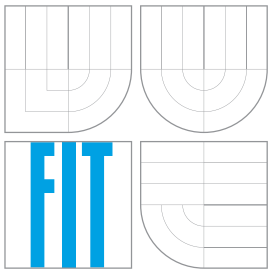
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VIZUÁLNÍ EDITOR ELEKTRICKÝCH SCHEMAT

ELECTRONIC CIRCUITS EDITOR

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL KADÁK

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. KUNOVSKÝ JIŘÍ, CSc.

BRNO 2009

Zadání práce

1. Seznamte se se způsoby zadávání a popisu elektrických obvodů ve světových standardech (Dymola, Matlab, Maple).
2. Seznamte se se způsoby výpočtu elektrických obvodů ve světových standardech (Dymola, Matlab, Maple).
3. Seznamte se s problematikou algebraických úprav při řešení elektrických obvodů a analyzujte možnosti aplikace parazitních kapacit a parazitních indukčností.
4. Navrhněte editor pro vizuální tvorbu elektrických schemat s automatickým vkládáním parazitních prvků.
5. Navržený editor implementujte a ověřte správnou funkci na experimentálních příkladech.
6. Srovnejte výsledky se světovými standardy(Dymola, Matlab, Maple).

Licenční smlouva

Licenční smlouva je uložena v archívu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Tato diplomová práce se zabývá možnostmi modelování elektrických obvodů a metodami řešení takových modelů. Zaměřuje se na analýzu dnešních systémů, aby jejich rysy mohly být dále použity v návrhu vlastního grafického editoru.

Abstract

This work deals with the possibilities of modeling electrical circuits and methods of solving these models. It focuses on the analysis of today's systems, so that their features can be used in our graphic editor design.

Klíčová slova

elektrické obvody, diferenciální rovnice, numerické metody, Taylorova řada, autonomní metoda, TKSL, grafický editor, parazitní kapacity

Keywords

electrical circuits, differential equations, numerical methods, Taylor series, autonomical method, TKSL, graphic editor, parasitic capacitances

Citace

Michal Kadák: Vizuální editor elektrických schemat, diplomová práce, Brno, FIT VUT v Brně, 2009

Vizuální editor elektrických schemat

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Ing. Jiřího Kunovského, CSc.

.....
Michal Kadák
24. mája 2009

Poděkování

Tímto bych rád poděkoval doc. Ing. Jiřímu Kunovskému, CSc. za ochotné konzultace i praktické rady. Díky patří i rodičům za jejich podporu při studiu.

Obsah

Kapitola 1

Úvod

Už pár rokov sa na našej fakulte vyvíja inovatívne simulačné prostredie, ktoré rieši výpočtovo veľmi náročné problémy za pomoci Taylorovho radu, s názvom TKSL. Aktuálna verzia TKSL disponuje textovým vstupom, ktorý definuje sústavy diferenciálnych rovníc. Na základe tohto vstupu simulátor rieši takto zadané sústavy diferenciálnych rovníc.

Ako si ukážeme v ďalších kapitolách, riešenie sústav diferenciálnych rovníc je vhodnou metodikou riešenia elektrických obvodov. To je jeden z dôvodov, prečo sa simulačný nástroj TKSL momentálne používa prevažne na počítanie napätí a prúdov v zložitých (ale aj jednoduchých) elektrických obvodoch.

Cieľom diplomovej práce je preskúmať spôsob zadávania a riešenia elektrických schém, zamyslieť sa nad využitím parazitných kapacít a indukčností, zvážiť problémy a z nich plynúce požiadavky na návrh a implementáciu a využiť ich v návrhu grafického editoru a ten následne implementovať.

Editor bude slúžiť ako nadstavba nad systémom TKSL. Grafická nadstavba zjednoduší zadávanie elektrických schém, a tým funkčne rozšíri systém TKSL, ktorý zatiaľ disponuje len textovým vstupom. Po zadaní (nakreslení) elektrického obvodu sa extrahuje model, ktorý bude následne spracovaný transformačným modulom. Tento modul transformuje model obvodu na sústavu diferenciálnych rovníc. Takto spracovaný vstup už je vhodný pre aktuálnu verziu TKSL.

Výsledkom mojej diplomovej práce je grafický editor k programu TKSL, ktorý bude spájať pozitíva profesionálnych systémov s potrebami systému TKSL a jeho výpočtových metód.

1.1 Členenie práce

Celá práca je členená do kapitol s názvami podľa tematických okruhov, ktoré popisujú.

1. Úvod

   Kapitola 1, ktorú práve čítate, slúži k uvedeniu do širšieho kontextu problematiky a popisuje motiváciu jej vzniku. V druhej časti popisuje štruktúru členenia práce.

2. Riešenie elektrických obvodov

   Nasledujúca kapitola, s číslom 2, sa zaoberá metódami riešenia elektrických obvodov. Má štyri podkapitoly, ktoré vysvetľujú princípy a postupy týchto metód.

3. Parazitné kondenzátory pri riešení elektrických obvodov

   Ďalšia kapitola, s číslom 3, popisuje výhody konceptu parazitných kapacít pri riešení obvodov diferenciálnymi rovnicami.

4. Modelovanie (zobrazovanie) elektrických obvodov

   Kapitola číslo 4 má za úlohu popísať zaužívané a štandardné techniky modelovania elektrických obvodov. Tieto techniky sa použijú v návrhu editoru.

5. Návrh a analýza systému TKSL PowerGear

   Nasledujúca kapitola, s číslom 5, slúži k návrhu a analýze systému s ohľadom na získané informácie v predchádzajúcich kapitolách.

6. Implementácia systému TKSL PowerGear

   Kapitola číslo 6 už podrobne popisuje postup implementácie navrhnutého systému.

7. Systém TKSL PowerGear

   V kapitole 7 sa nachádza popis systému z pohľadu užívateľa s návodom a ukážkami funkcií.

8. Možnosti rozšírenia systému TKSL PowerGear

   Predposledná kapitola 8 ukazuje niektoré rozšírenia, na ktoré je systém pripravený a v blízkej dobe môžu byť implementované.

9. Záver

   Posledná kapitola, s číslom 9, analyzuje dosiahnuté výsledky diplomovej práce.

Kapitola 2

Riešenie elektrických obvodov

2.1 Analytické riešenie

K riešeniu diferenciálnych rovníc (sústav rovníc) je možné pristupovať analyticky. Výsledným riešením je funkcia času. Konkrétnu hodnotu v určitom čase získame po dosadení daného času do výslednej rovnice. Hodnotu je možné určiť v ľubovoľnom bode, v ktorom je funkcia definovaná. Z toho vyplýva, že analytické riešenie je veľmi presné. Je však v mnohých prípadoch zložité a náročné na čas.

V teórii obyčajných diferenciálnych rovníc sa analytické riešenie dosahuje použitím obecných schém riešenia nájdených pre určité skupiny diferenciálnych rovníc. Hľadá sa teda riešenie v známom tvare. Najkomplexnejšie je popísané riešenie lineárnych rovníc s konštantnými koeficientmi. Ostatné typy sa snažíme vhodne transformovať na tieto rovnice. Lineárne rovnice s nekonštantnými koeficientmi transformujeme na lineárne rovnice s konštantnými koeficientmi, nelineárne rovnice transformujeme na lineárne, prípadne riešime priamou integráciou. Viac o analytickom riešení diferenciálnych rovníc je možné sa dočítať napr. v [7].

Analytické metódy sú v praxi málo využiteľné. Reálne úlohy z technickej a fyzikálnej praxe vedú často na veľmi zložité riešenie sústav diferenciálnych rovníc. S rozvojom a zvyšovaním výkonu výpočtovej techniky ich preto nahradzujú numerické metódy riešenia. Nedosahujú takú presnosť ako analytické metódy, pri akejkoľvek zmene parametrov je nutné uskutočniť celý výpočet znovu, umožňujú ale riešiť omnoho väčší rozsah problémov z reálnej praxe.[14]

2.2 Numerické riešenie

Numerické metódy sú obvykle jednoduchšie a rýchlejšie než analytické. Hľadať riešenie sústavy obyčajných diferenciálnych rovníc numerickou metódou má zmysel iba v prípade, že existuje jej jednoznačné riešenie. Pri použití numerických metód riešenia je výsledným riešením postupnosť hodnôt vo vopred zvolených časových bodoch. Hodnoty medzi danými bodmi je možné získať interpoláciou z už vypočítaných okolitých bodov, alebo opätovným aplikovaním zvolenej metódy s menším krokom výpočtu (rozostupom časových bodov). Pri použití nesprávneho kroku so zvolenou metódou však môže dôjsť k veľkej chybe výpočtu.

Majme sústavu n obyčajných diferenciálnych rovníc

$$\begin{array}{rcl} y_1'(t) & = & f_1(t, y_1, \ldots, y_n) \\ y_2'(t) & = & f_2(t, y_1, \ldots, y_n) \\ & \vdots & \\ y_{n-1}'(t) & = & f_{n-1}(t, y_1, \ldots, y_n) \\ y_n'(t) & = & f_n(t, y_1, \ldots, y_n) \end{array} \tag{2.1}$$

s počiatočnými podmienkami

$$\begin{array}{rcl} y_1(t_0) & = & y_{1,0} \\ y_2(t_0) & = & y_{2,0} \\ & \vdots & \\ y_{n-1}(t_0) & = & y_{n-1,0} \\ y_n(t_0) & = & y_{n,0} \end{array} \tag{2.2}$$

Základom väčšiny numerických metód pre riešenie počiatočných úloh na intervale $< a, b >$ je diskretizácia premennej. Množinu bodov $t_i, \quad i \in < 0, k >$ z intervalu $< a, b >$, kde

$$a = t_0 < t_1 < t_2 < \cdots < t_{k-1} < t_k = b$$

nazývame sieť. Jej prvky sú uzly siete.

Výraz

$$t_{i+1} - t_i = h$$

nazývame krokom siete v uzle x_i. Ak je $h_i = h = konst.$, jedná sa naviac o pravidelnú sieť.

Numerickým riešením sústavy 2.1 rozumieme postupnosť y_i hodnôt $y(t_0), y(t_1), \ldots y(t_k)$. Jednotlivé hodnoty odpovedajú hodnotám príslušných uzlov v sieti. Hodnoty exaktného riešenia, ktoré získame dosadením do analytického riešenia, označíme $Y_i = Y(t_i)$.

Pre použiteľnosť numerickej metódy je nutnou podmienkou existencia limity postupnosti y_i pre $h \to 0$, $i \to \infty$, kde $hi = t$ je pevné, t.j. postupnosť y_i musí konvergovať pre $h \to 0$ k exaktnému riešeniu $Y(t)$.

Numerické metódy riešenia diferenciálnych rovníc delíme na dva typy:

1. Metódy, ktoré zisťujú hodnoty funkcie $f(t, y)$ medzi jednotlivými uzlami siete (t_i, y_i). Sú zastúpené jednokrokovými metódami Runge-Kutta.

2. Metódy, ktoré počítajú hodnoty funkcie $f(t, y)$ iba v bodoch (t_i, y_i), kde y_i je hodnota numerického riešenia v bode $t = t_i$. Jedná sa o viackrokové metódy.

Oba uvedené typy využívajú k riešeniu iba prvé derivácie y.[14]

2.2.1 Jednokrokové metódy

Jednokrokové metódy spočívajú vo výpočte hodnoty y_{n+1} iba z hodnoty y_n vypočítanej v predchádzajúcom kroku. Táto skutočnosť prináša výhodu v prípade zmeny integračného kroku. Základom pre jednokrokové metódy je Taylorov rozvoj (rovnica 2.3):

$$y_{n+1} = y_n + hy_n' + \frac{h^2}{2!}y_n'' + \frac{h^3}{3!}y_n^3 + \cdots \tag{2.3}$$

Medzi jednokrokové metódy patria napríklad Eulerova metóda, metódy Runge-Kutta a metóda Taylorovho radu, ktorá bude bližšie popísaná v sekcii 2.3.

Eulerova metóda

Jedná sa o najjednoduchšiu metódu. Pre výpočet využíva iba prvé dva členy Taylorovho radu:

$$y_{n+1} = y_n + hy'_n \tag{2.4}$$

Vzťah 2.4 predstavuje rovnicu priamky, ktorá má smernicu $f(x_i, y_i)$ a prechádza bodom (x_i, y_i). Na intervale $< x_i, x_{i+1} >$ sa teda pohybujeme po dotyčnici k exaktnému riešeniu úlohy v bode (x_i, y_i).

Výsledok metódy je možné spresňovať skracovaním kroku výpočtu. Od určitej hranice sa však začne prejavovať výraznejšie zaokrúhľovacia chyba a znižovaním kroku výsledná chyba výpočtu vzrastie.[14]

Metódy Runge-Kutta

Tieto metódy patria medzi jednokrokové metódy, ktoré realizujú výpočet $f(t, y)$ aj medzi jednotlivými uzlami (t_i, y_i). Ich základom je vyjadrenie rozdielu medzi hodnotami riešenia y v bodoch t_{n+1} a t_n v tvare

$$y_{n+1} - y_n = \sum_{i=1}^{p} w_i K_i$$

kde w_i sú konštanty a

$$K_i = hf(t_n + a_i h, y_n + \sum_{j=1}^{i-1} b_{ij} k_j), \quad i = 1, \dots p$$

kde $h = t_{n+1} - t_n$ a a_i,b_{ij} sú konštanty a $a_1 = 0$.

Konštanty w_i,a_i a b_{ij} sú zvolené tak, aby riešenie zodpovedalo riešeniu Taylorovým radom v bode (t_n, y_n) až do p-tej mocniny kroku h. P-tý rád metódy Runge-Kutta je ekvivalentný p-tej mocnine.

Metód Runge-Kutta je viacero modifikácií, ktoré sa líšia predovšetkým koeficientmi a ohraničeným oborom absolútnej stability [1]. Najčastejšie používaná je metóda štvrtého rádu, ktorá má dobrú stabilitu aj presnosť.

Viac o metóde Runge-Kutta napríklad v [6].

2.2.2 Viackrokové metódy

Viackrokové metódy sú založené na výpočte hodnoty y_{n+1} pomocou k hodnôt vypočítaných v predchádzajúcich krokoch. Všeobecne ich môžeme vyjadriť:

$$y_{n+1} = \sum_{i=0}^{r} a_i y_{n-i} + h \sum_{i=-1}^{r} b_i y'_{n-i}$$

a_i, b_i sú konštanty.

Viackrokové metódy sú v porovnaní s niektorými jednokrokovými metódami presnejšie, no prinášajú viacero nevýhod. Pri štarte výpočtu je nutné použiť niektorú jednokrokovú metódu, ktorou vypočítame prvých k hodnôt potrebných k výpočtu prvej hodnoty viackrokovou metódou. Problematická je tiež zmena veľkosti kroku v priebehu výpočtu, kedy je napríklad potrebné si pri dvojnásobnom zväčšení kroku pamätať dvojnásobný počet predchádzajúcich krokov.[14]

2.3 Metóda Taylorovho radu

Taylorov rad je definovaný ako nekonečný mocninový rad

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^n(a)}{n!}(x-a)^n \tag{2.5}$$

Ak zvolíme počiatočnú podmienku $a1 = 0$ a $h = x1 - a1$, dostávame rovnicu

$$f(x_1) = f(0) + f'(0) \cdot h + \frac{f''(0)}{2!}h^2 + \frac{f''(0)}{3!}h^3 + \ldots + \frac{f^n(0)}{n!}h^n \tag{2.6}$$

Položíme $a2 = x1$ za predpokladu $h = x2 - a2 = x1 - a1$. Potom platí

$$f(x_2) = f(x_1) + f'(x_1) \cdot h + \frac{f''(x_1)}{2!}h^2 + \frac{f''(x_1)}{3!}h^3 + \ldots + \frac{f^n(x_1)}{n!}h^n \tag{2.7}$$

Hodnoty funkcie $f(x)$ v bodoch $x1$, $x2$ je možné vypočítať postupne využitím Taylorovho radu. Výsledok jedného kroku je potrebný pre výpočet ďalších čiastkových výsledkov. Parameter h je integračný krok, ktorý nemusí byť konštantný a pre jednotlivé kroky výpočtu sa môže meniť. Rýchlosť výpočtu a presnosť je závislá práve na veľkosti integračného kroku. So skracujúcim sa integračným krokom klesá rýchlosť výpočtu, ale rastie jeho presnosť. Počas výpočtu sčítavame čiastkové výsledky a v prípade, že rozdiel dvoch po sebe idúcich výsledkov je menší než presnosť, ktorú sme si stanovili pred začiatkom výpočtu, výpočet ukončíme.

K čiastkovým výpočtom sú potrebné vyššie derivácie funkcie. Tie je možné odvodiť z predchádzajúcich výpočtov, čím sa vyhneme zbytočnému a časovo náročnému výpočtu týchto derivácií (pozri [1]). Odvodenie si ukážeme na nasledujúcej sústave diferenciálnych rovníc.

$$y' = A \cdot y + B \cdot z \qquad\qquad z' = C \cdot y + D \cdot z \tag{2.8}$$

Počiatočné podmienky $y(0) = y_0$, $z(0) = z_0$.

Riešenie klasickým spôsobom:

$$y_1 = y_0 + h \cdot y'(0) + \frac{h^2}{2!} \cdot y''(0) + \frac{h^3}{3!} \cdot y'''(0) + \cdots \tag{2.9}$$

$$z_1 = z_0 + h \cdot z'(0) + \frac{h^2}{2!} \cdot z''(0) + \frac{h^3}{3!} \cdot z'''(0) + \cdots \tag{2.10}$$

Zjednodušenie výpočtu sústavy odvodením z predchádzajúcich výpočtov:

$$y_1 = y_0 + DY10 + DY20 + DY30 + \cdots \tag{2.11}$$

$$z_1 = z_0 + DZ10 + DZ20 + DZ30 + \cdots \tag{2.12}$$

Pre jednotlivé členy:

$$\begin{aligned} DY10 &= h \cdot y'(0) = h(A \cdot y + B \cdot z) \\ DY20 &= \frac{h}{2}(A \cdot DY10 + B \cdot DZ10) \\ DY30 &= \frac{h}{3}(A \cdot DY20 + B \cdot DZ20) \\ &\vdots \end{aligned}$$

$$
\begin{aligned}
DZ10 &= h \cdot z'(0) = h(C \cdot y + D \cdot z) \\
DZ20 &= \frac{h}{2}(C \cdot DY10 + D \cdot DZ10) \\
DZ30 &= \frac{h}{3}(C \cdot DY20 + D \cdot DZ20) \\
&\vdots
\end{aligned}
$$

Pre využitie metódy Taylorovho radu je kľúčová správna hodnota integračného kroku. Ak zvolíme nevhodnú veľkosť integračného kroku, môže sa stať metóda nestabilnou. Je to spôsobené tým, že vzniknutá chyba sa prenáša do ďalších výpočtov a celková chyba výpočtu môže narastať.

Metóda Taylorovho radu poskytuje, v porovnaní s ostatnými jednokrokovými metódami, ktoré sú z Taylorovho radu odvodené, vyššiu presnosť výpočtu. Predpokladá sa pritom využitie väčšieho počtu členov Taylorovho rozvoja než u ostatných metód (rádovo desiatky). Ak získame derivácie vyšších rádov, sme schopní dosiahnuť veľmi presných výsledkov, ktoré už môžu byť obmedzené iba modelom čísel v použitej počítačovej aritmetike. Ďalšou výhodou je možnosť paralelizácie metódy za účelom vyššieho výpočtového výkonu. [14]

Pri riešení elektrických obvodov diferenciálnymi rovnicami sa používa numerické aj analytické riešenie. Otázkou zostáva, akými metódami elektrické obvody riešiť.

