Eulerův vzorec

Pro rovinné grafy také platí následující vzorec, je to ovšem pouze implikace: Je-li G = (V, E) rovinný graf, pak | V | − | E | + s = 2, kde s je počet stěn nějakého rovinného nakreslení tohoto grafu.

1.7 Prohledávání grafu

Prohledávání grafu je systematický postup pro řešení následujících úloh. Prohledávání grafu je jedním z nejpoužívanějších algoritmů pro analýzu grafů pro automatické rozvržení.

1. Zjistit, zda vrchol t je dosažitelný z vrcholu r.
2. Najít množinu všech vrcholů dosažitelných z vrcholu r.
3. Najít (jakoukoliv) cestu z vrcholu r do vrcholu t (pokud existuje).
4. Najít (jakékoli) cesty z vrcholu r do všech vrcholů, které jsou z něj dosažitelné.
5. Pro všechny vrcholy, které jsou dosažitelné z vrcholu r, provést zadanou operaci.

Mezi nejznámější postupy patří

- prohledávání do šířky (breadth-first search)
- prohledávání do hloubky (depth-first search)
- metoda větví a mezí (branch and bound method)
- prohledávání s návratem (backtracking)

Prohledávání do hloubky a do šířky jsou pro další práci velmi důležité algoritmy, proto se na ně podíváme podrobně.

1.7.1 Prohledávání do šířky

Prohledávání do šířky (anglicky Breadth-first search, zkráceně BFS) je grafový algoritmus, který postupně prochází všechny vrcholy v dané komponentě souvislosti. Algoritmus nejprve projde všechny sousedy startovního vrcholu, poté sousedy sousedů atd. až projde celou komponentu souvislosti.

Prohledávání do šířky je algoritmus či metoda, která postupuje systematickým prohledáváním grafu přes všechny uzly. Nepoužívá při svém prohledávání žádnou heuristickou analýzu. Pouze prochází všechny uzly a pro každý projde jejich všechny následovníky. Přitom si poznamenává předchůdce jednotlivých uzlů a tím je poté vytvořen strom nejkratších cest k jednotlivým uzlům z kořene (uzlu, v kterém jsme začínali).

Z hlediska algoritmu, veškeré následovníky uzlu získané expandujícím uzlem jsou vkládané do FIFO fronty. FIFO fronta znamená, že první uzel, který do fronty vstoupil jí také první opustí. V typických implementacích, jsou uzly, které nebyly ještě objeveny označeny jako FRESH. Uzly, které se dostávají do fronty, které jsou v této chvíli právě vyšetřovány na jejich následníky jsou označeny jako OPEN a naposled uzly, které z fronty byly už vybrány a už s nimi nebudeme pracovat, jsou označeny jako CLOSE. Close uzly již nikdy v tomto běhu algoritmu nebudou prozkoumávány, mají vyplněné všechny informace. To znamená vzdálenost od kořenového (počátečního) uzlu, stav uzlu (CLOSE) a předchůdce. Více informací v [11].

1.7.1.1 Popis algoritmu

Na počátku algoritmu se provede inicializace. Poté se nastaví počáteční hodnoty pro jediný uzel, u kterého známe všechny informace a to pro počáteční uzel (viz. stav při inicializaci). Nyní se rozběhne cyklus, který běží, dokud je FIFO fronta neprázdná. Neprázdná fronta znamená, že máme stále uzly, s kterými pracujeme, jelikož do této fronty jsou vkládány pouze uzly, s kterými nyní pracujeme. Na počátku cyklu z fronty vyjmeme uzel. Pro všechny následníky tohoto uzlu budeme zjišťovat, jestli mají stav FRESH. Stav FRESH znamená, že uzel nebyl ještě nalezen ani prozkoumán. Když bude tato podmínka týkající se stavu splněna, nastaví se informace pro tyto uzly. Stav OPEN, vzdálenost od počátku o jedna větší než uzel, kterého je tento uzel následník. Vzdálenost je větší právě o jedna z důvodu, že je mezi těmito uzly pouze jedna hrana. Naposled se pro tyto následníky uzlu (u) nastaví předchůdce právě na hodnotu (u). Po nastavení všech těchto vlastností je uzel vložen do fronty. Když jsou prohledáni všichni následníci daného uzlu, tak se uzel uzavře, stav uzlu se bude rovnat CLOSE.

1.7.1.2 Stav při inicializaci

Všechny tyto datové struktury jsou na počátku algoritmu inicializované. Fifo fronta je inicializovaná jako prázdná fronta. V poli vzdáleností mají všechny prvky na počátku hodnotu nekonečno. Nekonečno je nastaveno proto, jelikož by cesta byla nekonečně daleká neboli, že neexistuje. Opačná hodnota nula je nastavena na začátku pouze uzlu počátečnímu, z kterého bude algoritmus začínat svůj běh. Nulová vzdálenost znamená, že se jedná o totožný uzel. Pole stavů je nastaveno na počátku na hodnoty FRESH pro všechny uzly mimo prvnímu který má stav OPEN. Pole předchůdců je nastaveno na null pro všechny uzly. Nakonec se do fronty vloží uzel s (startovací uzel). Velikosti jednotlivých polí jsou nastaveny na velikost shodnou s počtem uzlů v grafu.

1.7.1.3 Časová složitost

Asymptotická časová složitost algoritmu je O(|V| + |E|), kde V je množina vrcholů a E je množina hran grafu.

1.7.1.4 Prostorová složitost

Prostorová složitost je O(|V| + |E|). Řečeno jinak, prostorová složitost je O(BM), kde B je nejvyšší stupeň větvení stromu a M je nejvyšší délka cesty ve stromě. Tento velký nárok na prostor ve srovnání s nárokem prohledávání do hloubky je důvod, proč je prohledávání do šířky nepraktické pro rozsáhlejší problémy.

1.7.2 Prohledávání do hloubky

Prohledávání do hloubky (v angličtině označované jako depth-first search nebo zkratkou DFS) je grafový algoritmus pro procházení grafů metodou backtrackingu. Pracuje tak, že vždy expanduje prvního následníka každého vrcholu, pokud jej ještě nenavštívil. Pokud narazí na vrchol, z nějž už nelze dále pokračovat (nemá žádné následníky nebo byly všichni navštíveni), vrací se zpět backtrackingem. Více informací v [12].

1.7.2.1 Vlastnosti algoritmu

Algoritmus je úplný (vždy najde řešení, tzn. určitý cílový vrchol, pokud existuje), ale není optimální (pokud graf není strom, nemusí najít nejkratší možnou cestu k cíli). Algoritmus dokáže prohledat pouze souvislé komponenty grafu, není tedy vždy zajištěno ani to, že algoritmus projde všechny uzly. Předpokládáme, že graf má uzly ohodnocené číselnými hodnotami, které budeme dále používat zejména pro výběr jakým směrem, přes jaký uzel z následníků daného uzlu se má algoritmus vydat. V tomto algoritmu totiž panuje pravidlo výběru postupně právě těch následníků daného uzlu v pořadí hodnot, kterými jsou uzly hodnoceny. Od nižšího ohodnocení k vyšším. V grafu se vyskytují celkem čtyři typy hran. Jsou to hrany stromové, příčné, zpětné a dopředné. Algoritmus prochází hranami a postupuje tím stylem, že se snaží projít nejprve co nejhlouběji do nitra (z tohoto principu se také tomuto algoritmu říká prohledávání do hloubky), či co nejdál to jen jde. Tyto hrany, po kterých algoritmus prochází takto "co nejdál" grafem se nazývají hrany stromové. Z těchto hran je poté vytvořen strom nejkratších cest z kořene (počátečního uzlu). Když algoritmus zkouší nalézt další FRESH uzly po dalších hranách, tak se stane, že místo aby tento uzel měl stav FRESH tak bude OPEN. Tuto hranu poté pojmenujeme jako zpětnou. Tuto zpětnou vazbu vlastně ani nevyužijeme, jelikož díky ní pouze zjistíme, že dále pokračovat již není možné a musíme se algoritmem vracet zpět a zkoušet cestu přes jinou hranu vedoucí k dalšímu následníkovi. Jestliže narazíme z uzlu OPEN na uzel CLOSE, jedná se o příčnou či o dopřednou hranu. Mezi těmito dvěma typy hran rozlišujeme podle časové značky otevření uzlu. Jestliže se dostaneme z uzlu OPEN s časovou značkou otevření například 10 do uzlu CLOSE, přes dopřednou hranu tak bude platit, že tento CLOSE uzel bude mít

časovou značku otevření vyšší než 10. Dopředná hrana spojí uzel s nižší čas. značkou otevření s uzlem s vyšší značkou otevření. Jestliže se stane tato situace přesně opačně, že hrana spojí uzel s vyšší čas. značkou otevření s uzlem s nižší značkou otevření, bude se jednat o hranu příčnou. Stejně jako hranu zpětnou tak ani posledně dvě zmiňované hrany ve výsledném stromu nejkratších vzdáleností nevyužijeme a slouží nám pouze jen pro to, abychom zjistili, že se máme vydat jinou cestou.

1.7.2.2 Popis algoritmu

Celý algoritmus začíná inicializací všech uzlů grafů hodnotami stavů uzlů a nastavením jejich předchůdců. Každý uzel je nastaven jako FRESH uzel a žádný zatím nemá určeného svého předchůdce. To znamená, že v poli předchůdců jsou všechny hodnoty prvků nastaveny na null. Index (dále již časový index), který budeme dále využívat pro zapisování časových značek nastavíme na nulu. Poté je pro všechny uzly vyvolána metoda DFS-Projdi. V této metodě je na počátku stav uzlu, pro který byla tato metoda volána nastaven na OPEN. Do časové značky otevření uzlu se zapíše hodnota časového indexu, který se ve stejné době i inkrementuje, abychom mohli o jedna větší index použít pro další uzel. Poté se pro každého následníka uzlu, pro který bylo voláno DFS-projdi zjistí, jestli je jako následník ještě FRESH, zapíše se hodnota předchůdce a zavolá se rekurzivně znovu metoda DFS-Projdi na tohoto následníka, v kterém právě jsme. Jestliže se projdou všichni následníci uzlu, tento uzel se uzavře, neboli se tomuto uzlu přiřadí stav CLOSE a nastaví se pro něj časová značka uzavření použitím námi využívaného časového indexu, který se následně o jedna zvětší.

1.7.2.3 Rozdíl v algoritmu při aplikaci na neorientovaný graf

Algoritmus má i pro neorientovaný graf stejný průběh. Jediné v čem se celkově algoritmus liší oproti aplikaci na orientovaný graf, je že se zde nacházejí pouze dva ze čtyř různých typů hran. V tomto případě se v grafu budou vyskytovat pouze hrany stromové a zpětné. Zbývající dva typy hran dopředné a příčné v tomto typu grafu neexistuje.

1.7.2.4 Další využití algoritmu

Algoritmus prohledání do hloubky se může dále využívat například pro topologické uspořádání uzlů grafu, nalezení silných komponent grafu či zjištění acykličnosti grafu.

1.7.2.5 Topologické uspořádání uzlů grafu

Použijeme novou datovou strukturu zásobník. Dále necháme volně procházet algoritmus prohledání do hloubky po grafu a jedinou změnou, kterou v algoritmu vůči "normálnímu" průběhu uděláme, bude, že pokaždé když algoritmus nějakému uzlu přiřadí časovou značku uzavření tak tento daný uzel vložíme do zásobníku. Na konci běhu algoritmu máme v zásobníku topologicky uspořádané uzly.

1.7.2.6 Nalezení silných komponent

Pomocí algoritmu prohledání do hloubky určíme u všech uzlů časové značky uzavření. Pote graf, na kterém jsme tento algoritmus spustili poprvé, tak mu změníme orientaci na opačnou (každá hrana bude opačně orientovaná). Na tento opačně orientovaný graf spustíme znovu algoritmus prohledávání do hloubky a uzly budeme brát v klesajícím pořadí značek uzavření, jež jsme zjistili prvním průchodem algoritmu ještě na "originálně orientovaném" grafu. Po ukončení tohoto algoritmu už získáme stromy lesa, které nám budou určovat silné komponenty grafu.

1.7.2.7 Zjištění acykličnosti grafu

Zjištění acykličnosti grafu provedeme pouze přidáním podmínky, že jestliže algoritmus nalezne nějakou hranu, kterou určíme jako zpětnou tak se v grafu bude vyskytovat cyklus. Zpětná hrana totiž v grafu tvoří cykly.

1.7.2.8 Složitost algoritmu

Celková složitost algoritmu : O(|V| + |E|), kde V je počet vrcholů a E počet hran daného grafu.

2 Automatické kreslení diagramů

Kreslení diagramů – grafů, je částí oboru teorie grafů. Kreslení diagramů aplikuje topologii a geometrii k vytvoření dvou nebo tří rozměrné reprezentace dat. Automatické kreslení diagramů je motivováno designem elektronických obvodů (hledání nejlepšího schématu), analýzou sociální sítě, kartografií, bioinformatikou atp.

2.1 Hlavní typy diagramů

Existuje mnoho metod, kterými lze popsat informaci. Některé metody jsou zaměřeny na výkonnost při vyhledávání, jako např. relační databáze. Pro člověka jsou ale tyto metody popisu dat příliš složité na pochopení jejich struktury. Jednou z metod pro lepší pochopení této struktury jsou diagramy.

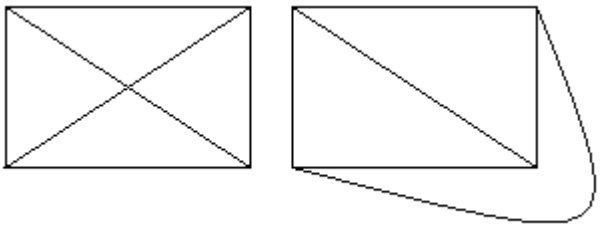
Diagram je dvourozměrnou symbolickou reprezentací odpovídající nějaké zobrazovací technice. Určité zobrazovací techniky používají třírozměrné zobrazení, které je ale následně projekcí zobrazeno do dvou rozměrů.

Vzhledem k tomu, že pomocí diagramů lze popisovat-zobrazovat mnoho typů dat a struktur, bylo vytvořeno nepřeberné množství typů diagramů.

