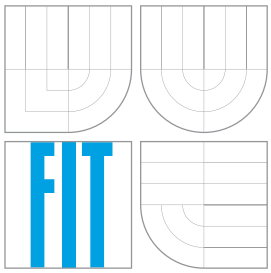
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

EDITOR ELEKTRONICKÝCH SCHÉMAT S ROZHRANÍM V QT

CIRCUIT DIAGRAM EDITOR WITH QT INTERFACE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

IVO ADAM

VEDOUCÍ PRÁCE
SUPERVISOR

Dr. Ing. PETR PERINGER

BRNO 2013

Abstrakt

Tato bakalářská práce se zabývá objektově orientovaným návrhem a implementací programu pro vytváření a upravování elektronických schémat. Program je implementován v jazyce C++ a jeho grafické uživatelské rozhraní je vytvořeno pomocí Qt. Všechny zdrojové kódy programu jsou volně dostupné. Výsledná aplikace je přenositelná, lze ji provozovat na operačních systémech Microsoft Windows XP/Vista/7/8 a na Linuxových distribucích.

Abstract

This bachelor thesis deals with object-oriented design and implementation of a computer application for creating and editing electronic circuit diagrams. The application is implemented in C++ and uses the Qt framework. All source codes are freely available. The application is portable, it can be run on operating systems Microsoft Windows XP/Vista/7/8 or Linux distributions.

Klíčová slova

editor, elektronické schéma, C++, objektově orientované programování, Qt

Keywords

editor, circuit diagram, C++, object-oriented programming, Qt

Citace

Ivo Adam: Editor elektronických schémat s rozhraním v Qt, bakalářská práce, Brno, FIT VUT v Brně, 2013

Editor elektronických schémat s rozhraním v Qt

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením pana Dr. Ing. Petra Peringera. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ivo Adam
13. května 2013

Poděkování

Děkuji Dr. Ing. Petru Peringerovi za vedení mé bakalářské práce a věcné připomínky při konzultacích.

Obsah

Kapitola 1

Úvod

Většina studentů technické školy elektrotechnického zaměření se během svého studia dostane do situace, kdy potřebuje na počítači nakreslit jednoduché elektronické schéma. Ostatně do stejné situace se může dostat i nějaký nadšenec do elektroniky, který by chtěl na svůj web umístit schéma zapojení zesilovače, nebo třeba automatického spínání světel. Nezkušený uživatel pro řešení této situace většinou použije některý grafický editor. Je zřejmé, že nakreslit schéma v takovém programu je nejen časově náročné, ale nakreslené schematické značky často neodpovídají standardu, což vede k nepěknému vzhledu výsledného schématu. Rovněž úprava takto nakresleného schématu může být časově náročná, především byl-li pro kreslení zvolen rastrový grafický editor. Mnohem lepším řešením je použít editor elektronických schémat, který byl přímo pro tuto problematiku navržen. Ovšem nalézt editor elektronických schémat, který by byl zdarma, snadno se ovládal a produkoval pěkně vypadající schémata, není snadný úkol. Z tohoto důvodu je cílem této práce vytvořit volně dostupný editor elektronických schémat. Jeho předností bude jednoduché a intuitivní ovládání a možnost exportu nakreslených schémat do vektorového formátu SVG, který je vhodný jak pro tisk, tak pro publikování na webu.

Text práce je rozdělen do pěti kapitol. Příští kapitola rozebírá problematiku editorů elektronických schémat, tvorby grafických uživatelských rozhraní a návrhových vzorů. Další kapitola se zabývá návrhem editoru. Popisuje formáty souborů používaných v editoru, návrh grafického uživatelského rozhraní a objektově orientovaný návrh založený na Qt *frameworku*.[1] Ve čtvrté kapitole je implementace a testování navrženého editoru. Jsou zde uvedeny minimální požadavky, značky obsažené v knihovně a postup při ověřování funkčnosti editoru. Poslední kapitola shrnuje dosažené výsledky a přínos celé práce.

[1] *Framework* lze zjednodušeně definovat jako ucelený soubor tematicky zaměřených knihoven. Český ekvivalent slova framework se nepoužívá, proto bude po zbytek práce používán anglický termín.

Kapitola 2

Přehled současného stavu

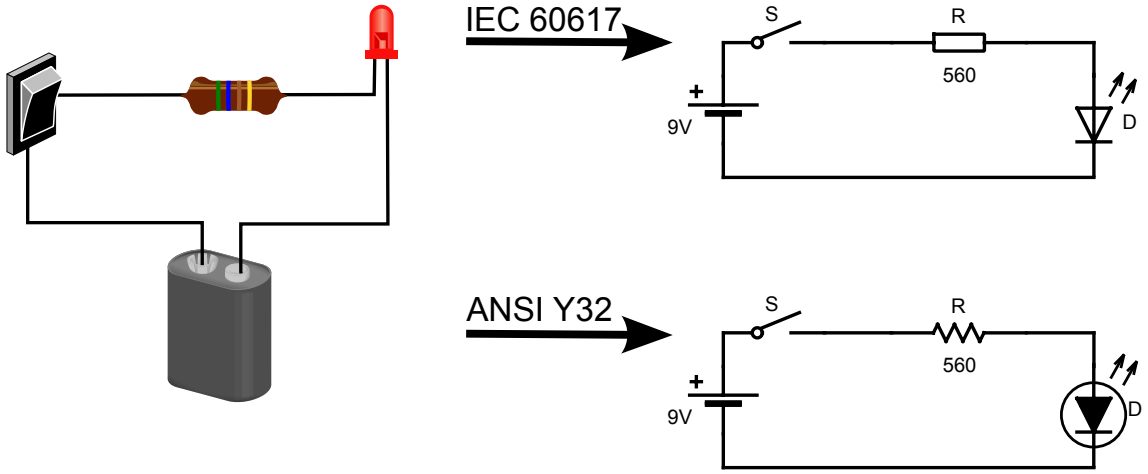
Tato kapitola se zabývá problematikou související s tvorbou programu pro vytváření a upravování elektronických schémat. Obsah kapitoly je rozdělen do tří částí. První popisuje problematiku editorů elektronických schémat. Ve druhé části je úvod do tvorby aplikací s grafickým uživatelským rozhraním, především pak pomocí Qt frameworku. Závěrečná část kapitoly obsahuje velmi stručný úvod do návrhových vzorů. Část o návrhových vzorech byla do kapitoly zahrnuta zejména proto, že některé z návrhových vzorů budou později použity v návrhu editoru.

2.1 Problematika editorů elektronických schémat

Pojem *editor* je z hlediska informatiky chápán jako počítačový program, který na základě jeho zaměření umožňuje vytvářet a upravovat specifický typ informace. *Editor elektronických schémat* je tedy software umožňující vytvářet a upravovat elektronická schémata. Pojmem elektronické schéma a problematikou s tímto pojmem spojenou se stručně zabývají následující odstavce.

Elektronický obvod [13] je vodivé propojení *elektronických součástek* [8]. Elektronická součástka je každé diskrétní zařízení nebo fyzická entita, která se v elektronickém systému používá k ovlivnění elektronů nebo jejich polí. Na základě propojení jednotlivých součástek plní obvod požadovanou funkci, například zesílení signálu. Pro připojení součástky do obvodu slouží její vývody, které má každá součástka minimálně dva.

Elektronické schéma je grafická reprezentace elektronického obvodu ve formě neorientovaného grafu. Jednotlivé součástky jsou zde zastoupeny standardizovanými symboly, které se nazývají *schematické značky*. Vodivé spoje jsou vyjádřeny čarami, které mezi sebou, až na výjimky, svírají vždy pravý úhel. Přestože je tvar schematických značek normalizován, můžeme nalézt totožný elektronický obvod reprezentovaný na pohled jinými schématy. Je to způsobeno tím, že norem definujících vzhled schematických značek existuje více a některé pak definují vzhled určitých značek jinak. Která norma je v daném státě platná záleží na jeho zeměpisné poloze. Například norma používaná v Evropě má název IEC 60617 [4], zatímco ve Spojených státech amerických se používá IEEE 315 [7] (také známá pod označením ANSI Y32), existují ovšem i další normy. Nejčastěji viditelný rozdíl evropské a severoamerické normy je pravděpodobně ve značce rezistoru, viz obrázek 2.1.



Obrázek 2.1: Rozdílná reprezentace elektronického obvodu v závislosti na použité normě

2.1.1 Požadavky na editor elektronických schémat

Každý editor elektronických schémat musí obsahovat kreslící plochu, na které bude uživatel vytvářet schéma. Další nezbytnou součástí editoru elektronických schémat je knihovna schematických značek, ze které bude uživatel vkládat připravené značky do jeho schémat. Knihovna značek by měla být snadno rozšiřitelná o další značky, aby nedocházelo k situaci, že schéma nelze dokončit kvůli chybějící značce. Mezi základní operace, které musí každý editor podporovat, patří:

- vytvořit nové schéma
- uložit schéma do souboru
- načíst schéma ze souboru
- vložit značku z knihovny do schématu
- propojit značky ve schématu pomocí čáry znázorňující vodič
- otočit či zrcadlově překlopit značku
- vložit text do schématu
- přesunout značku nebo text na jiné místo
- odstranit značku nebo text ze schématu
- exportovat schéma do některého běžně používaného grafického formátu

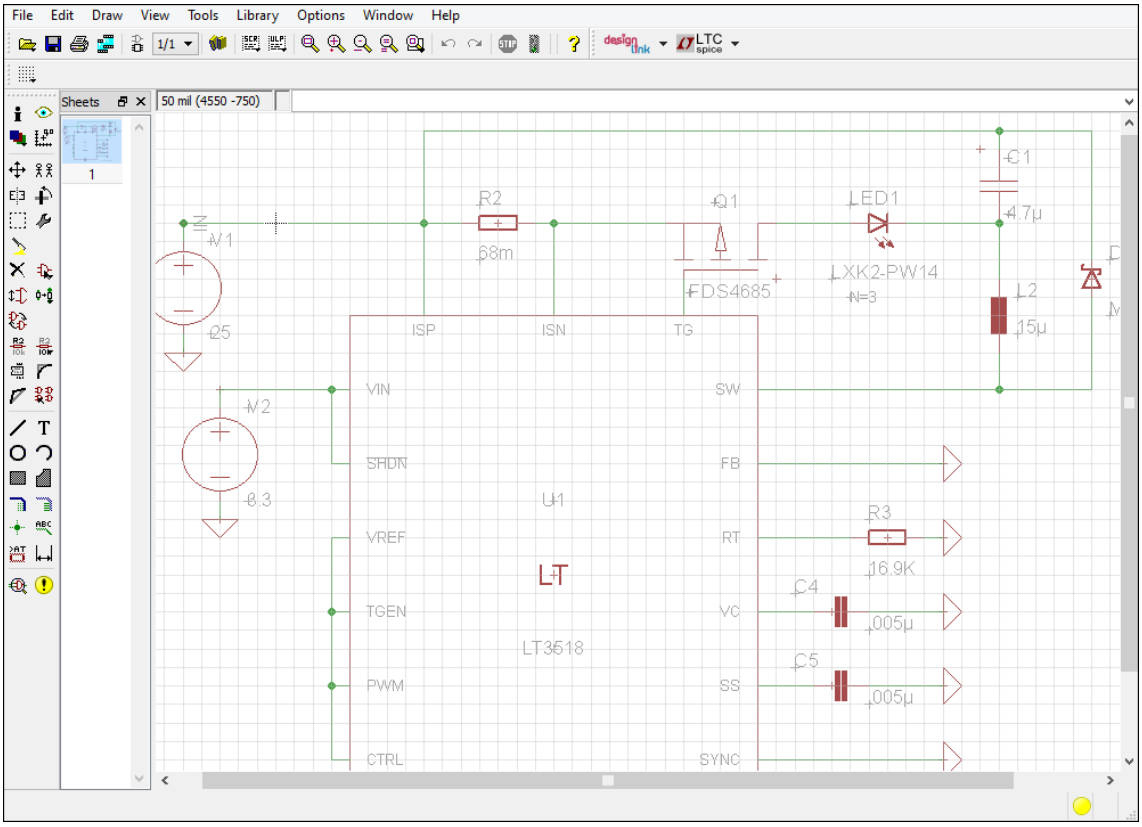
Většina editorů zvládá i další operace, které při kreslení schémat nejsou nezbytně nutné, ale v některých situacích mohou být užitečné. Mezi takové operace patří například zvětšování a zmenšování schématu, kopírování schématu nebo jeho části, možnost mít otevřených více schémat současně, vytváření hierarchických schémat a další.

2.1.2 Přehled existujících editorů elektronických schémat

Editory elektronických schémat se dají rozdělit z hlediska dostupnosti do dvou kategorií, na volně dostupné a komerční. Tato práce se zaměřuje na kategorii volně dostupných editorů. Mezi takové patří například **QUCS**, **XCircuit**, **gSchem**, **KiCad**, **Oregano**, **Fritzing** a **Dia**. Mezi komerční nástroje pro profesionální řešení patří například editor **EAGLE**, který bude dále použit jako reprezentant této kategorie. Základní informace o vybraných editorech elektronických schémat jsou uvedeny v následujících odstavcích.

