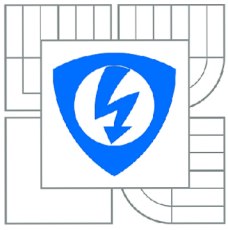
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

VYUŽITÍ MODELŮ V JAZYCE MODELICA V PROSTŘEDÍ MATLAB-SIMULINK

MODELICA MODELS USE IN MATLAB-SIMULINK ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

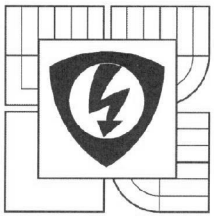
AUTOR PRÁCE
AUTHOR

Bc. JAN GLOS

VEDOUCÍ PRÁCE
SUPERVISOR

prof. Ing. PAVEL VÁCLAVEK, Ph.D.

BRNO 2015



VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student: Bc. Jan Glos
Ročník: 2

ID: 125430
Akademický rok: 2014/15

NÁZEV TÉMATU:

Využití modelů v jazyce Modelica v prostředí Matlab-Simulink

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvoření SW nástroje pro umožnění zahrnutí modelu v jazyce Modelica do schématu v prostředí Matlab-Simulink.

1. Seznamte se s postupy fyzikálního modelování v jazyce Modelica a zejména v prostředí OpenModelica a možnostmi importu modelů v jazyce Modelica do prostředí Matlab-Simulink a co-simulace v prostředích Matlab-Simulink a Openmodelica.
2. Analyzujte možnosti přípravy SW nástroje pro import modelů na základě standardu FMI pro použití v Matlab-Simulink a proveďte návrh funkcionality nástroje a jeho uživatelského rozhraní.
3. Vytvořte nástroj pro Matlab-Simulink, který umožní simulaci modelů v jazyce Modelica v prostředí Matlab-Simulink s využitím standardu FMI.
4. Vytvořte grafické uživatelské rozhraní pro ovládání nástroje a dále vytvořte rozhraní pro ovládání pomocí skriptů.
5. Sepište uživatelský návod k vytvořenému nástroji a proveďte křížovou kontrolu pro ověření kompatibility nástroje s FMU exportovanými různými nástroji.

DOPORUČENÁ LITERATURA:

[1] Torsten Blochwitz, Martin Otter: The Functional Mockup Interface for Tool independent Exchange of Simulation Models, 2011
[2] Peter Fritzson: OpenModelica System Documentation, 2013

Termín zadání: 9. 2. 2015 ***Termín odevzdání:*** 18.5.2015

Vedoucí práce: prof. Ing. Pavel Václavek, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Abstrakt
Tato diplomová práce řeší použití modelů vytvořených v jazyce Modelica v prostředí Matlab/Simulink. První část práce se věnuje jazyku Modelica a standardu Functional Mock-up Interface, což je standard pro výměnu a souběžnou simulaci dynamických modelů, který je podporován ve většině modelovacích nástrojů pro jazyk Modelica. Na základě tohoto standardu byl vytvořen nástroj FMUtoolbox doplňující prostředí Matlab/-Simulink o možnost načtení a simulace modelů ve formátu Functional Mock-up Unit. Nástroj poskytuje blok pro Simulink, grafické uživatelské rozhraní a možnost ovládání prostřednictvím příkazové řádky.

Klíčová slova
Modelica, Functional Mock-up Interface, FMI, Functional Mock-up Unit, FMU, výměna modelů, Matlab, Simulink, akauzální modelování, OpenModelica

Abstract
This thesis solves the use of Modelica models in Matlab/Simulink enviroment. The first part is focused on Modelica language and Functional Mock-up Interface, a standard way for model exchange and co-simulation of dynamic models, which is supported by most Modelica oriented tools. Based on this standard FMUtoolbox was created and it provides the ability to import and simulate models exported as Functional Mock-up Unit. The tool provides a Simulink block, graphical and command-line interface.

Keywords
Modelica, Functional Mock-up Interface, FMI, Functional Mock-up Unit, FMU, Model exchange, Matlab, Simulink, Acausal modeling, OpenModelica

Bibliografická citace:

GLOS, J. *Využití modelů v jazyce Modelica v prostředí Matlab-Simulink*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015. 72 s. Vedoucí prof. Ing. Pavel Václavek, Ph.D.

Prohlašuji, že svou diplomovou práci na téma „Využití modelů v jazyce Modelica v prostředí Matlab-Simulink“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne 13. května 2015

.....

podpis autora

Děkuji vedoucímu diplomové práce prof. Ing. Pavlovi Václavkovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne 13. května 2015

.....

podpis autora

Obsah

Úvod **11**

1 Jazyk Modelica a OpenModelica **12**
1.1 Jazyk Modelica . . . 12
1.2 OpenModelica . . . 13
1.3 Fyzikální modelování . . . 15

2 Functional Mock-up Interface **19**
2.1 Rozhraní modelu . . . 19
2.1.1 Matematický popis modelu . . . 19
2.1.2 Funkce modelu . . . 19
2.2 Popis modelu . . . 22

3 Návrh nástroje FMUtoolbox **24**
3.1 Import modelů v jazyce Modelica do Matlab/Simulink . . . 24
3.2 Návrh funkcionality . . . 24
3.3 Návrh uživatelského rozhraní . . . 26

4 Realizace nástroje FMUtoolbox **32**
4.1 Třída FMUclass . . . 33
4.2 Instalace FMUtoolboxu . . . 35
4.3 Příprava FMU pro simulaci . . . 38
4.4 Knihovna FMUtoolbox pro Simulink . . . 39
4.5 Uživatelské rozhraní . . . 40
4.6 Simulace FMU . . . 44
4.7 Další funkce FMUtoolboxu . . . 46
4.8 Ukládání provozních stavů (logging) . . . 50
4.9 Podpůrné nástroje pro FMUtoolbox . . . 51
4.10 Podpora FMU exportovaných z nástroje OpenModelica . . . 53
4.11 Návratové hodnoty funkcí FMU . . . 55
4.12 Dynamická knihovna - rozhraní Matlab/FMU . . . 55
4.13 Zpracování výjimek . . . 56

5 Křížová kontrola nástroje FMUtoolbox **58**
5.1 Nástroje pro křížovou kontrolu . . . 58
5.2 Výsledky kontroly . . . 59
5.3 Vyhodnocení křížové kontroly . . . 61

6 Praktické použití nástroje FMUtoolbox **62**
6.1 Model RC článku . . . 62
6.2 Tepelné ztráty elektromotoru . . . 64
6.3 Řízení stejnosměrného motoru . . . 66

7 Závěr **68**

Literatura **69**

Seznam použitých symbolů a zkratek **71**

Seznam příloh **72**

Seznam obrázků

1.1 OpenModelica Connection Editor - grafický pohled na model 14
1.2 OpenModelica Connection Editor - textový pohled na model 14
1.3 OpenModelica Connection Editor - parametry rezistoru 15
1.4 Příklad zápisníku v programu OMNotebook 16
1.5 Schéma RC článku 17
1.6 Model RC článku v prostředí OpenModelica 18
1.7 Model RC článku v prostředí Matlab/Simulink 18
2.1 Matematický popis modelu FMU - po částech spojitý systém 20
3.1 Grafické rozhraní nástroje FMUtoolbox 27
3.2 Příklad sekce atributy v GUI FMUtoolboxu 28
3.3 Nastavení parametrů FMU 28
3.4 Příklad tabulky výstupů v GUI FMUtoolboxu 29
3.5 Příklad tabulky vstupů v GUI FMUtoolboxu 29
3.6 Příklad konfigurace portů v FMUtoolboxu 30
3.7 Příklad možných vzhledů bloků FMUtoolboxu 31
4.1 Blokové schéma nástroje FMUtoolbox 32
4.2 Úvodní obrazovka instalátoru nástroje FMUtoolbox 36
4.3 Knihovna FMUtoolbox pro Simulink 40
4.4 Horní lišty okna GUI 41
4.5 Dialogové okno s varováním při nalezení FMU v jiném umístění 46
4.6 Subsystém se skalárními vstupy a výstupy 47
4.7 Vzhled bloku se skalárními vstupy a výstupy 47
4.8 Subsystém s vektorovým vstupem a výstupem 48
4.9 Vzhled bloku s vektorovým vstupem a výstupem 48
4.10 Subsystém se sběrnicovým vstupem a výstupem 48
4.11 Vzhled bloku se sběrnicovým vstupem a výstupem 49
4.12 Subsystém s kombinací skalárních, vektorových a sběrnicových vstupů a výstupů 49
4.13 Vzhled bloku s kombinací skalárních, vektorových a sběrnicových vstupů a výstupů 49
5.1 Křížová kontrola nástroje FMUtoolbox a FMU CoupledClutches 59
6.1 Schéma RC článku v jazyce Modelica upraveného pro export FMU 62
6.2 Schéma Simulink pro ověření simulace FMU - RC článek 63
6.3 Průběhy vstupních a výstupních signálů simulace RC článku 63
6.4 Model tepelných ztrát motoru 64
6.5 Graf výstupů simulace tepelného modelu motoru v prostředí OpenModelica 65
6.6 Graf výstupů simulace tepelného modelu motoru v prostředí Matlab/Simulink 65
6.7 Model stejnosměrného motoru v prostředí OpenModelica 66
6.8 Schéma zapojení pro identifikaci přenosu stejnosměrného motoru 66
6.9 Schéma zapojení pro ověření návrhu regulátoru 67
6.10 Průběh odezvy na skok žádané hodnoty uzavřené smyčky regulátoru a modelu motoru v jazyce Modelica 67

Seznam tabulek

4.1 Vektor Position a jeho význam 41
5.1 Křížová kontrola nástroje FMUtoolbox 60

Úvod

Tato práce se zabývá použitím modelů v jazyce Modelica v prostředí Matlab/Simulink. V první části práce je popsán jazyk Modelica a prostředí OpenModelica, druhá část je věnována standardu Functional Mock-up Interface (FMI) a v poslední části je řešen návrh nástroje FMUtoolbox pro import modelů v jazyce Modelica do prostředí Matlab/Simulink.

Motivací k této práci je možnost použití modelů využívajících fyzikální (akauzální) modelování v prostředí Matlab/Simulink, což může být v některých případech výrazně výhodnější, než použití kauzálních modelů vytvořených v Matlab/Simulink. Příkladem může být ověřování regulace na modelu, který je velmi podobný realitě. Pro návrh regulátorů se často používají jednoduché lineární modely (např. druhého řádu), které však nemusí přesně odpovídat realitě. Je tedy možné vytvořit dva modely, první pro návrh regulátoru a druhý pro jeho ověření. Pokud je pro vytvoření druhého modelu použito akauzální modelování, je možné jednoduše provést simulace za různých okolních podmínek (teplota), v okolí nelinearit nebo pro proměnlivé parametry modelu (náhodně působící poruchy apod.). Tento postup je označován jako testování MIL (Model in the loop) - regulátor i proces jsou modelovány v některém softwarovém nástroji (Matlab/Simulink ...).

Akauzální modely mohou být s výhodou použity také pro testování SIL (Software in the loop), kdy je model regulátoru nahrazen jeho softwarovou realizací (například v jazyce C nebo VHDL) a je provedena simulace této kombinace. Nabízí se možnost využít akauzální modely i pro testování PIL (Processor in the loop), přičemž regulátor je již realizován v mikroprocesoru či FPGA, vstupy a výstupy jsou simulovány pomocí modelu a předávány regulátoru pomocí komunikační sběrnice.

1. Jazyk Modelica a OpenModelica

Modelica je volně dostupný objektově orientovaný jazyk určený pro modelování rozsáhlých, komplexních a heterogenních fyzikálních systémů. Je vhodný pro různé systémy (mechanické, elektrické, elektronické, hydraulické, tepelné apod.) [1].

Modely vytvořené v jazyku Modelica jsou popsané pomocí diferenciálních, algebraických a diferenčních rovnic.

Jazyk Modelica využívá celá řada simulačních prostředí - například OpenModelica, Dymola, JModelica.org, MapleSim, SimulationX apod.

1.1. Jazyk Modelica

Model v jazyce Modelica je tvořen zdrojovým kódem, který je formálně definován specifikací jazyka Modelica [1]. Model může obsahovat různé objekty (rezistory, kapacitory, snímače apod.), které mohou být mezi sebou propojeny. K tomu slouží konektory (či porty) jednotlivých objektů.

Model je uložen jako textový soubor s příponou .mo. Základní struktura jednoduchého modelu je následující:

```
1 model NazevModelu
2     <promenneModelu>
3 equation
4     <rovniceModelu>
5 end NazevModelu;
```

Každý model začíná klíčovým slovem `model`, za nímž následuje název modelu. Na následujících řádcích jsou deklarovány proměnné modelu. Za klíčovým slovem `equation` následují rovnice modelu. Definice rovnic a i celého modelu je ukončena klíčovým slovem `end` a názvem modelu [2].

Například vztah mezi třemi proměnnými, z nichž jedna je součtem zbylých dvou proměnných, by bylo možné zapsat takto:

```
1 model Soucet
2     Real A;
3     Real B;
4     Real C;
5 equation
6     C = A + B;
7 end Soucet;
```

Rovnice na řádku 6 určuje, že proměnná `C` je součtem proměnných `A` a `B`, přičemž symbol = neznamená přiřazení do proměnné `C`, ale udává rovnost mezi proměnnými. Ekvivalentním zápisem rovnice na řádku 6 by bylo vyjádření $A + B - C = 0$.

Zdrojový kód je možné doplnit o dva různé typy komentářů. První typ (komentáře podobné např. jazyku C++; řádky 2 a 3 v následujícím příkladu) je překladačem jazyka Modelica ignorován (přesněji přeložen jako mezera). Druhý typ je označován jako „dokumentační komentář“ („documentation comment“), vkládá se vždy na konec řádku

a překladač jazyka Modelica jej použije jako popis daného prvku (modelu, proměnné, rovnice). Při exportu FMU jsou dokumentační komentáře použity jako `description` modelu nebo proměnné.

```
1 model Soucet "Soucet dvou cisel"
2     Real A; // Scitanec 1
3     Real B; // Scitanec 2
4     Real C "Vysledek";
5 equation
6     C = A + B "Rovnice souctu dvou cisel";
7 end Soucet;
```

Dědění modelů je možné pomocí klíčového slova `extends`, tím je možné znovu použít již existující kód bez nutnosti jeho opakování.

```
1 model Prumer "Prumer dvou cisel"
2     extends Soucet;
3     Real P; // Prumer
4 equation
5     P = C / 2;
6 end Prumer;
```

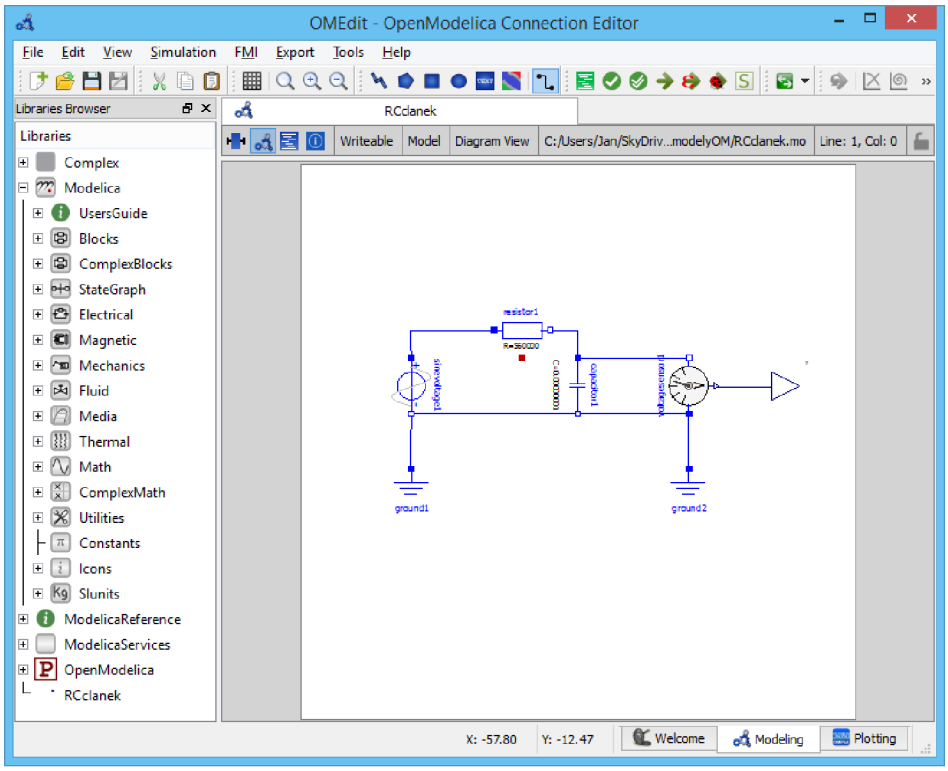
Jazyk Modelica obsahuje mnoho dalších možností, které mohou být použity při sestavování modelu. Popis celého jazyka Modelica by však svým rozsahem přesáhl tuto práci, proto jsou uvedeny pouze základní vlastnosti jazyka.

1.2. OpenModelica

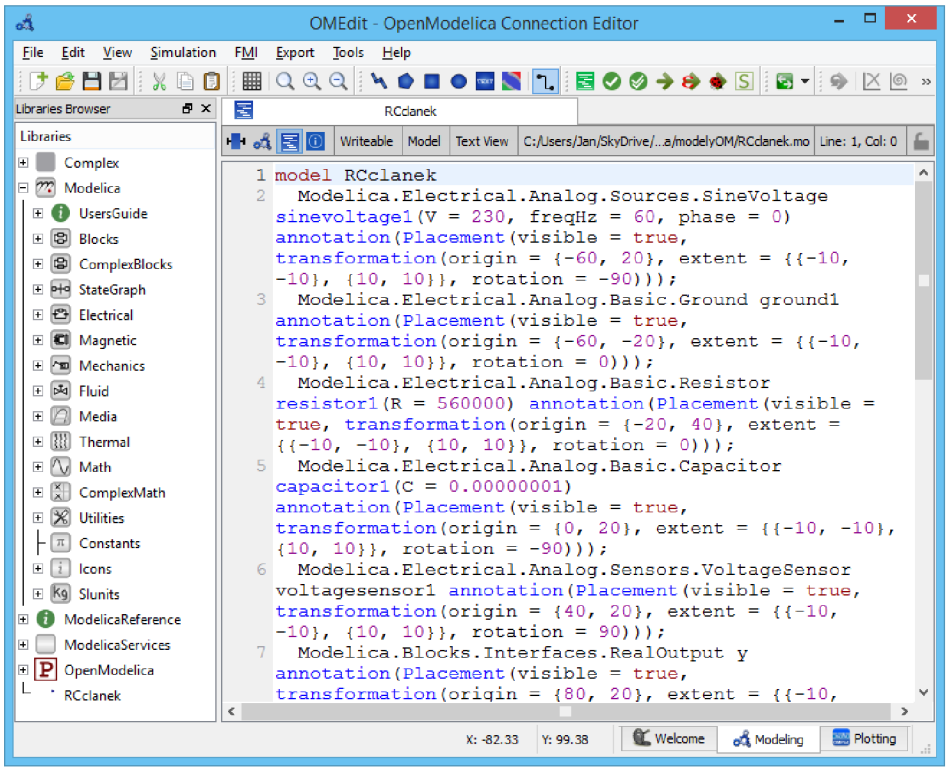
OpenModelica je open-source prostředí pro modelování a simulaci modelů v jazyce Modelica. Prostředí se skládá z jednotlivých nástrojů, které budou dále popsány.

OMEdit je zkratka pro *OpenModelica Connection Editor*, což je grafický a textový nástroj pro vytváření modelů v jazyce Modelica. Uživatelské rozhraní je znázorněno na obrázcích 1.1 a 1.2 - v horní části okna se nacházejí ovládací prvky programu, v levé části okna se nachází knihovna modelů a největší část zabírá vlastní model. Vytváření modelu je uživatelsky přívětivé, stačí požadované objekty přetáhnout z knihovny do modelu a následně propojit. Po dvojkliku na kterýkoliv objekt v modelu se otevře dialogové okno, které umožní nastavení parametrů daného objektu (viz obr. 1.3). Po přepnutí do textového módu se zobrazí zdrojový kód modelu v jazyce Modelica a je možné provádět jeho úpravy (např. úpravy parametrů objektů, přidávání a mazání objektů apod.).

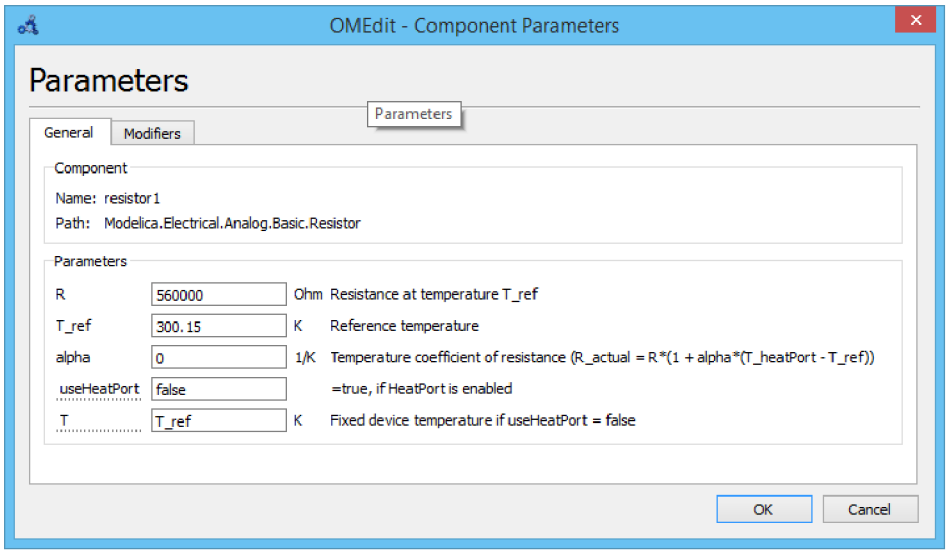
OMNotebook je nástroj pro vytváření WYSIWYG (What-You-See-Is-What-You-Get) interaktivních knih, který je možné použít například pro výuku programování v jazyce Modelica [3], dokumentování postupů při programování, anebo pro tvorbu tutoriálů. OMNotebook se mimo jiné používá pro interpretaci zápisníku („notebook“) DrModelica, což je interaktivní výukový materiál určený k usnadnění studia jazyka Modelica [4]. Interaktivní zápisník sjednocuje programování, dokumentaci k programům a vizualizaci výsledků [4], příklad použití prostředí OMNotebook je na obrázku 1.4.



Obrázek 1.1: OpenModelica Connection Editor - grafický pohled na model



Obrázek 1.2: OpenModelica Connection Editor - textový pohled na model



Obrázek 1.3: OpenModelica Connection Editor - parametry rezistoru

OMC (Advanced Interactive OpenModelica Compiler) je překladač používaný pro překlad modelu z jazyka Modelica do jazyka C.

OMShell (Interactive OpenModelica Shell) je prostředí ve stylu příkazového řádku, které umožňuje interpretaci jazyka Modelica, simulaci modelů, tisk grafů a podobně.

OpenModelica obsahuje i další nástroje (OMOptim, OMPython, atd.), které však nejsou z hlediska této práce důležité. Popis těchto nástrojů je možné nalézt v [5].

V prostředí OpenModelica je možné využít velké množství předdefinovaných objektů z knihoven jazyka Modelica. Jedná se zejména o Modelica Standard Library, což je standardizovaná a volně dostupná knihovna vytvářená společností Modelica Association. Knihovna momentálně obsahuje 1360 modelů či objektů a 1280 funkcí z oblastí mechaniky, magnetismu, matematiky, elektrických zapojení, termálních modelů a dalších. Mimo to je důležitá skupina `Blocks/Interfaces`, která obsahuje vstupní a výstupní objekty pro FMI (`RealInput, RealOutput, IntegerInput, IntegerOutput` ...).

