UNIVERZITA PALACKÉHO V OLOMOUCI
KATEDRA MATEMATICKÉ INFORMATIKY

ROČNÍKOVÝ PROJEKT

Editor Petriho sítí

leden 2001

Grézl Michal
Informatika, III.ročník

Abstrakt

Tento projekt umožňuje snadnou a rychlou editaci Petriho sítí. Je zpracován na platformě WIN32. Umožňuje export do obrázkového formátu EPS. Program pracuje jako ActiveX komponenta. Petriho síť je tvořena třemi základními stavebními kameny. Jsou to místo, přechod a hrana. Místo obsahuje jádra, hrana může být ohodnocena. Program je koncipován jako jednoduchý editor s využitím výstupu v různých koncových editorech, nejen v systému LaTeX, síť lze například vložit i do programu MS Word. Program je zpracován v jazyce C++ v programovacím prostředí Microsoft Visual C++.

Obsah

1 Popis a analýza

1.1 Úvod

Bylo zapotřebí řešit problém umístění většího množství obrázků Petriho sítí do textu pomocí ActiveX. Nejpřijatelnější řešení bylo vytvořit grafický editor, který by dokázal vytvářet a již existující Petriho sítě nějakým rozumným způsobem editovat. To vše s možností grafického výstupu do typografického systému LaTeX prostřednictvím konverze dokumentu do formátu EPS.

1.2 Analýza Petriho sítě

Petriho sítěmi je označována široká třída diskrétních matematických modelů (strojů), které umožňují popisovat specifickými prostředky řídící toky a informační závislosti uvnitř modelovaných systémů. Petriho sítě vznikly rozšířením modelovacích možností konečných automatů. Základním prostředkem popisu změny v modelovacím systému při použití konečného automatu je pojem stav a přechod mezi stavy.

V případě Petriho sítí je okamžitý stav konstituován určitými parciálními stavy spojenými s příslušnými místy sítě. K registrování dosažení jistého parciálního stavu odpovídajícímu příslušnému místu sítě slouží *značení místa*.

V případě, že místo modeluje logickou podmínku, pak toto místo obsahuje značku. V případě, že příslušná podmínka neplatí, pak toto místo značku neobsahuje.

Provedením přechodu (události) se odeberou značky vstupních míst (přestanou platit vstupní podmínky) a (vzniknou značky ve výstupních místech (začnou platit podmínky způsobené touto událostí).

1.3 Specifikace problému

Mělo by se rozhodnout jakým způsobem bude Petriho síť v dokumentu reprezentována. V zásadě jde o tyto položky:

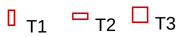
- jak budou vypadat parciální stavy
- jak budou zobrazeny události
- jak budou popisovány změny jednoho stavu na druhý
- jak budou značeny Petriho sítě

1.4 Parciální stavy

Parciální stavy, které se nazývají *místa Petriho sítě*, budou reprezentovány malými kružnicemi. *Značení místa* budou označeny černými tečkami nebo číslicí při vyšším počtu teček (pro lepší přehlednost).

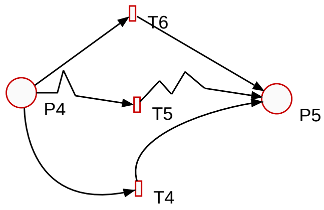
1.5 Události

Události, které se nazývají *přechody Petriho sítě*, budou reprezentovány pomocí obdélníků.



1.6 Změny stavu

Změny stavu budou vyjádřeny pomocí šipek. Eventuálně vícenásobná změna se zobrazí pomocí čísla nad šipkou.



1.7 Značení Petriho sítě

Každý *stav Petriho sítě* a každý *přechod Petriho sítě* bude označen jednopísmenným znakem s indexem.

1.8 Editace Petriho sítí

Petriho sítě se budou editovat myší. Kliknutím levým tlačítkem na ploše dokumentu si navolím *stavy Petriho sítě* nebo *přechody Petriho sítě* či *hrany*. Dopředu si na toolbaru určím co chci vytvářet (*stavy*, *přechody*, *text*, *hrany*). Atributy *stavu*, *přechodu*, *textu* nebo *hrany* se navolí pravým tlačítkem.

1.9 Souborové operace

Nahrávaní a ukládání Petriho sítí na disk bude ve svém vlastním formátu a dokumenty budou implicitně používat koncovku **.petrinet*.

1.10 Export do LaTeXu

Petriho síť půjde pomocí předdefinovaných maker exportovat do typografického programu LaTeX prostřednictvím obrázkového formátu EPS.

1.11 Další funkce editoru

Po vytvoření Petriho sítě bude umět (po inicializaci sítě) simulovat život Petriho sítě. Editor se přepne z editačního módu do módu simulace. Uživatel si bude moci klikat na jednotlivé *přechody Petriho sítě* a ihned uvidí změny v počtu značek (jader) ve *stavech Petriho sítě*. Kdykoliv bude možno celou síť uložit do souboru či exportovat do LaTeXu. Dále bude editor podporovat zadané rozhraní ActiveX.

2 Uživatelská část

Nyní se budeme podrobně zabývat možnostmi Editoru postupným projitím položek menu a dalšími možnostmi, které Editor nabízí.

2.1 Hlavní okno Editoru

Ovládání simulace slouží pro zapínání a vypínání simulačního módu. paleta nástrojů slouží pro výběr typu editačního nástroje. Vlastní editace se provádí v Editačním okně.

2.2 Hlavní menu Editoru

Budou popsány jen nejdůležitější položky a položky, které se normálně v aplikacích nevyskytují, MSVC++ vygeneruje standardní menu a u uživatele je předpokládána znalost rozhraní WINDOWS. Uživatelské menu se mění podle stavu, ve kterém se program zrovna nachází, pokud položka v menu není, nebo je zašedlá nejde, nebo by nemělo smysl ji používat.