Diagramy lze rozdělit do dvou hlavních typů:

- diagramy, které jsou popsány matematickou strukturou – graf, kde graf je množina uzlů a hran. Uzly zobrazují jednotlivé entity s jejich dvourozměrnou pozicí na diagramu. Hrany pak zobrazují relace mezi těmito entitami. Mezi nejznámější typy této třídy patří např.
    - stromové diagramy
    - síťové diagramy
    - diagramy toků
    - Eulerovy diagramy, Vennovy diagramy
    - mapy (speciálním případy – mapa metra v Praze)
- diagramy, které jsou známy spíše pod názvem grafy. Popisují závislosti dvou nebo více spojitých nebo diskrétních proměnných. Studium těchto typů diagramů není cílem této práce. Příkladem:
    - histogramy, sloupcové grafy
    - koláčové grafy
    - grafy funkcí
    - tabulkové – maticové grafy

2.2 Obecné metriky pro měření kvality rozvržení.



2-1 Kompletní K4

Na uvedeném obrázku je uveden graf K_4(kompletní graf o čtyřech vrcholech). Z obrázku lze vidět, že graf lze nakreslit dvěma způsoby. V jedné variantě se dvě hrany kříží, ve druhé variantě se nekříží žádná hrana. Je na čtenáři, nechť si určí která varianta pro něj lépe odpovídá grafu. Z obrázku vlevo však nejde poznat, jedná-li se o graf o čtyřech nebo pěti vrcholech.

Z předchozího odstavce lze zjistit dvě informace. A to, že kreslení grafu je subjektivní, ale také to, že kvalitu zobrazení lze poměřovat počtem křížení hran.

Jednou z metrik je tedy počet překřížení hran. Obecně je dobré minimalizovat počet překřížení, nejlépe se mu úplně vyhnout. Některé grafy nelze bez křížení nakreslit, některé lze, těm se říká planární grafy. „Dobré“ algoritmy jsou podle této metriky ty, které dokáží minimalizovat křížení.

Druhou možnou metrikou je blízkost vrcholů. Mnoho grafů vypadá lépe, pokud nespojené vrcholy nejsou kresleny blízko sebe. Metrikou tedy může být vzdálenost nespojených vrcholů. „Lepší“ algoritmy pak dávají nespojené vrcholy dál od sebe, a spojené blíž.

Při ručním kreslení diagramů se většinou snažíme pokrýt co nejmenší část papíru. Stejně tak může být právě pokreslená plocha metrikou kvality rozvržení.

Celkový vzhled diagramu taky ovlivňuje poměr stran nakresleného diagramu. Obvykle se snažíme zachovat rozumný poměr stran např. typicky počítačová obrazovka 4:3.

Bohužel se uvedené metriky vylučují a problém ideálního rozmístění se hodně podobá NP problému.

2.3 Problém volby vhodného algoritmu

Poté co jsme si ukázali metriky, které jsou vhodné dodržovat při kreslení, si ukážeme určitý postup, který je vhodný dodržet pro volbu algoritmu řešící náš problém. Je vhodné si zodpovědět na tyto otázky.

1. Musíme hrany kreslit rovné nebo zalomené? Algoritmy s kreslící pouze rovné hrany jsou jednodušší než s lomenými hranami. Ale pro zobrazení komplikovaných

diagramů jako jsou modely PCB desek se naopak hodí právě lomené (nejlépe ortogonální) hrany.

2. Potřebujeme pro aplikaci specifický algoritmus? Pokud máme graf, kde vrcholy reprezentují města a hrany reprezentují cesty – silniční síť, tak si prostě nemůžeme dovolit nakreslit vrcholy libovolně, ale musíme dodržet jejích fyzické umístění v mapě.
3. Je graf planární nebo je to strom? Pro každý typ se hodí jiné algoritmy.
4. Jak rychle by měl algoritmus pracovat? Pro interaktivní obnovování zobrazení musí náš algoritmus být velmi rychlý.

2.4 Metody

Existuje mnoho přístupů ke kreslení diagramů. Nyní si ukážeme několik možných přístupů.

2.4.1 Topologie – Tvar – Metriky

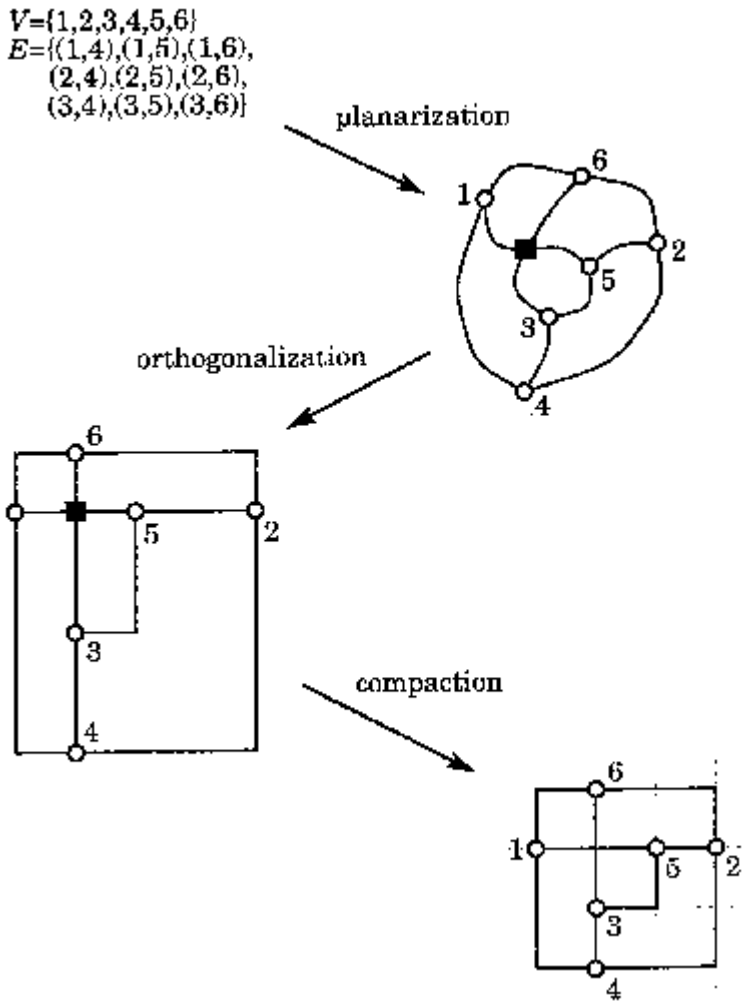
Tento přístup se skládá z třívrstvé úpravy původního grafu. Výsledná kresba je ortogonální, které se často používá při kreslení elektrických schémat. Tento přístup minimalizuje počet překřížení hran, protože prvním krokem je planarizace.

Tři vlastnosti, které se používají v této metodě jsou topologie, tvar, metrika. Dva grafy sdílí stejnou topologii, jestliže jeden muže být přetvořen do druhého použitím deformací beze změny pořadí vrcholů a hran. Dva grafy sdílí jeden tvar, jestliže jeden může být přetvořen na druhý zkrácením délek hran, ale nikoliv úhlů, které hrany svírají. Grafy sdílejí metriku, jestliže jeden může být přetvořen na druhý pouze za pomocí posunutí a rotace.

Prvním krokem této metody je obecně proces planarizace. V tomto kroku je graf transformován ze své matematické podstaty do 2D kresby, ve které je minimální počet překřížení. Jestliže by mělo dojít k překřížení, je toto překřížení nahrazeno pomocným vrcholem. Výstupem je topologie.

Po planarizaci bývá dalším krokem ortogonalizace. V tomto kroku jsou všechny úhly hran nahrazeny pravými úhly a vrcholy jsou zarovnány. Výsledkem je tvar.

Posledním krokem je kompakce. V tomto kroku jsou odstraněny pomocné vrcholy a celý graf je zmenšen tak aby zabíral co nejmenší obsah. Konečným výsledkem je metrika.



2-2 Topologie tvar metrika, převzato z [7]

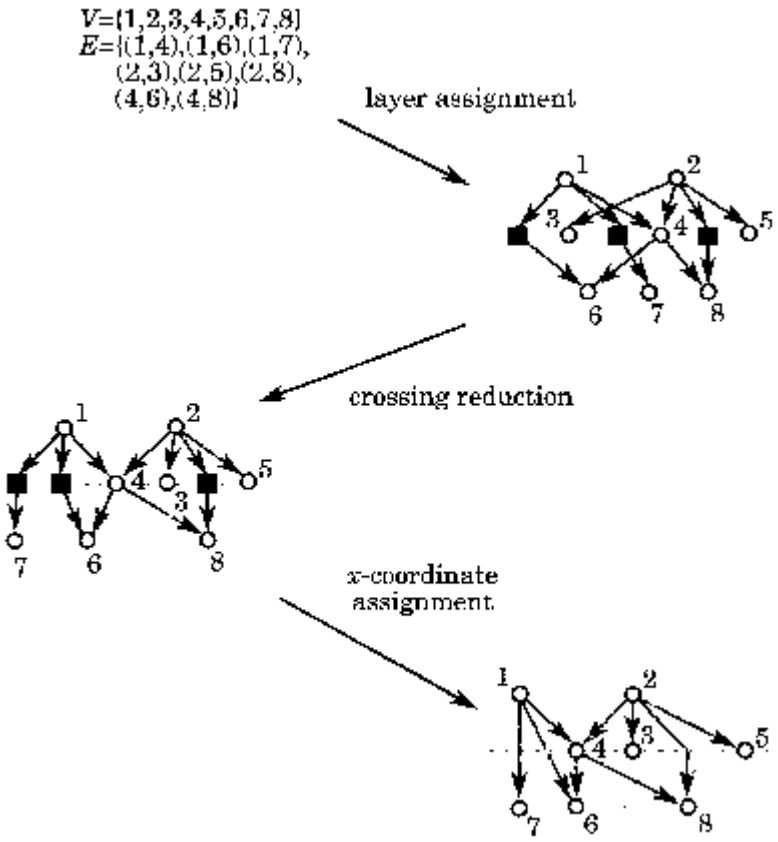
2.4.2 Hierarchie

Hierarchický přístup se používá pro zobrazení hierarchie orientovaných grafů. Hierarchie mají různorodé použití např. diagram tříd.

Prvním krokem v hierarchickém přístupu je přidělení vrstev. V tomto kroku jsou vrcholům přiděleny vrstvy, které začínají na vrchu grafu a prochází do nižších vrstev grafu. Každému vrcholu je přiděleno číslo L_n tak, že pokud existuje hrana mezi vrcholy U a V, vrchol U je ve vrstvě L_i a vrchol V je ve vrstvě L_j pak i < j. Jestliže existuje mezera mezi vrstvami taková, že by byla hrana z L_1 do L_3, pak umístíme pomocný vrchol ve vrstvě L_2.

Po tomto kroku následuje proces zvaný redukce překřížení. V tomto kroku jsou redukována překřížení tak, aby graf vypadal „lépe“.

Poslední krok se nazývá přidělení x-souřadnice. V této části se všem vrcholům přidělí pozice, odstraní se pomocné vrcholy a vykreslí všechny grafy. Zaměřujeme se také na minimalizaci zalomení vazeb a minimalizaci použité plochy.



2-3 Hierarchie, převzato z [7]

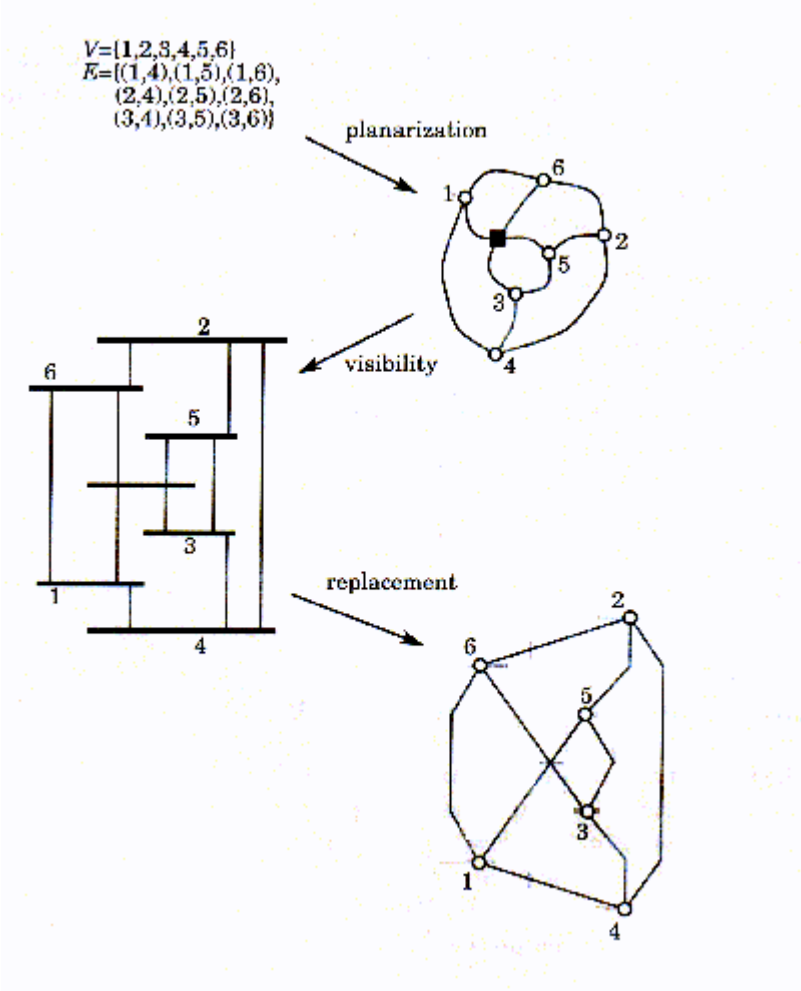
Obrázek ukazuje příklad metody hierarchizace. Přiřazení vrstev (vznik pomocných vrcholů), minimalizace překřížení a konečné kreslení s využitím pomocných vrcholů.

2.4.3 Viditelnost

Přístup viditelnosti pro kreslení grafů je obecnou technikou pro kreslení lomených hran. Stejně jako u techniky topologie-tvar-metrika je prvním krokem planarizace. A znovu klademe v tomto kroku důraz na redukci překřížení.

Krok viditelnosti je těžký na vysvětlení bez obrázku, který je uveden níže. V podstatě každý vrchol je přeměněn v pomocnou horizontální příčku. Hrany pak tvoří vertikální spoje.

Výsledek získáme po aplikaci posledního kroku – náhrada. V této části jsou horizontální příčky nahrazeny body a hrany jsou kresleny tak aby překrývaly vertikální spoje z druhého kroku.