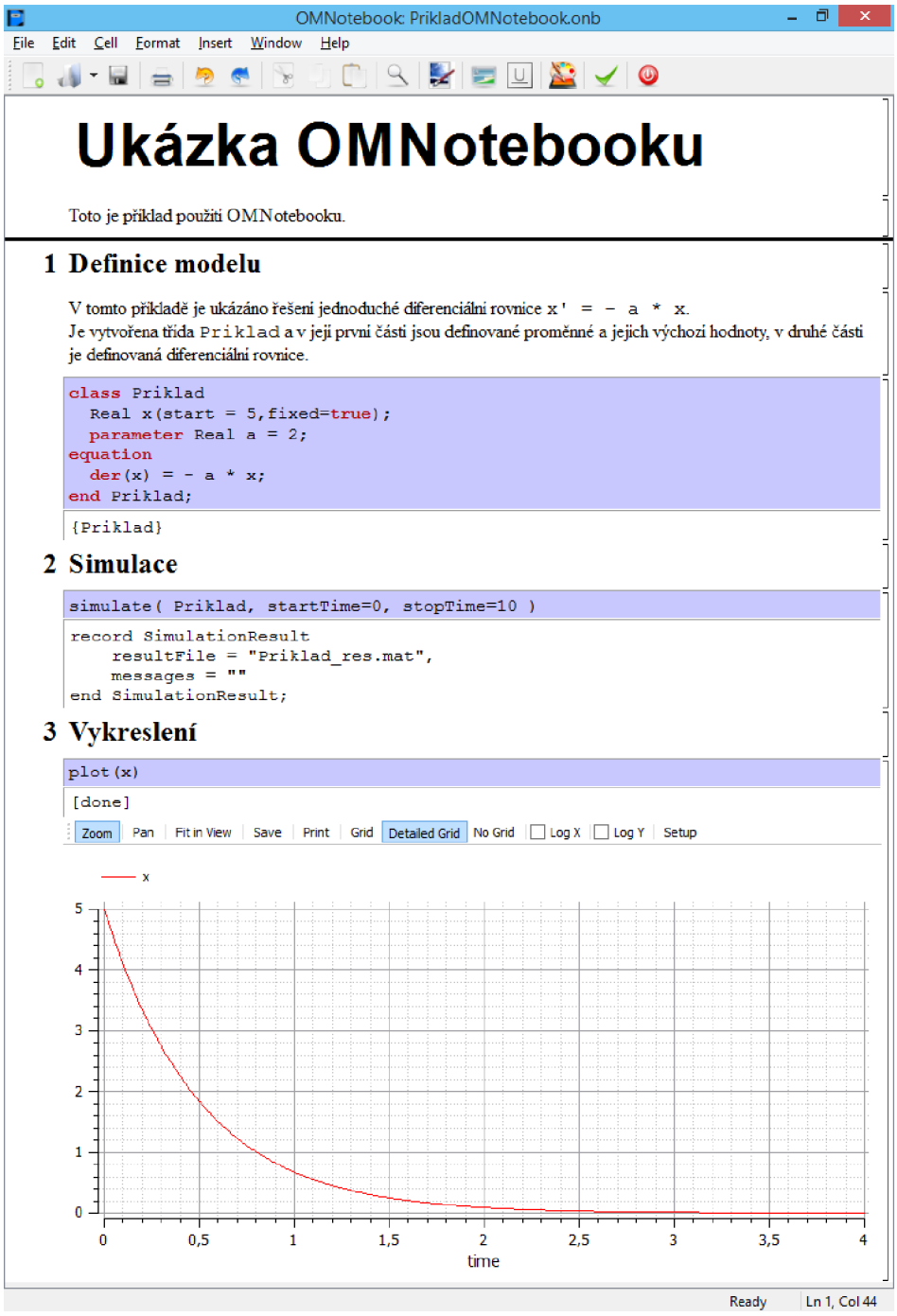
OpenModelica umožňuje exportovat a importovat FMU (viz kapitola 2). Obojí je možné v menu `FMI` v horní liště nástroje OMEdit. Druhá možnost exportu FMU je pomocí nástroje OMShell použitím příkazu

```
1 translateModelFMU(modelName)
```

kde `modelName` je název modelu. Pro možnost interakce FMU při simulace je nutné definovat vstupy a výstupy modelu pomocí objektů dle minulého odstavce.

1.3. Fyzikální modelování

Fyzikální modelování (nebo také akauzální, objektově orientované či deklarativní modelování) je způsob modelování, kdy model a jeho části (objekty) jsou popsány soustavami rovnic, avšak nepopisuje se algoritmus řešení těchto rovnic [6]. Vytvořený model reprezen-



Obrázek 1.4: Příklad zápisníku v programu OMNotebook

tuje fyzikální realitu modelovaného systému. Tento způsob modelování využívá například jazyk Modelica a rozšíření Simscape pro Matlab/Simulink.

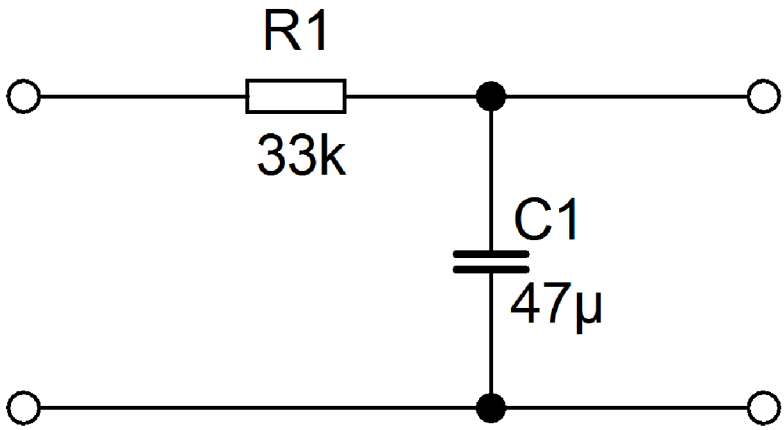
Naopak kauzální modelování (blokově orientované modelování) popisuje příčiny a následky určitých jevů, tedy postup výpočtu rovnic, kterými je model popsán [6]. Model je tvořen z bloků a jejich propojů, pokud je model složitější, tak se může ztrácet fyzikální

podstata systému. Tento způsob modelování využívá například Simulink v Matlabu (ne však Simscape).

Pro porovnání kauzálního a akauzálního modelování byl vybrán RC článek znázorněný na obr. 1.5. RC článek je možné popsat diferenciální rovnicí

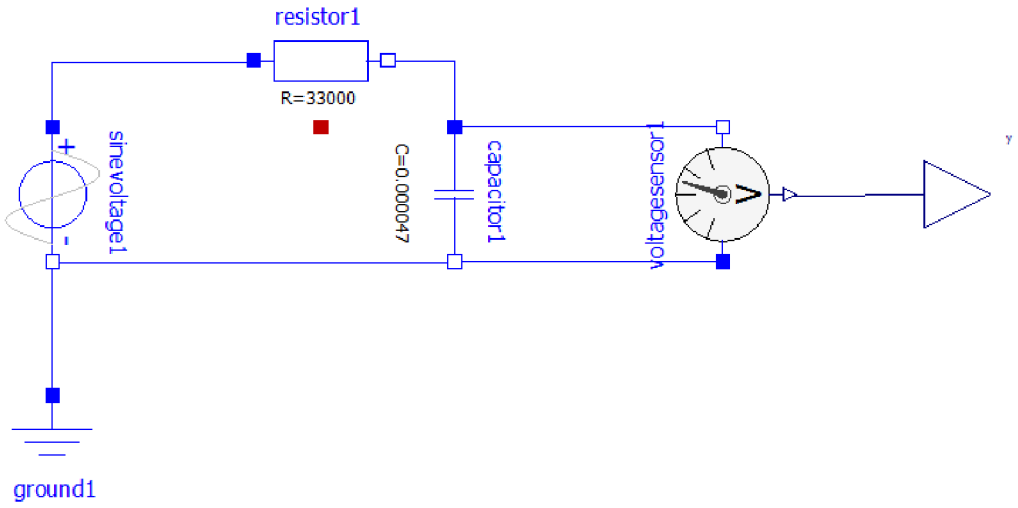
$$\frac{du_C(t)}{dt} = -\frac{1}{RC}u_C(t) + \frac{1}{RC}u(t), \tag{1.1}$$

kde $u_C(t)$ je výstupní napětí RC článku, R je elektrický odpor rezistoru, C je kapacita kondenzátoru a $u(t)$ je vstupní napětí RC článku.

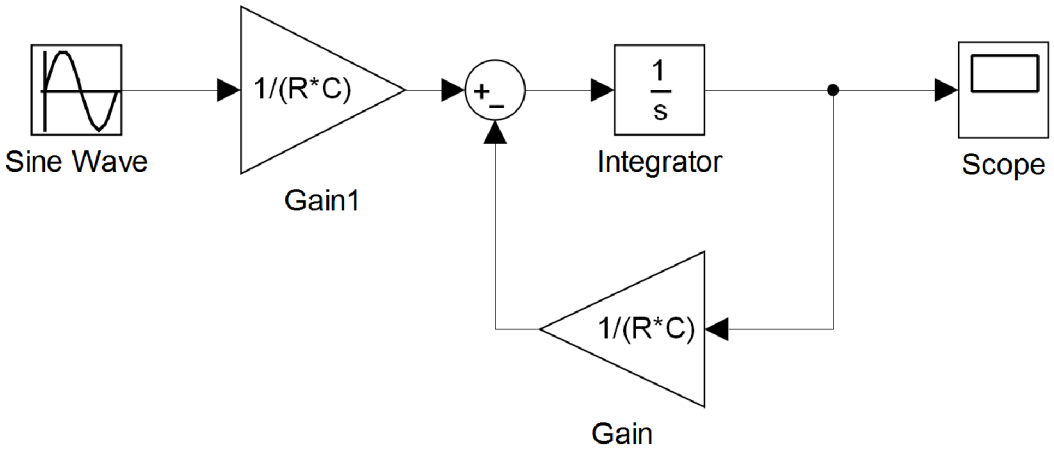


Obrázek 1.5: Schéma RC článku

Na obrázcích 1.6 a 1.7 je možné porovnat akauzální modelování v případě OpenModelicy a kauzální v prostředí Matlab/Simulink. Zapojení v OpenModelice odpovídá reálnému elektrickému zapojení na obr. 1.5. Zapojení v Matlab/Simulink odpovídá diferenciální rovnici tohoto systému (rovnice 1.1).



Obrázek 1.6: Model RC článku v prostředí OpenModelica



Obrázek 1.7: Model RC článku v prostředí Matlab/Simulink

2. Functional Mock-up Interface

Functional Mock-up Interface (FMI) je standard pro výměnu a souběžnou simulaci (co-simulation) dynamických modelů mezi různými nástroji [7]. Celá tato kapitola popisující standard FMI vychází z [8].

Při použití FMI pro výměnu modelů je vygenerován zdrojový kód v jazyku C, který popisuje dynamický systém. Tento kód může být následně volán jiným nástrojem při simulaci.

V případě použití souběžné simulace (co-simulace) je model simulován v původním nástroji a mezi nástroji probíhá předávání jednotlivých hodnot proměnných.

Model je v obou případech uložen nebo exportován jako Functional Mock-up Unit (FMU). Je tvořen jedním zip archívem s příponou „.fmu“ a obsahuje dvě základní části - XML soubor a kód modelu (buď jako zdrojový kód, nebo zkompilovaný do DLL knihovny). Mimo to může být v FMU obsažena ikona modelu, dokumentace, dodatečné DLL knihovny a podobně.

2.1. Rozhraní modelu

2.1.1. Matematický popis modelu

FMI standard pro výměnu modelů je navržen pro řešení systémů diferenciálních, algebraických a diferenčních rovnic. Takový systém bývá označován jako hybridní systém diferenciálních rovnic („hybrid ODE“ - ordinary differential equations).

Tento systém je po částech spojitý, nespojitosti se mohou vyskytovat v časech událostí („events“).

Pro systém rovnic jsou definovány dva stavové vektory:

- $\boldsymbol{x}(t)$ je vektor spojitých stavů a jeho hodnoty jsou spojité mezi událostmi
- $\boldsymbol{m}(t)$ je vektor diskrétních stavů a jeho hodnoty jsou konstantní mezi událostmi.

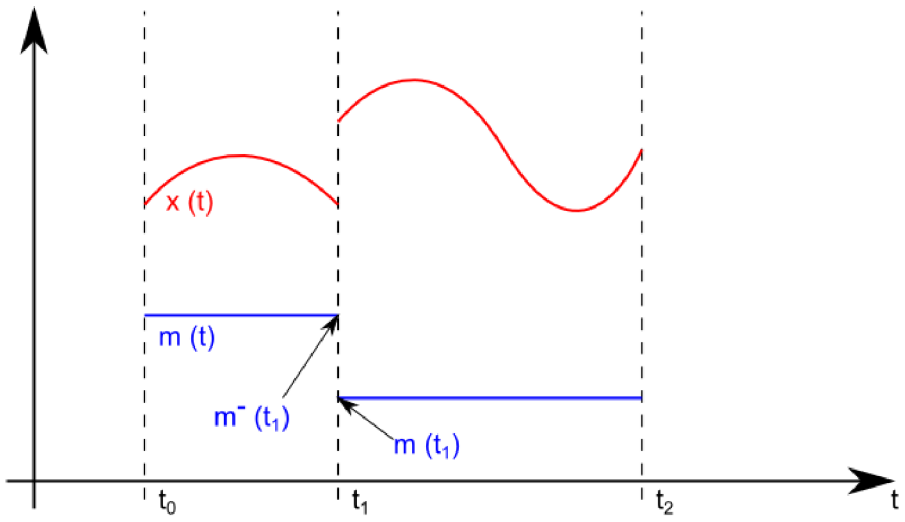
Příklad průběhu stavových vektorů v čase je zachycen na obr. 2.1. V časech t_0, t_1 a t_2 jsou výskyty událostí. Pro případ nespojitosti v čase události mohou mít stavové vektory $\boldsymbol{x}(t_i)$ a $\boldsymbol{m}(t_i)$ v čase t_i dvě hodnoty. Proto mají definované limitní hodnoty zleva $\boldsymbol{x}^-(t_i)$, $\boldsymbol{m}^-(t_i)$ a zprava $\boldsymbol{x}(t_i)$, $\boldsymbol{m}(t_i)$

Dále může model obsahovat další proměnné - parametry, vstupy, výstupy a interní proměnné. Parametry jsou v průběhu simulace (po inicializaci) konstantní, ostatní proměnné se mohou v průběhu simulace měnit.

2.1.2. Funkce modelu

V této kapitole jsou popsány funkce definované standardem FMI. Byly vybrány pouze důležité funkce, popis všech funkcí by přesáhl rozsah této práce.

Většina funkcí modelu má jako návratovou hodnotu výčtový typ `fmiStatus`, který je definován následovně:



Obrázek 2.1: Matematický popis modelu FMU - po částech spojitý systém

```
typedef enum {fmiOK,
              fmiWarning,
              fmiDiscard,
              fmiError,
              fmiFatal} fmiStatus;
```

Významy jednotlivých hodnot jsou:

- `fmiOK` - vše v pořádku
- `fmiWarning` - nastal problém, avšak simulace může pokračovat; byla zavolána funkce „`logger`“
- `fmiDiscard` - je nutné zmenšit vzorkovací periodu
- `fmiError` - nastal závažný problém, simulace nemůže pokračovat, je nutné zavolat funkci `fmiFreeModelInstance()`; byla zavolána funkce „`logger`“
- `fmiFatal` - závažný problém s modelem a všemi jeho instancemi; byla zavolána funkce „`logger`“

Pro vytvoření instance modelu se používá funkce `fmiInstantiateModel(..)`, jejíž deklarace je následující:

```
fmiComponent fmiInstantiateModel(fmiString instanceName,
                                 fmiString GUID,
                                 fmiCallbackFunctions functions,
                                 fmiBoolean loggingOn);
```

kde `instanceName` je název vytvářené instance, `GUID` je jednoznačný identifikátor pro kontrolu kompatibility zdrojových kódů a popisu modelu v xml souboru, `functions` je struktura ukazatelů na funkce pro alokaci a dealokaci paměti a na funkci `logger`, `loggingOn` je proměnná pro zapnutí nebo vypnutí logování pro ladění modelu.

Pro smazání dané instance modelu se používá funkce `fmiFreeModelInstance(..)`, která má jako jediný parametr ukazatel na danou instanci modelu. Funkce provede uvolnění veškeré alokované paměti a dalších zdrojů této instance.

```
1 void fmiFreeModelInstance(fmiComponent c);
```

Nastavení aktuálního času simulace je možné pomocí funkce `fmiSetTime(..)`. Parametrem této funkce je ukazatel na instanci modelu a požadovaný čas.

```
1 fmiStatus fmiSetTime(fmiComponent c, fmiReal time);
```

Funkce `fmiSetContinuousStates(..)` se používá k nastavení nových hodnot stavového vektoru.

```
1 fmiStatus fmiSetContinuousStates(fmiComponent c,
2                                  const fmiReal x[],
3                                  size_t nx);
```

kde `c` je ukazatel na instanci modelu, `x` je vektor nových hodnot stavů a `nx` je počet stavů ve vektoru (pro kontrolu). Pro získání aktuálního stavového vektoru je možné použít funkci `fmiGetContinuousStates(..)`, která má deklaraci shodnou s funkcí pro nastavení hodnot stavového vektoru.

Po provedení integračního kroku musí simulační prostředí zavolat funkci `fmiCompletedIntegratorStep(..)`. Návratovým parametrem `callEventUpdate` je určeno, zda je nutné volat funkci `fmiEventUpdate(..)`, nebo zda není třeba provádět žádnou akci.

```
1 fmiStatus fmiCompletedIntegratorStep(fmiComponent c,
2                                      fmiBoolean* callEventUpdate);
```

Pro zápis hodnot (vstupů, parametrů apod.) do FMU se používají funkce `fmiSetReal`, `fmiSetInteger`, `fmiSetBoolean` a `fmiSetString`. Níže je uvedena deklarace funkce `fmiSetReal`, deklarace ostatních funkcí je velmi podobná, liší se pouze názvem funkce a datovým typem parametru `value`. Parametr `vr` je vektor identifikátorů proměnných, `nvr` je počet prvků ve vektorech `vr` a `value`, `value` je vektor nových hodnot proměnných uspořádaný shodně s `vr`.

```
1 fmiStatus fmiSetReal (fmiComponent c,
2                       const fmiValueReference vr[],
3                       size_t nvr,
4                       const fmiReal value[]);
```

Pro čtení hodnot z FMU (zejména výstupů) se používá sada funkcí `fmiGetReal`, `fmiGetInteger`, `fmiGetBoolean` a `fmiGetString`. Pro ilustraci je níže uvedena deklarace funkce `fmiGetReal`, ostatní funkce se liší pouze názvem funkce a datovým typem parametru `value`. Parametry těchto funkcí jsou shodné s parametry funkcí pro zápis hodnot (viz `fmiSetReal` výše).

```
1 fmiStatus fmiGetReal (fmiComponent c,
2                       const fmiValueReference vr[],
```

```
3                               size_t nvr,
4                               fmiReal value[]);
```

Po dokončení simulace je nutné ji korektně ukončit pomocí funkce `fmiTerminate(..)`, která má pouze jediný parametr `c`, což je ukazatel na instanci modelu.

```
1 fmiStatus fmiTerminate(fmiComponent c);
```

2.2. Popis modelu

Popis modelu je tvořen xml souborem, jehož schéma je definované souborem `fmiModelDescription.xsd`. Může obsahovat následující položky:

- `attributes`
- `UnitDefinitions`
- `TypeDefinitions`
- `DefaultExperiment`
- `VendorAnnotations`
- `ModelVariables`

Povinná je pouze položka `attributes`, která obsahuje vlastnosti modelu, zejména název (`modelName`), identifikátor modelu (`modelIdentifier`), počet spojitých stavů (`numberOfContinuousStates`), počet indikátorů událostí (`numberOfEventIndicators`) a `guid`. Dále může obsahovat nepovinné vlastnosti, například popis (`description`), autora modelu (`author`), nástroj, kterým bylo FMU generováno (`generationTool`) a další.

Další důležitou (avšak nepovinnou) součástí popisu modelu je seznam proměnných (`ScalarVariable`) modelu v položce `ModelVariables`. Každá proměnná musí mít definovaný datový typ (`Real, Integer, Boolean, String` nebo `Enumeration`) a další povinné či nepovinné parametry (název, `valueReference`, variabilitu, kauzalitu apod.). Podle datového typu může mít každá proměnná další dodatečné nepovinné parametry (např. minimální a maximální hodnotu, výchozí hodnotu apod.).

Položka `UnitDefinitions` může obsahovat převodní vztahy mezi jednotkami pro zobrazení a základními jednotkami. Příkladem základní jednotky může být Kelvin, jednotky pro zobrazení potom můžou být °C a °F.

V položce `DefaultExperiment` mohou být definovány výchozí parametry simulace (počáteční čas (`startTime`), konečný čas (`stopTime`) a tolerance). Tyto parametry je možné před simulací upravit.

Položka `VendorAnnotations` obsahuje poznámky k danému FMU. Poznámky přísluší pouze danému nástroji a ostatní nástroje je ignorují, proto tato položka nemá pro výměnu modelů význam.

Datové typy je možné definovat v položce `TypeDefinitions`, každý datový typ musí mít název (`name`) a základní datový typ (`RealType, IntegerType, BooleanType,`

`StringType` nebo `EnumerationType`). Podle základního datového typu může mít každý datový typ další parametry (např. jednotky, minimální nebo maximální hodnotu a výčet prvků pro výčtový typ).

Příklad popisu modelu je uveden jako příloha 1 této práce.

3. Návrh nástroje FMUtoolbox

FMUtoolbox je softwarový nástroj doplňující funkcionalitu prostředí Matlab/Simulink o možnost simulace FMU. Je zaměřen na výměnu modelů, neřeší souběžnou simulaci (co-simulaci).

3.1. Import modelů v jazyce Modelica do Matlab/Simulink

Matlab/Simulink nemá přímou podporu pro modely v jazyce Modelica a nepodporuje import ani co-simulaci FMU. Pro použití modelů v jazyce Modelica v Matlabu je nutné použít některý nástroj, který tuto funkcionalitu doplní.

Jednou z možností je použití programu Dymola, který podporuje export modelu jako C-Mex S-funkci [9]. Nevýhodou tohoto řešení je závislost na komerčním software jednoho výrobce (ceny licencí, nemusí být podporovány modely z jiných nástrojů apod.).

Další možností je použití FMI Toolboxu od firmy Modelon, což je doplněk pro Matlab/Simulink obsahující knihovnu bloků pro Simulink. Toolbox umožňuje import a export FMU v Simulinku a co-simulaci FMU [10]. Nevýhodou tohoto řešení je licenční politika výrobce (roční licence na jeden počítač za vysokou cenu), chybějící podpora sběrnic („bus“) v Simulinku a omezená podpora FMU generovaných jinými nástroji (nástroj podporuje pouze FMU exportované nástroji Dymola, JModelica.org a SimulationX a to pouze na platformě `win32`).

Protože ani jedna z možností nebyla shledána jako vyhovující, je nutné vytvořit nový nástroj, který bude podporovat veškerou požadovanou funkcionalitu a zároveň bude vyhovující z pohledu licenční politiky. Pro nástroj byl zvolen název „FMUtoolbox pro Simulink“ („FMUtoolbox for Simulink“).

Po rozsáhlém průzkumu možností importu modelů v jazyce Modelica do Matlab/Simulink bylo rozhodnuto, že nástroj FMUtoolbox bude podporovat simulaci FMU s využitím DLL souboru obsaženého v FMU. Nevýhodou tohoto přístupu jsou možné problémy s kompatibilitou (32-bit FMU je možné použít pouze na 32-bit počítači s 32-bit Matlabem), avšak většina FMU neobsahuje zdrojové kódy modelů, takže tento přístup je jediný možný.

Byla zvažována i možnost překladu modelu v jazyce Modelica do jazyka C a následné simulace v rámci S-funkce, avšak vzhledem ke komplexnosti jazyka Modelica a k možným problémům s rozhraním modelu byla tato možnost zamítnuta. Jednalo by se o shodný přístup jako je popsán ve standardu FMI, avšak řešení by nebylo standardizované.

3.2. Návrh funkcionality

Tato kapitola popisuje požadavky na nástroj FMUtoolbox z hlediska funkcionality.

Blok Simulink

Pro možnost simulace FMU v modelu Simulinku je třeba vytvořit novou knihovnu. Tato knihovna bude obsahovat blok, případně bloky pro simulaci FMU. Blok bude mít proměnný počet vstupů a výstupů, při jeho otevření se zobrazí grafické uživatelské rozhraní (GUI).

Porty bloku Simulink

Kvůli zjednodušení práce s blokem v Simulinku bude možné volně nastavit vstupní a výstupní porty bloku. Počet portů může odpovídat počtu vstupů či výstupů nebo nemusí. Každý port může být typu `individual`, `mux` nebo `bus`. Typ `individual` určuje port ke spojení se skalárním signálem, typ `mux` ke spojení s vektorem hodnot a `bus` určuje port k připojení na sběrnici.

Na vstupu i výstupu bude možná libovolná kombinace portů různých typů. Nastavení portů a příslušnosti jednotlivých vstupů a výstupů do portů bude možné provést v grafickém rozhraní, aby nebylo nutné používat bloky typu `Mux`, `Demux`, `Bus Creator` a podobně.

