Univerza v Ljubljani

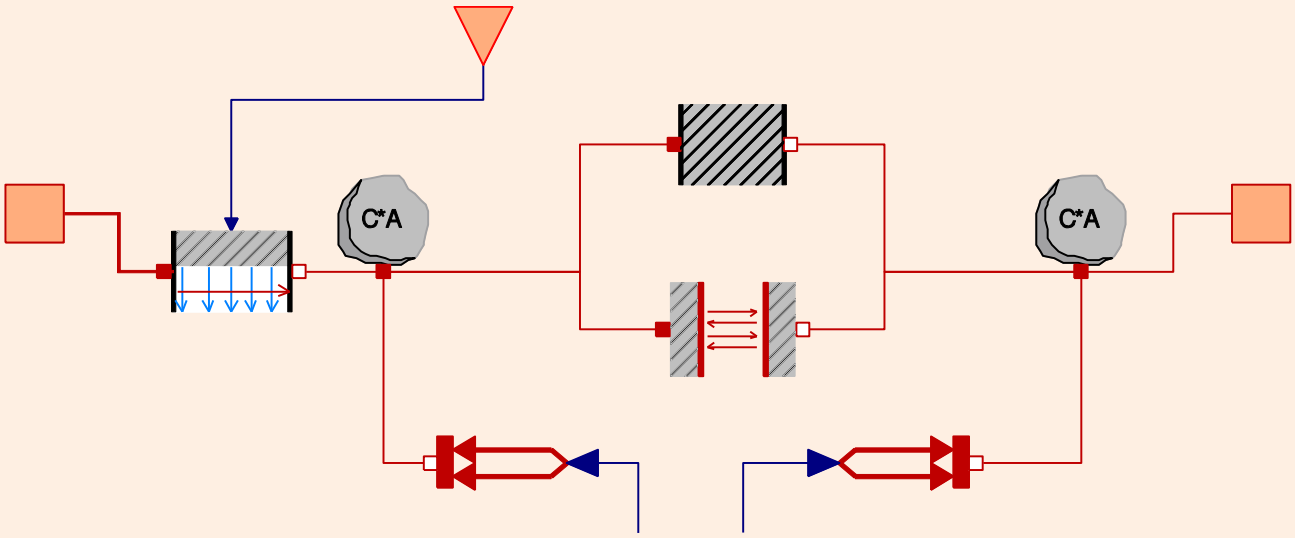
Fakulteta za elektrotehniko

Laboratorij za modeliranje, simulacijo in vodenje

Laboratorij za avtonomne mobilne sisteme

MODELICA

Borut Zupančič



Kazalo

1.

Modelica - večdomensko objektno-orientirano modeliranje

1.1 Večdomensko, fizikalno in objektno-orientirano modeliranje

Do sedaj smo se ukvarjali s konvencionalnimi splošno namenskimi bločno orientiranimi orodji. Glavna slabost pa je, da imajo zelo pomanjkljive OO značilnosti, kar pomeni, da z njimi ne moremo graditi knjižnice realnih komponent, ki bi bile nato ponovno uporabljive v raznih topoloških konfiguracijah. Prav taka gradnja pa je potrebna v sodobnem modeliranju, ko moramo graditi kompleksne modele, ki vsebujejo različna področja (npr. mehanika, elektrotehnika, procesna tehnologija, vodenje sistemov, informacijski sistemi, ...). Zato se je uveljavilo zelo aktualno poimenovanje - **večdomensko modeliranje**. Namreč zlasti v gradnji visoko tehnoloških izdelkov z veliko dodano vrednostjo je potrebno združevati mehanske, električne in procesne komponente ter sisteme vodenja. Tipična področja so mehatronika, avtomobilska industrija, letalska in vesoljska industrija, robotika, procesna industrija.

Prav tako v zadnjem času veliko slišimo o **fizikalnem modeliranju**. Izraz sicer nekoliko zavaja, saj je vsako modeliranje na nek način fizikalno. V tem primeru imamo v mislih računalniško podprto modeliranje, kjer uporabnikom zlasti na višjih hierarhičnih nivojih modela ni potrebno uporabljati diferencialnih enačb, prenosnih funkcij ali zapisa v prostoru stanj, ampak uporabljajo bolj praktične,

po možnosti grafično predstavljene gradnike, ki se nahajajo v knjižnicah. Taki modeli nazorno predstavljajo fiziko in topologijo realnega sistema in so zato zlasti primerni za obravnavo v interdisciplinarnih skupinah. Tak pristop daje možnost modeliranja tudi tistim, ki sicer niso eksperti iz področja modeliranja.

Moderne koncepte modeliranja je možno izvesti le z **objektno-orientiranimi** programskimi orodji. Objektna orientiranost se je rodila prav na področju simulacije, se uveljavila pri splošnonamenskih programskih jezikih in se v zadnjih dveh desetletjih vrnila na področja modeliranja in simulacij.

1.2 Značilnosti in problematika orodij za modeliranje in simulacijo

Pri snovanju sodobnejših orodij za modeliranje in simulacijo se srečamo s problemom standardizacije, saj je zadnji uspešni standard znan iz leta 1967 (CSSL'67, Strauss, 1967). Po tem se razvijalci kljub številnim poskusom niso več uspeli dogovoriti za nov standard. Problem bolj konvencionalno zasnovanih sodobnih orodij je tudi pomanjkanje objektne orientiranosti. Obstajajo sicer zelo učinkovita in uporabniško prijazna orodja za posamezna področja, npr.

- SPICE za modeliranje električnih in elektronskih sistemov,
- ASPEN Plus, SpeedUp za kemične procese,
- SIMPACK za prostorsko modeliranje teles v mehaniki, angl. Multy Body Systems.

Zgornja orodja so izjemno močna in učinkovita na enem področju, ne omogočajo pa učinkovitega vključevanja komponent iz drugih področij. Seveda pa obstaja veliko splošno namenskih simulacijskih orodij (npr. Simulink, ACSL), ki temeljijo na konvencionalnem vhodno-izhodnem zapisu komponent, blokov. Koncept izhaja iz konvencionalne analogne simulacije in iz standarda CSSL'67. To je zelo univerzalen a tudi zelo nizkonivojski pristop.

Zato je nastala potreba po učinkovitejših modelersko-simulacijskih pristopih, ki bi omogočili bolj 'fizikalni' način modeliranja:

- Bond grafi (povezovalni grafi) so grafično podprta modelerska tehnika. Zelo ilustrativno kažejo, kako se energija pretaka med komponentami (20-sim, knjižnica v jeziku Modelica, ...).
- OO pristopi: Dymola z jezikom Dymola in kasneje z jezikom Modelica, OpenModelica, MathModelica, Simscape (od l. 2008).

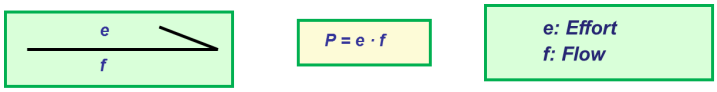
V nadaljevanju bomo spoznali predvsem sodobne OO pristope v povezavi z jezikom Modelica. Tehniko z Bond grafi bomo le bežno omenili.

Lahko zaključimo, da je jezik Modelica eden pomembnejših korakov v standardizaciji orodij za modeliranje in simulacijo.

1.3 Bond grafi

Bond grafe (Paynter, 1961, Borutsky, 2010) je v šestdesetih letih vpeljal prof. H.M. Paynter iz MIT. To je grafična modelerska tehnika, ki lepo ilustrira prehod energijskih tokov skozi sistem.

Energijski tok je predstavljen z usmerjenimi povezavami (slika 1.1). Dve spremenljivki s produktom določata pretok energije: e -potencialna (angl. effort, potential) in f - pretočna spremenljivka (angl. flow).



Slika 1.1: Bond - usmerjena povezava

Osnovne značilnosti pri modeliranju z Bond grafi podajajo naslednje alineje:

- Za omenjeno tehniko je veliko navdušencev, obstoja veliko knjig, konferenc. Navdušenci omenjajo številne situacije, ki jih je možno učinkovito rešiti s pomočjo Bond grafov.
- Verjetno so Bond grafi najuspešnejša metoda za poenostavljanje kompleksnejših modelov na osnovi majhnih energijskih tokov.

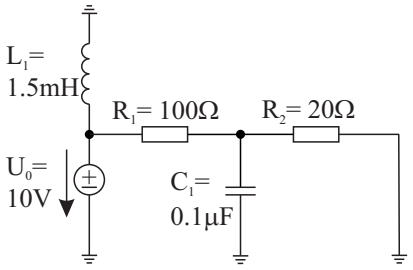
- Vzročni Bond grafi (angl. Causal Bond Graphs) so sistematični in učinkovit način za določitev vzročnosti v enačbah, kar je potrebno zagotoviti za učinkovito simulacijo.

- Čeprav so Bond grafi nastali že v šestdesetih letih prejšnjega stoletja, pa se niso zelo razširili. Zlasti je težko za to tehniko navdušiti ljudi, ki nimajo poglobljenega znanja o modeliranju.

Primer: Uporaba Bond grafov pri modeliranju električnega sistema

Na enostavnem primeru bomo ilustrirali uporabnost Bond grafov. Bond graf vedno izvira iz virov (SE - effort, SF - flow) in zaključuje v komponentah (upornost R, kapacitivnost C, induktivnost L itd.). Tako lahko nazorno opazujemo poteke energijskih tokov. Energija izvira iz virov in ima ponor v pasivnih komponentah električnega vezja. Produkt effort in flow spremenljivke v vsakem bondu je energija, ki se skozi povezavo prenaša. Razen tega ima Bond graf dve različni obliki spojev: spoj nič, ki pomeni, da so tam vse effort spremenljivke enake in je vsota flow spremenljivk enaka nič, in spoj 1 z enakimi effort spremenljivkami in vsoto flow spremenljivk enako nič.

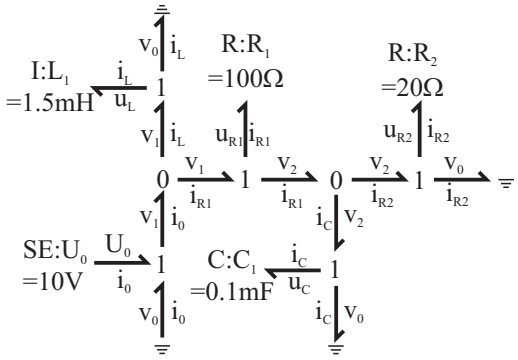
Slika 1.2 prikazuje testno električno shemo. Pomembno je, da na začetku ločimo vse točke z ničelnim potencialom (ozemljitvene točke).



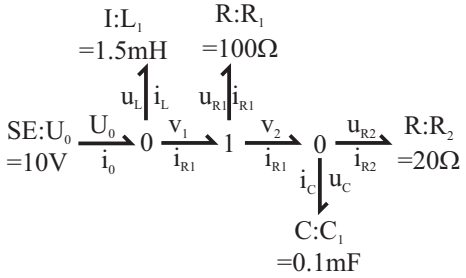
Slika 1.2: Električno vezje

Po enostavnih pravilih narišemo Bond graf - slika 1.3.

Ker so potenciali ozemljitvenih točk enaki nič, potem v te spoje teče energijski tok nič, zato lahko ustrezne bonde izločimo. Nadalje združimo dva zaporedna enako usmerjena bonda v en bond. Tako dobimo poenostavljeno shemo, ki jo prikazuje slika 1.4.

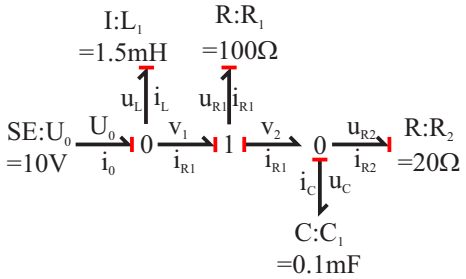


Slika 1.3: Bond graf električnega vezja



Slika 1.4: Poenostavljeni Bond graf električnega vezja

Vzročne bond grafe uporabimo za določitev vzročnosti, oz. za oblikovanje enačb, ki omogočajo simulacijo. Vzročnost pomeni, da iz vsake enačbe izrazimo na levo stran eno neznanko-spremenljivko. Po določenih pravilih narišemo v bond grafu rdeče črtice in nato iz grafa (slika 1.5) napišemo enačbe.



Slika 1.5: Vzročni Bond graf

Osnovna pravila so pri tem naslednja: pri virih je vzročnost fizikalno določena. Pri pasivnem elementu upornost sta obe vzročnosti mogoči (da tok povzroči napetost ali da napetost povzroči tok). Posebno pravilo pa velja za shranje-

valnike energije. Tam je treba na levo stran enačbe izraziti spremenljivko (stanj), ki je integral druge spremenljivke, saj na ta način z odvajanjem te enačbe dobimo izražene odvode spremenljivk stanj, ki so bistveni v konceptu digitalne simulacije.

S pomočjo vzročnih bond grafov napišemo sistem enačb

$$\begin{aligned} U_0 &= 10V \\ i_0 &= i_L + i_{R1} \\ \frac{di_L}{dt} &= \frac{1}{L_1} U_0 \\ U_{R1} &= U_0 - u_C \\ i_{R1} &= \frac{u_{R1}}{R_1} \\ i_C &= i_{R1} - i_{R2} \\ \frac{du_C}{dt} &= \frac{1}{C_1} i_C \\ i_{R2} &= \frac{u_{R2}}{R_2} \end{aligned}$$

1.4 Zgodovina OO modeliranja

Mimic je bilo prvo OO simulacijsko orodje iz leta 1969. Ideja OO je torej nastala na področju digitalne simulacije. Leta 1978 je H. Elmqvist za doktorsko disertacijo razvil okolje Dymola (Elmqvist, 1978), ki je predstavljalo modelirni predprocesor za takrat znane simulacijske in splošnonamenske programske jezike Simula, Pascal, Desire, ACSL, Fortran, Simulink. Dymola z jezikom Dymola je bil prvi na enačbah zasnovani OO modelirni jezik. Napredno OO modeliranje je opisal tudi F. Cellier v knjigi Continuous System Modeling leta 1991 (Cellier, 1991). Leta 1992 je podjetje Dynasim iz Lunda na Švedskem izdalo komercialno različico okolja Dymola. V tem času so se pojavile tudi pobude (EUROSIM Modelica Technical Committee 1 in Technical Chapter on Modelica pri ameriški Society for Computer Simulation) za nov OO jezik, ki so ga poimenovali Modelica. Od leta 1996 je bilo cca. 80 srečanj delovne skupine Modelica. V l. 2000 je nastala odprta neprofitna zveza Modelica Association. Le-ta razen delovnih sestankov razvojne skupine organizira tudi velike Modelica konference. Modelica se torej nenehno razvija, trenutno je na trgu verzija 3.2 (Modelica, 2010). L. 2006 je Dessault Systems je kupil Dynasim in od takrat trži tudi Dymolo. Trenutno je aktualna verzija Dymola 2012 z Modelica 3.2 (Dymola, 2011).

1.5 Primerjava bločnega in OO načina modeliranja

Splošnonamenska konvencionalno zasnovana simulacijska okolja temeljijo na bločnem, vhodno-izhodnem ali vzročnem (kavzalnem) modeliranju (angl. causal modelling), ki izhaja iz analogne simulacije in standarda CSSL'67 (npr. ACSL, Simulink). Vhodno-izhodni oz. vzročni zapis je zelo blizu zapisu v prostoru stanj

$$\begin{aligned} \dot{x} &= f(x,u,t) \\ y &= g(x,u,t) \end{aligned} \tag{1.1}$$

x so stanja, u vhodi in y izhodi sistema. Če narišemo simulacijsko (bločno) shemo po osnovnih simulacijskih metodah, ima orodje zelo malo dela, da iz sheme napiše enačbe v prostoru stanj, ki nato omogočajo enostavno simulacijo.

Matematiki govorijo o ODE obliki (ordinary differential equations – navadne diferencialne enačbe). Za prevedbo modela iz matematičnega zapisa v ODE obliko oz. v simulacijsko shemo pa je potrebno precej modelerskega znanja, potrebno je vložiti tudi precej dela, kar seveda tudi zelo poveča možnost napak.

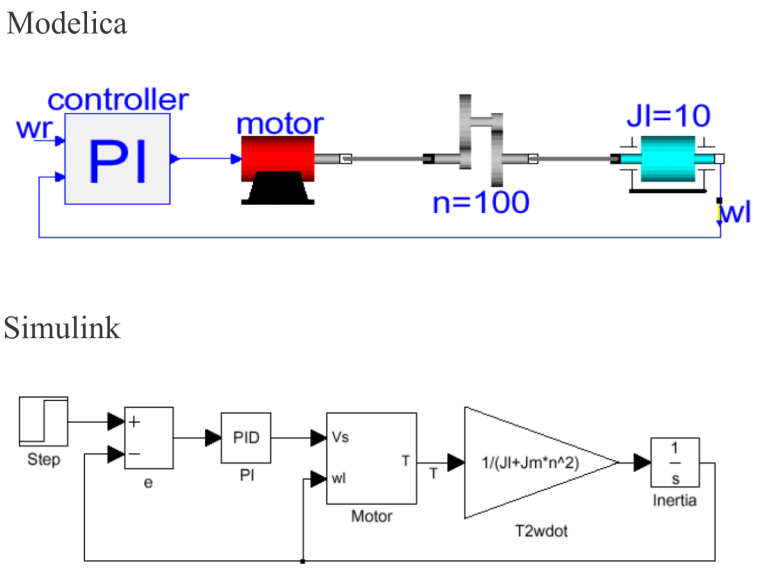
OO modeliranje (npr. z jezikom Modelica) pa omogoča nevzročno modeliranje (angl. acausal modelling). Ravnotežne enačbe so lahko navedene v poljubni obliki, torej ni potrebno predvideti, katera spremenljivka se mora izračunati iz posamezne enačbe. Tak pristop omogoča gradnjo ponovno uporabljivih komponent oz. komponent, ki se lahko uporabijo v različnih konfiguracijah. Matematično govorimo o sistemu DAE enačb (diferencialno algebrajske enačbe, angl. differential algebraic equations)

$$0 = f(\dot{x},x,y,u,t) \tag{1.2}$$

Modelersko okolje (npr. Dymola) s pomočjo simboličnega izračunavanja zgradi učinkovit simulacijski program, kot če bi ročno pretvorili zapis 1.2 v ODE obliko (1.1).

Razliko med kavzalnim in nekavzalnim modeliranjem prikazuje slika 1.6 za problem regulacije hitrosti rotacijskega sistema s pomočjo PI regulatorja in enosmernega motorja. Zgoraj je prikazan model v okolju Dymola-Modelica, spodaj pa v Maltlab-Simulinku. Opazimo, da zgornji model lepo prikazuje fizikalno strukturo realnega sistema, saj so razvidne vse osnovne komponente: regulator, motor, menjalnik oz. prestavno razmerje in vztajnik. V Simulinku pa je identičen

le regulacijski del, ostali gradniki pa niso razvidni. Vztajnost motorja, bremena in prestavno razmerje so združeni v ojačevalni faktor $\frac{1}{J_l} + J_m n^2$.



Slika 1.6: Modela regulacijskega sistema v Modelici in Simulinku

Še enostavnejši dokaz dobimo pri modeliranju električnih tokokrogov. V modelu v okolju Simulink se ohmske upornosti včasih prikažejo kot ojačenje R, včasih pa kot ojačenja $1/R$. Torej potrebujemo za tako enostavno komponento kot je upor dva različna gradnika.

Zaključki, ki izhajajo iz primera, so naslednji:

- V okoljih, ki podpirajo vzročno (kavzalno) modeliranje in kjer vhodni signali povzročijo izhodne signale ni možno zgraditi ponovno uporabljive komponente (npr. motor, upor, menjalnik, breme, ...).