2.4 Autonómna metóda

Autonómna metóda riešenia elektrických obvodov sa niekedy nazýva metódou priameho modelovania [1]. Princípom metódy je zostavenie sústavy rovníc na základe matematického popisu jednotlivých prvkov obvodu – pasívnych a akumulačných súčiastok, uzlov a vetiev. Pre každý prvok[1] zostavíme individuálne rovnicu. Získame väčší počet rovníc než u klasických metód, ale rovnice sú jednoduchšie.

Postup analýzy obvodu u autonómnej metódy je nasledovný:

1. Vyznačíme jednotlivé uzly obvodu a každému pridelíme neznáme napätie.
2. Vyznačíme vetvy obvodu. Vetva predstavuje časť obvodu medzi dvoma uzlami. Pre každú vetvu určíme smer prúdu od uzla s vyšším predpokladaným potenciálom k uzlu s nižším predpokladaným potenciálom.
3. Označíme pasívne a akumulačné prvky obvodu (ideálny rezistor a ideálna cievka a kondenzátor).

Z takto štruktúrovaného obvodu zostavíme rovnice postupom:

1. Pre každý uzol obvodu nedefinujeme súčet prúdov (prúdy priradené vetvám, ktoré uzol spája) podľa prvého Kirchhoffovho zákona.

$$\sum_{i=0}^{n} I_i = 0$$

Prúdy vstupujúce do uzla značíme ako kladné, vystupujúce ako záporné.

[1] Prvkom v tomto prípade rozumieme nielen súčiastky v obvode, ale aj uzly a vetvy obvodu.

2. Pre každú vetvu zapíšeme, že rozdiel napätí (potenciálov) koncových uzlov vetvy sa rovná súčtu napätí prvkov vo vetve. Rozdiel zapisujeme v smere predpokladaného prúdu v danej vetve.

$$U_X - U_Y = \sum_{i=0}^{n} U_{(R,C,L)i}$$

3. Pre každý pasívny a akumulačný prvok zapíšeme zodpovedajúce vzťahy medzi napätím a prúdom.

V obvode zohľadňujeme zdroje napätia tak, že predstavujú uzly obvodu. Takýto uzol zastupuje v jednej vetve koncové napätie rovné napätiu zdroja, v druhej vetve koncový nulový potenciál. Jeden reálny uzol obvodu, bezprostredne nachádzajúci sa pri póle zdroja (ak existuje), vyznačíme pomocným nulovým potenciálom ($U_0 = 0V$).

Každá rovnica popisujúca prvok obvodu je zostavovaná samostatne, odtiaľ názov autonómna metóda. [14]

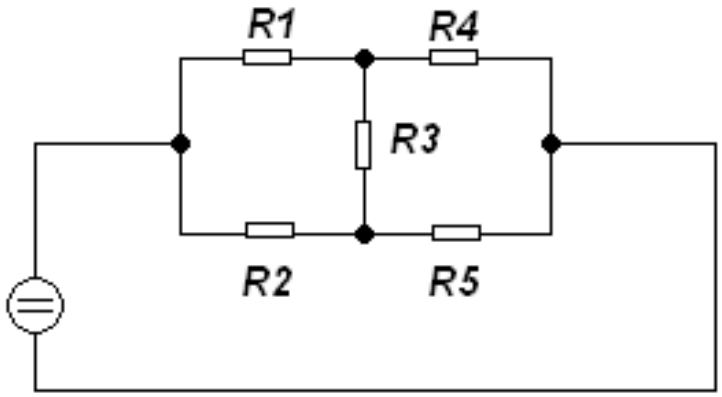
Veľmi zaujímavé metódy riešenia elektrických obvodov sú aj zatiaľ málo analyzované metódy parazitných kapacít a indukčností.

Kapitola 3

Parazitné kondenzátory pri riešení elektrických obvodov

Existuje veľké množstvo klasických postupov ako vyriešiť elektrické obvody zadané schémou. V tejto kapitole sa zameriame na výpočet elektrického obvodu klasickou metódou a následne použitím diferenciálnych rovníc a parazitných kapacít.

Z dôvodu názornosti a prehľadnosti zvolíme jednoduchý obvod, ktorý sa bude skladať len z odporov a zdroja napätia. Dôležité bude zapojenie odporov, ktoré sa bude musieť pri použití klasických metód prepočítavať. Majme elektrický obvod zadaný nasledovnou schémou:

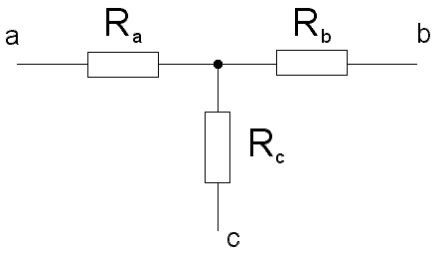


Obr. 3.1: Schéma jednoduchého obvodu

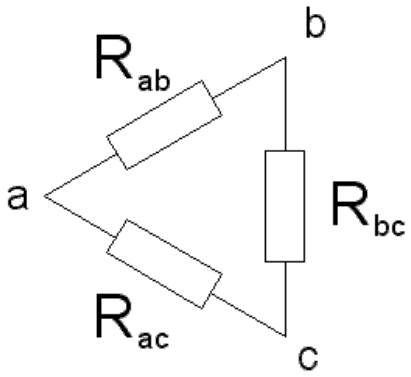
3.1 Klasická metóda riešenia jednoduchého obvodu

3.1.1 Transformácia trojuholník - hviezda

Niekedy sa pri riešení sietí stretneme s odporovým trojpólom tvaru hviezdy (obr. 3.2) alebo trojuholníka (obr. 3.3).



Obr. 3.2: Hviezda



Obr. 3.3: Trojuholník

Tieto trojpóly sú navzájom zameniteľné a často takouto zámenou môžeme docieliť zjednodušenie siete. Označíme si vrcholy trojuholníka a odpovedajúce vrcholy hviezdy písmenami a, b, c. Odpory jednotlivých ramien hviezdy označíme R_a, R_b, R_c podľa príslušnosti ramena k vrcholu. Podobne označíme odpory ramien trojuholníka R_{ab}, R_{bc}, R_{ac}. Skúsme spočítať odpory medzi vrcholmi a, b hviezdy. Tento odpor sa spočíta ako $R_a + R_b$. U trojuholníka je tento odpor rovný paralelnej kombinácii odporu R_{ab} s odporom daným súčtom $R_{ac}+R_{bc}$. Vzhľadom na to, že chceme nahradiť hviezdu trojuholníkom, prípadne trojuholník hviezdou, musí sa odpor medzi vrcholmi a, b u hviezdy rovnať odporu medzi vrcholmi a, b u trojuholníka, čím dostávame prvú rovnicu. Podobne pre vrcholy b, c a c, a dostaneme ďalšie dve rovnice, ktoré môžeme riešiť buď pre neznáme R_a, R_b, R_c alebo pre neznáme R_{ab}, R_{bc}, R_{ac}. Dostaneme tieto vzťahy:

$$R_{ab} = \frac{R_a R_b + R_b R_c + R_c R_a}{R_c} \tag{3.1}$$

a analogické dva, ktoré sa dajú popísať nasledovne: Odpor strany trojuholníka je rovný

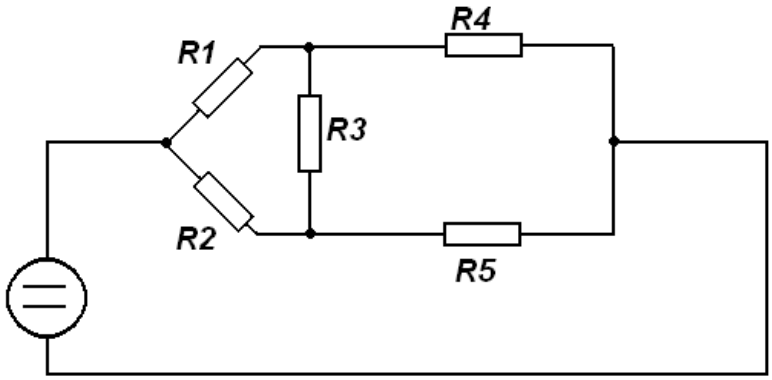
súčtu 3 možných súčinov odporov vetví hviezdy, ktorý sa podelí odporom protiľahlej strany hviezdy. Pre hviezdu dostaneme:

$$R_a = \frac{R_{ab}R_{ca}}{R_{ab} + R_{bc} + R_{ca}} \tag{3.2}$$

3.1.2 Výpočet klasickou metódou

Pri prvom pohľade na obvod z obrázku 3.1 sa nám môže javiť riešenie veľmi jednoduché a priamočiare. Stačí spočítať veľkosti odporov podľa známych vzorcov a vypočítať výsledný odpor celého obvodu. Pozrime sa však na obvod trochu detailnejšie. Problém nám robí odpor (R_3), ktorý je zapojený vertikálne a spája vrchnú vetvu so spodnou. Tento odpor nám bráni v jednoduchom spočítaní celkového odporu.

Pri druhom pohľade si môžeme všimnúť, že zapojenie ľavej časti je vlastne zapojenie do trojuholníka. Toto je názorne vidieť na obrázku 3.4.



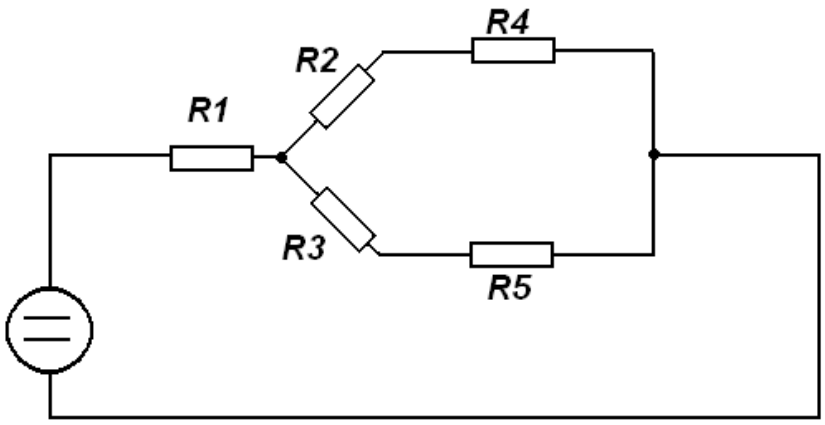
Obr. 3.4: Zapojenie do trojuholníka

Vzhľadom na toto zistenie bude ďalším krokom riešenia transformácia trojuholníkového zapojenia na hviezdu. Podľa vzorca 3.2 získame vzorec na výpočet jednotlivých nových odporov v zapojení do hviezdy (obrázok 3.5). Konkrétny vzorec na výpočet novej hodnoty odporu je:

$$R_{*1} = \frac{R_1 R_2}{R_1 + R_2 + R_3} \tag{3.3}$$

analogicky si môžeme odvodiť vzorce na výpočet ďalších dvoch nových odporov.

Po transformácii sme už dostali schému, v ktorej sa nenachádza vertikálne zapojený odpor ako v schéme pred transformáciou, a teda môžeme začať upravovať známymi vzorcami. Odpory R_2 a R_4 sú zapojené do série, teda ich môžeme spočítať a tým získame odpor $R_{2,4}$. Rovnakým spôsobom spočítame aj sériovo zapojené odpory R_3 a R_5. Získali sme dva paralelne zapojené odpory $R_{2,4}$ a $R_{3,5}$. Podľa vzorca pre spočítanie dvoch paralelne zapojených odporov spočítame predposledný odpor $R_{2,3,4,5}$.



Obr. 3.5: Transformovaný obvod na hviezdu

Vzorec pre sčítanie dvoch paralelne zapojených odporov, vychádza zo vzťahu:

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$$

Znamená to teda, že prevrátená hodnota výsledného odporu sa rovná súčtu prevrátených hodnôt oboch paralelne zapojených odporov. Takže vzorec pre náš výpočet po úprave je:

$$R_{2,3,4,5} = \frac{R_{2,4}R_{3,5}}{R_{2,4} + R_{3,5}}$$

Po tomto kroku nám v obvode zostali už len dva, do série zapojené, odpory $R_{2,3,4,5}$ a R_1. Keďže sú v sérii, tak ich môžeme jednoducho spočítať a tým získame celkový odpor obvodu. Výpočet prúdu, ktorý preteká obvodom je potom jednoduchý. Požijeme Ohmov zákon (3.4) a pomocou celkového odporu a veľkosti napätia vypočítame tečúci prúd.

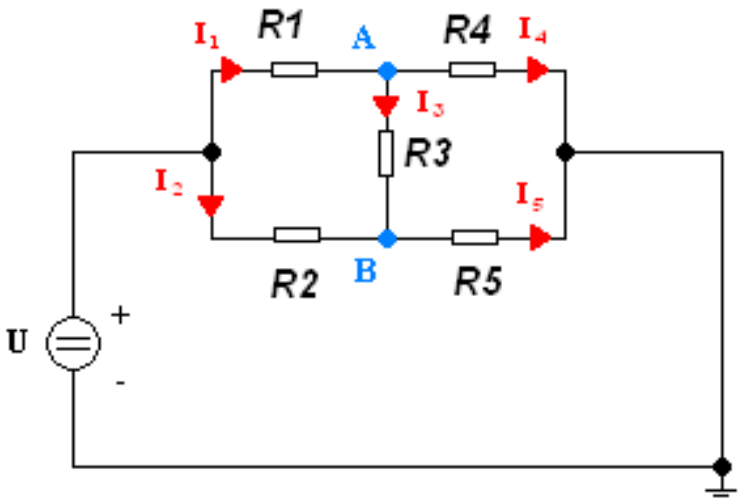
$$I = \frac{U}{R} \tag{3.4}$$

3.2 Použitie parazitných kondenzátorov

Ukázali sme si postup riešenia pomocou klasických metód. Ten istý obvod teraz vyriešime pomocou diferenciálnych rovníc a parazitných kondenzátorov.

Obvod je rovnaký ako v predchádzajúcom prípade. Je neupravený transformáciou na hviezdu, teda stále máme zapojený vertikálny odpor R_3. Výhoda použitia parazitných kondenzátorov je, že nepotrebujeme upravovať obvod transformáciou. Potrebujeme len nájsť uzol, v ktorom by sme chceli vedieť napätie. Presnejšie je to uzol, ktorého napätie nám pomôže vypočítať všetky hodnoty obvodu, ktoré chceme.

V našom obvode sme si vybrali uzly (obrázok 3.6), ktorými je pripojený práve ten odpor, ktorý nám bráni jednoduchému riešeniu, teda R_3. Samozrejme je možné zvoliť všetky uzly v obvode. Ale nie je treba počítať uzly, ktoré nakoniec nebudeme potrebovať k výpočtu. Preto je dôležité sa zamyslieť a vybrať vhodne.



Obr. 3.6: Uzly

Pri znalosti napätí v uzloch A a B sa dá použiť metóda uzlových napätí, ktorou sa nám podarí vypočítať prúdy tečúce jednotlivými vetvami. Metóda vychádza z 1. Kirchhoffovho zákona, ktorý znie: Súčet všetkých prúdov do uzla vstupujúcich sa rovná súčtu prúdov z uzla vystupujúcich. Konkrétne využitie metódy uzlových napätí je vidieť na rovniciach pre výpočet prúdov v našom obvode:

$$I_1 = \frac{U - U_A}{R_1}$$

$$I_2 = \frac{U - U_B}{R_2}$$

$$I_3 = \frac{U_A - U_B}{R_3}$$

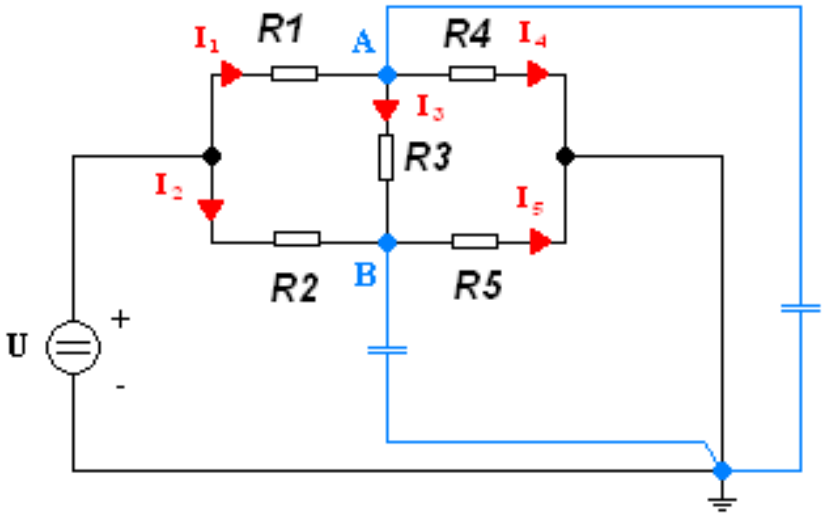
$$I_4 = \frac{U_A - 0}{R_4}$$

$$I_5 = \frac{U_B - 0}{R_5}$$

Ako je vidieť, tak prúd v konkrétnej vetve sa vypočíta z rozdielu napätia v uzle, z ktorého vyteká a napätia v uzle, do ktorého následne vteká. A tento rozdiel sa podelí odporom v konkrétnej vetve. V posledných dvoch rovniciach vidíme, že napätie uzla, do ktorého prúd vteká má pravdepodobne hodnotu 0. Je to spôsobené tým, že obvod je uzemnený a tieto dva prúdy vtekajú do tejto uzemnenej časti, ktorá ma nulové napätie.

Ukázali sme si, že pri znalosti napätia na uzloch A a B sme schopní vypočítať prúdy vo všetkých vetvách. Pre výpočet týchto napätí použijeme parazitné kondenzátory.

Parazitné kondenzátory sa pripoja na nami zvolené uzly a uzemnia sa (obrázok 3.7). Pri zopnutí obvodu sa parazitné kondenzátory začnú nabíjať a po určitom čase sa kondenzátory nabijú a prúd vo vetve kondenzátora prestane tiecť. Tento stav nazývame ustálený. Napätie na kondenzátore v tomto stave je napätie, ktoré potrebujeme pri ďalšom výpočte.