U každého portu bude možné nastavit vlastní uživatelský název, který bude zobrazen v bloku Simulinku. Názvy bude možné interpretovat jako LaTeX zápis, takže v bloku Simulinku bude možné zobrazit indexy (U_1), symboly (α) apod.

Simulace z příkazové řádky Matlabu

Pro opakované simulace nebo pro automatické ladění či testování je navrženo ovládání FMUtoolboxu pomocí příkazové řádky (tím je myšleno jak ovládání z konzole Matlabu, tak i pomocí m-file).

Pomocí příkazové řádky musí být možné vybrat požadované FMU (jeho umístění v počítači), nastavit parametry simulace a parametry FMU, vložit vstupní signály, aktivovat kteroukoliv proměnnou jako výstup a následně získat výstupní data z FMU.

Nastavení parametrů FMU

Hodnotu všech parametrů FMU (tzn. proměnných s variabilitou `parameter`) musí být možné před simulací nastavit. Je navrženo jako výchozí hodnotu parametrů použít atribut `start` z popisu modelu v souboru xml, aby měl uživatel přehled nad nastavenými hodnotami parametrů. Hodnoty parametrů musí být možné změnit jak pomocí grafického rozhraní (GUI), tak i pomocí příkazové řádky Matlabu.

Ukládání modelů v Simulinku

Je navrženo, aby veškeré uživatelské nastavení FMU a bloku v Simulinku bylo možné uložit do modelu Simulinku (jako u standardních bloků Simulinku). Jedná se zejména o hodnoty parametrů, nastavení portů bloku a další nastavení. Po uložení a následném načtení modelu Simulinku by měla simulace proběhnout shodně se simulací provedenou před uložením.

Multiplatformní použití

Každé FMU je určeno pro jednu nebo více platforem. Cílovou platformu určuje nástroj, kterým je FMU vygenerováno (tzn. pokud je FMU generováno pro použití např. na platformě `win32`, je možné jej simulovat pouze na této platformě, na ostatních platformách simulace tohoto FMU není možná).

FMUtoolbox musí podporovat platformy Windows 32-bit (`win32`), Windows 64-bit (`win64`) a Linux 64-bit (`linux64`). Možnost simulace konkrétního FMU nezávisí na nástroji FMUtoolbox, je určena nástrojem, který FMU exportoval. Nástroj FMUtoolbox nerealizuje možnost simulace FMU určených pro odlišné platformy (tzv. cross-platformní simulaci), pro možnost simulace konkrétního FMU musí být shodná platforma operačního systému, prostředí Matlab a platformy FMU (např. možná je kombinace Windows 32-bit, Matlab 32-bit a FMU 32-bit).

Logování

Pro případ řešení problémů musí být k dispozici možnost aktivovat logování pro ladění („debug logging“).

Simulace více FMU

FMUtoolbox musí umožňovat simulaci více instancí FMU současně (a to jak instancí stejného modelu, tak i různých modelů). Parametry každé instance FMU musí být možné individuálně nastavit, aby bylo možné souběžně simulovat chování modelu za různých podmínek (např. v závislosti na okolní teplotě apod.). Dále musí být možné bloky FMUtoolboxu navzájem propojit v modelu Simulink - např. výstup jednoho FMU použít jako vstup jiného FMU.

Kopírování bloků FMUtoolbox

Pro zjednodušení práce s bloky FMUtoolboxu v modelu Simulink bude možné blok standardními metodami (Ctrl+C apod.) zkopírovat a vložit. Nový blok bude mít veškerá nastavení a parametry shodné s původním blokem, po vytvoření kopie bude možné parametry či nastavení nového bloku změnit.

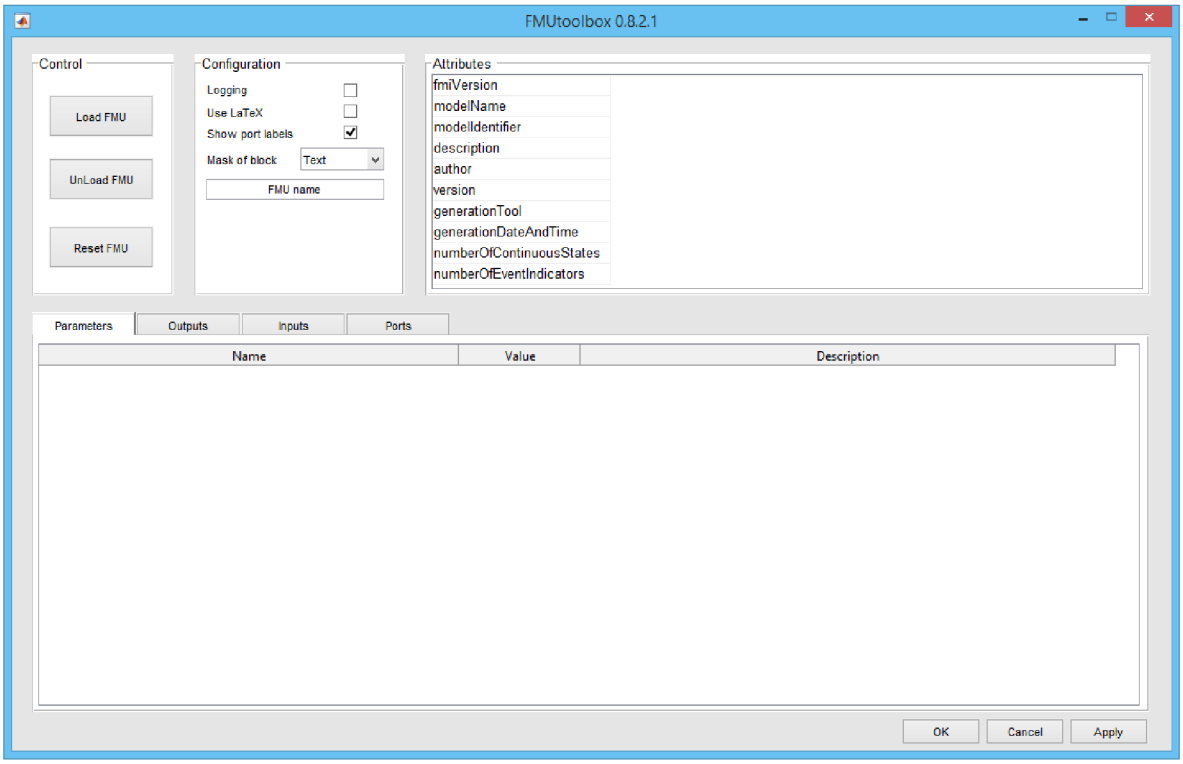
Instalátor

Bude vytvořen instalační program, který zajistí zkopírování potřebných souborů a provedení konfigurace nutné pro fungování FMUtoolboxu (zejména nastavení Matlab Set Path apod.).

3.3. Návrh uživatelského rozhraní

Pro nastavení parametrů a dalších možností pro simulaci FMU je navrženo uživatelské rozhraní (GUI). Návrh vychází z vlastností standardu FMI a jsou zohledněny požadavky

na uživatelská nastavení. Do uživatelského rozhraní lze také zahrnout blok Simulinku (zejména popisky portů, vzhled bloku apod.).



Obrázek 3.1: Grafické rozhraní nástroje FMUtoolbox

Na obrázku 3.1 je zobrazen návrh uživatelského rozhraní. Okno je rozděleno do několika sekcí: ovládání (Control), nastavení (Configuration) a atributy (Attributes). Poslední sekcí je spodní polovina okna, kde je možné přepínat mezi parametry (Parameters),výstupy (Outputs), vstupy (Inputs) a porty (Ports).

V sekci ovládání je možné načíst FMU do Matlabu, smazat FMU z Matlabu nebo znovu načíst FMU - tím se vyresetují veškerá nastavení daného FMU do výchozí konfigurace.

V sekci nastavení je možné pomocí checkboxu zapnout logování (debug logging) pro dané FMU a pomocí druhého checkboxu zapnout či vypnout zobrazování popisku portů bloku v Simulinku pomocí LaTeX syntaxe. Zaškrtnutím checkboxu dojde k překladu zdrojového kódu LaTeX do formátovaného výstupu (např. `\alpha` se zobrazí jako α). Názvy proměnných v FMU mohou obsahovat „_“ (podtržítko), v tom případě je vhodné tento checkbox vypnout a názvy proměnných se budou zobrazovat korektně (např. místo chybného „temperature$_s$ensor“ se bude zobrazovat „temperature_sensor“).

Sekce atributy obsahuje informace o FMU, které je možné vyčíst z popisu model v xml souboru. Tyto informace nejsou povinné, takže některá pole mohou být prázdná. Příklad atributů je na obrázku 3.2. Atribut `fmiVersion` určuje verzi FMI standardu, `modelName` je název modelu, `modelIdentifier` je identifikátor modelu, `description` je popis modelu, `author` je autor modelu, `version` je verze modelu, `generationTool` je označení nástroje, kterým bylo FMU vygenerováno v čase `generationDateAndTime`,

Attributes

fmiVersion	1.0
modelName	ControlledTemperature
modelIdentifier	ControlledTemperature
description	Control temperature of a resistor
author	
version	
generationTool	Dymola Version 2014 FD01 (64-bit), 2013-10-17
generationDateAndTime	2013-11-29T14:59:42Z
numberOfContinuousStates	1
numberOfEventIndicators	6

Obrázek 3.2: Příklad sekce atributy v GUI FMUtoolboxu

`numberOfContinuousStates` je počet spojitých stavů modelu a `numberOfEventIndicators` je počet indikátorů událostí.

V sekci parametrů je možné nastavit parametry modelu. Příklad načtených parametrů je na obr. 3.3. V prvním sloupci tabulky je název proměnné, v druhém sloupci je aktuální

Parameters

Name	Value	Description
TAmb	293.1500	Ambient Temperature
TDif	2	Error in Temperature
constantVoltage.V	10	Value of constant voltage
constantVoltage.p.v		Potential at the pin
heatCapacitor.C	1	Heat capacity of element (= cp*m)
heatingResistor.R_ref	10	Resistance at temperature T_ref
heatingResistor.T_ref	293.1500	Reference temperature
heatingResistor.alpha	0.0039	Temperature coefficient of resistance (R = R_ref*(1 + alpha*(heatPort.T - T_ref))
heatingResistor.T		Fixed device temperature if useHeatPort = false
fixedTemperature.T		Fixed temperature at port
thermalConductor.G	0.1000	Constant thermal conductance of material
idealSwitch.Ron	1.0000e-05	Closed switch resistance
idealSwitch.Goff	1.0000e-05	Opened switch conductance
ramp.height	25	Height of ramps
ramp.duration	6	Duration of ramp (= 0.0 gives a Step)
ramp.offset	25	Offset of output signal

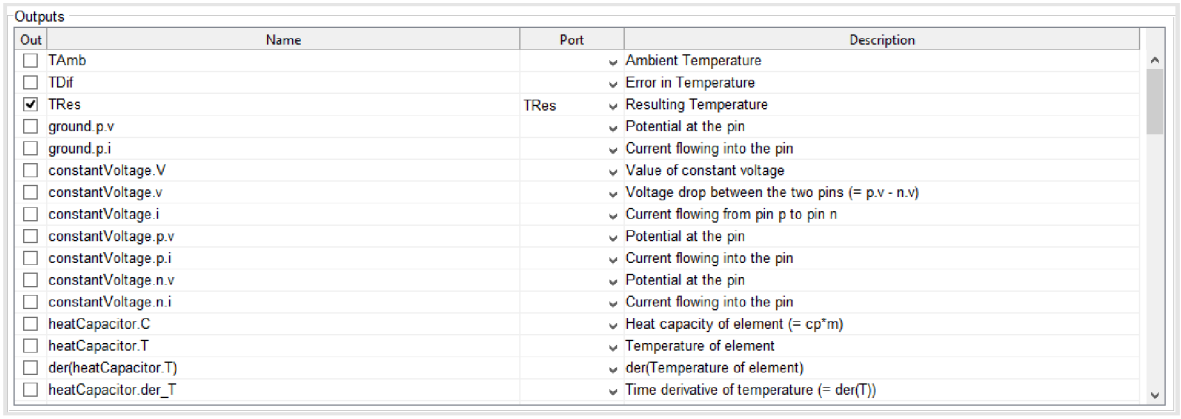
Obrázek 3.3: Nastavení parametrů FMU

hodnota parametru (kterou je možné změnit) a ve třetím sloupci je popis proměnné. Výchozí hodnoty parametrů jsou načítány z xml souboru s popisem modelu.

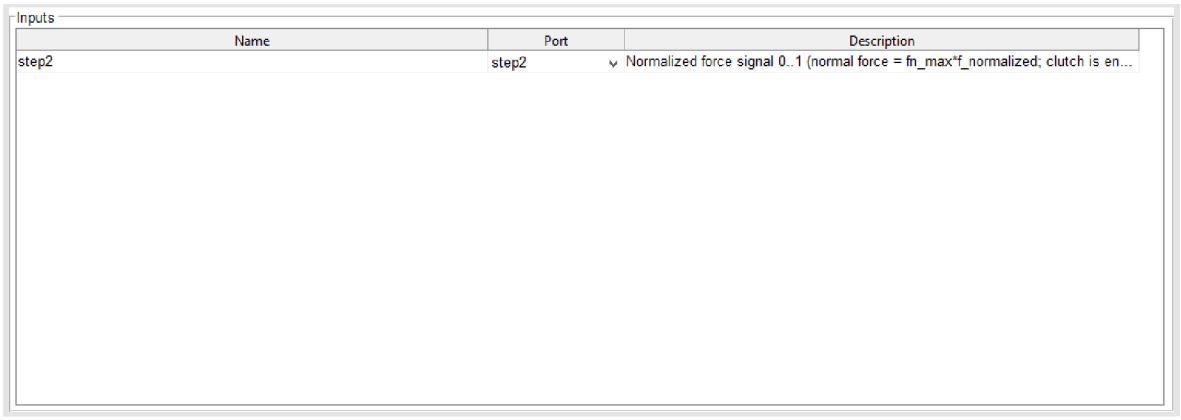
Sekce výstupů obsahuje tabulkový výpis všech proměnných modelu mimo vstupy. Příklad tabulky výstupů je na obr. 3.4. Sloupec `Name` obsahuje název proměnné, sloupec `Port` obsahuje název portu, do kterého je proměnná přivedena a sloupec `Description` obsahuje popis proměnné. Proměnné s kauzalitou `output` jsou po načtení FMU přiřazeny do příslušných portů (tedy jsou přivedeny na výstupy z bloku v Simulinku). Ostatní proměnné je možné přivést na výstup bloku vybráním portu, do kterého má být daná proměnná přivedena.

V sekci vstupů je tabulka obsahující všechny proměnné s kauzalitou `input`, její příklad je na obrázku 3.5. Sloupec `Name` obsahuje název proměnné, sloupec `Port` definuje port, ze kterého má být přiveden signál do této proměnné a `Description` obsahuje popis proměnné.

V sekci `Ports` je možné změnit nastavení portů, přidat nové nebo odebrat stávající. Sloupec `Name` obsahuje název portu, ten je možné změnit po kliknutí na stávající název.



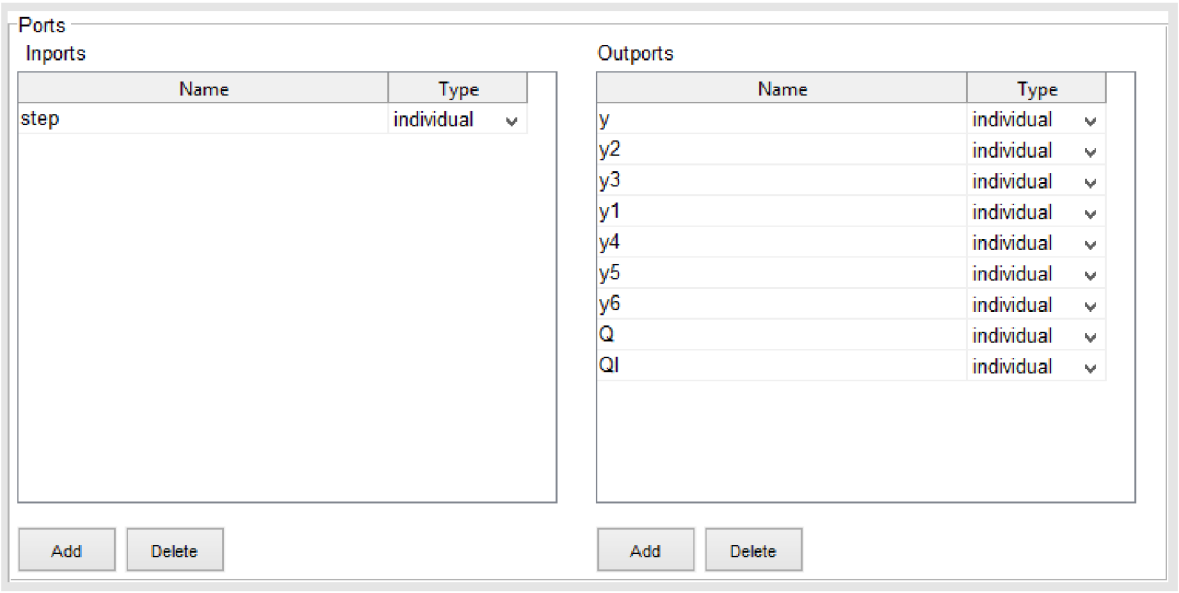
Obrázek 3.4: Příklad tabulky výstupů v GUI FMUtoolboxu



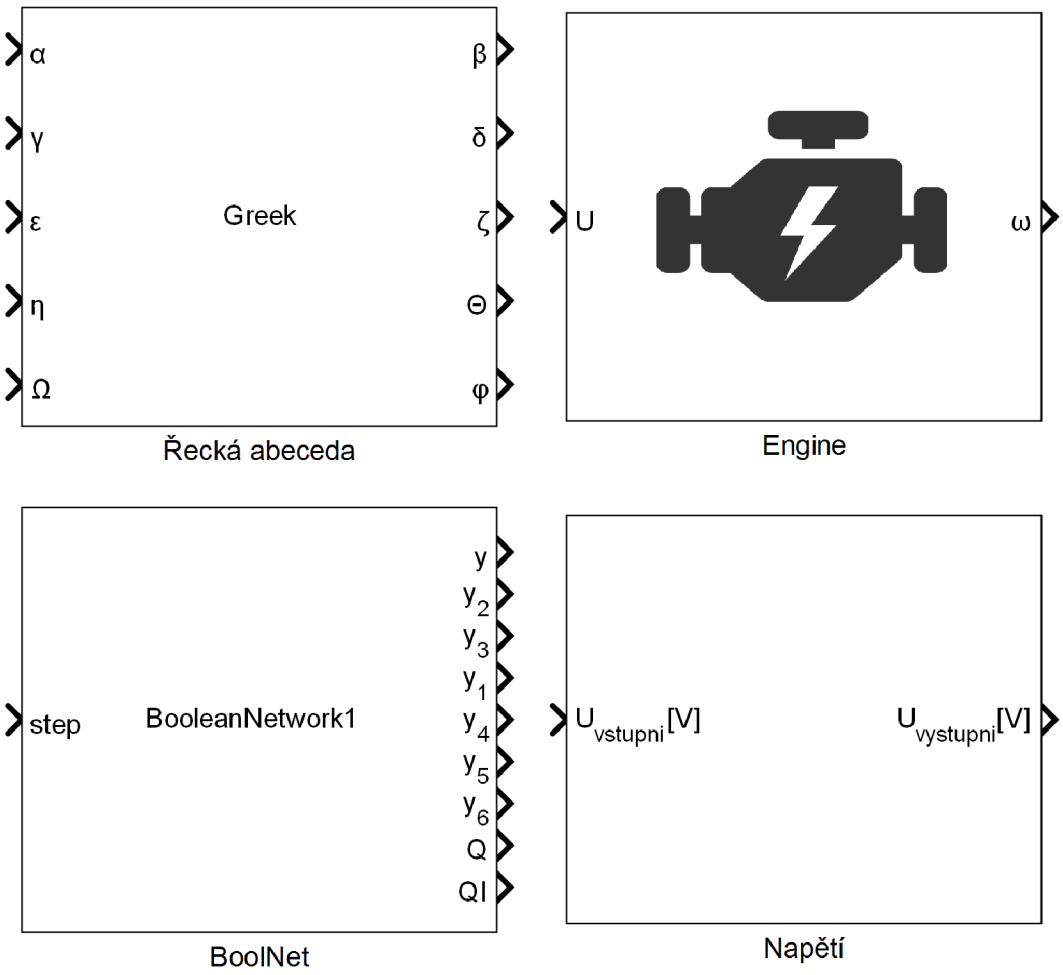
Obrázek 3.5: Příklad tabulky vstupů v GUI FMUtoolboxu

Sloupec `Type` může nabývat následujících hodnot: `individual`, `mux` a `bus`. Popis významu těchto hodnot je vysvětlen v kapitole 3.2. Tlačítkem `Add` je možné přidat nový port, tlačítkem `Delete` je možné odstranit vybrané porty. Příklad konfigurace portů je na obr. 3.6.

Další součástí uživatelského rozhraní je vzhled bloku v Simulinku. Pomocí dříve popsaných možností v uživatelském rozhraní je možné vzhled bloku upravit například do některé z podob viz obrázek 3.7. Zejména se jedná o změnu názvu bloku, zobrazení ikony bloku a různé možnosti popisu vstupních a výstupních portů bloku.



Obrázek 3.6: Příklad konfigurace portů v FMUtoolboxu

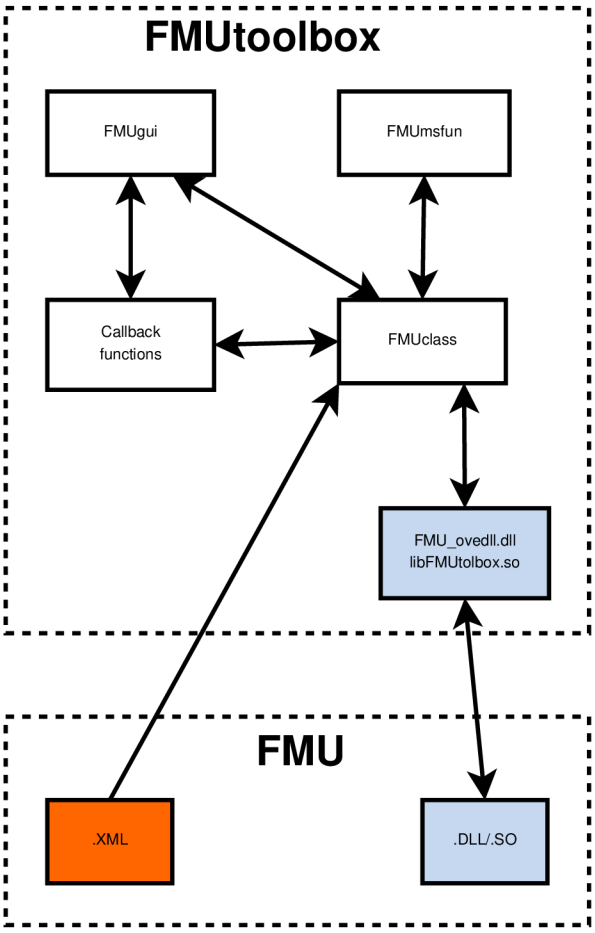


Obrázek 3.7: Příklad možných vzhledů bloků FMUtoolboxu

4. Realizace nástroje FMUtoolbox

Nástroj FMUtoolbox byl vytvořen jako doplněk prostředí Matlab/Simulink. Pro jeho tvorbu bylo použito několik programovacích jazyků, největší část je napsána přímo v jazyce Matlab, dále je využito jazyka C++ a několika dalších (bash script, NSIS apod.).

Celou realizaci FMUtoolboxu je možné rozdělit do několika částí. První z nich je třída `FMUclass`, která zajišťuje načtení FMU a podpůrné metody pro simulaci FMU. Druhou částí je dynamická knihovna (dynamic-link library pro Windows, shared object pro Linux), která zajišťuje rozhraní mezi třídou `FMUclass` a FMU. Třetí částí je S-Funkce, která zajišťuje simulaci FMU v Simulinku. Čtvrtou částí FMUtoolboxu je grafické uživatelské rozhraní, které je vytvořeno v jazyce Matlab. Poslední částí jsou podpůrné skripty pro instalaci, zvyšování verze FMUtoolboxu a další. Blokové schéma FMUtoolboxu je na obrázku 4.1.



