The Behavioral ToolBox

Documentation and Analysis of BTB

W.Bijlsma
J.W. van Dijk

Contents

1 Modeling by tearing and zooming 4

2 Systems Theory and The Behavioral Toolbox 10
- 2.1 Differential systems . . . 12
- 2.2 The elimination problem . . . 13
- 2.3 Observability and Controllability . . . 15
  - 2.3.1 Observability . . . 15
  - 2.3.2 Controllability . . . 16
- 2.4 Inputs, Outputs and States . . . 19
  - 2.4.1 Inputs and Outputs . . . 19
  - 2.4.2 State Representations . . . 21
- 2.5 Simulation . . . 23
- 2.6 Things for future releases of BTB . . . 27

3 Documentation 30
- 3.1 System Requirements . . . 30
- 3.2 Installation and getting started . . . 30
- 3.3 What is BTB . . . 31
- 3.4 Modeling a system . . . 32
  - 3.4.1 Create a new type of terminal . . . 34
  - 3.4.2 Create a connection . . . 34
  - 3.4.3 Create a module class . . . 35
  - 3.4.4 Insert a module in the system . . . 38
  - 3.4.5 Connect two terminals . . . 39
  - 3.4.6 Disconnect two terminals . . . 39
  - 3.4.7 Remove a module from your system . . . 40
- 3.5 Information and corrections . . . 40
  - 3.5.1 Information of the module . . . 41
  - 3.5.2 Information of the connection constraints . . . 41
  - 3.5.3 Change a module . . . 42
  - 3.5.4 Change a connection . . . 43

3.6 Behavior 44
3.6.1 Change the values of the parameters 44
3.6.2 Manifest / Latent assignment of the variables on the terminals 46
3.6.3 Show Behavior of the system 47
3.6.4 Properties of the system 54
3.7 Re-Usability 54
3.8 Simulation of the external behavior 55

4 The program **59**
4.1 Program analysis. 59
4.1.1 Specification of the program. 59
4.1.2 Description of the structure. 60
4.2 Program design. 61
4.2.1 Building an interconnected system. 61
4.2.2 Behavior of the interconnected system. 65
4.2.3 Simulation. 69

A OOP and the Universal Modeling Language **70**
A.1 Definitions 70
A.2 UML notation 72

Chapter 1

Modeling by tearing and zooming

A typical system consists of an interconnection of subsystems. The very first picture that comes to mind when thinking about a model is that of a *black box* with a number of *terminals*, say T. Terminals correspond physically to the way in which the subsystem is interfaced with its environment.

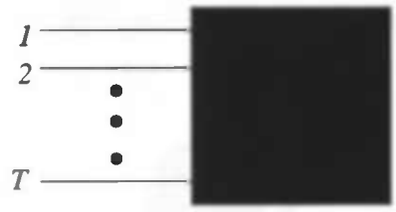

Figure 1.1: Black box

This can be formalized mathematically, by associating to the i-th terminal a variable s_i which takes its value in a set $\mathbb{S}_i$. The set of all *terminal variables* $\mathbb{S} = \mathbb{S}_1 \times \mathbb{S}_2 \times \ldots \times \mathbb{S}_T$ we can think of as the system's *terminal signal space*.

We are mainly interested in dynamical systems, in which case the terminal variable s evolves as a function of independent variables, such as time and space, which we indicate by a set $\mathbb{I}$, the so called *indexing set* of our model. So s is a function from $\mathbb{I}$ to $\mathbb{S}$. A priori all functions $s : \mathbb{I} \to \mathbb{S}$, denoted by $\mathbb{S}^{\mathbb{I}}$, can occur at the terminals. We can think of $\mathbb{S}^{\mathbb{I}}$ as the *universum* of our model, the set of all possible outcomes. The internal laws of the system restricts this set to a set $\mathfrak{B} \subset \mathbb{S}^{\mathbb{I}}$. This set $\mathfrak{B}$ represents the set of signals that actually can occur and is called the *terminal behavior* or *full behavior*.