Obr. 3.7: Parazitné kondenzátory

Pre výpočet napätí na uzloch si vytvoríme diferenciálne rovnice. Diferenciálna rovnica pre výpočet napätia na kondenzátore je:

$$U' = \frac{1}{C} I$$

V našom prípade budú rovnice pre výpočet vyzerať takto:

$$U'_A = \frac{1}{C} I_A$$

$$U'_B = \frac{1}{C} I_B$$

Prúdy IA a IB sú prúdy tečúce vo vetvách s pridanými kondenzátormi, čo je vidieť na obrázku 3.7. Pre ich výpočet použijeme 1. Kirchhoffov zákon. Rovnice budú vyzerať nasledovne:

$$I_1 - I_3 - I_4 = I_A$$

$$I_2 + I_3 - I_5 = I_B$$

V tejto fáze už máme všetky potrebné informácie a všetky rovnice sú zostavené. Stačí určiť hodnoty odporov a napätie na zdroji a s týmito veličinami vyriešiť vzniknuté diferenciálne rovnice. Pre tento účel použijeme aplikáciu TKSL.[5]

3.3 Príklad v TKSL

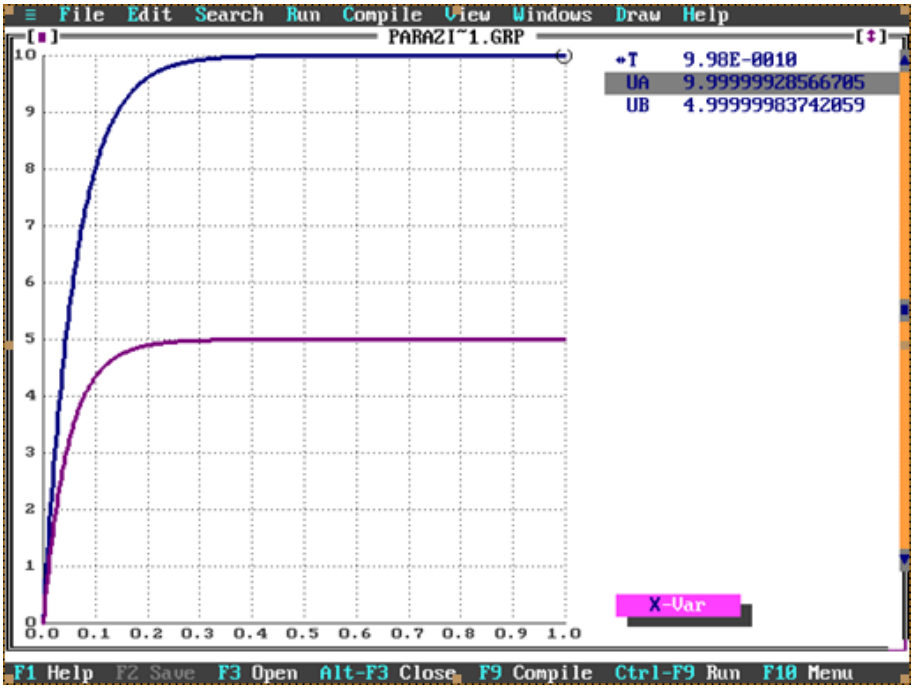
V predchádzajúcej časti sme si odvodili rovnice pre výpočet. V tejto časti si ukážeme, či boli naše teórie pravdivé, a to tak, že odsimulujeme priebeh nami vytvorených diferenciálnych rovníc v systéme TKSL. Podľa výstupných grafov sa ukáže, či kondenzátory pracujú ako sme predpokladali.

Zvoľme teda hodnoty odporov a to tak, že $R_1 = 200\Omega, R_2 = 300\Omega, R_3 = 500\Omega, R_4 = 100\Omega, R_5 = 50\Omega$ a vstupné napätie bude $U = 32V$.

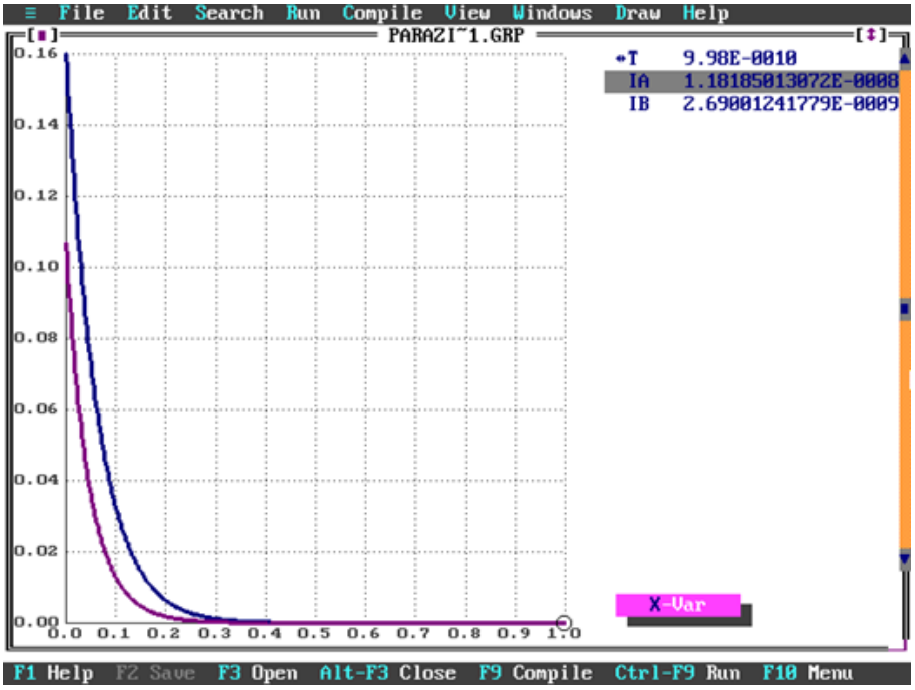
Potom po dosadení do vzorcov v časti 3.2 získame tento zdrojový kód pre systém TKSL:

```
var i1, i2, i3, i4, i5, iA, iB, uA, uB;
const
    u = 32, R1 = 200, R2 = 300, R3 = 500, R4 = 100, R5 = 50, C = 1e-12,
    eps=1e-20, Tmax=1e-9, dt=1e-12;
system
    iA=i1-i3-i4;
    iB=i2+i3-i5;
    uA'=1/C * iA &0;
    uB'=1/C * iB &0;
    i1= 1/R1 * (u-uA);
    i2= 1/R2 * (u-uB);
    i3= 1/R3 * (uA-uB);
    i4= 1/R4 * (uA-0);
    i5= 1/R5 * (uB-0);
sysend.
```

Po spustení simulácie nám TKSL vykreslí graf, v ktorom názorne uvidíme, ako sa jednotlivé hodnoty menili v čase. Na obrázku 3.8 je znázornený priebeh napätí na kondenzátoroch. Je vidieť, že napätia na kondenzátoroch sa na začiatku postupne nabíjali a po uplynutí určitého času sa ustálili na stabilných hodnotách.



Obr. 3.8: Priebeh napätí na kondenzátoroch

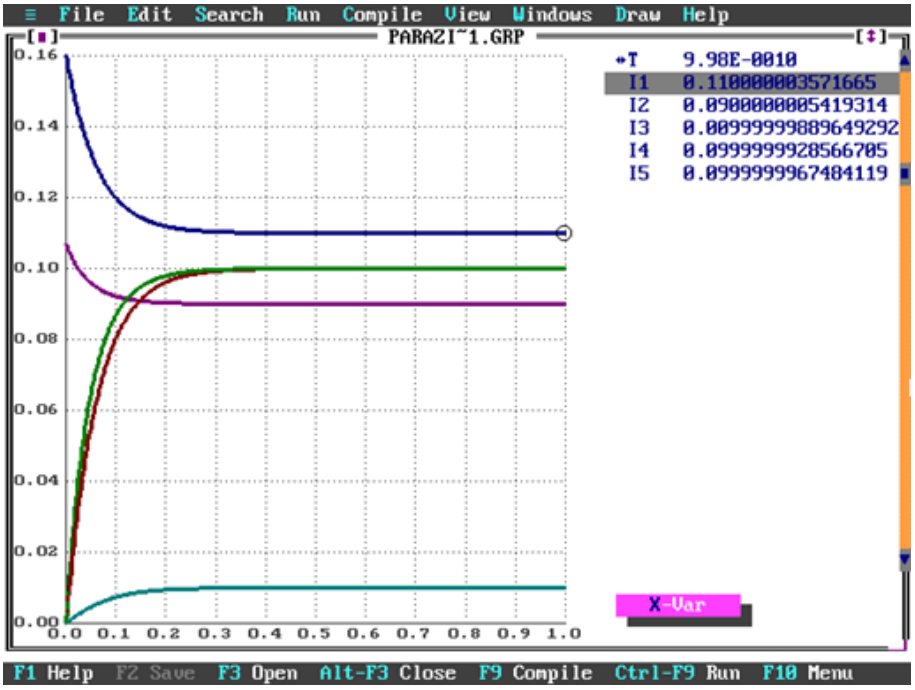


Obr. 3.9: Priebeh prúdov vo vetvách kondenzátorov

Na obrázku 3.9 je znázornený priebeh hodnôt prúdov vo vetvách kondenzátorov. Podľa simulácie je vidieť, že vo vetvách tiekol prúd, kým sa kondenzátory nenabili a neustálil sa stav. Po tomto okamihu sa prúdy minimalizovali a vetvami prestali prúdy tiecť.

Posledný obrázok 3.10 zobrazuje priebeh prúdov v obvode, na ktorom máme znázornené jednotlivé hodnoty prúdov. V čase, kedy nastal ustálený stav, sú vypočítané hodnoty v jednotlivých vetvách.

Z grafu sa dajú prečítať hodnoty vypočítaných uzlov v jednotlivých vetvách. Zvládli sme to, čo sme v časti 3.1 robili klasickými metódami a transformáciami, použitím diferenciálnych rovníc a parazitných kondenzátorov. Táto metóda je veľmi presná a rýchla a poskytuje nám možnosť riešiť aj zložitejšie schémy elegantnou cestou diferenciálnych rovníc.[5]



Obr. 3.10: Prúdy v obvode

3.4 Stiff systémy

Použitie parazitných kondenzátorov má aj svoje problémy. Ak chceme pripojiť k obvodu parazitný kondenzátor, tak by mal byť dostatočne malý, teda by mal mať oproti prvkom v obvode o niekoľko rádov menšiu hodnotu kapacity. Dôvod je, aby minimálne zmenil chovanie obvodu a tento kondenzátor sa dal zanedbať. Toto je problém pri riešení takej diferenciálnej rovnice, v ktorej sú nelineárne prvky, ktorých parametre sa líšia až v niekoľkých rádoch.

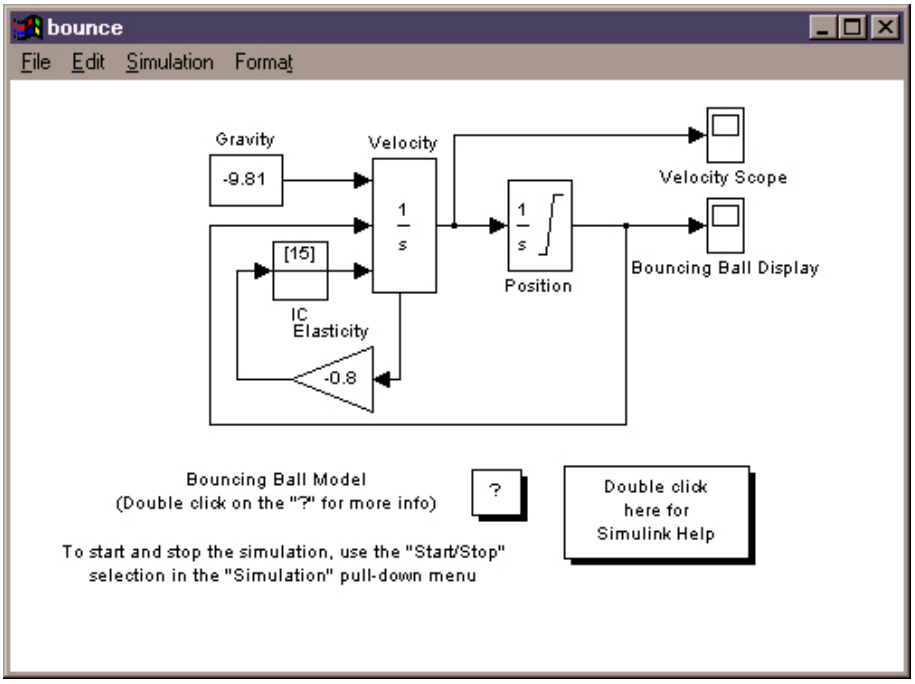
Takéto systémy diferenciálnych rovníc potrebujú na vyriešenie prvkov s nízkou hodnotou veľmi malý krok pri riešení pomocou Taylorovho radu. Na druhej strane sú tam prvky s vysokou hodnotou, ktoré budú mať dlhý simulačný čas. Takže v konečnom dôsledku bude simulácia trvať veľmi dlhú dobu. Také systémy nazývame STIFF systémy.[5]

Kapitola 4

Modelovanie (zobrazovanie) elektrických obvodov

4.1 Matlab/Simulink

Simulink je program pre simuláciu a modelovanie dynamických systémov, ktorý využíva algoritmy Matlabu pre numerické riešenia nelineárnych diferenciálnych rovníc. Poskytuje užívateľovi možnosť rýchlo a ľahko vytvárať modely dynamických sústav vo forme blokových schém a rovníc.



Obr. 4.1: Model padajúcej loptičky v Simulinku

Nový prístup k riešeniu diferenciálnych rovníc dovoľuje simulovať aj rozsiahle "stiff" systémy rýchlo, presne a s efektívnym využitím pamäte počítača. Pomocou Simulinku a jeho grafického editora je možné vytvárať modely lineárnych, nelineárnych, v čase diskrétnych

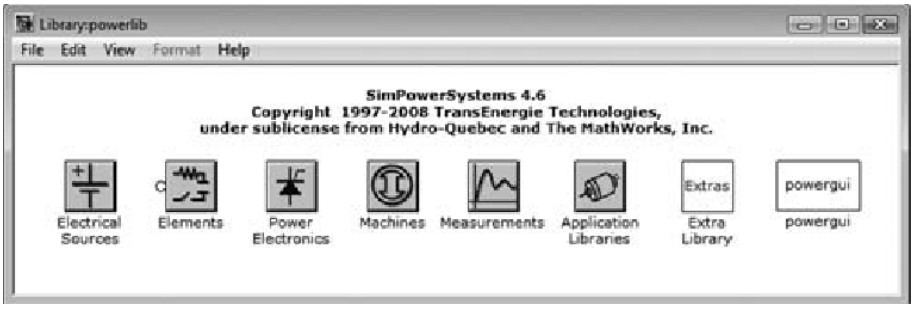
alebo spojitých systémov jednoduchým presúvaním funkčných blokov myšou. Aktuálna verzia Simulinku umožňuje spúšťať určité časti simulovanej schémy na základe výsledku logickej podmienky. Tieto spúšťané a povoľované subsystémy umožňujú použitie programu v náročných simulačných experimentoch.

Samozrejmosťou je otvorená architektúra, ktorá umožňuje užívateľovi vytvárať si vlastné funkčné bloky a rozširovať už tak bohatú knižnicu Simulink. Hierarchická štruktúra modelov umožňuje navrhnúť aj veľmi zložité systémy do prehľadnej sústavy subsystémov prakticky bez obmedzenia počtu blokov. Simulink, rovnako ako Matlab, dovoľuje pripájať funkcie napísané užívateľom v jazyku C. Grafické možnosti Simulinku je tiež možné priamo využiť k tvorbe dokumentácie. Medzi ďalšie vlastnosti Simulinku patrí nezávislosť užívateľského rozhrania na počítačovej platforme. Možnosť prenosu modelov a diagramov medzi rôznymi typmi počítačov umožňuje vytvárať rozsiahle modely, ktoré si vyžadujú spoluprácu väčšieho kolektívu riešiteľov na rôznych úrovniach. Na obrázku 4.1 je vidieť prostredie Simulink.[16]

4.2 Matlab/SimPowerSystems

SimPowerSystem patrí do rodiny produktov, ktoré rozširujú možnosti Matlabu v oblasti modelovania fyzikálnych sústav. Ide o takzvaný blockset, teda nadstavbu určenú pre Simulink. Po nainštalovaní niektorého z blocksetov pribudne v zozname knižníc Simulinku položka obsahujúca bloky s daným zameraním. SymPowerSystem sa zameriava na modelovanie energetických sústav. Nájdeme tu bloky určené na výrobu, úpravu, prenos a spotrebu elektrickej energie. Pomocou niekoľkých blokov môžeme sledovať vybrané stavové veličiny, a tak ich previazať so Simulinkom. Bez problémov tak môžeme navrhovať nadradené riadiace orgány aj pre také komplikované obvody, akými je napr. elektrizačná sústava. Na úspešnú inštaláciu tohto rozšírenia treba mať nainštalovaný Matlab a Simulink. Cena tejto nadstavby sa pohybuje okolo 3 700 eur na komerčné využitie, resp. 780 eur pre školy.

Po nainštalovaní SimPowerSystem pribudne v Simulinku knižnica powerlib. V nej nájdeme potrebné stavebné prvky - bloky vhodné na modelovanie energetických sústav. V aktuálnej verzii 4.6 sú už bloky radené hierarchicky v stromovej štruktúre, aby bola zabezpečená prehľadnosť a ľahšia orientácia.



Obr. 4.2: Knižnica Powerlib

K blokom knižnice SimPowerSystem sa môžeme dopracovať listovaním v knižniciach Simulinku alebo zadaním príkazu powerlib do príkazového riadku Matlabu. Knižnica powerlib sa skladá z týchto podknižníc:

- Electrical Sources Library - bloky generujúce elektrické signály,

- Elements Library - lineárne a nelineárne obvodové prvky,
- PowerElectronics Library - výkonové spínacie prvky,
- Machines Library - motory a generátory,
- Measurements Library - meracie bloky prúdu a napätia,
- Electric Drives Library - striedavé a jednosmerné elektrické pohony,
- Flexible AC Transmission Systems (FACTS) Library - pružné prenosové systémy,
- Distributed Resources library - modely veterných turbín,
- Extras Library - špeciálne riadiace a meracie bloky,
- Demos Library - demo modely,
- Obsolete Blocks - zastarané bloky pre spätnú kompatibilitu.

Knižnica obsahuje väčšinu bežne používaných blokov. Môže však nastať situácia, že potrebujeme blok, takpovediac, na mieru. Nové bloky sa dajú naprogramovať a zaradiť k existujúcim.

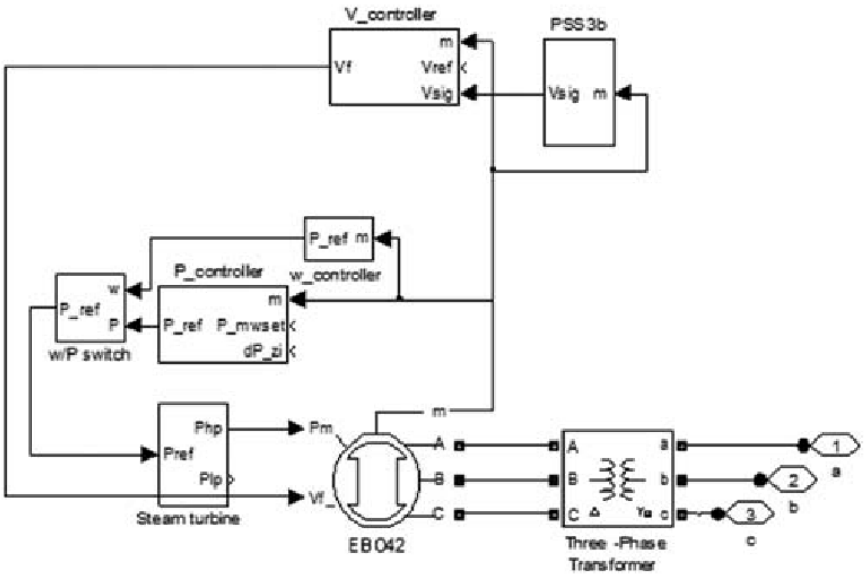
Simulačná schéma, ktorá predstavuje elektrický obvod, sa modeluje podobne ako v Simulinku. Bloky sa metódou drag&drop prenesú na pracovnú plochu a následne sa medzi nimi vytvoria väzby. Zásadným rozdielom oproti klasickým schémam je, že signály nemajú smer. Rešpektuje sa tým reálna situácia, kde vopred nevieme, ktorým smerom budú energie (signály) prúdiť. Preto nemožno prepájať bloky z knižníc zo Simulinku s blokmi zo SimPowerSystems. Komunikácia je možná len cez špeciálne bloky, ktoré rešpektujú smer signálov.