- V naravi so realni sistemi nevzročni (akavzalni). Nikoli namreč ne vemo, ali v uporu napetost povzroči tok ali tok povzroči napetost.

- Vzročnost (kavzalnost) je potrebna, ker moramo fizikalne zakone pretvoriti v računsko obliko, v program, ki omogoča na koncu tudi učinkovito simulacijo.

1.6 Značilnosti sodobnih akavzalnih večdomenskih okolij za modeliranje

Najpomembnejše lastnosti sodobnih večdomenskih okolij za modeliranje so naslednje:

- objektno orientirani pristop,
- enačbni pristop,
- kompleksno povezovanje,
- hibridno modeliranje,
- simbolično procesiranje.

1.6.1 Objektno orientirani pristop

V povezavi z modeliranjem se ne bomo spuščali v podrobnosti OO programiranja, ki uvaja pojme, kot so enkapsulacija, abstrakcija podatkov, dinamično povezovanje in še zlasti dedovanje, t.j. ustanavljanje novih modelnih razredov oz. njihovih primerkov (objektov) z uporabo že pripravljenih modelnih razredov. Ti novi modelni razredi podedujejo atribute oz. vedenje obstoječih modelnih razredov. Torej lahko uporabljamo že razvito kodo z majhnimi popravki.

S stališča modelerja objektni pristop zlasti pomeni, da gradi model podobno kot realni sistem, s pomočjo komponent. Komponente modela, ki se nahajajo v knjižnicah, se hierarhično povezujejo v celotni model. Vse komponente so ponovno uporabljive. Osnovna komponenta OO M&S orodij je modelni razred, ki je opisan s fizikalnimi zakoni.

Primer v Modelici:

```
Resistor R1(R=100)
```

`Resistor` - modelni razred
`R1`- primerek modelnega razreda

1.6.2 Enačbni pristop

V programskih jezikih pravzaprav nimamo enačbe v pravem pomenu besede ampak prireditev - spremenljivki na levi strani enačaja priredimo vrednost, ki se izračuna s pomočjo izraza na desni strani enačaja, npr. $x = 3z + y$.

V nekavzalnih modelirnih okoljih pa uporabljamo prave enačbe. Izrazi so na obeh straneh enačb, npr. $x + y = 3z$. Torej ne gre za prireditev ampak za pravo funkcijo enačaja oz. enačbe. Enačbe nimajo vnaprej določene kavzalnosti. Modelerski (pred)procesor simbolično obdela enačbe, iz vsake enačbe izrazi eno spremenljivko (neznanko) na levo stran (določi torej vhode in izhode) in enačbe tudi medsebojno razvrsti.

1.6.3 Kompleksni način povezovanja komponent preko priključkov - konektorjev

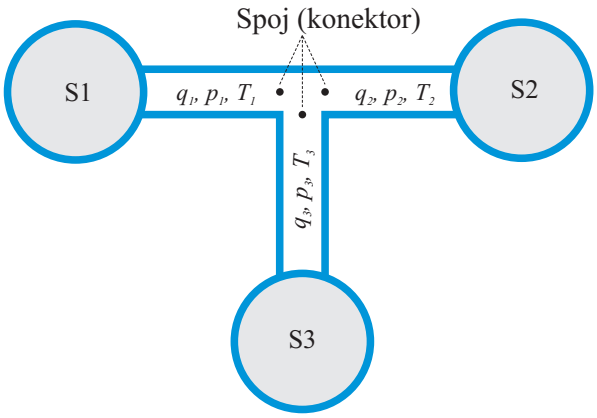
Izvedba povezovanja komponent preko priključkov je najpomembnejša značilnost večdomenskih OO okolij in osnova za delovanje celotnega koncepta. Povezovanje modelnih komponent (podmodelov) temelji na povezavi spremenljivk podmodelov, ki opisujejo medsebojna razmerja in vplive med njimi (npr. premik, kot, tok, tlak,...). Priključek (connector) je posebno določena struktura, v kateri zberemo vse spremenljivke, ki so pomembne pri povezovanju. S spajanjem priključkov povezujemo podmodele med seboj. Seveda lahko povezujemo le kompatibilne priključke (npr. priključek črpalke s priključkom rezervoarja). V okolju Modelica je ena od možnosti povezovanja z ukazom connect. Primer prikazuje povezavo iztoka črpalke z dotokom rezervoarja:

```
connect (crpalka.iztok, rezervoar.dotok)
```

Slika 1.7 prikazuje povezavo treh hidravličnih komponent preko ustreznih priključkov, konektorjev. V vsakem priključku nastopajo tri spremenljivke: pretok tekočine q_i, tlak p_i in temperatura T_i. Analiza pokaže, da imamo dva tipa spremenljivk:

- potencialne spremenljivke (privzete v Modelici): npr. potencial, temperatura tlak. Za te spremenljivke je značilno, da se s povezavo priključkov izenačijo

$$p_1 \;=\; p_2 \;=\; p_3 \tag{1.3}$$



Slika 1.7: Povezava treh hidravličnih komponent

$$T_1 = T_2 = T_3$$

- pretočne spremenljivke (v Modelici označene z besedo flow), npr. električni tok, moment, sila, ... S povezavo priključkov je vsota teh spremenljivk enaka nič

$$q_1 + q_2 + q_3 = 0 \tag{1.4}$$

Primer prikazuje definicijo priključka z imenom `iztok` in njegovih spremenljivk (pretok, tlak in temperatura) v Modelici.

```
connector iztok
   flow Real q;   // pretočni tip
   Real p;        // potencialni tip
   Real T;        // potencialni tip
end iztok;
```

Med procesiranjem modela se s pomočjo ustreznih definicij priključkov enačbe 1.3 in 1.4 avtomatsko generirajo.

Seveda se tudi pri konvencionalnih bločnih sistemih (npr. Simulink) komponente povezujejo. Vendar v vsakem priključku nastopa le ena potencialna spremenljivka. S povezavo komponent se te spremenljivke izenačijo (vhod bloka postane enak izhodu drugega bloka).

1.6.4 Hibridno modeliranje

S sodobnimi OO večdomenskimi okolji (npr. Modelica) se v jeziku nahajajo gradniki, ki omogočajo vključitev nezveznosti in diskretnih dogodkov. Najpogosteje srečamo:

- spremenljivke tipa Boolean,
- z relacijami prožene dogodke (npr. z if stavki),
- periodične dogodke v zvezno-diskretnih sistemih (npr. v diskretnih regulacijskih sistemih, uporaba when stavka),
- sisteme s spremenljivo strukturo.

1.6.5 Simbolično procesiranje

V prevajanju izvornega hierarhičnega modela se uporablja numerično in simbolično računanje:

- sploščenje (angl. flattening),
- izražava ene neznanke iz ene enačbe (vodoravno razvrščanje),
- razvrščanje enačb (vertikalno razvrščanje),
- zmanjšanje dimenzij in kompleksnosti (izločitev trivialnih enačb v1=v2, simbolična izločitev algebrajskih zank, zmanjšanje števila neznank s postopkom 'tearing', ...),
- zmanjšanje indeksa. Višji indeks v sistemih, ki so zapisani z DAE pomeni, da je potrebno uporabiti diferenciacijo za izračun odvodov. Višji indeks tipično dobimo zaradi omejitev med modeli.

1.7 Jezik Modelica

Modelica je jezik modeliranje fizikalnih sistemov z naslednjimi lastnostmi:

- OO modeliranje,
- hierarhično modeliranje,
- akavzalno modeliranje,
- večdomensko modeliranje,
- hibridno modeliranje,
- je tipiziran deklarativni tekstualni jezik.

Modelica torej ni orodje oz. modelersko okolje. Modelica je odprt modelerski jezik s prosto dostopnimi specifikacijami standarda. Standard se zlasti na rednih sestankih neprestano dopolnjuje. Trenutno je aktualna verzija 3.2 iz leta 2012.

Obstajajo prosto dostopna in komercialna okolja z Modelico:

- Dymola - podjetje Dassault systems (Francija) in Dynasim (Lund, Švedska) - prvo in verjetno nabolj izpopolnjeno okolje,
- OpenModelica - razvija Univerza v Linköpingu (Švedska) v sklopu fundacije Open Source Modelica Consortium - OSMC),
- MathModelica - podjetje MathCore iz Linköpinga, Švedska,
- SimulationX - podjetje ITI iz Drsdena, Nemčija,
- MapleSim - podjetje MapleSoft iz Kanade.

Standard je zasnovan tako, da modele opišemo pretežno na dva načina ali s kombinacijo obeh načinov:

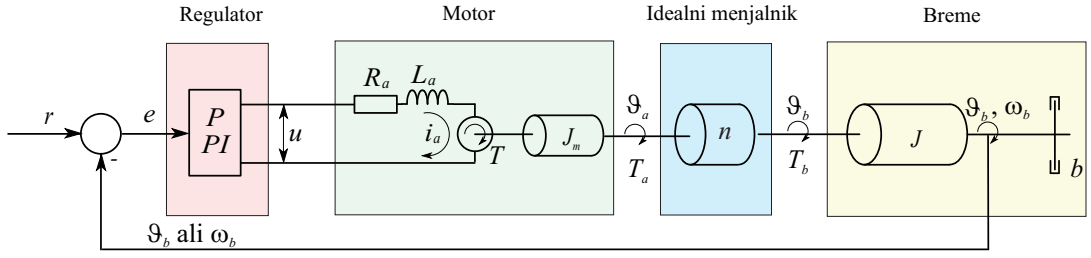
- Modeliranje na višjih hierarhičnih nivojih z vnaprej pripravljenimi knjižničnimi gradniki (npr. gradniki iz standardne Modelica knjižnice) z grafičnim povezovanjem ikon.
- Na nižjih hierarhičnih nivojih so podrobnosti opisane predvsem s pomočjo fizikalnih zakonov oz. ravnotežnih enačb, kar najenostavneje vnašamo v tekstovnem načinu z enačbami v povsem nevtralni obliki - ni potrebno iz enačb izraziti spremenljivk).

- Oba načina se lahko prepletata in dopolnjujeta. Tudi če gradimo modele na grafični način, se avtomatsko generirajo pregledni tekstovni programi.
- V primeru diskretnih in hibridnih modelov dodajamo mehanizme za proženje dogodkov.

Tako kot v OO splošnonamenskih programskih jezikih je tudi v Modelici osnovni gradnik modelni razred. Pomembnejši modelni razredi so naslednji:

- `class` (najbolj splošni razred brez omejitev),
- `partial class` (razred se ne more uporabljati samostojno ampak le za gradnjo drugih modelnih razredov),
- `model` (npr. za vrhnji nivo hierarhično zgrajenega modela),
- `block` (za gradnike, ki so kavzalnega tipa, torej z vnaprej znanimi vhodi in izhodi - značilni bloki konvencionalnih simulacijskih orodij),
- `function` (za funkcije kot v splošnonamenskih programskih jezikih),
- `connector` (modelni razred za opis priključkov - konektorjev),
- `package` (za vnaprej programirane kompleksnejše gradnike - knjižnice),
- `type` (za razširitev predefiniranih tipov spremenljivk).

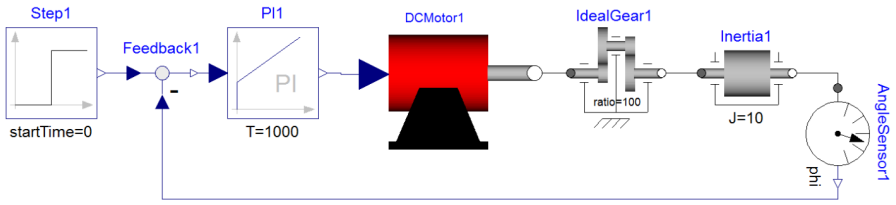
Za začetek si poglejmo možnost modeliranja s pomočjo gradnikov, ki so že pripravljeni. Zgradimo regulacijski sistem za regulacijo zasuka rotacijskega mehanskega sistema (slika 1.8) (`DCDrive`).



Slika 1.8: Regulacija zasuka ali hitrosti rotacijskega sistema - bločna shema

Uporabimo PI regulator (`PI1`), enosmerni motor (`DCMotor1`), idealno prestavno razmerje (`IdealGear1`), rotacijsko breme (`Inertia1`), merilnik zasuka (`AngleSensor1`),

generator želenega zasuka (`Step1`) in blok za izračun regulacijskega pogreška (`Feedback1`). Model zgradimo najprej z grafičnim urejevalnikom v okolju Dymola in ga prikazuje slika 1.9. Vsak uporabljeni gradnik je primerek modelnega



Slika 1.9: Regulacija zasuka rotacijskega sistema - model v Dymola-Modelica

razreda. Privzeto se primerek poimenuje kar z imenom modelnega razreda. Če iz enega modelnega razreda uporabimo več primerkov, se na koncu imena doda številka 1, 2, ... Seveda pa se lahko imena primerkov poljubno oblikujejo. V našem primeru smo mi vse poimenovali z imeni ustreznih modelnih razredov in številke 1 na koncu, ker v vseh primerih nastopa le en primerek.

Iz grafično zgrajenega modela naredi prevajalnik tudi tekstovni model, lahko pa modeliramo kar s tekstovnim editorjem in napišemo model. Za to potrebujemo več znanja in tudi priročnik, kjer lahko spoznamo osnovno sintakso gradnika: ime, parametri, oblike in imena priključkov.

```
Model DCdrive

   Step Step1
   Feedback Feedback1
   PI PI1 (T=1000, k=70)
   DCMotor DCMotor1
   IdealGear IdealGear1(ratio=100)
   Inertia Inertia1(J=10)
   AngleSensor AngleSensor1

equation

   connect(Step1.y, Feedback1.u1)
   connect(AngleSensor1.phi, Feedback1.u2)
   connect(Feedback1.y, PI1.u
   connect(PI1.y, DCMotor1.v1)
   connect(DCMotor1.flange_b1, IdealGear1.flange_a
```

```
  connect(IdealGear1.flange_b, Inertia1.flange_a)
  connect(Inertia1.flange_b, AngleSensor1.flange)

end DCdrive;
```

Vsak tekstovni model v jeziku Modelica ima dve sekciji: deklarativni del v našem primeru navedemo vse uporabljene komponente in njihove parametre (če ne navedemo parametrov, se uporabijo prevzete vrednosti) in enačbni del, ki v našem primeru sestoji iz 7 connect stavkov, ki opisujejo povezave izbranih gradnikov.

Primer:

```
Inertia Inertia1(J=10)
```

Uporabimo rotacijsko breme oz. vztajnik - iz modelnega razreda `Inertia` naredimo primerek `Inertia1` in v oklepaju navedemo parameter - t.j. vztrajnost `J=10`. V kolikor ne navedemo parametrov, se uporabijo privzete vrednosti, ki so navedene znotraj vsakega gradnika.

Zaradi večje preglednosti smo pri deklaraciji komponent navedli le ime modelnega razreda, kar pomeni, da bi program deloval, če bi bile vse kompomente zbrane v eni knjižnici. Ker pa je knjižnic več in so hierarhično urejene, moramo navesti celotno pot do te knjižnične komponente. Tako namesto vrstice

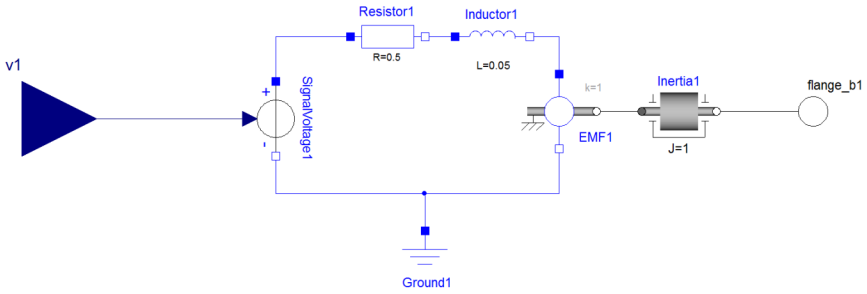
```
Inertia Inertia1(J=10)
```

navedemo vrstico

```
Modelica.Mechanics.Rotational.Components Inertia Inertia1(J=10)
```

Pomeni, da se modelni razred `Inertia` nahaja v skupini `Components`, ki je podskupina `Rotational`, ta pa je podskupina `Mechanics`, ki je knjižnica v standardni knjižnici `Modelica`.

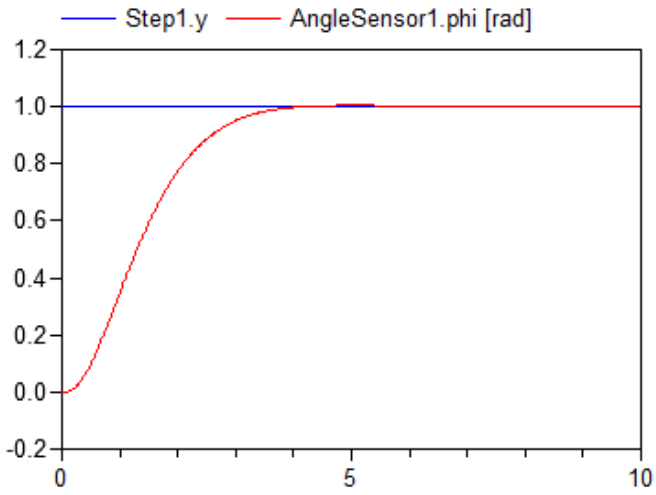
Model je hierarhično zgrajen, znotraj motorja se nahaja model, ki ga prikazuje slika 1.10. Uporabljen je iz literature znani najbolj poenostavljeni model, ki vsebuje zunanje krmiljeni napetostni vir, upor, induktivnost, elektromehanski transformator in vztrajnik motorja. Podmodel se povezuje v glavni model preko priključkov `v1` (celotna oznaka `DCMotor.v1`) in `flange_b1` (celotna oznaka



Slika 1.10: Podmodel motorja

`DCMotor.flange_b1`). Preko priključka `v1` se prenaša signal regulatorja, ki krmili napetostni vir znotraj modela motorja. Gre torej za zelo enostaven priključek, ki vsebuje le eno spremenljivko (potencialni tip). Na drugi strani pa se mehanski učinek motorja prenaša na rotacijsko mehansko os preko priključka `flange_b1`, ki vsebuje več spremenljivk: pretočno spremenljivko moment, in potencialne spremenljivke zasuk, kotna hitrost in kotni pospešek.

Slika 1.11 prikazuje rezultate simulacije. Referenčni zasuk se v trenutku $t = 0$ spremeni za $1rd$, zasuk rotacijskega sistema pa dobro sledi referenčni vrednosti brez pogreška v ustaljenem stanju.