An interconnected system is viewed as a collection of *modules* with *termi-*

nals, interconnected through an *interconnection architecture*. These modules, the building blocks of an interconnected system, are subsystems with terminals. One cannot connect terminals that are not of *adapted type*, i.e. that are not connectable. For instance, let us consider a physical system. In this case, connectability simply means that the terminals have to possess the same physical nature (e.g. you cannot connect an outlet of a tank to an electrical wire). But we do not have to constrict ourselves to physical systems! See also [4].

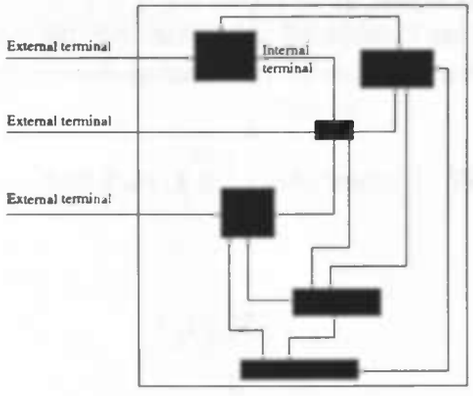

Figure 1.2: Interconnected system

A module belongs to a particular *module class*, which is defined by the number of terminals, the type of each of its terminals, its behavioral representation and by parameters. The architecture imposes relations between the variables at these terminals. Examples of modules classes:

module class	**Terminals**	**Type of terminals**
resistor	(terminal1, terminal2)	(electrical, electrical)
transistor	(collector, emitter, base)	(electrical, idem, idem)
2-inlet vessel	(inlet1, inlet2)	(fluidic, fluidic)

Examples of terminals:

Type of terminal	**Variables**	**Universum**
electrical	(voltage, current)	$\mathbb{R} \times \mathbb{R}$
1-D mechanical	(force, position)	$\mathbb{R} \times \mathbb{R}$
2-D mechanical	(position, attitude, force, torque)	$\mathbb{R}^2 \times \mathbb{S}^1 \times \mathbb{R}^2 \times \mathbb{R}$
thermal	(temperature, heat flow)	$\mathbb{R}_+ \times \mathbb{R}$

After interconnection, the architecture leaves some terminals available for interaction with the environment of the overall system. These terminals are the so

called *external* terminals. The connected terminals are called *internal* terminals.

A specific module is defined by giving its module class and the numerical values of the parameters; the full behavior of both internal and external terminals is then also defined. However, only some of these variables are of interest for the specific application at hand. We call these variables the *manifest* variables taking values in a signal space $\mathbb{W}$. The remaining variables are called the *latent* variables taking values in another signal space $\mathbb{L}$ (see e.g. [5], [1]). Usually, the variables on the *external terminals* are considered to be manifest; the variables on the *internal terminals* are considered to be latent. Now define the manifest behavior

$$\mathfrak{B} = \{w \in \mathbb{W} \mid \text{there exists } l \in \mathbb{L} \text{ such that} (w, l) \in \mathfrak{B}_{\text{full}}\}$$

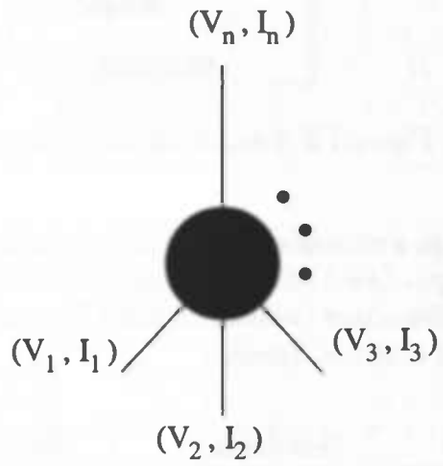

Figure 1.3: Electrical module

For instance, consider an electrical module, shown in figure 1.3. The interaction with the environment takes place through its wires. So the wires are the terminals. With each terminal we associate two real variables, the potential V and the current I (agreed to be positive when electrical current flows into the device). The laws of the device specify the behavior, which is a subset $\mathfrak{B}$ of $\mathbb{S}^{\mathbb{R}}$ where $\mathbb{S} = (\mathbb{R}^2)^n$ is the signal space and n denotes the number of terminals.