Potom, ako je namodelovaný elektrický obvod, a ako sú naparametrizované jednotlivé bloky, môžeme začať simuláciu ako v každom inom modeli v prostredí Simulink. Vždy, keď sa má simulácia spustiť, SimPowerSystems zavolá inicializačný mechanizmus. Tento inicializačný proces vypočíta ekvivalentný model v stavovom priestore z elektrického obvodu a zostaví ekvivalentný systém, ktorý môže byť simulovaný v Simulinku. Všetky tieto úkony sa dejú na pozadí a používateľ do nich nemusí zasahovať. Dôležitým blokom pri simuláciách elektrizačných sústav a elektrických obvodov je blok powergui. Pomocou tohto bloku môže simulácia bežať spojite, diskrétne alebo vo fázorovom priestore. Dôležité je podotknúť, že SimPowerSystems simuluje pri všetkých spomínaných typoch simulácií trojfázovo, čiže je pre nás jednoduché namodelovať asymetrické prvky a poruchy. Blok powergui ďalej umožňuje:

- Zobrazovať ustálené stavy napätí a prúdov elektrického obvodu, ako aj všetkých stavových premenných obvodu.
- Meniť začiatočné podmienky, napr. veľkosť napätia na kondenzátore a prúdu cez induktor.
- Inicializovať trojfázové siete, zahŕňajúc generátory, takže simulácia sa môže začať z ustáleného stavu.
- Zobrazovať graf impedancie proti frekvencii, keď je v sieti zapojený blok merania impedancie.

- Vykonávať FFT analýzu zo simulačných výsledkov.
- Generovať stavový priestor z elektrického obvodu.
- Počítať parametre vedení z parametrov stožiara a vodiča.

Elektrické prvky SimPowerSystems sa spájajú pomocou tzv. elektrických svoriek. Ide hlavne o prvky, napr. transformátor, vedenie, záťaž. V knižnici powerlib sa však nachádzajú aj bloky s klasickými svorkami vstup a výstup rovnako ako v prostredí Simulink. Tieto bloky sa uplatňujú hlavne pri meraní a riadení elektrických strojov, či už sú nimi generátory alebo motory. Toto riadenie môžeme modelovať aj pomocou štandardných blokov prostredia Simulink. Príkladom je model elektrárenského bloku EBO42 uvedený na obr. 4.3.



Obr. 4.3: Model elektrárenského bloku EBO42

Model bloku podľa tohto obrázka sa skladá z modelu synchrónneho generátora EBO42, budiaceho systému `V_controller`, parnej turbíny `Steam_turbine`, regulátora činného výkonu `P_controller`, regulátora otáčok `w_controller` pri ostrovnej prevádzke, zo systémového stabilizátora PSS3b a z blokového transformátora. Riadenie teda môže byť realizované vyššími formami riadenia, ako nám dovoľujú štandardné bloky SimPowerSystems.

Modelovací nástroj SimPowerSystems predstavuje nadstavbu, kde sa dajú riešiť jednak problémy spojené s nastavovaním lokálnych regulátorov alebo stabilizačných systémov. Nemusíme sa však zastaviť pri modelovaní na tejto úrovni. Dnes je už softvér v takom pokročilom štádiu vývoja, že sa pomocou neho dajú modelovať aj komplexné systémy, napr. určitá oblasť, alebo ostrov elektrizačnej sústavy. Ak by sme abstrahovali niektoré detaily, tak sa dajú vytvárať aj modely celých elektrizačných sústav. So zvyšujúcou sa podrobnosťou a zaraďovaním ďalších, aj menších zdrojov, prípadne vedení na nižších napäťových úrovniach narastá aj výpočtová náročnosť. Aktuálne je k dispozícii model elektrizačnej sústavy SR v tejto konfigurácii:

- Synchrónne generátory:
  - Jadrová elektráreň Mochovce: EMO: EMO11, EMO12, EMO21, EMO22

  - Jadrová elektráreň Jaslovské Bohunice EBO2: EBO21, EBO22
  - Jadrová elektráreň Jaslovské Bohunice EBO3: EBO31, EBO32
  - Jadrová elektráreň Jaslovské Bohunice EBO4: EBO41, EBO42
  - Uhoľná elektráreň Vojany EVO1: EVO12, EVO16
  - Uhoľná elektráreň Vojany EVO2: EVO21, EVO22, EVO23, EVO24
  - Uhoľná elektráreň Nováky ENO: ENOB1, ENOB2
  - Vodná elektráreň Liptovská Mara LMAR: LMAR1, LMAR2, LMAR3, LMAR4
  - Vodná elektráreň Gabčíkovo EGABC: EGABC1, EGABC2, EGABC3, EGABC4, EGABC5, EGABC6
  - Prečerpávacia vodná elektráreň ECV: ECV11, ECV12
  - Paroplynové centrum Bratislava: PPCBA, PPCBB

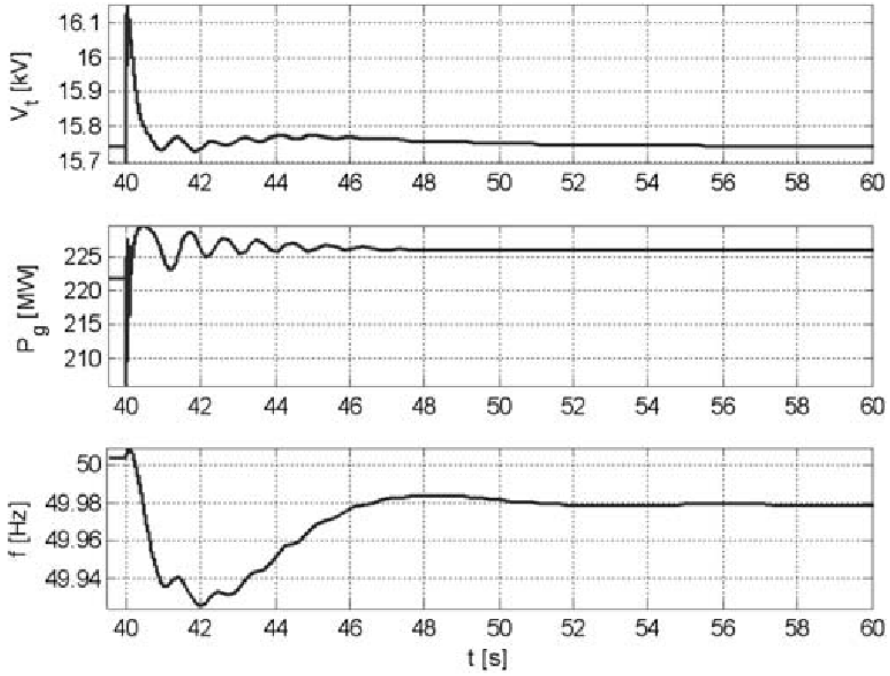
- Uzly:
  - 400 kV úroveň:
    * Domáce: Stupava, Podunajské Biskupice, Gabčíkovo, Križovany, Bošáca, Varín, Sučany, Spišská Nová Ves, Lemešany, Moldava, Rimavská Sobota, Levice, Horná Ždaňa, Liptovská Mara, Veľké Kapušany, Veľký Ďur
    * Zahraničné: Sokolnice (Cz), Nošovice (Cz), Göd (Hu), Gyor (Hu), Mukačevo (U), Krosno-Iskrzynia (Pl)
  - 200 kV úroveň:
    * Domáce: Senica, Križovany, Bystričany, Sučany, Lemešany, Veľké Kapušany
    * Zahraničné: Sokolnice (Cz)

- Prenosové vedenia:
  - 400 kV úroveň:
    * V043, V044, V046, V047, V497, V498, V448, V429, V439, V424, V425, V490, V491, V492, V493, V449, V426, V427, V428, V440, V477, V495, V496, V404, V405, V406, V407, V408
  - 200 kV úroveň:
    * V280, V283, V075, V274, V271, V072

Na takto vytvorenom modeli sa dajú realizovať rôzne experimenty a poruchy, napr. skrat na vedení či výpadok zdroja. Pomocou simulácie tak vidíme dosahy týchto porúch. Model možno, samozrejme, využiť aj na expertízy pri zisťovaní dosahov zaradenia nových zdrojov, vedení alebo rozvodní. Tiež sa dá využiť pri optimalizácii nastavenia určitých regulačných slučiek na konkrétnom generátore.

Modelovanie v prostredí SimPowerSystems nám umožňuje simulovať veľmi zložité deje, akým je aj prechod do ostrovnej prevádzky. Rôzne typy porúch môžu ovplyvniť prevádzku ESSR v UCTE natoľko, že fázový uhol zahraničných elektrizačných sústav bude rozdielny oproti fázovému uhlu v našej sústave. V takomto prípade dôjde k nedobrovoľnému odpojeniu ESSR od UCTE. Rozpojené sú potom vedenia v497, v424 a v280 vyvedené z rozvodne Sokolnice v ČR, vedenie v404 vyvedené z rozvodne Nošovice v ČR, zdvojené vedenie v477, v478 vyvedené z rozvodne Krosno-Iskrzynia v Poľsku, vedenie v440 vyvedené z rozvodne

Mukačevo na Ukrajine, v449 vyvedené z rozvodne Göd v Maďarskej republike a vedenie v448 vyvedené z rozvodne Gyor v Maďarskej republike. Pri automatickom prechode do ostrovnej prevádzky dochádza k veľkým osciláciám elektrických a mechanických veličín. Výsledky tohto experimentu opisuje obr. 4.4. V 40 s dôjde k odopnutiu cezhraničných vedení. Najväčší pokles frekvencie vidieť po 2 s od tohto odopnutia. Pokles nespôsobí aktiváciu frekvenčného odľahčovania. Oscilácie sú utlmené asi za 8 s, pri poklese frekvencie oproti pôvodnej hodnote o 20 mHz



Obr. 4.4: Prechod do ostrovnej prevádzky sledovaný na EMO11

Predkladané výsledky a model zložitého elektrického systémy dokazujú, že SimPowerSystems je skutočne plnohodnotná nadstavba pre Matlab/Simulink v oblasti energetiky. V aktuálnej verzii vieme riešiť modelovanie jednoduchých elektrických obvodov, ale aj pomerne komplexných systémov. Výraznou výhodou je prepojiteľnosť so Simulinkom a následne s Matlabom. Tento fakt otvára dvere dorábaniu vlastných blokov, prípadne tvorbe vlastných riadiacich algoritmov. Tiež je k dispozícii výpočtový aparát Matlabu na spracovanie dát. Vieme teda efektívne zhodnotiť kvalitu riadenia, prípadne definovať nežiaduce stavy. Modely vytvorené v SimPowerSystems však neustále vyžadujú veľký výpočtový výkon. Ide o komplikované modely, ktoré vyťažujú procesor na maximum. V oblasti optimalizácie výkonu sa neustále pracuje. Jedným z krokov je aktívne zapojenie akcelerátora a kompilátora počas simulácií. Ďalším krokom by, pravdepodobne, malo byť plnohodnotné vyťaženie viacjadrových systémov.[3]

4.3 Modelica/Dymola

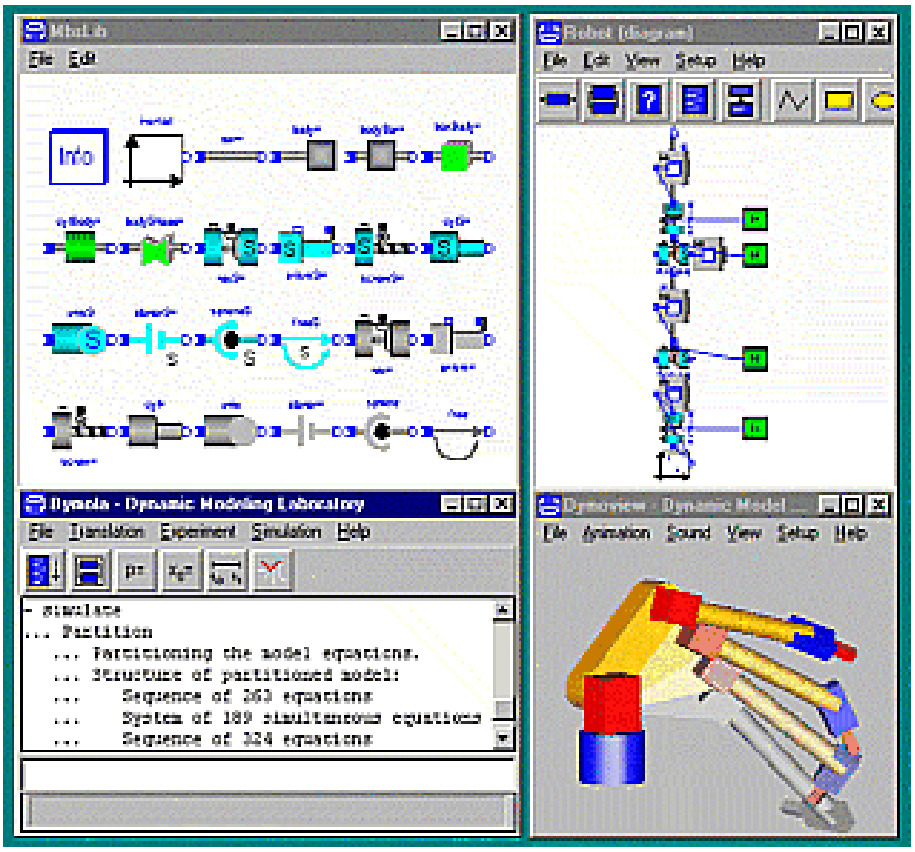
Dymola bola navrhnutá Hildingom Elmqvistom na Technickej univerzite v Lund (LTH) vo Švédsku ako súčasť jeho dizertačnej práce PhD.

Dymola je modelovacie prostredie, ktoré umožňuje užívateľovi namapovať hardvérové komponenty z fyzikálnych systémov priamo na zodpovedajúce softvérové komponenty.

Elmqvist zistil, že nesmú existovať žiadne rozdiely medzi vstupnými a výstupnými parametrami týchto objektov. Smer toku informácií môže byť určený len po zostavení elektrického modelu a je založený na zapojení tohto modelu komponent do celkového modelu.[2]

Dymola je objektovo-orientovaný jazyk a program určený na modelovanie rozsiahlych systémov. Znovupoužitie namodelovaných štruktúr a komponent je podporená využívaním knižnice obsahujúcej model tried a podporujúcej dedičnosť. Užívateľ nemusí ručne prevádzať rovnice vzťahujúce sa k modelu. To sa robí v Dymole symbolicky. Matice rovníc uľahčujú pohodlné modelovanie 3-D mechanických, riadiacich systémov. Dymola automaticky generuje potrebný čas a stavové udalosti pri nelineárnych rovniciach a pod.

Dymola umožňuje preložiť fyzikálny model na S-funkcie, M-file, alebo MEX-súbory, ktoré možno použiť ako blok v Simulink. Algebraické slučky sa automaticky riešia.[8]



Obr. 4.5: Dymola - ukážka modelovacieho prostredia

Dymola je postavená nad jazykom Modelica. Má svoje grafické užívateľské rozhranie a knižnice prvkov, ktoré sa dajú ďalej rozširovať. Model navrhnutý v tomto prostredí sa pred simuláciou prekladá do jazyka C. Výstup je kompatibilný so systémom MatLab.

Hlavné črty systému:

- Hierarchia funkčných komponent
- Skladanie modelov z komponent
- Jednoduché rozšírenie knižnice komponent
- Preklad modelu do jazyka C
- Kompatibilný výstup so systémom Matlab
- Symbolické riešenie niektorých rovníc
- Objektovo-orientovaný jazyk
- Umožňuje real-time hardware-in-the-loop

Simulačný jazyk Modelica sa vyvíja od roku 1996. Vznikol oddelením od Dymoly. Je pod ním podpísaná nezisková organizácia Modelica Association. Je schopný popísať model diferenciálnymi, algebraickými a aj diskrétnymi rovnicami. Môže kontrolovať fyzikálne jednotky ak je to potrebné.[15]

Celkový model systému je popísaný textovo a preto je možné ho editovať aj mimo grafického prostredia. Tento kód sa následne prekladá do jazyka C.

Dymola je momentálne komerčne najviac používaný systém na modelovanie fyzikálnych sústav.

Kapitola 5

Návrh a analýza systému TKSL PowerGear

5.1 Analýza požiadaviek

Pri analýze požiadaviek na systém PowerGear som sa inšpiroval profesionálnymi simulačnými nástrojmi, ktoré sú podrobne popísané v kapitole 4.

Rozhodol som sa vytvoriť systém, ktorý bude využívať pozitívne črty už existujúcich systémov. Vďaka tomu nebudem porovnávať funkčnosť nového systému so štandardmi, ale iba zdôrazním funkcie, ktoré sú prevzaté, prípadne funkcie, ktoré rozširujú funkčnosť v rámci štandardov.

Hlavné požiadavky na navrhovaný systém sú:

- Prívetivé grafické užívateľské rozhranie.
- Modelovanie elektrických obvodov jednoduchým klikaním myši.
- Komponenty modelu realizované pomocou funkčných blokov.
- Funkčné bloky[1] s variabilným počtom vstupov a výstupov.
- Hierarchické skladanie funkčných blokov.
- Knižnica blokov oddelená od samotného programu.
- Čitateľný popis blokov a aj uloženej schémy.
- Rozšíriteľnosť.
- Automatické doplnenie parazitných kapacít.

Užívateľsky prívetivé GUI a modelovanie rozsiahlych obvodov len za pomoci myši je pravdepodobne základ použiteľného editora. Druhý ciel je neobmedzovať užívateľa vstavanými prostriedkami bez možnosti bezprostrednej zmeny. Tento problém je vyriešený čitateľným popisom jednotlivých blokov, ktoré môže užívateľ bez väčších problémov editovať a tým pozmeniť funkciu bloku.

[1] Blokom rozumiem komponentu modelovaného systému, ktorá má určité vstupy a výstup a v rámci systému svoju funkciu.

Knižnica blokov je pri modelovaní rovnako dôležitá, ak nie dôležitejšia, ako samotná možnosť kreslenia modelov na plátno editora. Knižnica preto musí byť koncipovaná dostatočne jednoducho, aby bola jej štruktúra jasná aj menej znalému užívateľovi. Jednotlivé funkčné bloky v knižnici musia mať obecnú štruktúru, ktorá vyjadruje ich vizualizáciu a hlavne správanie v systéme. Pri koncepte funkčných blokov máme možnosť vytvárať nové bloky aj z blokov už existujúcich, čím vytvárame hierarchiu, ktorá zjednodušuje zápis a pridáva na prehľadnosti. V neposlednom rade nám tento koncept dovoľuje využiť simulačný nástroj na širšie spektrum, než len na modelovanie elektrických obvodov. Tým sa systém stáva univerzálnejším a teda rozšíriteľným.

Doplňujúce funkcie, ktoré uľahčia užívateľovi prácu sú:

- Približovanie a odďaľovanie modelovaného systému.
- Možnosť viacerých otvorených projektov na záložkách.
- Jednoduché a intuitívne vytvorenie nového bloku z už existujúcich.

5.2 Návrh

5.2.1 Funkčné bloky

Jednou z podmienok pre simulačný systém je jeho rozšíriteľnosť a obecnosť modelovania. Preto nie je žiaduce, aby systém obsahoval konkrétne stavebné bloky, ako napríklad cievku alebo kondenzátor. Takéto komponenty musia byť samotnému systému transparentné. Preto každá komponenta v modelovanom systéme bude mať obecnú štruktúru, z ktorej bude jasná jej funkcia, vizualizácia aj vnútorný stav. Takáto komponenta sa bude nazývať funkčný blok.

Funkčné bloky budú uložené vo formáte XML. Formát XML je momentálne veľmi rozšírený a preto nie je pre užívateľa veľkou prekážkou. Už podstata XML je v tom, aby bol prehľadný a užívateľsky čitateľný. Týmto je elegantne vyriešený problém s ukladaním informácií, aby boli čitateľné a editovateľné bežným užívateľom.