EAGLE - Easily Applicable Graphical Layout Editor

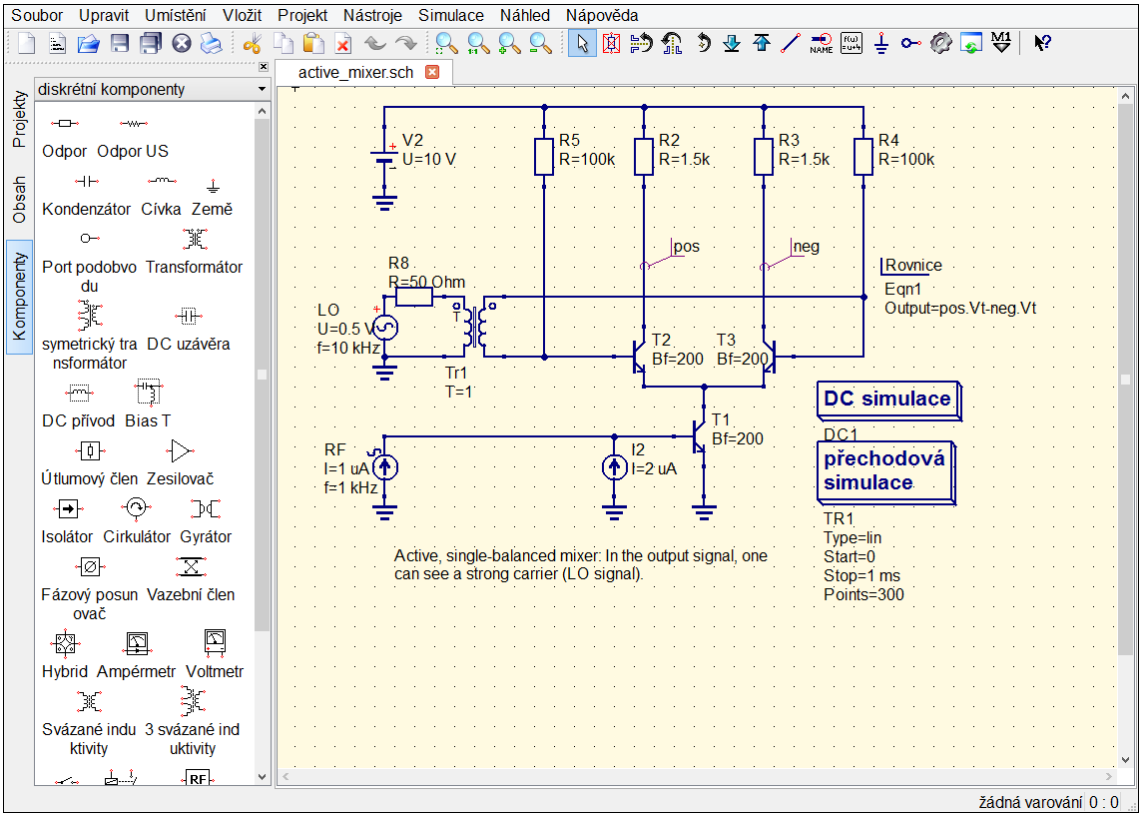
Tento profesionální nástroj [1] pro vytváření elektronických schémat a návrh plošných spojů se vyvíjí od roku 1988 až dodnes. S editorem EAGLE lze pracovat pod operačními systémy Microsoft Windows, Linux a Mac OS X. EAGLE lze pořídit v edici Light, Hobbyist, Standard a Professional. Ty se liší svými možnostmi a cenou. Edice Light je pro nekomerční použití dostupná zdarma, pro komerční použití je nutné zaplatit za jednouživatelskou licenci přibližně 1 600,- Kč. Edice Professional, která zahrnuje veškerou funkcionalitu, stojí pro jednoho uživatele zhruba 35 500,- Kč. Výhodou komerčních editorů pro profesionální řešení jsou jejich obrovské možnosti, díky kterým je možné nakreslit libovolně složité schéma. Rovněž bývá u takovýchto nástrojů dostupná technická podpora, jenž pomůže s případným řešením každého problému. Nevýhodou je pak kromě ceny i netriviální ovládání, uživatel je v mnoha případech nucen absolvovat školení, aby byl schopen využít pokročilejší možnosti editoru. Obrazovka se spuštěným editorem EAGLE Light je zachycena na obrázku 2.2.



Obrázek 2.2: Uživatelské rozhraní editoru EAGLE Light

QUCS - Quite Universal Circuit Simulator

QUCS [6] vzniknul ve své první verzi v roce 2003 a vyvíjí se dodnes. Je naprogramován v jazyce C++ a využívá framework Qt 4. Lze jej provozovat na všech běžně používaných operačních systémech.[1] QUCS umožňuje schémata nejen sestavovat a upravovat, ale rovněž simulovat jejich činnost, což je jistě v mnoha případech výhodné. Nastane-li ale situace, že schéma je potřeba pouze nakreslit a vytisknout, může být vestavěný simulátor spíše na obtíž, protože program obsahuje některé ovládací prvky a nastavení, které jsou potřebné pouze pro simulaci. V programu je obsaženo více než 150 schematických značek, ale přesto zde chybí některé základní, jako je LED (*Light-emitting diode*), žárovka nebo trimr. Rozdělení značek do skupin se v QUCS jeví chaoticky, rovněž značky v rámci skupiny nejsou uspořádány přehledně, jak se lze přesvědčit na obrázku 2.3.



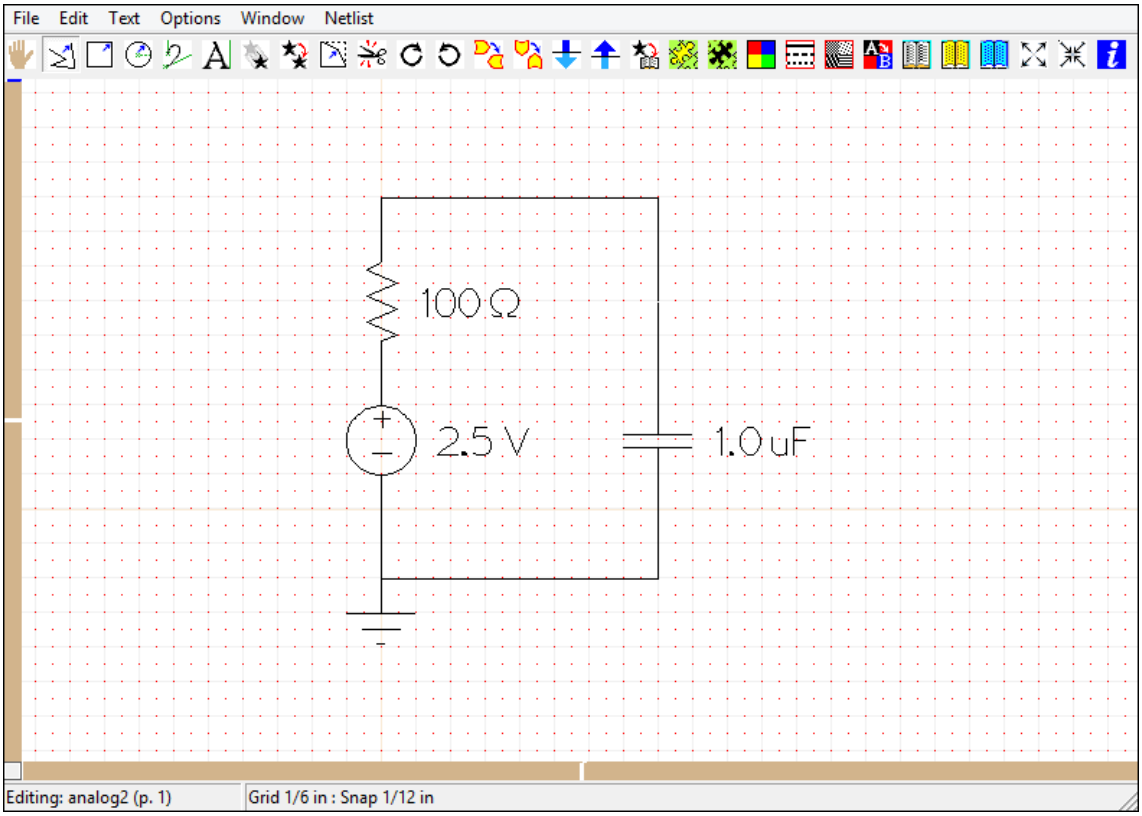
Obrázek 2.3: Uživatelské rozhraní editoru QUCS

XCircuit

Tento editor [10] je nejstarší ze zde uvedených nekomerčních nástrojů, jeho vývoj započal již v roce 1993 a přetrvává až do dnešní doby. Je implementován v programovacím jazyce C a stejně jako QUCS je přenositelný mezi všemi běžnými operačními systémy. XCircuit je velmi schopný nástroj, který ovšem není pro každého, a to z důvodu složitosti a neintuitivnosti jeho ovládání. Jako jeden z nedostatků přívětivosti ovládání lze zmínit, že editor obsahuje v horní části hlavní nástrojovou lištu s mnoha tlačítky (jak je možno vidět na

[1] Microsoft Windows, Linuxové distribuce, Mac OS X a FreeBSD.

obrázku 2.4), které nemají žádné popisky (ani po najetí myší). Uživatel může pouze na základě symbolu na ikoně tlačítka odhadovat, co se pod ním ukrývá za funkci. Absence popisků tlačítek není jediný nedostatek uživatelského rozhraní, program se celkově ovládá nestandardním způsobem a nakreslit triviální schéma může být pro mnoho uživatelů velmi komplikované. Problém s ovládáním lze vyřešit nastudováním uživatelské příručky k programu, ovšem její základní část má 62 stran (formátu A4), což může uživatele odradit.



Obrázek 2.4: Uživatelské rozhraní editoru XCircuit

gSchem

Vývoj editoru gSchem [2] byl zahájen v roce 1998 a stejně jako u předchozích programů pokračuje i v dnešní době. Je napsán v jazyce C a používá framework GTK+. gSchem je součástí velkého softwarového balíku gEDA, který obsahuje nástroje pokrývající kompletní problematiku návrhu elektroniky, včetně návrhu plošných spojů a simulací. Tento software není oficiálně přenositelný mezi operačními systémy. Podporuje pouze tzv. *Unix-like* systémy. Pro OS Windows ovšem existují neoficiální vydání. Ty mají sice nějaká omezení, ale pro kreslení základních schémat jsou vyhovující.

2.2 Tvorba aplikací s grafickým uživatelským rozhraním

Grafické uživatelské rozhraní (*GUI* z anglického *Graphical User Interface*) umožňuje komunikovat s počítačem prostřednictvím grafických ovládacích prvků. Grafický ovládací prvek, který se anglicky označuje *widget*, může být například tlačítko, menu, posuvník, formulář, dialog, textové pole, okno a podobně. Protože se v různých aplikacích používají ty stejné ovládací prvky, vzniky různé knihovny a frameworky, které poskytují podporu pro jejich vytváření, a tak výrazně zjednodušují tvorbu aplikací s GUI. Mezi běžně používané frameworky patří například WxWidgets, GTK+ nebo Qt. Program vytvořený v rámci této práce používá Qt.

Qt framework

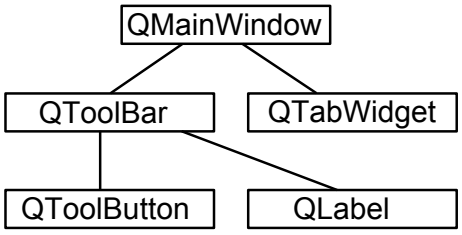
Qt framework[2] [9] je sada podpůrného software pro vytváření multiplatformních počítačových a mobilních aplikací. Jeho vývoj začal v roce 1995 a trvá dodnes, kdy je aktuální verze 5.0.2 [5]. Aplikace používající Qt framework je možné distribuovat pod licencí GNU GPL, GNU LGPL, nebo po splnění určitých kritérií i komerčně. Framework je objektově orientovaný a pro jeho implementaci byl použit programovací jazyk C++, který ale pomocí maker a speciálního generátoru kódu[3] rozšiřuje. Mezi rozšíření patří například mechanismus signálů a slotů, automatická správa paměti, nebo cyklus `foreach()` známý z jiných programovacích jazyků.

Obrázek 2.5: Logo Qt

Signály a sloty slouží ke komunikaci mezi objekty. Aby bylo možné v dané třídě signály a sloty používat, musí být třída potomkem třídy `QObject` a na začátku její definice musí být uvedeno makro `Q_OBJECT`. Zjednodušeně se dá tento mechanismus vysvětlit tak, že signály jsou vysílány na základě vzniku nějaké události, zatímco sloty jsou obslužné podprogramy, které se provedou jako reakce na vyslaný signál. Typicky se signály a sloty používají v grafickém uživatelském rozhraní, například tlačítko (`QPushButton`) vyšle signál `clicked()` vždy, když je stisknuto (toto chování je již zabudované v Qt). Samotné vyslání signálu ovšem nevyvolá žádnou odezvu. Má-li program na vyslaný signál reagovat, musí se signál nejprve připojit k nějakému slotu. Slot je zde chápán jako libovolná metoda, která se v definici třídy nachází v bloku uvedeném klíčovým slovem `slots`. Pro propojení signálu a slotu se používá metoda `QObject::connect()`.