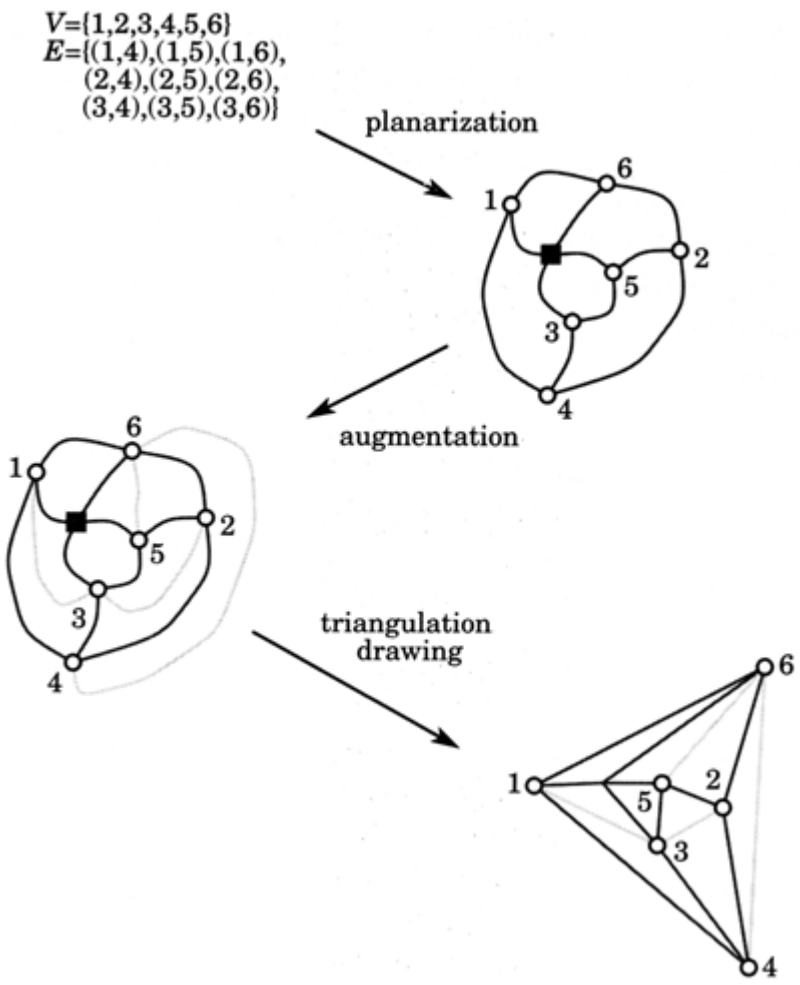
2-4 Přístup podle viditelnosti, převzato z [7]

2.4.4 Přírůstková metoda

Tento přístup je podobný předchozímu. Opět je prvním krokem planarizace.

Další krok se nazývá přírůstky. V tomto procesu jsou pomocné vrcholy kresleny v pořadí tak aby všechny plochy grafu měly právě tři vrcholy. V příkladu uvedeném na obrázku si můžeme všimnout, že plocha (3,2,4,5) má čtyři vrcholy. V tomto kroku pak přidáme pomocnou hranu (3,2), abychom dostali plochy (3,5,2) a (3,2,4) právě o třech vrcholech.

Posledním krokem je triangulace. Graf je nakreslen pomocí trojúhelníků definovaných v předchozím kroku. Samozřejmě, že pomocné hrany nekreslíme.



2-5 Přírůstková metoda, převzato z [7]

2.4.5 Rozložení založené na minimalizaci vzájemného působení sil. (Spring embedding)

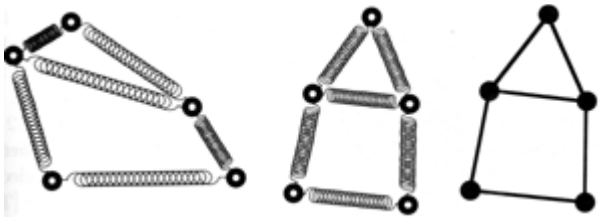
Hlavním cílem této třídy algoritmů je snížit počet překřížení hran a zaručit, že všechny hrany budou přibližně stejně dlouhé.

Algoritmus toho docílí přiřazením určitých sil pro množinu hran a pro množinu vrcholů. Nejvyužívanější je přiřazení pružnosti (Hookův zákon) pro množinu hran a elektrického náboje pro vrcholy (Coulombův zákon). Celý proces kreslení diagramu je pak vlastně fyzikální simulací. Síly působící na uzly je pak sbližují nebo naopak oddalují. Tento proces je iterativně opakován do té doby, dokud není dosaženo rovnovážného stavu (equilibrium) – další krok již nepřinese mnoho změn. V této chvíli je diagram nakreslen, napozicován. Síly jsou v rovnovážném stavu.

Algoritmus může používat sofistikovanější modely působení sil, než pouze mechanické a elektrické. Například logaritmickou pružnost (místo lineární) a magnetické nebo gravitační síly.

Výsledky těchto rozvržení vypadají vždy velmi dobře. V případě použití výše zmíněného fyzikálního modelu docílíme jednotné délky hran (díky pružnosti) a nespojené vrcholy budou ve větších vzdálenostech než spojené, díky elektrické odpudivé síle.

Zatímco kreslení grafů – diagramů může být složitým problémem, algoritmy založené na silovém působení, obvykle nepotřebují žádné speciální znalosti o teorii grafů, jako je např. planárnost.



2-6 Spring embedding, převzato z [7]

2.4.5.1 Výhody

- dobrá kvalita výsledku, alespoň u středně velkých diagramů (50 – 100 vrcholů) dosahují tyto algoritmy dobrých výsledků z hlediska estetiky – a to hlavně: jednotná délka hran, jednotná vzdálenost rozmístění vrcholů ukazující symetrii. Právě symetrie je velmi náročná na udržení a jiné třídy algoritmů s nimi mají velké problémy.
- flexibilita, je velmi snadné změnit algoritmus rozmístění pouze změnou vlastností fyzikálního modelu.
- intuitivnost, vzhledem k tomu, že jsou založeny na známém a dobře popsaném fyzikálním modelu, lze očekávat jejich chování.
- jednoduchost, jednoduchý algoritmus může být implementován na několika řádcích kódu.
- interaktivita, iterativní postup může být použít pro vytvoření animace a uživatel může vidět, jak diagram vzniká, a tím může vidět doposud skryté vlastnosti diagramu

2.4.5.2 Nevýhody

- časové složitost, v každé iteraci musí být navštívena každá dvojice vrcholů, třída těchto metod je velmi pomalá.
- lokální minima, lze si povšimnout, že proces umístění může skončit, aniž by očividně bylo vše správně umístěno. Celková energie se dostala do minima, ale pouze lokálního

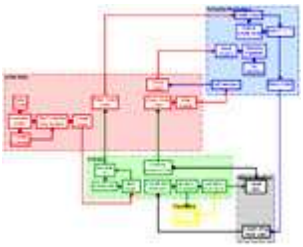
2.4.6 Rozděl a panuj

Přístup vychází ze svého názvu, rozděluje graf na menší podgrafy, které mohou být jednoduše vykresleny. Graf je pak složením těchto podgrafů. Je mnoho rozdílných technik, které popisují jak provádět rozdělení a kreslení podgrafů.

2.5 Nejběžnější rozvržení diagramů v praxi.

V této kapitole si ukážeme několik typů rozvržení a jejich použití v praxi.

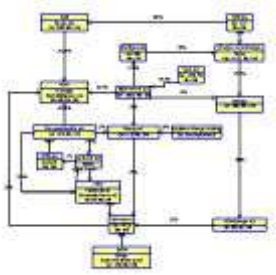
2.5.1 Shlukové rozvržení



Toto rozložení lze s využít pro znázornění shlukové hierarchie. Tu lze nalézt tam, kde lze jednoznačně určit skupinu příslušnosti jednotlivých vrcholů. Rozvržení, ač vypadá jednoduše, lze využít při návrhu plošných spojů.

2-7 převzato z [4]

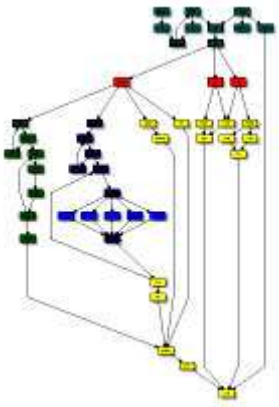
2.5.2 Ortogonální rozvržení (UML)



Rozložení je podobné předchozímu, avšak není třeba tak důsledného oddělení jednotlivých skupin. Algoritmus lze nalézt například ve výše uvedené aplikaci Microsoft SQL Enterprise Manager, pro znázornění struktury relační db. Dále pak, jak je v názvu uvedeno, pro zobrazení struktur při UML modelovaní.

2-8 převzato z [4]

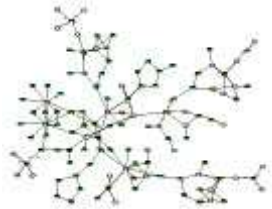
2.5.3 Hierarchické rozložení



Pro praktickou část velmi inspirativní hierarchické rozložení se snaží v grafu najít počátek a konec diagramu a při vykreslování postupovat právě od počátku ke konci a proplouvat hierarchií. Používá se k zobrazování hierarchických struktur. Z velké části bude toto rozložení využito pro rozložení diagramu modelu procesů.

2-9 převzato z [4]

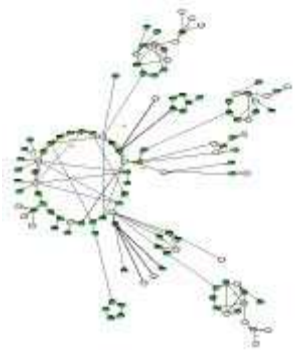
2.5.4 Symetrické rozložení



Prvním zástupcem algoritmů pro pozicování rozsáhlých grafu je symetrické pozicování. Podobné výsledky získáme z rodiny algoritmů založených na minimalizaci sil.

2-10 převzato z [4]

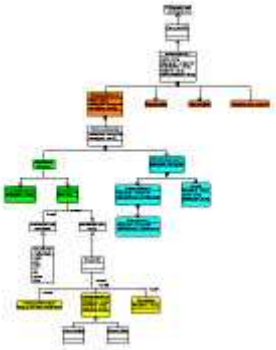
2.5.5 Kruhové rozložení



Druhým zástupcem rozložení, u kterých vynikají jejich přednosti až při narůstajícím počtu pozicovaných vrcholů. Algoritmus nemá příliš velké uplatnění, protože sice pozicuje diagramy tak, aby byly přehledné, ale bohužel nepokrývá příliš jejich hierarchii.

2-11 převzato z [4]

2.5.6 Stromové rozložení



Stromové rozložení přistupuje k zobrazení diagramu jako průchod stromem. Uvedený obrázek je typickým zástupcem zobrazení organizační struktury.

2-12 převzato z [4]

3 Dostupný software

Pro kreslení diagramů lze dnes existuje obrovské množství programů. Vzhledem k tomu, že diagram je jednou z nejlepším metod pro reprezentaci struktury, je právě možnost jeho kreslení součásti většiny větších programů pracující s daty. Příkladem takové větší aplikace s integrovanou podporou kreslení diagramů jsou například vývojářské nástroje (tzv. IDE).

Existují samozřejmě samostatné aplikace, které se zaměřují na (polo)profesionální kreslení diagramů. Většina těchto programů lze použít jako komponentu pro výstup diagramů. Často se v praxi používá např. Microsoft Visio (for Enterprise Architects), které lze použít samostatně, nebo jako komponentu pro diagramy.

Jak bylo uvedeno výše existují tedy programy s integrovaným kreslením nebo přímo samostatné produkty. Další možnou klasifikací je na základě typu kreslení.

Programy ve kterých si uživatel může diagramy kreslit ručně lze použít SmartDraw, MS Visio, Dia, OmniGraffle, Inspiration, ConceptDraw. Tyto programy většinou využívají pro kreslení grafické rozhraní a podporují WYSIWYG (what you see is what you get). Existuje několik programů-knihoven pro vývojáře např. JGraph (Java), Netron (.NET).

Oproti první poměrně velké skupině programů existuje menší skupina, která nás ale více zajímá, a to programy podporující automatické generování diagramů. Jak již bylo uvedeno, automatické rozvržení závisí na typu diagramu, neexistuje žádný univerzální nástroj, který by dokázal uživateli dát to, co chce. Existují pouze programy, které se zabývají určitou skupinou problému. Např. v aplikaci SQL Enterprise Manager lze najít komponentu která dokáže reprezentovat tabulky v databázi (a jejich závislosti) pomocí diagramu. Dalším příkladem je Class Diagram ve vývojovém prostředí MS Visual Studio, který dokáže zobrazit diagram tříd v daném projektu. Existují tedy různá rozhraní pro ladící software, CASE nástroje, profilery které obsahují automaticky generované diagramy.

4 ATTIS.PM

Praktická část této práce bude směřována k automatickému generování diagramů pro tuto aplikaci.

ATTIS.PM je společným dílem firem ISDIM a ATTN. Aplikace se je v neustálém vývoji již několik let a nyní se připravuje verze 3.0. Program slouží k popisu organizační struktury a pro modelování business procesů ve firmách.

ATTIS.PM je platforma, která propojuje a integruje nástroje řízení organizace s cílem zvýšit její "výkonnost".

ATTIS.PM je sada sofistikovaných softwarových nástrojů pro zlepšování výkonnosti organizací s důrazem na procesní přístup k řízení. Poskytují přístup ke všem potřebným datům a informacím o organizaci, tak aby podporovaly všechny aspekty rozhodovacího procesu.

ATTIS.PM pomáhají zvyšovat výkon organizace. Jsou nástroji pro plánování, implementaci, komunikaci a kvalifikované zapojení zaměstnanců do kontinuálního zlepšování výkonnosti organizace. Dotýkají se a slouží všem pracovníkům, zvyšují hodnotu každého jejich rozhodnutí, zajišťují podporu pro vzájemnou spolupráci a jejich komunikaci, jsou pilířem motivačního a hodnotícího systému postaveného na sledování výkonu jednotlivých útvarů a každého jednotlivce. To vše v kontextu strategie.

softwarové nástroje ATTIS.PM

- ATTIS.ORG definování a modelování organizační struktury včetně přiřazení zaměstnanců
- ATTIS.BPM přehledný popis a modelování firemních procesů
- ATTIS.MBO podpora řízení prostřednictvím cílů
- ATTIS. modul BSC podpora metody strategického měření výkonnosti - Balanced ScoreCard
- ATTIS. modul motivace - provázání ukazatelů s motivačním systémem organizace

ATTIS.PM se skládá ze tří modulů.

Výchozím je modul Business Modelling (BPM) pro popis, správu a modelování všech firemních procesů, a to jak formou položkového popisu, tak i formou přehledných diagramů. V BPM modulu se definují základní vazby, jak mezi procesy, tak i v jednotlivých činnostech, včetně matic zodpovědností a přiřazení k jednotlivým zaměstnancům.

V modulu organisational (ORG) se v návaznosti na procesy a činnosti definuje organizační struktura firmy, včetně přiřazení jednotlivých zaměstnanců do této struktury.

V modulu management by objectives (MBO) se vytvářejí a provádí se "reporting" výsledků definovaných ukazatelů sloužících pro měření výkonnosti definovaných procesů a činností či měření výkonnosti jednotlivých organizačních útvarů.