Slika 1.11: Rezultati simulacije

1.7.1 Osnove tekstovnega modeliranja

Jezik Modelica lahko učinkovito uporabljamo brez vgrajenih knjižnic tako, da enostavno uporabimo enačbe matematičnega (teoretičnega) modela. V vseh priročnikih splošnonamenskega programiranja najprej izpišejo tekst 'HelloWord'. Pri modeliranju seveda ne izpisujemo tekstov, pač pa je tu najenostavnejši dinamični model prvega reda

$$\dot{x} = ax \qquad x(0) = 1 \qquad a = 1 \tag{1.5}$$

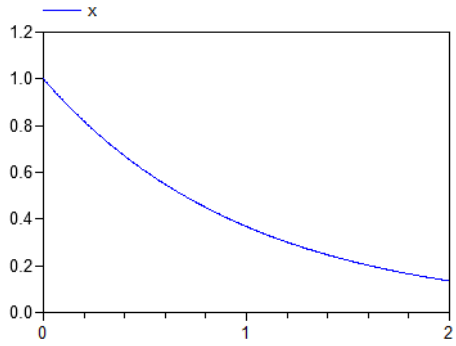
Opis modela v Modelici:

```
model HelloWorld "Sistem 1. reda"
  Real x(start=1);
  parameter Real a = -1;
equation
  der(x)= a*x;
  end HelloWorld;
```

V prvi vrstici vedno navedemo ime modelnega razreda. V tem primeru smo uporabili modelni razred `model` (lahko bi uporabili tudi `class`). Nato sledijo vrstice, v katerih deklariramo vse spremenljivke in parametre modela. `x` deklariramo kot spremenljivko tipa `Real`. Ker predstavlja stanje sistema, lahko v oklepaju s komando start podamo začetno vrednost. Privzete začetne vrednosti vseh spremenljivk pa so nič. `a` je konstanta, ki je nespremenljiva znotraj simulacijskega teka, lahko pa jo spreminjamo med simulacijskimi teki. Deklariramo jo s stavkom `parameter`, pred imenom parametra pa navedemo tip konstante - v našem primeru `Real`. Konstanti priredimo ustrezno vrednost. Sledi sekcija `equation`. `der` je sistemski ukaz, ki označuje odvod. V izrazu der(x)=a*x ima znak = v resnici vlogo enačaja in ne le prireditve. Namesto izraza der(x)=a*x bi lahko vnesli tudi enačbo der(x)+a*x=0. Modelica program zaključimo s stavkom `end`.

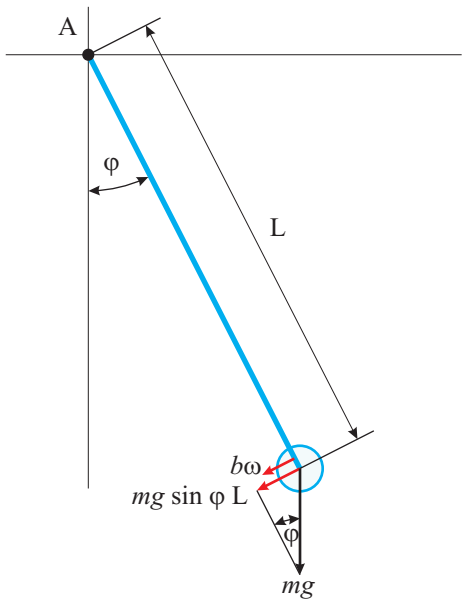
S pomočjo uporabniškega vmesnika v okolju Dymola nastavimo čas opazovanja $2s$, poženemo simulacijo in izrišemo želeno spremenljivko. Lahko pa uporabimo tudi interaktivni eksperimentalni komandi `simulateModel("HelloWorld", stopTime=2)` in `plot({"x"})`. Rezultate prikazuje slika 1.12.

Primer nelinearnega sistema - matematično nihalo



Slika 1.12: Rezulati simulacije sistema 1.reda

Nelinearno matematično nihalo prikazuje slika 1.13. Nihalo lahko modeliramo z



Slika 1.13: Nelinearno matematično nihalo

rotacijskimi enačbami (upoštevamo L, φ, vztrajnostni moment nihala pri rotaciji okoli točke A, ...), lahko pa bi modelirali s pomočjo translatornih enačb, kjer bi uporabili Newtonove zakone v smeri x in y. Newtonov rotacijski zakon pravi, da je vsota vseh momentov na rotirajočo maso enaka vztrajnostnemu momentu pomnoženem z rotacijskim pospeškom. Nastopata pa dva momenta: moment zaradi teže in moment zaradi dušenja. Ker oba momenta delujeta proti ravnovesni legi oz. v obratni smeri od pozitivno označenega kota φ, imata v enačbi negativni

predznak

$$\begin{aligned} \sum T_i &= J\alpha = J\dot{\omega} \\ -mgsin(\varphi)L - b\omega &= J\dot{\omega} \\ J &= mL^2 \end{aligned} \tag{1.6}$$

Program v jeziku Modelica je naslednji:

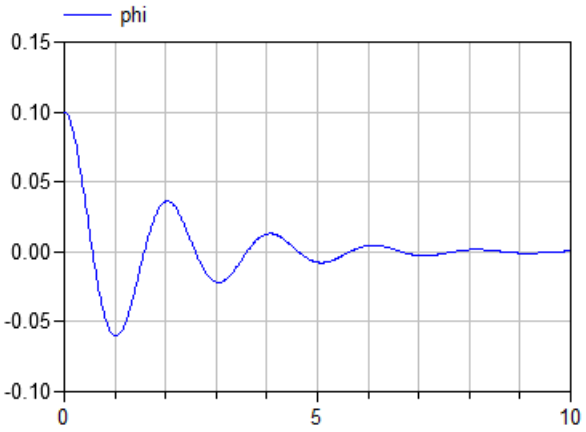
```
model pendulum
  constant Real g=9.81;
  parameter Real m=1;
  parameter Real L=1;
  parameter Real J=m*L^2;
  parameter Real b=1;
  Real phi(start=0.1);
  Real w;
equation
  der(phi) = w;
  -m*g*sin(phi)*L-b*w=J*der(w);
end pendulum;
```

Glede na sistem prvega reda v prejšnjem primeru imamo sedaj dva tipa konstant: s stavkom `parameter` označimo konstante, ki se lahko pri eksperimentiranju spreminjajo (npr. različne vrednosti v različnih simulacijah), s stavkom `constant` pa deklariramo konstanto, ki jo ne nameravamo spreminjati (gravitacijski pospešek). Vidimo tudi, da lahko parameter deklariramo s pomočjo matematičnega izraza, ki vsebuje več deklariranih konstant ($J = ml^2$). Rezultate simulacije prikazuje slika 1.14.

Primer: Modeliranje sistema zapisanega z diferencialno - algebrajskimi enačbami (DAE)

Prejšnja modela sta bila zapisana z diferencialnimi enačbami in bi jih enostavno predstavili v obliki zapisa v prostoru stanj $\dot{x} = f(x,t)$. V bolj zahtevnem modeliranju nastopajo diferencialne in algebrajske enačbe, torej razen spremenljivk stanja nastopajo še druge algebrajske spremenljivke. Okolja Modelica omogočajo, da navedemo enačbe v splošni DAE obliki

$$f(t, \dot{x}(t), x(t), y(t), u(t), p) = 0 \tag{1.7}$$



Slika 1.14: Rezultati simulacije nelinearnega nihala

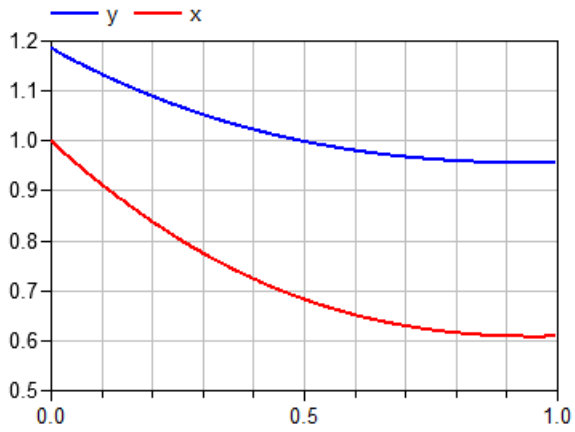
kjer je t neodvisna spremenljivka, $\dot{x}(t)$ vektor odvodov stanj, $x(t)$ vektor stanj, $y(t)$ vektor algebrajskih spremenljivk, $u(t)$ vektor vhodov in p vektor parametrov in/ali konstant. Model

$$\begin{aligned} \dot{x} + sin(y) &= sin(t) \\ y - x &= e^{-0.9y} cos(x) \\ x(0) &= 1 \end{aligned} \tag{1.8}$$

zato lahko napišemo v Modelici v nespremenjeni obliki:

```
model DAEexample
Real y;
Real x(start=1,fixed=true);
equation
der(x)+sin(y)=sin(time);
y-x=exp(-0.9*y)*cos(x);
end DAEexample;
```

$x(t)$je spremenljivka stanj, $y(t)$ je algebrajska spremenljivka, stavek `Real x(start=1,fixed=true)` definira 'vsiljeno' začetno stanje (sicer lahko v določenih primerih začetno stanje v postopku inicializacije spremeni vrednost), primer pa prikazuje tudi tri vgrajene funkcije (`sin`, `cos`, `exp`) in sistemsko neodvisno spremenljivko za čas - `time`. Rezultate simulacije prikazuje slika 1.15.



Slika 1.15: Rezultati simulacije modela DAE

1.7.2 Podrobnejši opis

Konstante, parametri, spremenljivke

V prejšnjih primerih smo spoznali deklaracijo konstant, parametrov in spremenljivk realnega tipa (`Real`) npr.

```
constant Real g=9.82;
parameter Real m=1;
Real phi (start=1);
```

To so najobičajnejše sestavine pri zveznih dinamičnih modelih. Za zahtevnejše modeliranje pa so lahko konstante, parametri in spremenljivke tudi drugačnih tipov, podobno kot v programskih jezikih:

`Boolean`	logični tip z vrednostjo true ali false
`Integer`	celoštevilčni tip (1,2,3,...)
`String`	tekstovni tip
`discrete Real`	diskretni tip

Tipi sestavin imajo določene prednastavljene lastnosti. Začetne vrednosti so običajno nič, če njihova začetna vrednost ni drugače postavljena z ukazom `start`. Začetne vrednosti logičnih spremenljivk so `false`, začetne vrednosti tekstovnih spremenljivk pa so prazni nizi. Prednastavljene lastnosti se postavijo z ukazom `type` v Modelica knjižnici.

Komentarji

Komentarji se vnašajo na tri načine:

"tekst"

Tak komentar se pogosto nahaja na koncu deklarativnih stavkov in ga prevajalnik ustrezno obdela v smislu preglednosti modela ali specifičnega eksperimentiranja. Na ta način je zapisana tudi grafična informacija, če smo model zgradili na grafični način - "annotations"

//komentar
/*komentar*/

To sta prava komentarja izvorne kode. Prevajalnik ignorira komentarje.

Fizikalne veličine

Kvaliteto modela v Modelici zelo izboljšamo, če namesto standardnih konstant, parametrov in spremenljivk uporabimo predefinirane fizikalne veličine. Model postane bolj pregleden, hkrati pa dobimo dodatne zmožnosti: uporabo enot (in preverjanja konsistentnosti), definiranje minimalnih in maksimalnih vrednosti, lahko uporabimo ene enote za izračunavanja, druge za prikaz rezultatov (npr. rd in ^{o}). Pri rezultatih simulacije se jasno navedejo tudi enote posameznih konstant, parametrov in spremenljivk.

Za definicijo fizikalne veličine se uporabi ukaz `type`. Veličina vsebuje več atributov: ime veličine, kvantiteta (ime fizikalne spremenljivke, običajno isto kot ime veličine), enota v skladu z ISO standardom, včasih enota za prikazovanje (če je različna od enote za izračunavanje), minimalna in maksimalna vrednost, ...

Primer:

```
type Angle = Real(quantity = "Angle", unit = "rad", displayUnit = "deg");
type Torque = Real(quantity = "Torque", unit = "N.m");
```

V primeru z ukazom type definiramo dve fizikalni veličini, ki nastopata v rotacijskih mehanskih sistemih, kot `Angle` in moment `Torque`. Definirali smo ime,

enoto in prikazno enoto.

Že razviti primer matematičnega nihala je z upoštevanjem lastno predefiniranih fizikalnih veličin naslednji:

```
model pendulum2
  type angle=Real(unit="rad", displayUnit="deg");
  type velocity=Real(unit="rad/s");
  type gravity=Real(unit="m/s2");
  type mass=Real(unit="kg");
  type length=Real(unit="m");
  type inertia=Real(unit="kg.m2/rad");
  type dumping=Real(unit="N.m.s/rad");
    constant gravity g=9.81;
    parameter mass m=1;
    parameter length L=1;
    parameter inertia J=m*L^2;
    parameter dumping b=1;
    angle phi( start=.1);
    velocity w;
  equation
    der(phi) = w;
    -m*g*sin(phi)*L-b*w=J*der(w);
end pendulum2;
```

Definirali smo torej 5 fizikalnih veličin z ustreznimi enotami, katerih konsistentnost se med prevajanjem preverja.

Standardna knjižnica Modelica vsebuje okoli 450 predefiniranih fizikalnih veličin v knjižnici SIunits (oz. pot Modelica.SIunits) in temelji na ISO mednarodnem standardu enot. (ISO 31-1992 "General principles concerning quantities, units and symbols", ISO 1000-1992 " SI units and recommendations for the use of their multiples and of certain other units".

Že razviti primer matematičnega nihala je z upoštevanjem v knjižnici SIunits predefiniranih fizikalnih veličin naslednji:

```
model pendulum1
  constant Modelica.SIunits.Acceleration g=9.81 "Gravity acceleration";
```

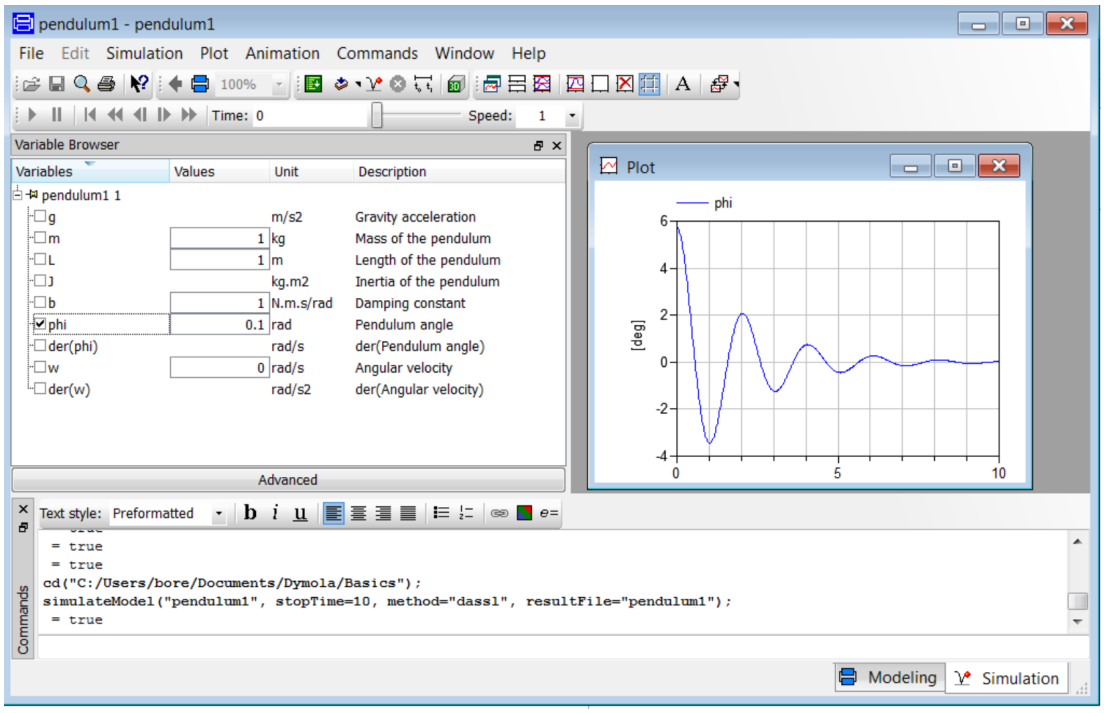
```
  parameter Modelica.SIunits.Mass m=1 "Mass of the pendulum";
  parameter Modelica.SIunits.Length L=1 "Length of the pendulum";
  parameter Modelica.SIunits.MomentOfInertia J=m*L^2 "Inertia of the pendulum";
  parameter Modelica.SIunits.RotationalDampingConstant b=1 "Damping constant";
  Modelica.SIunits.Angle phi(start=0.1) "Pendulum angle";
  Modelica.SIunits.AngularVelocity w "Angular velocity";
equation
  der(phi) = w;
  -m*g*sin(phi)*L-b*w=J*der(w);
end pendulum;
```

Na prvi pogled se zdi, da je postopek kar zahteven, saj lahko naredimo tudi precej tipkarskih napak. Vendar okolje Dymola omogoča, da veličino enostavno najdemo v knjižnici `Modelica.SIunit` (če iščemo `angle`, pritiskamo a in prikazujejo se vse veličine s prvo črko a) nato pa preko uporabniškega vmesnika definiramo atribute veličine. Lahko pa v tekstovno okno modela prenesemo le ime, ostalo pa ročno dodamo. Model vsebuje tudi komentarje na koncu deklaracijskih stavkov. To niso običajni komentarji, ampak jih obdela tudi prevajalnik, tako da se ti teksti pojavijo tudi pri eksperimentiranju z modelom.

Slika 1.16 prikazuje izgled zaslona pri eksperimentiranju v okolju Dymola z modelom nihala. Opazimo tudi enote in komentarje pri posameznih veličinah. Izrisan je potek spremenljivke phi v ^{o}, ker je v fizikalni veličini angle definirana prikazna enota ^{o}. Računska enota pa je *rd*. Preverjanje konsistentnost enot je privzeto vključeno, lahko pa se tudi izključi. V primeru nekonsistentnosti se izpiše opozorilo, a se s postopkom eksperimentiranja lahko nadaljuje. Primer prikazuje del programa v Modelici, ki vključuje Newtonov zakon, vendar smo pomotoma namesto pospeška v enačbi uporabili hitrost.

```
model Newton
   parameter Modelica.SIunits.Mass m = 10 "Mass";
   Modelica.SIunits.Force f = 10 "Force";
   Modelica.SIunits.Velocity v "Velocity";
equation
   m*v = f;
end Newton;
```

Pri prevajanju se izpiše naslednje opozorilo:



Slika 1.16: Simulacija nihala

```
Running variable unit checking on this model generates the following log:
Warning: Incompatible units in
equation
m*v = f;
The part
m*v
has unit m.kg.s-1
The part
f
has unit m.kg.s-2
in equation
m*v = f;
```

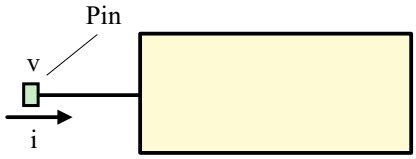
1.7.3 Objektno modeliranje

Do sedaj opisano modeliranje je zelo učinkovito, če imamo model predstavljen z diferencialnimi in algebrajskimi enačbami. Že ta način prinaša veliko prednosti

pred konvencionalnimi pristopi. Kompleksnejše modele pa običajno gradimo na hierarhičen OO način (tudi grafično) s pomočjo predpripravljenih knjižničnih gradnikov. V tem delu si bomo ogledali, kako zgradimo komponento, ki jo lahko uporabimo pri povezovanju v zahtevnejši model.

Priključki in povezovanje

Kompleksen način povezovanja komponent je najpomembnejša značilnost obravnavanih orodij. Opisana je v poglavju 1.6.3. Vsaki komponenti moramo definirati ustrezne priključke (`connector`), v katerih nastopajo pretočne in potencialne spremenljivke. Pri električnih vezjih sta to tok in napetost, pri mehanskih rotacijskih sistemih pa moment, kot, kotna hitrost. Slika 1.17 prikazuje priključek komponente električnega vezja.



Slika 1.17: Priključek (`connector`) v električni komponenti

V Modelici definiramo konektor z naslednjim programom:

```
connector Pin
Voltage v;
flow Current i;
end Pin;
```

Priključek smo poimenovali `Pin`. Upoštevali smo, da sta `Voltage` in `Current` definirani fizikalni veličini. Privzeto so vse spremenljivke potencialnega tipa, zato le spremenljivko `Current` definiramo kot pretočno z oznako `flow`. `v` je potencial priključka in je tipa `Voltage`, `i` pa tok priključka tipa `Current`. Pozitivna smer toka kaže proti komponenti.