2.2.1 Soubor

- Export

  Uoží dokument ve formátu EPS

2.2.2 Editace

- Vyber vše
  Vybere všechny objekty
- Přidej místo
  Vybere nástroj pro přidávání míst do dokumentu.
- Přidej přechod
  Vybere nástroj pro přidávání přechodů do dokumentu.
- Přidej poznámku
  Vybere nástroj pro přidávání textu do dokumentu.
- Přidej hranu
  Vybere nástroj pro přidávání hran do dokumentu.
- Vymazat
  Vybere nástroj pro mazání zvolených objektů.

2.2.3 Zobraz

- Podmenu Zvětšení
  Výběr zvětšení či zmenšení dokumentu. Tato položka používá stejné velikosti jako nástroj *lupa*.

2.2.4 Možnosti

- Podmenu Konverzní tabulka
  Zobrazí dialog obsluhující vestavěnou tabulku náhrady fontů při exportu do EPS.

2.3 Editační mód

V tomto módu je umožněna editace dokumentu (*míst*, *přechodů*, *hran*, *textových poznámek*). Výběrem z palety nástrojů se určí akce provedená po stisku levého tlačítka myši. Poklepáním pravým tlačítkem na objekt se zobrazí dialog, který umožní nastavit jeho atributy. Pravé tlačítko stisknuté nad volnou plochou v dokumentu zobrazí kontextové menu se základními akcemi editoru.

2.3.1 Popis jednotlivých nástrojů.

Výběr objektů Pokud je vybrán nástroj *výběru objektů* a uživatel klikne levým tlačítkem provede se toto:

- V případě míst, přechodů a poznámek se tyto orámují přerušovaným obdélníkem. Je to symbol toho, že objekt je označen a připraven pro práci se schránkou.
- V případě lomených *hran* se zobrazí všechny kontaktní body této hrany a je umožněno jejich přesouvání. V případě jednoduché *hrany* se také pouze orámuje přerušovanou čarou.
- Při kliknutí a tažení se zobrazuje rámeček. Všechny objekty uvnitř budou po puštění tlačítka označeny.

Místo, přechod, poznámka Nástroje *místo*, *přechod* a *poznámka* přidají místo, přechod a textovou poznámku. Při přidání *textové poznámky* se nejprve zobrazí dialog s jejími atributy a kde je možno zadat její obsah.

Lupa Nástroj *lupa* cyklicky mění zvětšení. Implementovaná jsou zvětšení 200%, 300%, zmenšení 50% a samozřejmě normální velikost.

Hrana Nástroj *hrana* přidá hranu a to následujícím způsobem: Pokud chce uživatel přidat jednoduchou hranu, může to udělat dvěma způsoby:

- Kliknutím na počáteční objekt a táhnutím na koncový, přičemž se interaktivně zobrazuje tvar hrany.
- Kliknutím na počáteční a poté na koncový objekt.

Pokud chce uživatel přidat lomenou hranu, klikne na počáteční objekt, potom kliká na místa uzlů hrany a nakonec na koncový objekt. Hrana se postupně vykresluje. Při stisku pravého tlačítka se vytvářená hrana zruší. Lomená hrana může být po dokončení vytváření změněna na Bezierovu křivku. Provede se to po stisku pravého tlačítka nad hranou a zaškrtnutím políčka Bezier v dialogu Atributy hrany. V současné implementaci se bezierova křivka řídí pouze podle prvních a posledních dvou uzlů hrany. Z toho plyne i nutnost mít minimálně tyto 4 uzly, pokud jich má hrana méně, upozorní uživatele chybový dialog na nemožnost provedení tohoto kroku.

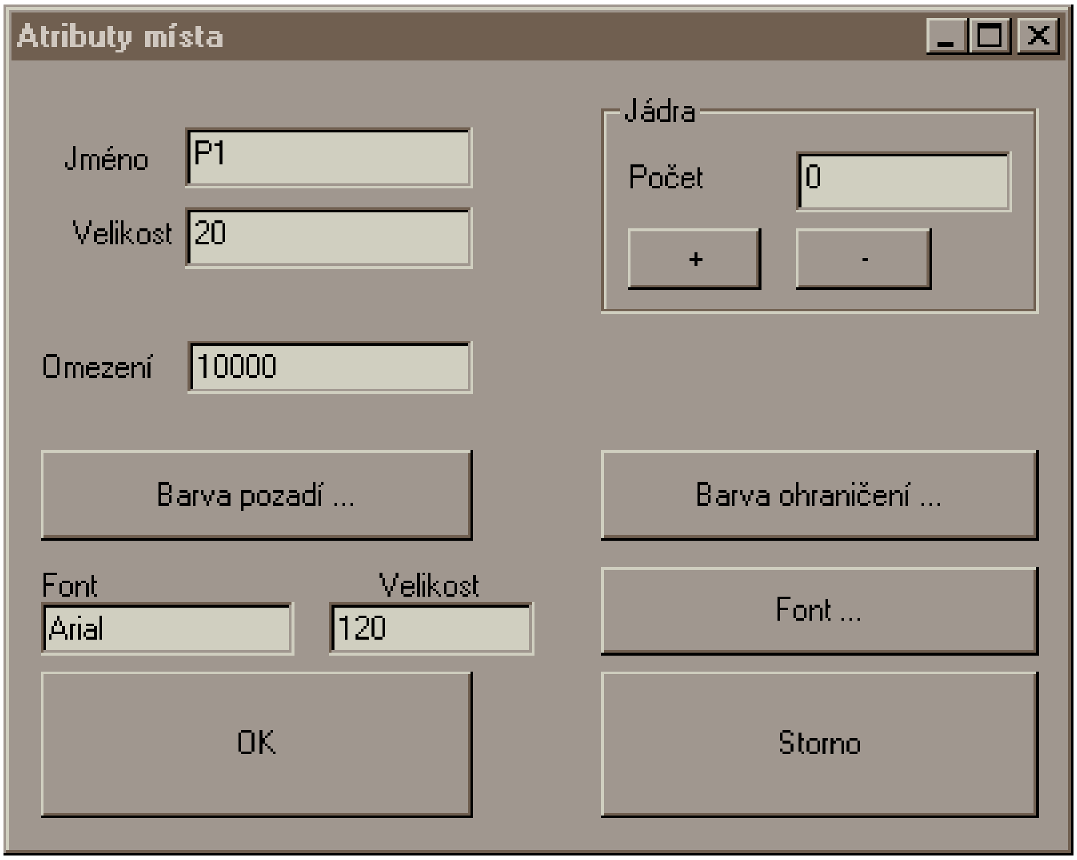
Uzel Nástroj uzel slouží pro manipulaci s uzly lomené hrany. Uživatel zvolí nástroj, stiskne levé tlačítko nad uzlem a táhnutím jej přesune. Při použití lomené hrany lze zapnout mód, při kterém je vykreslována jako beziérová v menu jejích preferencí. Potom tento nástroj slouží k editaci řídícího polygonu.