Modul MBO je možné rozšířit o nástavbu balanced scorecard (BSC) pro řízení a vyhodnocování úspěšnosti strategie, sledování plnění strategických cílů, to znamená práci s ukazateli v kontextu této metody. Jednotlivé ukazatele je možné v modulu motivace přiřadit jednotlivým pracovníkům a vytvořit tak základ motivačního a hodnotícího systému, postaveného na sledování jejich výkonnosti.

Účelem implementace automatického rozložení je usnadnění práce uživatelům. V praxi se při modelování procesů ve firmách používá několik stovek procesů, zobrazených na desítkách diagramů. Uživatel nemusí při modelování využívat kreslení diagramů, ale modely může definovat mimo a diagram nakreslit později. Pro výslednou reprezentaci modelu však zobrazuje pomocí diagramu, který musí ručně napozicovat, což při uvedeném počtu diagramů uživatele obtěžuje.

V další kapitole se zaměříme na analýzu jednotlivých diagramů a následně na návrh implementace.

4.1 Analýza

Jedním z požadavků pro verzi 3.0 bylo automatické generování rozvržení diagramů pro organizační strukturu a pro diagramy procesů.

Při analýze bylo zjištěno, že se pro rozvržení příliš nehodí ani jedna z metod uvedených výše. Proto bude napsán nový algoritmus využívající následující poznatky:

- algoritmy a postupy uvedené v teoretické části.
- analýza již nakreslených diagramů uživateli v předchozích verzích programu

Výsledný algoritmus rozvržení musí splňovat tyto požadavky:

- uživatel nesmí čekat při rozvržení
- navrhnout diagram tak, aby uživatel do něj již nemusel zasahovat

Je zřejmé, že algoritmus nelze navrhnout jednoduše, ale že to bude „běh na dlouhou trať“. Bude to neustálý proces sběru požadavků od uživatelů a jejich implementace. Problémem bude subjektivnost těchto požadavků.

4.1.1 Analýza diagramů organizační struktury

Obecně diagram organizační struktury bude v nové verzi úplnou novinkou. V předchozích verzích aplikace se organizační struktura popisovala stromovou hierarchií. Proto analýza vychází právě z této hierarchie.

Diagramy organizační struktury budou stromy (acyklické grafy). Na diagramu se bude vyskytovat více nezávislých organizačních struktur. Diagram může být velmi rozsáhlý, může obsahovat několik stovek vrcholů.

Algoritmus bude ke kreslení diagramu přistupovat jako ke kreslení stromu. Je potřeba zajistit následující požadavky:

- diagram bude zobrazen jako strom vizualizovaný vertikálně. Předchůdce bude zobrazen nad svými následovníky a bude vycentrován
- v grafu může byt více stromů, je potřeba nalézt všechny kořeny všech stromů a zobrazit všechny stromy vedle sebe.
- hrany zobrazovaného stromu musí být kresleny ortogonálně (pouze vertikálně a horizontálně)

4.1.2 Analýza diagramů modelů procesů

V předchozích verzích již bylo implementováno kreslení diagramů procesů. Analýza tedy vychází z již nakreslených diagramů uživateli. K dispozici již je několik stovek diagramů procesů, ze kterých se lze inspirovat.

Diagram procesů neobsahuje příliš mnoho vrcholů, obvykle se objevují diagramy s menším počtem vrcholů (do cca. 10). Ale nejsou výjimkou složitější diagramy.

Diagram je velmi podobný diagramu toků (flow-chart). Diagram má většinou (není pravidlem) jeden počáteční vrchol, od kterého se diagram odvíjí. Diagram se může větvit, větve následně spojovat, obsahovat cykly a je zakončen několika vrcholy. Diagram modelů procesů zobrazuje určitou časovou následnost jednotlivých procesů (vrcholů), která musí být z diagramu zřejmá.

Diagram může být kreslen vertikálně i horizontálně, při větším počtu procesů (vrcholů) by diagram měl být kreslen ve více řádcích-sloupcích.

Diagram modelů procesů neobsahuje pouze procesy, ale i jiné entity, související s danou metodikou (jsou to dokumenty, přílohy, vstupy, výstupy, zodpovědnost apod.). Pro analýzu je postačující, že to jsou „méně“ důležité vrcholy, nezasahující do toku diagramu. Tyto entity budou kresleny co nejblíže k procesům, na které jsou navázány. Tyto vrcholy by neměly křížit hlavní tok diagramu.

4.2 Návrh implementace

Aplikace ATTIS.PM využívá pro kreslení diagramů open-source komponenta Netron Graph Library. Knihovna již částečně podporuje automatické rozvržení diagramů, avšak všechny jsou neuspokojivé. Tato práce se bude snažit rozšířit knihovnu o podporu výše uvedených algoritmů.

4.2.1 Rozvržení diagramu organizační struktury

Algoritmus bude rekurzivně procházet stromem, metodou „pre-order“. Nejprve se bude snažit napozicovat potomky, a potom teprve vypočte pozici předka. Vše pak provede pro každý zjištěný strom v diagramu.

4.2.2 Rozvržení diagramu modelu procesů.

Algoritmus nelze příliš předem určit, protože bude implementován iterativním postupem (implementace - testování – sběr výsledků – implementace ..). Toto bude opakováno, dokud nebudou diagramy uspokojivě pozicovány.

Návrh algoritmu bude založen na podstatě časové následnosti jednotlivých procesů. Jednotlivé uzly budou pozicovány od počátku diagramu. Po zjištění pozice uzlu, budou vypočteny další možné pozice pro nové vrcholy. Pokud nebude na diagramu již místo (příliš mnoho větví), bude diagram zpětně posunut.

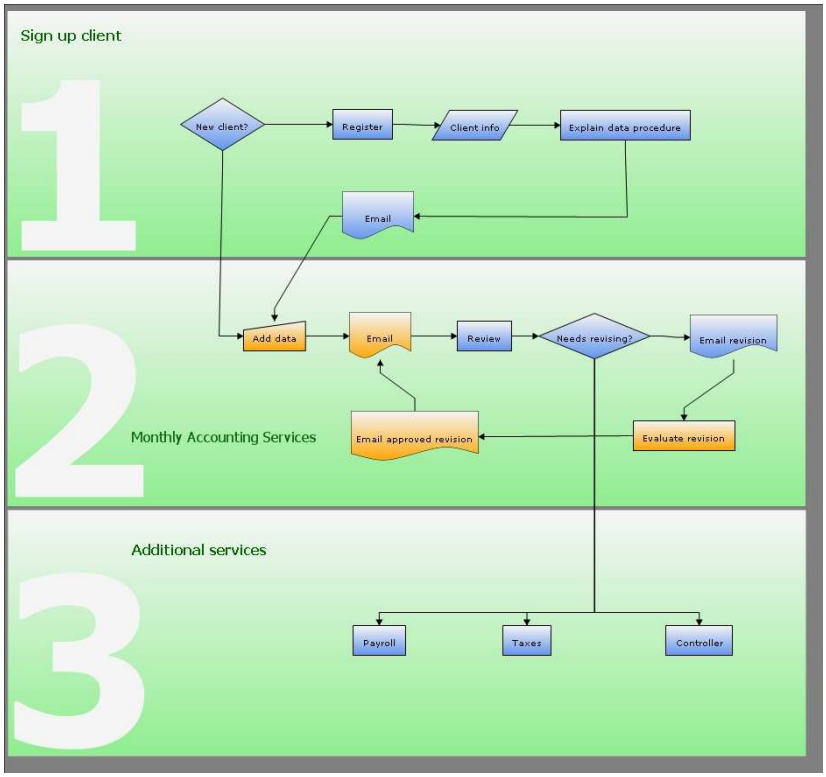
U každého vrcholu je potřeba zajistit informaci, zdali se účastní hlavního toku diagramu, nebo zdali je to jen méně důležitá entita (více v analýze).

5 Netron Graph Library

Aplikace ATTIS.PM používá pro kreslení diagramů knihovnu Netron. Celá praktická část této práce byla proto implementována jako rozšíření této knihovny. Knihovnu si v krátkosti představíme

Knihovna netron je určena pro kreslení diagramů, kreslení grafů a souborem postupů pro rozvržení diagramů.

Umožňuje vytvářet inovativní, vysoce interaktivní aplikace s minimem programování v C#. Je plně upravovatelný pomocí využití dědičnosti ze základních tříd. Piktogramy a jejich funkce, které programátor potřebuje využívat pro uzly a hrany grafu mohou být zakompilovány v separátních knihovnách a dynamicky linkovány pomocí mechanizmu reflexe .NET platformy za běhu.



5-1 Ukázka typického výstupu z Netron

Knihovna je intenzivně používána v mnoha aplikacích po celém světe. S knihovnou je dodáváno i bohaté grafické uživatelské rozhraní – Cobalt IDE, jehož hlavním účelem je ukázat možnosti knihovny. Cobalt IDE obsahuje mnoho vlastností a technik, které umožňují vytvořit aplikaci s podporou kreslení diagramů téměř okamžitě.

5.1 Automatické rozvržení

Poslední verze knihovny má kompletně přepsanou architekturu automatického rozvržení a nyní nově podporuje i fyzikou inspirovaná rozložení. Programová část této práce využívá starší verzi knihovny, která tyto algoritmy neobsahuje. Další práci to ale v ničem nebrání, protože tyto algoritmy jsou pro aplikaci ATTIS.PM z logiky diagramů nepoužitelné. Pro informaci čtenáře je ale uvedeme. Názvy budou uvedeny v originále, jednak z důvodu problémového překladu a jednak z důvodu názvu tříd, který je odvozen od názvu algoritmu.

- Ballon tree layout – potomci uzlu jsou seskupováni v okolí předka
- Force directed layout – systém fyzikálních sil (pružnost, gravitace ...)
- Fruchterman-Rheingold – jeden z klasických algoritmů
- Radial tree layout – potomci jsou seskupování kolem kořenového uzlu
- Random layout – jeden z nejjednodušších, uzly jsou rozprostřeny náhodně po plátně
- Standard tree layout – klasický algoritmus, který však není dostačující pro potřeby ATTIS.PM, bude ukázáno v kapitole o implementaci vlastního stromového rozložení.

5.2 Vestavění do aplikací

Knihovna je sama o sobě k ničemu; její síla je až v kontextu business aplikace, do které je vestavěna. Mnoho aplikací spatřilo světlo světa až právě díky vestavěné možnosti kreslení diagramů a také faktu že kód aplikace a kód kreslící části může být plně oddělen.

Dále budou uvedeny některé z klíčových vlastností podpory vestavění Netron do aplikací.

- Podpora WinForm designer komponenty – podporuje mechanizmus .NET designu, v přípravě je i pro ASP.NET
- Bohatá množina komponent pro uživatelské rozhraní pro změnu veškerých parametrů diagramu (styl čar, barva výplní, zakončení čar, z-souřadnice atp.)
- Podpora SmartTag, což je novinka od Microsoft v designování aplikací ve VS 2005
- V poslední verzi je to i konečně dlouho chybějící podpora Undo/Redo
- Podpora skriptování C# - runtime kompilace vlastních skriptů
- Bohaté API a DOM pro jednoduché rozšíření vlastností knihovny
- Open Source!! – jedna z klíčových vlastností pro využívání v aplikacích, GPL licence bez jakýchkoliv triků
- Kompletně přepsáno s podporou .NET 2.0
- Bohatá dokumentace
- Cobalt IDE přináší další již předimplementované vlastnosti
- Simulace dat – podpora pro simulaci konečných automatů apod.

5.3 Analýza grafů

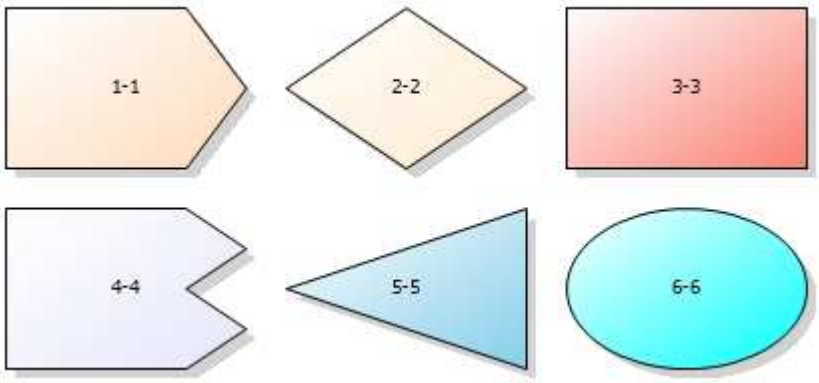
Knihovna Netron podporuje několik z mnoha algoritmů pro analýzu grafů – Kruskalův, Dijkstrův, Floydův, Primův algoritmus, algoritmus kritické cesty, prohledávání do šířky i do hloubky.

5.4 Piktogramy a předlohy piktogramů

Netron obsahuje základní množinu jednoduchých piktogramů. Jedním z požadavků zadavatele programu ATTIS.PM bylo navrhnout několik vlastních piktogramů. Knihovna Netron výborným způsobem umožňuje návrh vlastních piktogramů díky dědičnosti ze základní třídy.

Požadavky na piktogramy pro ATTIS.PM:

- Černé orámování
- Stínovaná výplň
- Stín



5-2 Ukázka piktogramů ATTIS.PM

5.5 Další vlastnosti knihovny

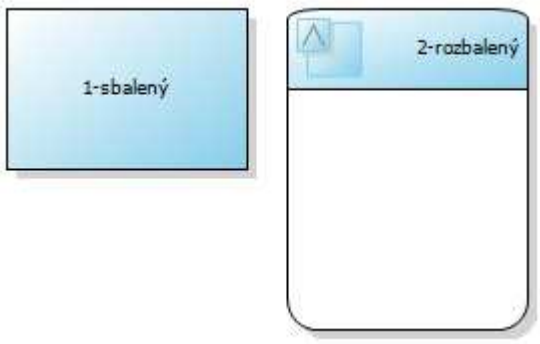
Podpora schránky:

- Plná podpora: Kopírovat, Vyjmout, Vložit
- Obrázky i text může být kopírován z/do schránky
- Podpora Drag and Drop pro vkládání textu na plátno

Vlastnosti zobrazení, operace s plátnem

- Přiblížení/oddálení pomocí myši, kláves, tlačítek
- Více stránek, obdoba záložek
- Více vrstvé piktogramy

- Jednotky reálného světa – palec, centimetr
- Seskupovaní entit s možností rozbalení/sbalení – bylo implementováno speciálně jako požadavek pro ATTIS.PM



5-3 Rozbalitelné piktogramy

- Mřížka a přichytávání k ní
- Z-souřadnice pro určení vrstvy entit

Ostatní nástroje pro kreslení (mimo kreslení grafu)

- Kreslení křivek
- Kreslení polygonů
- Volný text
- Obdélníky, elipsy

5.6 Licence, podpora, aktualizace

Podle současného licenčího ujednání je knihovna volně k použití. Jako většina Open Source projektů je podpora řešena pomocí diskuzního fóra uživatelů, blogu autora; neexistuje tedy žadná oficiální podpora.