Automatická správa paměti není tak zásadní rozšíření jazyka C++, jako signály a sloty, může ovšem předejít chybám vedoucím k úniku paměti. V objektově orientovaných aplikacích slouží ke korektnímu uvolnění paměti objektu jeho speciální metoda nazývaná destruktor. Používáme-li Qt, nejsou destruktory v některých případech potřeba. Při vytváření instance Qt třídy můžeme jako parametr konstruktoru předat ukazatel na nadřazený objekt. Qt tak vytváří hierarchii objektů (viz obrázek 2.6), které jsou mezi sebou ve vztahu rodič-potomek (*parent-child relationship*). Zanikne-li libovolný objekt, Qt zajistí řádné uvolnění paměti všech jeho potomků. Typickým příkladem je hlavní okno programu, které obsahuje desítky ovládacích prvků (tlačítka, nástrojové lišty, záložky, textová pole, ...). Dojde-li k odstranění hlavního okna, Qt automaticky odstraní i všechny ovládací prvky v něm obsažené.

[2] Někdy označován jako *Qt widget toolkit*.
[3] Nazývaného *Meta Object Compiler* nebo *moc*.



Obrázek 2.6: Příklad hierarchie objektů

Při tvorbě aplikací s grafickým uživatelským rozhraním lze z Qt využít desítky tříd, které umožňují snadné vytvoření standardních ovládacích prvků. Například pro vytvoření hlavního okna aplikace je připravena třída `QMainWindow`, pro tlačítko třída `QToolButton`, pro nástrojovou lištu třída `QToolBar` a tak dále. Kromě grafických uživatelských prvků je v Qt zabudována podpora pro přístup k databázím, zpracování XML dokumentů, počítačové sítě, 2D/3D grafiku a další. Některé třídy obsažené v Qt frameworku spolu mohou tvořit nějaký logický celek řešící určitou problematiku, takový celek pak může být sám označován jako framework. Jedním z takových frameworků obsažených v Qt je *Graphics View Framework*.

Graphics View Framework [3] poskytuje podporu pro vytváření dvojrozměrných grafických scén, které mohou obsahovat velký počet uživatelsky definovaných grafických prvků. Framework vychází z architektury MVC (*Model-View-Controller*), která odděluje datový model, aplikační logiku a prezentační logiku do tří nezávislých celků. Důvod tohoto rozdělení je především kvůli vytvoření přehlednějšího a snáze udržovatelného zdrojového kódu. Rozdělení na 3 nezávislé celky není v případě Graphics View Frameworku úplně přesné. Třída `QGrpahicsView`, která zajišťuje vizualizaci datového modelu, také zpracovává události myši, jimiž dále řídí datový model a tím pádem na sebe bere i roli dispečera (*controller*). Dalo by se tedy mluvit spíše o architektuře *Model-View*. Datový model je reprezentován třídou `QGraphicsScene`. Ta slouží jako kontejner pro jednotlivé prvky scény. Základní třídou pro prvek scény je `QGraphicsItem`, od této obecné třídy jsou dále odvozeny třídy pro některé běžně používané grafické prvky. Například pro vkládání textu je ve frameworku obsažena třída `QGraphicsTextItem`, pro vkládání čar `QGraphicsLineItem` a tak dále. Ve zmíněných třídách je mimo jiné zabudována podpora pro tyto operace:

- **Detekce kolizí** – Voláním metody `QGraphicsItem::collidingItems()` získá libovolný prvek scény seznam všech prvků, se kterými je aktuálně v kolizi.
- **Události myši** – Metoda `QGraphicsScene::mousePressEvent()` je automaticky zavolána vždy, když uživatel stiskne tlačítko myši nad přidruženou instancí třídy `QGraphicsView`. Obdobně je zpracováván pohyb myši (`mouseMoveEvent()`) a uvolnění tlačítka myši (`mouseReleaseEvent()`).
- **Pohyb prvků ve scéně pomocí mechanismu táhni a pusť (*drag & drop*)** – Volání metody `QGraphicsView::setDragMode()` s parametrem `RubberBandDrag` umožní pohyb prvků pomocí mechanismu *drag & drop* zcela automaticky.

2.3 Návrhové vzory

Návrhový vzor [11] je obecný popis typického řešení nějakého často se vyskytujícího problému při objektově orientovaném návrhu. Návrhové vzory popisují především vztahy a komunikaci mezi třídami a objekty. Vzory lze klasifikovat podle účelu na tvořivé, strukturální a na vzory chování.

Tvořivé vzory (*Creational Patterns*) řeší vytváření objektů v systému. Do této kategorie patří tyto návrhové vzory: Abstraktní továrna (*Abstract Factory*), Jedináček (*Singleton*), Prototyp (*Prototype*), Stavitel (*Builder*) a Tovární metoda (*Factory Method*).

Strukturální vzory (*Structural Patterns*) se zaměřují na uspořádání tříd a objektů v systému. Mezi strukturální vzory patří: Adaptér (*Adapter*), Dekorátor (*Decorator*), Fasáda (*Facade*), Most (*Bridge*), Muší váha (*Flyweight*), Skladba (*Composite*) a Zástupce (*Proxy*).

Vzory chování (*Behavioral Patterns*) charakterizují způsoby, podle nichž třídy či objekty vzájemně jednají a rozdělují si povinnosti. Patří mezi ně: Iterátor (*Iterator*), Interpret (*Interpreter*), Návštěvník (*Visitor*), Obnovitel (*Memento*), Pozorovatel (*Observer*), Prostředník (*Mediator*), Příkaz (*Command*), Řetěz odpovědnosti (*Chain of responsibility*), Stav (*State*), Strategie (*Strategy*) a Šablonová metoda (*Template Method*).

Stručný popis vybraných návrhových vzorů je uveden v následujících odstavcích. Podrobnosti a popis ostatních návrhových vzorů lze nalézt v [11].

Jedináček

Návrhový vzor Jedináček zajišťuje existenci maximálně jedné instance dané třídy a poskytuje k ní globální přístup. Jedna z možností jak implementovat návrhový vzor Jedináček je definovat soukromý konstruktor a pro vytvoření a přístup k instanci používat statickou metodu vracející referenci na instanci. Tato možná implementace je uvedena v následující ukázce kódu.

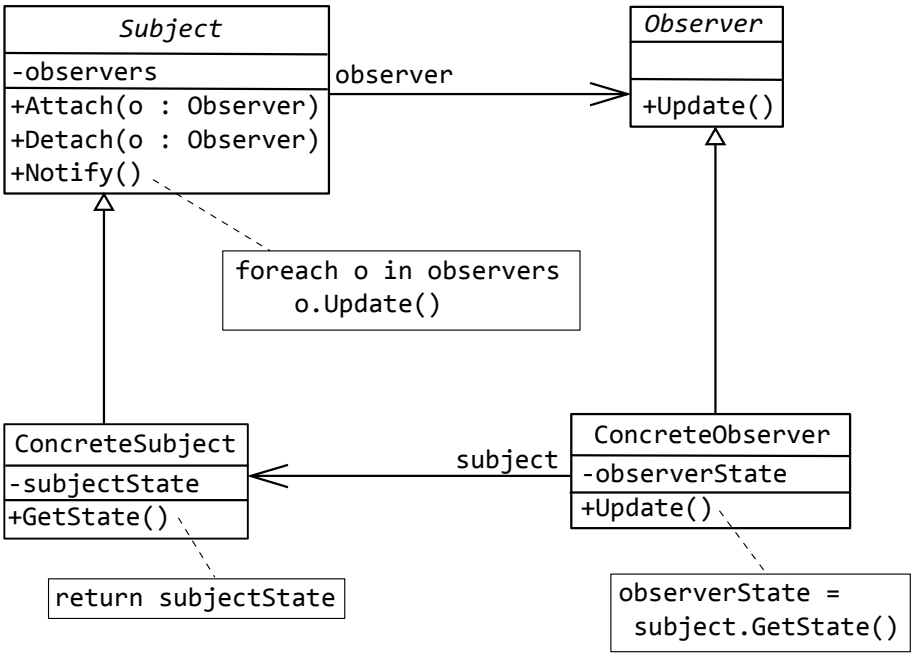
```
class Singleton {
public:
    static Singleton& getInstance() {
        static Singleton instance;
        return instance;
    }
private:
    Singleton();
    Singleton(Singleton const&);
    void operator=(Singleton const&);
}
```

Ukázka kódu 2.1: Implementace vzoru Jedináček v C++

Pozorovatel

Tento návrhový vzor umožňuje k nějakému objektu připojit libovolné množství jiných objektů, které ho pozorují a reagují na změnu jako stavu. Objekt, který je pozorován, se označuje jako *Subject* a objekt jenž pozoruje se nazývá *Observer*. *Observer* se pomocí veřejného rozhraní objektu *Subject* přihlásí k odběru nějaké jeho události. Každý *Subject* zná

všechny objekty *Observer*, které ho pozorují. Dojde-li ke změně v objektu *Subject*, tak o ní automaticky informuje všechny závislé objekty *Observer* pomocí volání jejich veřejné metody. Strukturu tohoto návrhového vzoru znázorňuje obrázek 2.7. Vzor Pozorovatel nachází uplatnění především v systémech založených na zpracování událostí.



Obrázek 2.7: Struktura návrhového vzoru Pozorovatel

Iterátor

Iterátor se používá k vytvoření jednotného rozhraní, které dovoluje sekvenční průchod přes objekty uložené v nějakém kontejneru, aniž bychom znali jeho interní reprezentaci. Iterátor poskytuje záruku, že při průchodu libovolně složitým kontejnerem bude každý prvek zpřístupněn právě jednou. Nemůže tak dojít k situaci, že by byl nějaký prvek opomenut, nebo naopak zpracován vícekrát. Typické použití iterátoru je například v implementaci cyklu `foreach()`.

Kapitola 3

Návrh editoru elektronických schémat

V návrhu editoru, především pak v jeho ovládání, jsou použity poznatky získané při zkoušení již existujících editorů elektronických schémat. Byla snaha zahrnout do návrhu dobrá a fungující řešení z konkurenčních editorů.

3.1 Použité souborové formáty

Navržený editor dokáže pracovat se čtyřmi souborovými formáty. Dva z nich jsou standardní grafické formáty, do nichž je umožněn export nakreslených schémat. Konkrétně se jedná o vektorový formát SVG [12] (*Scalable Vector Graphics*) a rastrový formát PNG [15] (*Portable Network Graphics*). Zbylé dva formáty jsou nestandardní a byly navrženy speciálně pro potřeby editoru. Jeden z nich je použit pro knihovnu schematických značek a druhý pro ukládání schémat při operaci *uložit*. Oba formáty byly navrženy pomocí značkovacího jazyka XML [14] (*eXtensible Markup Language*), který umožňuje definovat aplikačně specifický značkovací jazyk. Detailní popis navržených formátů je uveden v následujících dvou podkapitolách.

3.1.1 Formát knihovny schematických značek

Jedním ze základních požadavků na editor elektronických schémat (viz kapitola 2.1.1) je přítomnost knihovny schematických značek. Knihovnu značek lze do programu zahrnout minimálně dvěma způsoby. První možnost je definovat značky přímo ve zdrojovém kódu programu. Toto řešení má ovšem velkou nevýhodu v tom, že při každé úpravě knihovny schematických značek je nutné přeložit program znovu. Druhý způsob, který byl použit v tomto editoru, je definovat všechny potřebné informace o značkách v externím souboru, jehož obsah je aplikací načten při spuštění. Nevýhoda provázanosti zdrojového kódu a knihovny značek zde odpadá, a proto lze značky snadno odebírat, přidávat či upravovat.

Základní informace o schematické značce, které je nutné v knihovně uchovávat, jsou jméno, tvar, standardní označení[1], rozměry a fyzikální jednotka. Všechny uvedené informace, až na tvar, lze snadno zapsat ve formě řetězce znaků. Tvar je zapisován jako seznam základních geometrických tvarů, které dohromady tvoří symbol schematické značky. Poslední informace, kterou je nutné uchovávat, je umístění přípojných bodů značky. Ty je

[1] Například R pro rezistor, C pro kondenzátor, atp.

nutné znát především kvůli automatické kontrole správnosti zapojení obvodu. Pro snazší orientaci ve schematických značkách je vhodné definovat v knihovně určité kategorie a značky do nich roztřídit. V tomto editoru jsou značky rozřazeny do kategorií na základě „příbuznosti“, například kategorie *Diody* obsahuje standardní diodu, LED, Zenerovu diodu a tak dále.