Pairing of terminals by the interconnection architecture implies an *interconnection* law. Examples:

Pair of terminal	Variables terminal 1	Variables terminal 2	Interconnection constraints
electrical	(V_1, I_1)	(v_2, I_2)	$V_1 = V_2, I_1 + I_2 = 0$
1-D mechanical	(F_1, q_1)	(F_2, q_2)	$F_1 + F_2 = 0, q_1 = q_2$
thermal	(Q_1, T_1)	(Q_2, T_2)	$Q_1 + Q_2 = 0, T_1 = T_2$
fluidic	(p_1, f_1)	(p_2, f_2)	$p_1 = p_2, f_1 + f_2 = 0$
information processing	m-input u	m-output y	$u = y$

Example: consider the electrical circuit shown in figures 1.4 and 1.5.

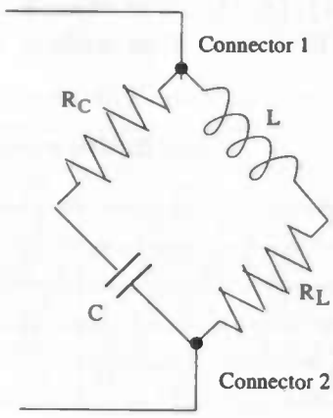

Figure 1.4: RLC circuit

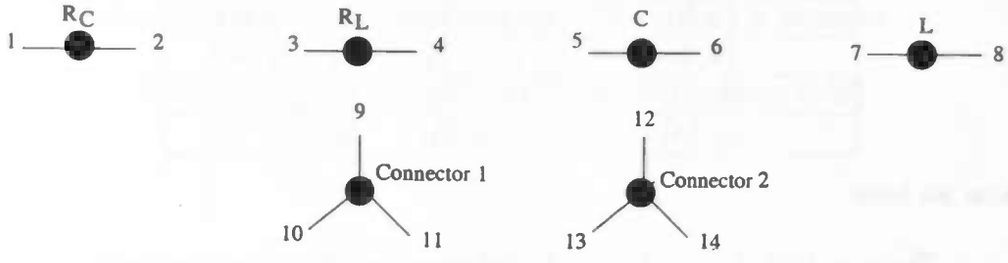

Figure 1.5: RLC circuit after tearing

This circuit consists of the following modules:

Module	From module class	Terminals	Parameter
R_C	resistor	(1,2)	R in ohms
R_L	resistor	(3,4)	R in ohms
C	capacitor	(5,6)	C in farad
L	inductor	(7,8)	L in henry
Connector1	3-terminal connector	(9,10,11)	
Connector2	3-terminal connector	(12,13,14)	

The interconnection architecture consists of the pairing of the terminals: $\{10, 1\}$, $\{11, 7\}, \{2, 5\}, \{8, 3\}, \{6, 13\}, \{4, 14\}$. The external terminals are $\{9, 12\}$. The other terminals are internal terminals. This leads to the following equations for the full behavior:

Modules	Constitutive equations	
R_C	$I_1 + I_2 = 0$	$V_1 - V_2 = R_C I_1$
R_L	$I_3 + I_4 = 0$	$V_3 - V_4 = R_L I_3$
C	$I_5 + I_6 = 0$	$C\frac{d}{dt}(V_5 - V_6) = I_5$
L	$I_7 + I_8 = 0$	$V_7 - V_8 = L\frac{d}{dt} I_7$
Connector1	$I_9 + I_{10} + I_{11} = 0$	$V_9 = V_{10} = V_{11}$
Connector2	$I_{12} + I_{13} + I_{14} = 0$	$V_{12} = V_{13} = V_{14}$

Interconnection pair	Interconnection equations	
$\{10, 1\}$	$V_{10} = V_1$	$I_{10} + I_1 = 0$
$\{11, 7\}$	$V_{11} = V_7$	$I_{11} + I_7 = 0$
$\{2, 5\}$	$V_2 = V_5$	$I_2 + I_5 = 0$
$\{8, 3\}$	$V_8 = V_3$	$I_8 + I_3 = 0$
$\{6, 13\}$	$V_6 = V_{13}$	$I_6 + I_{13} = 0$
$\{4, 14\}$	$V_4 = V_{14}$	$I_4 + I_{14} = 0$