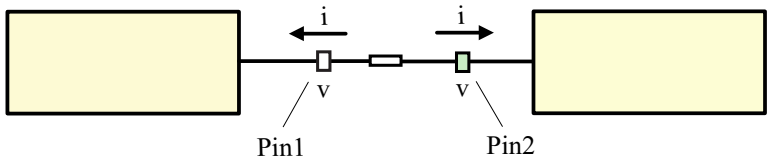
V rotacijskih mehanskih sistemih uporabljamo priključek `Flange` (slov. prirobnica):

```
connector Flange
Angle phi;
flow Torque T;
end Flange;
```

Ime priključka je `Flange`, `Angle` je potencialna, `Torque` pa pretočna spremenljivka.

Vsaka Modelica knjižnica vsebuje definicije potrebnih priključkov, preko katerih povezujemo komponente.

Dve električni komponenti povežemo s povezavo priključkov, kot prikazuje slika 1.18. Pri grafičnem modeliranju priključka `Pin1` in `Pin2` povežemo kar s pomočjo



Slika 1.18: Povezava dveh električnih komponent

miške, pri tekstovnem modeliranju pa s stavkom

```
connect(Pin1,Pin2)
```

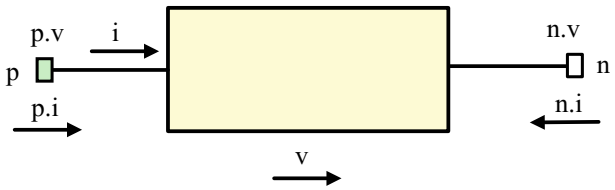
`Pin1` in `Pin2` sta dva priključka oz. primerka modelnega razreda `Pin`. Med procesiranjem modela se imena spremenljivk razširijo tako, da se vključi ime modelnega razreda (npr. `Pin1.v` pomeni napetost v priključku `Pin1`, `Pin2.i` pomeni tok v priključku `Pin2`). Nato se zgradita dve enačbi

```
Pin1.v=Pin2.v
Pin1.i+Pin2.i=0
```

Prva enačba torej izenači oba potenciala (Kirchofov napetostni zakon), druga izenači vsoto obeh tokov z nič (Kirchofov tokovni zakon). Podobne relacije veljajo v hidravličnih, mehanskih in drugih sistemih.

Modelni razredi in dedovanje

Modelne razrede gradimo tako, da podedujejo lastnosti enostavnejših modelnih razredov in dodajo nove lastnosti. Pomembno vlogo ima modelni razred `partial class`, kar pomeni, da se ga lahko uporabi le za gradnjo drugih modelnih razredov, ne pa tudi samostojno. Skupna lastnost večine električnih komponent(slika 1.19) je, da imajo dva priključka, ki sta v sliki označena kot pozitivni `p` in negativni `n`.



Slika 1.19: Modelni razred `twopin`

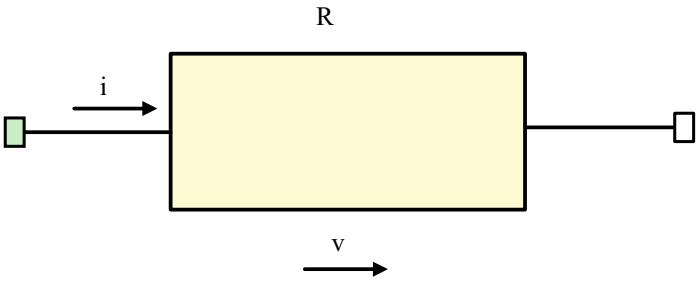
Komponenta `twopin` vsebuje 'pozitivni' priključek `p` in 'negativni' priključek `n`. Dodatno označimo napetost `v` in tok `i`.

Modelni razred `twopin` opišemo z naslednjim programom v jeziku Modelica:

```
partial class TwoPin
 Pin p;
 Pin n;
 Voltage v;
 Current i;
equation
 v = p.v - n.v;
 p.i + n.i = 0;
 i=p.i;
end TwoPin;
```

Enačbe definirajo znane relacije v električnih shemah. Napetost na komponenti je razlika potencialov v priključkih, vsota vseh tokov, ki pritekajo v komponento preko priključkov, je enaka nič. Pozitivna smer toka skozi komponento pa je označena v smeri toka skozi priključek `p`. Beseda `partial` označuje, da je modelni razred nepopoln, a primeren, da s pomočjo dedovanja in dodajanja kvalitativne enačbe definiramo upor, tuljavo, kondenzator, ...

Modelni razred upora `Resistor` (slika 1.20) izdelamo tako, da le ta podeduje vse lasnosti modelnega razreda `TwoPin` (s komando `extends TwoPin`), doda pa se osnovna enačba upora, t.j.ohmov zakon `R*i=v`.



Slika 1.20: Modelni razred upora `Resistor`

```
class Resistor "Ideal resistor"
extends TwoPin;
parameter Resistance R=1 "Resistance";
equation
R*i=v;
end Resistor;
```

Fizikalna veličina upornost - `Resistance` je definirana v standardni knižnici SIunit s komando `type`:

```
type Resistance=Real(quantity="Resistance", unit="ohm", min=0);
```

Definiranih je več atributov: `quantity` - fizikalno ime, običajno enako imenu veličine, `unit` - enota in `min` - minimalna vrednost. Pri deklaraciji velečine `Resistance` v modelnem razredu `Resistor` navedemo ime primerka `R` in vrednost (`R=1`).

Identično zgradimo modelne razrede tuljave in kondenzatorja:

```
class Inductor "Ideal inductor"
extends TwoPin;
parameter Inductance L=1 "Inductance";
```

```
equation
v=L*der(i);
end Inductor;

class Capacitor "Ideal capacitor"
extends TwoPin;
parameter Capacitance C=1 "Capacitance";
equation
i=C*der(v);
end Capacitor;
```

V preprostih električni vezjih potrebujemo npr. še enosmerni napetostni vir. Modelni razred `stepVoltage` je naslednji:

```
class stepVoltage "Ideal voltage source"
extends TwoPin;
parameter Voltage U=1;
equation
  U = v;
end stepVoltage;
```

Prav tako potrebujemo ozemljitveno točko. Ker ima ozemljitvena točka le en priključek, ne uporabimo modelnega razreda `Twopin`.

```
class Ground
Pin p;
equation
p.v = 0;
end Ground;
```

V rotacijskih sistemih imamo tudi komponente z dvema priključkoma. Zato je smiselno zgraditi modelni razred `partial model`, ki vsebuje osnovne lastnosti, z imenom `TwoFlange`

```
partial class TwoFlange
Flange a, b;
end TwoFlange;
```

Deklariramo dva primerka, t.j. dva priključka iz razreda `Flange`, `a` in `b`.

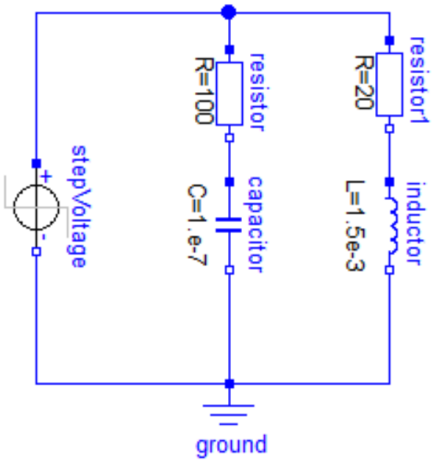
S pomočjo modelnega razreda `TwoFlange` lahko razvijemo več komponent npr. rotacijsko breme (vztrajnostnik) z vztrajnostnim momentom `J`.

```
class Shaft
extends TwoFlange;
parameter Inertia J=1;
AngularVelocity w;
equation
a.phi=b.phi;
der(a.phi)=w;
J*der(w)=a.T+b.T;
end Shaft;
```

`Inertia` in `AngularVelocity` sta deklarirani fizikalni veličini v knjižnici SIunit. Pri enačbah uporabimo osnovne mehanske zakone: zasuk v obeh priključkih je enak (togo telo), velja pa seveda Newtonov zakon: $J\alpha = \sum T_i$.

Primer: Električna shema

Slika 1.21 prikazuje preprosto električno vezje, ki je narisano (modelirano) v okolju Dymola Modelica. Tekstovni program v Modelici s pomočjo zgoraj pripravljenih



Slika 1.21: Električno vezje

gradnikov je spremenjen le pri deklaraciji SI fizikalnih veličin: medtem ko smo v

zgornjih gradnikih izpustili navedbo celotne poti do ustrezne knjižnice, smo tu to dodali (npr. `Voltage` smo zamenjali z `Modelica.SIunits. Voltage`).

```
model circuit
Resistor R1(R=100) ;
Resistor R2(R=20);
Capacitor C(C=1e-7);
Inductor L(L=1.5e-3);
stepVoltage U0(U=1);
Ground Common;
equation
connect(U0.p, R1.p);
connect(U0.p, R2.p);
connect(R1.n, C.p);
connect(R2.n, L.p);
connect(C.n, U0.n);
connect(L.n, U0.n);
connect(U0.n, Common.p);
end circuit;
```

```
model circuit
Resistor R1(R=100);
Resistor R2(R=20);
Capacitor C(C=1e-7);
Inductor L(L=1.5e-3);
stepVoltage U0(U=1);
Ground Common;
equation
connect(U0.p, R1.p);
connect(U0.p, R2.p);
connect(R1.n, C.p);
connect(R2.n, L.p);
connect(C.n, U0.n);
connect(L.n, U0.n);
connect(U0.n, Common.p);
end circuit;

connector Pin
Modelica.SIunits.Voltage v;
flow Modelica.SIunits.Current i;
```

```
end Pin;

partial class TwoPin
 Pin p;
 Pin n;
 Modelica.SIunits.Voltage v;
 Modelica.SIunits.Current i;
equation
 v = p.v - n.v;
 p.i + n.i = 0;
 i=p.i;
end TwoPin;

class Resistor "Ideal resistor"
extends TwoPin;
parameter Modelica.SIunits.Resistance R=1 "Resistance";
equation
R*i=v;
end Resistor;

class Capacitor "Ideal capacitor"
extends TwoPin;
parameter Modelica.SIunits.Capacitance C=1 "Capacitance";
equation
i=C*der(v);
end Capacitor;

class Inductor "Ideal inductor"
extends TwoPin;
parameter Modelica.SIunits.Inductance  L=1 "Inductance";
equation
v=L*der(i);
end Inductor;

class stepVoltage "Ideal voltage source"
extends TwoPin;
parameter Modelica.SIunits.Voltage U=1;
equation
  U = v;
end stepVoltage;
```

```
class Ground
Pin p;
equation
p.v = 0;
end Ground;
```

Predstavili smo torej tekstovni OO model na osnovi modelnih razredov, ki smo jih sami zgradili. Vsi ti modelni razredi so seveda že narejeni v standardni knjižnici Modelica. Če bi želeli uporabiti že narejene gradnike, bi morali v deklaraciji gradnikov navesti ustrezno pot: Namesto

```
Resistor R1(R=100);
```

bi napisali

```
Modelica.Electrical.Analog.Basic.Resistor R1(R=100);
```

Model električnega vezja seveda lahko modeliramo tudi brez OO pristopa z vpisom vseh Kirchofovih enačb. V tem primeru je program v Modelici:

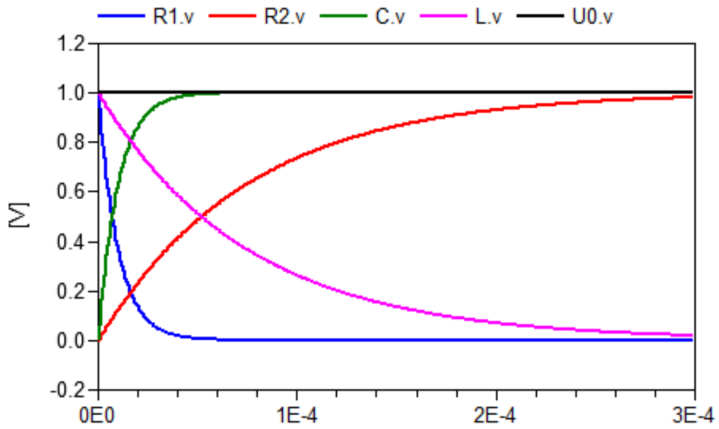
```
model Vezje
  parameter Real R1=100;
  parameter Real R2=20;
  parameter Real C=1.e-7;
  parameter Real L=1.5e-3;
  Real U0;
  Real i;
  Real ir1;
  Real ir2;
  Real ur1;
  Real ur2;
  Real ul;
  Real uc;

equation
  ir1 = C*der(uc);
  ul = L*der(ir2);
  ur1 + uc = U0;
  ur2 + ul = U0;
```

```
  i = ir1 + ir2;
  ur1 = ir1*R1;
  ur2 = ir2*R2;
  U0 = 1;

end Vezje;
```

Prepričamo se, da na vse tri opisane načine (grafično (slika 1.21) in oba tekstovna načina) dobimo enake rezultate. Vse napetosti prikazuje slika 1.22.



Slika 1.22: Poteki napetosti v električnem vezju.

1.8 Dogodkovno in hibridno modeliranje

Realni sistemi imajo ponavadi zvezno vedenje, kar pomeni, da so vse veličine zvezne funkcije časa. Vendar pa se pogosto nekatere veličine spreminjajo mnogo hitreje kot druge. Zato postanejo modeli enostavnejši in preglednejši, če takim komponentam dodelimo diskretno vedenje, kar pomeni, da se veličine spremenijo v neskončno kratkem času oz. hipno, nezvezno. Nekaj primerov, kjer je upravičeno tako modeliranje:

- trenje v mehanskih sistemih,
- odboj žoge,

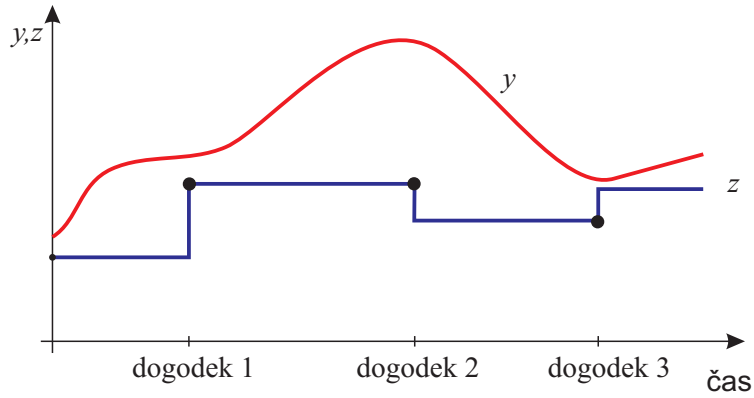
- električna vezja z diodami in drugimi polprevodniškimi komponentami,
- ventili in črpalke v kemičnih procesih, ...

Seveda pa so še drugačni vidiki dogodkovnega in hibridnega modeliranja, kjer dogodkovni del ni posledica poenostavitev v modeliranju, ampak imamo povsem naravno kombinacijo različnih podmodelov (zveznih ali diskretnih), ki se aktivirajo pri določenih pogojih. Za to je značilno zlasti področje vodenja sistemov. Tu se pokaže potreba po zamenjavi regulatorja ob določenih pogojih ali pa periodično proženje dogodkov v primeru regulacijakih sistemov z diskretno izvedenim regulatorjem.

Dogodek je določen s časovnim trenutkom (s trajanjem 0) Prisoten je (logični) pogoj, ki proži dogodek Z dogodkom je povezan niz spremenljivk Ustrezno vedenje, povezano z dogodkom, npr. niz enačb se aktivira (ali deaktivira) pri nastopu dogodka.

Govorimo o komponentah z dogodkovno (diskretno) dinamiko. Hibridni model torej združuje zvezne in dogodkovne aktivnosti. Z vpeljavo obravnavanih poenostavitev lahko zelo pospešimo simulacijo, vendar le, če okolje za modeliranje in simulacijo pravilno detektira in obdela nezveznosti oz. diskretne dogodke. Hibridno modeliranje torej prinaša nove nevšečnosti, a bistveno več prednosti.

Zato jezik Modelica dopušča tudi t.i. diskretne spremenljivke, ki se spreminjajo le v določenih trenutkih - dogodkih, med dogodki pa so vrednosti konstantne. Slika 1.23 prikazuje eno zvezno in eno diskretno veličino.



Slika 1.23: Zvezna spremenljivka y in diskretna spremenljivka z

Diskretni dogodek ima naslednje značilnosti:

- Določen je s časovnim trenutkom (s trajanjem 0).
- Prisoten je (logični) pogoj, ki proži dogodek.
- Z dogodkom je povezan niz spremenljivk.
- Ustrezno vedenje, povezano z dogodkom, npr. niz enačb se aktivira (ali deaktivira) pri nastopu dogodka.

Primeri diskretnih spremenljivk so spremenljivke tipa `Real` s predpono `discrete`, spremenljivke tipa `Boolean` in `Integer`. Nekaj primerov:

Zvezne spremenljivke	Diskretne spremenljivke
`Real x;`	`discrete Real x;`
`Voltage v;`	`Integer i;`
`Current i;`	`Boolean b;`

Razen ustrezno deklariranih spremenljivk vsebuje Modelica za modeliranje hibridnih sistemov predvsem dva načina za proženje dogodkov:

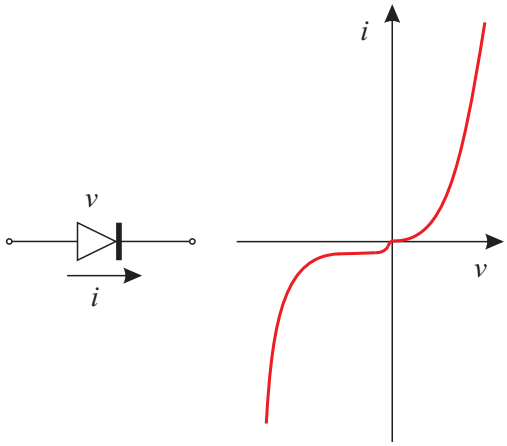
- `if-than-else` stavek za izvedbo pogojnih in nezveznih komponent oz. modelov,
- `when` stavek, ki omogoča, da se enačbe izvršijo le v trenutkih nezveznosti.

Proženje dogodka `if-then-else`

S stavkom lahko spreminjamo pogoje v različnih delovnih točkah, menjamo regulacijske algoritme med delovanjem ipd. Torej po nastopu nekega pogoja veljajo druge enačbe ali izrazi. Primer prikazuje model omejevalnika (limiterja), bloka, ki ima vhod `v` in izhod `y`:

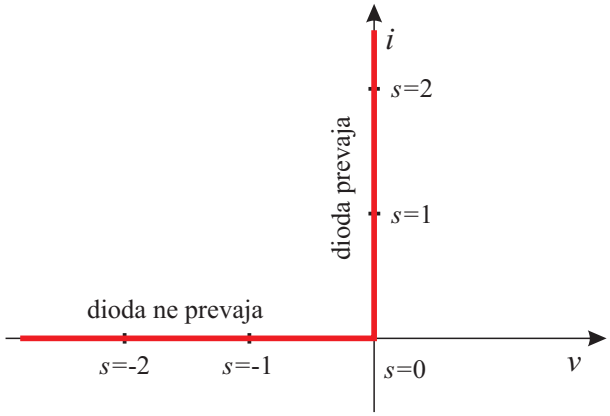
```
y = if v > limit then limit else v;
```

Bolj zahteven je model idealne diode. Dejansko karakteristiko prikazuje slika 1.24. Za negativne napetosti dioda ne prevaja, tok je približno nič. Za poz-



Slika 1.24: Karakteristika diode

itivne napetosti pa postane upornost zelo majhna in napetost približno nič. Zato pogosto uporabljamo poenostavljeni model s karakteristiko na sliki 1.25. Program v Modelici je naslednji:



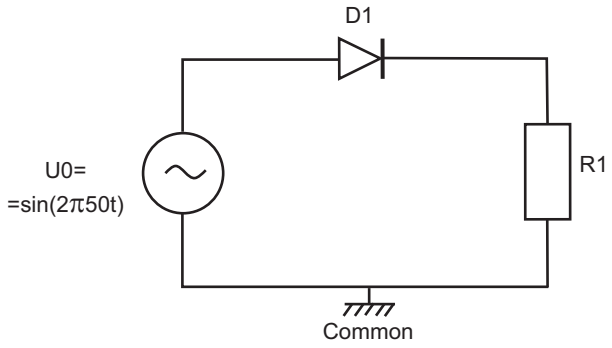
Slika 1.25: Karakteristika idealne diode

```
class Diode "Ideal diode"
  extends TwoPin;
  Real s;
  Boolean off;
equation
  off = s < 0;
  if off then
```

```
    v=s
  else
    v=0;  //conditional equations
  end if;
  i = if off then 0 else s; //conditional expression
end Diode;
```

Vidimo dva primera uporabe stavka if: za pogojne enačbe in za pogojni izraz. Uporabili smo pomožno spremenljivko `s`. V zaprtem področju je napetost enaka pomožni spremenljivki, tok je nič. V prevodnem delu je tok enak spremenljivki s, napetost pa je nič.