Současný vývoj je zastaven, protože autor přešel k vývoji zcela nové knihovny pro kreslení zvané Unfold. Hlavní změnou je opuštění GDI programování a přechod k paradigmatu Microsofts Presentation Fundations.

5.7 Shrnutí

Netron je výbornou knihovnou pro podporu kreslení diagramů v aplikacích. Kód knihovny je napsán přehledně, dobře dokumentován a snado se dá upravovat. Mínusem je ale slabší podpora, nicméně lepší než žádná.

Z osobní zkušenosti velmi doporučuji používání této knihovny.

6 Implementace

Následující část se bude věnovat praktické práci. Programátorská část byla z velké části řízena požadavky ATTIS. Kromě hlavního tématu – automatické rozvržení diagramů, se týkala dílčích úprav knihovny Netron. Některé ukázky této práce byly uvedeny v předchozí kapitole.

Byly implementovány čtyři algoritmy pro automatické rozvržení diagramu. Dva z nich jsou vcelku primitivní, ale budou uvedeny, protože jsou často v aplikaci ATTIS.PM používány. V současné době jsou používány v aplikaci všechny čtyři algoritmy.

U každého metody bude uveden účel, pro jaký byl programován a jak je v aplikaci ATTIS.PM používán, vysvětlen jeho algoritmus, uvedení výsledků.

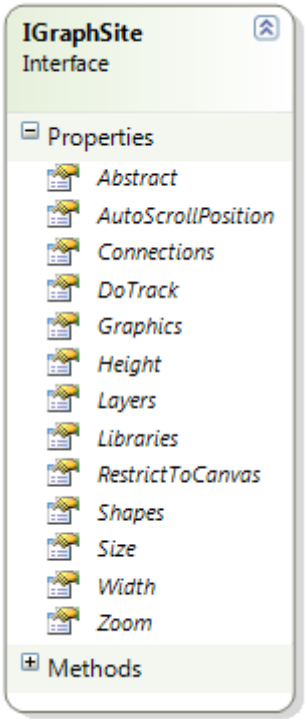
6.1 Hierarchie podstatných tříd Netron

V této podkapitole budou uvedeny pouze podstatné třídy, které byly v praktické části použity a souvisí s tématem automatického rozvržení diagramů.

Jak již bylo uvedeno v předchozí kapitole, Netron podporuje automatické rozvržení diagramů. Pokud chce programátor vytvořit vlastní rozvržení, stačí, aby ve svou třídu zdědil z abstraktní třídy GraphLayout a implementoval všechny povinné položky.

Nejdůležitějšími položkami je hlavní metoda StartLayout, ve které programátor provede svůj kód rozvržení. Informace o diagramu pak získá z vlastnosti Site, které implementuje rozhraní IGraphSite.

6.1.1 Rozhraní IGraphSite.



6-1 Rozhraní IGraphSite

Uvedené rozhraní je velmi podstatné a má za úkol abstrahovat matematickou podstatu grafu. K tomu slouží dvě základní vlastnosti. Shapes a Connections, jsou to v podstatě kolekce uzlů resp. hran.

Dále rozhraní definuje kolekci Libraries, což je soubor informací o knihovnách ve kterých se mají hledat informace o tvarech piktogramů.

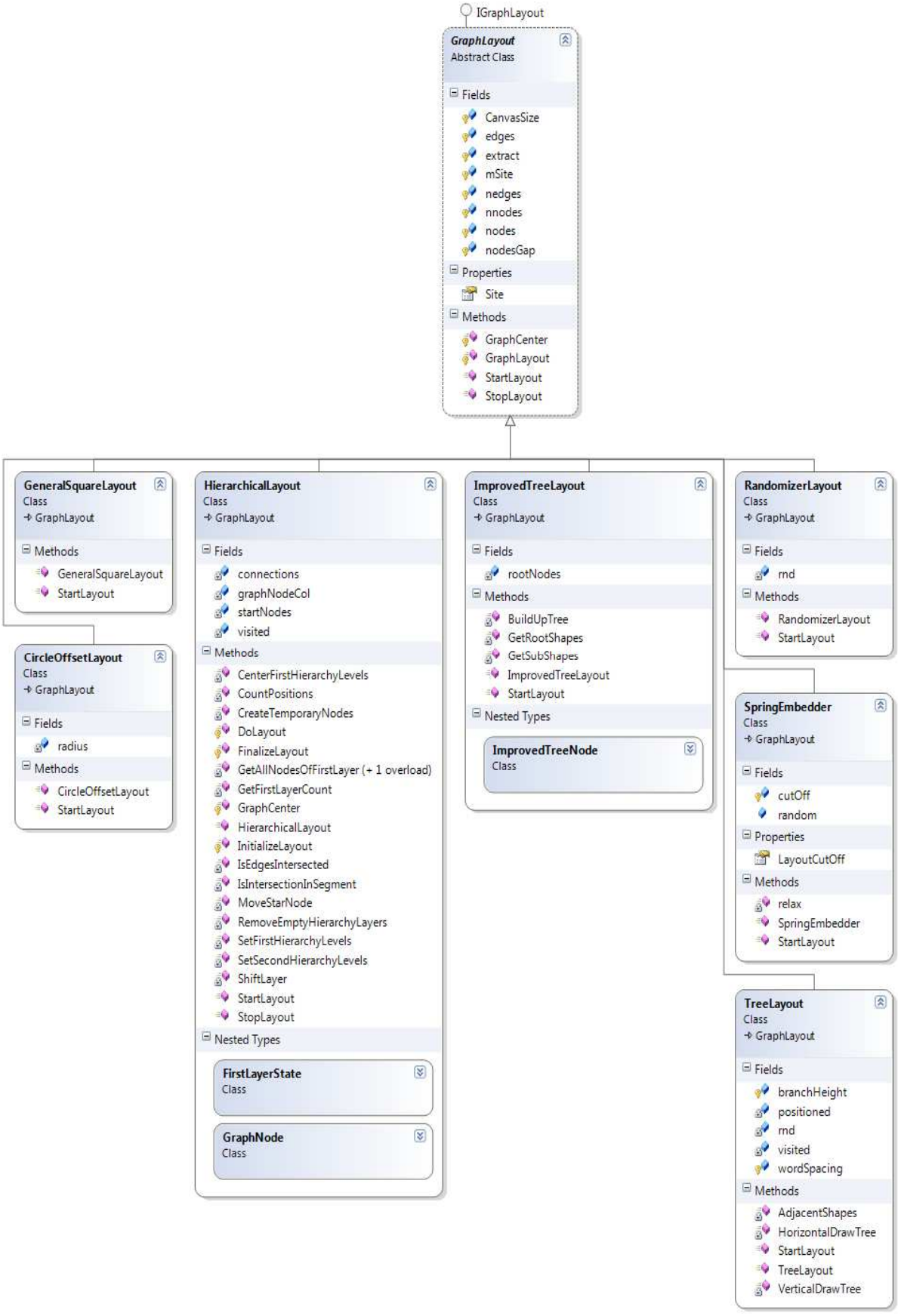
Layers obsahuje informace o vrstvách. Vrstvy nebyly součástí řešení algoritmů pro automatické rozvržení.

Width, Height obsahují šířku, výšku kresleného diagramu na plátně.

Zoom obsahuje informaci o aktuálním přiblížení diagramu.

6.1.2 Hierarchie GraphLayout

Na následujícím obrázku je zobrazena hierarchie tříd knihovny Netron, přímo související s automatickým rozvržením diagramů. V levé části jsou pak zobrazeny nově implementované třídy. Pravý, poslední sloupec obsahuje pro názornost i původní třídy, řešící také automatické rozvržení.



6-2 Hierarchie GraphLayout (vygenerováno pomocí Class Diagram ve Visual Studio 2008)

6.2 Circle offset layout

Nejedná se zcela o metodu rozvržení diagramu, ale o částečnou úpravu rozvržených diagramů.

6.2.1 Účel

V aplikaci ATTIS.PM lze na diagramu zobrazovat u každé entity její pod-entity. Například procesní krok má své vstupy, výstupy. Tyto entity jsou na diagramu zobrazovány piktogramem (menších rozměrů) s hranou vedoucí k mateřskému procesnímu kroku. Uživatel si může zvolit, které pod-entity chce zobrazit a které ne.

Pokud se některý z těchto piktogramů zobrazuje na diagramu poprvé, je tento piktogram nakreslen do levého horního rohu. Uživatel je takto seznámen s informací, že je zobrazen poprvé a může si jej umístit, kam potřebuje.

Nastaví-li si zobrazení více piktogramů, musí si je všechny umístit, což může zdržovat. Místo toho lze využít právě výše zmíněného algoritmu.

Algoritmus najde všechny nepozicované piktogramy a posune je náhodně na kružnici okolo mateřského uzlu. Předpokládá se, že existuje právě jedna hrana spojující piktogram s mateřským uzlem.

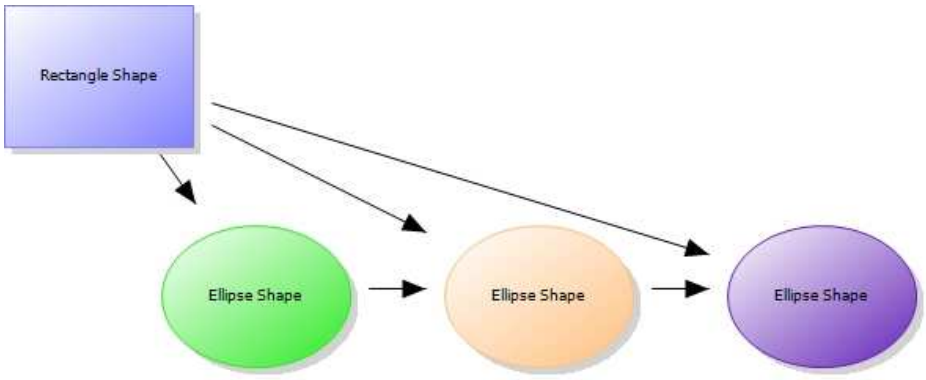
6.2.2 Popis algoritmu

Algoritmus je jednoduchý a bude popsán slovně.

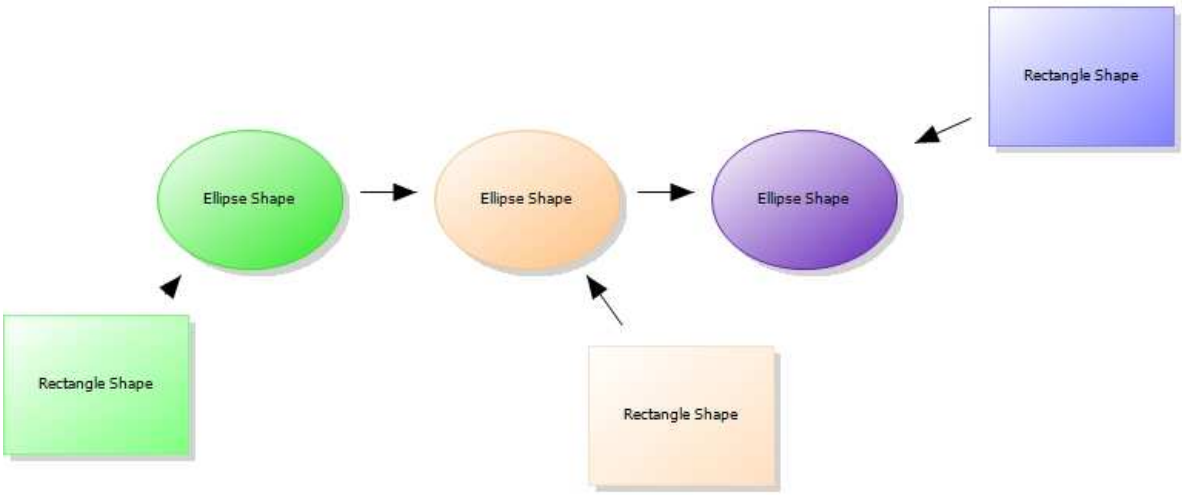
- Získej kolekci všech nepozicovaných uzlů
- Pro každý uzel v této kolekci:
    - Najdi jeho mateřský uzel
    - Zvol náhodně umístění na kružnici kolem mateřského uzlu
    - Překrývá se pozice s jiným piktogramem?
        - Opakuj pozicování

6.2.3 Výsledky

Na následujících obrázcích je ukázána situace před a po použití algoritmu. Poznamenejme, že obrázek je pouze ilustrativní, v reálných situacích jsou pozicované diagramy složitější.



6-3 Poprvé zobrazené piktogramy na plátně



6-4 Automatické rozložené piktogramy

6.3 General square layout

Opět jednoduchý algoritmus, který se však často používá.

6.3.1 Účel

V ATTIS.PM existují dva druhy procesních diagramů. Diagramy činností a diagramy procesních kroků. Činnosti jsou v podstatě složky pro diagramy procesních kroků. Mezi činnostmi neexistují žádné vztahy. Z matematického hlediska jsou to grafy s prázdnou množinou hran.

Obecně uživatelé kreslí tyto diagramu jako soubor uzlů uspořádaných do obdelníku, častěji čtverce.

Algoritmus je inspirován právě uživateli a umožňuje tuto operaci provést v jednom kroku.