Now we have

$$\mathfrak{B}_{\text{full}} = \{(V_1, I_1, \ldots, V_{14}, I_{14}\} \mid \text{the above equations are satisfied}\}.$$

Suppose that the variables corresponding to the external terminals 9 and 12 are the manifest variables. Denote $w = (V_9, I_9, V_{12}, I_{12})$ and
$l = (V_1, I_1, \ldots, V_8, I_8, V_{10}, I_{10}, V_{11}, I_{11}, V_{13}, I_{13}, V_{14}, I_{14})$. This yields

$$\mathfrak{B} = \{w \in \mathbb{R}^4 \mid \text{there exists } l \in \mathbb{R}^{24} \text{ such that } (w, l) \in \mathfrak{B}_{\text{full}}\}$$

Example with an higher dimensional index set:

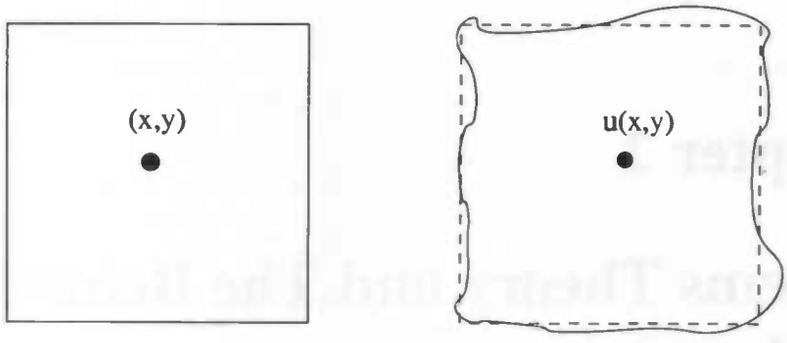

Figure 1.6: Elastic membrane

The interaction of this module with its environment takes place through the whole module. So it has one terminal. Indicate by $u(x,y,t) = (u_x(x,y,t), u_y(x,y,t))$ the displacement at time t of a point at position (x,y) when the membrane is not deformed. Denote by F the external force applied to the point of the membrane in position (x,y) at time t. The system is regarded as a distributed mechanical terminal with terminal variables (u_x, u_y, F_x, F_y). The equation of motion is

$$\rho \frac{\partial^2}{\partial t^2} u = (\lambda + \mu)\nabla(\nabla \bullet u) + \mu \nabla^2 u + F$$

with parameters ρ, the mass density, and constants λ and μ (lamé constants), describing the elastic behavior of the membrane. This yields the behavior

$$\mathfrak{B} = \{(u_x, u_y, F_x, F_y) : \mathbb{R}^3 \to \mathbb{R}^4 \mid \text{above equations hold}\}.$$

Chapter 2

Systems Theory and The Behavioral Toolbox

In this chapter we introduce the essential mathematical concepts that are involved with modeling systems as done in the previous chapter. We only concentrate on systems whose trajectories can be described as solutions (most generally in the sense of distributions) to a set of linear (partial) differential equations.

Definition 2.0.1 *A dynamical system is a triple* $\Sigma = (\mathbb{I}, \mathbb{W}, \mathfrak{B})$ *with* $\mathbb{W}$ *the signal space,* $\mathbb{I}$ *the index set and* $\mathfrak{B} \subset \mathbb{W}^{\mathbb{I}}$ *the behavior of the system, i.e. the set of all trajectories allowed by the system.*

There are two crucial aspects concerning this definition. First of all, it abandons the idea of a system as input/output map (signal processor) and all the quantities for which no hierarchical or cause/effect structure is a priori given. Secondly, there is a clear distinction between the feasible trajectories and its representation (equations, graphs, grammar rules, ...).
Classical notions such as *linearity* and *shift-invariance* are at this abstract level already defined. A dynamical system is called *linear* if $\mathbb{W}$ is a vector space and $\mathfrak{B}$ is a linear subspace of $\mathbb{W}^{\mathbb{I}}$, and is called *shift-invariant* if $\mathbb{I}$ is a semigroup under addition and $\sigma^t\mathfrak{B} = \mathfrak{B}$ for all $t \in \mathbb{I}$, where σ^t is the shift operator defined by $(\sigma^t w)(x) = w(x + t)$.