Mazání Jeden objekt se vymaže vybráním nástroje mazání (kříž) a stiskem levého tlačítka nad objektem. Pokud je objekt místo, nebo přechod a jsou na něj navázány nějaké hrany, pak se tyto vymažou také. Skupinu objektů lze vymazat tak, že se označí a pak se na ně levým stiskem tlačítka myši použije nástroj mazání.

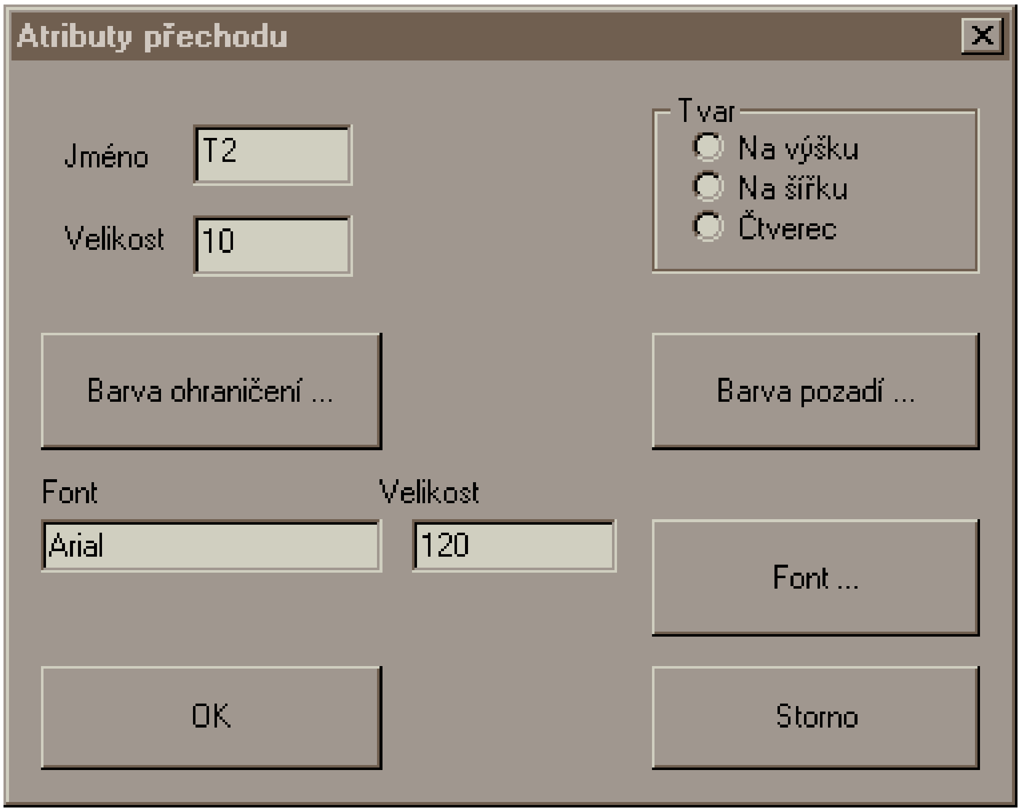
2.3.2 Atributy prvků Petriho sítě

Každý objekt má své atributy, které se dají nastavit po stisku pravého tlačítka myši nad objektem. Jmenovitě jsou to tyto:

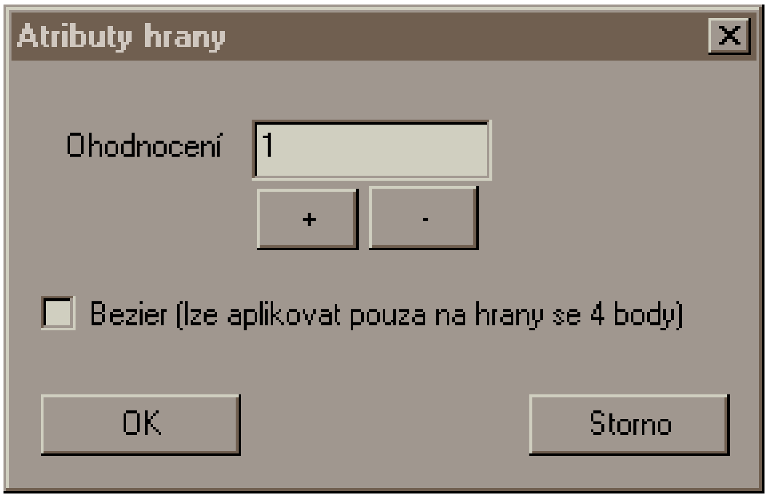
Místo Místo má tyto měnitelné atributy: Barva, velikost, počet jader, maximální počet jader, popiska, font a velikost fontu popisky.