6.3.2 Popis algoritmu

Metoda je dvouprůchodová a její složitost lineární. Slovně by se dala zapsat:

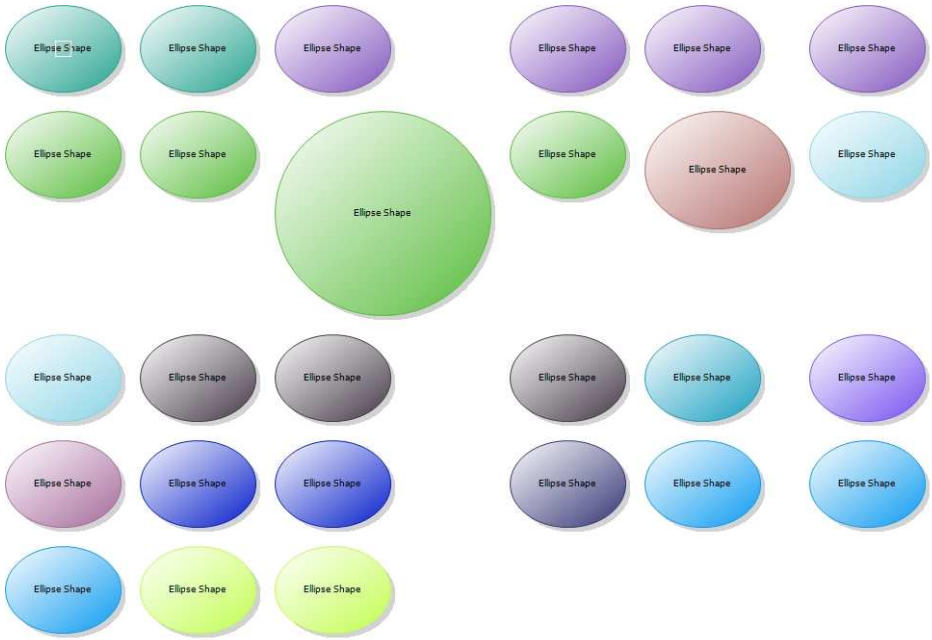
- Vypočti stranu čtverce (pokud není parametrizováno, bere se automaticky druhá odmocnina z počtu pozicovaných uzlů
- Přes všechny sloupce čtverce
    - Přes všechny řádky
        - Vypočti index uzlu
        - Nastav uzlu na vypočteném indexu na aktuálně procházený řádek
        - Vypočti další možnou pozici v závislosti na výšce piktogramu aktuálního uzlu
- Přes všechny sloupce čtverce
    - Přes všechny řádky
        - Vypočti index uzlu
        - Nastav uzlu na vypočteném indexu na aktuálně procházený sloupec
        - Vypočti další možnou pozici v závislosti na šířce piktogramu aktuálního uzlu

6.3.3 Důvod dvouprůchodovosti algoritmu

Čtenáře jistě napadne, proč byl použit dvou-průchodový algoritmus pro řešení problému. Důvodem je splnění požadavku na kontrolu velikosti piktogramů. Pokud není velikost piktogramů stejná, algoritmus automaticky upravuje mezery mezi piktogramy, tak aby všechny piktogramy byly ve stejném řádku, sloupci, tak jak je znázorněno na obrázku v podkapitole výsledky.

6.3.4 Výsledky

Algoritmus je již plně používán v ATTIS.PM. Obrázek je pořízen testovací aplikace, rozložení reálných dat bohužel nebylo k dispozici.



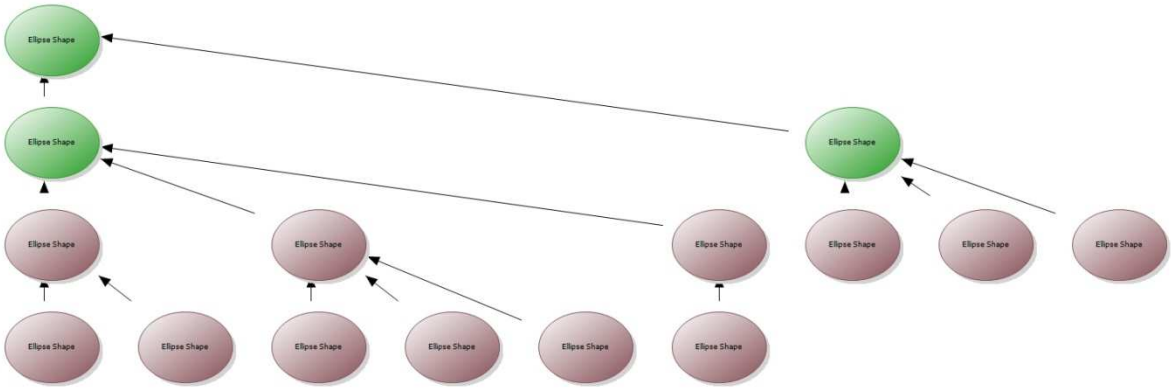
6-5 Schopnost algoritmu reagovat na různé velikosti piktogramů

6.4 Improved tree layout

Původně se mělo jednat pouze o úpravu původního algoritmu v Netronu, později bylo osamostatněno do vlastního algoritmu.

6.4.1 Účel

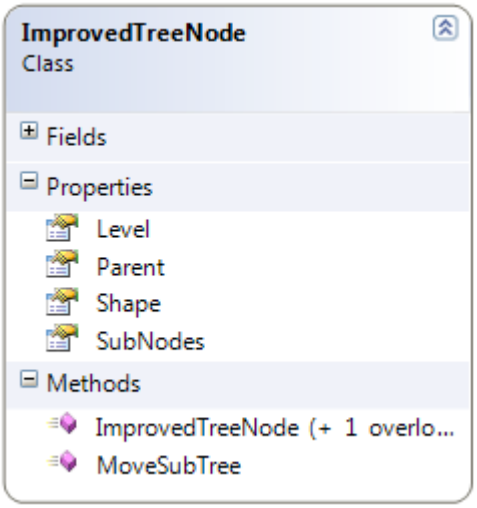
Kromě procesního modelu lze v aplikaci ATTIS.PM možné popisovat organizační strukturu. Graf organizační struktury je strom. Původně se v aplikaci používal originální algoritmus z Netronu. Hlavním problémem algoritmu bylo nevycentrování předka nad potomky, což vylučuje pohodlné použití a uživatelé si na to často stěžovali.



6-6 Rozložení původním algoritmem dodaným v Netron Graph Library

6.4.2 Popis algoritmu

6.4.2.1 ImprovedTreeNode



6-7 ImprovedTreeNode třída

Pro lepší přehlednost algoritmu vznikl vnořený typ ImprovedTreeNode reprezentující uzel stromu.

Třída zaobaluje uzel stromu a obsahuje následující vlastnosti:

- Level – určující úroveň ve stromu
- Parent – odkaz na typ ImprovedTreeNode reprezentujícího předka uzlu
- SubNodes – kolekce potomků uzlu

Dále obsahuje rekurzivní metodu MoveSubTree, která vrací šířku podstromu daného uzlu

6.4.2.2 Sestavení stromu

V prvním kroku se v daném grafu naleznou všechny stromy a sestaví se z nich pomocná struktura typu ImprovedTreeNode.

Nejprve se naleznou v grafu všechny potenciální kořeny stromů. Jsou to ty, ze kterých nevede žádná hrana. Pak se postupně prochází od kořene a metodou průchodu pre-order se sestavuje struktura stromu.

Při sestavování stromu se do vlastnosti Level ukládá úroveň uzlu ve stromu, ze kterého pak bude vypočten jeden z rozměrů (vertikální) pozice.

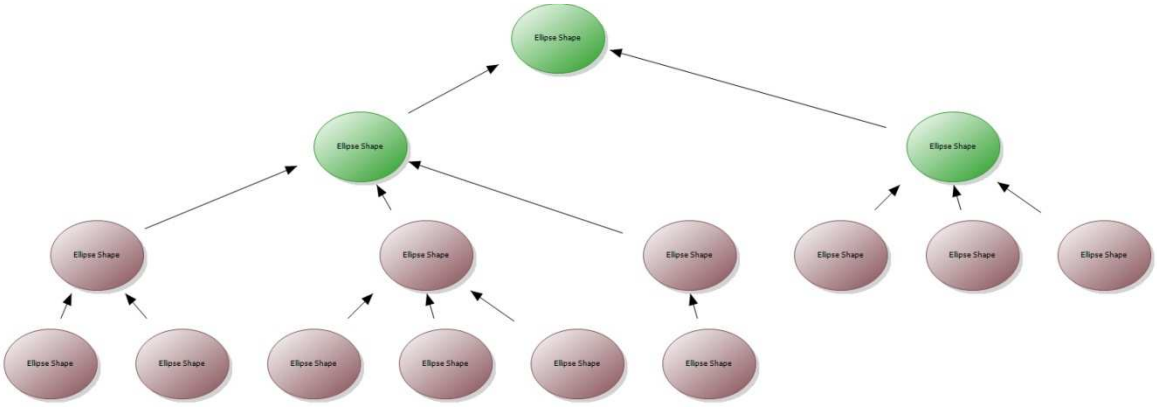
6.4.2.3 Pozicování podstromu

Hlavní metodou řešící pozicování je rekurzivní metoda MoveSubTree, která je postupně volána pro kořeny všech nalezených stromů.

Metoda prochází stromem do hloubky a při návratu je vrácena šířka podstromu. Z této hodnoty je pak vypočtena pozice předka – vycentrování. Tím je vypočten druhý rozměr (horizontální) pozice.

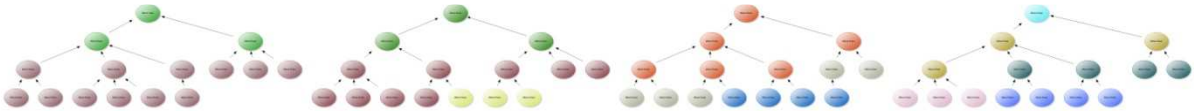
6.4.3 Výsledky

V tomto algoritmu bylo dosaženo velmi pěkných výsledků, a to nejen v „akademickém“ použití, ale i přímo v nasazení na reálných datech v aplikaci ATTIS.PM pro zobrazení organizační struktury.



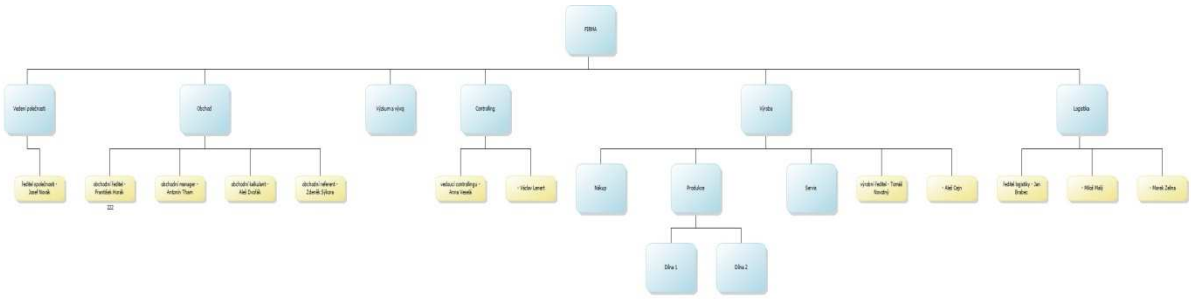
6-8 Aplikace algoritmu na stromovou strukturu

Algoritmus dokáže detekovat více stromů v diagramu. Byl to jeden z požadavků na algoritmus řešený výhradně pro ATTIS.PM



6-9 Detekce více stromů

V praxi se v aplikaci ATTIS.PM využívá. Následující obrázek byl pořízen přímo jako výstup skutečných dat.



6-10 Strom organizační struktury

6.5 Hierachical layout

Jedná se o nejsložitější rozvržení řešené v této práci.

6.5.1 Účel

ATTIS.PM dokáže procesní strukturu importovat z excelovských tabulek. Pokud uživatel této možnosti využije, veškeré diagramy související s importem nejsou pozicovány. Uživatel si je samozřejmě může nakreslit, což ho ale zdržuje, nebo může využít automatického rozložení.

Diagramy procesů jsou obecné orientované grafy, které obsahují jeden nebo více počátečních uzlů (takových ke kterým nevede žádná hrana, pouze v nich hrany začínají).

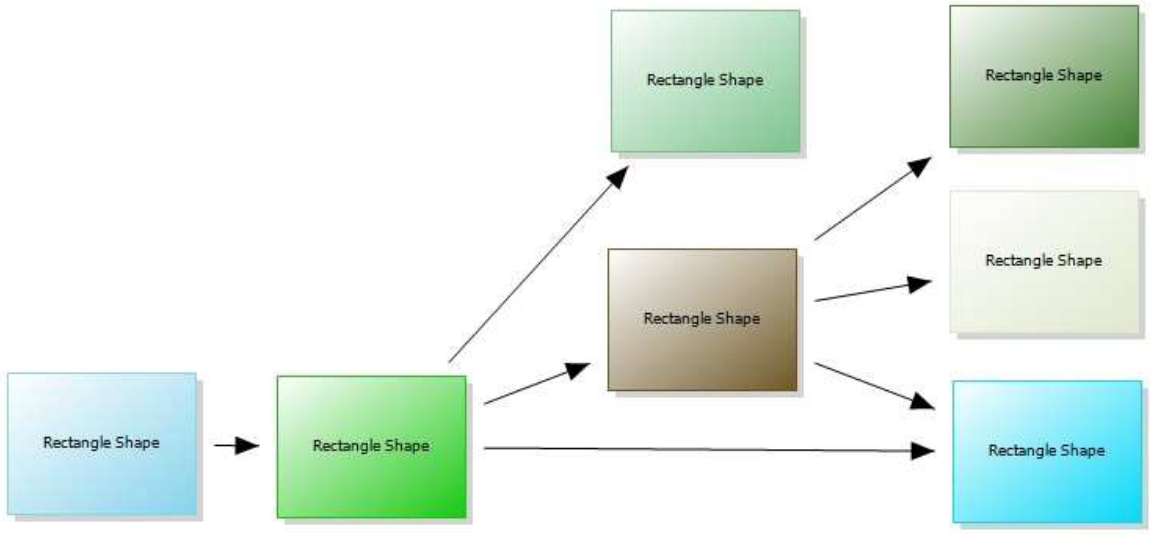
Oproti analýze uvedené v teoretické části, bylo upuštěno od zahrnutí sémantického významu jednotlivých uzlů, z důvodu pracného napojení algoritmu na ostatní aplikace. Tímto zůstává algoritmus zcela nezávislý na klientské aplikaci.

6.5.2 Popis algoritmu

Hlavní algoritmus je rozložen do pěti hlavních kroků.

- Pozice sloupce pro daný uzel
- Vytvoření pomocných uzlů
- Pozice řádku pro daný uzel
- Centrování ve sloupcích
- Vypočtení pozice

Uvedených pět kroků bude vysvětleno v následujících podkapitolách. Pro lepší pochopení bude každý krok ukázán na pozicováním následujícího diagramu.



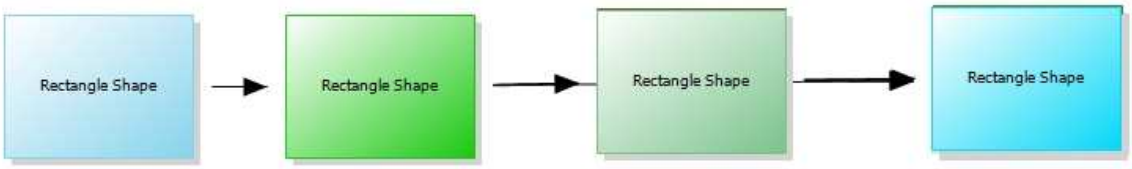
6-11 Ručně pozicovaný diagram

6.5.2.1 Pozice sloupce pro daný uzel

V prvním kroku se pro každý uzel zjistí jeho sloupec. Rekurzivně se prochází grafem pomocí upraveného algoritmu prohledávání do šířky stromu.

Pro každý uzel se zjistí jeho potomci a všem potomkům se zvýší jejich pozice ve sloupci, vše se provádí pouze tehdy, mají-li potomci menší pozici ve sloupci než jejich předek. Tím je zajištěna detekce smyček v diagramu.