Example: *Maxwell's equations* describe the possible realizations of the fields $\vec{E} : \mathbb{R} \times \mathbb{R}^3 \to \mathbb{R}^3$, $\vec{B} : \mathbb{R} \times \mathbb{R}^3 \to \mathbb{R}^3$, $\vec{j} : \mathbb{R} \times \mathbb{R}^3 \to \mathbb{R}^3$ and $\rho : \mathbb{R} \times \mathbb{R}^3 \to \mathbb{R}$. These equations are

$$\begin{aligned} \nabla \bullet \vec{E} &= \frac{1}{\varepsilon_0}\rho \\ \nabla \times \vec{E} &= -\frac{\partial}{\partial t}\vec{B} \\ \nabla \bullet \vec{B} &= 0 \\ c^2 \nabla \times \vec{B} &= \frac{1}{\varepsilon_0}\vec{j} + \frac{\partial}{\partial t}\vec{E} \end{aligned}$$

with ε_0 the *dielectric* constant of the medium and c the speed of light in the medium. They define a dynamical system $\Sigma = (\mathbb{R} \times \mathbb{R}^3, \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}, \mathfrak{B})$ with behavior $\mathfrak{B}$ the set of all fields $(\vec{E}, \vec{B}, \vec{j}, \rho) : \mathbb{R} \times \mathbb{R}^3 \to \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}$ that satisfy Maxwell's equations.

As seen in chapter 1, also systems with latent variables should be considered:

Definition 2.0.2 *A dynamical system with latent variables is a quadruple* $\Sigma_L = (\mathbb{I}, \mathbb{W}, \mathbb{L}, \mathfrak{B}_{\text{full}})$, *with* $\mathbb{I}$ *the index set,* $\mathbb{W}$ *the manifest signal space,* $\mathbb{L}$ *the latent signal space and* $\mathfrak{B}_{\text{full}} \subset \mathbb{W}^{\mathbb{I}} \times \mathbb{L}^{\mathbb{I}}$ *the full behavior of the system*

Note that if we write $\widetilde{\mathbb{W}} = \mathbb{W} \times \mathbb{L}$, then $\Sigma_L = (\mathbb{I}, \widetilde{\mathbb{W}}, \mathfrak{B}_{\text{full}})$ is a dynamical system as in definition 2.0.1. A dynamical system with latent variables induces a dynamical system in the sense of definition 2.0.1 as follows:

Definition 2.0.3 *Let* $\Sigma_L = (\mathbb{I}, \mathbb{W}, \mathbb{L}, \mathfrak{B}_{\text{full}})$ *be a dynamical system with latent variables. The manifest dynamical system induced by* Σ_L *is the dynamical system* $\Sigma = (\mathbb{I}, \mathbb{W}, \mathfrak{B})$, *with behavior* $\mathfrak{B}$ *defined as:*

$$\mathfrak{B} = \{w : \mathbb{I} \to \mathbb{W} \mid \text{there exist } l \in \mathbb{L} \text{ such that } (w, l) \in \mathfrak{B}_{\text{full}}\}$$

2.1 Differential systems

What we are really after in modeling a system, is its behavior $\mathfrak{B}$, the set of admissible trajectories. Although a behavior can be described in various ways, there is one class of systems that plays a prominent role: the *differential systems*. A differential system is defined, with $\mathbb{I} = \mathbb{R}^n$ and $\mathbb{W} = \mathbb{R}^w$, as a system whose behavior $\mathfrak{B}$ is described by all solutions of a finite set of partial differential equations of the form:

$$f(x, w(x), \ldots, (\frac{\partial}{\partial x})^k w(x)) = 0$$

Here we use the multi-index notation $(\frac{\partial}{\partial x})^k = \frac{\partial^{(k_1+\cdots+k_n)}}{\partial x_1^{k_1} \cdots \partial x_n^{k_n}}$ with $k = (k_1, \ldots, k_n)$. We also assume that only a finite number of such partial derivatives are involved. This class of differential systems has an important subclass, the class of *differential systems with constant coefficients*. These are systems whose behavior consists of all solutions of equations of the form:

$$R(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})w = 0$$

where $R \in \mathbb{R}^{\bullet \times w}[\xi_1, \ldots, \xi_n]$ is a polynomial matrix in n indeterminates with w columns and an arbitrary (finite) number of rows. For obvious reasons, $\mathfrak{B} = \ker(R(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}))$ is called a *kernel representation* with kernel R.
Of course, one has to specify what kind of solutions one is looking for. In the following, we are only concerned with smooth solutions or solutions in the sense of distributions. The set of distributions with respect to a test-function space $\mathfrak{D}(\mathbb{R}^n, \mathbb{R}^m)$ is denoted by $\mathfrak{D}'(\mathbb{R}^n, \mathbb{R}^m)$.

First we have to introduce some notation. A module $\mathfrak{X} \subset \mathbb{R}^m[\xi_1, \ldots, \xi_n]$, generated by a finite number of elements $x_1, \ldots, x_n \in \mathfrak{X}$ (a *noetherian module*), is denoted as $\mathfrak{X} =< x_1, \ldots, x_n >$ or by $\mathfrak{X} =< X >$ with $X = \mathrm{mat}(x_1, \ldots, x_n)$ the matrix which has $x_1, \ldots, x_n$ as its columns.

A fundamental question is: suppose we have two kernel representations, when do they define the same behavior? This is answered in the following theorem:

Theorem 2.1.1 *Let* $R_1, R_2 \in \mathbb{R}^{\bullet \times w}[\xi_1, \ldots, \xi_n]$. *Then* $\ker(R_1(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})) = \ker(R_2(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})) \Leftrightarrow < R_1^T >=< R_2^T >$

This theorem establishes the correspondence between behaviors and submodules of $\mathbb{R}^w[\xi_1, \ldots, \xi_n]$. But this depends on the solution space under consideration:

it holds for $\mathfrak{C}^\infty$ or $\mathfrak{D}'$ solutions. It also reflects the fact that a behavior admits many representations. A consequence of the theorem is, since we can transform a polynomial matrix in one indeterminate in a matrix of full row rank and that according to the theorem the corresponding behavior will not change, that we can always find a full row rank kernel representation for a kernel behavior involving one indeterminate. This result was also derived in [5]. This is a so called *minimal representation*. For systems involving more than one indeterminate, it is more complicated. Example: suppose we have a behavior defined by $\frac{\partial}{\partial x}w_1 = 0$, $\frac{\partial}{\partial y}w_1 = 0$ with manifest variables w_1, w_2. Obviously, the corresponding kernel matrix doesn't have independent rows. But we can not replace these two equations by a single one and still maintain a kernel behavior. This example shows that things work really different for ND-systems.

In the Behavioral Toolbox:

A minimal kernel representation is calculated, with the use of the m-file of The Polynomial Toolbox *prowred.m*. So the result is in fact stronger: it is *row proper* or *row reduced*. This means for a polynomial matrix R that the matrix R_{hc}, which consists of the coefficients of the entries of R corresponding to the highest degree of each row, has full row rank. The m-file interfacing Matlab and BTB (short for Behavioral Toolbox) is *iprowred.m*.

2.2 The elimination problem

Suppose the behavior of a system, with manifest variables w and latent variables l, admits a kernel representation with kernel matrix $R \in \mathbb{R}^{\bullet\times(w+1)}[\xi_1, \ldots, \xi_n]$. In this case we have:

$$\mathfrak{B}_{\text{full}} = \{(w,l) : \mathbb{R}^n \to \mathbb{W} \times \mathbb{L} \mid R(\tfrac{\partial}{\partial x_1}, \ldots, \tfrac{\partial}{\partial x_n}) \begin{bmatrix} w \\ l \end{bmatrix} = 0\}$$
$$\mathfrak{B} = \{w : \mathbb{R}^n \to \mathbb{W} \mid \text{there exists } l \in \mathbb{L} \text{ such that } (w,l) \in \mathfrak{B}_{\text{full}}\}$$

A natural question arises: is there also a differential representation of the behavior which does not involve any latent variables? As we will see, such a representation exists.