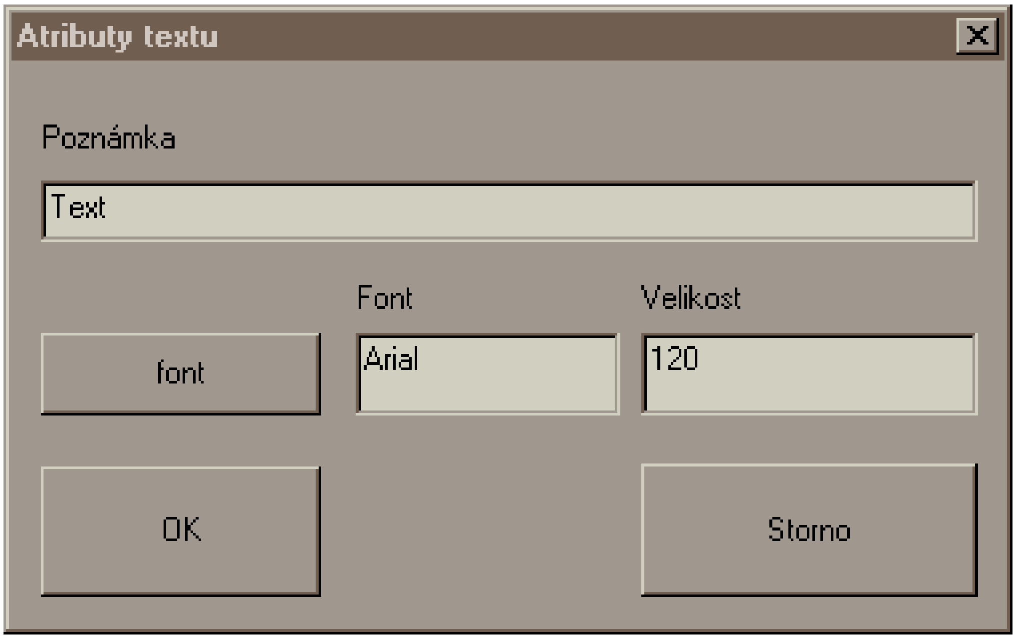
Přechod Přechod má tyto měnitelné atributy: Barva, velikost, tvar, popiska, font a velikost fontu popisky.



Hrana Místo má tyto měnitelné atributy: Ohodnocení, bezierova křivka.



Text Textová poznámka má tyto měnitelné atributy: Text, font a velikost fontu jimž je vykreslena.



2.4 Simulační mód

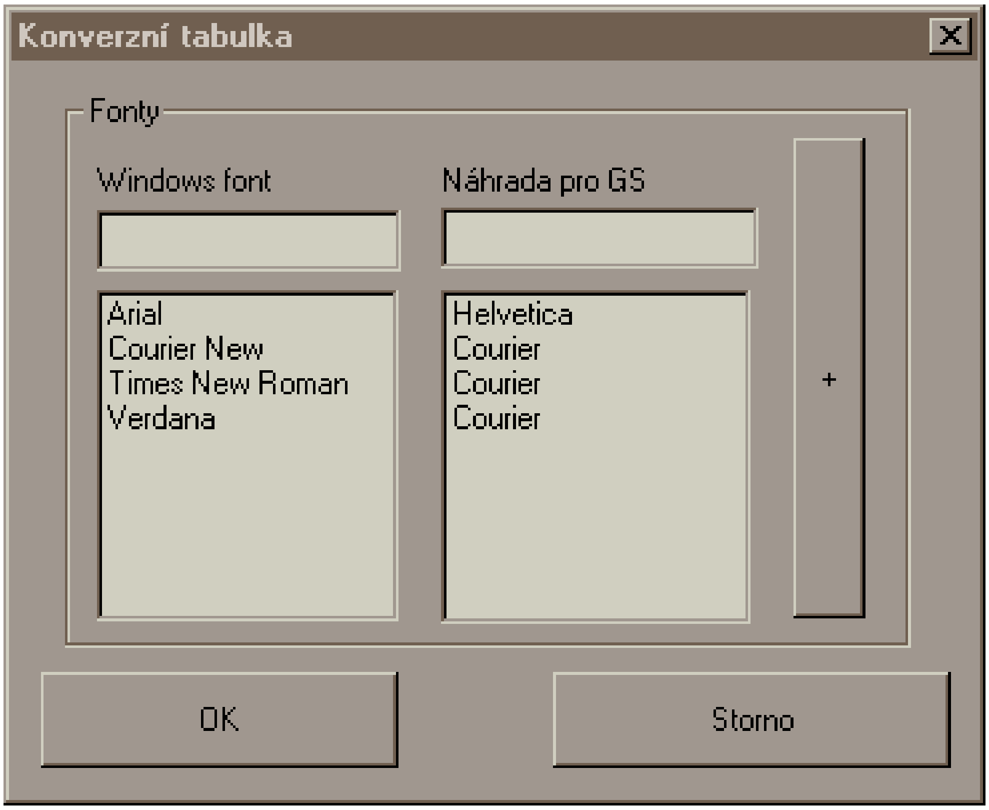
V tomto módu se provádí řízená simulace života *Petriho sítě*. Kliknutím levým tlačítkem myši na proveditelný *přechod* se tento *přechod* provede. To znamená, že ze všech *míst*, ze kterých vycházejí hrany do tohoto *přechodu* se odebere počet *jader* podle ohodnocení hrany a do všech *míst*, do kterých vedou hrany z tohoto *přechodu* se přidá počet *jader* podle ohodnocení hrany. Jsou zachována všechna omezení daná funkčností Petriho sítě.

2.5 Export do EPS, LaTeXu

Po zadání názvu souboru se provede konverze a export do formátu EPS. Provede se také konverze fontů podle vnitřní tabulky programu. Tento krok je nutný důvodu různých písem používaných v systému TeX a v systému Windows. Pokud je nutno převést obrázek na platformu s jiným kódováním znaků, je nutno převod řešit externím konvertorem. Toto řešení jsem zvolil z důvodu existence mnoha kvalitních programů pro převod kódování znaků. EPS je formát textový a je proto zcela bezpečné provádět převod takovýmito pomůckami.

ActiveX metoda vracející kód v LaTeXu provede export do uživatelem zvoleného souboru a poté volajícímu programu vrátí LaTeXový kód umožňující vložit tento obrázek. Při vložení zkopírovaných objektů do jiného programu se postupuje podle známých postupů editace vložených OLE dokumentů.

Na následujícím obrázku je ukázka dialogu, ve kterém uživatel nastavuje shodné fonty pro export.