Obrázek 4.1: Blokové schéma nástroje FMUtoolbox

4.1. Třída FMUclass

Pro reprezentaci FMU v prostředí Matlab/Simulink byla vytvořena třída `FMUclass`. Objekt této třídy odpovídá načtené instanci FMU. Pomocí tohoto přístupu je zajištěna možnost souběžné simulace více instancí FMU (a to jak různých modelů, tak i shodných, tzn. několikanásobné načtení jednoho FMU).

Při vytváření třídy `FMUclass` byl použit manuál `Matlab Object-Oriented Programming` [11], ze kterého vychází i text této kapitoly.

Základní struktura třídy `FMUclass` je uvedena na následujících řádcích.

```
1 classdef FMUclass < handle
2     properties
3         ...
4     end % properties
5     methods
6         ...
7     end % methods
8 end % classdef
```

Na prvním řádku je definován název třídy a určena nadřazená třída („superclass“), od které je třída odvozena. Následují vlastnosti a metody třídy. V prostředí Matlab může být třída typu `value` nebo `handle`, toto je určeno nadřazenou třídou.

Objekt třídy typu `value` je pevně svázán s proměnnou, do které byl přiřazen při vytváření konstruktorem. Při přiřazení dané proměnné do jiné proměnné je vytvořena kopie objektu třídy.

Naopak při vytváření objektu třídy typu `handle` je do proměnné přiřazen pouze ukazatel na objekt třídy. Při přiřazení dané proměnné do jiné proměnné je pouze zkopírován ukazatel na danou instanci třídy (v obou proměnných je ukazatel na shodný objekt a není vytvářena kopie objektu).

Třída `FMUclass` je typu `handle`, funkcím mimo třídu je předáván ukazatel na danou instanci třídy a tím je zajištěna integrita objektu (všechny části kódu používají aktuální verzi objektu, nejsou vytvářeny žádné kopie objektu). Použití třídy typu `value` nebylo v tomto případě vhodné, bylo by nutné předávat funkcím objekt třídy hodnotou a stejným způsobem ji vracet, zároveň by bylo nutné zajišťovat aktuálnost objektu.

Vlastnosti třídy jsou uspořádány do několika bloků, tím je umožněno řízení přístupu a viditelnosti daných vlastností. Pomocí modifikátoru `SetAccess` je možné nastavit pravidla pro přístup k dané vlastnosti třídy pro zápis. Modifikátor může nabývat hodnot `public`, `protected`, `private` a `immutable`. Veřejné vlastnosti (`public`) je možné nastavovat bez omezení, chráněné vlastnosti (`protected`) jsou přístupné k zápisu z dané třídy nebo jejích podtříd, privátní vlastnosti jsou přístupné k zápisu pouze z dané třídy a neměnitelné vlastnosti (`immutable`) je možné nastavit pouze v konstruktoru dané třídy. Výchozím nastavením vlastností je možnost `public`. Modifikátorem `Hidden` je možné určit, zda dané vlastnosti třídy mají být viditelné pro uživatele. Tento modifikátor může nabývat binárních hodnot, pro `true` je daná vlastnost skrytá, pro `false` je daná vlastnost viditelná. Výchozí hodnota tohoto modifikátoru je `false`. V jazyce Matlab existují i další modifikátory vlastností, ty však nebyly při vytváření třídy použity. Na následujících řád-

cích je příklad skryté vlastnosti `actualTime` s možností nastavení hodnoty pouze z dané třídy či její podtřídy.

```
properties ( Hidden = true, SetAccess = protected )
    actualTime = 0
end % properties
```

Zároveň je pro tuto vlastnost nastavena výchozí hodnota (v tomto případě na „0“), takže po zavolání konstruktoru třídy má vlastnost přiřazenu definovanou hodnotu až do okamžiku změny hodnoty pomocí některé z metod třídy.

Metody třídy jsou také rozděleny do několika skupin. Pomocí modifikátoru `Access` je určeno, který kód může danou metodu zavolat. Možné hodnoty tohoto modifikátoru jsou `public` (neomezený přístup), `protected` (možnost volání pouze z metod dané třídy nebo jejích podtříd), `private` (možnost volání pouze z metod dané třídy) a také je možné specifikovat seznam tříd, jejichž metody mohou danou metodu volat. Výchozí hodnota modifikátoru `Access` je `public`. Pomocí modifikátoru `Hidden` je možné určit, zda má být daná metoda pro uživatele viditelná. Možné hodnoty tohoto modifikátoru jsou `true` a `false`, přičemž pro hodnotu `true` je daná metoda skrytá a pro hodnotu `false` je daná hodnota viditelná pro uživatele. Výchozí hodnota modifikátoru `Hidden` je `false`. Modifikátorem `Static` je možné určit, zda má být daná metoda statická (tzn. metoda nezávisí na objektu třídy a nevyžaduje jej jako argument metody). Modifikátor nabývá binárních hodnot (pro `true` je metoda statická, pro `false` metoda statická není) a výchozí hodnota modifikátoru je `false`. Na následujících řádcích je příklad skryté chráněné metody (tzn. uživatel metodu nemůže vidět a je možné ji volat pouze metodami dané třídy či její podtřídy).

```
methods (Hidden, Access = protected)
    function pcInfo ( this )
        ...
        ...
        ...
    end % function pcInfo
end % methods (Hidden, Access = protected)
```

Statické metody jsou využívány zejména pro definici struktur, protože Matlab neumožňuje definici vlastních datových typů. Návratovou hodnotou těchto funkcí je prázdná struktura, která může být použita v různých částech kódu. Tím je zajištěno, že struktura bude mít vždy stejná pole a zároveň se definice struktury vyskytuje pouze na jednom místě.

Zvláštním případem metody je konstruktor objektu třídy, což je metoda s názvem shodným s názvem třídy. Na tuto metodu jsou další požadavky, které jsou detailně popsány v [11].

```
classdef FMUclass < handle
    methods
        function this = FMUclass( .. )
            ...
            ...
        end % function this = FMUclass
```

```
7       end % methods
8   end % classdef
```

Druhým zvláštním případem metody je destruktor objektu třídy, což je metoda s názvem `delete`. Tato metoda musí mít jako vstupní parametr objekt dané třídy a nesmí mít žádné jiné vstupní a výstupní parametry. Další požadavky na destruktor jsou definovány v [11].

```
1   classdef FMUclass < handle
2       methods
3           function delete(this)
4               ...
5               ...
6           end % function delete
7       end % methods
8   end % classdef
```

Pro nastavení hodnot některých vlastností třídy jsou používány tzv. „setter“ metody třídy. Jedná se zejména o ty vlastnosti, které musejí být nastavovány z kódu mimo třídu `FMUclass` a nebo je při jejich nastavení nutno kontrolovat přípustné hodnoty těchto vlastností. Příklad „setter“ metody je na následujících řádcích.

```
1   methods
2       function set.actualTime ( this, aTime )
3           ...
4           % kontroly parametru a dalsi potrebne kroky
5           ...
6           this.actualTime = aTime;
7       end % function set.actualTime
8   end % methods
```

Setter metody musí být umístěny do bloku `methods` bez jakýchkoliv modifikátorů. Přístup (čtení, zápis) k dané vlastnosti se řídí modifikátory použitými v sekci `properties`. Nastavení hodnoty dané vlastnosti se provádí standardním přiřazením hodnoty do vlastnosti, jak je uvedeno na následujícím příkladu.

```
1   ControlledTemperature.actualTime = 5;
```

`ControlledTemperature` je název konkrétní instance třídy `FMUclass`. Při provedení tohoto příkazu se automaticky zavolá metoda `set.actualTime` a provede se veškerý kód uvedený v této metodě.

Podobným způsobem lze realizovat i „getter“ metody pro vlastnosti třídy, avšak při vytváření nástroje `FMUtoolbox` nebyly „getter“ metody použity, jejich popis je uveden v [11].

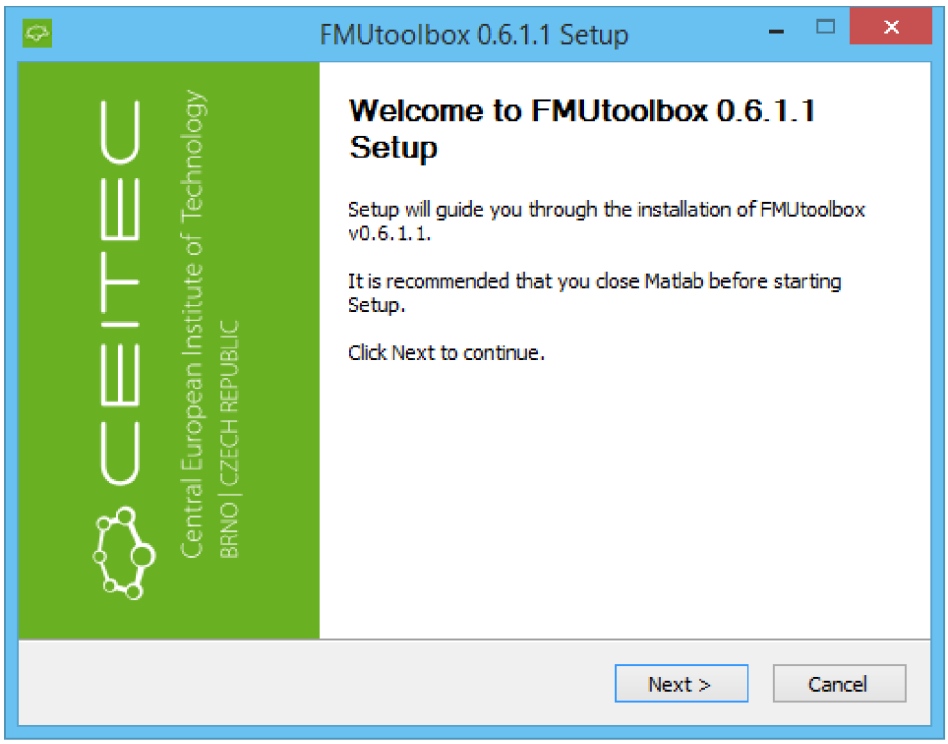
4.2. Instalace FMUtoolboxu

Pro instalaci nástroje FMUtoolbox byly vytvořeny instalátory pro operační systémy Microsoft Windows a pro Linux. Oba instalátory vyžadují administrátorská práva, v operačním

systému Windows je uživatel vyzván pomocí Řízení uživatelských účtů. V operačním systému Linux je nutné instalaci spustit pod uživatelem root (např. pomocí příkazu `sudo`).

Instalační program pro Microsoft Windows byl vytvořen pomocí Nullsoft Scriptable Install System (NSIS), což je open-source systém pro vytváření instalátorů v operačním systému Windows.

NSIS využívá pro tvorbu instalátoru skript, který je možné vytvořit v jakémkoliv textovém editoru. Pro vytvoření skriptu instalátoru nástroje FMUtoolbox byl využit editor PSPad, který umožňuje i kompilaci instalátoru (pomocí klávesové zkratky Ctrl+F9). Výstupem po kompilaci je spustitelný soubor s příponou „.exe“. Instalátor má grafické uživatelské rozhraní, které odpovídá standardním instalačním programům v prostředí Windows. Úvodní obrazovka instalátoru je na obrázku 4.2. Průběh instalace z uživatelského pohledu je popsán v návodu, který je umístěn v příloze této práce.



Obrázek 4.2: Úvodní obrazovka instalátoru nástroje FMUtoolbox

Skript byl vytvořen dle dokumentace k systému NSIS [12]. Mimo standardní funkcionalitu je před zahájením instalace provedena kontrola, zda není Matlab spuštěn. Pokud by byl spuštěn, mohlo by to ohrozit úspěšnost instalace a existuje riziko ztráty dat (v případě otevřených neuložených souborů). Postup při kontrole je následující:

1. Vyhledání procesu „matlab.exe“ - pokud je nalezen, pokračuje se dalšími kroky, v opačné případě je tato část kódu přeskočena.

2. Aktivace dialogového okna, uživatel je vyzván k ukončení Matlabu nebo zrušení instalace.

3. V případě stisknutí tlačítka „Retry“ se znovu provede kontrola, zda je Matlab spuštěn.

4. V případě stisknutí tlačítka „Cancel“ dojde k ukončení instalace.

Realizace této kontroly v jazyce NSIS je uvedena na následujících řádcích.

```
# check if Matlab is running
  Matlab_Running_check_install_start:
    # is matlab.exe process running? store result into $R0
    ${nsProcess::FindProcess} "matlab.exe" $R0

    # 0 means success - process found
    ${If} $R0 == 0
      MessageBox MB_RETRYCANCEL|MB_ICONEXCLAMATION \
      "Please close Matlab before continuing." \
      IDCANCEL false IDRETRY true
      true:
        GoTo Matlab_Running_Check_install_start
      false:
        MessageBox MB_OK "Installation terminated."
        Quit
    ${EndIf}
```

Další zajímavou částí instalátoru je provedení úpravy Matlab Path, což je provedeno pomocí Matlab skriptu, který je spuštěn z instalátoru pomocí následujícího kódu.

```
# Edit Matlab pathdef - start Matlab and run mfile

Var /GLOBAL MatlabAddPath

strcpy $MatlabAddPath \
"$\"$MatlabDir\bin\matlab.exe$\" -nodisplay -nosplash -nodesktop \
-minimize -r $\"run($\'$INSTDIR\FMUaddpath.m');exit;$\""

nsExec::Exec $MatlabAddPath

# check if Matlab is running - it takes a quite long time to start it
# and execute few commands
Matlab_Running_check_install:
    # is matlab.exe process running? store result into $R0
    ${nsProcess::FindProcess} "matlab.exe" $R0

    # 0 means success - process found
    ${If} $R0 == 0
        Sleep 1000
        GoTo Matlab_Running_Check_install
    ${EndIf}
```

Tato část kódu spustí Matlab a v něm zavolá skript „FMUaddpath.m“, který zajistí doplnění prohledávaných cest Matlabu o potřebné cesty a po provedení skriptu se Matlab ukončí. Instalátor následně periodicky kontroluje, zda již je Matlab ukončen a poté může instalace pokračovat.

Instalátor pro operační systém Linux byl vytvořen jako Bash Self-Extracting Script. Instalátor nemá grafické uživatelské rozhraní, jeho ovládání probíhá pomocí příkazové řádky. Pro vytvoření instalátoru se používá skript „createInstaller.sh“, který z potřebných souborů a ostatních skriptů vytvoří soubor s příponou „bsx“. Po spuštění instalátoru se nejdříve spustí skript „decompress.sh“, který zajistí extrakci souborů do dočasné složky a spuštění skriptu „installer.sh“, který provede vlastní instalaci.

Stejně jako u instalátoru pro Windows se kontroluje, zda je spuštěn Matlab.

```
1 # check if matlab is running, if so, prompt user to close it
2 while ps -e | grep "[M]ATLAB" > /dev/null; do
3     read -p "Matlab is running, please close it and then press any key."
4 done
```

Instalátor nemůže pokračovat, pokud je Matlab spuštěn. Doplnění Matlab Path pomocí bash skriptu probíhá obdobně jako v případě NSIS skriptu, je spuštěn Matlab a zavolán skript „FMUaddpath.m“ pomocí následujících příkazů.

```
1 echo "Please wait, setting path in Matlab."
2 $MATLABDIR/bin/matlab -nodisplay -nosplash -nojvm -nodesktop -r
3     "run('FMUaddpath.m');exit;" > /dev/null
4 echo "Matlab path file edited and saved."
```

Odinstalace v případě systému Windows probíhá standardně v Ovládacích panelech pomocí nástroje Programy a funkce (či Přidat nebo odebrat programy). Pro operační systém Linux je připraven skript, který je umístěn ve složce nástroje (matlabroot/toolbox/fmutoolbox/uninstall/).

4.3. Příprava FMU pro simulaci

Aby bylo možné FMU simulovat, je třeba FMU k simulaci připravit, tzn. zkopírovat do dočasné složky, rozbalit a načíst soubor XML.

Prvním krokem je vytvoření nové instance třídy `FMUclass` pomocí konstruktoru `this = FMUclass( pathOrFMU, fmuName )`. Tato metoda implementuje jak konstruktor z cesty k FMU, tak i kopírovací konstruktor („copyconstructor“), protože Matlab nepodporuje přetěžování funkcí (ve smyslu několikanásobné definice funkce s různými datovými typy parametrů). Prvním parametrem metody je buď cesta k souboru FMU, pokud je vytvářen použit konstruktor z cesty k FMU, nebo objekt třídy `FMUclass`, pokud je nová instance třídy vytvářena pomocí kopírovacího konstruktoru. Druhým (nepovinným) parametrem metody je požadovaný název proměnné v základním pracovním prostoru („base workspace“), tento parametr se používá zejména při simulaci FMU pomocí příkazové řádky či skriptu.

Všechny další potřebné operace zajišťuje funkce `loadFMU ( this )`. Úvodním krokem je zjištění typu operačního systému pomocí funkce `pcInfo ( this )`, která využívá interní funkci `computer` a její výstup (řetězce `'PCWIN64'`, `'PCWIN'`, `'GLNXA64'` a `'MACI64'`) převede na identifikaci operačního systému používanou FMU (řetězce `'win64'`, `'win32'`, `'linux64'` a `'mac'`). Následujícím krokem je kontrola, zda je operační systém podporován

pomocí funkce `checkCompatibility ( this )`. V případě nepodporovaného operačního systému je vyvolána výjimka a načítání FMU je ukončeno.

Dalším krokem je zkopírování FMU do dočasné složky, cesta k ní je zjištěna pomocí interní funkce `tempdir`. Následně je provedena změna přípony souboru FMU (z „.fmu“ na „.zip“). V případě, že již soubor či složka existuje, je proveden pokus o odstranění. V případě neúspěchu je vyvolána výjimka a načítání FMU je ukončeno. Tyto kroky provádí funkce `changeEXTfmu( this )`.

Poté je třeba soubor rozbalit, k tomu slouží funkce `unZIPfmu( this )`, která pomocí interní funkce `filenames = unzip( zipfilename, outputdir )` rozbalení provede a následně ze seznamu souborů získá cestu k XML souboru.

Ten je pomocí funkce `loadXMLfmu( this )` načten do struktur ve vlastnostech (properties) třídy `FMUclass`. Jedná se o proměnné modelu (`ModelVariables`), atributy modelu (`attributes`), definice jednotek (`UnitDefinitions`), definice datových typů (`TypeDefinitions`) a výchozí experiment (`DefaultExperiment`).

Následujícím krokem je výběr správné knihovny pomocí funkce `selectProperDll ( this )`. Tato metoda podle typu operačního systému (zjištěného pomocí funkce `pcInfo ( this )`) vybere odpovídající knihovnu, tzn. správný typ souboru (.dll pro Windows, .so pro Linux) a odpovídající architekturu (32-bitů nebo 64-bitů). V každém FMU mohou být knihovny pro libovolný počet cílových operačních systémů, výběr správné knihovny je důležitý pro korektní funkci nástroje (není možné kombinovat knihovny Windows a Linux a ani knihovny různých architektur).

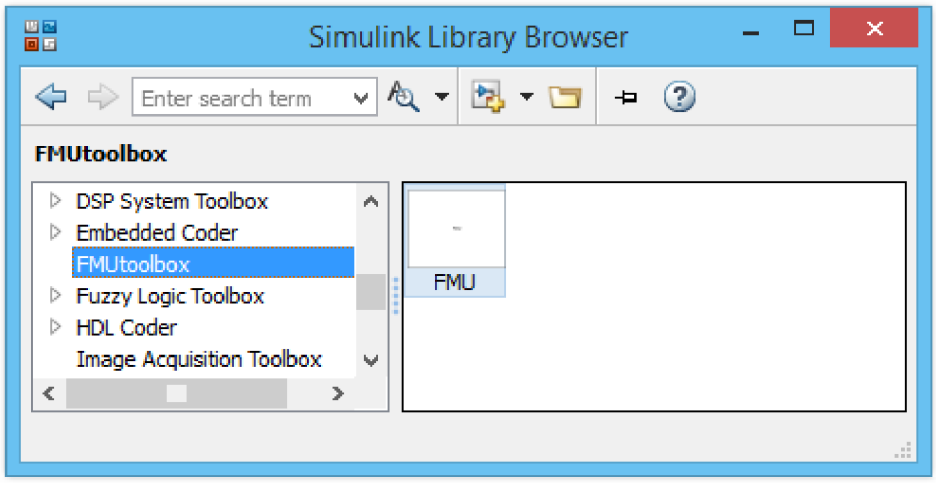
Poté jsou provedeny další přípravné kroky pro simulaci - nastavení výchozích hodnot proměnných modelu pomocí funkce `setVariablesStartValues ( this )`, příprava výchozích portů bloku FMU v Simulinku s využitím funkce `prepareSimulinkPorts ( this )` a nastavení výchozího textu masky bloku FMU v Simulinku pomocí funkce `set.simulinkMaskText(this, maskText)`, což je tzv. setter metoda pro vlastnost `simulinkMaskText` třídy `FMUclass`.

4.4. Knihovna FMUtoolbox pro Simulink

Pro simulaci FMU byla vytvořena knihovna Simulink (viz obrázek 4.3).

Její doplnění do prohlížeče knihoven Simulinku probíhá automaticky při instalaci nástroje FMUtoolbox. Pro korektní přidání knihovny bylo nutné vytvořit soubor `slblocks.m` a vlastní knihovnu bloků (model Simulink). Zde nastal problém s kompatibilitou formátů modelů (Matlab verze 2014b vyžaduje model ve formátu .slx, naopak starší verze Matlabu podporují pouze formát .mdl). Tento problém byl vyřešen při instalaci, kdy je v případě potřeby provedena konverze modelu ve formátu .mdl do formátu .slx.

Blok FMU z knihovny FMUtoolbox je nutné přetáhnout do modelu Simulink. Následně je možné dvojklikem na blok FMU otevřít grafické uživatelské rozhraní nástroje FMUtoolbox, jehož popis je uveden v následující kapitole.



Obrázek 4.3: Knihovna FMUtoolbox pro Simulink

4.5. Uživatelské rozhraní

Nástroj FMUtoolbox nabízí jak grafické uživatelské rozhraní, tak i možnost ovládání pomocí příkazové řádky či skriptů.

Grafické uživatelské rozhraní (dále jako GUI) je vytvořeno přímo v prostředí Matlab. Avšak z důvodu potřeby pokročilých funkcionalit nebylo možné použít nástroj GUIDE (graphical user interface design environment), celé GUI bylo vytvořeno programově (tzn. byl kompletně sepsán zdrojový kód GUI) dle [13].

Okno GUI je vytvořeno příkazem `figure`, který je volán následujícím způsobem:

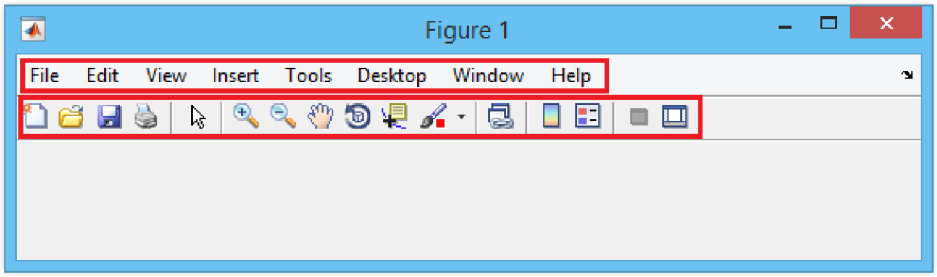
```
f = figure(          'Visible','off',...
                     'Name',['FMUtoolbox ' version],...
                     'Resize','off',...
                     'NumberTitle','off',...
                     'Color',[0.898,0.898,0.898],...
                     'Position',[10,50,1105,680]);
```

Horní lišty okna GUI (na obr. 4.4 označeny červeně) je možné deaktivovat pomocí příkazů

```
set(f,'Toolbar','none');
set(f,'MenuBar','none');
```

V okně GUI jsou umístěny panely, ve kterých jsou vloženy vlastní prvky GUI (textová pole, tlačítka, tabulky apod.). Pro vytvoření panelu se používá příkaz `uipanel`, jehož příklad použití následuje.

```
%% panel Control
pCtrl = uipanel(     'Title','Control',...
                     'Units','pixels',...
                     'FontSize',10,...
                     'BackgroundColor','w',...
                     'Position',[20 435 135 230]);
```



Obrázek 4.4: Horní lišty okna GUI

left	Vzdálenost od levého okraje nadřazeného objektu
bottom	Vzdálenost od spodního okraje nadřazeného objektu
width	Šířka objektu
height	Výška objektu

Tabulka 4.1: Vektor Position a jeho význam

Parametrem `'Title'` je určen nadpis panelu, `'Units'` určuje jednotky velikosti a pozice, `'FontSize'` velikost písma, `'BackGroundColor'` barvu pozadí panelu a `'Position'` určuje velikost a pozici panelu.