Štruktúra funkčného bloku

XML súbor funkčného bloku bude obsahovať osem povinných sekcií a jednu nepovinnú. Medzi povinné sekcie patria štyri vizualizačné sekcie:

- Sekcia čiar.
- Sekcia kriviek.
- Sekcia kružníc.
- Sekcia textov.

Vizualizačné sekcie popisujú, ako bude daný blok vykreslený pri modelovaní na plátno. Je zrejmé, že každá sekcia bude obsahovať niekoľko záznamov, ktoré budú definovať jednotlivé grafické elementy. Všetky tieto elementy sa vykreslia a vo výsledku zobrazia nami požadovaný tvar.

Asi nie je nutné zdôrazňovať, že v sekcii čiar budú elementy čiar, ktoré budú definované začiatočným a konečným bodom. Obdobne to bude pre sekcie kriviek a kružníc. Sekcia textov bude mať len bod začiatku textu, text samotný a jeho veľkosť. Vďaka čiaram, kružniciam a krivkám, by mal byť užívateľ schopný nadefinovať komponentu tak, aby sa zobrazovala podľa jeho predstáv. Texty sú na doplnenie zložitých komponent, ktoré budú vyžadovať interný popis niektorých svojich častí.

Ďalej sa v štruktúre nachádzajú tri sekcie vnútorného stavu:

- Sekcia vstupov.
- Sekcia výstupov.
- Sekcia premenných.

Definovať sekciu vstupov a aj výstupov nemá pre niektoré komponenty význam, ale z obecného hľadiska je nutné zaviesť rozdielny popis pre vstupy a výstupy. Obe sekcie teda budú charakterizovať miesta komponenty, na ktoré sa pripoja vodiče (v prípade iného modelu sa môžu tieto spoje nazývať rôzne). Taktiež ponesú informáciu o názve tohto prípojného miesta.

Sekcia premenných bude charakterizovať aktuálny vnútorný stav komponenty. Počet premenných (rovnako ako všetkého ostatného) je variabilný a teda každá komponenta môže mať niekoľko vnútorných premenných alebo aj žiadnu.

Posledná povinná sekcia je sekcia funkcie, v ktorej bude zaznamenaný popis bloku pomocou sústav alebo jednej diferenciálnej rovnice.

Nepovinná sekcia je potrebná k hierarchickému členeniu blokov. Táto sekcia bude obsahovať blokové schéma, z ktorého je nový blok zložený. Aj tento nový blok môže samozrejme obsahovať popis sústavou diferenciálnych rovníc, ak tam túto sústavu vložíme. Taktiež môžeme rovnako nadefinovať premenné.

5.2.2 Knižnica

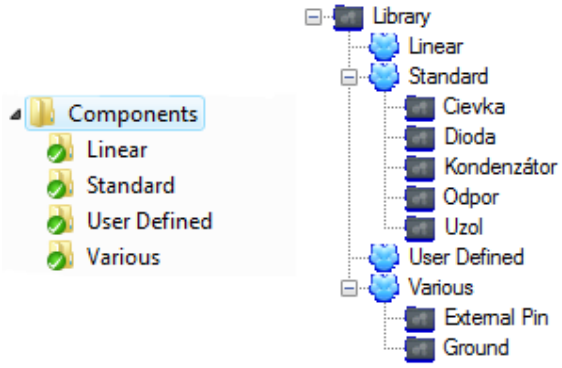
Z podkapitoly 5.2.1 je jasné, že jednotlivé bloky, ktoré budú viditeľné v knižnici simulátoru, sú na disku uložené vo forme XML súborov, čím podtrhávajú užívateľskú prívetivosť. Rovnakým smerom sa uberá aj koncept knižnice, ktorá je maximálne odtienená od samotného programu.

Užívatelia sú zvyknutý na adresárové štruktúry vo Windows, preto je pre nich viac než príjemné pracovať aj s knižnicou TKSL PowerGear ako s adresárovou štruktúrou.

Riešenie knižnice je teda veľmi jednoduché v rámci návrhu a o to zložitejšie v rámci implementácie. Celá knižnica sa bude nachádzať v adresári, ktorý bude niesť nemenný názov *Components*. Tento adresár bude mať adresárovú štruktúru, ktorá bude zodpovedať skupinám v knižnici (obrázok 5.1). V týchto adresároch už budú jednotlivé XML súbory definujúce funkčné bloky. Názvy skupín budú teda rovnaké ako názvy adresárov v koreňovom adresári *Components*.

5.2.3 Kolmé kreslenie vodičov

Charakteristickou črtou všetkých editorov elektrických obvodov je, že vodiče sú na seba kolmé. Pri bližšom pohľade na tento problém si uvedomíme, že riešenie nebude triviálne. Musíme si stanoviť ako moc flexibilne sa nám jednotlivé vodiče budú zalamovať.



Obr. 5.1: Ukážka generovania knižnice: vľavo adresárová štruktúra, vpravo dynamicky generovaný strom knižnice.

Ja som zvolil len jedno zalomenie momentálne kreslenej čiary. Neznamená to, že by nakreslený vodič mohol byť zalomený len raz. Znamená to, že pri aktuálnom ťahaní časti vodiča (segmentu) sa táto časť zalomí len raz.

Vodič na plátne je definovaný ako usporiadaná n-tica bodov (súradníc X a Y) $< p_0, p_1, p_2, ..., p_n >$, pričom platí, že p_0 je počiatočný bod vodiča a p_n koncový.

Segment S je definovaný ako usporiadaná m-tica bodov taká, že $m = 3$ a $S \subset V$, pričom V je vodič a S je segment vodiča V.

Vodič sa teda skladá zo segmentov. Pričom pre segmenty S_1 a S_2, ktoré nasledujú bezprostredne po sebe, platí, že konečný bod prvého segmentu je zhodný s počiatočným bodom druhého segmentu.

Postup kreslenia vodiča bude nasledovný:

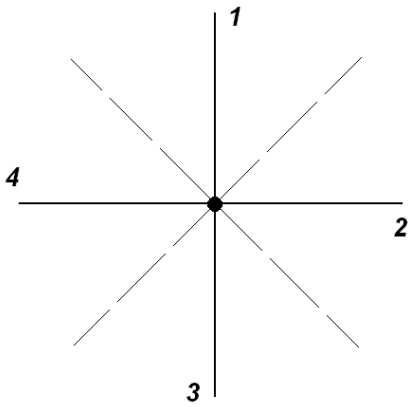
- Užívateľ klikne na vstup komponenty.
- Pri ťahaní myšou sa mu bude zobrazovať segment vznikajúceho vodiča, ktorý sa bude maximálne v jednom bode zalamovať.
- Pri kliknutí na kresliacu plochu sa nový segment zapíše ako súčasť vodiča a ťahá sa ďalší segment rovnakým spôsobom.
- Pri kliknutí do vstupu alebo výstupu inej komponenty sa kreslenie vodiča ukončí.

Po tomto algoritme nám vznikne ľubovoľne mnohokrát zalomení vodič. Ktorý bude presne navrhnutý podľa potrieb užívateľa.

Rozhodovací algoritmus, ktorý má posúdiť akým smerom sa bude vodič kresliť je založený na nasledujúcich vzťahoch, ktoré vyplývajú z obrázka 5.2, na ktorom sú znázornené štyri zóny (označené číslami jedna až štyri).

Je jasné, že ak bude koncový bod B v zóne 1 (prípad 1), tak aj bod zalomenia bude v zóne 1.

Nech bod A je začiatočný bod segmentu (na obrázku 5.2 sa nachádza v mieste pretnutia osí) a bod B je koncový bod segmentu (na obrázku sa môže nachádzať v jednej zo štyroch zón ohraničených prerušovanou čiarou), potom platí:



Obr. 5.2: Zóny kolmého zalomenia (Začiatok segmentu je v strede - bod A).

- pre prvý prípad:

$$|X_B - X_A| < |Y_B - Y_A|$$
$$|Y_B - Y_A| > 0$$

- pre druhý prípad:

$$|X_B - X_A| > |Y_B - Y_A|$$
$$|X_B - X_A| > 0$$

- pre tretí prípad:

$$|X_B - X_A| < |Y_B - Y_A|$$
$$|Y_B - Y_A| < 0$$

- pre štvrtý prípad:

$$|X_B - X_A| > |Y_B - Y_A|$$
$$|X_B - X_A| < 0$$

Pomocou predchádzajúcich vzťahov sme schopní jednoducho vypočítať súradnice bodu, v ktorom nastane zalomenie.[5]

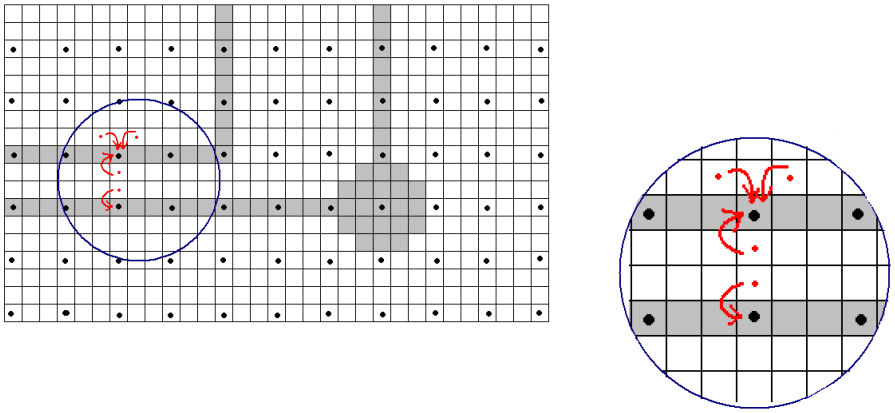
5.2.4 Mriežka

Pripojenie vodiča k vstupu komponenty je oveľa jednoduchšie, ak je miesto umiestnenia viacbodová oblasť a nielen jeden konkrétny bod. Na to, aby sme celú plochu rozdelili na viacbodové plochy, použijeme koncept mriežky.

Pomocou mriežky rozdelíme kresliacu plochu, ktorej jeden bod je jeden pixel, na väčšie časti. Zvoľme teda mriežku tak aby bola dostatočne jemná a neobmedzovala užívateľa v kreslení, ale musí byť hrubšia ako najmenší rozlíšiteľní bod.

Optimálne rozloženie mriežky je 3 pixely. To znamená, že medzi susednými bodmi mriežky budú práve dva pixely. Je to výhodné z dôvodu, že ak sa klikne na ľavý pixel z tejto dvojice tak je jednoduché vypočítať, že kliknutý bod v mriežke bude práve ten

vľavo. Analogicky, ak klikneme na pravý pixel z dvojice, v mriežke to bude práve ten bod v pravo. Je jasné, že medzi dvoma pixelmi už nie je žiadny, na ktorý sa dá kliknúť. Takže je toto rozloženie exaktné. Konkrétnu situáciu mriežky ukazuje obrázok 6.5.5.



Obr. 5.3: Mriežka a transformácia bodov.

Z obrázka je jasné, že všetky vodiče budú ležať na bodoch mriežky. Komponenty už nemôžeme nakresliť tak jednoducho ako vodiče, takže jednotlivé body komponent môžu a budú zasahovať mimo body mriežky. Je veľmi dôležité aby ich vstupy a výstupy boli zarovnané presne na body mriežky.

Pri dodržaní umiestnenia dôležitých bodov na mriežku už nebude problém jednoducho pripájať vodiče na vstupy a výstupy komponent. Je jasné, že pri pohybe nakreslenou cievkou alebo odporom, sa takýto objekt bude umiestňovať nie podľa aktuálnej pozície myši, ale jeho súradnice sa prepočítajú na mriežku. [5]

Detaily riešenia mriežky sú popísané v implementácii editora.

Kapitola 6

Implementácia systému TKSL PowerGear

6.1 Platforma a vývojové prostredie

Systém TKSL a TKSL/C je momentálne najviac používaný na platforme Windows. Preto som zvolil práve Windows ako platformu pre grafický editor. Vývojové prostredie bežiace pod Windows a spĺňajúce potreby pre čo najjednoduchšiu a najmodernejšiu implementáciu je podľa môjho názoru Microsoft Visual Studio 2008. Ako programovací jazyk som zvolil C#[10] a technológiu .NET.

6.1.1 .NET

.NET je zastrešujúci názov pre súbor technológií v softwarových produktoch, ktoré tvoria celú novú platformu, ktorá je dostupná pre Web, Windows a aj Pocket PC.

Pre tvorbu aplikácií splňujúcich tieto myšlienky vydal Microsoft Visual Studio .NET, ktoré bolo oproti predchádzajúcej verzii rozšírené o jednoduchý návrh webových XML služieb .NET, a .NET Framework, zaisťujúci prostredie potrebné pre beh aplikácií a ponúkajúci ako spúšťacie rozhranie, tak potrebné knižnice, ako Java. Visual studio sa skladá z jazykov VB, J#, C# a Managed C++. Týmito jazykmi napísanú aplikáciu potom môžete bez problémov previesť do ASP.NET (Web) alebo do aplikácie pre pocket PC (.NET Compact Framework)

.NET Framework je pre majiteľa operačného systému Windows (vyžadovaná je minimálne verzia Windows 98) k dispozícii zdarma ako samostatný komponent, ktorý sa do systému doinštaluje (býva šírená na CD či DVD rôznych počítačových časopisov; dá sa taktiež stiahnuť cez Windows Update).

K svojmu behu ju vyžadujú väčšinou programy, ktoré pracujú s Active directory alebo SQL vďaka tomu, že .Net má tieto technológie dobre prístupné (rovnaký autor - Microsoft). .NET sľubuje aj lepšie grafické prostredie a operační systém Windows Vista je na tejto technológii postavený.

K dispozícii je aj verzia .NET Compact Framework (.NET CF) pre Pocket PC s operačným systémom Windows Mobile, ktorá je s "klasickou" verziou kompatibilná a tak nie je nutné kompilovať rôzne aplikácie pre PC a PDA - na oboch systémoch bude aplikácia fungovať rovnako. Ako už názov napovedá, .NET CF neobsahuje všetky objekty, funkcie a metódy z pôvodného .NET Frameworku.

GNU obdoba .NET sa nazýva DotGNU. Jej časť nazývaná DotGNU Portable.NET umožňuje spúšťať všetky .NET aplikácie na unixových platformách (Linux, BSD, Mac OS X, Solaris, AIX) a dokonca pomocou nástrojov Cygwin a Mingw32 aj na Windows.

V prostredí operačných systémov Linux, UNIX, Mac OS X je k dispozícii aj sada nástrojov kompatibilných priamo s Microsoft .NET pod názvom Mono. Túto sadu nástrojov však nevyvíja firma Microsoft, ale v obvyklom duchu opensource skupina dobrovoľných vývojárov.

Dve technológie často spojované s .NET sú ASP.NET a jazyk C#.[18]

6.2 Štruktúra funkčných blokov v XML formáte

V prvom rade je nutné stanoviť presnú štruktúru XML súboru, ktorý bude reprezentovať funkčný blok.

V podkapitole 5.2.1 sú uvedené potrebné sekcie v XML súbore. Každá sekcia bude obsahovať niekoľko (variabilné množstvo) záznamov. Je nutné myslieť na to, akým spôsobom sa budú potom jednotlivé záznamy načítať. Každý záznam v sekcii musí mať svoje unikátne meno v rámci sekcie. Zvolil som číselné pomenovanie začínajúce od čísla 1.

Toto značenie uľahčí postupné načítanie. Algoritmus sa bude snažiť načítať zo sekcie záznam s názvom "1", ak sa mu to podarí pokúsi sa načítať záznam s názvom o jedna väčším. Zastaví sa až na zázname, ktorý nebude existovať. Napríklad majme v sekcii čiar štyri čiary. Algoritmus skúsi načítať záznam s názvom "1", to sa mu podarí. Ako druhý skúsi záznam s názvom "2". Až príde na záznam s názvom "5", zistí, že taký záznam sa v sekcii nenachádza a skončí.

Týmto spôsobom budú realizované všetky záznamy v sekciách, kde sa očakáva ich variabilné množstvo. Štruktúra dokumentu bude vyzerať takto:

```
<Nini>

  <Section Name="Lines">
    <Key Name="1" Value="-30,00,-15,00" />
    <Key Name="2" Value="30,00,15,00" />
  </Section>
  <Section Name="Circles">
  </Section>
  <Section Name="Curves">
    <Key Name="1" Value="-15,00,-15,-10,-5,-10,-5,00" />
    <Key Name="2" Value="-5,00,-5,-10,5,-10,5,0" />
    <Key Name="3" Value="5,0,5,-10,15,-10,15,0" />
  </Section>
  <Section Name="Texts">
    <Key Name="1" Value="10,20,Cievka L,8" />
  </Section>
  <Section Name="InPins">
    <Key Name="1" Value="-30,00,pin1" />
  </Section>
  <Section Name="OutPins">
    <Key Name="1" Value="30,00,pin2" />
```

```
  </Section>

  <Section Name="Variables">
    <Key Name="1" Value="L,5" />
  </Section>

  <Section Name="MagicBox">
    [OPTIONAL] <Key Name="SubMagicBoxCircuit" Value="vnorený XML záznam">
    <Key Name="Math" Value="i'=1/L*U" />
    <Key Name="Detail" Value="cievka" />
  </Section>

</Nini>
```

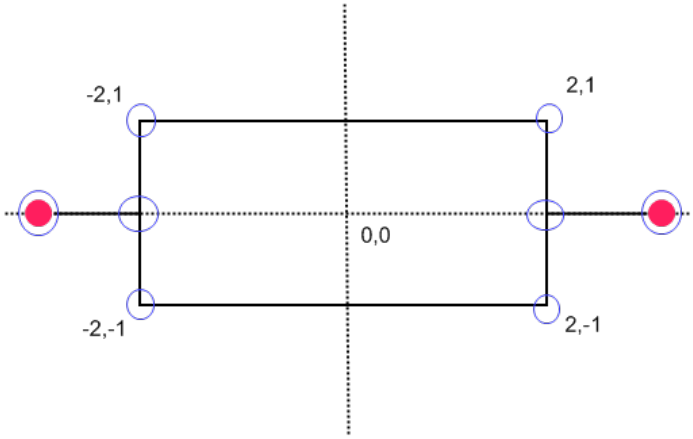
Z príkladu je vidieť, že variabilný počet záznamov sa očakáva v sekciách: *Lines*, *Circles*, *Curves*, *Texts*, *InPins*, *OutPins* a *Variables*. Z príkladu je tiež vidieť, že sekcie nemusia obsahovať žiadny záznam.

Sekcia MagicBox obsahuje záznam *Math*, ktorý popisuje správanie bloku diferenciálnou rovnicou. Ďalej v sekcii nájdeme *Detail* s textovým popisom bloku. Ako posledný záznam je voliteľný *SubMagicBoxCircuit*, ktorý obsahuje schéma bloku vo formáte XML. Tento záznam sa používa pri vytváraní nových blokov z blokov už existujúcich a obsahuje zapojenie blokov, ktoré tvoria tento nový blok.