3 Programátorská část

K softwarovému zpracování bylo zadáno zpracování na platformě WIN32 a programovací jazyk Microsoft Visual C++ s využitím objektové knihovny MFC (*Microsoft Foundation Classes*), která umožňuje zapouzdření API funkcí platformy Windows do objektových typů v C++ (podle knihy [2]). Nezmiňuji se zde přímo o MS Windows, protože program byl také testován ve WINE, což je implementace WIN API v operačním systému Linux.

Pro vygenerování kostry této aplikace byl použit *AppWizard*, který je součástí Microsoft Visual C++ a který ušetřil hodně práce. Základní kostra - rámec - aplikace byla vytvořena jako MDI (*Multiple document* - pro více dokumentů) s podporou automation a ActiveX.

3.1 Popis tříd vytvořených *AppWizardem*

AppWizard vygeneroval tyto třídy:

- CPetriApp
- CMainFrame
- CChildFrame
- CPetriDoc
- CPetriView

- CAboutDlg

3.1.1 Třída *CPetrinetApp*

Tato třída zapouzdřuje všechny funkce, které umožňují aplikaci spouštět a ukončovat. Dále provádí tyto činnosti:

- v MDI aplikaci vytvoří hlavní okno
- umožňuje zařazení a uvolnění aplikace v seznamu aktivních oken aplikací
- zpracuje příkaz pro otevření dokumentu nebo otevře prázdný dokument

3.1.2 Třída *CMainFrame*

Instance této třídy poskytuje možnost vytvoření většího množství dětských oken. Přičemž vytvoření dětského okna je podmíněno zprávou, která je vyvolána uživatelem.

3.1.3 Třída *CChildFrame*

Je vkládána do seznamu dětských oken, jehož správcem je třída *CMainFrame*, která se stará o správu a přidělování práv aktivnímu dětskému oknu.

3.1.4 Třída *CPetrinetDoc*

Tato třída je třída dokumentová a má na starosti práci s daty v dokumentu. Vždy, když je potřeba zobrazit data v oknech zavolá tato třída metodu *UpdateView()* a tím požádá třídu *CPetrinetView* o překreslení okna. Obsahuje pointer na objekt obsahující všechna data jádra (dále jen hlavní objekt).

3.1.5 Třída *CPetriView*

Tato třída je třída pohledová a má na starosti zobrazování dat v oknech. K zobrazování používá metodu *OnDraw()* a touto metodou překreslí okna, o jejichž překreslení požádá třída *CPetriDoc*. Kreslící funkce volané v metodě OnDraw() jsou obsaženy v souboru petrinet_draw.h a petrinet_draw.c. K metodám a instancím v *CPetrinetDoc* se přistupuje pomocí metody *GetDocument()*, tato metoda zajistí vrácení informací z právě aktivního editačního okna.

3.1.6 Třída *CAboutDlg*

Tato třída se stará o dialog *O Aplikaci*

3.2 Soubory tříd vygenerovaných k jednotlivým dialogům

Soubory jsou pojmenované prefixem dialog_ plus jméno objektu, nebo akce, vztahující se k dialogu. Jsou kompletně vygenerované pomocí MSVC++, jejich obsah je však pochopitelný i bez znalosti výše zmíněného produktu. Instance tříd odpovídají prvkům dialogu.

3.3 Hlavičkové soubory tříd tvořících jádro aplikace

3.3.1 defines.h

Obsahuje konstanty použité v projektu.

3.3.2 list.h

Obsahuje implementaci seznamu pomocí template.

3.3.3 stack.h

Obsahuje implementaci zásobníku pomocí template.

3.3.4 petri-core.h

Třída petri_net. Základní třída jádra, zapouzdřuje všechny objekty, provádí nad nimi různé operace. S touto třídou komunikuje manažer, tedy třída *CPetrinetDoc*.

instance:

- clist <place *> *list_place - Seznam ukazatelů na objekt place (místo).
- clist <transition *> *list_transition - Seznam ukazatelů na objekt transition (přechod).
- clist <hrana *> *list_edge - Seznam ukazatelů na objekt edge (hrana).
- clist <note *> *list_note - Seznam ukazatelů na objekt note (textová poznámka).
- int place_max_id - Nejvyšší id ze seznamu place.
- int transition_max_id - Nejvyšší id ze seznamu transition.
- int hrana_max_id - Nejvyšší id ze seznamu edge.
- int note_max_id - Nejvyšší id ze seznamu note.
- petri_font * fonts - Ukazatel na objekt třídy petri_font.
- font_struct *f_p - Popis fontu užitého při konstrukci nového místa.
- font_struct *f_t - Popis fontu užitého při konstrukci nového přechodu.

metody:

- petri_net() - Konstruktor.
- petri_net() - Destruktor.
- hrana *get_edge_by_id(int) - Vrací pointer na hranu podle jejího id.
- place *get_place_by_id(int) - Vrací pointer na místo podle jeho id.
- transition *get_transition_by_id(int) - Vrací pointer na přechod podle jeho id.
- note *get_note_by_id(int) - Vrací pointer na poznámku podle její id.