Slika 1.26 prikazuje preprosto usmerniško vezje z izmeničnim virom napetosti, uporom in diodo. Tok teče le v vsake pol periode, ko dioda prevaja.



Slika 1.26: Preprosto usmerniško vezje

. celotem program v Modelici pa je naslednji:

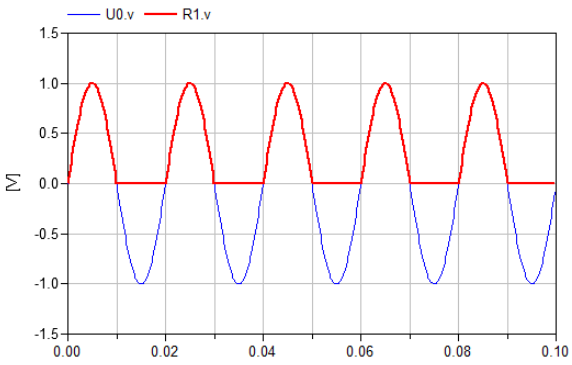
```
model diode_circuit
 Resistor R1(R=100);
 Diode D1;
 sinVoltage U0(U=1);
 Ground Common;

equation
 connect( U0.p, D1.p);
 connect( D1.n, R1.p);
 connect( R1.n, Common.p);
 connect( U0.n, Common.p);
```

```
 end diode_circuit;

class sinVoltage "Ideal voltage source"
extends TwoPin;
parameter Modelica.SIunits.Voltage U=1;
equation
 v=U*sin(6.28*50*time);
end sinVoltage;
```

Ker so bili vsi drugi modelni razredi že prej opisani, na tem mestu dodajamo le modelni razred za sinusni napetostni vir amplitude 1 in frekvence 50 Hz. Potek napetosti na uporu opisuje slika 1.27.



Slika 1.27: Potek napetosti na uporu

Proženje dogodka `when`

Operator `when` se v Modelici uporablja za proženje enačb, ki se izvršijo le enkrat, v enem trenutku oz. ob izpolnjenem logičnem pogoju. Sintaksa je naslednja:

```
when <conditions> then
   <equations>
end when;
```

Tipični so **časovno pogojeni dogodki** (angl. time events). Logični pogoji torej vsebujejo le neodvisno spremenljivko - običajni čas, kar pomeni, da je razpored

dogodkov znan vnaprej. Zato so tudi trenutki nastopov nezveznosti znani vnaprej in omogočajo enostavnejše postopke za odpravo problemov numerične integracije preko nezveznosti. Primer:

```
when time >= 10.0 then
  ...
end when;
```

Druga vrsta so **stanjski dogodki** (angl. state events). Dogodki se prožijo v odvisnosti od odvisnih spremenljivk (stanj sistema) in jih torej ni mogoče predvideti vnaprej. Numerično integriranje preko nezveznosti je bolj zahtevno in vključuje identifikacijo trenutka nezveznosti (angl. postopek zero crossing).

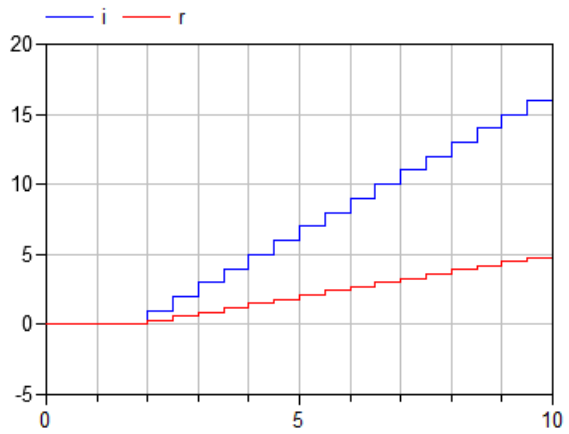
```
when sin(x) > 0.5 then
  ...
end when;
```

Proženje periodičnih dogodkov - `when+sample`

Periodično proženje dogodkov je zlasti pomembno pri modeliranju vzorčenja - npr. v regulacijskih sistemih z računalniško izvedenimi regulatorji. Ukaz `sample(T0,Ts)` vrne vrednost true in proži dogodke v trenutkih `T0+i*Ts,i=0,1,2, ...` , kjer je `T0` začetni čas, `Ts` je čas vzorčenja. Uporabimo ga za logični pogoj stavka `when`. Primer prikazuje generiranje dveh stopničasto naraščajočih signalov. Eden je celoštevilčnega tipa, eden pa realnega.Ukaz `pre` uporabljamo za oznako pretekle vrednosti neke spremenljivke (npr. nova (trenutna) vrednost je stara (shranjena) vrednost +1).

```
model SamplingClock
  Integer i;
  discrete Real r;
equation
  when sample(2,0.5) then
    i = pre(i)+1;
    r = pre(r)+0.3;
  end when;
end SamplingClock;
```

Rezultate simulacije prikazuje slika 1.28.

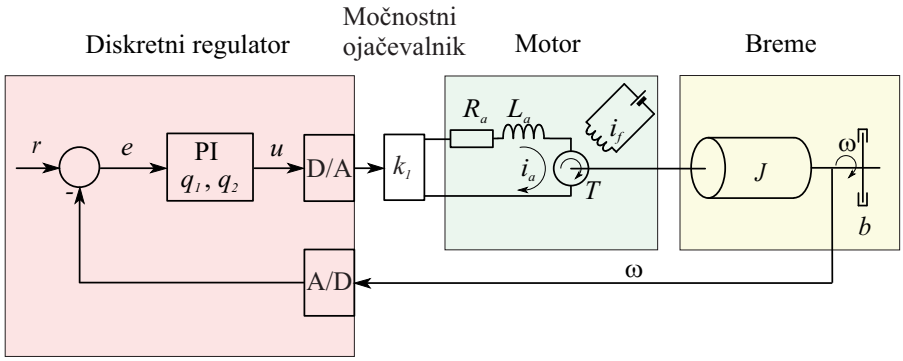


Slika 1.28: Proženje periodičnih dogodkov - generiranje naraščajočih stopničastih signalov

Primer: Diskretna PI regulacija hitrosti vrtenja motorja

Na podoben način izvedemo program za diskretno regulacijo. Namesto stavkov, ki določata `i` in `r`, vključimo algoritem za diskretni PI regulator.

Slika 1.29 prikazuje bločno shemo regulacijskega sistema z diskretnim PI regulatorjem in modelom enosmernega motorja. Regulator regulira hitrost vrtenja (kotno hitrost) motorja.



Slika 1.29: Diskretna regulacija hitrosti vrtenja

Pri konstantnem statorskem toku i_f je moment T, ki ga generira motor, proporcionalen rotorskemu toku

$$T = k_2\, i_a \tag{1.9}$$

pri čemer je k_2 momentna konstanta motorja. Gradnik v modelu imenujemo elektromehanski pretvornik (angl. electro-motoric force - EMF).

Za rotorski tokokrog pa velja električna enačba

$$L_a \frac{di_a}{dt} + R_a\, i_a + k_3 \frac{d\vartheta}{dt} = k_1\, u \tag{1.10}$$

kjer je k_3 indukcijska konstanta in $k_3 \frac{d\vartheta}{dt}$ ustrezna inducirana napetost, ki je seveda proporcionalna hitrosti vrtenja motorja. L_a je induktivnost, R_a pa upornost rotorskega tokokroga. k_1 je ojačenje močnostnega ojačevalnika, ki signal regulatorja spreminja v ustrezno napetost. Momentna ravnotežna enačba pa ima obliko

$$J\frac{d^2\vartheta}{dt^2} + b\,\frac{d\vartheta}{dt} = T = k_2\, i_a \tag{1.11}$$

Pri tem je J vztrajnostni moment motorja in bremena, b pa je ustrezni koeficient viskoznega dušenja.

Diskretni PI regulator je opisan z algoritmom

$$u(k) = u(k-1) + q_0 e(k) + q_1 e(k-1) \tag{1.12}$$

$u(k)$ in $u(k-1)$ sta trenutna in pretekla vrednost regulirnega signala, $e(k)$ in $e(k-1)$ pa trenutna in pretekla vrednost signala pogreška. q_0 in q_1 sta parametra diskretnega PI regulatorja in se izračunata iz zveznih parametrov k_p in k_i ter iz časa vzorčenja T_s.

Vsi parametri so razvidni iz programa v jeziku Modelica:

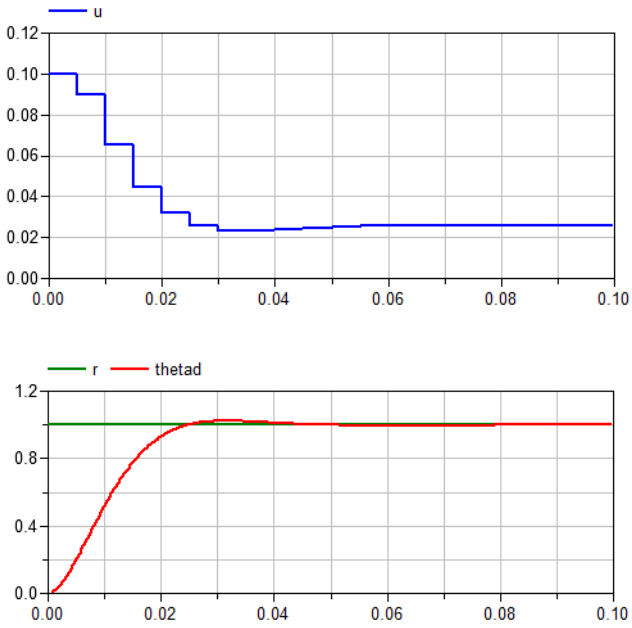
```
model DCmotor_velocity_discrete
  "Regulacija kotne hitrosti, Primer 8.1"
  parameter Real Ra=0.5 "upornost rotorja";
  parameter Real La=1.5E-03 "induktivnost rotorja";
  parameter Real k1=2 "mocnostni ojacevalnik";
  parameter Real k2=0.05 "momentna konstanta";
  parameter Real k3=0.05 "indukcijska konstanta";
  parameter Real kp=0.1 "P regulator";
  parameter Real ki=2 "I clen regulatorja";
```

```
  parameter Real J=0.00025 "vztrajnostni moment";
  parameter Real b=0.0001 "dusenje";
  parameter Real r=1 "zelena kotna hitrost";
  parameter Real Ts=0.005 "cas vzorcenja";
  parameter Real q0=kp "par. reg.";
  parameter Real q1=-kp*(1-Ts*ki/kp) "par. reg.";
  Real T;
  Real ia;
  Real iad;
  Real theta;
  Real thetad;
  Real thetadd;
  discrete Real e;
  discrete Real u;

equation
//emf
  T=k2*ia;
//el. enacba rotorskega tokokroga
  La*iad+Ra*ia+k3*thetad=k1*u;
//mehanske enacbe
  J*thetadd+b*thetad=T;
  iad=der(ia);
  thetad=der(theta);
  thetadd=der(thetad);

//diskretni PI regulator
  when sample(0,Ts) then
    e=r-thetad;
    u=pre(u)+q0*e+q1*pre(e);
  end when;
end DCmotor_velocity_discrete;
```

`pre(u)` in `pre(e)` uporabimo za izvedbo členov $u(k-1)$ in $e(k-1)$ v enačbi 1.12. Rezultate simulacije pri času vzorčenja $T_s = 5ms$ prikazuje slika 1.30. Lepo je razvidna razlika med diskretno spremenljivko `u` (regulirni signal)in zvezno spremenljivko `thetad` (regulirani signal).



Slika 1.30: Rezultati simulacije regulacije hitrosti vrtenja

Trenutna sprememba spremenljivke - `when+reinit`

Vrednost časovno zvezne spremenljivke stanja lahko trenutno spremenimo z ukazom reinit znotraj when-stavka. Znan je model skakajoče žoge, ki se ponavadi nahaja med testnimi primeri simulacijskih orodij. Žogo spustimo z določene začetne višine in z začetno hitrostjo. Ko doseže tla, se odbije. Odboj modeliramo tako, da v trenutku spremenimo hitrost. Hitrost po odboju ima obratno in zaradi izgube energije nekoliko nižjo vrednost. Za modeliranje uporabimo Newtonove zakone

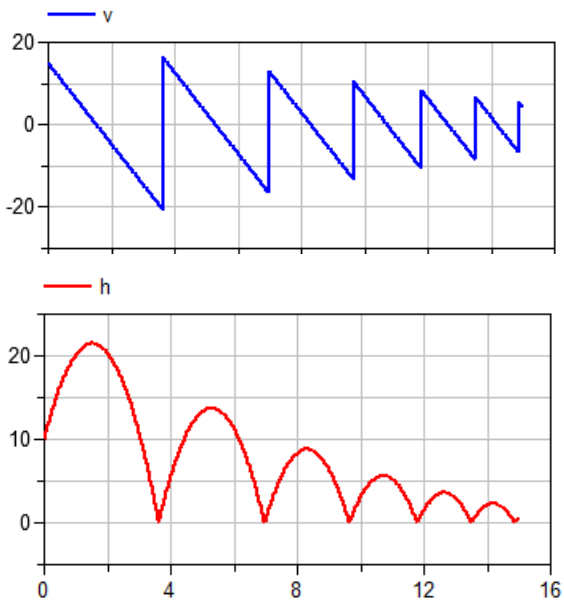
$$\begin{aligned} \sum F_i &= ma \\ -mg &= ma = mh'' \\ h'' &= -g = -9.81\ [m/s^2] \\ h(0) &= 10\ [m] \\ v(0) &= 15\ [m/s] \\ h(t) = 0 \longrightarrow v &= kv \quad k = -0.8 \end{aligned} \tag{1.13}$$

Program v jeziku Modelica je naslednji:

```
model BouncingBall "the bouncing ball model"
  parameter Real g=9.81;   //gravitational acc.
  parameter Real k=0.80;   //elasticity constant
  Real h(start=10);
  Real v(start=15);
equation
  der(h) = v;
  der(v)=-g;
  when h<0 then
    reinit(v, -k*v);
  end when;
  end BouncingBall;
```

Ko gre hitrost iz pozitivne v negativno področje, spremenimo vrednost hitrosti v skladu z enačbo $v = -0.8v$.

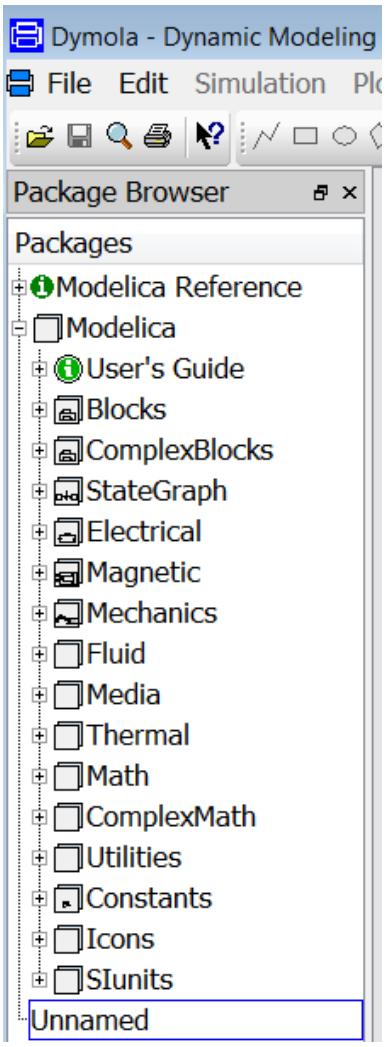
Slika 1.31 prikazuje potek hitrosti $v(t)$ in višine $h(t)$.



Slika 1.31: Potek hitrosti $v(t)$ in višine $h(t)$

1.9 Standardna knjižnica Modelica

Modelica okolja imajo že pripravljeno knjižnico pogosto uporabljenih komponent. Knjižnica je standardizirana kot je standardiziran jezik Modelica.



Standardna knjižnica Modelica je torej sestavni del same Modelice. Vsebuje konstante, deklaracije types, priključke, delne modele (partial models), in modelne gradnike iz različnih področij. Knjižnice so razvijali strokovnjaki iz posameznih področij skupaj z dobrimi poznavalci jezika Modelica. Trenutno so na voljo naslednje področja:

Blocks
Vhodno-izhodni bloki po vzoru tradicionalnih simulacijskih okolij, npr. prenosne funkcije, fitri, regulatorji, viri signalov (kot bloki v Simulinku) - zvezno in diskretno.
StateGraph
Modeliranje dogodkovnih sistemov s stanjskimi grafi. Ločeno obstaja še kompleksnejša knjižnica.
Electrical
Modeli električnih in elektronskih komponent: analogna vezja, digitalna vezja, električni stroji, ...
Magnetic
Komponente za gradnjo modelov elektromagnetnih sistemov.
Mechanics
Eno in tro dimenzionalni mehanski sistemi (translatorni, rotacijski, multy body).

Fluid
Eno dimenzionalni modeli termo-tekočinskih komponent. **Media**
Snovni modeli, ki se uporabljajo tudi v drugih knjižnicah, npr. v Fluid.

Thermal
Modeliranje toplotnih sistemov z vsemi bistvenimi prenosi toplote: konvekcija,

sevanje, prevajanje.

Math
Matematične funkcije (exp, sin, ..), matrično računanje (det, eig, ...).

Utilities
Funkcije za pisanje skriptnih datotek (manipuliranje z datotekami, znakovnimi spremenljivkami, ...).

Constants
Matematične in fizikalne konstante (pi, eps, R, sigma, ...).

Icons
Definicije ikon, ki se pogosteje uporabljajo.

SIunits
Definicija fizikalnih veličin po standardu ISO 31-1992.

Razen knjižnic za gradnjo modelov je na voljo še knjižnica **Modelica Reference**, ki je pravzaprav opis jezika Modelica.