Parametr `'Position'` se používá pro všechny objekty v GUI a jeho význam je pro všechny typy objektů shodný. Jedná se o čtyřprvkový vektor `[left bottom width height]`, význam jednotlivých hodnot je uveden v tabulce 4.1.

Tlačítko v GUI je možné vytvořit pomocí příkazu `uicontrol`, jehož příklad použití následuje.

```
% button Load FMU
uicontrol(pCtrl,     'Style','pushbutton',...
                     'String','Load FMU',...
                     'Callback',@bLoadClick,...
                     'Position',[15 150 100 40]);
```

Prvním parametrem příkazu je grafický rodič daného prvku (v tomto případě panel `pCtrl`). Parametrem `Style` je určen typ ovládacího prvku a může nabývat hodnot `'checkbox'` (zaškrtávací pole), `'edit'` (textové pole), `'listbox'` (seznam možností), `'popupmenu'` (rozevírací seznam), `'pushbutton'` (tlačítko), `'radiobutton'` (přepínač), `'slider'` (posuvník), `'text'` (statické textové pole) a `'togglebutton'` (přepínací tlačítko). Pomocí parametru `String` je určen text ovládacího prvku. Parametr `Callback` definuje funkci zpětného volání („callback function“) ve formátu ukazatele na funkci („function handle“), pole buněk, kde v první buňce je ukazatel na funkci a v dalších buňkách jsou parametry funkce, nebo ve formátu řetězce znaků, který je platným zdrojovým kódem jazyka Matlab. Posledním použitým parametrem v tomto příkladě je `Position`, jehož význam byl již objasněn v předchozím odstavci.

Dalším objektem použitým v GUI je tabulka, jejíž vytvoření je možné pomocí příkazu `uitable`.

```
% table Parameters
tPar = uitable(      pPar,...
```

```
3                       'Data',[],...
4                       'RowName',[],...
5                       'ColumnName',{'Name', 'Value', 'Description'},...
6                       'ColumnEditable',[false true false],...
7                       'CellEditCallback',@tParEdit,...
8                       'ColumnWidth',{400,116,510},...
9                       'FontSize',10,...
10                      'BackgroundColor',[1 1 1],...
11                      'Position',tablePos);
```

Prvním parametrem je grafický rodič tabulky, parametrem `Data` je definován obsah tabulky (ve formátu pole buněk nebo nebo matice číselných hodnot). Pomocí parametrů `RowName` a `ColumnName` je možné určit názvy řádků a sloupců. Parametrem `ColumnEditable` je možné definovat, které sloupce může uživatel editovat, následující parametr `CellEditCallback` obsahuje ukazatel na funkci zpětného volání („callback function“), která je volána po editaci tabulky uživatelem. Šířku sloupců je možné určit pomocí parametru `ColumnWidth`. Parametr `FontSize` určuje velikost písma obsahu buněk, zbylé dva parametry (`BackgroundColor` a `Position`) jsou již popsány výše v textu.

Veškeré ovládací prvky GUI mají definovány funkce zpětného volání („callback functions“), které jsou volány při uživatelských akcích. Všechny funkce musí akceptovat minimálně dva parametry, které jsou automaticky funkci předávány při volání funkce. Prvním z parametrů je ukazatel („handle“) na grafický objekt, který vyvolal funkci. Obvyklý název parametru je `hObject` a používá se pro přístup k vlastnostem objektu v GUI (např. k aktuální hodnotě prvku, k datům v tabulkách apod.).

Druhým parametrem je proměnná (obvykle pojmenovaná `eventdata`), ve které jsou umístěny podrobnější informace související s voláním funkce. Například při editaci tabulky (nastavení parametrů FMU) je v tomto parametru obsažena proměnná datového typu `CellEditData`, jejímiž prvky jsou indicie editované buňky (číslo řádku a sloupce), původní a nová data editované buňky a další prvky. Data z tohoto parametru jsou používána například pro předávání původních a nových názvů portů bloku, ke zjišťování sloupce, ve kterém došlo k úpravě a podobně.

Příklad zpětně volané funkce („callback function“) je na následujících řádcích. Tato funkce provádí změnu portu pro určitou proměnnou FMU (tzn. přeřadí výstup proměnné z jednoho portu do druhého nebo proměnnou přidá do portu). Nejprve je z automaticky předaných parametrů funkce zjištěn název proměnné (`varname`), pro kterou je prováděna změna. Dalším krokem je získání názvu portu (`newPortName`), do kterého má být proměnná přiřazena. Získaná dvojice hodnot (`varname` a `newPortName`) je následně uložena do struktury pro další použití.

```
1       function tOutEdit(hObject,eventdata)
2           fmu = getFMUobject(gcb);
3           data = get(hObject,'Data');
4           index = eventdata.Indices(1);
5           varname = data{index,1};
6
7           uData = get(hObject,'UserData');
8
9           if isempty(uData)
```

```
10              uData = fmu.emptyIOuDataStruct();
11              k = 1;
12          else
13              k = length(uData)+1;
14          end % if
15
16          newPortName = eventdata.NewData;
17
18          uData(k).varName = varname;
19          uData(k).portName = newPortName;
20
21          set(hObject,'UserData',uData);
22      end %function tOutEdit
```

Tyto funkce jsou umístěny přímo v souboru s definicí GUI, další funkce potřebné pro funkcionalitu GUI jsou umístěné v separátních souborech. Důvodem pro toto rozdělení je přehlednost a možnost volání těchto funkcí i mimo soubor s definicí GUI. Funkce zajišťují například úpravy portů bloku, ukládání a načítání dat, kopírování bloků FMU a podobně. Podrobný popis těchto funkcí by rozsahem přesáhl tuto práci, proto není uveden.

Otevírání GUI je řešeno pomocí funkce `openActivateGui( blockHandle )`, jejímž jediným parametrem je ukazatel na aktuální blok (který je otevírán). Tato funkce zkontroluje, zda pro daný blok je již GUI otevřeno. V tom případě dojde pouze k aktivaci daného okna s dříve načtenými daty a provedenými změnami. V opačném případě dojde k vytvoření nového okna grafického rozhraní zavoláním funkce `FMUgui()`. Otevírání a aktivace oken GUI funguje shodně i pro více instancí FMU a to jak různých, tak i shodných modelů (např. s různými parametry). Podrobný popis GUI je uveden v uživatelském návodu, který je přiložen k této práci.

Druhou částí uživatelského rozhraní je ovládání pomocí příkazové řádky Matlabu (nebo pomocí skriptu Matlabu). Pro každé FMU, které má být simulováno, je vytvořena instance třídy `FMUclass`. Pomocí veřejných metod a vlastností třídy může uživatel provádět různá nastavení, spustit simulaci nebo získat informace o FMU (proměnné modelu apod.). Díky tomuto přístupu jsou využívány shodné metody třídy, jako při ovládání pomocí GUI, není tedy třeba spravovat dvě oddělené části kódu. Pouze bylo nutné vytvořit několik nových metod pro simulaci pomocí příkazové řádky.

První z těchto metod je `setSimulationParameters ( .. )`, která se používá pro nastavení parametrů simulace.

```
1 setSimulationParameters ( startT, stopT, solverType,...
2                           Solver, fixedStepSize,...
3                           minStepSize, maxStepSize,...
4                           relTol, absTol)
```

Pomocí této metody může uživatel nastavit délku simulace, solver, simulační krok, tolerance a podobně. Tato nastavení jsou uložena v odpovídajících vlastnostech třídy a následně jsou před simulací použita pro její nastavení.

Druhou z těchto metod je `getInputsTemplate( )`, která připraví novou strukturu formálně shodnou se strukturou `ScopeData` (struktura výstupních dat z bloku Scope). Tato struktura je předána uživateli k naplnění vstupními daty a následně jsou tato data pou-

žita jako vstupy FMU při simulaci. Důvodem pro toto řešení byla standardizace vstupních dat, navíc struktura `ScopeData` je známá většině uživatelů Matlabu.

Vstupní data jsou do FMU předávány pomocí metody `setInputValues( inputValues )`, jejímž jediným parametrem je struktura se vstupními daty. Metoda zkontroluje data po formální stránce (počty vzorků, datové typy apod.) a následně jsou data uložena ve vlastnosti třídy pro použití při simulaci.

Další metodou je `setModelVariableAsOutput( varName, setUnset )`, kterou může uživatel použít k přidání proměnné modelu mezi výstupy. Tím budou hodnoty této proměnné ukládány a předány po skončení simulace uživateli. Metoda v prvním kroku zkontroluje předané parametry (zda proměnná s názvem `varName` existuje v daném FMU a zda je parametr `setUnset` typu `logical`). Podle parametru `setUnset` metoda buď vytvoří nový port bloku a danou proměnnou přiřadí do tohoto portu (v případě hodnoty `true`), nebo proměnnou vyřadí z daného portu a port je smazán (v případě hodnoty `false`).

Pomocí metody `simulate( )` může uživatel spustit simulaci. Před vlastním spuštěním simulace je zavolána metoda `prepareSimulinkModel( )`, která programově vytvoří schéma v Simulinku obsahující potřebné bloky. Tyto bloky také mezi sebou propojí a nastaví veškeré parametry simulace (koncový čas, krok, tolerance apod.). Následně je spuštěna simulace schématu Simulink a po jejím ukončení jsou výstupní data uložena do vlastnosti `savedSimData` třídy `FMUclass`. Tato vlastnost třídy je veřejná, takže uživatel může výstupní data libovolně dále použít.

Pro simulaci pomocí příkazové řádky je možné použít veřejné vlastnosti třídy `FMUclass`. Pomocí těchto vlastností je možné zjistit podrobné informace o FMU, zejména seznam proměnných, atributy FMU, nastavení simulace a podobně.

Podrobný popis rozhraní pro simulaci FMU pomocí příkazové řádky Matlabu je uveden v uživatelském manuálu, který je přílohou této práce.

4.6. Simulace FMU

Pro vlastní simulaci FMU byla vytvořena Level-2 MATLAB S-Funkce (dále jako S-Funkce) dle [14].

V části `setup` je provedeno nastavení počtu vstupů a výstupů na základě kauzality proměnných v případě vstupů a na základě přiřazení proměnných do portů v případě výstupů. Dále jsou provedeny úpravy nastavení vstupů a výstupů (dimenze apod.). Podle počtu spojitých stavů FMU je provedeno nastavení počtu spojitých stavů bloku S-Funkce.

```
1 block.NumContStates = fmu.attributes.numberOfContinuousStates;
```

Ve funkci `Start` je provedena příprava FMU k simulaci pomocí metody `prepareFMUsim` třídy `FMUclass`. V rámci této funkce je provedeno načtení sdílených knihoven, volání funkce `fmiInstantiateModel`, nastavení aktuálního času FMU, volání funkce `fmiInitialize` a alokace ukazatelů na stavové a další proměnné používané pro komunikaci s knihovnou FMU.

Následuje provedení tzv. nultého kroku pomocí metody `zeroStep` třídy `FMUclass`. Tato metoda získá z FMU aktuální hodnoty stavového vektoru, nominální hodnoty sta-

vového vektoru a hodnoty dalších proměnných (zejména výstupů apod.) a uloží je do atributů instance třídy `FMUclass`.

Dalším krokem funkce `Start` je inicializace stavového vektoru S-Funkce (hodnoty stavů jsou získány z vlastností instance třídy `FMUclass`).

Funkce `Outputs` nejdříve nastaví hodnoty vstupů do atributu `modelvariables` třídy `FMUclass` s využitím metody `setModelVariable( nameCell, val )`. Dalším krokem je zjištění aktuálního času simulace pomocí příkazu

```
1 actualTime = get_param(bdroot,'SimulationTime');
```

a následně nastavení času v FMU pomocí metody `setTime( time )` třídy `FMUclass`.

Poté dojde k zápisu hodnot vstupů do FMU pomocí metody `setVariables( this )`, která pro každý datový typ proměnných zavolá funkci `setVarsOneType( this, typeName )`. Tato funkce následně data z atributu `modelvariables` třídy `FMUclass` převede do vhodného formátu (ukazatel na pole `valueReference`, čímž jsou definovány proměnné k nastavení, a ukazatel na pole hodnot odpovídajících těmto proměnným). Podle datového typu je volána odpovídající funkce z dynamické knihovny, která předá výše definovaná data přímo do instance FMU.

Dalším krokem je nastavení aktuálního stavového vektoru S-Funkce do FMU pomocí metody `setContinuousStates( this, x )` třídy `FMUclass`, která zkontroluje platnost předaných dat a s využitím dynamické knihovny nastaví hodnoty stavů do FMU.

Pomocí funkce `DllgetEventIndicators` jsou získány indikátory událostí, jejichž aktuální hodnoty jsou porovnány s předešlými hodnotami a pokud u některého z indikátoru došlo ke změně znaménka, je detekována událost.

Detekce událostí je prováděna také při volání funkce `DllcompletedIntegratorStep`, kdy je v návratovém parametru `callEventUpdate` indikováno, zda došlo k události.

Třetí možností detekce událostí je porovnáním aktuálního času simulace s očekávaným časem další události.

Pokud je jednou z výše popsaných metod detekována událost, musí být zavolána funkce `fmiEventUpdate( .. )`. Pomocí návratových parametrů této funkce je možné zjistit typ události a provést odpovídající akci (ukončit simulaci, získat aktuální stavový vektor, zjistit očekávaný čas další události apod.). Volání funkce je nutné opakovat, dokud funkce nevrátí hodnotu `fmiTrue`.

Dalším krokem je získání aktuálních hodnot proměnných, které jsou předány na výstup bloku S-Funkce.

V části `Derivatives` je volána pouze funkce `fmiGetDerivatives( m, der_x, nx )`, pomocí které jsou z FMU načteny aktuální hodnoty derivací stavů, které jsou přiřazeny do vlastnosti `Derivatives` S-Funkce a následně je využívá solver Matlabu.

```
1 block.Derivatives.data = fmu.x_der.Value;
```

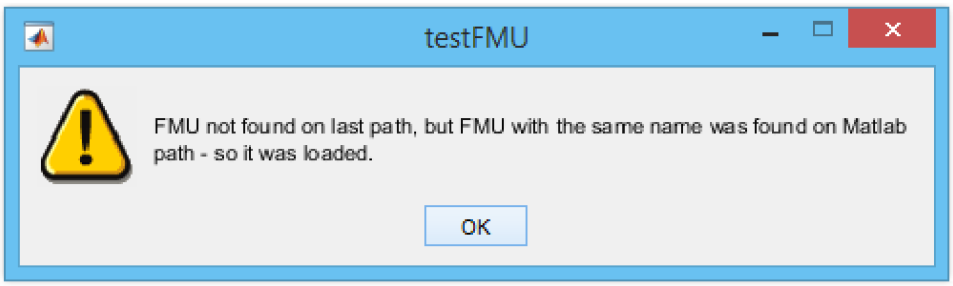
Část `Terminate` obsahuje pouze volání metody `terminateSim( this )` třídy `FMUclass`. Tato metoda zajistí korektní ukončení simulace pomocí funkce `fmiTerminate` a následně uvolnění alokované paměti a dalších zdrojů pomocí `freeModelInstance`.

4.7. Další funkce FMUtoolboxu

Model s blokem FMUtoolbox je možné standardně uložit a následně načíst. V ideálním případě (pokud je FMU stále ve stejném umístění) se po otevření uloženého modelu načte FMU do prostředí Matlab a je možné model simulovat.

Ukládání probíhá pomocí funkce `saveFMUblock( )`, která je volána automaticky při ukládání (je to callback funkce Simulink bloku). Funkce všechna uživatelská data uloží do struktury definované v třídě `FMUclass`.

Načítání při otevírání modelu se provádí pomocí funkce `loadFMUblock( )`, která je automaticky volána při načítání bloku. Funkce hledá soubor FMU na původní absolutní cestě, pokud jej na tomto umístění nalezne, dojde k načtení FMU. Pokud není soubor FMU nalezen na původní absolutní cestě, funkce se pokusí nalézt toto FMU (shodný název souboru) ve složkách Matlab Path. Pokud nalezne jeden soubor FMU, načte jej a informuje o tom uživatele pomocí dialogu varování (příklad dialogu je na obrázku 4.5 a zdrojový kód následuje dále v textu).



Obrázek 4.5: Dialogové okno s varováním při nalezení FMU v jiném umístění

```
1 warndlg(['FMU not found on last path, ',...
2             'but FMU with the same name was found on Matlab path '...
3             '- so it was loaded.'],...
4             uData.matlabName);
```

Pokud nalezne více souborů FMU se shodným názvem, je načten první z nich a uživatel je opět upozorněn pomocí dialogového okna. Pokud dojde k načtení FMU, jsou následně aplikována uživatelská nastavení, která byla přiřazena do struktury při ukládání modelu. V případě, že FMU není nalezeno, je o tom uživatel informován pomocí dialogového okna a pro simulaci je nutné soubor FMU vybrat ručně.

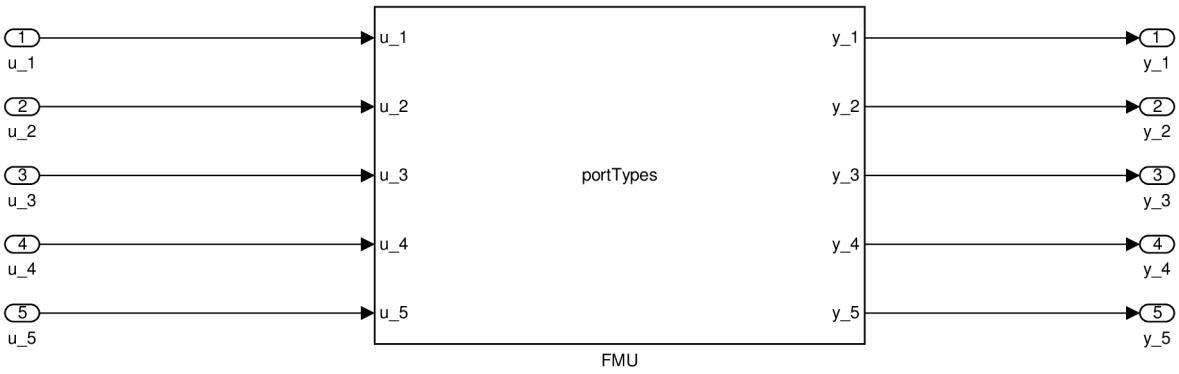
Standardní cestou (pomocí `Ctrl+C` a `Ctrl+V`, pravého tlačítka myši a potáhnutím atd.) je možné Simulink blok FMU zkopírovat. Při kopírování bloku je načtena nová instance daného FMU pomocí kopírovacího konstruktoru, který zajišťuje identické uživatelské nastavení FMU (zejména parametry FMU, porty a další nastavení).

Blok FMU podporuje různé typy signálů jak pro vstupní, tak i pro výstupní porty bloku. Výchozím typem portů je individuální typ (skalární hodnota). Pomocí uživatelského rozhraní je možné u každého portu změnit jeho typ na vektor (`mux`) nebo sběrnici (`bus`). V případě použití vektoru nebo sběrnice je možné do portu přiřadit libovolné množství jednotlivých signálů (hodnot proměnných FMU). Každý port bloku může být jiného

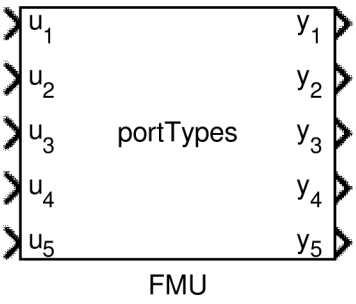
typu, tím je možné vytvořit v podstatě libovolnou kombinaci vstupně-výstupních portů. Navíc je možné měnit pořadí portů (vybraný port přemístit výše či níže).

Protože blok Simulinku pro S-Funkci nepodporuje sběrnice, bylo nutné toto vyřešit jiným způsobem. Bylo rozhodnuto, že vstupy a výstupy bloku S-Funkce budou skalární hodnoty a zapouzdředním bloku S-Funkce do subsystému bude vyřešeno korektní rozhraní s ostatními bloky ve schématu Simulink.

V případě skalárního vstupu nebo výstupu je v subsystému provedeno pouze propojení portu bloku S-Funkce s odpovídajícím portem subsystému. Příklad propojení bloku S--Funkce a portů subsystému pro skalární vstupy a výstupy je na obr. 4.6, vnější vzhled bloku je zobrazen na obr. 4.7.



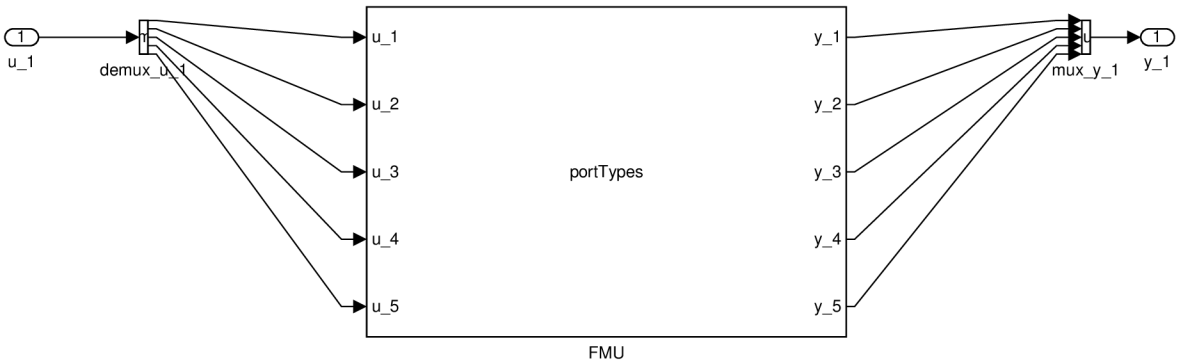
Obrázek 4.6: Subsystém se skalárními vstupy a výstupy



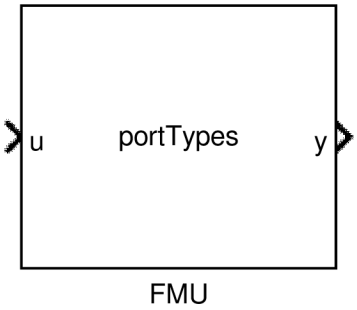
Obrázek 4.7: Vzhled bloku se skalárními vstupy a výstupy

Pro výstup typu vektor je do subsystému doplněn blok `Mux`, na jehož vstupy jsou přivedeny výstupní signály z bloku S-Funkce a výstup z bloku `Mux` je přiveden na odpovídající výstupní port subsystému. Pro vstupní port typu vektor je do subsystému doplněn blok `Demux`, na jehož vstup je připojen port subsystému a jeho výstupy jsou propojeny s blokem S-Funkce. Příklad propojení bloku S-Funkce a portů subsystému pro vektorový vstup a výstup je na obr. 4.8 a vnější vzhled bloku je zobrazen na obr. 4.9.

V případě výstupu typu sběrnice je do subsystému doplněn blok `BusCreator`, který vytvoří sběrnici ze skalárních signálů z bloku S-Funkce. Výstup bloku `BusCreator` je přiveden na odpovídající výstupní port subsystému. Pro vstupní port typu sběrnice jsou do subsystému přidány bloky `Bus To Vector` a `Demux`. Sběrnicový signál ze vstupu subsystému je přiveden na vstup bloku `Bus To Vector` a výstup tohoto bloku je připojen na vstup bloku `Demux`. Výstupy tohoto bloku jsou propojeny se vstupy S-Funkce. Příklad

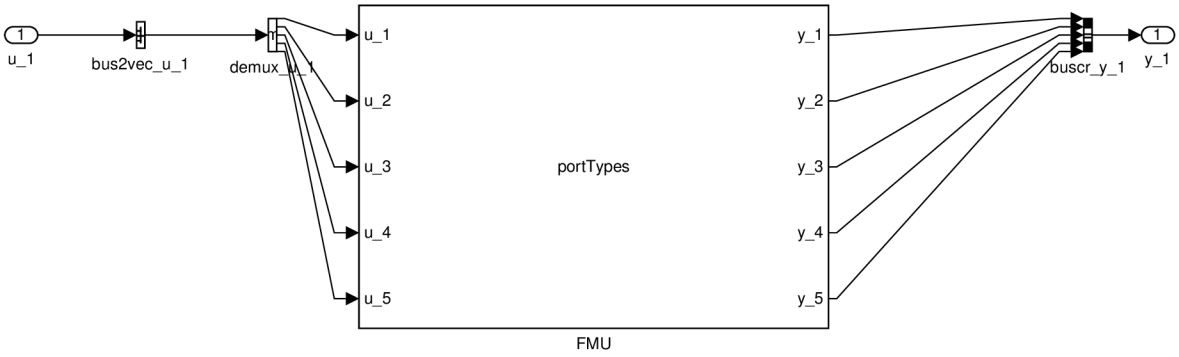


Obrázek 4.8: Subsystém s vektorovým vstupem a výstupem



Obrázek 4.9: Vzhled bloku s vektorovým vstupem a výstupem

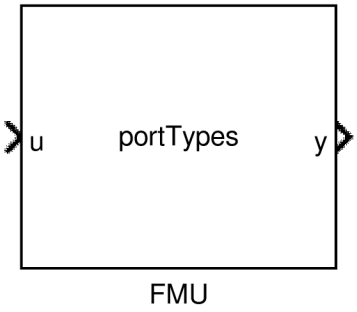
propojení bloku S-Funkce a portů subsystému pro vektorové vstupy a výstupy je na obr. 4.8, vnější vzhled bloku je na obr. 4.11.