Popis struktury navrženého XML dokumentu je uveden v podobě DTD (*Document Type Definition*) v následující ukázce kódu.

```
<!ELEMENT library   (category+)>
<!ELEMENT category  (component+)>
<!ELEMENT component (paint, pins)>
<!ELEMENT paint     ((line | rectangle | arc | ellipse | parent)+)>
<!ELEMENT pins      (pin+)>
<!ELEMENT line      EMPTY>
<!ELEMENT rectangle EMPTY>
<!ELEMENT arc       EMPTY>
<!ELEMENT ellipse   EMPTY>
<!ELEMENT parent    EMPTY>
<!ELEMENT pin       EMPTY>

<!ATTLIST category  name    CDATA #REQUIRED>
<!ATTLIST category  name_cs CDATA #IMPLIED>
<!ATTLIST component name    CDATA #REQUIRED>
<!ATTLIST component name_cs CDATA #IMPLIED>
<!ATTLIST component label   CDATA #IMPLIED>
<!ATTLIST component width   CDATA #REQUIRED>
<!ATTLIST component height  CDATA #REQUIRED>
<!ATTLIST component unit    CDATA #IMPLIED>
<!ATTLIST line      x1      CDATA #REQUIRED>
<!ATTLIST line      y1      CDATA #REQUIRED>
<!ATTLIST line      x2      CDATA #REQUIRED>
<!ATTLIST line      y2      CDATA #REQUIRED>
<!ATTLIST rectangle x       CDATA #REQUIRED>
<!ATTLIST rectangle y       CDATA #REQUIRED>
<!ATTLIST rectangle w       CDATA #REQUIRED>
<!ATTLIST rectangle h       CDATA #REQUIRED>
<!ATTLIST arc       x       CDATA #REQUIRED>
<!ATTLIST arc       y       CDATA #REQUIRED>
<!ATTLIST arc       w       CDATA #REQUIRED>
<!ATTLIST arc       h       CDATA #REQUIRED>
<!ATTLIST ellipse   x       CDATA #REQUIRED>
<!ATTLIST ellipse   y       CDATA #REQUIRED>
<!ATTLIST ellipse   w       CDATA #REQUIRED>
<!ATTLIST ellipse   h       CDATA #REQUIRED>
<!ATTLIST parent    name    CDATA #REQUIRED>
<!ATTLIST parent    x       CDATA #REQUIRED>
<!ATTLIST parent    y       CDATA #REQUIRED>
```

Ukázka kódu 3.1: DTD knihovny schematických značek

Jak je vidět v uvedeném DTD dokumentu, lze pro specifikaci tvaru schematické značky používat čáru (`line`), obdélník(`rectangle`), oblouk(`arc`), elipsu(`ellipse`) a odkaz na již definovanou značku, který je označený `parent`. Pro definici čáry je nutné uvést její počáteční (x_1, y_1) a koncový (x_2, y_2) bod. U obdélníku, oblouku a elipsy se uvádí levý horní roh opsaného obdélníku (x, y), šířka `w` a výška `h`. Nejzajímavějším prvkem při definování tvaru je odkaz na již definovanou značku, což vnáší do knihovny koncept dědičnosti a usnadňuje tak specifikaci příbuzných značek. Pro korektní odkaz na existující značku je nutné uvést

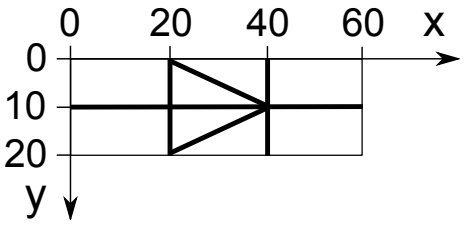
její jméno (`name`) a relativní posun v ose `x` a `y`. Všechny grafické prvky musejí být vždy vykresleny do pomyslného obdélníku, který definují atributy `width` a `height` v elementu `component`. Pro specifikaci přípojného bodu značky se používá element `pin`, který má atributy `x` a `y` jenž definují relativní polohu bodu v rámci značky. Při definici přípojných bodů je důležité zajistit, aby všechny přípojné body v rámci značky měli mezi sebou v obou osách rozdíl roven násobku hodnoty 20. Jestliže se toto pravidlo nedodrží, tak nebude možné zapojit všechny přípojné body, protože některé budou ležet mimo rastr nakreslený na pozadí.

Pro snadnější představu nyní následuje ukázka jednoduché knihovny o dvou schematických značkách, na kterých bude názorně vysvětlen princip kreslení a souřadnicový systém.

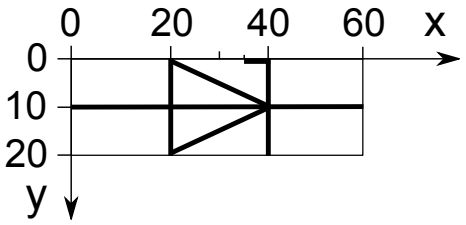
```
<?xml version="1.0" encoding="UTF-8" ?>
<library>
  <category name="Diodes">
    <component name="Diode" label="D" width="60" height="20">
      <paint>
        <line x1="0"  y1="10" x2="60" y2="10"/>
        <line x1="40" y1="0"  x2="40" y2="20"/>
        <line x1="20" y1="0"  x2="20" y2="20"/>
        <line x1="20" y1="0"  x2="40" y2="10"/>
        <line x1="20" y1="20" x2="40" y2="10"/>
      </paint>
      <pins>
        <pin x="0"  y="10"/>
        <pin x="60" y="10"/>
      </pins>
    </component>
    <component name="Zener␣diode" label="D" width="60" height="20">
      <paint>
        <parent name="Diode" x="0" y="0"/>
        <line x1="35" y1="0"  x2="40" y2="0"/>
      </paint>
      <pins>
        <pin x="0"  y="10"/>
        <pin x="60" y="10"/>
      </pins>
    </component>
  </category>
</library>
```

Ukázka kódu 3.2: Knihovna schematických značek, která obsahuje značku obyčejné diody a Zenerovy diody

Uvedená knihovna obsahuje standardní diodu o rozměrech 60x20 obrazových bodů. Označení diody je Dx, kde x značí pořadové číslo diody v rámci schématu. Dioda sestává z pěti čar a přípojné body má na souřadnicích $(0, 10)$ a $(60, 10)$, jak je vidět na obrázku 3.1. Druhým prvkem knihovny je Zenerova dioda, která je zobrazena na obrázku 3.2. Ta v rámci definice tvaru využívá odkaz na standardní diodu, jejíž tvar rozšiřuje o čáru vedoucí z bodu $(35, 0)$ do bodu $(40, 0)$.



Obrázek 3.1: Značka obyčejné diody



Obrázek 3.2: Značka Zenerovy diody

3.1.2 Formát uložených schémat

Formát pro uložení schématu byl pojmenován podle editoru, tedy CircuitDiagramEditor (*.cde*). Formát byl navržen především proto, že se nepodařilo najít žádný již existující, který by byl dostatečně jednoduchý a přitom pokrýval všechny požadavky editoru. Popis struktury v podobě DTD je uveden v následující ukázce kódu.

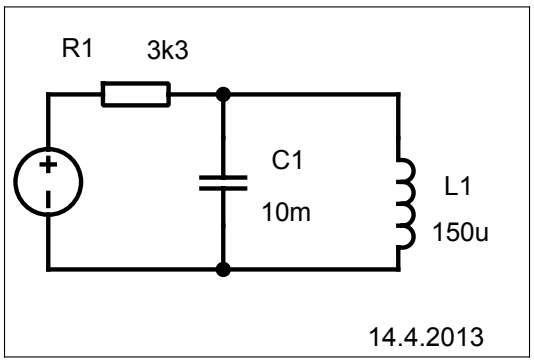
```
<!ELEMENT circuit   ((text | wire | line | component)*)>
<!ELEMENT component (text, text)>
<!ELEMENT text      EMPTY>
<!ELEMENT wire      EMPTY>
<!ELEMENT line      EMPTY>

<!ATTLIST circuit    width     CDATA #REQUIRED>
<!ATTLIST circuit    height    CDATA #REQUIRED>
<!ATTLIST text       x         CDATA #REQUIRED>
<!ATTLIST text       y         CDATA #REQUIRED>
<!ATTLIST text       content   CDATA #REQUIRED>
<!ATTLIST text       type      CDATA #REQUIRED>
<!ATTLIST wire       x1        CDATA #REQUIRED>
<!ATTLIST wire       y1        CDATA #REQUIRED>
<!ATTLIST wire       x2        CDATA #REQUIRED>
<!ATTLIST wire       y2        CDATA #REQUIRED>
<!ATTLIST line       x1        CDATA #REQUIRED>
<!ATTLIST line       y1        CDATA #REQUIRED>
<!ATTLIST line       x2        CDATA #REQUIRED>
<!ATTLIST line       y2        CDATA #REQUIRED>
<!ATTLIST component  type      CDATA #REQUIRED>
<!ATTLIST component  x         CDATA #REQUIRED>
<!ATTLIST component  y         CDATA #REQUIRED>
<!ATTLIST component  rotation  CDATA #REQUIRED>
<!ATTLIST component  mirrorH   CDATA #REQUIRED>
<!ATTLIST component  mirrorV   CDATA #REQUIRED>
```

Ukázka kódu 3.3: DTD formátu *.cde*

Z uvedeného DTD vyplývá, že každé schéma (`circuit`) má nějakou šířku (`width`) a výšku (`height`). Schéma se skládá z textů (`text`), vodičů (`wire`), čár (`line`) a schematických značek (`component`). O každém textu se ukládají jeho souřadnice (x, y), obsah (`content`) a typ (`type`). Typ značí, jedná-li se o text patřící ke schematické značce či nikoliv. U vodiče a čáry se uchovává počáteční (x_1, y_1) a koncový (x_2, y_2) bod. Každý vodič i čára se vždy ukládá jako úsečka. Je-li například vodič ve schématu lomená čára, bude uložen jako více úseček. Rozdíl mezi vodičem a čárou je ten, že vodič se používá k propojení značek, zatímco čára ke kreslení rámečků, razítek a podobně. U schematické značky se uchovává její pozice (x, y), typ (`type`) a transformace. Transformace je rozdělená do 3 atributů, `rotation` značí

natočení a musí nabývat jedné z hodnot $(0, 90, 180, 270)$, `mirrorH` a `mirrorV` značí horizontální a vertikální překlopení značky, hodnota atributu je buď ano (`yes`) nebo ne (`no`). Každý element schematické značky v sobě musí obsahovat dva elementy textu, které reprezentují její hodnotu (*type="value"*) a označení (*type="label"*). Pro snadnější pochopení formátu bude nyní uveden příklad schématu a jeho reprezentace ve formátu *.cde*.



Obrázek 3.3: Příklad jednoduchého schématu

```
<?xml version="1.0" encoding="UTF-8"?>
<circuit width="2000" height="1000">
  <text x="162"  y="135"  content="14.4.2013" type="normal"/>
  <line x1="240" y1="160" x2="0"   y2="160"/>
  <line x1="0"   y1="0"   x2="0"   y2="160"/>
  <line x1="240" y1="0"   x2="240" y2="160"/>
  <line x1="0"   y1="0"   x2="240" y2="0"/>
  <wire x1="100" y1="120" x2="20"  y2="120"/>
  <wire x1="100" y1="100" x2="100" y2="120"/>
  <wire x1="180" y1="120" x2="100" y2="120"/>
  <wire x1="180" y1="40"  x2="180" y2="60"/>
  <wire x1="100" y1="40"  x2="180" y2="40"/>
  <wire x1="100" y1="40"  x2="100" y2="60"/>
  <component type="Inductor" x="187" y="60" rotation="90" mirrorH="no"
      mirrorV="no">
    <text x="197" y="66" content="L1" type="label"/>
    <text x="191" y="87" content="150u" type="value"/>
  </component>
  <component type="Capacitor" x="110" y="60" rotation="90" mirrorH="no"
      mirrorV="no">
    <text x="118" y="54" content="C1" type="label"/>
    <text x="113" y="78" content="10m" type="value"/>
  </component>
  <component type="Resistor_EU" x="20" y="35" rotation="0" mirrorH="no"
      mirrorV="no">
    <text x="22" y="3" content="R1" type="label"/>
    <text x="61" y="4" content="3k3" type="value"/>
  </component>
  <component type="Voltage_source" x="5" y="120" rotation="270" mirrorH="no"
      mirrorV="no">
    <text x="20" y="37" content="" type="label"/>
    <text x="20" y="95" content="" type="value"/>
  </component>
</circuit>
```

Ukázka kódu 3.4: Reprezentace schématu uvedeného na obrázku 3.3 ve formátu *.cde*

3.2 Grafické uživatelské rozhraní

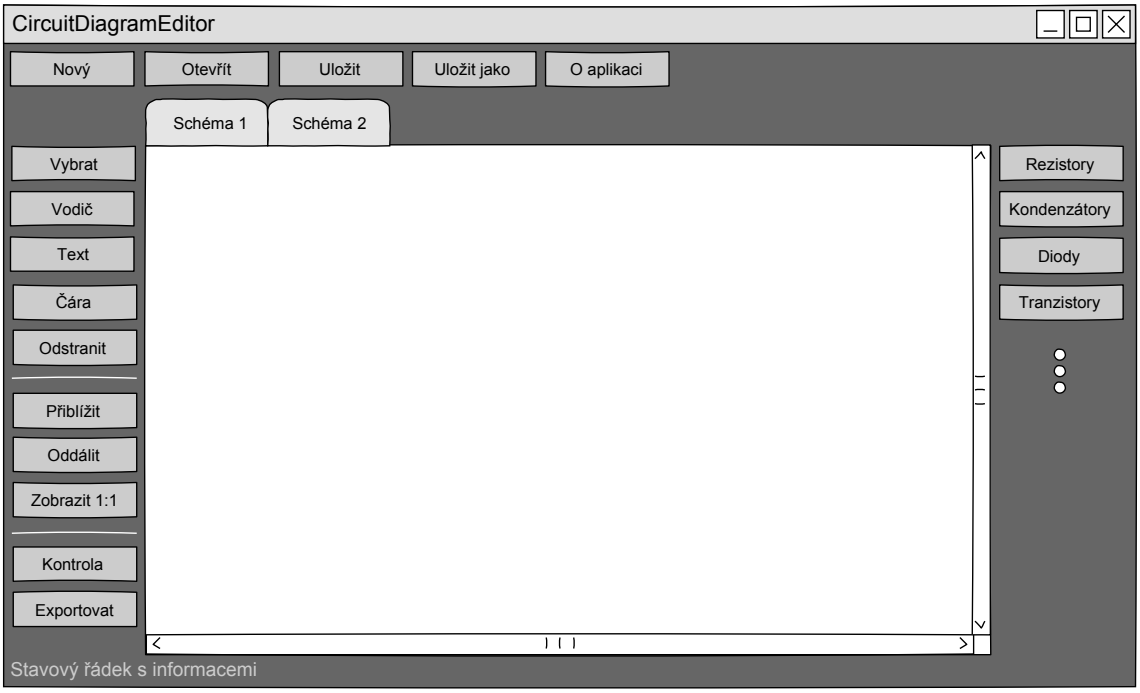
Při návrhu uživatelského rozhraní byl kladen maximální důraz na jednoduchost a intuitivnost ovládání. Jedním z cílů bylo vyvarovat se typickému rozbalovacímu menu (*Soubor, Úpravy, Zobrazit, ...*), které není příliš přehledné. Veškeré ovládání programu je řešeno pomocí tlačítek seskupených do logických celků a umístěných do nástrojových lišt (*toolbar*). Jak je možno vidět na obrázku 3.4, jsou v editoru celkem 3 nástrojové lišty, první z nich je zarovnaná k hornímu okraji okna aplikace a obsahuje tlačítka, která umožňují:

- vytvořit nové schéma (*Nový*)
- načíst uložené schéma (*Otevřít*)
- uložit aktuálně zobrazené schéma (*Uložit*)
- uložit aktuálně zobrazené schéma pod jiným názvem (*Uložit jako*)
- zobrazit informace o editoru (*O aplikaci*)

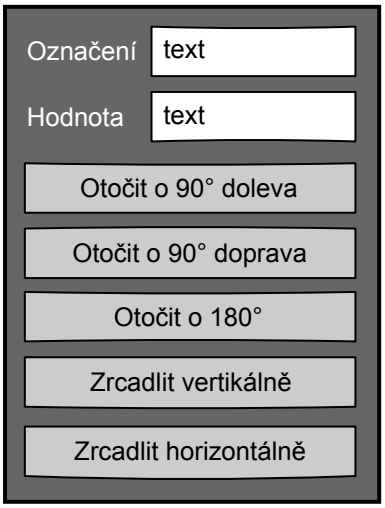
Druhá lišta je zarovnaná k levému okraji okna a tlačítka v ní obsažené umožňují:

- přesunout vloženou značku, vodič, text nebo čáru na jiné místo (*Vybrat*)
- vložit vodič (*Vodič*)
- vložit text (*Text*)
- vložit čáru (*Čára*)
- odstranit značku, vodič, text nebo čáru ze schématu (*Odstranit*)
- přiblížit schéma (*Přiblížit*)
- oddálit schéma (*Oddálit*)
- zobrazit schéma v původním měřítku (*Zobrazit 1:1*)
- zkontrolovat, jsou-li přípojné body všech prvků schématu zapojeny (*Kontrola*)
- exportovat schéma do grafického formátu (*Exportovat*)

Třetí lišta je u pravého okraje okna aplikace a obsahuje tlačítka pro vkládání schematických značek. Jejich počet a pojmenování závisí na obsahu souboru s knihovnou značek, který byl popsán v kapitole 3.1.1. Rotace a zrcadlení schematické značky se provádí ve speciálním dialogu (viz obrázek 3.5), který se zobrazí po stisku pravého tlačítka myši nad vloženou schematickou značkou. U dolního okraje okna je potom stavový řádek (*statusbar*), který poskytuje drobnou nápovědu k právě prováděné činnosti. Mezi nástrojovými lištami a stavovým řádkem zbývá velká plocha, která je určena pro kreslení schémat. Editor podporuje možnost mít více otevřených schémat zároveň. Přepínání mezi jednotlivými schématy je řešeno pomocí záložek, které jsou umístěny mezi kreslící plochou a horní nástrojovou lištou. Levá a pravá lišta je pak zobrazena pouze tehdy, je-li otevřeno alespoň jedno schéma, protože nemá smysl nabízet uživateli možnost vkládat značky a další prvky, pokud je není kam vložit.



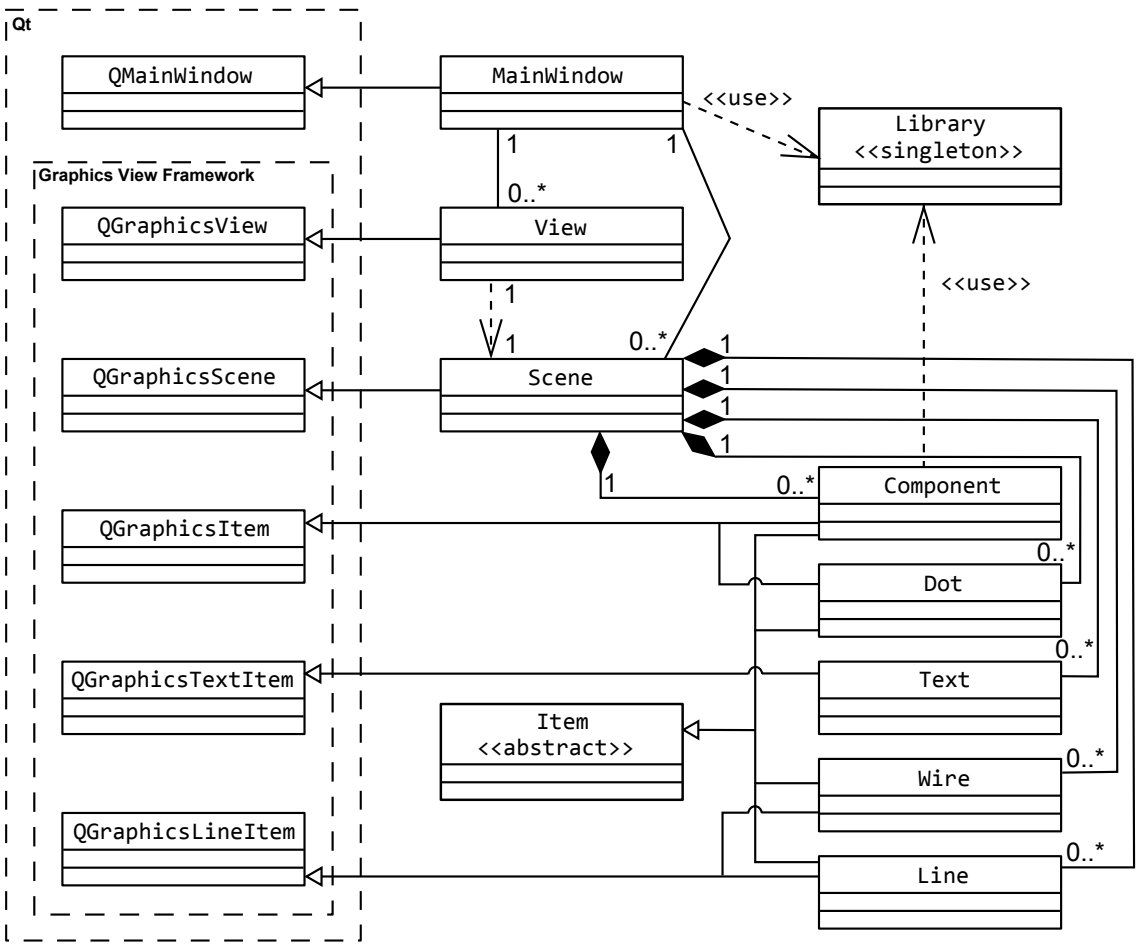
Obrázek 3.4: Návrh grafického uživatelského rozhraní



Obrázek 3.5: Dialog umožňující měnit označení a hodnotu značky a provádět její transformace

3.3 Objektově orientovaný návrh s využitím Qt

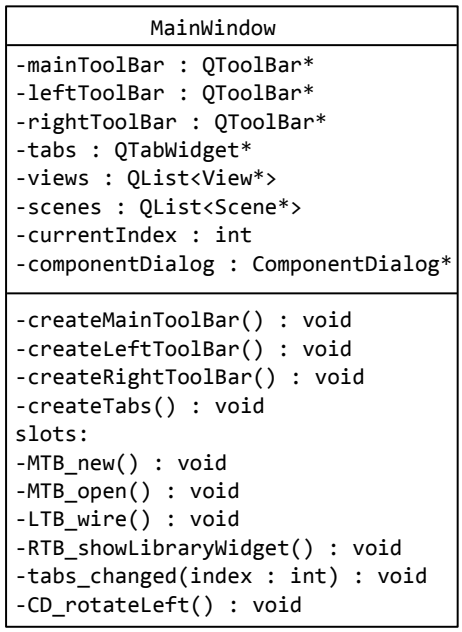
Třídy editoru a jejich vzájemné vztahy znázorňuje diagram tříd, který je na obrázku 3.6. Jedná se o zjednodušený diagram, ze kterého jsou vypuštěny některé nepodstatné části. Stejně tak jsou vypuštěny atributy a metody jednotlivých tříd. Ty jsou uvedeny až dále, při detailním popisu konkrétní třídy. Na uvedeném diagramu je vidět, že jádro aplikace je postaveno nad Graphics View Frameworkem, který byl popsán dříve v kapitole 2.2.



Obrázek 3.6: Zjednodušený diagram tříd

Třída `MainWindow`

Tato třída zajišťuje vytvoření hlavního okna aplikace, včetně všech prvků grafického uživatelského rozhraní. V konstruktoru se postupně volají metody `createMainToolBar()`, `createLeftToolbar()`, `createRightToolbar()` a `createTabs()`, které vytvoří horní, levou a pravou nástrojovou lištu a centrální *widget* se záložkami. Každé tlačítko v nástrojové liště je připojeno ke speciálnímu slotu, který zajišťuje adekvátní odezvu na jeho stisknutí. Například stisk tlačítka pro vytvoření nového schématu vyvolá metodu `MTB_new()`[2]. Kromě prvků GUI a obslužných metod pro tlačítka obsahuje MainWindow ještě 2 seznamy – `scenes` a `views`. Ty uchovávají aktuálně otevřené datové modely (instance třídy `Scene`) a jejich „zobrazovací *widgety*“ (instance třídy `View`). Pro identifikaci, která instance třídy `View` je aktuálně viditelná, se používá proměnná `currentIndex`. Ta obsahuje pořadové číslo aktuálně zobrazené záložky, které se při přepnutí záložky automaticky aktualizuje v metodě `tabs_changed(int index)`. Protože se záložky a instance tříd `Scene` a `View` vytvářejí a ruší vždy zároveň, může být pořadové číslo zobrazené záložky použito jako pořadové číslo aktuálně viditelného schématu (tj. „index“ do seznamů `views` a `scenes`). Dále `MainWindow` obsahuje dialog s názvem `componentDialog`, který je vyvolán při stisku pravého tlačítka myši nad vloženou schematickou značkou. Dialog poskytuje jednak textová pole pro změnu označení a hodnoty značky a také tlačítka umožňující provádět se značkou rotace či překlápění.



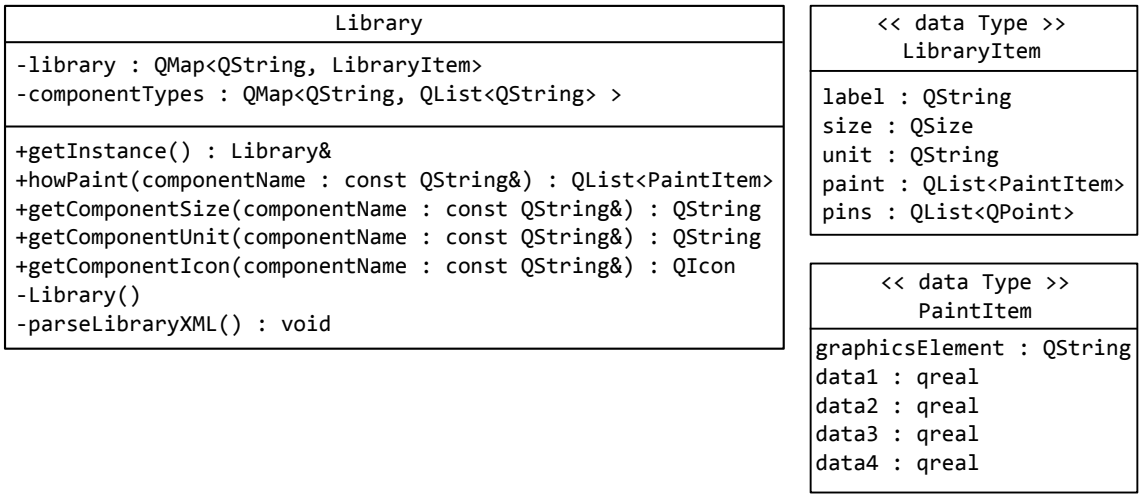
Obrázek 3.7: Zjednodušený diagram třídy `MainWindow`

Třída `Library`

Třída `Library` slouží jako knihovna schematických značek. Třída implementuje návrhový vzor Jedináček, protože editor je navržen pro práci vždy právě s jednou knihovnou značek. K instanci třídy `Library` lze přistoupit pouze pomocí statické metody `getInstance()`. Ta při prvním zavolání vytvoří instanci třídy `Library` a naplní její atributy daty, které získá ze souboru *library.xml*[3]. Při opakovaném volání metoda `getInstance()` vrací pouze referenci na již zkonstruovaný objekt. Všechny potřebné informace o schematických značkách jsou obsaženy v proměnné `library`, která mapuje řetězec (v tomto případě název značky v anglickém jazyce) na strukturu `LibraryItem` (viz obrázek 3.8). Každá veřejná metoda třídy proto přebírá jako parametr anglický název značky a vrací konkrétní informaci o ní. Například metoda `howPaint()` vrací seznam struktur `PaintItem`, které dohromady tvoří návod jak danou schematickou značku vykreslit pomocí základních geometrických tvarů.