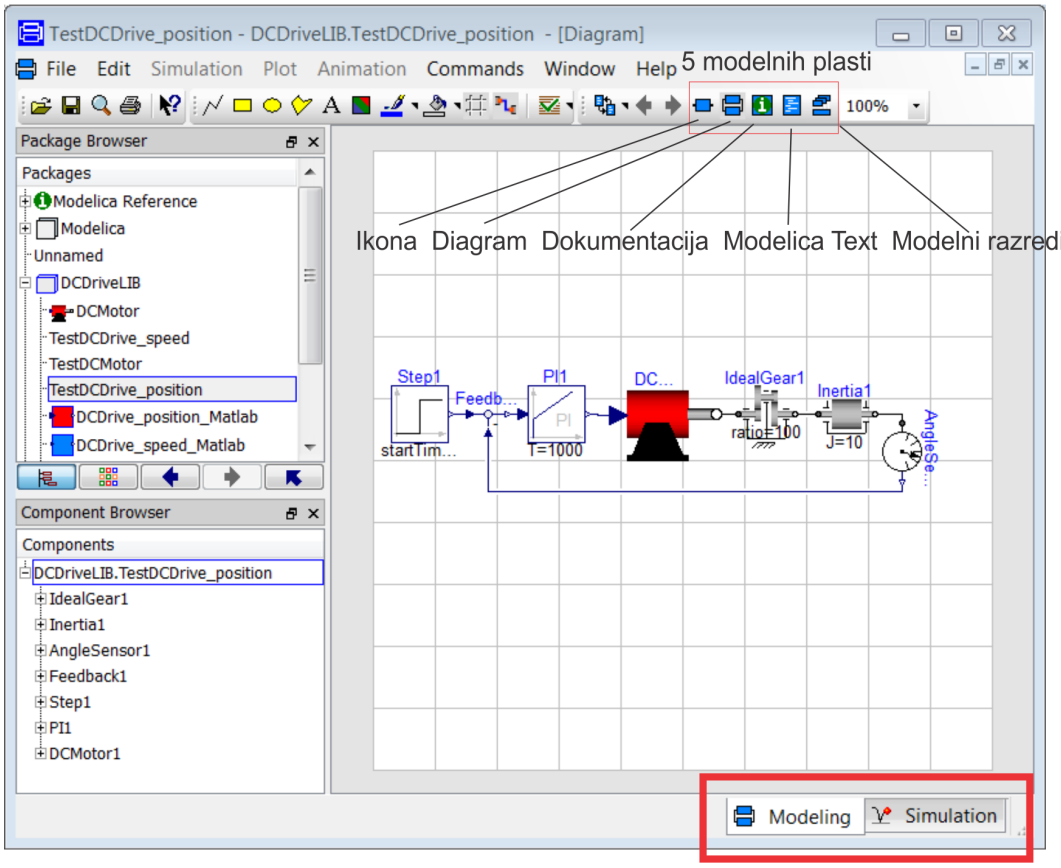
Obstajajo pa še številne druge knjižnice. Nekatere je razvila Modelica Association. Mnogo knjižnic je javno dostopnih, nekatere pa so plačljive. Natančna informacija je na voljo na spletni strani https://www.modelica.org/libraries

1.10 Okolje Dymola

Okolje Dymola omogoča razvijanje in izvajanje modelov v jeziku Modelica. Modele ali modelne razrede se včasih opiše grafično (predvsem na višjih hierarhičnih nivojih), včasih tekstovno (predvsem na nižjih hierarhičnih nivojih). Okolje spoštuje osnovni koncept modeliranja in simulacije kot enovitega cikličnega in iterativnega postopka in je temu prirejen tudi osnovni izgled uporabniškega vmesnika. Med delom namreč menjavamo pretežno dve okni - Modelling in Simulation. Način izberemo v desnem spodnjem kotu zaslona.

1.10.1 Modeliranje

Ko vstopimo v okolje Dymola, se prikaže zaslon v načinu Modelling, kar prikazuje slika 1.32.



Slika 1.32: Izgled zaslona pri zagonu okolja Dymola

V tem načinu razvijamo nove ali editiramo obstoječe modele. Glavno okno je namenjeno prikazu modela. Okno v levem zgornjem delu (Package Browser - knjižnično okno) prikazuje knjižnice, ki so na voljo za gradnjo modelov. Le-ta poteka tako, da iz knjižnic potegnemo z miško ikone v glavno okno modela in jih med sabo povežemo. Tekstovne modele pa pišemo v tekstovnem urejevalniku. Okno v spodnjem levem delu (Component Browser - zgradba modela) pa prikazuje drevesno komponentno zgradbo odprtega modela.

Modelni razred kot osnovni gradnik vsakega modela je sestavljen iz večih plasti:

- Icon layer
- Diagram layer
- Documentation layer
- Modelica Text layer
- Used Classes layer

Prvi dve plasti sta grafični, ostale tri pa tekstovne. Ni pa nujno, da vsak modelni razred vsebuje vse plasti. V glavnem oknu je prikazana le ena plast. Med različnimi plastmi preklapljamo s pomočjo opravilne vrstice, kot prikazuje slika 1.32. V tej opravilni vrstici je priporočljiva uporaba ikone `Check`, s pomočjo katere lahko preverjamo pravilnost modela med razvijanjem, še pred celotnim prevajanjem.

Icon layer - ikona

Plast prikazuje ikono komponente, kar je zlasti pomembno, če komponento povezujemo na višjih hierarhičnih nivojih.

Diagram layer - diagram

Plast se odpre kot privzeta predstavitev in prikazuje modelni razred s pomočjo grafičnega diagrama, v katerem so razvidne komponente, priključki in povezave.

Documentation layer - dokumentiranje

Plast hrani dokumentacijo modelnega razreda oz. omogoča ustrezno dokumentirati modelni razred. To je torej okno pomoči, ki opisuje model, njegove parametre, uporabnost, posebnosti, ... Lahko vključujemo tekstovno in grafično informacijo.

Modelica Text layer - program v Modelici

Ta plast prikazuje tekstovni program v jeziku Modelica z deklarativnim in enačbnim delom. Deli programa, ki so opisani z grafiko, so privzeto skriti in označeni

s posebnima ikonama (ena ikona v deklarativnem, ena pa v enačbnem delu). Ti dve ikoni se zamenjata s tekstom, če kliknemo na + pri ikoni ali z desnim klikom izberemo `Expand>Show components and connect`.

Used Classes layer - vsi modelni razredi

Plast vsebuje vse modelne razrede celotne hierarhične strukture nekega modela vključno z definicijami priključkov. To je pravzaprav koda celotnega modela v obliki tekstovnega programa. Vendar ne omogoča editiranja. Tudi tu so grafično definirane komponente privzeto skrite.

1.10.2 Simulacija

Potem ko končamo z modeliranjem, preklopimo zaslon v način simulation. Način nam omogoča eksperimentiranje z modelom. Izgled zaslona prikazuje slika 1.33. Glavni namen te faze eksperimeniranja je prikaz časovnih potekov spremenljivk, v bolj zahtevnih primerih pa tudi prikaz animacije.

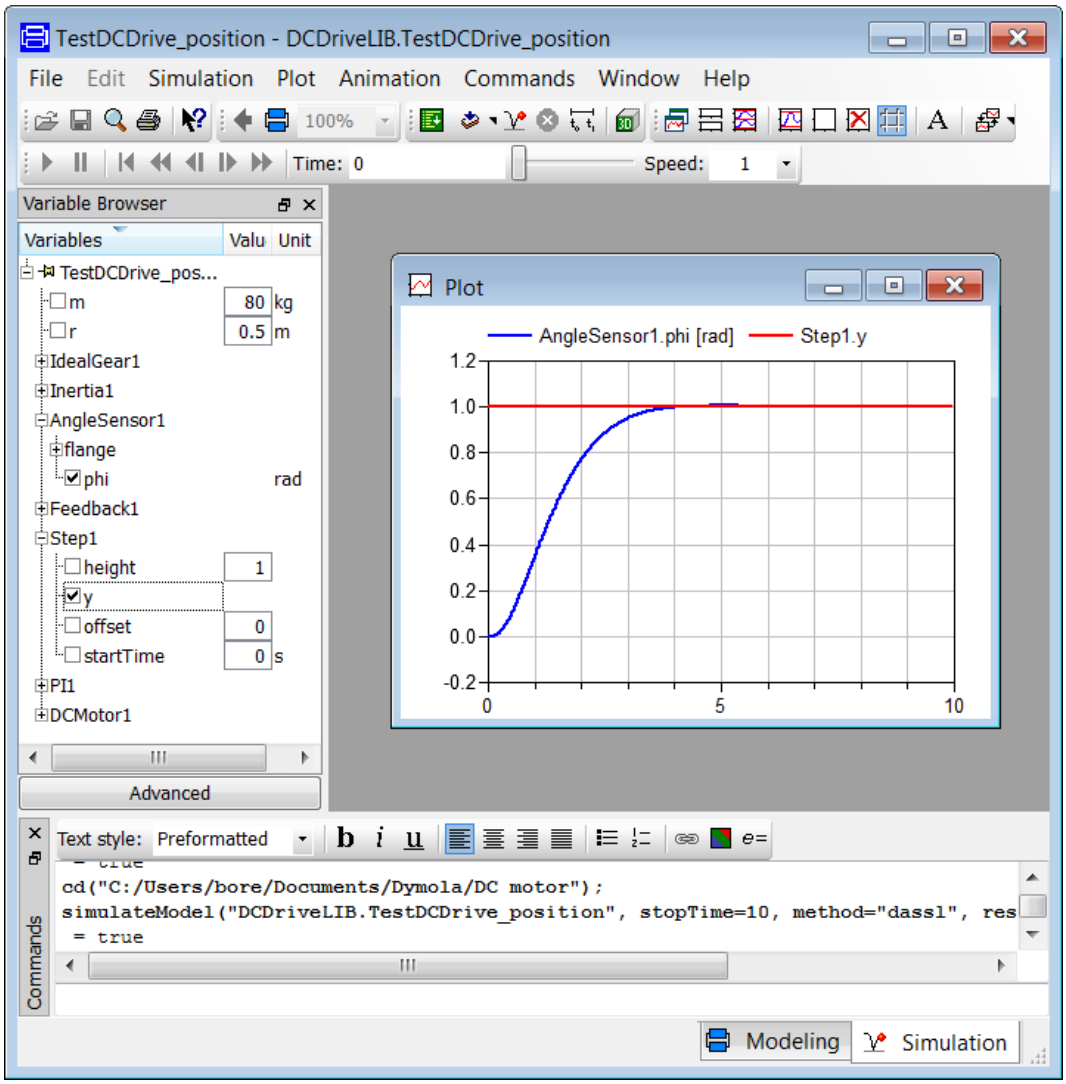
Variable browser - manipuliranje s spremenljivkami

Okno, ki se odpre v levem delu zaslona, vsebuje drevesno strukturo spremenljivk v modelu. Izberemo lahko spremenljivke za grafični prikaz, nastavljamo začetne pogoje, vrednosti parametrov. Izpisujejo se tudi morebitne enote enote in pa komentarji za posamezne spremenljivke. Čeprav privzeto način Simulation ne prikazuje diagram modela, ga ga lahko odpremo, saj na ta način laže določimo spremenljivke za izris (v diagramu se z miško dotaknemo priključka in izpiše se celotno ime spremenljivke).

Plot window - okno za prikaz spremenljivk Privzeto se v desnem delu odpre eno okno za izris spremenljivk. Lahko pa odpremo dodatna okna. V vsako okno lahko narišemo poljubno število spremenljivk, lahko pa ga razdelimo na dve podokni.

Command window - ukazno okno

Komandno okno je privzeto odprto v simulacijskem načinu, lahko pa ga odpremo tudi v načinu modeliranja. Nahaja se v spodnjem deli zaslona in je sestavljeno



Slika 1.33: Izgled zaslona med fazo simulacije

iz dveh delov: command log pane in command input line.

Command log pane (dokumentiranje eksperimeniranja)

V tem oknu se zbirajo koristne informacije med eksperimentiranjem: ukazi uporabnika, odgovori na komande, uporabnik pa lahko doda še razne informacije za bolj pregledno dokumentiranje postopkov.

Informacijo, ki se nabira med eksperimeniranjem, lahko shranimo na datoteko s

komando `File>Save Log`. Možni pa so trije načini shranjevanja: ukazi uporabnika in odgovori na ukaze, celotna informacija kot dokumentacija eksperimentiranja, samo ukazi, ki se lahko kasneje uporabijo za izvedbo eksperimenta s pomočjo skriptne datoteke.

Command input line (ukazna vrstica)

Preko te vrstice se lahko vpisujejo razni ukazi (npr. start simulacije, nastavitev parametrov, začetnih vrednosti, izbira spremenljivk za prikaz, ..), ki se sicer izvajajo tudi preko orodnih vrstic uporabniškega vmesnika in nekaterih drugih oken. Zato pri enostavnejšem eksperimentiranju ukazne vrstice ne uporabljamo. Pomembna pa je za eksperimeniranje s pomočjo skriptnega načina.

Message window (sporočilno okno)

Okno se privzeto ne prikazuje, ustrezna informacija pa nastane med procesiranjem modela (`Simulation>Translate` ali `Simulation>Simulate`). Uporabnik odpre okno z ukazom `Simulation>Show Log`. Okno se prikazuje tudi v načinu modeliranja, če npr. ob kliku ikone `Check` sistem zazna napake. V zgornjem delu okna je opravilna vrstica, s katero izbiramo med `Syntax Error`, `Translation`, `Dialog Error` in `Simulation`.

`Syntax error`

Predvsem se tu javljajo sintaksne napake pri modeliranju s tekstovnim načinom v jeziku Modelica.

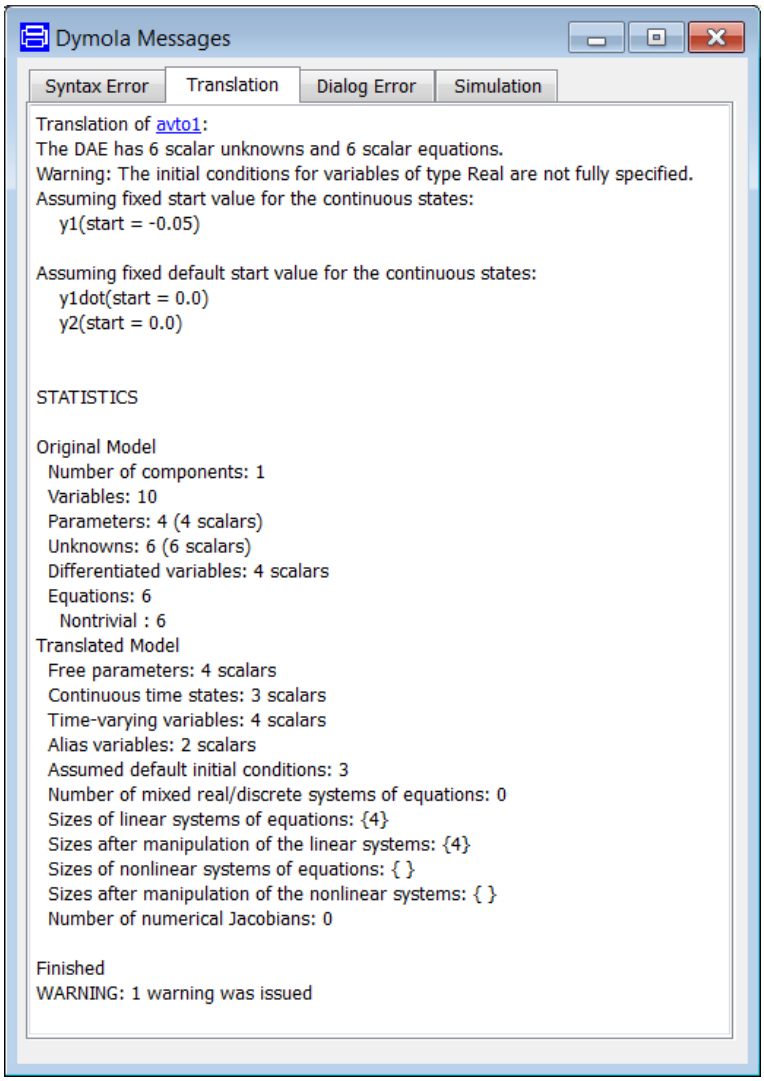
`Translation`

Prikazuje informacijo in statistiko pri prevajanju in simboličnem preurejanju: število neznank in enačb, manjkajoči začetni pogoji, število stanj, ... Informacije so zelo koristne pri razhroščevanju zahtevnejših modelov. Slika 1.34 prikazuje okno po prevajanju modela avtomobilskega vzmetenja (`avto1.mo`).

`Dialog Error`

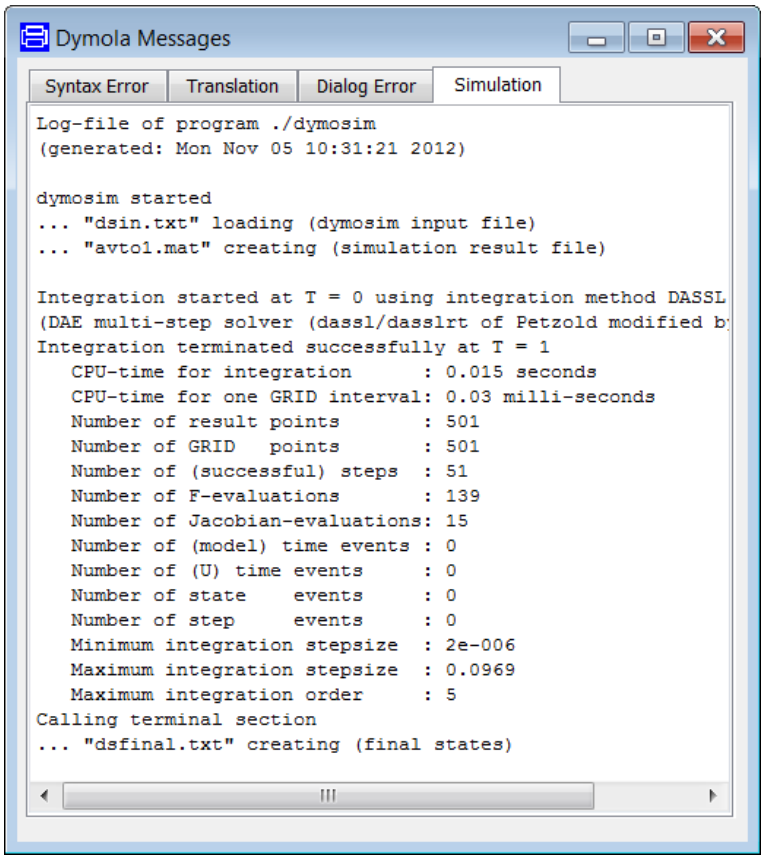
Sporočajo se zahtevnejše napake, ki jih npr. Check postopek pri modeliranju ne ugotovi.

`Simulation`



Slika 1.34: Sporočilno okno Translation

Izpiše se pomembna informacija v zvezi z izvajanjem simulacije: začetni in končni čas simulacije, izbrani integracijski postopek, potreben CPU čas, število točk rezultatov simulacije, identifikacija dogodkov, število izračunov Jacobijeve matrike, ... Informacija je pomembna za razhroščevanje in učinkovitejše izvajanje simulacije. Slika 1.35 prikazuje okno po prevajanju modela avtomobilskega vzmetenja (`avto1.mo`).



Slika 1.35: Sporočilno okno Simulation

1.11 Uporaba Dymola-Modelica bloka v okolju Matlab-Simulink

Pri povezovanju različnih okolij je seveda prednost, če lahko združimo dobre lastnosti posameznih okolij. Glavna prednost okolja Simulink je delovanje v okolju Matlab, kar omogoča, da učinkovito programiramo simulacijske eksperimente: optimizacijo, linearizacijo, analizo ustaljenega stanja, ... Prednost okolja Dymola-Modelica je seveda večdomensko OO in fizikalno modeliranje, ki omogoča gradnjo resnično ponovno uporabljivih komponent. Dymola-Matlab vmesnik omogoča (z nekaj dodatnimi m datotekami), da shemo v Modelici (s specificiranimi vhodi in izhodi) uporabimo kot Dymola blok v okolju Simulink.