Obrázek 4.10: Subsystém se sběrnicovým vstupem a výstupem

Na obrázku 4.12 je ilustrována možnost použití libovolné kombinace typů vstupů a výstupů pro jedno FMU. Pro ověření této funkcionality byl v prostředí OpenModelica vytvořen jednoduchý model, který byl následně exportován do formátu FMU. Model pouze přiřadí hodnotu vstupu na výstup v odpovídajícím pořadí. Poté byl model s využitím nástroje FMUtoolbox importován do prostředí Matlab/Simulink a bylo provedeno nastavení vstupních a výstupních portů.

První vstup je ponechán jako skalární hodnota, stejně tak je skalární i první výstup.

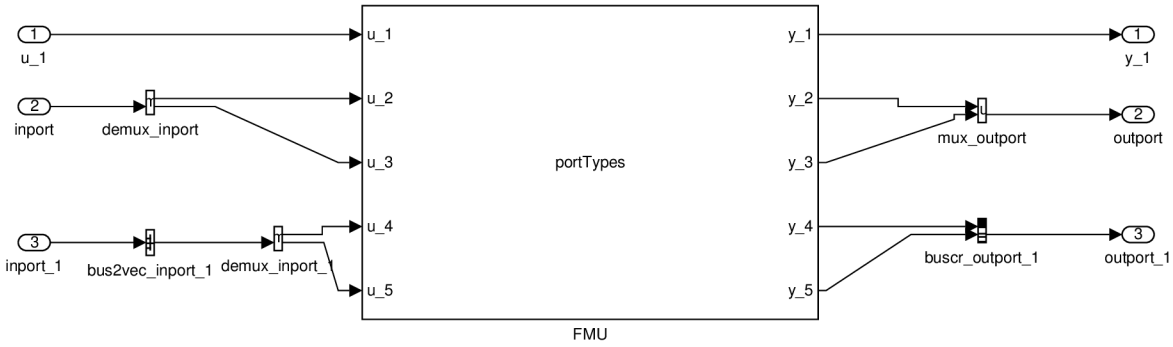


Obrázek 4.11: Vzhled bloku se sběrnicovým vstupem a výstupem

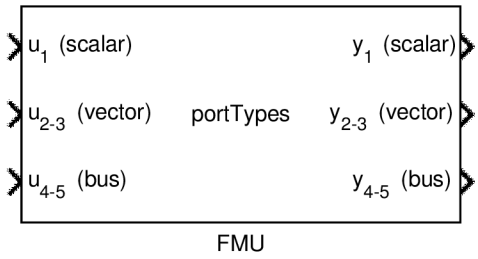
Druhý vstupní port bloku je ve formátu vektor a obsahuje druhý a třetí vstup S-Funkce, na tento port je tedy možné připojit dvouprvkový vektorový signál. Druhý a třetí výstup S-Funkce jsou sdruženy do vektorového výstupu.

Třetí vstupní port bloku je typu sběrnice, jsou do něj přiřazeny čtvrtý a pátý vstup S-Funkce. Do tohoto portu je nutné připojit sběrnicový signál se dvěma prvky. Čtvrtý a pátý výstupní port jsou seskupeny do výstupního portu typu sběrnice.

Na obrázku 4.13 je ukázán vnější vzhled bloku, jehož porty byly nastaveny dle předcházejícího textu. Tímto bylo prakticky předvedeno, že u bloku nástroje FMUtoolbox je možné nastavit libovolnou kombinaci vstupních a výstupních portů.



Obrázek 4.12: Subsystém s kombinací skalárních, vektorových a sběrnicových vstupů a výstupů



Obrázek 4.13: Vzhled bloku s kombinací skalárních, vektorových a sběrnicových vstupů a výstupů

Všechny výše popsané úpravy v subsystému (propojování bloku S-Funkce a portů subsystému, doplňování pomocných bloků apod.) jsou prováděny automaticky na základě

nastavení provedeného uživatelem v GUI. Pro zajištění této funkcionality bylo vytvořeno několik callback funkcí (`addSubsystemMuxBus`, `clearSubsystemIO` a `setSubsystemIO`), které jsou volány při potvrzení dané změny uživatelem.

Přidávání bloků do schématu Simulink je možné pomocí příkazu `add_block ( .. )`, příklad jeho použití je na následujících řádcích.

```
1 add_block('built-in/BusCreator',...
2     [blockHandle '/buscr_' portName],...
3     'Position',...
4     [portPos(1)-20 portPos(2)-5 portPos(1)-15 portPos(2)+10]);
```

Prvním parametrem funkce je název bloku, který má být přidán do schématu. Druhým parametrem cílové umístění bloku (model, do kterého má být blok vložen a jeho požadovaný název). Dále mohou následovat nepovinné parametry funkce, v tomto případě je nastavena pozice vkládaného bloku ve schématu. Pozice je definována pomocí vektoru `Position` (viz tabulka 4.1).

Propojení bloků mezi sebou je řešeno pomocí funkce `add_line ( .. )`, příklad jejího použití je na následujích řádcích.

```
1 add_line(blockHandle,...
2     ['buscr_' portName '/1'],...
3     [portName '/1']);
```

Prvním parametrem funkce je ukazatel na schéma Simulinku, ve kterém mají být bloky propojovány. Druhým parametrem funkce je výstupní port bloku, ze kterého má být propoj veden. Třetím parametrem je vstupní port bloku, kam má být propoj veden. Pořadí parametrů není možné zaměnit (není možné provést propojení z výstupu na vstup) a konkrétní port bloku je nutné definovat včetně číselného označení portu a to i v případě, že daný blok má pouze jeden port.

4.8. Ukládání provozních stavů (logging)

Pro účely zpětného řešení možných problémů při simulaci FMU je v dynamické knihovně vytvořena funkce `fmuLogger( .. )`. Ukazatel na tuto funkci je předáván jako parametr funkce `instantiateModel( .. )` do dané instance FMU.

Funkce `fmuLogger` po jejím zavolání zapíše do souboru s názvem `FMUlog.txt` aktuální čas, stav, název instance FMU, kategorii a zprávu. Všechny tyto informace jsou předány funkci `fmuLogger` při jejím volání z FMU. K tomu může dojít při výskytu chyby při simulaci (popsáno v kapitole 2.1.2) nebo při volání kterékoliv funkce z FMU, pokud je povolené logování („debug logging“). Zapnutí logování je možné prostřednictvím uživatelského rozhraní nebo pomocí vlastnosti `logging` při využívání simulace pomocí příkazové řádky.

Do stejného souboru jsou zapisovány i chybové hlášky při načítání nebo uvolňování dynamických knihoven FMU. Tyto problémy mohou nastat, pokud se nezdařilo načtení knihovny (např. knihovna je závislá na jiné knihovně, která nebyla nalezena). Při uvo-

lňování knihoven mohou nastávat problémy, pokud FMU korektně neuvolní alokovanou paměť, načtené knihovny a další použité zdroje.

Na následujících řádcích je uveden příklad souboru se zprávami o simulaci FMU. Jedná se pouze o velmi zkrácený ilustrační příklad, celý záznam může mít např. 6000 řádků (simulační čas 10 s).

```
1  Sun Apr 12 18:23:11 2015
2   0 ControlledTemperature  fmiInstantiateModel completed
3  Sun Apr 12 18:23:11 2015
4   0 ControlledTemperature  fmiSetTime to 0
5  Sun Apr 12 18:23:11 2015
6   0 ControlledTemperature  fmiInitialize...
7  Sun Apr 12 18:23:11 2015
8   0 ControlledTemperature  fmiInitialize completed
9  Sun Apr 12 18:23:11 2015
10  0 ControlledTemperature  fmiInitialize
11 Sun Apr 12 18:23:11 2015
12  0 ControlledTemperature  fmiGetContinuousStates
13 Sun Apr 12 18:23:11 2015
14  0 ControlledTemperature  fmiGetNominalContinuousStates
15 Sun Apr 12 18:23:11 2015
16  0 ControlledTemperature  fmiGetReal: #r335544320# = 293.15
17 Sun Apr 12 18:23:11 2015
18  0 ControlledTemperature  fmiGetReal: #r335544321# = 20
19 Sun Apr 12 18:23:11 2015
20  0 ControlledTemperature  fmiGetBoolean: #b335544322# = 0
21 Sun Apr 12 18:23:12 2015
22  0 ControlledTemperature  fmiSetTime to 0
23 Sun Apr 12 18:23:12 2015
24  0 ControlledTemperature  fmiSetContinuousStates
25 Sun Apr 12 18:23:12 2015
26  0 ControlledTemperature  fmiCompletedIntegratorStep
27 Sun Apr 12 18:23:12 2015
28  0 ControlledTemperature  fmiGetEventIndicators
29 Sun Apr 12 18:23:12 2015
30  0 ControlledTemperature  fmiEventUpdate...
31 Sun Apr 12 18:23:12 2015
32  0 ControlledTemperature  fmiEventUpdate completed
33         .
34         .
35         .
36 Sun Apr 12 18:23:37 2015
37  0 ControlledTemperature  fmiTerminateModel
38 Sun Apr 12 18:23:37 2015
39  0 ControlledTemperature  fmiFreeModelInstance
```

4.9. Podpůrné nástroje pro FMUtoolbox

Pro zjednodušení vývoje nástroje FMUtoolbox byl využit verzovací systém Git. Bylo použito `Gitflow Workflow` [15], které je odvozeno z modelu větví („branch model“) od

Vincenta Driessena [16]. Při tomto přístupu probíhá vývoj ve větvi `develop`, ze které mohou vycházet větve nových vlastností („feature branches“), název je ve formátu např. `feature/busSupport`. Při vývoji nástroje FMUtoolbox bylo využito 10 větví pro nové vlastnosti.

Pro vydávání nových verzí je určená větev `master`, do které je vždy před vydáním nové verze provedeno sloučení („merge“) z větve `develop` za pomoci větve typu `release` (např. `release/1.2.1.1`).

Použitím verzovacího systému Git se výrazně zjednušil vývoj nástroje na více platformách (Microsoft Windows 32-bit, 64-bit a Linux). Také se tím minimalizovalo riziko ztráty částí zdrojových kódů (existuje záloha celého průběhu vývoje nástroje), v případě chyby existuje možnost návratu zpět a je možné v historii vyhledávat změny souborů.

Souběžně s verzovacím systémem byl používán i systém pro správu chyb („issue tracker“), který je s verzovacím systémem propojen (při odevzdávání („commit“) je možné uzavřít lístek, tím se úpravy s lístkem propojí). V průběhu vývoje nástroje bylo vytvořeno cca 40 lístků (obsahujících chyby či požadavky na novou funkcionalitu nebo vylepšení stávající), až na výjimky jsou všechny vyřešeny.

Na začátek každého souboru se zdrojovým kódem byla umístěna hlavička, ve které jsou umístěny kontakty, číslo verze a datum vydání verze.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   FMUtoolbox for Simulink                                                    %
%   https://bitbucket.org/vaclavek/fmutoolbox                                  %
%                                                                              %
%   Jan Glos (CEITEC BUT)                                                      %
%   jan.glos@ceitec.vutbr.cz                                                   %
%                                                                              %
%   Version: 0.6.1.1                                                           %
%   Date: 11/2014                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Pro změnu verze a data byl vytvořen bash skript, ruční úprava by byla příliš náročná (jedná se o cca 30 souborů). Skript uživatele vyzve k odevzdání („git commit“) všech souborů a zkontroluje, zda v úložišti není některý soubor upravený a zároveň neodevzdaný.

Následně je uživatel vyzván k určení nového čísla verze a skript navrhne aktuální datum k odsouhlasení (nebo je možné zadat vlastní).

Skript poté projde všechny soubory v úložišti a upraví v nich hlavičku souboru (pokud je v souboru nalezena).

Spuštění skriptu je možné na operačním systému Linux, nebo i na Microsoft Windows s využitím prostředí MinGW.

Verzovací schéma bylo použito ve formátu 1.5.3.2, kde na první pozici („1“) je hlavní verze („major version“), na druhé pozici („5“) je vedlejší verze („minor version“), na třetí pozici („3“) je typ verze (0 - alfa, 1 - beta, 2 - release candidate, 3 - release) a čtvrtá pozice označuje pořadí (např. verze po opravě chyby bude mít vyšší číslo na poslední pozici).

Dalším podpůrným nástrojem je využívání poznámek `%ToDo` a `%FixMe`. V prostředí Matlab je možné vygenerovat report, který vypíše seznam těchto poznámek, je tedy možné zkontrolovat, zda byly opraveny všechny chyby a zda nechybí dodělat některé části kódu.

Pro optimalizaci kódu v prostředí Matlab byl použit nástroj Profiler, který pro vykonanou funkci zobrazí čas potřebný pro vykonání operací prováděných v této funkci (volání jiných funkcí, jednotlivé příkazy apod.). Při hledání neoptimálních částí kódu je možné se

postupně dostat až k jednotlivým řádkům kódu, které vyžadují dlouhý čas pro zpracování. Pomocí tohoto nástroje se podařilo zkrátit dobu simulace vzorového FMU na cca jednu desetinu původního času. Například úpravou kódu S-Funkce pro přiřazení hodnot výstupů na porty bloku S-Funkce se podařilo snížit časovou náročnost na jednu pětinu původní náročnosti (díky časově optimálnímu filtrování proměnných a snížení počtu průchodů cyklu).

4.10. Podpora FMU exportovaných z nástroje OpenModelica

Prostředí OpenModelica podporuje export FMU, avšak pouze pro určité platformy (32-bit FMU pro Windows, 64-bit FMU pro Linux). Při pokusu o simulaci FMU exportovaného z OpenModelicy pomocí nástroje FMUtoolbox docházelo k chybám (a to jak v operačním systému Microsoft Windows, tak i na operačním systému typu Linux (Ubuntu)).

Na operačním systému MS Windows nastával problém při načítání dynamické knihovny z FMU. Kvůli chybě v kódu nástroje OpenModelica nebylo možné k ladění použít `Logger` z FMU, proto byl využit nástroj `Dependency Walker`. Pomocí tohoto nástroje bylo zjištěno, že dynamická knihovna exportovaná z OpenModelica závisí na dalších dynamických knihovnách. Podrobnější analýza problému byla provedena pomocí nástroje `Process Monitor v3.1` ze skupiny nástrojů Windows Sysinternals. Při načítání dynamické knihovny byly zaznamenány pokusy o načtení dalších dynamických knihoven, většina z nich je přidána do FMU při jeho exportu. Tyto knihovny se ale nedařilo načíst, protože cesta k rozbalenému FMU nebyla obsažena v systémové proměnné `PATH`. Proto je vždy před načítáním dynamické knihovny z FMU provedeno doplnění systémové proměnné `PATH` o cestu k rozbalenému FMU.

```
% temporarily add FMU temp folder into PATH (there might be
% additional dlls on which the simulation is dependent, like in
% FMU exported from OpenModelica)
[pathstr,~,~] = fileparts(this.fmudll);
setenv('PATH', [getenv('PATH') ';' pathstr]);
```

Po načtení dynamické knihovny z FMU je opět přidaná cesta z proměnné `PATH` odebrána.

```
% remove FMU temp folder from PATH
newPATH = strrep(getenv('PATH'),[';' pathstr],'');
setenv('PATH', newPATH);
```

I po této úpravě dochází k neúspěšným pokusům o načítání některých dalších dynamických knihoven (`quserex.dll` apod.), avšak tyto knihovny nejsou nutné pro simulaci FMU.

Při ladění simulace FMU exportovaných z nástroje OpenModelica byly zjištěny problémy s použitím funkce `Logger`, které byly způsobeny chybami v kódu nástroje OpenModelica. Na následujících řádcích je výpis souboru vytvořeného „logu“, který není čitelný.

```
Sun Jun 01 12:27:34 2014
 0 OM_night log fmiSetTime: time=%.16g
Sun Jun 01 12:27:34 2014
```

```
4   0  log fmiSetReal: nvr = %d
5  Sun Jun 01 12:27:34 2014
6   0  log fmiSetReal: #r%d# = %.16g
7  Sun Jun 01 12:27:34 2014
8   0 Ra) < log fmiInitialize: toleranceControlled=%d relativeTolerance=%g
9  Sun Jun 01 12:27:34 2014
10  0 Ra) < log fmiInitialization: succeed
11 Sun Jun 01 12:27:34 2014
12  3 Ec) < error %s: Invalid argument %s = NULL.
13 Sun Jun 01 12:27:34 2014
14  0 Ec) < log fmiFreeModelInstance
```

Tyto problémy byly nahlášeny vývojovému týmu nástroje OpenModelica pomocí systému trac.openmodelica.org/OpenModelica, lístek #2706. Jednalo se o chybnou práci s pamětí, kdy si FMU nevytvořilo kopii názvu FMU ve vlastním paměťovém prostoru, což je v rozporu se standardem FMI [8], kapitola `2.5 Creation and Destruction of Model Instances`: *The string-valued arguments instanceName and GUID passed to this function, must be copied inside this function, because there is no guarantee for a string lifetime after this function returned.* Pro prokázání této chyby byl využit nástroj `FMU Compliance Checker`, aby byla vyloučena možnost chyby v nástroji FMUtoolbox. `FMU Compliance Checker` je oficiální nástroj pro kontrolu shody FMU se standardem FMI. Při kontrole FMU byl nalezen další problém s uvolněním alokované paměti, část výpisu z nástroje `Compliance Checker` je na následujících řádcích.

```
1 [ERROR][FMUCHK] FMU does not make an internal copy of provided
2                 instance name (violation of fmiString handling)
3 [ERROR][FMUCHK] Memory leak: freeMemory was not called for 1
4                 block(s) allocated by allocateMemory
```

Oba tyto reportované problémy byly opraveny v „changesetu“ 20915, který je již obsažen v oficiálním vydání OpenModelica 1.9.2 Beta.

Na operačním systému Linux se objevil problém při volání funkce `fmiFreeModelInstance(fmiComponent c)`, která korektně neuvolnila veškeré použité zdroje (načtené dynamické knihovny), což bylo v rozporu se standardem FMI [8], kapitola `2.5 Creation and Destruction of Model Instances`: *Dispose the given model instance and deallocate all the allocated memory and other resources that have been allocated by the functions of the Model Exchange Interface for instance “c“.* Po volání funkce `fmiFreeModelInstance(fmiComponent c)` nebylo možné uvolnit dynamickou knihovnu FMU pomocí funkce `dlclose`. Tento problém byl také nahlášen pomocí systému trac.openmodelica.org/OpenModelica, lístek #2772.

Tento problém byl vyřešen v „changesetu“ 21774, upravené zdrojové kódy jsou součástí aktuální verze nástroje OpenModelica.

Při testování bylo také zjištěno, že pro simulaci FMU exportovaného nástrojem OpenModelica je nutné mít tento nástroj nainstalovaný (FMU využívá dynamické knihovny, které v něm nejsou obsaženy). Toto platí pouze pro použití v rámci operačního systému Linux (na MS Windows jsou dynamické knihovny v FMU obsaženy).

4.11. Návratové hodnoty funkcí FMU

Většina funkcí FMU má jako návratovou hodnotu příznak („flag“) `status` typu `fmiStatus`, jak již bylo popsáno v kapitole 2.1.2. Standardem FMI je definováno, jaká akce přísluší jednotlivým hodnotám.

Pro korektní reakci na hodnotu příznaku byla vytvořena metoda `function fmiStatusHandle (this, status, fcnName)`. Tato funkce podle hodnoty parametru `status` provede požadovanou akci (varování uživatele, chybové hlášky, ukončení simulace apod.).

Druhým parametrem metody je název funkce FMU, která byla volána před zavoláním metody. Název je následně použit ve varováních nebo chybových hláškách.

Ukončení simulace je prováděno pomocí příkazu

```
1 set_param(bdroot, 'SimulationCommand', 'stop');
```

kde `bdroot` je funkce vracející název nejvyšší úrovně aktuálního modelu („top-level system“), druhým parametrem je vlastnost modelu, která má být upravena (`SimulationCommand`) a posledním parametrem je akce, která má být provedena. Možné hodnoty posledního parametru jsou `'start', 'stop', 'pause', 'continue', 'connect' (to a target), 'update', 'WriteDataLogs'`, které provedou akce v odpovídajícím pořadí - spuštění simulace, ukončení simulace, pozastavení simulace, pokračovat v simulaci, připojení (pro Real-Time Windows Target simulaci), aktualizaci proměnných a zápis proměnných do pracovního prostředí („workspace“).

V případě, že se ukončení simulace korektně nepodaří (v průběhu se vyskytne výjimka), je tato výjimka zachycena a doplněna k původní chybové hlášce. Tím je zajištěno, že uživatel bude informován o všech problémech, které nastaly v průběhu simulace a je možné tyto problémy předat k řešení. Pokud je při simulaci proveden zápis do logu, je o tom uživatel informován pomocí varovného nebo chybového hlášení.

4.12. Dynamická knihovna - rozhraní Matlab/FMU

Pro zajištění rozhraní mezi prostředím Matlab a FMU byla vytvořena dynamická knihovna (dynamic-link library pro Windows, shared object pro Linux).

Vytvoření tohoto rozhraní bylo nutné z několika důvodů, mezi nejzásadnější patří problém s voláním funkce `fmiInstantiateModel( .. )`, která má jako vstupní parametr `fmiCallbackFunctions functions`, což je struktura ukazatelů na funkce zpětného volání („callback functions“). Struktura obsahuje ukazatele na funkce `logger`, `allocateMemory` a `freeMemory`. V prostředí Matlab není možné vytvořit strukturu ukazatelů na funkce a předat ji parametrem při volání funkce dynamické knihovny.

Dalším problémem byly datové typy proměnných, které jsou definovány standardem FMI. Avšak Matlab neumožňuje definovat vlastní datové typy a předávat je při volání funkcí dynamické knihovny.

Dynamická knihovna byla vytvořena na základě `FMU Software Development Kit`, což je volně dostupná vývojová sada určená k demonstraci použití FMU. Dále může sloužit také jako výchozí bod pro vývoj aplikací pro import a export FMU, takto byla

sada použita při vývoji nástroje FMUtoolbox. FMU SDK byla rozšířena a upravena pro potřeby nástroje FMUtoolbox.

Zdrojové kódy dynamické knihovny jsou shodné pro všechny její verze (32-bitové a 64--bitové DLL pro MS Windows, SO pro Linux). V případě nutnosti odlišného zdrojového kódu pro MS Windows a Linux bylo využito direktiv preprocesoru pro řízení podmíněného překladu.

```
#if defined __MINGW32__ || defined _MSC_VER
    // zdrojovy kod pro MS Windows
#else
    // zdrojovy kod pro Linux
#endif
```

Tímto způsobem je vyřešeno např. volání různých funkcí pro operační systémy MS Windows a Linux (např. funkce `LoadLibrary( .. )` a `dlopen( .. )`), načítání odlišných knihoven (např. `windows.h` a `dlfcn.h`) a další.

Dynamické knihovny pro operační systém MS Windows jsou kompilovány pomocí nástroje Microsoft Visual Studio, dynamická knihovna pro operační systém Linux je kompilována pomocí utility `make`, která volá kompilátor `g++`.

4.13. Zpracování výjimek

Při realizaci nástroje FMUtoolbox byla důsledně prováděna kontrola parametrů všech funkcí a i detekce možných výjimečných stavů. Po detekci např. parametru s chybným datovým typem nebo jeho hodnotou mimo definovaný rozsah je vyvolána výjimka („exception“), která je buď odchycena v bloku kódu `try .. catch` nebo případně způsobí ukončení prováděné akce. Příkazy `try .. catch` jsou používány pouze v případě, kdy nelze nestandardní stav detekovat jinými metodami.