- int can_do_net(int) - Vrací true, pokud je možno aktivovat přechod s id v parametru.
- int do_net(int) - Provede přechod z jednoho p. stavu do druhého v přechodu id.
- int do_all() - Provede automatický přechod, zjistí které přechody je možno aktivovat a potom je všechny provede.
- void check_hrana() - Zkontroluje napojení hrany.
- int create_place(int, char*, char*, int) - Vytvoří nové místo. Parametry: počet jader, jméno, jméno fontu, velikost fontu. Vrací id nově vytvořeného objektu.
- int create_transition(int, char *, char *, int) - Vytvoří nový přechod. Parametry: reserved, jméno, jméno fontu, velikost fontu. Vrací id nově vytvořeného objektu.
- int create_hrana(int, char *) - Vytvoří novou hranu. Parametry: váha, jméno. Vrací id nově vytvořeného objektu.
- int connect(int hrana_id, int place_id, int transition_id, int direction) - Propojí přechod a místo hranou. Parametry: id hrany, id místa, id přechodu, směr (viz. defines.h). Tato funkce pracuje pouze s již vytvořenou hranou!
- int create_note(int, int, char *) - Vytvoří novou poznámku. Parametry: pozice x, pozice y, text. Vrací id nově vytvořeného objektu.
- void clear() - Zruší všechny objekty.
- int get_edge_coordinates(hrana *, int *, int *, int *, int *, int) - Funkce použitá při exportu, vrací ořezanou úsečku reprezentující hranu. Parametry" pointer na hranu, (parametry předávané odkazem) souřadnice, koeficient zvětšení.
- int eps_export(char *) - Export do souboru se jménem v parametru.
- int get_item_by_position(int, int, int *, int *) - Vrací id objektu a jeho ty podle pozice. Parametry: pozice x, pozice y, (předané odkazem) id, typ.
- int get_node_by_position(int, int, int *, int *) - Vrací true pokud je nalezen uzel nějaké hrany na zadané pozici. V posledních dvou parametrech je po návratu uloženo id hrany, pozice uzlu v hraně. Parametry: pozice x, pozice y, (předané odkazem) id, typ.
- int delete_item(int, int) - Smaže objekt. Parametry: id, typ.
- int delete_all() - Smaže všechny objekty.
- void select_font(petri_eps*, char *, int) - Nastaví font pro export objektu podle fontu objektu, font je převeden podle tabulky fontů, a jeho velikost je přizpůsobena formátu eps. Parametry: objekt eps, jméno fontu, velikost fontu.
- void get_bounding_box(int*, int*) - Zjistí oblast pro export.

3.3.5 petri-eps.h

Třída petri-eps slouží k exportu do EPS. Obsahuje metody, které generují postscriptový kód pro základní tvary použité v dokumentu jako kružnice, obdélník, text apod. PS kód se vrací jako char *. Třída používá filozofii podobnou WIN API co se týče barev a fontů, tj. nastaví se jednou a pak se používají dokud nejsou změněny.

instance:

- double R - Složka RGB, červená.
- double G - Složka RGB, zelená.
- double B - Složka RGB, modrá.
- int colors - Indikuje zda se generuje šedý, nebo barevný výstup.
- char * font - Jméno fontu.
- int font_size - Velikost fontu.
- int bbox_x, bbox_y, bbox_xx, bbox_yy - Oblast, kterou zabírá obrázek.

metody:

- petri_eps() - Konstruktor.
- void set_bounding_box(int, int) - Nastaví oblast pro obrázek.
- char *prolog() - Vrací postscriptový prolog.
- char *start() - Vrací začátek postscriptového kódu.
- char *start_page() - Vrací začátek stránky.
- char *end_page() - Vrací konec stránky.
- char *trailer() - Vrací konec ps souboru.
- char *eps_ellipse(int x, int y, int xrad, int yrad, int fill) - Vrací kód pro elipsu.
- char *eps_text(int x, int y, char *text) - Vrací kód pro text.
- char *eps_box(int x, int y, int xx, int yy, int fill) - Vrací kód pro obdélník.
- char *eps_normal_line(int x, int y, int xx, int yy) - Vrací kód pro čáru.
- char *eps_arrow_line(int x, int y, int xx, int yy)- Vrací kód pro čáru se šipkou.
- char *eps_bezier_line(int x, int y,int r1, int r2,int r3, int r4, int xx, int yy) - Vrací kód pro bezierovu křivku.
- void set_RGB(double r, double g, double b) - Nastavuje barvu.
- void enable_colors() - Zapíná barevný režim.
- void disable_colors() - Vypíná barevný režim.
- int colors_enabled() - Vrací stav barevného režimu.
- void set_font(char *) - Nastaví jméno fontu.
- void set_font_size(int) - Nastaví velikost fontu.

3.3.6 petri-font.h

Tento hlavičkový soubor obsahuje 2 třídy. Třída font_struct symbolizuje font, obsahuje jeho jméno a velikost. Třída petri_font obsahuje konverzní tabulku fontů a obsahuje metody pro manipulaci s ní.

Třída petri_font

instance: clist <font_pair*> font_table - Seznam fontů a jejich ekvivalentů.

metody: petri_font() - Konstruktor.
char * find_font_substitute(char *) - Metoda nalezne a vrátí náhradu za font, který je jí dán v parametru.
void set_font_pair (char *, char *) - Nastaví pár font, ekvivalent.

Třída font_struct

instance:

- char *font - jméno fontu
- int font_size - velikost

metody:

- font_struct() - Konstruktor.
- font_struct(char *, int) - Konstruktor. Parametry: jméno, velikost.
- font_struct(font_struct *) - Copy Konstruktor.
- ˜font_struct() - Destruktor.
- char *get_font() - Vrátí jméno.
- void set_font(char *) - Nastaví jméno.
- int get_font_size() - Vrátí velikost.
- void set_font_size(int f) - Nastaví velikost.

3.3.7 petri-graphics.h

Obsahuje sadu funkcí pro manipulaci s geometrickými objekty. Jsou zde obsaženy také základní definice bodu a čáry.

- struct mg_point - Struktura pro definici bodu.
- struct mg_line - Struktura pro definici úsečky.
- double vector(double, double); - Vrací rozdíl druhého a prvního parametru (tedy jednu složku vektoru)
- double line_length(int, int, int, int) - Vrací velikost úsečky dané parametry.

- mg_line clip_circle(int, int, int, int, int, int) - Vrací ořezanou úsečku. Ořezání je provedeno vůči kružnici, v jejímž středu je jeden z konců úsečky dané prvními čtyřmi parametry, pátý parametr je poloměr kružnice a šestý směr úsečky (tj. jestli v kružnici začíná, nebo končí viz. defines.h)

- int point_belongs(mg_point, mg_point, mg_point) - Vrací zda na úsečce definované prvními dvěma parametry leží bod daný parametrem třetím.

- mg_point clip_liang_barsky(double x1, double y1, double x2, double y2, double xw_min, double yw_min, double xw_max, double yw_max) - Ořeže úsečku v okénku. Vrátí bod, který je průsečíkem úsečky a obdélníku okénka. První čtyři parametry definují úsečku, druhé čtyři okénko.