[2]MTB je prefix používaný pro snadnější orientaci. Napovídá, že tlačítko je umístěno v hlavní nástrojové liště (*MainToolBar*).

[3]Formát souboru `library.xml` je popsán v kapitole 3.1.1.

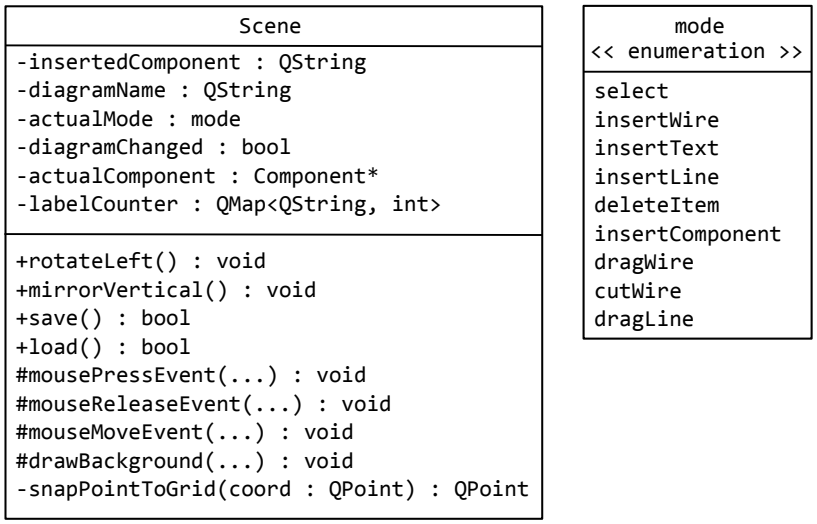


Obrázek 3.8: Zjednodušený diagram třídy `Library` a datových struktur `LibraryItem` a `PaintItem`

Třída `Scene`

Třída `Scene` reprezentuje datový model schématu. Při vytvoření instance této třídy se na její pozadí pomocí metody `drawBackground()` vloží vertikální a horizontální linky, které spolu tvoří rastr. Při vkládání značky nebo vodiče do schématu je pak umožněno vložit ji pouze na takové místo, aby všechny její přípojné body byly na průsečíku vertikální a horizontální linky. Pro zarovnání bodu do rastru na pozadí lze použít metodu `snapPointToGrid()`. Dále třída obsahuje metodu `save()`, která převádí obsah datového modelu schématu do XML dokumentu, jehož struktura byla uvedena v kapitole 3.1.2. Opak metody `save()` je metoda `load()`, která z uloženého XML dokumentu opět zkonstruuje všechny potřebné datové struktury. Třída rovněž zajišťuje, aby se požadovaná transformace (rotace, zrcadlení) provedla se správnou značkou. Používá k tomu ukazatel `actualComponent`, který vždy ukazuje na právě manipulovanou značku nebo na `NULL`. Každé schéma uchovává nějaké počítadlo vložených značek, aby bylo možné určit pořadí právě vložené značky a přiřadit jí tak správné označení (např.: D1, R1, D2, ...). K tomuto účelu slouží proměnná `labelCounter`, což je datová struktura *slovník*, která mapuje řetězec na číslo a umožňuje tak počítání výskytu daného řetězce. Interakce s uživatelem je zajištěna pomocí metod `mousePressEvent()`, `mouseMoveEvent()` a `mouseReleaseEvent()`. Automatické volání těchto metod při událostech myši zajišťuje Graphics View Framework, konkrétně pak třída `QGraphicsView`. Chování těchto metod je závislé na aktuálním módu[4], ve kterém se scéna nachází. Například je-li aktuální mód scény `insertWire`, tak se v metodě `mousePressEvent()` vloží do schématu vodič. Naproti tomu nachází-li se scéna v módu `insertComponent`, tak se v metodě `mousePressEvent()` vloží do schématu schematická značka.

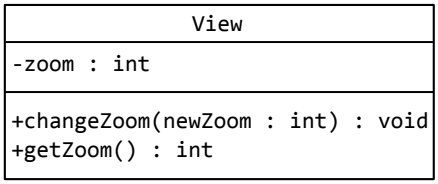
[4]Výčet všech možných módů (*mode*) je uveden na obrázku 3.9.



Obrázek 3.9: Zjednodušený diagram třídy `Scene` a výčtu `mode`

Třída `View`

Třída `View` zajišťuje zobrazení obsahu třídy `Scene`. Ve třídě `QGraphicsView`, od které je tato třída zděděná, je již implementováno téměř vše potřebné, a proto je třída `View` velmi jednoduchá. Obsahuje pouze podporu pro přibližování a oddalování pohledu. V atributu `zoom` je uchována aktuální hodnota přiblížení. Pro přečtení této hodnoty slouží metoda `getZoom()`. Při stisku tlačítka přiblížení nebo oddálení je zavolána metoda `changeZoom(int newZoom)`, která přebírá jako parametr novou hodnotu přiblížení. V těle metody je nová hodnota uložena a měřítko scény přepočítáno tak, aby jí odpovídalo.

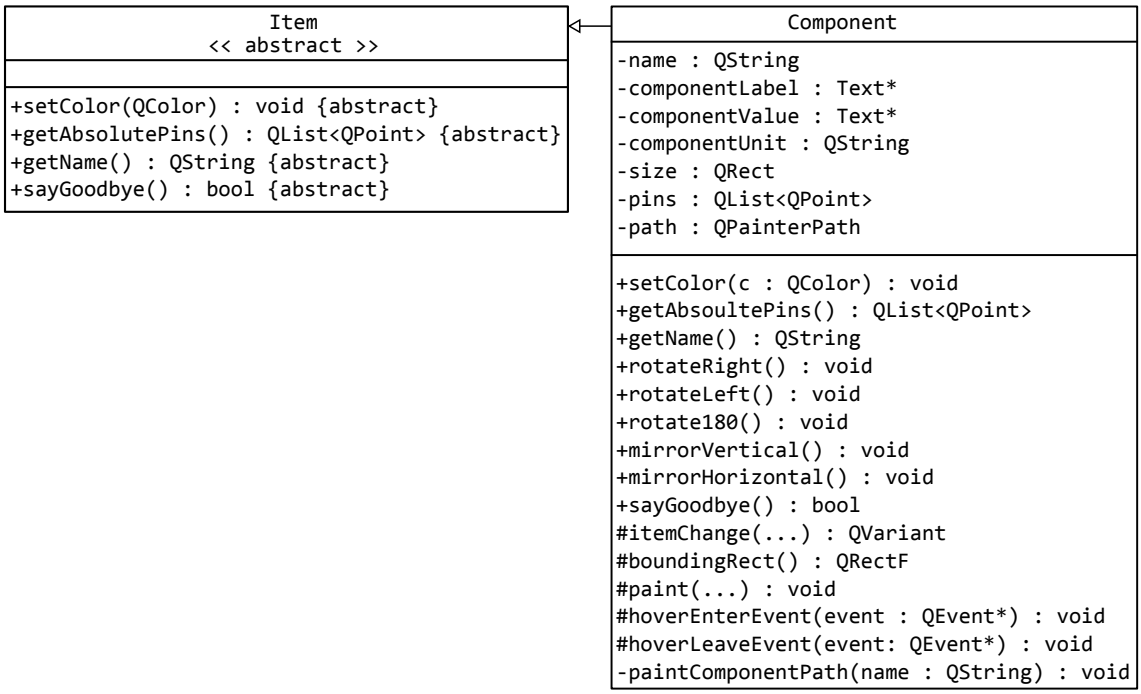


Obrázek 3.10: Zjednodušený diagram třídy `View`

Třídy `Item`, `Component`, `Wire`, `Line`, `Dot` a `Text`

`Item` je abstraktní třída, jejímž jediným úkolem je deklarovat 4 základní metody, které musejí všichni její potomci implementovat. První metoda je `setColor()`, ta zajišťuje změnu barvy daného grafického prvku. Další metoda je `getAbsolutePins()`, ta vrací seznam absolutních souřadnic, na nichž se nacházejí přípojné body daného prvku. Třetí metoda `getName()` vrací název, který slouží k identifikaci typu prvku. Poslední metoda, která má označení `sayGoodbye()`, se volá těsně před destrukcí objektu a umožňuje mu tak reagovat na vzniklý požadavek o odstranění. Rozdíl mezi destruktorem je v tom, že metoda `sayGoodbye()` může požadavek na odstranění odmítnout a tím pádem k odstranění prvku nedojde. Třídy, které dědí ze třídy `Item` a zároveň z některé „verze“ třídy `QGraphicsItem`, jsou `Component`, `Wire`, `Line`, `Text` a `Dot`. Třída `Wire` reprezentuje vodiče. Třída `Dot` reprezentuje „tečky“ naznačující spojení vodičů. Třída `Text` reprezentuje textové popisky. Třída `Line` reprezentuje čáry, které mohou být ve schématu použity například pro kreslení rámečků nebo razítek. Třída `Component` reprezentuje schematické značky. Následující popis se zaměřuje pouze na třídu `Component`, která je ze všech uvedených tříd nejsložitější. Až na pár detailů jsou třídy `Wire`, `Text`, `Dot` a `Line` pouze zjednodušené verze třídy `Component`.

Při vytváření instance třídy `Component` musí být předán do konstruktoru název značky, která má být zkonstruována. Název značky je pak v konstruktoru použit jako parametr pro metody třídy `Library`, které značce vrátí všechny potřebné informace, tedy rozměry, souřadnice přípojných bodů, fyzikální jednotku a tvar. Jak již bylo zmíněno dříve, tvar značky je vykreslen ze základních geometrických tvarů. Aby se nemusela značka při každém překreslení stále opakovaně sestavovat, je vytvořena pouze jednou a výsledný symbol je uložen do instance třídy `QPaintPath`, označené identifikátorem `path`. Současně s vytvářením třídy `Component` se vždy vytvoří i dvě instance třídy `Text`, které slouží pro uchování označení a hodnoty schematické značky. Dále každá značka poskytuje veřejné metody `rotate[Left|Right|180]()` a `mirror[Vertical|Horizontal]()`, které s ní umožňují provádět transformace. Aby nedošlo k situaci, že se součástka vychýlí z rastru, tak se vždy při změně pozice značky volá metoda `itemChange()`. V té je zajištěno zaokrouhlení souřadnice aktuální pozice na nejbližší souřadnici odpovídající mřížce na pozadí.



Obrázek 3.11: Zjednodušený diagram třídy `Item` a `Component`

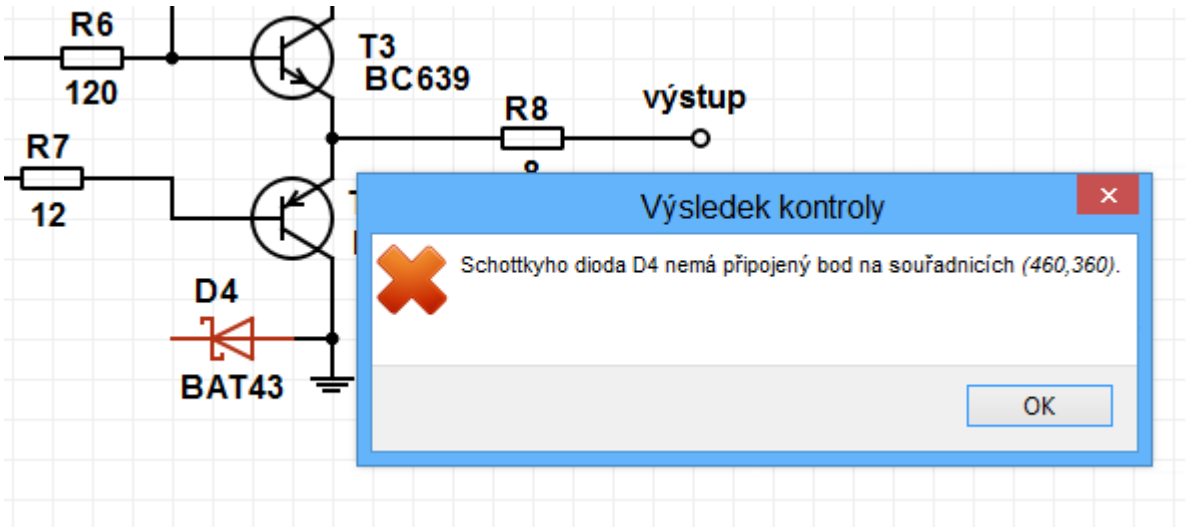
Použité návrhové vzory

V návrhu editoru mají být obsaženy některé návrhové vzory, což vzhledem k tomu, že editor je postaven na frameworku Qt, není příliš obtížné. Qt obsahuje desítky návrhových vzorů. Pokud je použito, jsou automaticky použity i návrhové vzory. Jeden z návrhových vzorů, který je v programu použit, je Jedináček. Na Jedináčkovi je založena třída `Library`. Vzor Skladba je použit ve třídě `QGraphicsItem`. Tento návrhový vzor umožňuje, aby se s jednou komponentou zacházelo stejným způsobem, jako se skupinou komponent. Toho je využíváno při označení více grafických prvků ve scéně, s nimiž lze potom pohybovat stejným způsobem, jako s prvkem jediným. Mechanismus signálů a slotů implementuje návrhový vzor Pozorovatel. V roli *Subject* zde vystupují signály a v roli *Observer* sloty. Tento návrhový vzor je rovněž obsažen v *Model-View* architektuře Graphics View Frameworku, konkrétně

ve třídách `QGraphicsScene` (*Subject*) a `QGraphicsView` (*Observer*). Třídy `QString` a `QFile` implementují vzor Fasáda. Tento vzor poskytuje vyšší úroveň rozhraní a zjednodušuje tak použití podsystému, což je v tomto případě řetězec jazyka C a ukazatel na objekt `FILE`. Cyklus `foreach()`, který je v programu často využíván pro procházení kontejnerů, je postaven na návrhovém vzoru Iterátor.