V případě potřeby vyvolání výjimky je využita třída `MException` a související příkazy `throw`, `rethrow` a `addCause`. Příklad vyvolání výjimky je na následujících řádcích.

```
% check parameters
if (nargin ~= 2)
        msgIdent = 'FMUclass:set_tNextTimeEvent';
        msgText = 'Wrong count of parameters provided.';
        err = MException(msgIdent,msgText);
        throw(err);
end % if
```

Tato část zdrojového kódu vyvolá výjimku v případě, že počet parametrů při volání funkce není rovný dvěma. Parametrem konstruktoru objektu třídy `MException` je identifikátor chybové zprávy (`msgIdent`), což je textový řetězec obsahující identifikaci funkce, která výjimku vyvolala. Tento řetězec je používán v doporučeném formátu `component:mnemonic`, kde `component` je komponenta Matlabu (v tomto případě třída `FMUclass`) a `mnemonic` je podrobnější popis zdroje chyby (v tomto případě označení metody třídy). V některých případech je využito i více upřesnění zdroje výjimky (za další dvojtečkou). Následně je výjimka vyvolána pomocí funkce `throw`. Příkaz `rethrow` se používá v případě zachy-

cení výjimky pomocí bloku `try .. catch`, je provedeno doplnění dalšího popisu výjimky pomocí funkce `addCause`. Tato funkce rozšíří původní objekt výjimky o dodatečný identifikátor a zprávu, aby mohl být lépe detekován zdroj výjimky a její důvod. Vyvolání zkombinovaných výjimek je následně realizováno pomocí funkce `rethrow`.

5. Křížová kontrola nástroje FMUtoolbox

Křížová kontrola („Cross-Check“) je soubor pravidel, jejichž cílem je zdokumentovat a zlepšit kvalitu nástrojů využívajících standard FMI, pomoci výrobcům zlepšit zjistit problémy s podporou standardu FMI, detekovat a opravit problémy v FMI standardu, zlepšit reputaci FMI a upřesnit pravidla zveřejňování informací o nástrojích na stránkách fmi-standard.org [17].

Dokumentem [17] je definován proces kontroly vzájemné kompatibility nástrojů využívajících standard FMI. Pro každý nástroj podporující export FMU jsou na server fmi-standard.org nahrány FMU exportované tímto nástrojem. Každé FMU je doplněno o metadata, zejména referenční výstupní data simulace modelu, vstupní signály, parametry simulace apod. V dokumentu je definována přesná struktura těchto souborů (většinou ve formátu CSV) a i upřesňující pravidla pro formát CSV (oddělovač, číselný formát, použití uvozovek pro názvy proměnných apod.). Při provádění křížové kontroly je proveden import FMU do testovaného nástroje a výše uvedené soubory jsou použity pro simulaci. V případě úspěšného importu FMU a souladu výstupu simulace s referenčními výstupy simulace minimálně tří FMU exportovaných jedním nástrojem jsou nástroje považovány za kompatibilní. Pokud je nástroj pro import FMU kompatibilní minimálně se třemi nástroji pro export FMU, obdrží nástroj status „Available“ (zeleně podbarvený), který značí, že nástroj je dostupný a kompatibilní se standardem FMI.

Výsledky křížové kontroly jsou po provedení testů nahrány na server fmi-standard.org, kde jsou volně k dispozici pro uživatele nástrojů a standardu FMI.

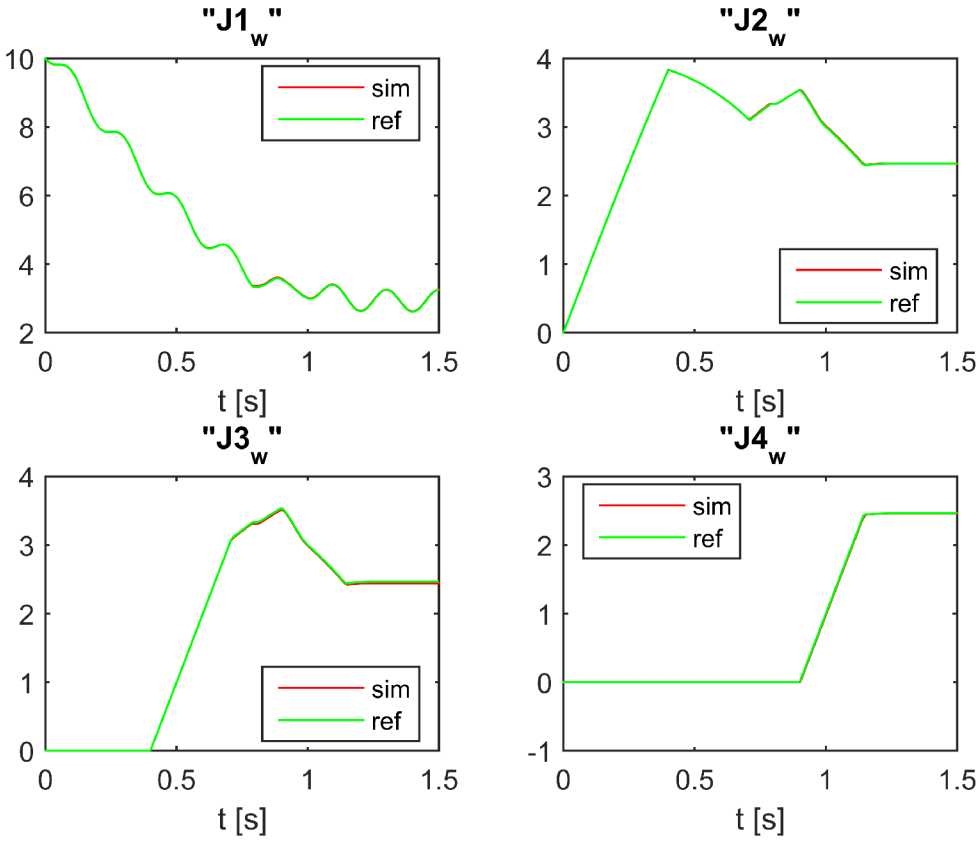
5.1. Nástroje pro křížovou kontrolu

Pro provádění křížové kontroly byly vytvořeny skripty v jazyce Matlab. Skript `crossCheck.m` očekává adresářovou strukturu nástroj-verze_nástroje-složka_FMU (což odpovídá struktuře složek po stažení vzorových FMU z serveru server fmi-standard.org. Tato struktura je skriptem načtena a uživatel je vyzván k výběru nástroje a jeho verze. Pro všechny FMU daného nástroje a verze je provedena kontrola pomocí skriptu `simulateAndSaveData.m`.

Tento skript využívá rozhraní pro simulaci FMU pomocí příkazové řádky v prostředí Matlab. Pomocí konstruktoru vytvoří novou instanci třídy FMUclass a následně načte FMU. Dalším krokem je načtení parametrů simulace ze souboru s příponou `opt` a provedení potřebných nastavení simulace pro dané FMU pomocí metody `setSimulationParameters( .. )` třídy `FMUclass`. Následně je provedeno načtení vstupních dat a referenčních výstupních dat. Po provedení simulace jsou výsledky exportovány ve formátu CSV a jsou uloženy grafy výsledných průběhů výstupů do formátu `png`.

5.2. Výsledky kontroly

Pro ilustraci použití nástrojů pro křížovou kontrolu bylo zvoleno FMU `CoupledClutches` exportované z nástroje Dymola. Pomocí výše uvedených skriptů byla provedena simulace FMU a porovnání výstupů FMU s referenčními hodnotami. Na obrázku 5.1 jsou zobrazeny výsledné průběhy simulovaných a referenčních hodnot výstupů. Průběhy jsou téměř



Obrázek 5.1: Křížová kontrola nástroje FMUtoolbox a FMU CoupledClutches

totožné a překrývají se, rozdíly mezi průběhy jsou znatelné pouze při podrobném zkoumání. Rozdíly mezi průběhy jsou způsobeny delší periodou vzorkování při simulaci FMU, při kratší periodě vzorkování by byly průběhy naprosto totožné.

Křížová kontrola byla provedena pro všechny dostupné vzorové FMU, výsledky této kontroly jsou uvedeny v příloze této práce. Souhrnné výsledky křížové kontroly jsou uvedeny v tabulce 5.1. Pro každou kombinaci nástroje a platformy jsou uvedena buď tři čísla, nebo zkratka `N/A`. První číslo určuje, kolik FMU bylo korektně odsimulováno, druhé určuje počet odmítnutých FMU a třetí číslo určuje počet FMU, u kterých selhala simulace. Tento způsob interpretace výsledků křížové kontroly je definován v [17]. Navíc byla doplněna zkratka `N/A`, která značí, že pro danou platformu není k dispozici žádné FMU exportované určeným nástrojem.

V následujícím textu jsou popsány problémy zjištěné při provádění křížové kontroly, zejména je zdůvodněno zamítnutí žlutě označených FMU v tabulce 5.1.

platforma	win64			win32			linux64		
AMESim	2	0	0	3	0	0	3	0	0
CATIA	0	3	0	0	3	0	N/A		
ControlBuild	0	3	0	0	3	0	N/A		
Dymola	7	0	0	7	0	0	N/A		
FMIToolbox_MATLAB	N/A			0	3	0	N/A		
JModelica.org	N/A			4	0	0	0	3	0
LMS_Virtual.Lab_Motion	N/A			0	1	0	N/A		
MapleSim	3	0	0	3	0	0	0	2	0
OpenModelica	N/A			3	0	0	N/A		
OPTIMICA_Studio	N/A			0	3	0	N/A		
Silver	N/A			1	2	0	N/A		
SimulationX	4	0	1	4	0	1	N/A		

Tabulka 5.1: Křížová kontrola nástroje FMUtoolbox

Všechny FMU exportované z nástroje CATIA neměly žádné proměnné označené jako výstup a v souboru s referenčními daty je velké množství proměnných (desítky, např. 80), což je v rozporu s [17], bod 9 (je doporučeno maximálně 10 proměnných). Proto jsou všechny FMU exportované nástrojem CATIA označeny jako zamítnuté („rejected“) a to jak FMU pro platformu `win64`, tak i FMU pro platformu `win32`.

FMU exportované nástrojem ControlBuild se nepodařilo simulovat, nebylo možné načíst XML soubor s popisem modelu. Formát souboru XML je odlišný od formátu podporovaného prostředím Matlab. Z tohoto důvodu jsou všechny FMU exportované nástrojem ControlBuild označeny jako zamítnuté („rejected“).

U FMU exportovaných nástrojem SimulationX nebyl poskytnut soubor s nastaveními simulace (soubor typu „opt“) a zároveň chybí soubor ReadMe, takže bylo nutné zvolit vlastní parametry simulace. Navíc soubory s referenčními výsledky simulace mají špatné formátování (názvy proměnných nejsou v uvozovkách) a používají formátování UTF-16 LE. I přes tyto evidentní rozpory s [17] se podařilo všechny FMU mimo jednoho odsimulovat (jak FMU pro platformu `win64`, tak i pro platformu `win32`).

Všechny FMU exportované nástrojem JModelica.org pro platformu `win32` neměly potřebné proměnné nastaveny jako výstup, avšak i přes tento nedostatek se podařilo všechny vzorové FMU korektně odsimulovat. FMU exportované nástrojem JModelica.org pro platformu `linux64` se nepodařilo odsimulovat, docházelo k problémům s přístupem k paměti.

FMU exportované nástrojem FMIToolbox_MATLAB byly zamítnuty, protože vzorové FMU a referenční soubory nejsou formálně správně (chybné názvy souborů, velké množství výstupních proměnných, chybějící soubor s nastavením simulace apod.).

Ze stejného důvodu bylo zamítnuto i jediné vzorové FMU exportované nástrojem LMS_Virtual.Lab_Motion a všechny FMU exportované nástrojem OPTIMICA_Studio.

Dvě FMU exportované nástrojem Silver byly zamítnuty kvůli nedostatkům v referenčních datech (chyběly některé průběhy vstupních proměnných apod.), avšak jedno FMU se podařilo korektně odsimulovat.

5.3. Vyhodnocení křížové kontroly

Při provádění testů křížové kontroly byla otestována kompatibilita všech dostupných vzorových FMU s nástrojem FMUtoolbox. Souhrn výsledků kontroly je v tabulce 5.1. V této tabulce jsou dvě platformy (`win32` a `win64`) označeny zeleně, což značí ověřenou funkčnost nástroje pro danou platformu (v souladu s [17] je platforma označena zeleně, pokud se podaří importovat minimálně tři FMU od každého z nejméně tří nástrojů). Pro platformu `linux64` nebyla křížová kontrola průkazná a to zejména z důvodu malého počtu dostupných vzorových FMU pro tuto platformu. Platforma `linux32` byla záměrně vynechána, protože dle [18] není Matlab pro 32 bitovou verzi operačního systému Linux podporován a ani do budoucna nebude podporován. Proto tedy nemá smysl tuto platformu v souvislosti s nástrojem FMUtoolbox uvádět.

Provedením křížové kontroly byla doložena kompatibilita nástroje FMUtoolbox s FMU exportovanými nástroji různých výrobců a tím i soulad se standardem FMI [8]. V průběhu prováděním křížové kontroly bylo zjištěno, že někteří výrobci zásadním způsobem nedodržují pravidla křížové kontroly [17]. Nedostatky jsou zejména v podpůrných metadatech pro provedení kontroly (chybějící soubory, špatný formát souborů a podobně) a častým problémem je označení proměnných jako výstup (některé FMU nemají žádnou proměnnou označenou jako výstup, některé naopak mají desítky proměnných označených jako výstup).

Při provádění křížové kontroly se ukázalo, že někteří výrobci nejsou schopni dodržet ani jednoduchá pravidla křížové kontroly (jedenáct bodů v [17]), takže jsou na místě pochybnosti o kompatibilitě FMU exportovaných nástroji těchto výrobců se standardem FMI. Naopak někteří výrobci mají standard FMI velmi dobře implementován a stejně tak důsledně dodržují pravidla křížové kontroly včetně nepovinných prvků.

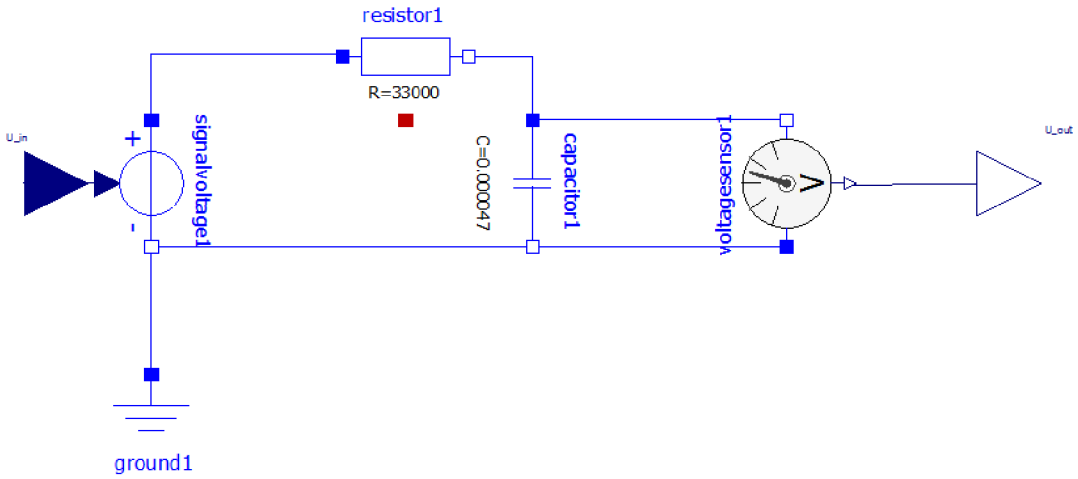
I přes uvedené problémy je však možné říci, že křížová kontrola plní většinu svých účelů popsaných v [17]. Jedinou výjimkou je zlepšení reputace standardu FMI, kde stále existují velké možnosti vylepšení, stále je mnoho nástrojů, které standard FMI dle výrobce podporují, avšak křížová kontrola neproběhne korektně (a to ve více nástrojích, nejedná se o problém FMUtoolboxu).

6. Praktické použití nástroje FMUtoolbox

V této kapitole budou popsány možnosti použití nástroje FMUtoolbox včetně demonstrace tohoto použití.

6.1. Model RC článku

Jako první příklad byl zvolen RC článek, který již byl použit pro porovnání kauzálního a akauzálního modelování v kapitole 1.3. Model v jazyce Modelica byl upraven tak, aby bylo možné exportovat FMU, které bude mít potřebné vstupy a výstupy. Upravené schéma je na obrázku 6.1.

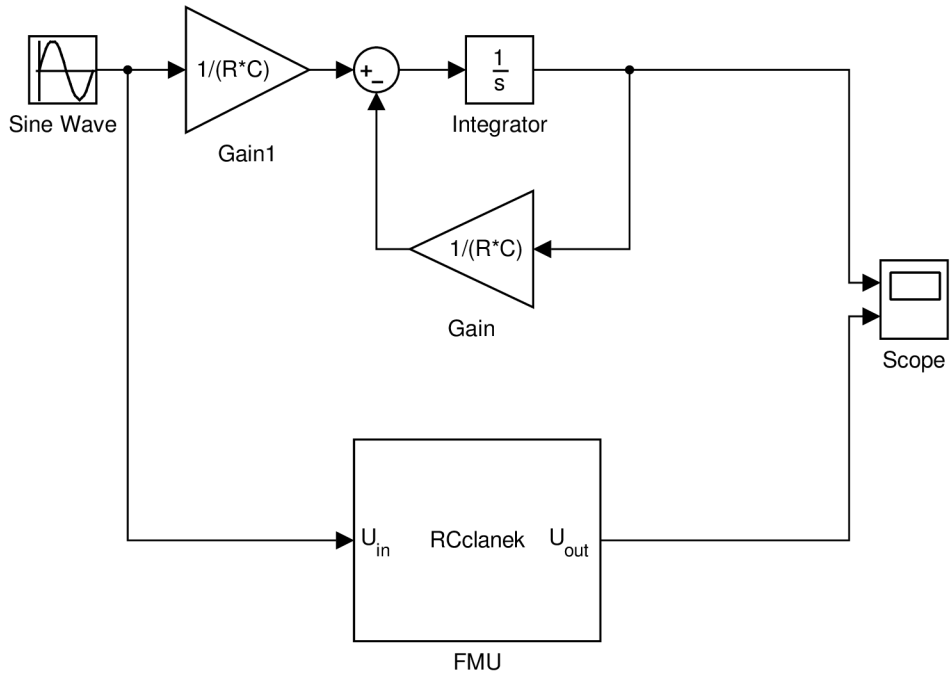


Obrázek 6.1: Schéma RC článku v jazyce Modelica upraveného pro export FMU

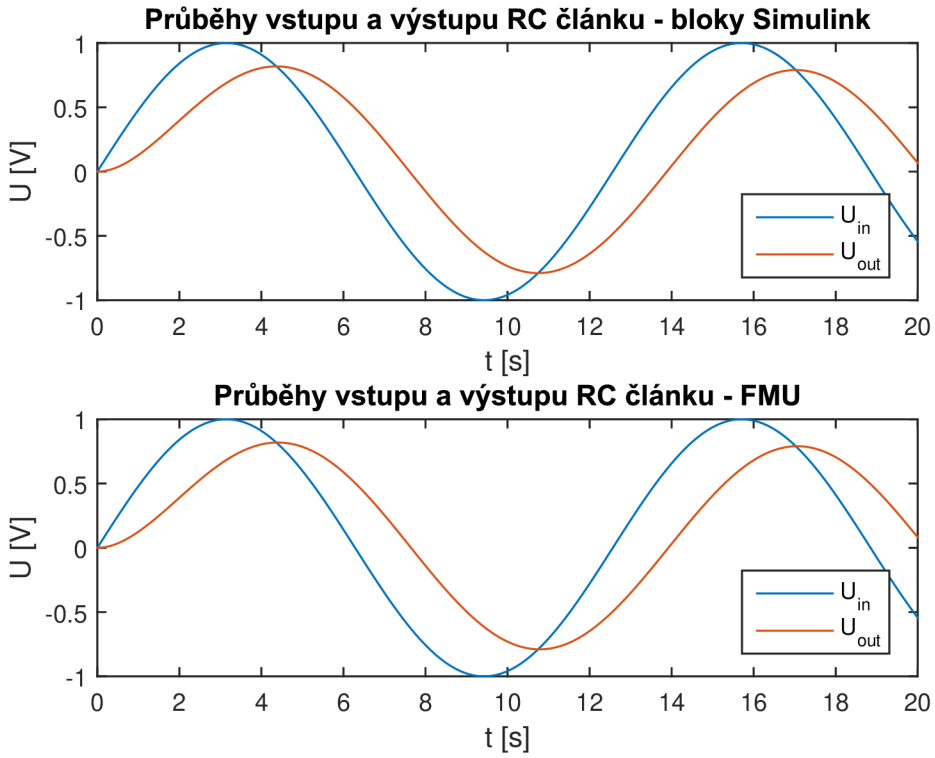
Tento model byl exportován jako FMU a následně s využitím nástroje FMUtoolbox načten do prostředí Matlab/Simulink. V tomto prostředí bylo sestaveno simulační schéma pro ověření funkčnosti simulace FMU, viz obr. 6.2.

Následně byla provedena simulace tohoto schámatu, výsledné průběhy jsou na obr. 6.3.

Byla ověřena funkčnost nástroje FMUtoolbox pro simulaci jednoduchého FMU - modelu RC článku. Průběh výstupního signálu z bloku FMU je shodný s výstupem z modelu RC článku realizovaného pomocí bloků Simulink.