Hodnota každého záznamu je jeden reťazec. Pri záznamoch, ktoré popisujú napríklad jednu čiaru je nutné využiť tento reťazec na uloženie dvoch bodov čiary. Preto sú reťazce tvorené z hodnôt oddelených čiarkou (CSV). Reťazce jednotlivých sekcií znamenajú:

- Sekcia čiar: $[X_1, Y_1, X_2, Y_2]$, kde X_1, Y_1 sú súradnice začiatočného bodu čiary a X_2, Y_2 koncového.
- Sekcia kriviek: $[X_1, Y_1, X_2, Y_2, X_3, Y_3, X_4, Y_4]$, kde beziérova krivka je charakterizovaná štyrmi bodmi.
- Sekcia kružníc: $[X_1, Y_1, R]$, kde prvé dve hodnoty sú súradnice ľavého horného bodu a R je polomer kružnice.
- Sekcia textov: $[X_1, Y_1, text, veľkosť]$, kde prvé dve hodnoty sú súradnice ľavého horného bodu, za nimi nasleduje samotný text a posledná hodnota vyjadruje veľkosť písma.
- Sekcia vstupov a výstupov: $[X_1, Y_1, N]$, kde prvé dve hodnoty sú súradnice vstupu alebo výstupu a posledná hodnota je jeho názov.
- Sekcia premenných: $[N, H]$, kde N je značka premennej a H vyjadruje jej číselnú hodnotu.

Súradnice bodov sú orientované podľa súradnicového systému, ktorého počiatok sa nachádza v strede komponenty, ako je to znázornené na obrázku 6.1.



Obr. 6.1: Súradnicový systém pri popise komponenty.

6.3 Štruktúra projektu

Projekt je delený do troch vrstiev. Prvá vrstva ja nazvaná *Common* (6.4) a obsahuje základné, prevažne statické triedy, ktoré sú používané vo všetkých ostatných vrstvách. Význam základnej spoločnej triedy je, že môže byť pripojená ku každej inej triede a nevznikne nám cyklická závislosť.

Druhá vrstva sa nazýva *ComponentLayer* (6.5) a zastrešuje objekty komponent, plátna, spojov medzi komponentami a v neposlednom rade ukladanie a načítanie. V stručnosti sa dá povedať, že táto vrstva zastrešuje všetko, čo sa týka kreslenia, teda modelovania systémov.

Posledná vrstva je GUI (Grafické užívateľské rozhranie - 6.6), ktorá implementuje len jednotlivé okná, tlačidlá, dialógy a iné vizuálne efekty. Tieto potom volajú funkcie vrstvy *ComponentLayer* (6.5). Tým je GUI absolútne odtienené od vykreslovacích a výpočtových mechanizmov.

6.4 Vrstva základných spoločných funkcií – Common

Táto vrstva je obecná pre viacero programov a teda obsahuje aj triedy, ktoré sa momentálne nepoužívajú, ale očakáva sa ich použitie v budúcom rozvoji programu.

Najdôležitejšie triedy sú:

- NameHelper
- NiNiHelper
- PointHelper
- Serializer

Trieda NameHelper slúži k prideľovaniu unikátnych názvov k práve vytvoreným komponentám. Pracuje na jednoduchom princípe počítadla, ktoré sa pri každom vygenerovanom

unikátnom názve inkrementuje. Vždy sa pri generovaní názvu použije názov typu komponenty a ten sa zkonkatenuje s hodnotou počítadla.

Trieda NiNiHelper zaobaľuje open-source nástroj NiNi[9]. Tento nástroj slúži na jednoduchú správu konfiguračných súborov ako sú XML, INI, CONFIG. TKSL PowerGear ukladá všetky konfigurácie[1] vo formáte XML a trieda NiNiHelper poskytuje funkcie na jednoduché ukladanie a načítanie používaných štruktúr programu.

NiNiHelper považuje za konfiguračné premenné len tie, ktoré sú označené atribútom *SettingsPropertyAttribute*. Takto je ošetrené, ktoré premenné v triede sa budú plniť z konfiguračného súboru[2], prípadne, ktoré sa budú zapisovať do konfiguračného súboru.

Trieda PointHelper slúži na rozšírenie operácií s typom Point v jazyku C#. Nad samotným typom nie je definované sčítanie, tak som nadefinoval funkciu, ktorá realizuje sumu nad objektmi Point, ktoré dostane ako parameter.

Serializer je veľmi užitočná trieda, ktorá nám akýkoľvek serializovateľný objekt serializuje na reťazec v XML formáte. Ak objekt obsahuje aj iné serializovateľné objekty, prípadne ich zoznamy, alebo iné štruktúry, rekurzívne sa serializujú aj tie. Trieda samozrejme dokáže aj reverznú operáciu a teda deserializovať XML záznam na pôvodný objekt.

6.5 Vrstva komponent - Component Layer

6.5.1 Plátno - Canvas

Trieda *Canvas*, ako napovedá názov, reprezentuje plátno. V našom prípade je to modelovací priestor, na ktorom užívateľ modeluje systém klikaním myši. Trieda dedí vlastnosti internej komponenty v .NET *PictureBox*. Tým pádom obsahuje základné funkcie na zobrazovanie a kreslenie.

Trieda obsahuje dva základné zoznamy:

- zoznam prvkov *MBComponent* (podkapitola 6.5.2),
- zoznam prvkov *Wire* (podkapitola 6.5.3).

Trieda si pri inicializácii vytvára inštanciu mriežky (podkapitola 6.5.5), ktorá pokrýva plátno aktuálnej triedy *Canvas*. Je nutné alokovať všetky prostriedky pri každej inštancii, pretože program využíva záložky a preto vytvára nové plátno na každú záložku.

Trieda predefinováva metódu Paint triedy *PictureBox*. V tejto metóde prechádza oba zoznamy a na každom prvku zoznamov volá funkciu Paint s parametrom Graphics ([11]). Tento parameter reprezentuje objekt vykresľovania pomocou GDI+. Všetko zapísané[3] do objektu Graphics bude vykreslené na plátne.

Dôležitou časťou triedy sú metódy, ktoré pridávajú nový objekt do mriežky a odstraňujú zmazaný objekt z mriežky. K tomuto slúžia funkcie *AddToGrid* a *DeleteFromGrid*. Obe metódy sú preťažené a teda ich telo sa vyberie na základe typu parametru.

Ak je parameter typu MBComponent, pridá sa do mriežky komponenta, resp. jej obdĺžnikové ohraničenie. Každá komponenta má funkciu vracajúcu obdĺžnik, ktorý ohraničuje komponentu. Do mriežky sa potom vpíšu všetky body, do ktorých zasahuje spočítaný obdĺžnik.

[1] Konfiguráciou sa myslí aj popis funkčného bloku.
[2] Konfiguračným súborom sa v našom prípade myslí XML súbor s popisom funkčného bloku.
[3] Zapisuje sa pomocou pripravených metód objektu.

Ako druhý krok sa vpíšu prvky mriežky, ktoré budú symbolizovať vstupy a výstupy komponenty. Každý prvok mriežky takto pridaný obsahuje referenciu na objekt, ktorý má charakterizovať. Takto máme rýchly prístup k prvkom na modelovacom plátne. Zo súradníc kliknutia myši spočítame súradnice v mriežke a v objekte mriežka nájdeme referenciu na objekt, ktorý sa nachádza na tomto mieste.

Ak je parameter typu Wire, tak je generovanie prvkov mriežky zložitejšie v tom, že spoj je tvorený niekoľkými segmentmi. Vložiť horizontálny segment nie je zložité, pretože leží v jednom riadku. Ak je segment vertikálny je treba prepočítať súradnice na stĺpce matice mriežky.

Metóda na odstránenie komponent a spojov z mriežky je zjednodušená tak, že každý prvok (komponenta aj spoj) má uložený zoznam prvkov mriežky, ktoré mu prináležia. Pri odstraňovaní z mriežky je teda algoritmus veľmi jednoduchý, Stačí prejsť všetky prvky, ktoré si odstránený objekt pamätá a jeden po druhom ich zmazať z mriežky.

Pri vkladaní novej komponenty na plátno sa trieda Canvas prihlási k odberu udalosti *NeedRepaint*, ktorú vyvolá komponenta napríklad pri posune alebo pri rotácii a tak dostane trieda plátna správu, aby sa prekreslila.

Pri vložení nového spoja sa trieda prihlási k odberu udalosti *OnDelete*. Reakcia na túto udalosť je odstránenie spojnice z plátna a z mriežky. Udalosť je nutná, pretože spoje sú závislé na existencii komponent. Ak zmažeme komponentu, vieme určiť, ktorá to bola, pretože ju máme označenú. Táto komponenta ale zmaže aj všetky spojnice s inými komponentami. Vtedy nastáva situácia, že spojnice vyvolajú udalosť OnDelete, a tým sa sami nahlásia plátnu na zmazanie.

Aby sme boli schopný pracovať s triedou Canvas, musíme nadefinovať funkcie, ktoré sa vykonajú pri akciách myšou. Tieto funkcie sú definované v GUI, pretože sú pre každý Canvas rovnaké. Vždy pri vytvorení novej triedy plátna sa automaticky naviažu.

Za zmienku ešte stojí možnosť približovania a odďaľovania, ktorá je riešená pomocou maticovej transformácie objektov na plátne. Táto transformácia sa vytvorí priamo na objekte Graphics metódou $ScaleTransform$ ([13]). Parametrami na vstupe vyberie zmenšenie plátna v smere X a Y, ktoré sa potom aplikuje na každý objekt na plátne.

Jednou z posledných dôležitých funkcií je automatické doplnenie parazitných kapacít. K tomu slúži funkcia $FillParasiticCapacitances$. Funkcia vytvorí nový blok kondenzátoru a pripojí ho do existujúceho uzla v sieti. Takto vytvorený kondenzátor hneď pripojí na zem. Takto pripojí parazitné kondenzátory na každý uzol v obvode.

Pri vytváraní komponent systém spočíta ich posunutie, aby sa neprekrývali, a ako druhý krok doplní vodiče medzi zemou a kondenzátorom a medzi kondenzátorom a uzlom ku ktorému parazitnú kapacitu pripojujeme (Obrázok 7.6).

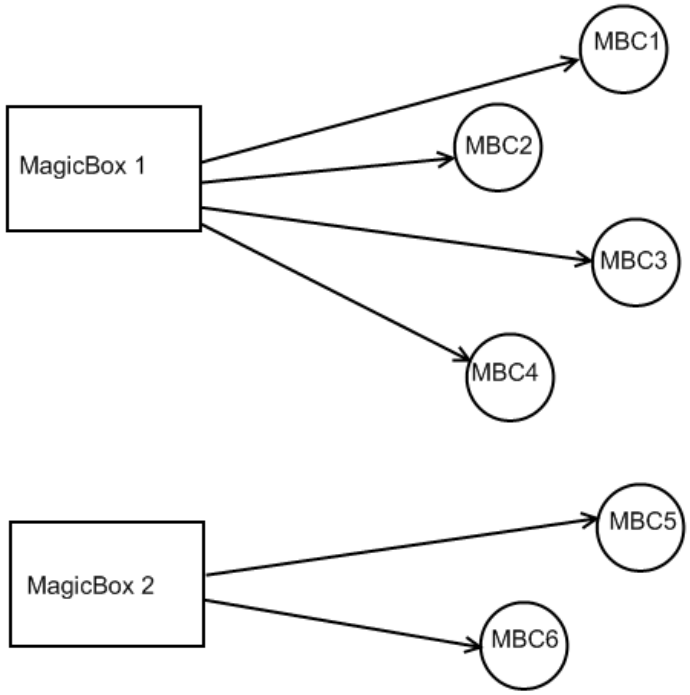
6.5.2 Bloky - MagicBox a MBComponent

Najdôležitejšia súčasť je interpretácia funkčných blokov. V knižnici sa nachádzajú funkčné bloky, ktoré chceme rozmiestňovať na plátne. Funkčné bloky v knižnici sú implementované triedou $MagicBox$. Táto trieda ja navrhnutá tak, aby nebrala ohľad na komponentu, ktorú popisuje. Je to obecná trieda, v ktorej sa dá nadefinovať vizuálna stránka a vnútorné správanie. Tým pádom systém pracuje len s množinou "magických škatuliek" a ich vnútorné vlastnosti sú pre neho transparentné.

MBComponent je trieda, ktorá v sebe nesie referenciu na objekt MagicBox a je to vlastne reprezentácia objektu MagicBox na plátne. Napríklad ak MagicBox popisuje diódu, tak týchto diód môžeme namodelovať hneď niekoľko. A každá táto namodelovaná dióda na plátne už nie je objektom triedy MagicBox, ale MBComponent.

Obrazne sa dá prirovnať tento koncept k triedam a ich inštanciám. Ak by MagicBox bola v rámci obraznosti trieda, tak MBComponent je inštancia tejto triedy. Na obrázku 6.2 je znázornené, že objekty MagicBox sú základnými kameňmi pri tvorbe objektov MBComponent. Objekt MB reprezentuje vizuálnu stránku, typ, zoznam premenných a ich počiatočné hodnoty. Taktiež definuje popis diferenciálnymi rovnicami a prípadnú vnútornú štruktúr pre prípad, že bol zostrojený z existujúcich komponent. Je to v podstate programová reprezentácia konfiguračného XML súboru funkčného bloku.

Objekt MBC obsahuje referenciu na objekt MB a teda má prístup k základnej špecifikácii, ktorú rozširuje o konkrétne umiestnenie na plátne a nastavenie vnútorných premenných pre konkrétnu komponentu v systéme. Taktiež obsahuje záznamy o pripojených komponentách.



Obr. 6.2: Vzťah triedy MagicBox a MBComponent.

Trieda MBComponent obsahuje rad metód. Najhlavnejšia z nich je metóda Paint. Táto prejde všetky vizualizačné útvary rodičovskej inštancie triedy MagicBox. Každý jeden vizualizačný objekt transformuje na svoje súradnice na plátne a vykreslí ho.

Pri inicializácii objektu triedy MBComponent sa generuje zoznam premenných na základe premenných definovaných v rodičovskom objekte MB. Tieto sú potom nastavované v rámci konkrétnej MBC.

Ako druhé sa generujú objekty triedy PIN, ktoré na rozdiel od objektov DPin v triede MagicBox, neobsahujú informácie o umiestnení. Obsahujú len informácie o aktuálne pripojenej komponente na vstup reprezentovaný objektom Pin a názov tohto prípojného bodu.

Ďalšie potrebné metódy triedy nám vrátia Pin, na ktorý užívateľ práve klikol. Je to potrebné hlavne v rámci rotácie komponenty. Túto funkciu poskytuje metóda *GetComponentPin*. Metóda má ako parameter súradnice kliknutia myšou a vzhľadom na umiestnenie komponenty na plátne a jej aktuálneho natočenia nájde správny Pin.

Vyššie už bolo spomenuté, že trieda MBComponent obsahuje metódu, ktorá vráti obdĺžnik, ktorý opisuje komponentu. Výpočet obdĺžnika je založený na najľavejšom a najvrchnejšom bode všetkých vykresľovaných útvarov a na najnižšom najpravejšom bode. Z nich je spočítaný obdĺžnik, ku ktorému sa ku každej strane pridá 6 pixelov. Je to z dôvodu, aby obdĺžnik presahoval hranice komponenty a užívateľ tak mal možnosť jednoduchšie označiť komponentu kliknutím na jej okraj a nie len na stred.

Trieda obsahuje vlastnosť Rotate, ktorá udáva uhol natočenia v stupňoch. Tento uhol sa používa v metóde Paint, kde za pomoci metód *RotateAt* a $TransformPoints$ zrotuje každý bod vykresľovaných tvarov maticovou transformáciou. K tomuto sa využíva triedy $Matrix$[12].

Posledné spomeniem udalosti spojené s triedou MBComponent. Trieda môže vyvolať tri udalosti:

- OnDelete – vyvolá sa, ak je komponenta odstránená z plátna.
- OnDrag – vyvolá sa, ak je komponenta ťahaná (premiestňovaná) užívateľom.
- NeedRepaint – vyvolá sa, ak sa na komponente zmenila niektorá z vyzualizačných častí (napríklad sa natočila o iný uhol) a je treba ju prekresliť.

6.5.3 Spoje - Wire

Koncept spojov je založený na tvorení jednotlivých segmentov. Spoj sa vytvorí pri kliknutí na vstup alebo výstup nejakej komponenty. Pri inicializácii sa nastaví komponenta ako začiatočná a vygenerujú sa prvé tri súradnice spoja.

Prvá súradnica už je pevná a tretia je tam, kde sa práve nachádza myš. Druhá (prostredná) súradnica je bod zalomenia segmentu. Tento sa pri každej zmene konečnej súradnice prepočíta. Ak klikneme inde než na vstup alebo výstup komponenty, aktuálne body sa uložia a už sa s nimi nedá hýbať. Potom sa vytvoria nové dva body segmentu s tým, že posledný bod predchádzajúceho segmentu je prvý bod nového. Takto sa postupuje, až kým užívateľ nepripojí spoj ku komponente.

Uloženie referencie na počiatočnú komponentu a generovanie prvých troch bodov sa uskutoční v konštruktore. Na spočítanie novej hodnoty stredového bodu (bodu zalomenia) slúži metóda *ComputeCenter*. Metódy na aktualizáciu spoja pri zmene polohy alebo natočenia pripojenej komponenty sú *UpdateSegemnt* (pri zmene koncovej komponenty) a *UpdateSegemntReverse* (pri zmene počiatočnej komponenty).

Pri napojení na komponentu sa trieda Wire prihlási k odberu jej udalostí *OnDrag* a *OnDelete*. Vďaka tomu vie, kedy má prepočítať svoju polohu (OnDrag - komponenta mení svoju polohu) a kedy je nutné zmazať spoj (OnDelete - komponenta bola odstránená).

Metóda Paint je jednoduchá, pretože vykreslenie spoja znamená vykresliť niekoľko čiar, pričom ich súradnice máme presne dané.

6.5.4 Ukladanie a načítanie

Ukladanie a načítanie modelov je realizované cez serializáciu do XML. Pre uloženie kompletného elektrického zapojenia je nutné uložiť ako komponenty, tak všetky vodiče modelu. Aby systém nemusel náhodne rozložiť komponenty po ploche, ukladá sa aj ich aktuálna poloha.

Pred samotným serializovaním sa vytvoria prechodné objekty *WirePureData* a *ComponentPureData*. Tieto sa generujú metódou v triedach MBComponent a Wire. Je to z dôvodu serializácie celého objektu. Pôvodné objekty obsahujú aj referencie a iné premenné, ktoré serializovať nechceme.

Zoznamy objektov *WirePureData* a *ComponentPureData* sa uložia do výsledného objektu PureDataObj, ktorý sa následne serializuje v triede *Serializer*. Tá prevedie celý objekt na reťazec vo formáte XML, ktorý sa uloží do súboru na disk.

Objekt *ComponentPureData* obsahuje tieto premenné:

- ClickPoint – súradnice stredu komponenty, podľa ktorých ju môžeme pri načítaní vložiť na správne miesto.
- Rotate – uhol otočenia komponenty.
- Variables – zoznam premenných s hodnotami pre konkrétnu komponentu.
- Name – názov komponenty v rámci modelu.
- MagicBoxName – názov triedy MagicBox (napr. Odpor), ktorý slúži na dodatočné dohľadanie referencie na rodičovskú triedu MB.

Objekt *WirePureData* obsahuje tieto premenné:

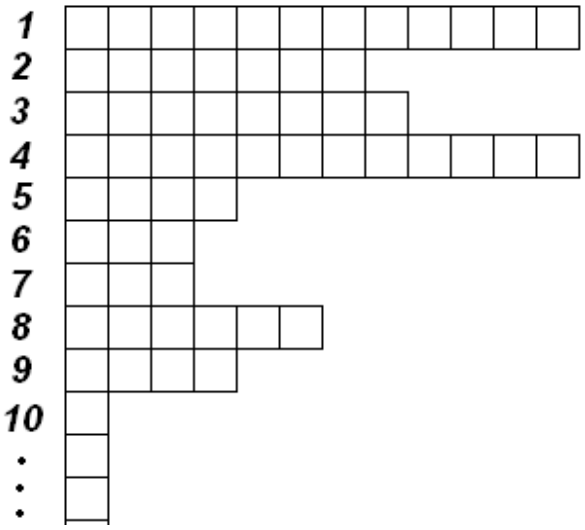
- Points – súradnice jednotlivých bodov vodiča.
- StartComponentName – názov prvej komponenty (MBComponent), ku ktorej je vodič pripojený.
- EndComponentName – názov druhej komponenty (MBComponent), ku ktorej je vodič pripojený.