Automatická kontrola chybně zapojených prvků

Při kontrole správnosti zapojení prvků se nejprve vytvoří seznam všech prvků schématu, který lze snadno získat voláním metody `QGraphicsScene::items()`. Tento seznam se prochází v cyklu `foreach()`. V každé iteraci cyklu je otestováno, jestli na každý přípojný bod aktuálně testovaného prvku navazuje přípojný bod nějakého jiného prvku. Jestliže žádný prvek na aktuálně testovaný prvek nenavazuje, znamená to, že aktuálně testovaný prvek není správně zapojen. V takovém případě je na to uživatel upozorněn chybovým hlášením, které obsahuje souřadnice nezapojeného přípojného bodu. Najít nezapojený bod podle jeho souřadnic není složité, protože aktuální souřadnice kurzoru myši jsou vidět ve stavovém řádku. Mimo to se navíc nezapojená značka, nebo vodič zvýrazní červenou barvou (viz obrázek 3.12).

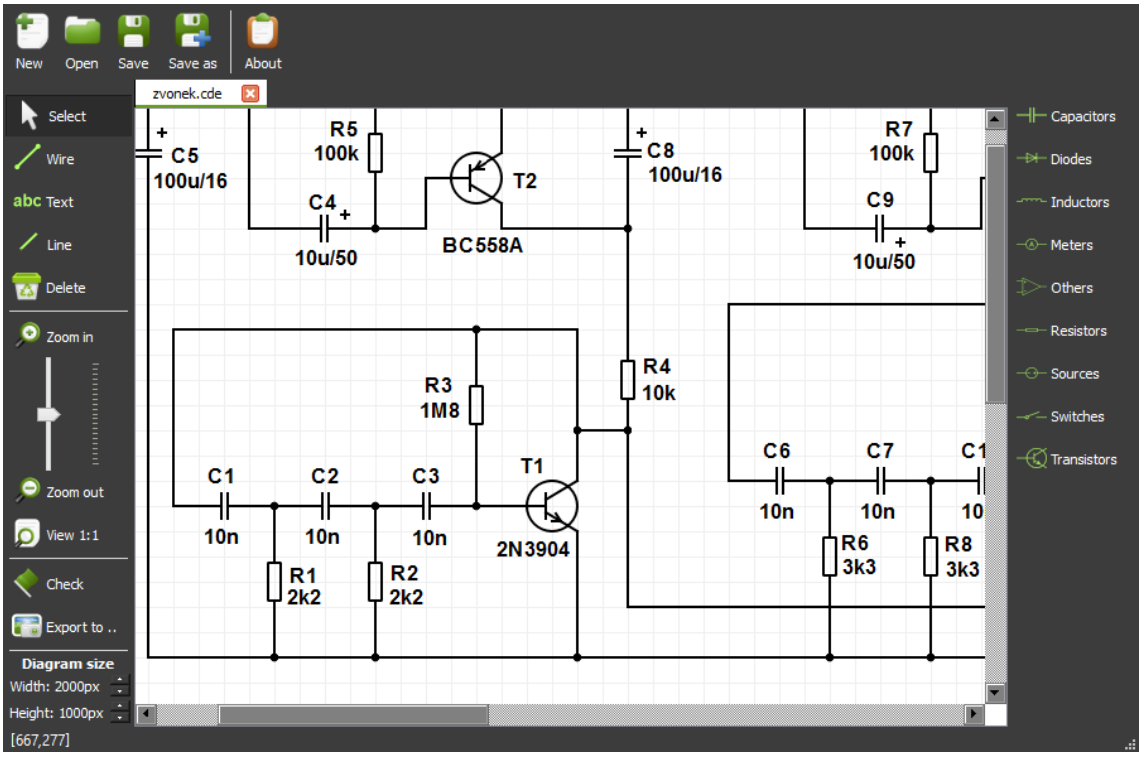


Obrázek 3.12: Chybové hlášení, které informuje o chybně zapojené schematické značce

Kapitola 4

Implementace a testování editoru

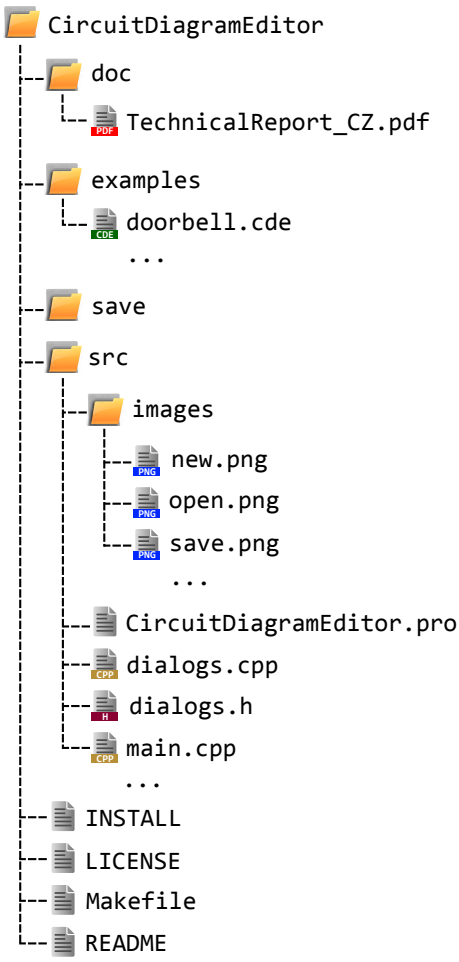
Podle návrhu uvedeném v kapitole 3 byl vytvořen editor *CircuitDiagramEditor*. Je implementován v programovacím jazyce C++ a při vývoji byl použit Qt framework ve verzi 5.0.1. Program ovšem nevyužívá žádné z nejnovějších rozšíření Qt. Požadovaná verze Qt je 4.7, nebo libovolná novější. Editor je přenositelný, lze jej provozovat pod 32 a 64 bitovými operačními systémy Microsoft Windows XP/Vista/7/8 a na Linuxových distribucích. V editoru jsou obsaženy dvě jazykové mutace, angličtina a čeština. Volba jazyka probíhá automaticky na základě nastaveného jazyka v operačním systému. Je-li prostřední nastaveno na češtinu (`LANG=cs_CZ`), tak se editor automaticky spustí v českém jazyce, v opačném případě se spustí v angličtině. Vzhled anglické verze editoru si můžete prohlédnout na obrázku 4.1.



Obrázek 4.1: Uživatelské rozhraní programu CircuitDiagramEditor

4.1 Organizace archivu se zdrojovými kódy

Archiv má kořenovou složku *CircuitDiagramEditor*, která obsahuje složky *doc*, *examples*, *save*, *src* a soubory *INSTALL*, *LICENSE*, *Makefile* a *README*. V souboru *README* je stručně vysvětleno ovládání programu. Ve složce *doc* je tato technická zpráva, rovněž se do této složky dá vygenerovat dokumentace programem *Doxygen*. Složka *examples* obsahuje nakreslená schémata ve formátu CDE. Složka *save* je prázdná, je nachystaná pro ukládání uživatelských schémat. Ve složce *src* jsou všechny zdrojové kódy programu, včetně všech obrázků použitých v grafickém uživatelském rozhraní.



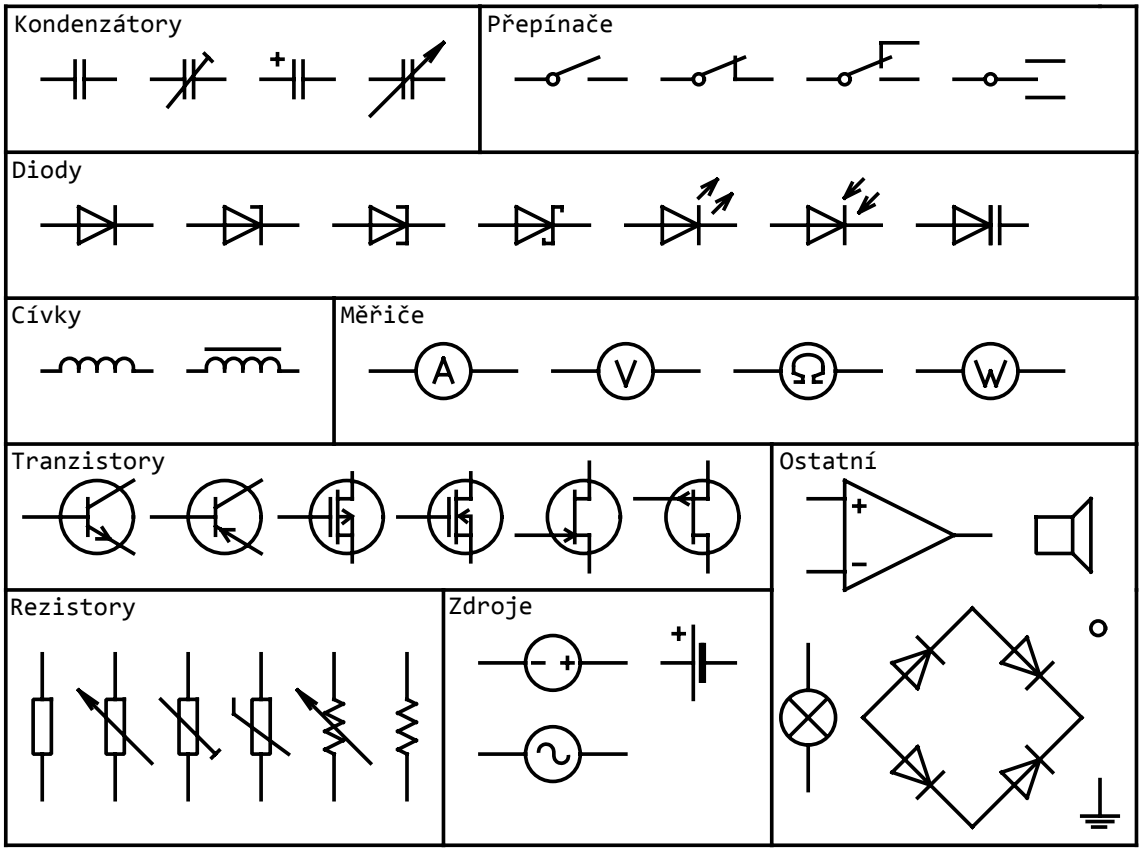
Obrázek 4.2: Adresářová struktura archivu se zdrojovými kódy

Překlad zdrojových kódů

Program se nedodává v binární podobě, je nutné jej sestavit ze zdrojových kódů. Pro úspěšný překlad zdrojových kódů je nutné mít v systému nainstalovaný překladač jazyka C++ (např. *GCC – GNU Compiler Collection*) s podporou standardu *C++98* a Qt framework verze 4.7, nebo novější. Jsou-li splněny uvedené podmínky, je možné přeložit CircuitDiagramEditor stejným způsobem jako se překládá kterákoliv jiná aplikace používající Qt. To znamená, že je nutné nejprve z platformově nezávislého projektového souboru vygenerovat platformově závislý *Makefile*. K tomu se používá `qmake`, kterému se jako parametr předá název projektového souboru, v tomto případě *CircuitDiagramEditor.pro*. Potom lze aplikaci přeložit vygenerovaným souborem *Makefile*. Na Linuxových distribucích je překlad jednodušší, lze použít soubor *Makefile*, který je součástí archivu. Napsáním příkazu `make` se provedou všechny výše popsané operace automaticky. Po úspěšném přeložení je možné spustit program příkazem `make run`. Alternativní možností jak sestavit program je otevřít soubor *CircuitDiagramEditor.pro* v integrovaném vývojovém prostředí *QtCreator* a stisknout tlačítko *Spustit*. Ať už je zvolena jakákoliv z popsaných metod překladu, tak je pro správný běh aplikace nutné mít v adresáři se spustitelným souborem obsaženy soubory *library.xml* a *images.qrc*, které jsou dodány v archivu ve složce *src*.

4.2 Schematické značky obsažené v knihovně značek

Jako knihovna schematických značek je použit soubor *library.xml*. Ten aktuálně obsahuje definici 42 schematických značek, které jsou rozděleny do 9 kategorií (viz obrázek 4.3). Přidání, odebrání, úprava nebo přesun značky do jiné kategorie lze provést upravením XML souboru *library.xml*. Způsob, jakým je značka v knihovně definována, byl uveden v kapitole 3.1.1.