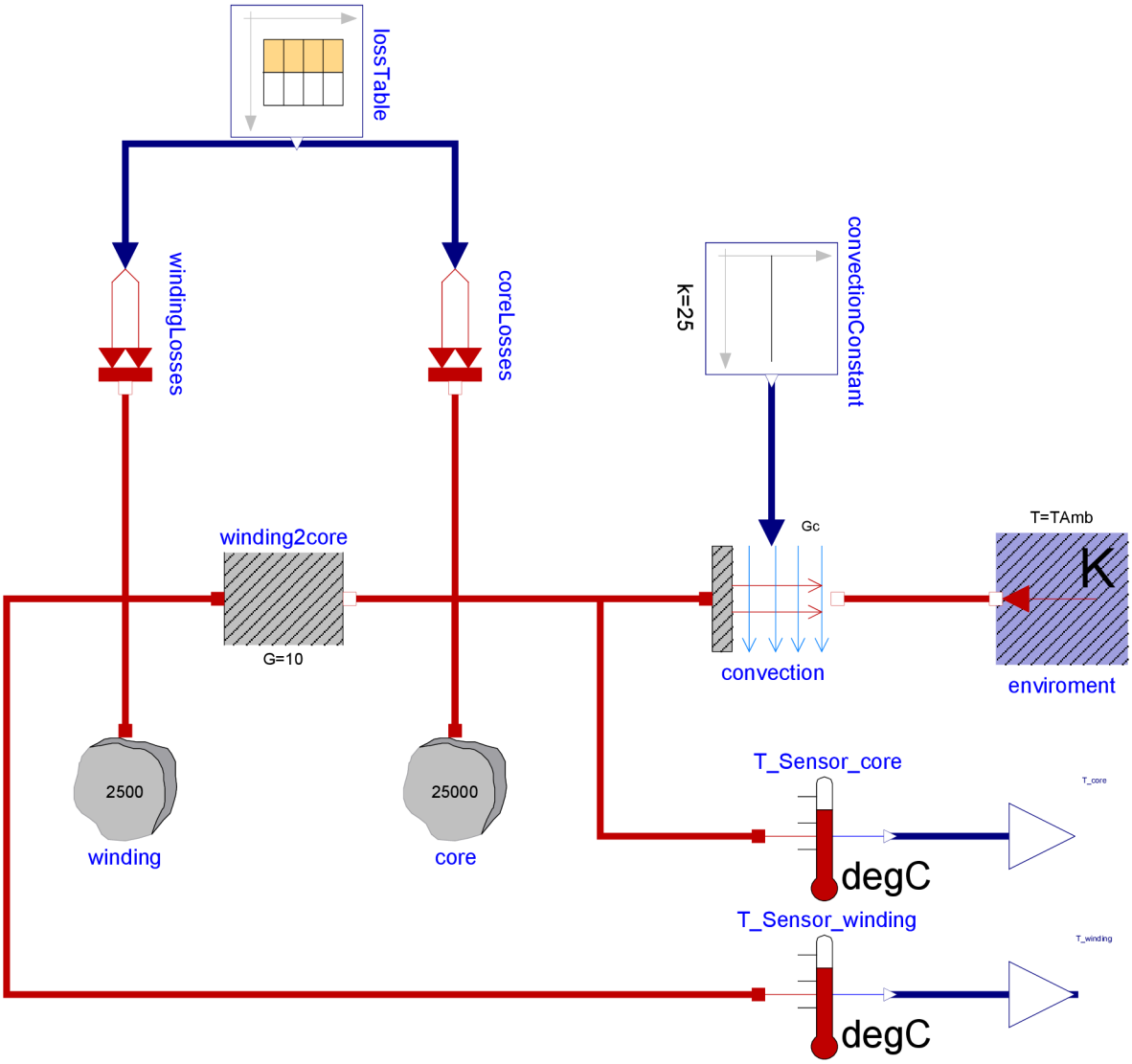
Obrázek 6.2: Schéma Simulink pro ověření simulace FMU - RC článek



Obrázek 6.3: Průběhy vstupních a výstupních signálů simulace RC článku

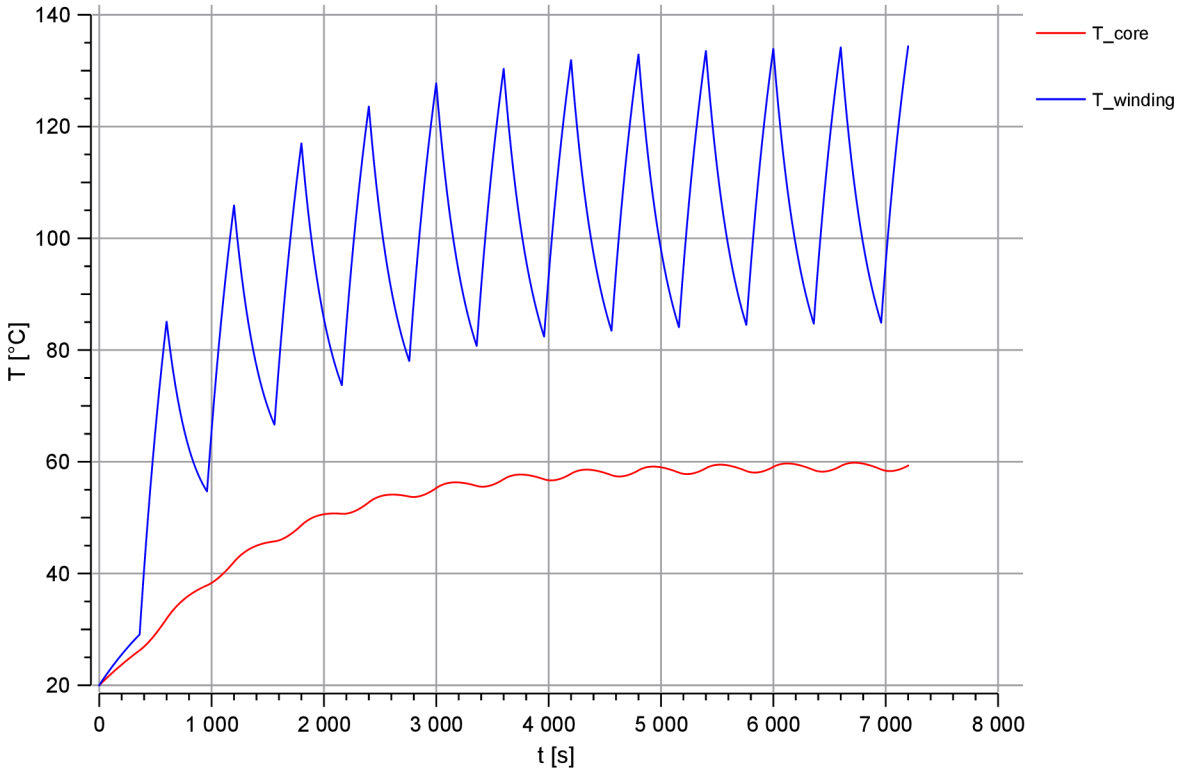
6.2. Tepelné ztráty elektromotoru

Jako druhý příklad byl vybrán model tepelných ztrát elektromotoru, který byl odvozen od vzorového modelu Modelica.Thermal.HeatTransfer.Examples.Motor z knihovny modelů OpenModelica. Vzorový model byl upraven tak, aby teploty vinutí a statoru byly výstupem FMU. Výsledné schéma modelu je na obr. 6.4.

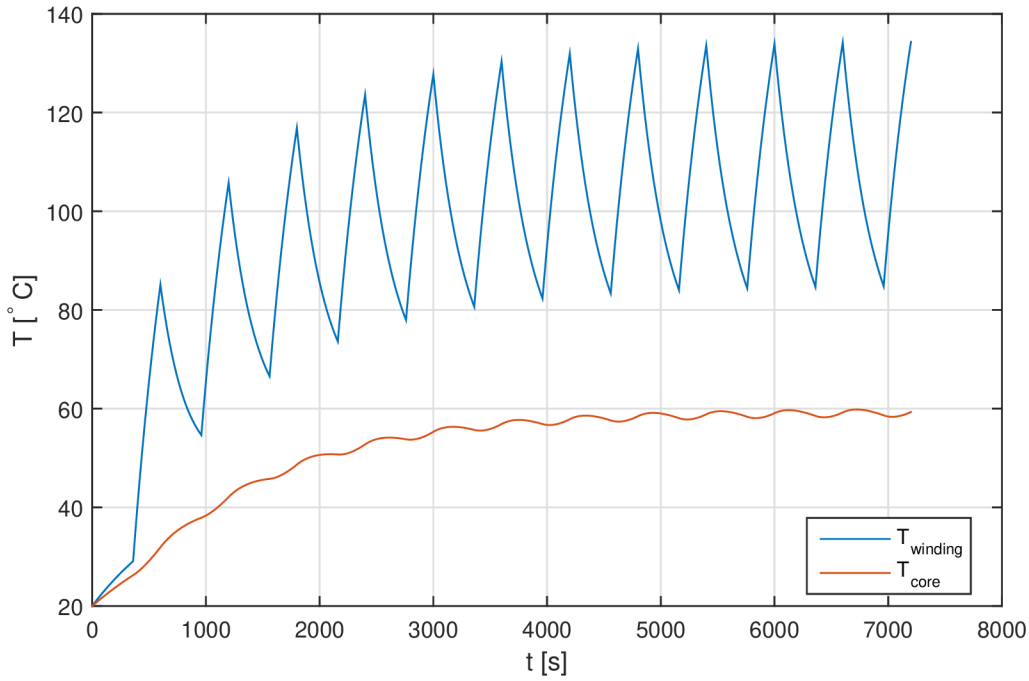


Obrázek 6.4: Model tepelných ztrát motoru

Byla provedena simulace tepelného modelu v prostředí OpenModelica a následně byl model exportován jako FMU. S využitím nástroje FMUtoolbox byl model simulován v prostředí Matlab/Simulink. Na obrázcích 6.5 a 6.6 je možné porovnat průběhy výstupů simulace v prostředí OpenModelica a v prostředí Matlab/Simulink.



Obrázek 6.5: Graf výstupů simulace tepelného modelu motoru v prostředí OpenModelica

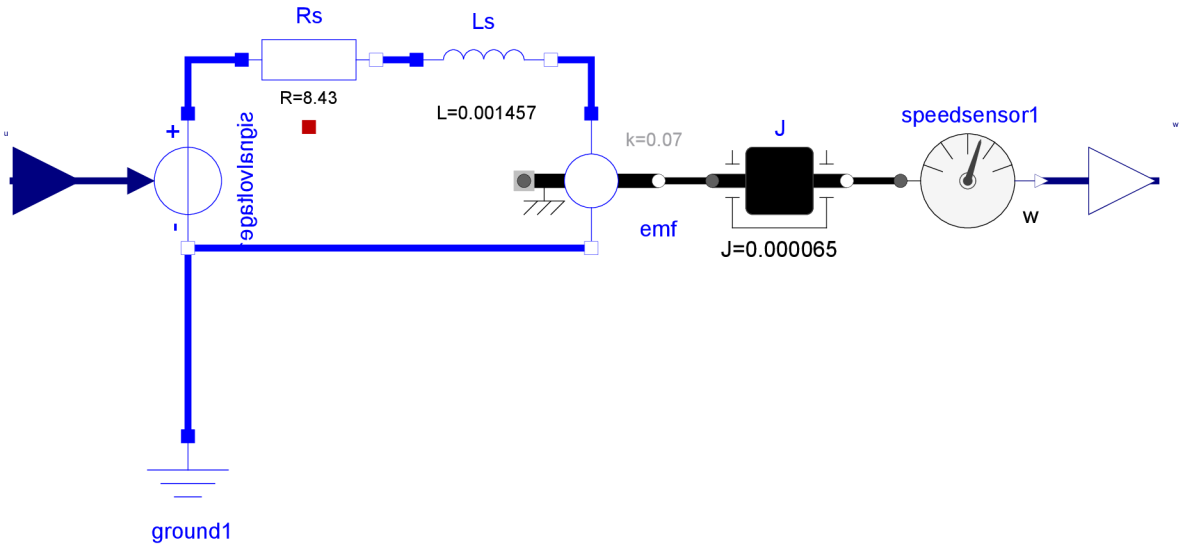


Obrázek 6.6: Graf výstupů simulace tepelného modelu motoru v prostředí Matlab/Simulink

6.3. Řízení stejnosměrného motoru

Třetí příklad použití nástroje FMUtoolbox se zabývá využitím tohoto nástroje pro návrh regulátoru pro řízení otáček stejnosměrného motoru a pro ověření jeho funkčnosti.

V prostředí OpenModelica byl vytvořen model stejnosměrného motoru, viz obr. 6.7. Tento model byl exportován jako FMU a následně s využitím nástroje FMUtoolbox načten



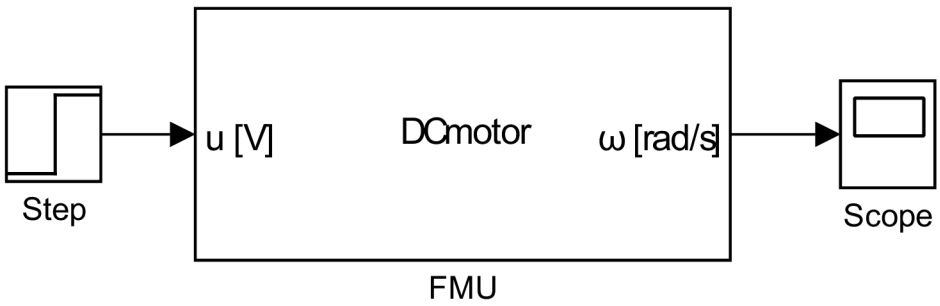
Obrázek 6.7: Model stejnosměrného motoru v prostředí OpenModelica

do prostředí Matlab/Simulink. V dalším kroku byla provedena identifikace této soustavy metodou využívající přechodovou charakteristiku. Na vstup bloku FMUtoolbox byl připojen jednotkový skok, výstup bloku FMUtoolbox byl použit pro identifikaci (viz obr. 6.8). Výsledný přenos soustavy byl určen ve tvaru

$$F_s(p) = \frac{14{,}28}{0{,}111p + 1}. \tag{6.1}$$

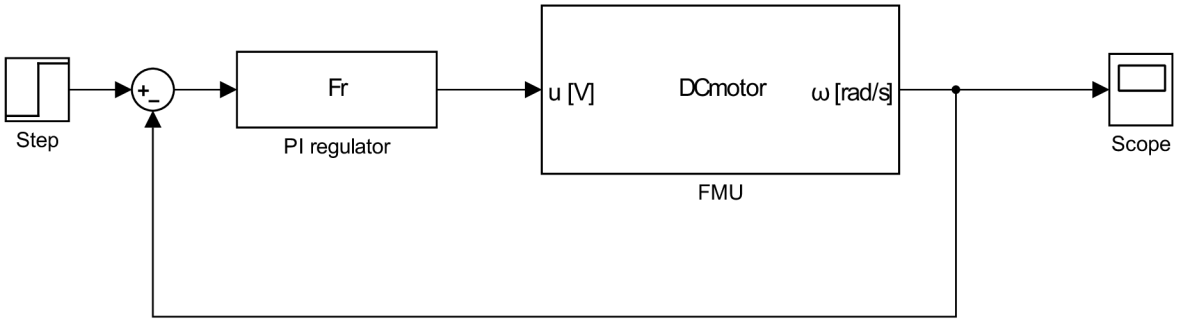
Pro tuto soustavu byl navržen PI regulátor pomocí metody požadovaného rozložení pólů uzavřeného obvodu. Regulátor byl navržen ve tvaru

$$F_R(p) = \frac{2{,}8(1 + 0{,}027p)}{p}. \tag{6.2}$$

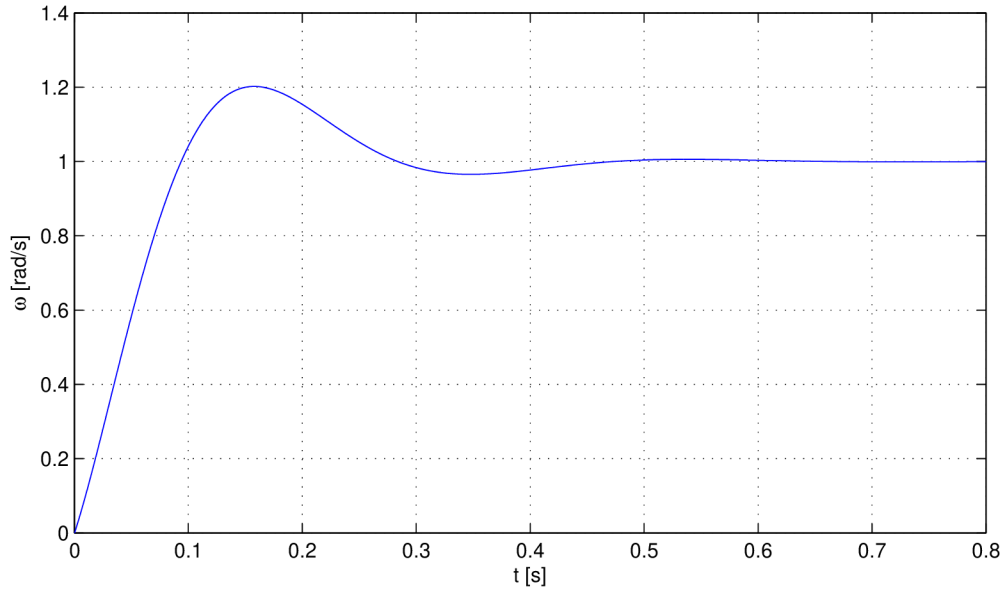


Obrázek 6.8: Schéma zapojení pro identifikaci přenosu stejnosměrného motoru

Následně byl regulátor zapojen do uzavřené regulační smyčky s modelem motoru (viz obr. 6.9) pro ověření funkčnosti regulace. Odezva na jednotkový skok žádané hodnoty je na obrázku 6.10.



Obrázek 6.9: Schéma zapojení pro ověření návrhu regulátoru



Obrázek 6.10: Průběh odezvy na skok žádané hodnoty uzavřené smyčky regulátoru a modelu motoru v jazyce Modelica

V tomto příkladě bylo prakticky ukázáno, že nástroj FMUtoolbox je možné použít při návrhu a ověřování regulátorů. Jeho použití je výhodné zejména při návrhu regulátoru pro složitější soustavu, u které je možné sestavit přesný akauzální model v jazyce Modelica. Návrh regulátoru je vhodné provést s využitím jednoduššího modelu, v tomto případě byl použit operátorový přenos prvního řádu, který byl získán identifikací akauzálního modelu. Navržený regulátor je možné ověřit v zapojení s vytvořeným přesným modelem v jazyce Modelica pomocí nástroje FMUtoolbox. Ověření je možné provést v různých pracovních bodech a pomocí uživatelského rozhraní je možné změnit parametry motoru.

7. Závěr

Tato diplomová práce se zabývá použitím fyzikálních (akauzálním) modelů v jazyce Modelica v prostředí Matlab/Simulink. Pro doplnění prostředí Matlab/Simulink o možnost použití modelů v jazyce Modelica byl vytvořen nástroj FMUtoolbox, který umožňuje simulovat modely v jazyce Modelica exportované do formátu Functional Mock-up Unit dle standardu Functional Mock-up Interface.

Vytvořený nástroj implementuje veškerou funkcionalitu potřebnou pro import a simulaci Functional Mock-up Unit v prostředí Matlab/Simulink. Zároveň bylo realizováno uživatelské rozhraní dle provedeného návrhu, a to jak jeho grafická část, tak i část pro ovládání nástroje z příkazové řádky prostředí Matlab/Simulink.

Při křížové kontrole nástroje FMUtoolbox se vzorovými FMU bylo doloženo, že nástroj je kompatibilní se standardem Functional Mock-up Interface na platformách `win32` a `win64`, pro platformu `linux64` nebyla kontrola průkazná kvůli malému množství vzorových FMU, avšak nástroj je plně funkční i na této platformě.

K nástroji FMUtoolbox byl vytvořen podrobný uživatelský návod, který je umístěn v příloze této práce.

Praktické použití nástroje FMUtoolbox bylo demonstrováno na třech příkladech. V prvním z nich jsou ukázány modely RC článku v jazyce Modelica a v prostředí Simulink a průběhy získané simulací obou modelů jsou shodné. Druhý příklad se zabývá složitějším modelem tepelných ztrát elektromotoru, který je simulován v prostředí OpenModelica a v prostředí Matlab. Výsledné průběhy simulace byly porovnány a jsou shodné. Ve třetím z nich je ilustrován postup návrhu regulátoru otáček pro stejnosměrný motor. Regulátor je následně ověřen v kombinaci s modelem motoru v jazyce Modelica, tento test je označován jako Model in the Loop (MIL). Na tomto praktickém příkladě bylo ukázáno, že nástroj FMUtoolbox je možné výhodně použít pro testy MIL a následně i Software in the Loop (SIL) a Processor in the Loop (PIL), ve všech případech může být realizace modelu procesu shodná, mění se pouze realizace regulátoru.

Prostřednictvím křížové kontroly a praktických příkladů bylo doloženo, že vytvořený nástroj FMUtoolbox je funkční, je implementován v souladu se standardem FMI a je kompatibilní s většinou nástrojů pro export FMU.

Pro simulace Hardware in the Loop (HIL) se nabízí možnost vytvoření sady nástrojů pro realizaci hardwarového modelu procesu (procesor s potřebnými vstupy a výstupy pro připojení regulátoru a patřičným SW vybavením), který by byl realizován na základě modelů v jazyce Modelica a standardu FMI.

Další vývoj nástroje FMUtoolbox může také zahrnovat implementaci standardu FMI 2.0, který je v současné době postupně doplňován do nástrojů pro export FMU.

Literatura

[1] MODELICA ASSOCIATION. *Modelica - A Unified Object-Oriented Language for Systems Modeling: Language Specification*, 2012.

[2] TILLER, Michael. *Modelica by Example*. [Online], 2014 [cit. 2014-12-02]. Dostupné z: <http://book.xogeny.com>

[3] FERNSTROM, Anders, AXELSSON, Ingemar, FRITZSON, Peter, aj. *OMNotebook – Interactive WYSIWYG Book Software for Teaching Programming*. [Online], 2006 [cit. 2014-12-21]. Dostupné z: <https://modelica.org/publications/papers/2006-03-10-Fernstrom-etal-TeachingWorkshop-NotebookTeaching.pdf>

[4] SANDELIN, Eva-Lena Lengquist, MONEMAR, Susanna, FRITZSON, Peter, aj. *DrModelica An Interactive Environment for Learning Modelica and Modeling using MathModelica*. [Online], 2003 [cit. 2014-12-21]. Dostupné z: <https://modelica.org/publications/papers/2006-03-10-Fernstrom-etal-TeachingWorkshop-NotebookTeaching.pdf>

[5] FRITZSON, Peter, POP, Adrian, ASGHAR, Adeel, aj. *OpenModelica Users Guide*. [Online], 2014 [cit. 2014-12-22]. Dostupné z: <https://openmodelica.org/svn/OpenModelica/trunk/doc/OpenModelicaUsersGuide.pdf>

[6] MATTSSON, Sven Erik, ELMQVIST, Hilding, OTTER, Martin. *Physical system modeling with Modelica*. *Control Engineering Practice*, ročník 6, č. 4, 1998: s. 501 – 510. ISSN 0967-0661. doi:http://dx.doi.org/10.1016/S0967-0661(98)00047-1. Dostupné z: <http://www.sciencedirect.com/science/article/pii/S0967066198000471>

[7] BLOCHWITZ, Torsten, OTTER, Martin. *The Functional Mockup Interface for Tool independent Exchange of Simulation Models*. [Online], 2011 [cit. 2014-12-02]. Dostupné z: <http://www.researchgate.net/publication/225025178_The_Functional_Mockup_Interface_for_Tool_independent_Exchange_of_Simulation_Models>

[8] MODELISAR. *Functional Mock-up Interface for Model Exchange*. [Online], 2010 [cit. 2014-12-23]. Dostupné z: <https://svn.modelica.org/fmi/branches/public/specifications/v1.0/FMI_for_ModelExchange_v1.0.pdf>

[9] Claytex. *Dymola and Simulink*. [Online], 2014 [cit. 2014-12-23]. Dostupné z: <http://www.claytex.com/products/dymola/features/simulink-interface/>

[10] Modelon. *FMI Toolbox for MATLAB/Simulink*. [Online], 2014 [cit. 2014-12-23]. Dostupné z: <http://www.modelon.com/products/fmi-toolbox-for-matlab/>

[11] THE MATHWORKS, Inc. *Matlab: Object-Oriented Programming.* [Online], 2015 [cit. 2015-04-15].
Dostupné z: <http://www.mathworks.com/help/pdf_doc/matlab/matlab_oop.pdf>

[12] *NSIS Users Manual.* [Online], 2015 [cit. 2015-04-17].
Dostupné z: <http://nsis.sourceforge.net/Docs/>

[13] THE MATHWORKS, Inc. *Matlab: Creating Graphical User Interfaces.* [Online], 2015 [cit. 2015-04-17].
Dostupné z: <http://www.mathworks.com/help/pdf_doc/matlab/buildgui.pdf>

[14] THE MATHWORKS, Inc. *Simulink: Developing S-Functions.* [Online], 2015 [cit. 2015-04-16].
Dostupné z: <http://www.mathworks.com/help/pdf_doc/simulink/sfunctions.pdf>

[15] Atlassian. *Comparing Workflows: Gitflow Workflow.* [Online], 2015 [cit. 2015-03-11].
Dostupné z: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

[16] DRIESSEN, Vincent. *A successful Git branching model.* [Online], 2010 [cit. 2015-04-16].
Dostupné z: <http://nvie.com/posts/a-successful-git-branching-model/>

[17] JUNGHANNS, Andreas. *FMI Cross Check: How to Improve FMI Compliance.* [Online], 2014 [cit. 2015-03-10].
Dostupné z: <https://svn.fmi-standard.org/fmi/branches/public/CrossCheck_Results/FMI_Cross_Check_Rules_v3_2014_07_31.pdf>

[18] THE MATHWORKS, Inc. *Platform Road Map for the MATLAB and Simulink Product Families.* [Online], 2015 [cit. 2015-04-13].
Dostupné z: <http://www.mathworks.com/support/sysreq/roadmap.html>

Seznam použitých symbolů a zkratek

C	elektrická kapacita
CSV	Comma-separated values (hodnoty oddělené čárkami)
DLL	Dynamic-link library
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FPGA	Field Programmable Gate Array (Programovatelné hradlové pole)
GUI	Graphical user interface (grafické uživatelské rozhraní)
$\boldsymbol{m}$	vektor diskrétních stavů
MIL	Model in the Loop
NSIS	Nullsoft Scriptable Install System
OMC	Advanced Interactive OpenModelica Compiler
OMEdit	OpenModelica Connection Editor
OMShell	Interactive OpenModelica Shell
PIL	Processor in the Loop
R	elektrický odpor
SIL	Software in the Loop
SO	Shared Object
t	čas
$u(t)$	elektrické napětí v čase
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WYSIWYG	What-You-See-Is-What-You-Get
$\boldsymbol{x}$	vektor spojitých stavů
XML	Extensible Markup Language

Seznam příloh

1. Příklad popisu modelu (xml soubor)
2. Uživatelský manuál nástroje FMUtoolbox pro Simulink (FMUtoolbox for Simulink - User Guide)
3. Nosič CD
   (a) `DiplomovaPrace` - elektronická verze diplomové práce
   (b) `FMUtoolbox` - instalační soubory nástroje FMUtoolbox
   (c) `CrossCheck` - podrobné výsledky křížové kontroly
   (d) `PraktickePouziti` - modely a další soubory s ukázkami použití nástroje FMUtoolbox
   (e) `PopisModelu` - elektronická verze Příkladu popisu modelu (xml soubor)
   (f) `UzivatelskyManual` - elektronická verze uživatelského návodu k nástroji FMUtoolbox