Da lahko v Matlabu uporabimo vmesnik, moramo v Matlabovo pot (path) vključiti naslednje vrstice (ob predpostavki, da smo okolje Dymola namestili na

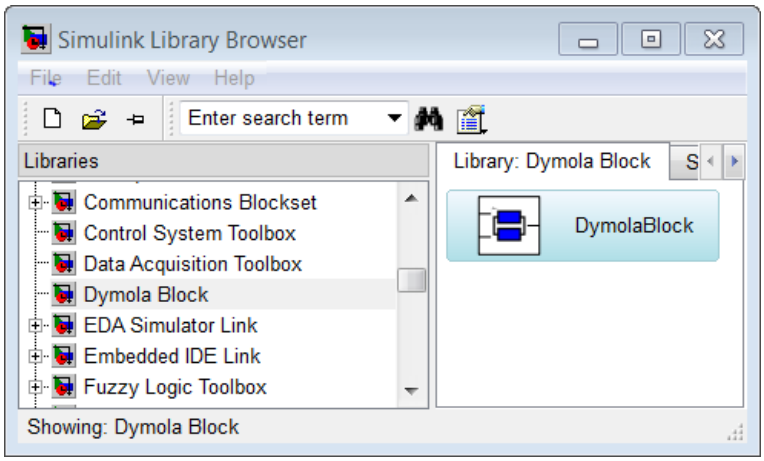
`Program Files\Dymola`):

```
Program Files\Dymola\Mfiles
Program Files\Dymola\Mfiles\dymtools
Program Files\Dymola\Mfiles\tray
```

Prav tako moramo zagotoviti delujoče Matlab/MEX okolje, kar pomeni, da je v Matlabu možno prevajanje v obliko, ki jo potem okolje Matlab lahko izvaja. Zmožnost prevajanja lahko preverimo z ukazom `mex matlab\extern\examples\mex\yprime.c`

Vmesnik podpira Visual Studio C++ prevajalnik (LCC prevajalnik ni več podprt).

Dymola-Modelica vmesnik (blok) do Simulinka se nahaja v Simulinkovem knjižničnem oknu DymolaBlock in ima ime DymolaBlock z ikono, ki jo prikazuje slika 1.36.



Slika 1.36: Knjižnično okno DymolaBlock

DymolaBlock predstavlja model v Modelici in se poveže s Simulink bloki ali drugimi Dymola bloki. Črtice na levi in desni se med prevajanjem zamenjajo z ustreznimi priključki. DymolaBlock je ovoj okoli S-funkcije MEX bloka, torej je vmesnik do programa v C-ju, ki ga naredi okolje Dymola za model v Modelici.

Uporabniški vmesnik za DymolaBlock

Preden DymolaBlock v Simulinku povežemo z ostalimi Simulink bloki, dvakrat kliknemo na blok, da se odpre uporabniški vmesnik. Le ta omogoča izbiro modela, prevajanje v Simulink MEX datoteko, nastavitev parametrov prevajalnika, nastavitev vrednosti parametrov modela v Modelici in določitev začetnih vrednosti. Ime modela in ime datoteke modela v Modelici lahko določimo tako, da ustrezen model odpremo v Dymoli in v vmesniku izberemo `Select from Dymola`.

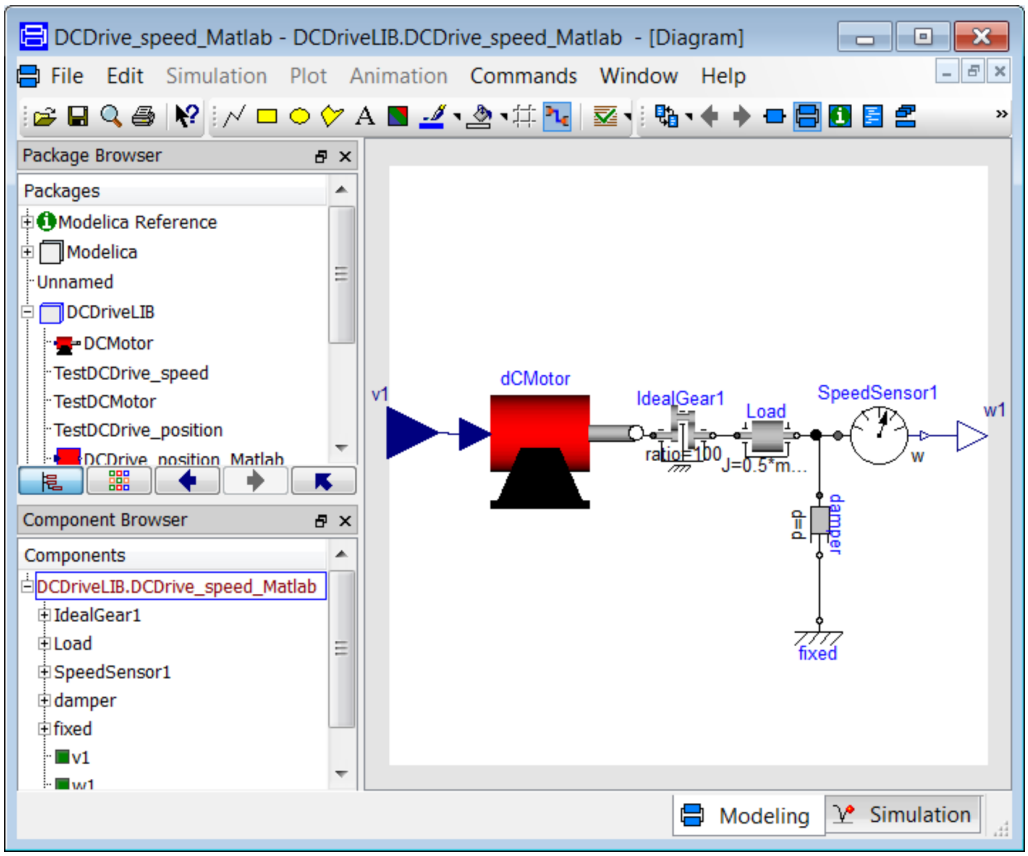
`Edit model` uporabimo, če okolje Dymola in ustrezen model še nista odprta. Nato s komado `Compile model` prevedemo Modelica model. Ikona v Simulinku dobi prave vhode in izhode (priključke) in šele sedaj lahko v Simulinku dokončamo povezovanje. Z dvoklikom na DymolaBlok pa lahko nastavljamo parametre in začetne vrednosti.

Slika 1.37 prikazuje v Dymoli pripravljen model z enosmernim motorjem in vztrajnikom. Na levi je pripravljen priključek za izhod iz regulatorja, ki bo določal vzbujanje motorja, na desni pa je priključek za merjeno kotno hitrost.

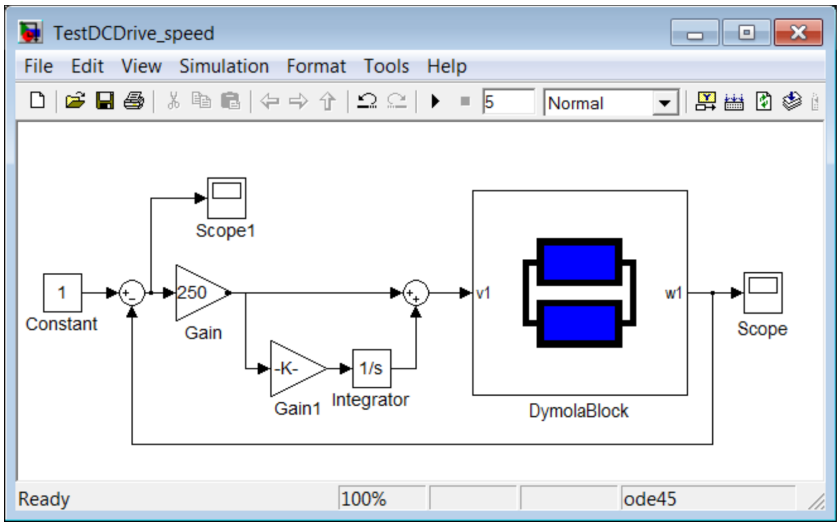
Slika 1.38 prikazuje Simulink model z regulacijsko shemo za regulacijo hitrosti vrtenja in z vsebovanim Modelica blokom DymolaBlock.

Slika 1.39 prikazuje uporabniški vmesnik bloka DymolaBlock.

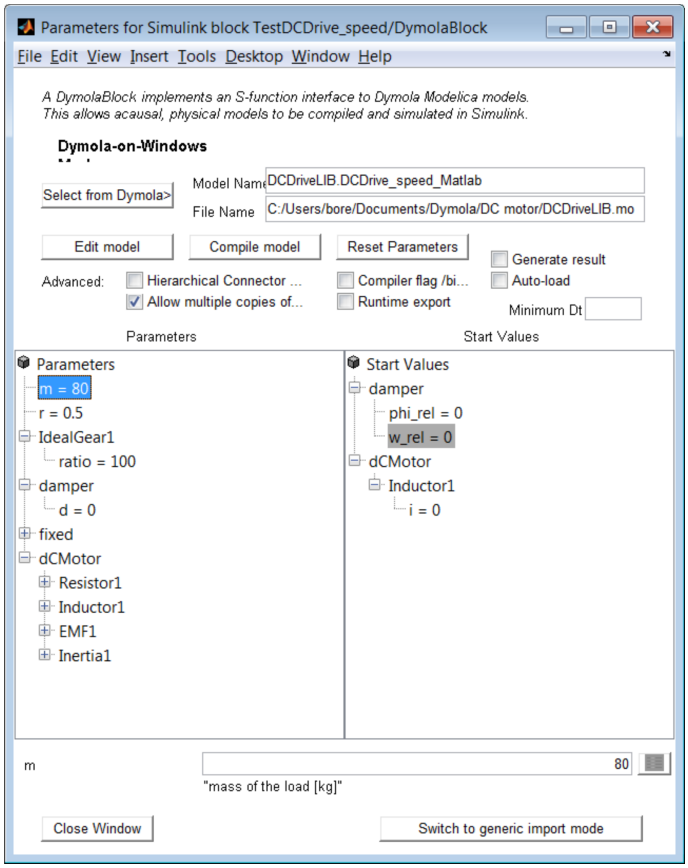
Slika 1.40 pa prikazuje potek hitrosti vrtenja pri stopničasti spremembi želene hitrosti.



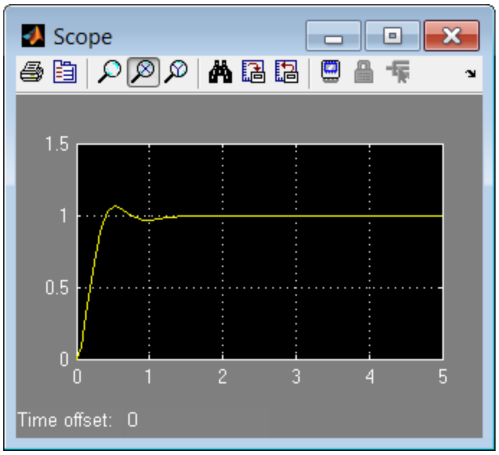
Slika 1.37: Model enosmernega motorja v Dymoli-Modelici



Slika 1.38: Simulink shema za regulacijo hitrosti vrtenja



Slika 1.39: Uporabniški vmesnik bloka DymolaBlock



Slika 1.40: Rezultati simulacije

Literatura

Aburdene, M.F. (1988), *Digital Continuous System Simulation*, Wm.C. Brown Publishers, Dubuque, Iowa, USA.

Cellier, F.E. (1991), *Continuous System Modeling*, Springer - Verlag, New York, USA.

Cha P. D., Molinder J.I.(2006), *Fundamentals of Signals and Systems: A Building Block Approach*, Cambridge University Press, UK.

Dabney J.B., Harman T.L. (2004), *Mastering SIMULINK* , Prentice Hall, Upper Saddle River, N.J., USA.

Dymola (2011), Multi-engineering modeling and simulation, Users manual, ver. 2012. Dessault System, Dynasim AB, Sweden, Lund.

Elmqvist, H. (1978), *A structured model language for large continuous systems*, Ph. D. diss. Rep. CODEN: LUTFD2/(TFRF-1015), Departement of Automatic Control, Lund Institute of Technology, Sweden.

Karba, R. (1994), *Modeliranje procesov*, Založba Fakultete za elektrotehniko in Fakultete za računalništvo in informatiko, Univerza v Ljubljani, SLO

Korn, G.A. and J.V. Wait (1978), *Digital Continuous System Simulation*, Prentice Hall, Englewood Clifs, N.J., USA.

Matko, D., B. Zupančič, R. Karba (1992), *Simulation and Modelling of Continuous Systems - A Case Study Approach.* Prentice Hall.

Matko, D. (1998): *Identifikacije*, Založba Fakultete za elektrotehniko in Fakultete za računalništvo in informatiko, Univerza v Ljubljani, SLO

Mihelič, F. (2006), *Signali*, Založba Fakultete za elektrotehniko in Fakultete za računalništvo in informatiko, Univerza v Ljubljani, SLO

Mlakar, J. (2002), *Linearna vezja in signali*, Založba Fakultete za elektrotehniko in Fakultete za računalništvo in informatiko, Univerza v Ljubljani, SLO

Modelica (2010), A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, Version 3.2, Modelica Association, http://www.modelica.org/documents/ModelicaSpec32.pdf

Neelamkavil, F. (1987), *Computer Simulation and Modelling*, John Willey, New York, USA.

Oblak, S., Škrjanc, I. (2008), *Matlab s Simulinkom, priročnik za laboratorijske vaje* , Univerza v Ljubljani, Založba FE in FRI, Slovenija.

Ogata K. (2010), *Modern Control Engineering*, Fifth edition, Prentice Hall, USA.

Schmidt, B. (1987), "What does simulation do? Simulation's place in the scientific method." *Syst. Anal. Model. Simul.* (Benelux journal), 4 (1987) 3, 193-211.

Simulink, Dynamic System Simulation Software (2009), Users manual, R2009b, Math Works, Inc., Natick, MA, ZDA.

Smith, D.J.M (1995), *Continuous System Simulation*, Chapman & Hall, London, UK.

Sodja, A, B. Zupančič (2009), *Modelling thermal processes in buildings using an object-oriented approach and Modelica*. Simulation Modelling Practice and Theory, vol. 17, no. 6, str. 1143-1159.

Strauss, J.C. (1967), "The SCi continuous system simulation language." *Simulation*, no.9, 281-303.

Strejc, V. (1959): *Aproximation aperiodischer Übertragungscharakteristiken*, Regelungstechnik,7:124-128.

Strmčnik, S., R.Hanus, Đ. Juričić, R. Karba, Z. Marinšek, D.Murray-Smith, H. Verbruggen, B. Zupančič (1998):*Celostni pristop k računalniškemu vodenju procesov*, Univerza v Ljubljani, Fakulteta za elektrotehniko, SLO

Zupančič B. (1989), *Sinteza simulacijskega jezika pri računalniško podprtem načrtovanju sistemov avtomatskega vodenja*, Doktorska disertacija, Fakulteta za elektrotehniko in računalništvo, Univerza v Ljubljani, Ljubljana, Slovenija.

Zupančič B. (1992), *SIMCOS- jezik za simulacijo zveznih in diskretnih dinamičnih sistemov*, Fakulteta za elektrotehniko in računalništvo, Univerza v Ljubljani, Slovenija.

Zupančič, B. (2010a): *Simulacija dinamičnih sistemov*, Založba Fakultete za elektrotehniko in Fakultete za računalništvo in informatiko, Univerza v Ljubljani, SLO

Zupančič, B. (2010b), *Zvezni regulacijski sistemi, I.del*, Založba Fakultete za elektrotehniko in Fakultete za računalništvo in informatiko, Univerza v Ljubljani, SLO

Zupančič, B. (2010c), *Zvezni regulacijski sistemi, II.del*, Založba Fakultete za elektrotehniko in Fakultete za računalništvo in informatiko, Univerza v Ljubljani, SLO

Zupančič, B. (2012):*Avtomatsko vodenje sistemov*, delovna verzija učbenika (http://msc.fe.uni-lj.si/Books.asp?book=1), Fakulteta za elektrotehniko, Univerza v Ljubljani, SLO

Howell, J.R. (1993): *Application of Monte Carlo to Heat Transfer Problems, Advances in Heat Transfer*, Academic Press, San Diego, Vol 5, 1-54.

Howell, J.R. (1998): *The Monte Carlo Method in Radiative Heat Transfer*, Journal of Heat Transfer, ASME, Vol. 120, 547-560.

Annino, J. S., E.C. Russel (1979), "The ten most frequent causes of simulation analysis failure - and how to avoid them!" *Simulation*, 137-140.

Araki, M. (1985), *Industrial applications in Japan of computer aided design packages for control systems.* Preprints of the 3 IFAC/IFIP International symposium CADCE '85, Copenhagen, Denmark, 71-76.

Åström, K.J. (1985a), *A Simnon tutorial.* Department of Automatic Control, Lund Institute of Technology, CODEN: LUTFD2/(TFRT-3168)/ (1982), Sweden.

Åström, K.J. (1985b), "Computer aided tools for control system design", in M.J.Jamshidi and C.J. Herget (eds.), *Computer-aided control system engineering*, North-Holland, Amsterdam, Netherlands, 3-40.

Atherton, D.P. (1986), *Simulation in control system design.* Preprints of the IFAC/IMACS International symposium on simulation of control systems, Wien, Austria, 53-59.

Baker, N.J.C., P.J. Smart (1983), "The SYSMOD simulation language". *Proceedings of the* 2st *European simulation congress*, Aachen, W. Germany, 281-286.

Baker, N.J.C., P.J. Smart (1985), "Elements of the SYSMOD simulation language". *Proceedings of the* 11th *IMACS world congress*, Oslo, Norwege, vol.2.

Barker, H.A., P. Townsend, M. Chen, J. Harvey (1988b), "CES - a workstation environment for computer aided design in control systems". *Preprints of the* 4th *IFAC Symposium on Computer aided design in control systems CADCS '88*, China, Beijing, 248-251.

Bausch - Gall, I. (1987), "Continuous system simulation languages". *Veh. Syst. Dyn. (Netherlands)*, vol.16, 347-366.

Bekey, G.A. and W.J. Karplus (1968). *" Hybrid Computation"*, John Wiley, New York, USA.

Blum, J.J. (1969), *Introduction to Analog Computation*, Harcourt, brace & world, inc., New York, USA.

Boullard, L., L.D. Coen (1985), "A multi - microprocessor system for parallel computing in simulation". *Proceedings of the* 11th *IMACS world congress*, Oslo, Norwege, vol.3, 163-166.

Breitenecker, F., Solar, D. (1986), "Models, methods and experiments - modern aspects of simulation languages". *Proceedings of the* 2nd *European simulation congress*, Antwerpen, Belgium, 195-199.

Bosch, P.P.J. Van den, A.J.W. Van den Boom (1985), "Industrial applications in the Netherlands of computer - aided design packages for control systems; trends and practice". *Preprints of the* 3rd *IFAC/IFIP International symposium CADCE '85*, Copenhagen, Denmark, 58-61.

Bosch, P.P.J. Van den (1987), *Simulation program PSI (manual).*Delft university of technology, Netherlands.

Breitenecker, F. (1989). "Need for Hybrid Simulation?", *Proceedings of the* 3rd *European Simulation Congress*, (D. Murray - Smith, J. Stephenson and R.N. Zobel Eds.), Edinburgh, 421-425.