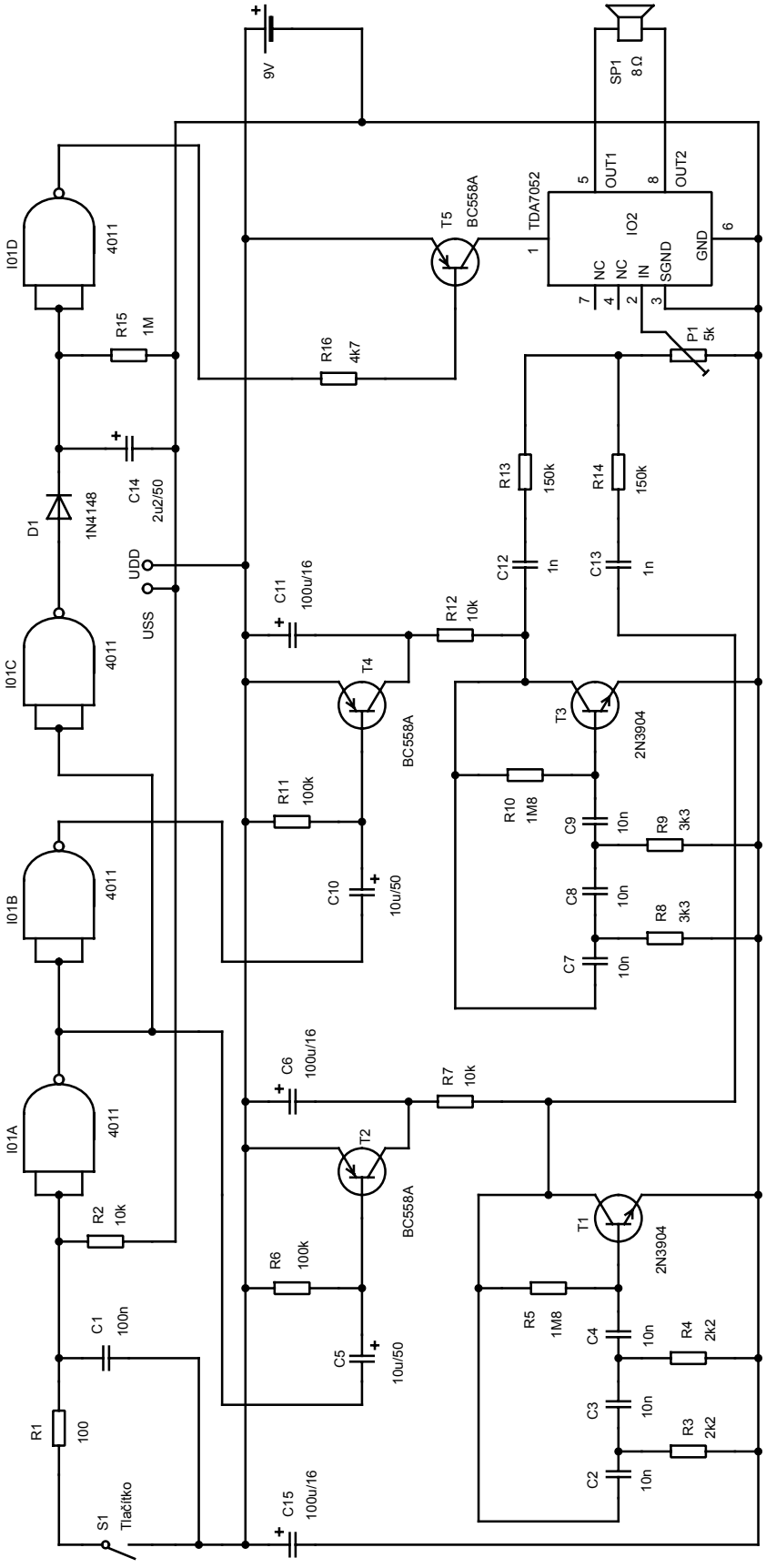


Obrázek 4.3: Schematické značky definované v souboru *library.xml*

4.3 Experimentální ověření funkčnosti

Editor byl vyvíjen v integrovaném vývojovém prostředí *QtCreator*, které mimo jiné poskytuje nástroje pro ladění (*GDB – The GNU Project Debugger*, *Valgrind*). Funkčnost editoru byla těmito nástroji ověřována průběžně během jeho vývoje. V rámci vývoje byly s editorem rovněž prováděny stovky experimentů, které jeho funkčnost ověřily v praxi.

Na obrázku 4.4 je netriviální schéma nakreslené pomocí CircuitDiagramEditoru, které demonstruje jeho funkčnost. Značka hradla NAND a zesilovače označeného TDA7052 byla přidána do editoru pouze krátkodobě, pro potřeby uvedeného schématu. Značky logických členů a integrovaných obvodů nejsou v aktuální verzi knihovny zahrnut. Ukázky dalších schémat je možno nalézt v archivu editoru ve složce *examples*.



Obrázek 4.4: Schéma zapojení bytového zvonku *„bim-bam“* vytvořené editorem

Vložení nakresleného schématu do HTML stránky

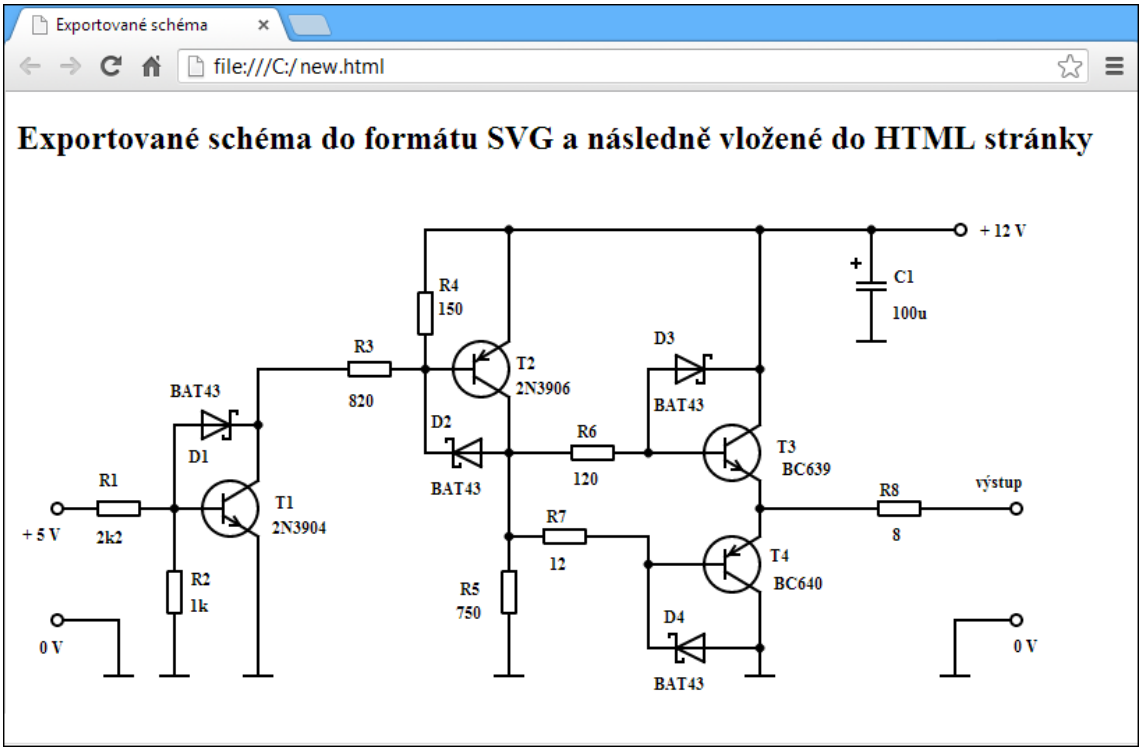
Schéma ve formátu PNG lze vložit do HTML stránky zápisem `<img src="schema.png">`. Vhodnější je ovšem vkládat na internet schémata ve formátu SVG, protože lze vektorové schéma libovolně zvětšit bez ztráty kvality. Pro vložení schématu ve formátu SVG lze použít například značku `<object>` (viz ukázka kódu 4.1). Ve všech novějších internetových prohlížečích, které podporují HTML 5, lze použít pro vložení i tzv. *inline SVG* (viz ukázka kódu 4.2). Vzhledem k velikosti generovaných SVG souborů se v praxi doporučuje používat vkládání pomocí značky `<object>`. Oba následující HTML kódy produkují stejný výstup, který je možné vidět na obrázku 4.5.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Exportovane schema</title>
    </head>
    <body>
        <h2>Exportovane schema do formatu SVG a nasledne vlozene do HTML
            stranky</h2>
        <object data="schema.svg" type="image/svg+xml"></object>
    </body>
</html>
```

Ukázka kódu 4.1: Jednoduchá HTML stránka, do které byl vložen obrázek ve formátu SVG pomocí značky `<object>`

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Exportovane schema</title>
    </head>
    <body>
        <h2>Exportovane schema do formatu SVG a nasledne vlozene do HTML
            stranky</h2>
        <svg width="471.664mm" height="233.186mm" xmlns="http://www.w3.org
            /2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"  version="
            1.2" baseProfile="tiny">
          <desc>An SVG drawing created by the Circuit Diagram Editor</desc>
          <defs></defs>
          <g fill="none" stroke="black" stroke-width="1" fill-rule="evenodd"
              stroke-linecap="square" stroke-linejoin="bevel" >
            <text fill="#000000" fill-opacity="1" stroke="none" xml:space="
                preserve" x="425" y="230" font-family="Times" font-size="12"
                 font-weight="700" font-style="normal">
            R4
            </text>
            <!-- dalsi SVG prikazy -->
          </g>
        </svg>
    </body>
</html>
```

Ukázka kódu 4.2: Jednoduchá HTML stránka, do které byl vložen obrázek ve formátu SVG pomocí *inline SVG*



Obrázek 4.5: Schéma ve formátu SVG zobrazené v prohlížeči Google Chrome 26

Kapitola 5

Závěr

V rámci této bakalářské práce byl vytvořen program CircuitDiagramEditor, který umožňuje uživatelsky přívětivě vytvářet a upravovat jednodušší elektronická schémata. Vytvořený editor je funkční a pro kreslení základních elektronických schémat zcela použitelný. Umožňuje automaticky zkontrolovat, jestli jsou všechny schematické značky a vodiče připojeny do schématu a vytvořená schémata lze exportovat do grafického formátu SVG nebo PNG. Editor obsahuje dvě lokalizace, anglickou a českou. Volba lokalizace probíhá automaticky podle aktuálně nastaveného jazyka v operačním systému. Program je distribuovaný pod licencí GNU General Public License, což mimo jiné znamená, že jeho zdrojové kódy jsou volně přístupné. V rámci vývoje programu byly navrženy také dva nové souborové formáty, jeden pro definici knihovny schematických značek a druhý pro ukládání schémat.

Pro použití editoru v praxi by bylo vhodné doplnit řadu dalších vlastností, jako jsou například operace Zpět (*Undo*) a Opakovat (*Redo*), nebo export nakresleného schématu do nějakého standardního formátu pro simulaci (například *netlist* programu *SPICE – Simulation Program with Integrated Circuit Emphasis*). Zajímavým, ale implementačně složitým rozšířením by bylo propojení editoru s fotoaparátem (nebo skenerem) a následný automatický import vyfocených (naskenovaných) schémat.

Literatura

[1] CadSoft EAGLE. http://www.cadsoftusa.com/, [cit. 2013-04-17].

[2] gEDA Project Wiki. http://wiki.geda-project.org/geda:download, [cit. 2013-04-17].

[3] Graphics View Framework. http://qt-project.org/doc/qt-4.8/graphicsview.html, [cit. 2013-04-17].

[4] IEC 60617 - Graphical Symbols for Diagrams. http://std.iec.ch/iec60617, [cit. 2013-04-17].

[5] Qt Project. http://qt-project.org/, [cit. 2013-04-17].

[6] Qucs project. http://qucs.sourceforge.net/, [cit. 2013-04-17].

[7] Standard IEEE 315. http://www.scribd.com/doc/47366971/Standard-IEEE-315-Simbologia-Electrica, [cit. 2013-04-17].

[8] Electronic component - Wikipedia. http://en.wikipedia.org/wiki/Electronic_component, [cit. 2013-05-10].

[9] Blanchette, J.; Summerfield, M.: *C++ GUI Programming with Qt 4 Second Edition*. Prentice Hall, 2008, iSBN 0-13-235416-0.

[10] Edwards, T.: XCircuit. http://opencircuitdesign.com/xcircuit/, [cit. 2013-04-17].

[11] Gamma, E.; Helm, R.; Johnson, R.; aj.: *Návrh programů pomocí vzorů*. Grada, 2003, iSBN 80-247-0302-5.

[12] Lilley, C.; Schepers, D.: W3C SVG Working Group. http://www.w3.org/Graphics/SVG/, [cit. 2013-04-17].

[13] Láníček, R.: *Elektronika - obvody, součástky, děje*. BEN-Technická literatura, 2002, iSBN 80-86056-25-2.

[14] Quin, L.: Extensible Markup Language (XML). http://www.w3.org/XML/, [cit. 2013-04-17].

[15] Roelofs, G.: PNG (Portable Network Graphics) Home Site. http://www.libpng.org/pub/png/, [cit. 2013-04-17].