Výsledek tohoto kroku je zobrazen na následujícím obrázku. První krok nastavuje pouze první rozměr – sloupec, proto jsou všechny piktogramy v jednom řádku.

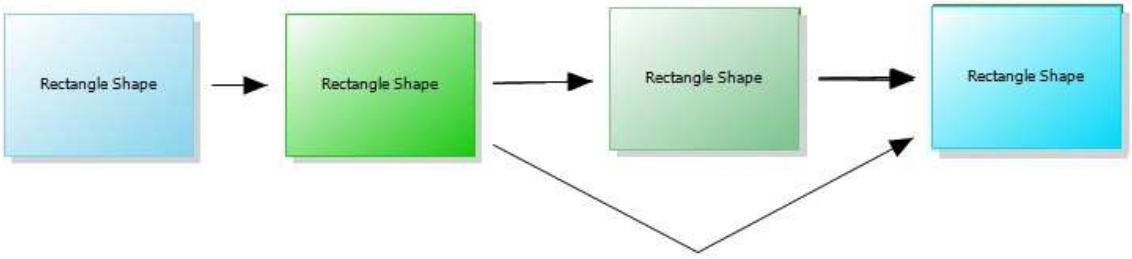


6-12 Pozice sloupce pro každý uzel

6.5.2.2 Vytvoření pomocných uzlů

Dalším krokem je zavedení pomocných uzlů. Pokud existuje hrana, jejíž počátek začíná ve sloupci i a končí ve sloupci j, tak že $|i - j| > 1$, je pro tuto hranu v každém sloupci k, kde $i < k < j$, případně $j < k < i$ (pro opačně orientované hrany) zaveden pomocný uzel.

Na obrázku je zobrazen jeden pomocný uzel (zalomení hrany). Pro lepší viditelnost je zobrazen v novém řádku, ve skutečnosti by ale po tomto kroku nebyl vidět, protože vše by bylo nakresleno v jednom řádku.



6-13 Pomocné uzly

6.5.2.3 Pozice řádku pro daný uzel

Jedná se o časově nejnáročnější krok. Cílem tohoto kroku je nalézt takovou pozici každého uzlu v řádku, tak aby se pokud možno nekřížila žádná hrana. Pokud nelze diagram nakreslit bez křížení hran, algoritmus po tomto zjištění upraví svou podmínku pro počet maximálního křížení hran.

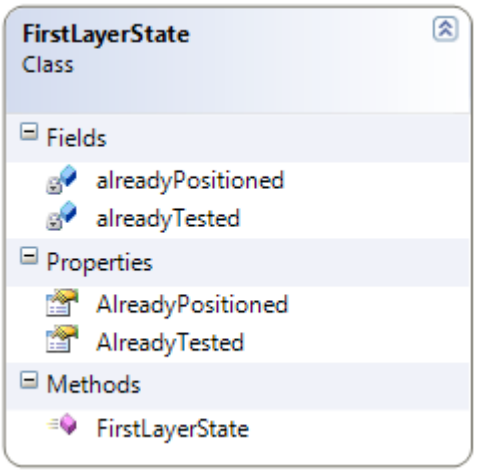
Algoritmus prochází postupně přes všechny sloupce a kontroluje, zdali existuje křížení hran vedoucích ze sloupce i do sloupce $i+1$, a snaží se nalézt.

Existují dva možné konce:

- Končící v řešení posledního sloupce – řešení nalezeno

- Končící v řešení prvního sloupce – vyzkoušeny všechny možné kombinace, řešení však nenalezeno, následuje úprava podmínky

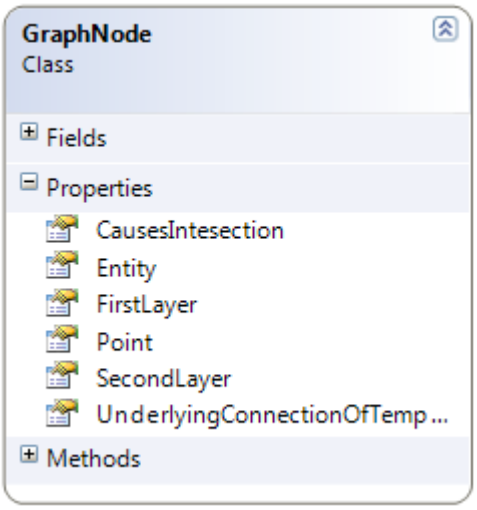
Pro tento algoritmus byly založeny dvě pomocné třídy.



6-14 FirstLayerState třída

Třída FirstLayerState slouží k uchování stavu pozicování při přechodu mezi sloupci. Stav je uchován ve dvou kolekcích. Jedna z nich je kolekce již napozicovaných uzlů v daném sloupci a druhá obsahuje již testované uzly v dané sloupci. Pokud první kolekce obsahuje všechny uzly sloupce, pak existuje řešení pro daný sloupec a algoritmus přechází k pozicování dalšího sloupce. Pokud se ale dříve naplní kolekce AlreadyTested všemi uzly daného sloupce a v kolekci AlreadyPositioned je méně uzlů, pak je řešení nenalezeno a vrací se řešení k předchozímu sloupci.

Ukládání stavů při přechodech je řešeno zásobníkem. Zásobník je v tomto případě náhradou rekurze.



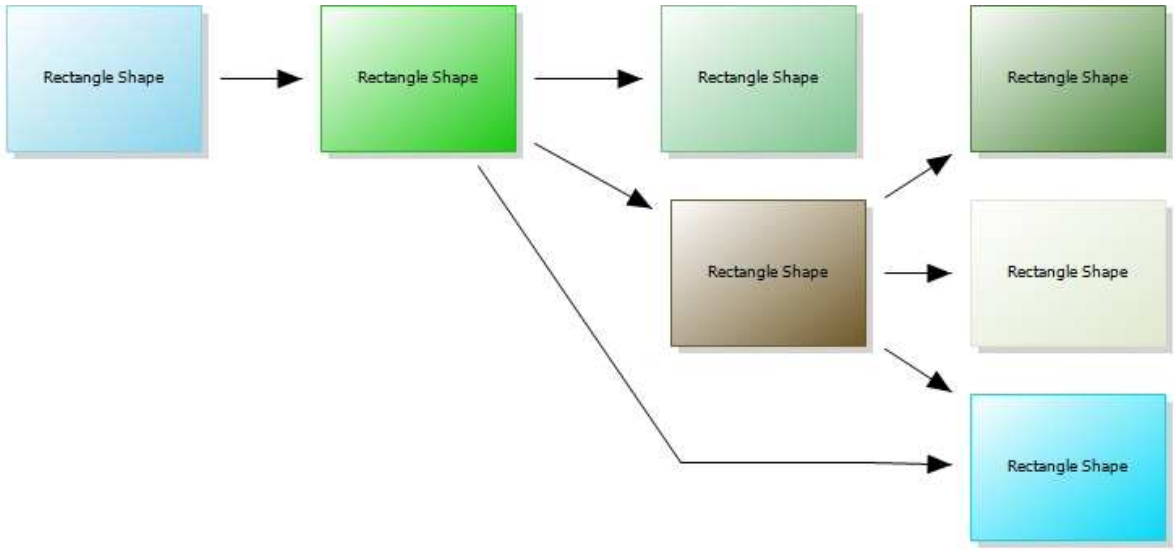
6-15 GraphNode třída

GraphNode je třída popisující uzel grafu. Důležitou vlastnosti je CausesIntersection, která označuje příznakem, zdali současná pozice uzlu způsobuje překřížení hran.

Dále FirstLayer, SecondLayer označuje aktuální sloupec, řádek ve kterém se uzel nachází.

Poslední podstatnou vlastností je UnderlyingConnectionOfTempNode ve které je u pomocných uzlů uložena hrana pro kterou vznikly.

Pro uvedený diagram existuje více možností rozložení uzlů v řádcích. Každé z těchto řešení je stejně optimální, algoritmus vrací první z nalezených. Po provedení tohoto kroku nad ukázkovým diagramem dostáváme následující výsledek.



6-16 Diagram po kroku hledání pozice řádku

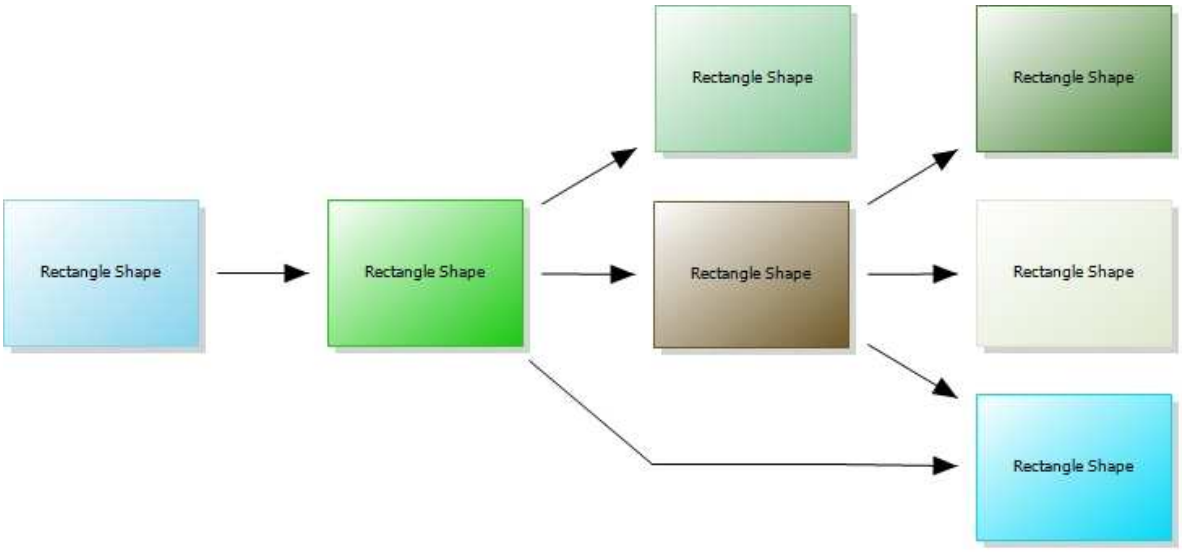
Tato část přináší zatím velké výkonnostní problémy. V dalším vývoji je potřeba se zaměřit hlavně na náhradu řešení problému křížení hran průchodem do šířky jiným algoritmem. Je uvažováno nad náhradou za backtracking, případně prohledávání do hloubky. Problém je, že předem nelze určit minimální počet křížení hran v grafu, resp. jedná se o řešení problému rovinnosti grafu, který lze řešit ale jeho časová náročnost je obrovská.

Problém prohledávání by ale šlo urychlit pomocí genetického algoritmu. Kódování genu lze provést jako souřadnice jednotlivých uzlů a jako fitness funkci zvolit počet křížení hran. Problémem je ale opět jak zvolit minimální křížení hran (minimální fitness) se kterou má algoritmus skončit.

6.5.2.4 Centrování ve sloupcích

Předposlední úpravou je centrování ve sloupcích. Je to spíše jen stylistická úprava, která však hodně pomáhá lepšímu výslednému zobrazení diagramu.

Nejprve se nalezne sloupec k s největším počtem uzlů. Uzly v ostatních sloupcích se pak posunou tak, aby hlavní linie diagramu vedla prostředním uzlem sloupce k



6-17 Finální podoba po pozicování

6.5.2.5 Vypočtení pozice

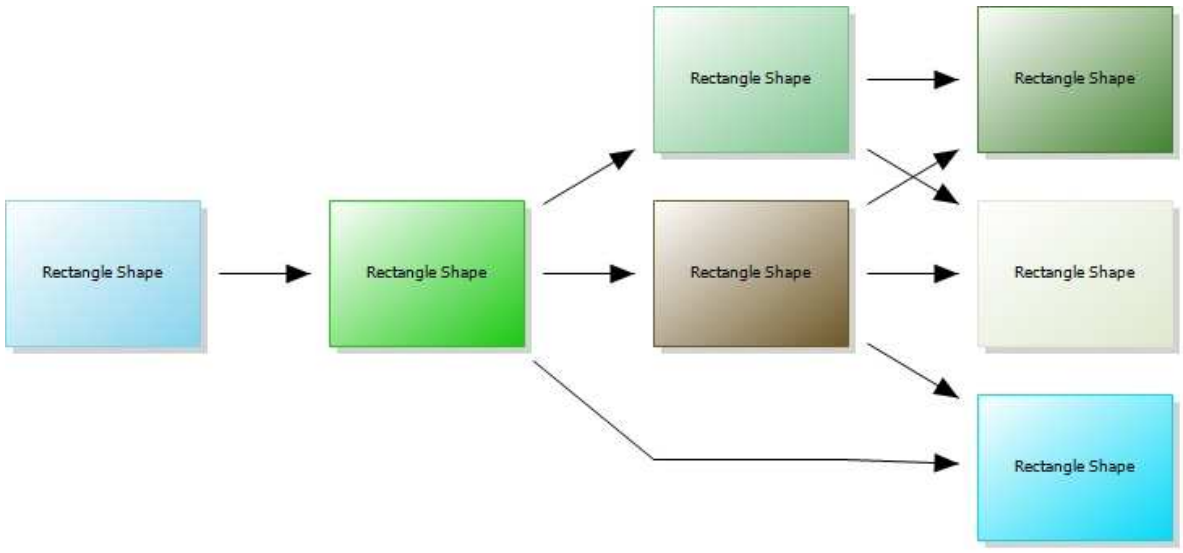
Poslední části je vypočtení přímo pozice piktogramu na plátně. Prochází se postupně všechny uzly, pokud se jedná o původní uzel, je piktogramu vypočtena jeho pozice, pokud se však jedná o pomocný uzel, je k tomuto uzlu nalezena příslušná hrana a na tuto hranu je přidán zlomový bod.

6.5.3 Výsledky

6.5.3.1 Planární a neplanární grafy.

V kapitole Popis algoritmu bylo ukázáno pozicování planárního grafu. Pokud do diagramu přidáme dvě hrany, získáme graf, který není planární; nelze jej tedy nakreslit bez křížení hran.