Britt, J.I, J. S. Wareck, J.A. Smith (1991), "A computer - aided process synthesis and analysis environment".In J.J. Siirola, I.E. Grossmann, G. Stephanopoulos (Eds.), *Foundations of Computer - Aided Process Design*, Elsevier, Amsterdam, 381-307

Bruijn, M. A. de, F.P.J. Soppers (1986), "The Delft parallel processor and software for continuous system simulation". *Proceedings of the* 2nd *European simulation congress*, Antwerpen, Belgium, 777-783.

Butcher, J.C. (1964), "Implicit Runge-Kutta processes", *Math. Comp. 18*, 50.

Carlson A., G. Hannauer, T. Carey , P.J. Holsberg (1967), *Handbook of Analog Computation*. Electronic Associates, Inc. Princeton, N.J., USA.

Cellier, F.E. (1979), *Combined continuous / discrete system simulation by use of digital computers, Techniques and tools*. Ph.D. Swiss Federal Institute of Technology, Zürich, Switzerland.

Cellier, F.E. (1983), "Simulation software - today and tomorrow." *Proceedings of the IMACS conference*, Nantes, France.

Cellier, F.E. (1991), *Continuous System Modeling*, Springer - Verlag, New York, 1991, USA.

Chen, S. (1989), *Toward the future. In Supercomputers: directions and technology and applications*, National Academy Press, Washington D.C.,USA, 51-65

Colijn, A.W., P.D. Ariel (1986), "Continuous system simulation languages on supercomputers". *Proceedings of the 1986 Summer Computer Simulation Conference*, Reno, USA, 3-7.

Crosbie, R.E., F.E. Cellier (1982), "Progres in simulation language standards"; *An activity report for Technical Committee TC3 on simulation software. Proceedings of the* 10^{th} *IMACS world congress*, Montreal, Canada, vol.1. 411-412.

Crosbie, R.E.(1984), "Simulation on microcomputer - experiences with ISIM". *Proceedings of the 1984 Summer Computer Simulation Conference*, Boston, Massachusetts, USA, vol.1, 13-17.

Crosbie, R.E., S. Javey, J.S. Hay, J.G. Pearce (1985a), "ESL - a new continuous system simulation language". *Simulation* , vol.44, no.5, 242-246.

Crosbie, R.E. (1986), "Developments in ESL and other's for the 1990's". *Proceedings of the 1986 Summer Computer Simulation Conference*, USA, 313-314.

Cser J., P.M. Brujn, Verbruggen (1986), "Music - a tool for simulation and real - time control". 4^{th} *IFAC/IFIP symposium on the software for computer control, SOCOCO'86*, Graz, Austria, 58-62.

Delebeque, F. and S. Steer (1986), "Some remarks about the design of an interactive CACSD package: The BLAISE experience", in K.L.Lineberry (ed.),*Proc. IEEE* 3^{rd} *Symp. on computer aided control system design*, New York, USA, 48-51.

Hindmarsh, A. C. (1983), "ODEPACK, a systematized collection of ODE solvers", *IMACS Transactions on Scientific Computing*, 1, 55-64.

D'Hollander, E. H. (1985), "A multiprocessor for dynamic simulation". *Proceedings of the 1985 Summer Computer Simulation Conference*, USA, 112-117.

Divjak, S. (1975), *Synthesis of time optimal digital simulation system for continuous dynamical systems*. Ph.D. Faculty of electrical engineering, University of Ljubljana, Yugoslavia.

Duncan, R. (1990), "A survey of parallel computer architectures". *IEEE Computer*, February 1990, 5-16.

EAI Handbook of Analog Computation, (A Carlson, G. Hannauer, T. Carey and P.J. Holsburg Eds.)(1967), Electronic Associates, Inc., West Long Branch, New Jersey, USA.

EAI-580 Analog-Hybrid Computing System - Reference Handbook(1968), West Long Branch, New Jersey, USA.

EAI-2000 Analog Reference Handbook(1982), West Long Branch, New Jersey, USA.

Eckelmann, P. (1987), "The transputer - component for the 5 generation systems". *Proceedings VLSI and computers. First international conference on computer technology, systems and applications*, Hamburg, West Germany, 977.

Elmqvist, H. (1975), *SIMNON, an interactive simulation program for nonlinear systems*. Users manual. Departement of Automatic Control, Lund Institute of Technology, Sweden, Report 7502.

Elmqvist, H. (1977), "SIMNON: an interactive simulation program for nonlinear system". *Proceedings Simulation'77*, Montreux, France.

Elmqvist, H. (1978), *A structured model language for large continuous systems*, Ph. D. diss. Rep. CODEN: LUTFD2/(TFRF-1015), Departement of Automatic Control, Lund Institute of Technology, Sweden.

Elmqvist, H., S.E. Matson (1986), "A simulator for dynamical systems using graphics and equations for modeling". *Proceedings of the IEEE Control System Society. Third symposium on computer aided control system design*, Arlington, USA, 134-139.

Elmqvist, H. (1994), *Dymola - Dynamic Modeling Language*, User's Manual, Dynasim AB, Lund, Sweden.

Fadden E.J (1987), "SYSTEM 100: Time - critical simulation of continuous systems". *Multiprocessor and Array Processor Conference, publishing number 87 02mT15*, San Diego, USA.

Fischlin, A., M. Mansour, M. Rimvall, W. Schaufelberger, (1986), "Simulation and computer aided control system design in engineering education", *IFAC/IMACS International symposium on simulation of control systems*, Wien, Austria, 61-73.

Fishman, G.S. (1973), *"Concepts and methods in discrete event digital simulation"*, John Wiley, New York, USA.

Fritzson P. (2004), *Principles of Object Oriented Modeling and Simulation with Modelica 2.1*, IEEE Press, John Wiley&Sons Inc., Publication, USA.

Gauthier, J.E. (1987), "ACSL and simulators". *Proceedings of the 1987 Summer Computer Simulation Conference*, Orlando, USA, 73-77.

Gear, C.W. (1971), *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall.

Gear, C.W. (1984), "Efficient step size control for output and discontinuities", *Trans. Soc. Comput. Sim. 1*, 27-31.

Gear, C.W. (1986), "The potential for parallelism in ordinary differential equations". Proc. Int. Conf. *Numerical Mathematics*. 33-48.

Gear, C.W., O. Osterby (1984), "Solving ordinary differential equations with discontinuities". *ACM Trans. Math. Software, 10*, 23-44.

Giloi, W.K. (1975), *Principles of Continuous System Simulation* B.G. Teubner, Stuttgart.

Goforth, R.R., R.M. Crisp (1988), "Simulation with microcomputers: an overview". *Acces, a journal of microcomputer applications*, feb. 1988, vol.7, no.1, 4-14.

Golden, D.G. (1985), "Software engineering considerations for the design of simulation languages". *Simulation*, vol.45, no.4, 169-178.

Grierson, W.O. (1986), "Perspectives in simulation hardware and software architecture". *Modeling, Identification and Control (norwegian research bulletin)*, vol.6 ,no. 4, 249-255.

Gustaffson, K. (1988), *Stepsize control in ODE-Solvers-Analysis and Synthesis*, Licentiate Thesis TFRT-3199, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden

Gustaffson, K. (1990), "Using control theory to improve stepsize selection in numerical integration of ODE", *Preprints of* 11^{th} *IFAC World Congress*, Tallin, Estonia, USSR, vol. 10, 139-144.

Hairer, E., C. Lubich (1988), "Extrapolation at stiff differential equations", *Numer. Math.*, Vol. 52, 377-400.

Hairer, E., G. Wanner (1990), *Solving Ordinary Differential Equations II. Stiff and Differential–Algebraic Problems* , Springer Verlag.

Hall, G., J.M. Watt (1976), *Modern numerical methods for ordinary differential equations*, Clarendon Press, Oxford.

Hallin, H.J., S.A.R. Hepner (1984), "Solving benchmark problems with PSCSP and other simulation languages". *Proceedings of the 1984 Summer Computer Simulation Conference*, Boston, Massachusetts, USA, vol.1, 3-8.

Hallin, H.J (1988), "Simulation im Zeitalter von Supercomputern und Mini-Supercomputern". *Proceedings 5. Symposium Simulationstechnik*, Aachen, W. Germany, 2-13.

Hamblen, J.O. (1987), "Parallel continuous system simulation using transputer". *Simulation, vol.49, no.6*, 249-253.

Havranek, W.A. (1983), "Simulation in the 80s". *Proceedings of the 1883 Summer Computer Simulation Conference*, Vencouver, Canada, vol.1, 523-528.

Hay, J.L., R.E. Crosbie (1981), *Outline proposal for a new standard for continuous system simulation language (CSSL 81)*. Computer Simulation Centre, Departement of Electrical Engineering, University of Salford, USA.

Hay, J.L., R.E. Crosbie (1984), "ISIM - a simulation language for microprocessors". *Simulation* , sep. 1984, USA, 133-136.

Hay, J.L., R.E. Crosbie (1986), "Parallel processor simulation with ESL". *Proceedings of the 1986 Summer Computer Simulation Conference*, Reno, USA, 959-964.

Heinrich, R. (1986), "Simulation in control engineering". *Proceedings of the* 2^{nd} *European simulation congress*, Antwerpen, Belgium, 10-13.

Herget, C.J. (1988), "Survey of existing computer-aided design in control systems packages in the United States of America". *Preprints of the* 4^{th} *IFAC Symposium on Computer aided design in control systems CADCS '88*, China, Beijing, 46-50.

Hooper, J.W. (1987), "Strategy- related characteristics of discrete event languages and models". *Simulation*, vol. 46, no. 4, 153-158.

Huber, R.M., A. Guasch (1985), "Towards a specification of a structure for continuous system simulation languages". *Proceedings of the* 11^{th} *IMACS world congress*, Oslo, Norwege, 109-113.

Jackson, A.S. (1960), *Analog Computation*, McGraw Hill, London.

Karba, R., B. Zupančič, F. Bremšak, A. Mrhar and S. Primožič (1990), "Simulation tools in pharmacokinetic modelling", *Acta Pharm. Jugosl.*, Vol. 40, 247-262.

Karplus, W.J. (1984a), "Selection criteria and performance evaluation methods for peripheral array processors". *Simulation*, vol.43, no.3, 125-131.

Karplus, W.J. (1984b), "The changing role of peripheral array processors in continuous systems simulation", *Proceedings of the 1984 Summer Computer Simulation Conference*, Boston, Massachusetts, USA, vol.1, 1-13.

Kettenis, D.L. (1986), "COSMOS: a member of a new generation simulation languages". *Proceedings of the* 2^{nd} *European simulation congress*, Antwerpen, Belgium, 263-269.

Kleinert, W., D. Solar, F. Berger (1983), "Status report on TU Vienna's hybrid time sharing system". *Proceedings of the* 2^{st} *European simulation congress*, Aachen, W. Germany, 193-200.

Kleinert, W., M. Graff, R. Karba, B. Zupančič (1988), "Simulation einer Destillationskolonne - Modellierung mit SIMCOS und Vergleich der Ergebnisse von ACSL und SIMSTAR Simulationen". *Proceedings of the* 5^{th} *symposium Simulationstechnik*, Aachen, W. Germany, 254-259.

Korn, G.A. (1983a), "A wish for simulation - language specifications". *Simulation* ,jan. 1983, USA.

Korn, G.A. (1983b), "Direct - executing languages for interactive simulation and computer - aided experiments." *Proceedings of the* 2^{nd} *European simulation congress*, Aachen, W. Germany, 225-233.

Koskossidis, D.A. and C.J. Brennan (1984), "A Review of Simulation Techniques and Modelling", *Proceedings of the 1984 Summer Computer Simulation Conference (W.D. Wade ed.)*, Vol.1, Boston, USA, 55-58.

Landauer, J, P. (1988), *EAI STARTRAN environment.* Users manual, Electronic Associates, Inc., West Long Branch, New Jersey, USA.

Landauer, J.P. (1988b), "Real-time simulation of the Space Shuttle main engine on the SIMSTAR multiprocessor". *Proceedings of the SCS Multiconference on Aerospace Simulation III*, San Diego, USA.

Lapidus, L., J.H. Seinfeld (1971), *Numerical Solution of Ordinary Differential Equations*, Academic Press New York and London.

Lincoln, A. (1988), A modular parallel computer system for high perfomance real-time simulation. *Proceedings of the* 12th *IMACS world congress*, Paris, France, vol.2., 619-621.

Marquardt, W. (1991), "Dynamic process simulation - recent progress and future challenges." *Preprints of CPC IV, Forth International Conference on Chemical Process Control*, South Padre Island, Texas, USA.

Matko, D., B. Zupančič, S. Divjak (1989), "New concepts in control system simulation". *Proceeding of the* 3rd *European Simulaton Congress*, Edinburgh, Scotland, 444-446.

$PC - MATLAB^{TM}$ for MS-DOS personal computers, The MathWorks Inc. SouthNatick, USA (1989).

Mc Leod, J.P.E. (1986a), "Computer modelling and simulation: the changing chalenge." *Simulation*, vol.46, no.3, 114-118.

Mc Leod, J.P.E. (1987b), "What is simulation?" (Simulation in the service of society). *Simulation*, 219-221.

Mitchel & Gauthier, Assoc. (1981), *ACSL: advanced continuous simulation language.* (user guide / reference manual).

Myers, W. (1990), "Parallel programming: views from the trenches." *IEEE Software, jan. 1990.*

Nilsen, N.R. (1984), *The CSSL IV simulation language (reference manual).* Simulation Service, Chatsworth, California, USA.

Nilsen, N.R. (1985), "Recent advances in CSSL IV." *Proceedings of the* 11th *IMACS world congress*, Oslo, Norwege, vol.3, 101-103.

Ören, T.I. (1974),"A bibliography of bibliographies on modelling, simulation and programing", *Simulation*, Vol.23, No.3, 90-95 and Vol.23, No.4, 115-116.

Ören, T.I., B.P. Ziegler (1979), "Concepts for advanced simulation methodologies." *Simulation*, vol.32, no.3.

Pearce, J.G., P. Holliday, J.O. Gray (1985), "Survey of parallel processing in simulation." *Proceedings of the* 2nd *European workshop on parallel processing techniques for simulation*, Manchester, United Kingdom, 183-202.

Press, W.H., B.P., Flannery, S.A.,Teukolsky, W.T. Vetterling (1986), *"Numerical recipes - The Art of Scientific Computing"*, Cambridge University Press, UK.

Pritsker, A.A.B (1979), "Comparisons of definitions of simulation." *Simulation* ,vol.33, no.2, 61-63.

Rimvall, M., F.E. Cellier (1985), "The matrix environment as enhancement to modelling and simulation." *Proceedings of the* 11th *IMACS world congress*, Oslo, Norwege, vol.3., 93-96.

Rimvall, M., F.E. Cellier (1986), "Evaluation and perspectives of simulation languages following the CSSL standard." *Modeling, Identification and Control (norwegian research bulletin), vol.6 ,no. 4*, 181-199.

Saucedo R., Schirring E. E. (1986), *Introduction to Continuous and Digital Control Systems*, Mcmillan Applied System Science, New York, USA.

Schmid, C. (1988), "Tehniques and tools of CACDS." *Preprints of the* 4^{th} *IFAC Symposium on Computer aided design in control systems CACDS '88*, China, Beijing, 67-75.

Schmidt, G. (1980), *Simulationstechnik*. R. Oldenbourg, München, W. Germany.

Schmidt, B. (1986), "Classification of simulation software." *Syst. Anal. Model. Simul. (Benelux journal), 3 (1986) 2*, 133-140.

Schmidt, A., F. Scheider (1988), "Erfahrungen mit Hardware in-the-loop Simulation an der Workstation XANALOG XA-1000". *Proceedings of the* 5th *symposium Simulationstechnik*, Aachen, W. Germany, 2-13.

Schrage, M.H., D.F. Mc Ardle (1986), " New array processor continuous simulation system uses tactile sensing icon programming." *Proceedings of the 1986 Summer Computer Simulation Conference*, Reno, USA, 138-142.

Schumann A., D. Matko and B. Zupančič (1991), "Simulation of a diesel engine using a digital simulation language," MELECON '91, Ljubljana, Slovenia.

Shah, M., J. (1988), *Engineering simulation: tools and applications using the IBM PC family.* Prentice Hall, Englewood Cliffs, New Jersey, USA.

Shneiderman, B. (1987), *Designing the user interface.* Addison - Wesley Publishing Company, USA.

Simulink, Dynamic System Simulation Software (2009), Users manual, R2009b, Math Works, Inc., Natick, MA, ZDA.

Smith, J.M. (1977), *Mathematical modeling and digital simulation for engineers and scientists.* John Wiley & Sons, Inc.

Smith, D.J.M (1995), *Continuous System Simulation*, Chapman & Hall, London, UK.

Sodja, A, B. Zupančič (2009), *Modelling thermal processes in buildings using an object-oriented approach and Modelica.* Simulation Modelling Practice and Theory, vol. 17, no. 6, str. 1143-1159.

Spriet, J.A. and G.C. Vansteenkiste (1982), *Computer Aided Modelling and Simulation*, Academic Press, London, U.K.

Steppard, S. (1983), "Applying software engineering to simulation." *Simulation*, 13-19.

Strauss, J.C. (1967), "The SCi continuous system simulation language." *Simulation*, no.9, 281-303.

Syn, W.M., H. Dost (1985), "On the dynamic simulation language (DSL/VS) and its use in the IBM corporation." *Proceedings of the* 11$^{\text{th}}$ *IMACS world congress*, Oslo, Norwege, vol.3, 115-118.

Šega, M., S. Strmčnik, R.Karba and D.Matko (1985), "Interactive program package ANA for system analysis and control design," *Prep.* 3$^{\text{rd}}$ *IFAC int. Symp. CADCE'85*, Copenhagen, 95-98.

Taylor, H., D.K. Friderick, C.M. Rimvall, H.A. Sutherland (1990), "Computer - aided control engineering environments: architecture, user interface, data - base management, and expert aiding." *Preprints of* 11$^{\text{th}}$ *IFAC World Congress*, Tallinn, USSR, vol. 10, 55-65.

Terrel, J.T. (1988), *Introduction to digital filters.* Macmillan Education, Houndmills.

Tesler, L.G. (1989), "Achieving a pioneering outlook with supercomputing." *Supercomputers: directions in technology and applications*, National Academy Press, Washington D.C., USA, 90-95.

Tomovic, R. and W.J. Karplus (1962), *High Speed Analog Computers*, John Wiley, New York, USA.

TUTSIM user's manual for IBM PC computers (1983). Twente University of Technology, Netherlands.

Tyso, A. (1985), "Simulation as a tool in operational safety, reliability and control." *Modeling, Identification and Control* (norwegian research bulletin), vol.6 ,no. 3, 127-140.

Worlton, J. (1989), *Existing conditions. In Supercomputers: directions in technology and applications*, National Academy Press, Washington D.C., 21-50

Ziegler, B.P. (1976), *Theory of modelling and simulation.* John Wiley & Sons, Inc., New York, USA.

Ziegler, B.P. (1987), *Simulation objectives: experimantal frames and validity. System & Control Encyklopedia*, Pergamon Press, 4388-4392.

Zupančič, B., D. Matko, M. Šega, P. Tramte (1986), "Simulation in the program package ANA." *Proceedings of the* 2nd *European simulation congress*, Antwerpen, Belgium, 314-318.

Zupančič, B., D. Matko, R. Karba, M. Atanasijević, Z. Šehić (1991), "Extensions of the simulation language SIMCOS towards continuous - discrete complex experimentation system." *Preprints of the IFAC Symposium CADCS '91* , University of Wales, Swansea, U.K., 351-356.

Paynter H.M. (1961), *Analysis and design of engineering systems.* The M.I.T. Press, Cambridge.

Borutsky, W. (2010), *Bond graph methodology, development and analysis of multidisciplinary dynamic system models.* Springer.