3.3.8 petri-nodes.h

V tomto hlavičkovém souboru jsou obsaženy třídy pro objekty místo, přechod a hrana. Jejich obsah popíšu jen slovně. Každá třída obsahuje instance pro své vlastní atributy: Barvu, velikost, font atd. Místo a přechod obsahují seznamy příchozích a odchozích hran. Každá hrana obsahuje ukazatele na objekty které spojuje. Metody jsou funkce pro manipulaci s instancemi. Hrana obsahuje dále metody pro práci s lomenými čárami. Metody jsou velmi jednoduché a jen vzácně překračuje délka jejich kódu několik řádků.

Třída *place* Tato třída se používá k implementaci *stavů Petriho sítě*. Umožňuje vytváření a editaci těchto stavů a simulaci životních cyklů *Petriho sítě*.

Třída *transition* Tato třída se používá k implementaci *přechodů Petriho sítě*. Umožňuje vytváření a editaci těchto přechodů a simulaci životních cyklů *Petriho sítě*.

Třída *hrana* Tato třída umožňuje vytvářet, navazovat a editovat hrany. Tyto hrany slouží k logickému propojení *stavů* a *přechodů*, které je potřeba pro znázornění *Petriho sítě* a pro simulaci životních cyklů *Petriho sítě*.

3.3.9 petri-note.h

Tato metoda obsahuje definici třídy *note*, kterou je popsán objekt dokumentu textová poznámka. Její struktura je stejná jako třídy v petri-nodes.h. Obsahuje instance jako text poznámky, pozici, barvu, font, id a metody pro manipulaci s nimi.

3.3.10 petri-select.h

Tato třída zajišťuje manipulaci se seznamem vybraných objektů. Obsahuje seznam objektů, které se vyberou jedním z možných způsobů (viz. uživatelská část). Umožňuje také manipulaci, přesun, s vybranými objekty. Při stisku tlačítka myši je nastavena počáteční pozice. Při pohybu myši se nastavuje diference původní a aktuální pozice myši. Při puštění tlačítka je přičtena k pozici všech vybraných objektů. Diference může být samozřejmě i záporná. Tento způsob změny pozice je mnohem jednodušší, než prostě měnit přímo souřadnice objektů.

- struct _selected_object - Struktura, ze které je složen seznam vybraných objektů. Obsahuje id, typ a velikost objektu.

instance:

- int moving_objects_status - Je nastavena na 1pokud uživatel hýbe vybranými objekty.
- mg_point moving_objects_original_position - Původní pozice objektů, tedy ta ke které se vztahuje diference, pozice myši při stisku tlačítka.
- int moving_objects_x_diff, int moving_objects_y_diff; - Diference pohybu, tj. rozdíl mezi nynější a původní pozicí myši. Jedna pro souřadnici x, druhá pro y.
- mg_point select_rectangle_start - Začátek obdélníku, kterým uživatel vybírá objekty.
- mg_point select_rectangle_end - Konec obdélníku, kterým uživatel vybírá objekty (nelze zde říct který roh to je, protože zde můžou nastat různé situace v závislosti na kvadrantech).
- int select_rectangle_status - Vrací, zda uživatel právě vybírá objekty.
- petri_net *net - Ukazatel na hlavní objekt petri_net (viz. petri-core.h).
- clist < _selected_object * >*selected_objects - Seznam vybraných objektů.

metody:

- petri_select(petri_net * main_net) - Konstruktor.
- ~petri_select() - Destruktor.
- int add_object_to_list_of_selected(int, int) - Přidá objekt do seznamu. Parametry: id, typ.
- int list_of_selected_first() - Nastaví ukazatel v seznamu objektů na první a vrací zda jsou v seznamu nějaké položky.
- int list_of_selected_next() - Nastaví ukazatel v seznamu objektů na další.
- int list_of_selected_get_current_type() - Vrací typ objektu na který zrovna ukazuje ukazatel v seznamu objektů.
- int list_of_selected_get_current_id() - Vrací id objektu na který zrovna ukazuje ukazatel v seznamu objektů.
- mg_point list_of_selected_get_current_position() - Vrací opravdovou pozici objektu, na který zrovna ukazuje ukazatel v seznamu objektů. Vyžádá si informace přímo od objektů petriho sítě.
- int list_of_selected_unselect_all() - Zruší výběr objektů.
- int list_of_selected_select_all() - Vybere všechny objekty v dokumentu.
- int is_st_selected() - Vrací, zda je vybrán nějaký objekt.
- int items_selected(int) - Vrací kolek objektů typu parametr je vybráno.
- int places_selected() - Počet vybraných míst.
- int transitions_selected() - Počet vybraných přechodů.

- int notes_selected() - Počet vybraných poznámek.
- int edges_selected() - Počet vybraných hran.
- int move_selected_objects() - Přesune vybrané objekty na pozice plus diference.
- int is_id_selected(int, int) - Vrátí, zda je vybrán objekt definovaný parametry: id, typ.
- int is_in(int x, int y) - Zjistí jestli je pozice definovaná parametry uvnitř obdélníku, kterým uživatel vybírá objekty.
- int put_selected_into_list_of_selected() - Vloží do seznamu objekty, které se nacházejí uvnitř obdélníku, kterým uživatel vybírá objekty.
- void set_moving_objects_difference(int, int) - Nastavuje diferenci. Parametry: x, y - aktuální pozice myši.
- void delete_selected() - Smaže fyzicky vybrané objekty, potom je zruší i ze seznamu.
- int remove_from_list(int id, int type) - Zruší objekt pouze ze seznamu vybraných objektů.

3.3.11 petri-string.h

Tento hlavičkový soubor obsahuje několik funkcí pro práci se stringy v pojetí C, tedy ukončenými 0.

- char *copy(char *, int, int) - Vrátí zkopírovaný string, je pro něj naalokováno potřebné množství paměti. Parametry: zdrojový string, pozice odkud, pozice kam.
- char *ltrim(char *) - Ořeže mezery zleva.
- char *rtrim(char *) - Ořeže mezery zprava. Obě funkce vrací nově naalokovaný string.