Pri načítaní sa použije reverzný postup. Získa sa XML súbor, ktorý sa deserializuje na objekt PureDataObj. Z neho sa vyberú objekty *WirePureData* a *ComponentPureData*. Najskôr sa z objektov *ComponentPureData* vytvoria objekty MBC. V konštruktore sa dohľadajú správne referencie na MB. Ako druhý krok sa vytvoria objekty Wire, pri ktorých sa dohľadajú komponenty, ktoré sme pridali na plátno v prvom kroku.

6.5.5 Mriežka

Mriežka je prostriedok pre väčší komfort užívateľa. Na druhú stranu sa dá využiť aj na účely rýchleho získania informácií o aktuálnej pozícii kurzora myši. Vždy, keď kurzor prechádza ponad plátno, nachádza sa nad určitým bodom mriežky. Tento bod obsahuje informácie o objekte, ktorý sa na ňom nachádza, a teda o objekte, nad ktorým je práve kurzor. Je to jednoduchšie a rýchlejšie než vždy podľa súradníc prechádzať všetky objekty a zisťovať, nad ktorým práve sme.

Mriežka je implementovaná ako riedka matica (obrázok 6.3). Tým pádom nezaberá toľko pamäťových prostriedkov ako normálna matica N x N. Riedka matica je implementovaná ako pole zoznamov položiek mriežky. Teda y-ová súradnica je pevne daná veľkosťou poľa a x-ová súradnica môže mať ľubovoľnú veľkosť. Problémom je len to, že každá položka v zozname musí obsahovať parameter index, ktorý určuje x-ovú súradnicu.



Obr. 6.3: Mriežka - riedka matica.

6.6 GUI

Grafické rozhranie nemá skoro žiadnu zložitú funkčnosť. Zoskupuje všetky grafické prvky ako: tlačidlá, panely, textové polia a mnoho ďalších. Každá udalosť, ktorá je v GUI vyvolaná, smeruje do vrstvy komponent (6.5).

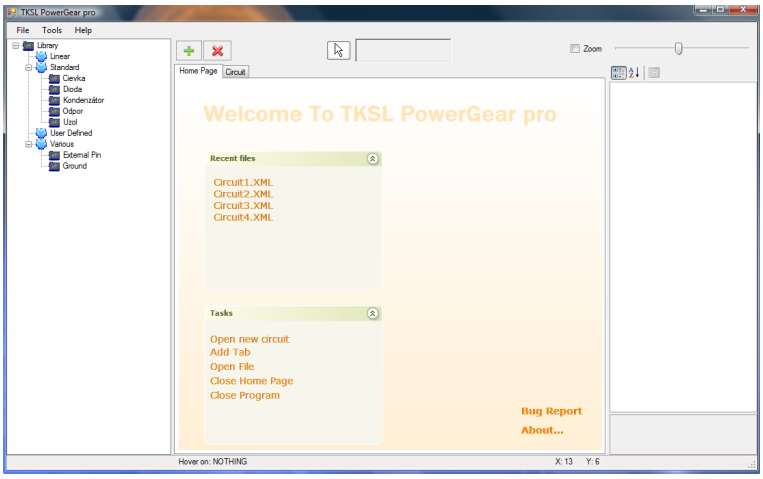
Gui obsahuje rozšírenú funkčnosť a to obsluhu záložiek. Pre tento účel slúži $TabHelper$. Táto trieda obsluhuje tvorbu nových záložiek a automatické generovanie prvkov, ktoré do nich patria. Taktiež sa stará o korektné zatvorenie záložiek a uvoľnenie zdrojov.

Najdôležitejšou súčasťou je naviazanie udalostí[17] na kliknutie myši (stlačenie/pustenie myši, pohyb myši) pri generovaní nového plátna (Canvas). Každé plátno má naviazané rovnaké funkcie na udalosti myši. Tieto udalosti sú definované na hlavnom formulári.

Veľmi výhodné sa ukázalo využitie komponenty $PropertyGrid$[4], ktorá veľmi elegantne zobrazuje povolené vlastnosti objektu, ktorý je do nej priradený. Dokonale spĺňa účel nastavovania vnútorných premenných komponent, názvov a typov.

Kapitola 7

Systém TKSL PowerGear



Obr. 7.1: Úvodná obrazovka TKSL PowerGear.

Aktuálny systém TKSL PowerGear podporuje tieto základné funkcie:

- Vytvorenie modelu z funkčných blokov nachádzajúcich sa v knižnici blokov.
- Uloženie a následné načítanie modelu.
- Jednoduchá tvorba nových funkčných blokov z blokov už existujúcich.
- Približovanie a odďaľovanie modelu.
- Podpora záložiek.

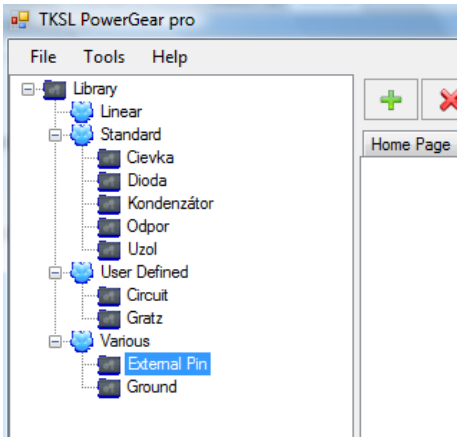
7.1 Knižnica funkčných blokov

V ľavej časti programu sa nachádza knižnica (Obrázok 7.2) komponent vo forme stromu. Komponenty sú členené do skupín podľa druhu komponenty.

Knižnica je uložená v adresári programu v zložke *Components*. Každá skupina má vlastnú zložku s rovnakým menom. V zložkách skupín sa nachádzajú jednotlivé XML súbory

definujúce funkčné bloky. Tieto bloky je možné mazať, pridávať, kopírovať a hlavne editovať podľa vlastnej vôle.

Ako užívateľ máte všetky dostupné možnosti, ako meniť prvky knižnice, prípadne meniť charakteristiky samotných prvkov. Popis štruktúry prvkov je popísaný v podkapitole 6.2.



Obr. 7.2: Dynamicky generovaná knižnica TKSL PowerGear.

7.2 Modelovanie

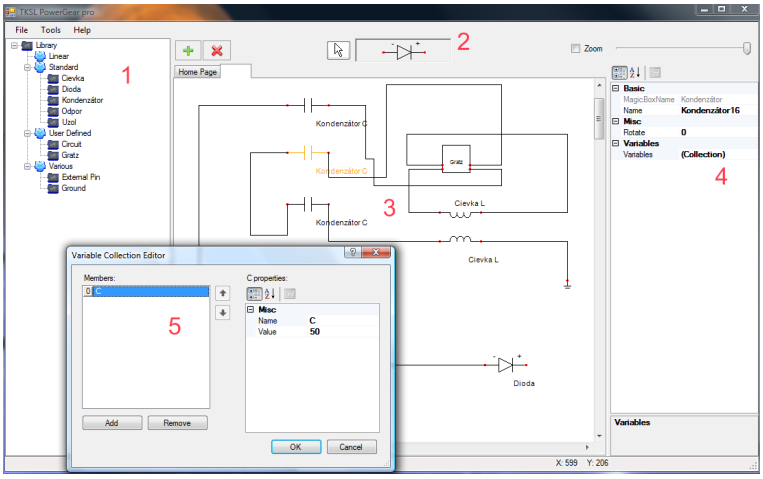
Z podkapitoly 7.1 vieme, že knižnica lokalizovaná v ľavej časti obsahuje všetky dostupné komponenty. Kliknutím vyberieme blok, ktorý chceme naniesť na plátno. Zvolený blok sa zobrazí aj v strede vrchnej časti. Ak máme blok zvolený, kliknutím na plátno ho umiestnime na miesto kliknutia. Ak stlačíme myšie tlačidlo a ťaháme, blok sa posúva po plátne, kým tlačidlo myši nepustíme.

Ak znovu klikneme na blok a držím tlačidlo myši stlačené môžeme s blokom znovu pohybovať. Pri kliknutí na blok, sa tento blok označí (zmení svoju farbu) a na pravej strane programu sa zobrazia jeho informácie a editovateľné prvky.

V tomto paneli je hlavné si mimo možnosti nastavenia názvu všimnúť aj možnosť zmeny premenných (Variables). Pri zmene premenných najskôr klikneme na tlačidlo vpravo v poli Variables. Otvorí sa nám nové okno (Obrázok 7.3), v ktorom sú jednotlivé premenné zoradené v ľavej časti a v pravej časti sa nachádza ich názov a hodnota. Po ukončení editovania potvrdíme tlačidlom OK.

Kreslenie vodičov je realizované jednoduchým kliknutím na prípojný bod (vstup/výstup) bloku. Vtedy sa systém automaticky prepne do režimu modelovania spojov. Pri každom kliknutí zahneme vodič podľa potreby. Modelovanie spoja sa ukončí kliknutím na iný vstup/výstup bloku.

Zjednodušenie modelovania prináša funkcia *SelectOnly*, ktorá sa aktivuje/deaktivuje kliknutím na ikonu kurzora, ktorá je umiestnená na vrchnej časti editora. Tento mód zabraňuje modelovaniu blokov a spojov, povoľuje len ich posúvanie a mazanie.

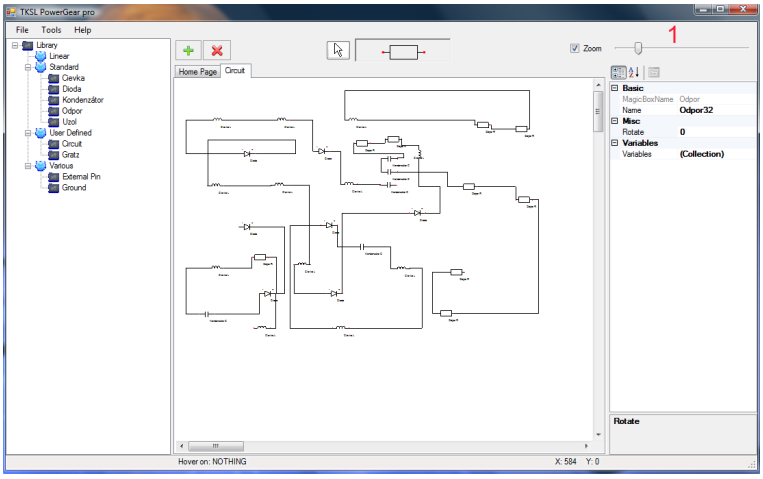


Obr. 7.3: Ukážka modelovania (1 - knižnica, 2 - práve zvolený blok, 3 - plátno s modelom, 4 - editačný panel, 5 - rozšírené editačné okno s premennými).

7.2.1 Zoom

Pri modelovaní rozsiahleho systému je veľmi žiadaná možnosť oddialenia. Vďaka nej sme schopní sledovať oveľa väčšiu časť modelu. K tomuto slúži posuvník v pravom hornom rohu (7.4). Pri označení políčka Zoom sa aktivuje približovania a oddaľovanie. Posunom bežca doprava sa približujeme k modelu, opačným smerom ho oddaľujeme.

Pri manipulácii s touto lupou je dočasne deaktivované samotné modelovanie. To znamená, že pri oddialenom alebo priblíženom pohľade, užívateľ nemôže zasahovať do modelu. Pre opätovné povolenie editácie je nutné vypnúť mód lupy a to tak, že odznačíme políčko Zoom.

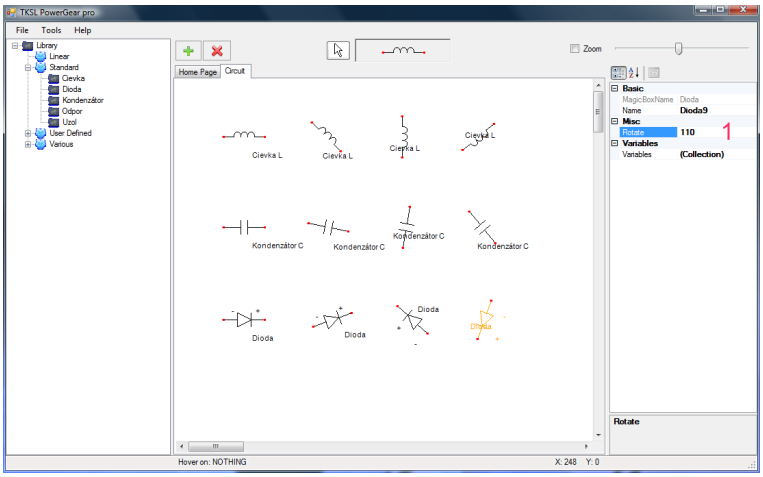


Obr. 7.4: Ukážka lupy (1 - bežec oddialenia a priblíženia).

7.2.2 Rotácia

V podkapitole 7.2 je popis, ako sme schopní editovať vnútorný stav bloku. V rovnakom nastavení sa nachádza aj možnosť rotácie komponenty. K tomuto účelu slúži vlastnosť *Rotate* (obrázok 7.5).

Hodnota rotácie je v stupňoch od 0 po 360. Blok je teda možné rotovať do ktoréhokoľvek uhla.



Obr. 7.5: Ukážka rotácie (1 - miesto pre zadanie uhla rotácie).

7.2.3 Mazanie

Zmazanie ktorejkoľvek časti modelu je možné tlačidlom *Del*, alebo položkou v menu Tools → Delete Selected.

Pri zmazaní bloku sa zmažú aj všetky spoje, ktoré vedú do neho alebo z neho. Pred pokusom o zmazanie je potrebné mať vyznačenú časť, ktorú chceme odstrániť. Táto časť sa označí kliknutím na ňu a po označení zmení svoju farbu.

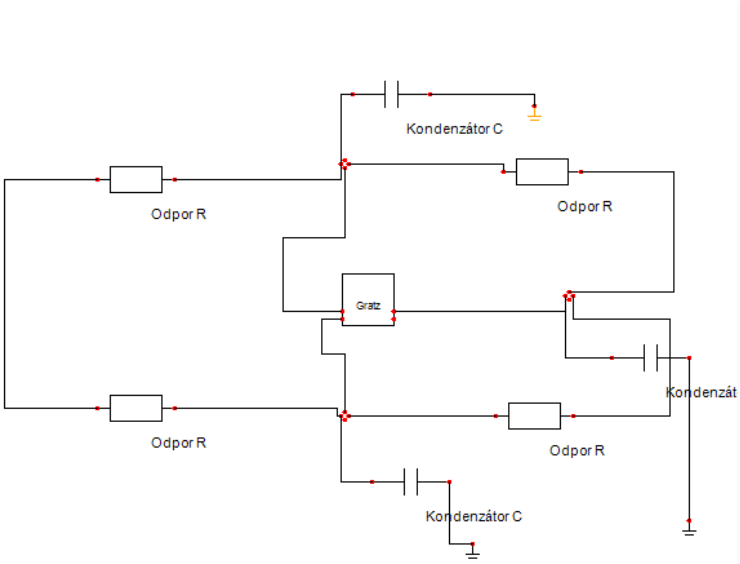
7.2.4 Ukladanie a načítanie

Ak chceme uložiť nami namodelovaný systém, je postup obdobný ako u väčšiny iných programov. Ukladá sa ten model, ktorý máme aktuálne otvorený v záložke. Pre uloženie klikneme v menu na File → Save. Model sa ukladá do XML súboru, ktorý sme schopní ručne editovať. Načítanie je možné cez položku menu File → Open.

7.2.5 Generovanie parazitných kapacít

Pre automatické doplnenie parazitných kapacít, ktoré nám vyriešia problémy v obvode (kapitola 3), stačí jediný krok. Týmto krokom je kliknutie na položku v menu Tools → Fill parasitic capacitances.

Systém automaticky doplní kondenzátory (Obrázok 7.6) do miest uzlov a doplní aj ich automatické uzemnenie. Tým nám pri výpočte pomôže vyriešiť nežiaduce javy, ktoré by vznikali, ak by tam parazitné kapacity neboli.

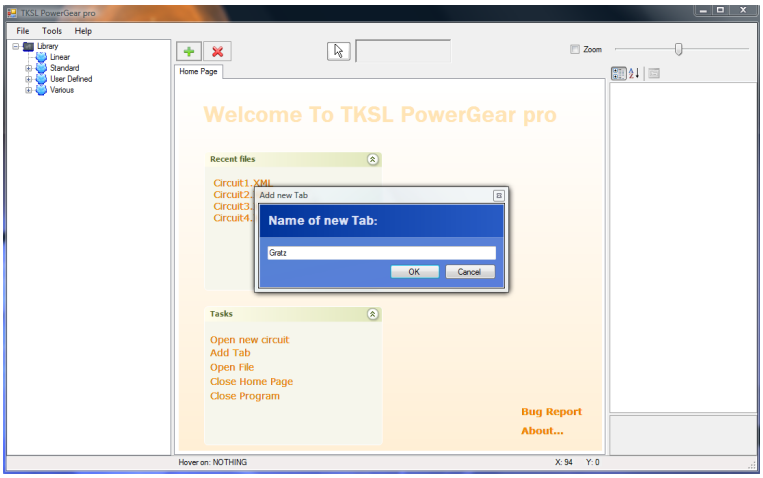


Obr. 7.6: Doplnenie parazitných kapacít.

7.3 Vytvorenie nového bloku

Ako príklad vytvorenia nového bloku som vybral vytvorenie dvojcestného usmerňovača (Graetzovo zapojenie [19]).

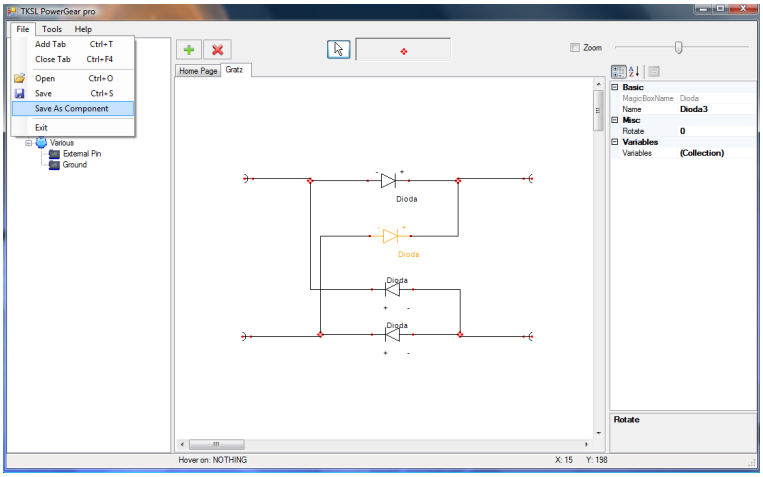
V prvom rade je nutné vytvoriť plátno, ktoré nazveme tak, ako chceme volať náš nový blok. V našom príklade je to *Gratz* (Obrázok 7.7).



Obr. 7.7: Vytvorenie novej záložky s plátnom pod názvom Gratz.

Ak sme vytvorili záložku správne, vidíme prázdne plátno a záložka sa volá *Gratz*. Na toto plátno namodelujeme náš dvojcestný usmerňovač. Nová komponenta musí mať definované vstupy a výstupy. Tieto namodelujeme tak, že z knižnice zo skupiny Various natiahneme štyri bloky s názvom *ExternalPin*. Tieto bloky reprezentujú prípojné body pre

našu novú komponentu (Obrázok 7.8).