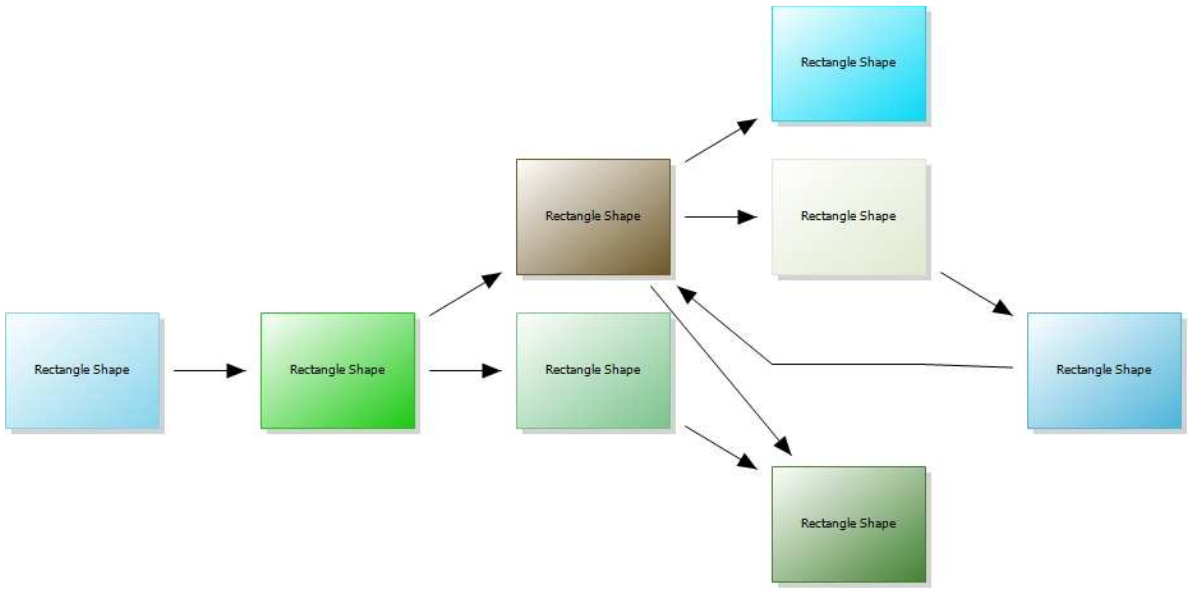
Algoritmus v tuto situaci detekuje a hledá řešení s jedním křížením hran. Výsledkem je uveden na obrázku.



6-18 Neplanární graf

6.5.3.2 Graf obsahující kružnici

Pokud do ukázkového diagramu přidáme další uzel a hranu tak, aby vytvářela kružnici, algoritmus toto opět detekuje na úrovni pozicování sloupců a hrany vedoucí zpět ignoruje.



6-19 Graf obsahující kružnici

6.5.3.3 Selhání algoritmu

Pokud zavedeme cyklickou hranu mezi prvními dvěma uzly, algoritmus pozicování selže. Problém totiž nastává v tom, že algoritmus není schopen detekovat počáteční uzel grafu. Není to ani tak problém algoritmu jako vlastnost grafu.

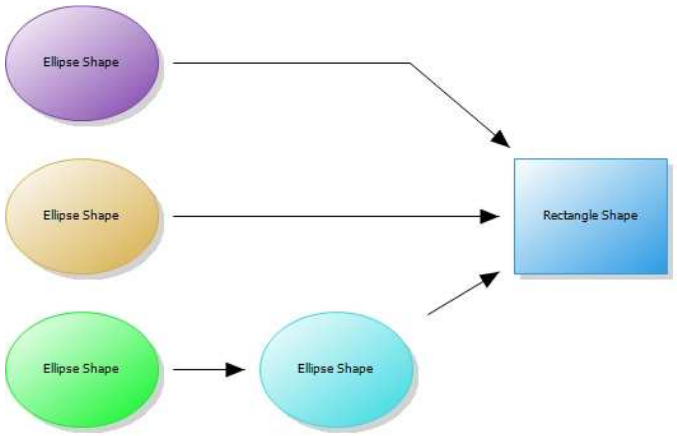
Možným řešením by bylo náhodné určení počátečního uzlu, což nebylo ve finální verzi do algoritmu implementováno.

Varianta byla testována, ale pro několikanásobné rozvržení dával algoritmus zcela odlišná řešení.

6-20 Selhání algoritmu

6.5.3.4 Rozložení stromu

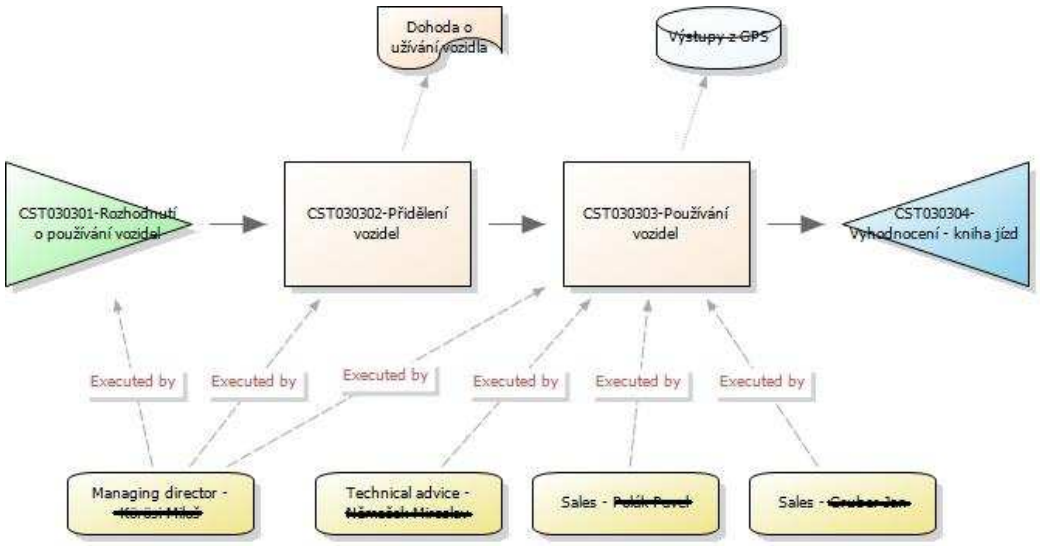
Zajímavé je pozorovat jak algoritmus reaguje na pozicování stromu a porovnání s algoritmem přímo pro rozložení stromu



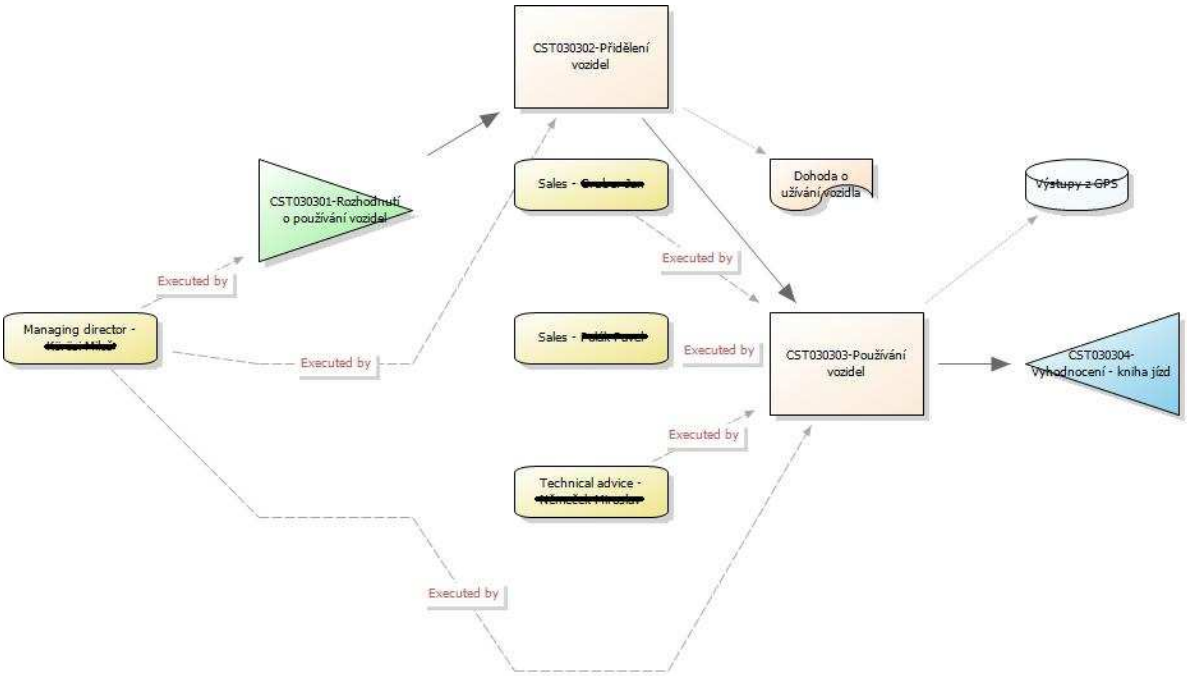
6-21 Aplikace na stromovou strukturu

6.5.3.5 Grafy z praxe – aplikace ATTIS.PM

V poslední řadě uvedeme srovnání diagramu rozvrženého uživatelem a algoritmem.



6-22 Rozložené uživatelem



6-23 Rozložené algoritmem

Lze pozorovat, že algoritmus poznal hlavní linii diagramu. Stejně jako uživatel algoritmus dokázal rozložit diagram bez překřížení hran. Hlavní nevýhodou v tomto rozvržení je to, že algoritmus neví nic o sémantice diagramu – hlavně vlastnost žlutých piktogramů, které se neúčastní hlavního toku diagramu a chybně je do něj zařadil. Nicméně i tak algoritmus našel obstojné řešení.

6.6 Testovací aplikace

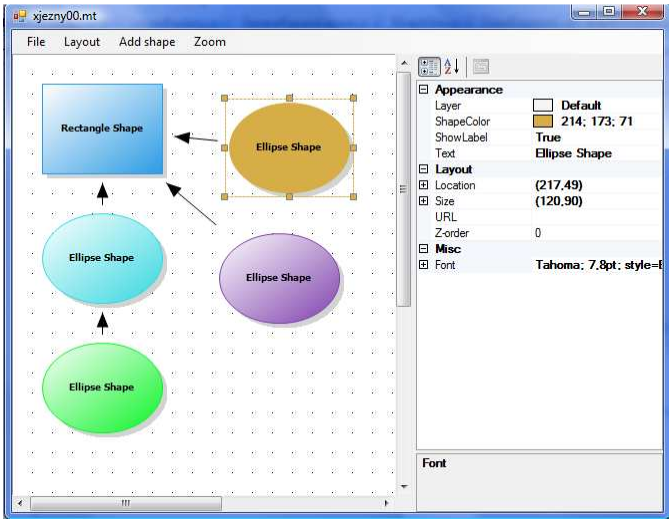
Pro účely testování a implementaci jednotlivých algoritmů byla napsána jednoduchá GUI aplikace, ve které si lze zmiňované algoritmy vyzkoušet.

Podporované funkce:

- Menu File
  - Otevírání diagramu
  - Ukládání diagramu
  - Uložení diagramu jako bitmapy
- Menu Layout
  - General square layout
  - Tree layout – původní z Netron
  - Hierarchical layout
  - Improved tree layout
- Menu Add shape
  - Přidání obdélníkového piktogramu
  - Přidání elipsového piktogramu

- Menu Zoom
    - Přiblížení diagramu
    - Oddálení diagramu
    - Automatické oddálení pro zobrazení celého diagramu
    - Zobrazení 1:1

Dále při dvojkliku na kterýkoliv piktogram se zobrazí tabulka jeho základních grafických vlastností



6-24 Hlavní okno pomocné aplikace

Aplikace je obsažena na kompaktním disku, který tvoří přílohu této práce.

7 Závěr

Původním záměrem této práce bylo navrhnout algoritmy pro automatické rozvržení diagramů. Rozvržení diagramů je ale v podstatě analýzou matematické struktury – grafu, kterou diagram znázorňuje.

Teoretická část této práce se proto z velké části zabývala teorií grafů. Vzhledem k tomu že teorie grafů je obsáhlá, byly uvedeny pouze přímo související partie. Byl kladen důraz na objasnění průchodu grafem, ať již do šířky nebo do hloubky, protože oba typy byly použity v praktické části.

V praktické části byly uvedeny čtyři specializované algoritmy pro rozvržení diagramů. Dva z nich jsou jednodušší a dva složitější. Nicméně všechny čtyři jsou používány v aplikaci ATTIS.PM, což bylo cílem této práce.

Nesložitějším a nejvíce prostudovaným byl hierarchický layout, který jak bylo uvedeno v analýze, nejde jednoznačně určit za dokončený, protože se jedná o iterativní vývoj.

Dalším možný vývoj jde rozdělit podle hlavních aspektů na dvě části. Z hlediska výkonu a z hlediska kvality.

Hierarchické rozvržení má díky použitému algoritmu, prohledávání do hloubky velkou časovou náročnost, což nepůsobí příliš dobrým dojmem. V práci byly navrženy i možné úpravy tohoto algoritmu a jejich problémy.

Z hlediska kvality bylo dosaženo u tohoto layoutu dobrých výsledků. Problémy se objevují u cyklických grafů, které jsou však z hlediska sémantiky popisovaných diagramu nepodstatné.

Práce na prvních třech algoritmech se dá považovat za ukončenou, poskytují uspokojivé výsledky. Vývoj hierarchického rozložení ukončen rozhodně není a bude dále pokračovat nad rámec této práce.

Literatura

[1] Force based layout, Wikipedia, Dokument dostupný na URL: http://en.wikipedia.org/wiki/Force-based_layout

[2] Drawing graph nicely, Algorithm, 1997, Dokument dostupný na URL: http://www2.toki.or.id/book/AlgDesignManual/BOOK/BOOK4/NODE168.HTM

[3] Automatic Graph Drawing, TU Wien, 2002, Dokument dostupný na URL: http://www.ads.tuwien.ac.at/research/graphDrawing.html

[4] Graph Drawing, Wikipedia, Dokument dostupný na URL: http://en.wikipedia.org/wiki/Graph_Drawing

[5] International Symposium on Graph Drawing, Wiipedia, Dokument dostupný na URL: http://en.wikipedia.org/wiki/International_Symposium_on_Graph_Drawing

[6] Graph Layouter Spring, CpAN, 2004, Dokument dostupný na URL: http://search.cpan.org/~pasky/Graph-Layderer-0.02/lib/Graph/Layouter/Spring.pm

[7] Graph drawing tutotial, Roberto Tamassia, Brown University, 2002, Dokument dostupný na URL: http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf

[8] Netron Graph Library, Dokument dostupný na URL: http://www.orbifold.net/Netron/info.php

[9] Netron Graph Library Licence, Dokument dostupný na URL: http://www.gnu.org/copyleft/gpl.html

[10] ATTIS.PM, Dokument dostupný na URL: http://www.attis.cz/pages/software.software.aspx

[11] Breadth first search, Wikipedia, Dokument dostupný na URL: http://en.wikipedia.org/wiki/Breadth-first_search

[12] Depth first search, Wikipedia, Dokument dostupný na URL: http://en.wikipedia.org/wiki/Depth-first_search

[13] Graph algorithms, Wikipedia, Dokument dostupný na URL: http://en.wikipedia.org/wiki/Category:Graph_algorithms

Seznam příloh

Příloha 1. CD – obsahuje zdrojové kódy, elektronickou podobu zprávy a spustitelný program xjezny00.mt