3.3.12 petri-undo.h

Třída obhospodařující undo a redo seznamy. Každý jednotlivý stav objektu je pro jednoduchost ukládán kompletně. Hospodárnější by jistě bylo ukládat například jen proběhlé akce. To je ovšem velmi náročný způsob na implementaci a paměťové nároky zvoleného řešení jsou minimální použitím struktur, které naše reálné objekty pouze abstrahují, neukládají se tedy do paměti kompletní objekty jen jejich atributy.

struktury:

- struct _t - Struktura obsahující atributy přechodu.
- struct _p - Struktura obsahující atributy místa.
- struct _e - Struktura obsahující atributy hrany.
- struct _ n- Struktura obsahující atributy textové poznámky.

Třída undo_buffer Symbolizuje buffer, do kterého se ukládají stavy editovaného dokumentu.

instance:

- clist < _t * >*t - Seznam přechodů.
- clist < _p * >*p - Seznam míst.
- clist < _e * >*e - Seznam hran.
- clist < _n * >*n - Seznam poznámek.
- int place_max_id - Maximální id všech míst.
- int transition_max_id - Maximální id všech přechodů.
- int hrana_max_id - Maximální id všech hran.
- int note_max_id - Maximální id všech poznámek.

metody:

- int prepare_undo_structure(petri_net * pnet) - Uloží současný stav dokumentu do bufferu.
- petri_net *create_net_from_undo_structure() - Obnoví (zruší současný a vytvoří nový, předchozí) dokument z bufferu.

Třída undo_redo Tato třída umožňuje manipulaci s zásobníky undo bufferů a redo bufferů. Každý buffer obsahuje obraz dokumentu. Undo a redo fungují zjednodušeně takto: Při každé změně dokumentu se stav před ní uloží do undo bufferu. Při akci undo se nejprve do redo bufferu uloží aktuální stav a poté se z undo bufferu obnoví stav předchozí. Při redo akci se se uloží do undo bufferu stav aktuální a z redo bufferu se obnoví stav dokumentu. Je nutné si uvědomit, že buffery jsou uloženy v zásobníku, který umožňuje operace push pro uložení a pop pro vybrání.

instance:

- petri_net * net - Ukazatel na hlavní objekt.
- cstack <undo_buffer *> *undo_stack - Zásobník minulých stavů, undo.
- cstack <undo_buffer *> *redo_stack - Zásobník následujících stavů, redo.

metody:

- undo_redo(petri_net *) - Konstruktor, jako parametr je zde ukazatel na hlavní objekt.
- ~undo_redo() - destruktor
- petri_net *synchronize() - Vrací vnitřní ukazatel na hlavní objekt. Tato metoda je důležitá, protože při undo nebo redo akci se hlavní objekt zruší a dokument jej tedy náhle nezná. V metodě do_undo(), nebo do_redo() se však vytvoří znovu z patřičného zásobníku. Prostřednictvím této metody se dokumentu sdělí nový ukazatel na hlavní objekt. Dokument se po akci "synchronizuje".

- int register_change(petri_net*) - Uloží aktuální stav dokumentu do zásobníku undo. Je nutno zavolat tuto metodu **před** jakoukoliv změnou. Parametr: ukazatel na hlavní objekt.
- int do_undo() - Uloží aktuální stav do redo zásobníku, zruší hlavní objekt a operací pop ze zásobníku undo získá a vytvoří nový hlavní objekt.
- int can_undo() - Vrací, zda je možno provést undo.
- int can_redo() - Vrací, zda je možno provést redo.
- int do_redo() - Zaregistruje změnu pomocí register_change(), a vytvoří nový hlavní objekt ze zásobníku redo.

3.4 Ostatní

3.4.1 petrinet_draw.h

Tento hlavičkový soubor obsahuje sadu funkcí, které obstarávají veškeré kreslení. Netvoří žádnou třídu, bylo by to zbytečné. Každá funkce bere jako parametr, kromě jiných, také ukazatel na DC do kterého bude kreslit. Funkce jsou oddělené z důvodu portability, jelikož jsou závislé na platformě WINDOWS.

3.5 Rozhraní ActiveX

Program je napsán jako ActiveX komponenta, tj. je schopen poskytovat své funkce jiným programům či komponentám. Rozhraní `ITeXEditor`, které program realizuje, je společné pro všechny editory realizované jako bakalářský projekt na katedře informatiky Univerzity Palackého, a tudíž mohou být užity jako komponenty pro nějaký nadřízený editor TEXu/LATEXu. Rozhraní obsahuje následující metody (definice jsou v IDL jazyku):

- `HRESULT NewTeXObject([in] BSTR filename)` - vytváří nový dokument editoru.
- `HRESULT EditTeXObject([in] BSTR filename)` - otevře specifikovaný, již existující, dokument v editoru.
- `HRESULT GetInfo([out,retval] BSTR * text)` - vrací informace o autorovi, programu apod.
- `HRESULT GetTeXSettings([out,retval] BSTR * text)` - vrací TEX nastavení, která jsou potřeba, aby kód v TEXu/LATEXu, který komponenta generuje, byl korektní.
- `HRESULT GetTeXText([out,retval] BSTR * text)` - vrací kód v TEXu / LATEXu, který popisuje vzhled dokumentu editoru.

4 Závěr

V tomto projektu jsem se snažil vytvořit aplikaci, která by měla zpříjemnit vytváření a editaci Petriho sítí v typografickém prostředí LATEX, což se mi doufám podařilo. Aplikace potřebuje k běhu tyto soubory:

- Petri.exe - vlastní aplikace

- napoveda.htm - nápověda k aplikaci
- napoveda.exe - prohlížeč nápovědy
- ole_test.exe - soubor k testování ActiveX rozhraní, není potřebný přímo pro běh aplikace.

Reference

[1] Andrews, Mark: *Programujeme v jazyce Visual C++*. Computer Press, Praha, 1997.

[2] Virius, Miroslav: *Visual C++ 5.0*. Grada Publishing, Praha, 1997.