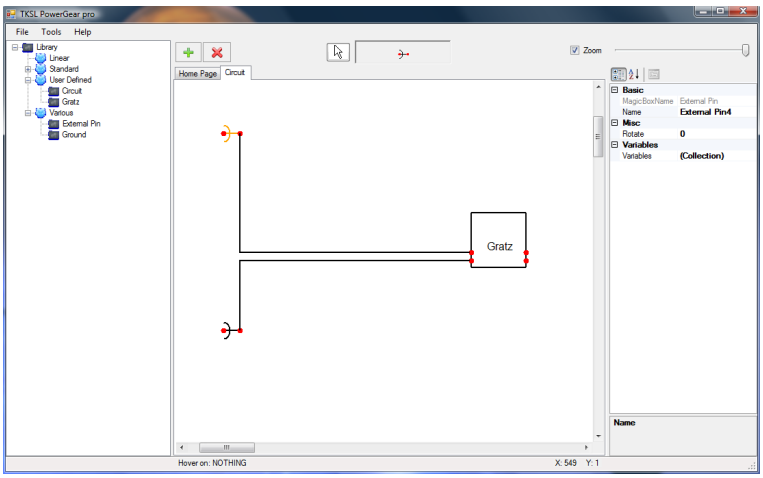


Obr. 7.8: Namodelovaný usmerňovač.

Podľa toho, kde sú umiestnené prípojné body, sa vygeneruje vizuálna časť nového bloku. Jeho telo bude obyčajný štvorec a na stranách bude mať vstupy. Umiestnenie vstup je závislé na umiestnení prípojných bodov. Ak sme dali prípojné body viac vpravo, budú aj vstupy na novej komponente zobrazené na pravej strane bloku. Analogicky pre ostatné strany.

Ak sme ukončili modelovanie bloku, uložíme ho kliknutím na položku menu File → Save As Component. Systém automaticky uloží komponentu do skupiny $UserDefined$ a nazve ju podľa mena záložky.

Po uložení sa blok vloží do knižnice vľavo a sme schopní ho používať rovnako ako všetky ostatné bloky (Obrázok 7.9).



Obr. 7.9: Použitie nového bloku usmerňovača.

Kapitola 8

Možnosti rozšírenia systému TKSL PowerGear

8.1 Transformácia grafu pripojených komponent na sústavu diferenciálnych rovníc

Editor ako taký nemá žiadne priame prepojenie na TKSL. Je koncipovaný ako prvý krok grafickej nadstavby nad TKSL. Druhým krokom bude transformačný modul, ktorý sa začlení medzi editor TKSL PowerGear a samotné TKSL. Tento modul bude prevádzať grafovú reprezentáciu prepojenia komponent modelu na sústavy diferenciálnych rovníc v špecifickom formáte TKSL. Tieto sa už len spočítajú a výsledky sa budú interpretovať.

Transformačný modul bude tvoriť most z Editora do systému TKSL. Bude musieť pracovať s hierarchickou štruktúrou, ktorá bola navrhnutá pre TKSL PowerGear. Taktiež bude využívať vnútorné premenné jednotlivých komponent, ktoré budú definované v Editore.

Transformácie nebudú triviálne a budú musieť riešiť množstvo algebraických úprav a zložité prechádzanie grafom. Preto je transformačný modul predmetom ďalšej diplomovej práce, ktorá pomôže dotvoriť profesionálny grafický nástroj nad systémom TKSL.

8.2 Mnohonásobné klonovanie

Pri vytváraní rozsiahleho modelu nastáva situácia, kedy by nám veľmi uľahčilo prácu mnohonásobné klonovanie blokov. Nie je to len jednoduché "Copy&Paste". Je nutné nadefinovať zapojenie nových naklonovaných blokov. Ľahko nagenerujeme tisíce komponent v rade za sebou, ale ťažko budeme každú ručne pripájať do siete.

Návrh systému je pripravený na túto funkciu. Každé zapojenie je možné jednoducho serializovať a následne deserializovať. Vďaka tomu je možné so schémou, alebo jej časťou, ľubovoľne manipulovať. Zásadný problém je definovanie prípojných bodov na komponente (sústave komponent), ktorú klonujeme a aj na komponente (sústave komponent), ku ktorej sa bude pripájať. Funkciu, ktorá bude plniť definíciu pripojení pri klonovaní, je nutné navrhnúť dostatočne obecne, pretože systém podporuje rôzne variácie vstupov a výstupov.

8.3 Editor vizuálnej podoby blokov

Otvorená štruktúra popisu funkčných blokov a jej obecnosť je momentálne závislá na ochote užívateľa editovať XML súbory. Príjemnou črtou programu by bol editor funkčných blokov, ktorý by vygeneroval XML súbor automaticky, podľa toho, čo užívateľ zadal. Najväčší prínos tejto funkcie je v definovaní vizuálnej podoby bloku, ktorá sa určite ľahšie popisuje kreslením čiar než textovým popisom.

K tomuto editoru je možné využiť už existujúci koncept plátna s mriežkou, ktorý by sa obohatil o zadávanie jednotlivých vizuálnych elementov, ako je čiara, krivka, kružnica a popisné texty. Jednotlivé útvary už by boli zarovnané na mriežku, a tým by podporili zarovnanie na plátne so schémou.

Neoceniteľné by určite bolo aj presné zadanie vstupných a výstupných miest, ktoré by taktiež boli zarovnané presne na mriežku. Definícia premenných a iných vnútorných stavov už nie je problém. Táto časť je už obecne vyriešená aj v stávajúcom editore.

Kapitola 9

Záver

Cieľom práce v prvej časti bolo zoznámiť sa so spôsobmi riešenia a hlavne zadávania a modelovania elektrických (a iných) modelov.

Ako je vidieť v kapitole 2, riešenie zložitých obvodov je možné hlavne pomocou diferenciálnych rovníc. Ukázali sme si, že existuje množstvo známych spôsobov riešenia diferenciálnych rovníc, ale existuje aj nie moc známa metóda pomocou Taylorovho radu, ktorá je vyvíjaná na Fakulte Informačných Technológií na VUT v Brne.

Riešenie zložitých zapojení diferenciálnymi rovnicami nie je vždy jednoduché a občas vedie na obrovské algebraické problémy. Tieto problémy môžu nastať aj pri relatívne jednoduchých schémach a zabraňujú nám vyriešiť zadaný obvod. V rámci týchto problémov sa rozvíja metóda výpočtu pomocou parazitných kapacít, ktorá je popísaná v kapitole 3. Parazitné kapacity nám po ustálení prechodných dejov v obvode vyriešia mnohé problémy a hlavne nás odbremenia od algebraických slučiek.

Metóda parazitných kapacít je jednou z kľúčových záležitostí pri skúmaní popisov elektrických obvodov diferenciálnymi rovnicami a čaká ju ešte veľký vývoj. Študuje sa aj metóda parazitných indukčností, ktorá zaznamenala podobné pozitívne výsledky.

Druhá časť práce sa zamerala na vývoj simulačného nástroja, v ktorom je užívateľ schopný namodelovať systém a tento systém neskôr odsimulovať v nástroji TKSL.

V kapitole 4 som uviedol pár známych systémov určených na modelovanie a simulovanie elektrických (a iných) modelov. Tieto systémy mi poslúžili ako vodítko pri návrhu grafického editora TKSL PowerGear, ktorý je užívateľsky prívetivý a disponuje funkciami a možnosťami, ktoré sú v tejto sfére štandardom.

TKSL PowerGear je grafická nadstavba systému TKSL. Systém TKSL disponuje len textovým vstupom, ktorý je pri zložitých modeloch neprehľadný a hlavne vyžaduje od užívateľa znalosť syntaxe a popisu obvodu diferenciálnymi rovnicami. Z toho dôvodu vznikla potreba vyvinúť systém, ktorý by poskytol užívateľovi komfort pri zadávaní elektrickej schémy.

Hlavnou črtou systému je jeho pripravenosť na spoluprácu s TKSL. Od začiatku bol navrhovaný v súlade s metódami, ktoré budú pri transformácii schémy na sústavy rovníc potrebné. To znamená, že obsahuje automatizované funkcie na generovanie parazitných kapacít do modelu.

Transformácia schémy na sústavu diferenciálnych rovníc nie je triviálna záležitosť a bude vyžadovať oddelený vývoj. Preto systém zatiaľ neumožňuje simuláciu, ale slúži len ako

grafické prostredie na modelovanie.

Na druhej strane sa systém inšpiroval profesionálnymi editormi a využíva ich silné stránky. Za spomenutie stojí oddelená knižnica komponent, ktorá je otvorená a každá jej časť editovateľná. Ako veľké plus považujem hierarchické členenie komponent a obecnú štruktúru komponenty, ktorej správanie v systéme a vizuálne spracovanie sa dá nadefinovať presne podľa predstavy užívateľa. Vďaka tomu je systém schopný modelovať aj iné systémy než elektrické obvody. Každá komponenta, ktorá sa dá popísať diferenciálnou rovnicou, je modelovateľná v systéme.

Systém TKSL PowerGear je plne funkčný editor modelov s otvorenou koncepciou knižnice prvkov, ktorý je pripravený na rozšírenie o modul transformácie obvodu na sústavu diferenciálnych rovníc. Po rozšírení by mal slúžiť ako plnohodnotné, užívateľsky prívetivé simulačné prostredie s profesionálnymi črtami.

Literatúra

[1] Antoš, R.: *Modelování lineárních a nelineárních elektronických obvodů*. Diplomová práca, FEI VUT v Brne, 2001.

[2] Cellier, F.: Dymola: Environment for Object-oriented Modeling of Physical Systems. 2005, [Online; navštíveno 3. 01. 2009].
URL http://people.inf.ethz.ch/~fcellier/Res/Soft/Dymola_engl.html

[3] Foltin, M.; Ernek, M.: Matlab (7) SimPowerSystems. *AT&P journal*, 11 2008.

[4] Gold, M.: Using Property Grid in C#. 2004, [Online; navštíveno 17. 05. 2009].
URL http://www.c-sharpcorner.com/UploadFile/mgold/PropertyGridInCSharp11302005004139AM/PropertyGridInCSharp.aspx

[5] Kadák, M.: *Parazitné kapacity pri riešení elektrických obvodov*. Bakalárska práca, FIT VUT v Brne, 2007.

[6] Konvalina, J.: *Datové vstupy simulačního systému TKSL*. Diplomová práca, FIT VUT v Brne, 2006.

[7] Krupková, V.; Studená, V.: *Matematická analýza 2*. Nakladatelství VUT v Brne, 1991.

[8] MathWorks: Dymola, Dynamic Modeling Laboratory. 2008, [Online; navštíveno 3. 01. 2009].
URL http://www.mathworks.com/products/connections/product_main.shtml?prod_id=62

[9] Matzelle, B. R.: Nini .NET Configuration Library. 2009, [Online; navštíveno 17. 05. 2009].
URL http://nini.sourceforge.net/manual.php

[10] Microsoft: C# Tutorials. 2009, [Online; navštíveno 17. 05. 2009].
URL http://msdn.microsoft.com/cs-cz/library/aa288436(en-us).aspx

[11] Microsoft: Graphics Class. 2009, [Online; navštíveno 19. 05. 2009].
URL
http://msdn.microsoft.com/en-us/library/system.drawing.graphics.aspx

[12] Microsoft: Matrix Class. 2009, [Online; navštíveno 19. 05. 2009].
URL http://msdn.microsoft.com/en-us/library/system.drawing.drawing2d.matrix.aspx

[13] Microsoft: ScaleTransform Method. 2009, [Online; navštíveno 19. 05. 2009].
URL http://msdn.microsoft.com/en-us/library/zhc2xxtx.aspx

[14] Minárik, M.: *Autonómna metóda riešenia elektrických obvodov*. Bakalárska práca, FIT VUT v Brne, 2007.

[15] Peringer, P.: Dymola/Modelica. [online], 2006, [navštívené 3. 01. 2009].
URL www.fit.vutbr.cz/study/courses/IMS/public/prednasky/IMS-dymola.pdf

[16] Petřek, J.: *Grafický systém nad TKSL/C*. Diplomová práca, VUT v Brne, 2001.

[17] Sanju: Events In C#. 2004, [Online; navštíveno 17. 05. 2009].
URL http://www.csharphelp.com/archives/archive253.html

[18] Wikipedie: .NET — Wikipedie: Otevřená encyklopedie. 2009, [Online; navštíveno 17. 05. 2009].
URL http://cs.wikipedia.org/w/index.php?title=.NET&oldid=3753976

[19] Wikipedie: Usměrňovač — Wikipedie: Otevřená encyklopedie. 2009, [Online; navštíveno 17. 05. 2009].
URL http://cs.wikipedia.org/w/index.php?title=Usm%C4%9Br%C5%88ova%C4%8D&oldid=3730580

Zoznam použitých skratiek a symbolov

TKSL – Taylor Kunovsky Simulation Language

GUI – Grafické užívateľské rozhranie

XML – eXtensible Markup Language

CSV – Comma-separated values

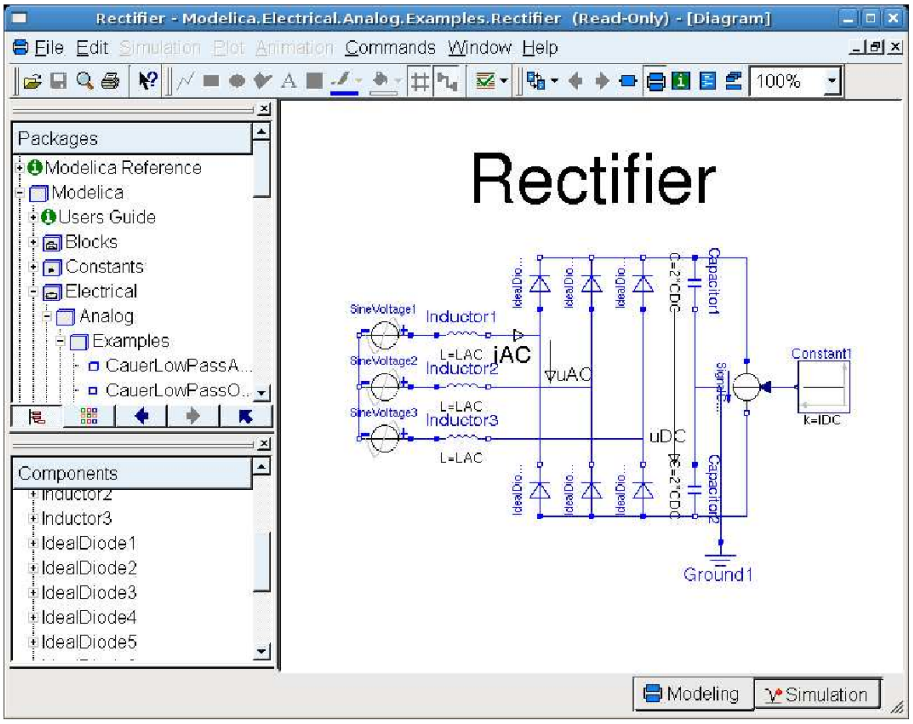
GDI+ – Graphics Device Interface Plus

MB – Magic Box

MBC – Magic Box Component

Príloha A

Dymola - ukážky



Obr. A.1: Ukážka grafického rozhrania Dymoly

```
model rc "Model el. obvodu"
  Resistor     R(R=1000);
  Capacitor    C(C=0.001);
  SineVoltage  U(offset=5, V=0.5, freqHz=1);
  Ground       ground;
equation
  connect(U.n, C.n);
  connect(U.p, R.p);
  connect(R.n, C.p);
  connect(U.n, ground.p);
end rc;
```

Obr. A.2: Príklad: elektrický obovd – RC článok

```
connector Pin
  Voltage      v;
  flow Current i;  // flow => suma=0
end;

partial model OnePort "bázová třída"
  Pin p, n;
  Voltage v;   // napětí
  Current i;   // proud
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;
```

Obr. A.3: Definícia základných komponent

```
model Resistor "ideální rezistor"
  extends OnePort;
  parameter Real R(unit="Ohm") "odpor";
equation
  R*i = v;
end Resistor;

model Capacitor "ideální kondenzátor"
  extends OnePort;
  parameter Real C(unit="F") "kapacita";
equation
  C * der(v) = i;  // diferenciální rovnice
end Capacitor;
```

Obr. A.4: Ideálny rezistor/kondenzátor

Príloha B

TKSL PowerGear – obsah konfiguračných súborov základných funkčných blokov

```
<Nini>
  <Section Name="Lines">
    <Key Name="1" Value="-30,00,-15,00" />
    <Key Name="2" Value="30,00,15,00" />
  </Section>
  <Section Name="Circles">
  </Section>
  <Section Name="Curves">
    <Key Name="1" Value="-15,00,-15,-10,-5,-10,-5,00" />
    <Key Name="2" Value="-5,00,-5,-10,5,-10,5,0" />
    <Key Name="3" Value="5,0,5,-10,15,-10,15,0" />
  </Section>
  <Section Name="Texts">
    <Key Name="1" Value="10,20,Cievka L,8" />
  </Section>
  <Section Name="InPins">
    <Key Name="1" Value="-30,00,pin1" />
  </Section>
  <Section Name="OutPins">
    <Key Name="1" Value="30,00,pin2" />
  </Section>
  <Section Name="Variables">
    <Key Name="1" Value="L,5" />
  </Section>
  <Section Name="MagicBox">
    <Key Name="Math" Value="i'=1/L*u" />
    <Key Name="Detail" Value="cievka" />
  </Section>
</Nini>
```

Konfiguračný XML súbor špecifikujúci cievku.

```
<Nini>

  <Section Name="Lines">
    <Key Name="1" Value="-5,10,-5,-10" />
    <Key Name="2" Value="5,10,5,-10" />
    <Key Name="3" Value="-30,00,-5,00" />
    <Key Name="4" Value="30,00,5,00" />
  </Section>
  <Section Name="Circles">

  </Section>
  <Section Name="Curves">

  </Section>
  <Section Name="Texts">
    <Key Name="1" Value="10,20,Kondenzátor C,8" />
  </Section>
  <Section Name="InPins">
    <Key Name="1" Value="-30,00,pin1" />
  </Section>
  <Section Name="OutPins">
    <Key Name="1" Value="30,00,pin2" />
  </Section>

  <Section Name="Variables">
    <Key Name="1" Value="C,50" />
  </Section>

  <Section Name="MagicBox">


    <Key Name="Math" Value="u'=1/C*i" />
    <Key Name="Detail" Value="kondenzator" />

  </Section>

</Nini>
```

Konfiguračný XML súbor špecifikujúci kondenzátor.

```
<Nini>

  <Section Name="Lines">
    <Key Name="1" Value="-20,10,20,10" />
    <Key Name="2" Value="-20,-10,20,-10" />
    <Key Name="3" Value="-20,10,-20,-10" />
    <Key Name="4" Value="20,10,20,-10" />
    <Key Name="5" Value="-30,00,-20,00" />
    <Key Name="6" Value="30,00,20,00" />
  </Section>
  <Section Name="Circles">

  </Section>
  <Section Name="Curves">

  </Section>
  <Section Name="Texts">
    <Key Name="1" Value="10,20,Odpor R,8" />
  </Section>
  <Section Name="InPins">
    <Key Name="1" Value="-30,00,pin1" />
  </Section>
  <Section Name="OutPins">
    <Key Name="1" Value="30,00,pin2" />
  </Section>

  <Section Name="Variables">
    <Key Name="1" Value="R,50" />
  </Section>


  <Section Name="MagicBox">


    <Key Name="Math" Value="i=U/R" />
    <Key Name="Detail" Value="odpor" />

  </Section>

</Nini>
```

Konfiguračný XML súbor špecifikujúci odpor.