Definition 2.2.1 *Let* $T \in \mathbb{R}^{p\times s}[\xi_1, \ldots, \xi_n]$. *The set of Sygygies of* T *is the set* SYZ$(T) = \{h \in \mathbb{R}^s[\xi_1, \ldots, \xi_n] \mid Th = 0\}$

It is easily seen that SYZ(T)$\subset$ $\mathbb{R}^s[\xi_1,\ldots,\xi_n]$ is a module over $\mathbb{R}[\xi_1,\ldots,\xi_n]$. Therefore, it is also referred to as *Sygygy module* of T. Since $\mathbb{R}^s[\xi_1,\ldots,\xi_n]$ is a noetherian module and SYZ(T)$\subset$ $\mathbb{R}^s[\xi_1,\ldots,\xi_n]$, there exist a polynomial matrix H such that $< H >=$SYG(T). Now we state the following result:

Theorem 2.2.2 *Let $\mathfrak{B}$ be the manifest behavior corresponding to the latent variable representation $N(\frac{\partial}{\partial x_1},\ldots,\frac{\partial}{\partial x_n})w = M(\frac{\partial}{\partial x_1},\ldots,\frac{\partial}{\partial x_n})l$. Then the following are equivalent:*

(i) $w \in \mathfrak{B}$

(ii) $h \in \mathrm{SYG}(M^T) \Rightarrow h^T(\frac{\partial}{\partial x_1},\ldots,\frac{\partial}{\partial x_n})N(\frac{\partial}{\partial x_1},\ldots,\frac{\partial}{\partial x_n})w = 0$

This result depends again on the solution space choosen! The theorem holds for $\mathfrak{C}^\infty$ or distributions but, for example, not for solutions that are $\mathcal{L}^1_{\mathrm{loc}}$.
A consequence of this theorem and the the fact that a Sygygy module is noetherian is the following corollary:

Corollary 2.2.3 *(Elimination theorem) Let $\mathfrak{B}$ be the manifest behavior corresponding to the latent variable representation $N(\frac{\partial}{\partial x_1},\ldots,\frac{\partial}{\partial x_n})w = M(\frac{\partial}{\partial x_1},\ldots,\frac{\partial}{\partial x_n})l$. Then there exists a polynomial matrix H such that*

$$\mathfrak{B} = \ker(H^T(\frac{\partial}{\partial x_1},\ldots,\frac{\partial}{\partial x_n})N(\frac{\partial}{\partial x_1},\ldots,\frac{\partial}{\partial x_n}))$$

This corollary is referred to as the *elimination theorem*. In order to compute a kernel representation of the manifest behavior, we have to find a set of generators of the Sygygy module of M^T. For systems with a one dimensional index set, in Matlab's *Polynomial Toolbox* there is an implementation for finding a set of generators.

In the Behavioral Toolbox:

The Polynomial Toolbox command *xab* is used. This command solves the polynomial matrix equation $XA = B$. If B is a zero matrix of any size, it computes a basis for the left nullspace of A. BTB uses the m-file *ieliminate.m*, interfacing BTB and Matlab

---

2.3 Observability and Controllability

Two central concepts in systems theory are the notions of *observability* and *controllability*. Classically, for state space systems (see section 2.4.2), observability is defined as the possibility of deducing the state from an observed output. Controllability is defined as the possibility of transferring the state from any initial state to any final state value. In behavioral systems theory however, see [5],[1] and references therein, these notions are extended to more general model classes of systems and do not depend a priori on a input/state/output structure being given.

2.3.1 Observability

In behavioral systems theory, observability means the possibility of deducing some variables while observing the other ones. However, we restrict attention to observability of latent variables:

Definition 2.3.1 *Let $\Sigma = (\mathbb{I}, \mathbb{W}, \mathbb{L}, \mathfrak{B}_{\text{full}})$ be a dynamical system with latent variables. The latent variable l is observable from w if*

$$(w, l) \in \mathfrak{B}_{\text{full}} \text{ and } (w, l') \in \mathfrak{B}_{\text{full}} \Rightarrow l = l'$$

If the full behavior is linear, then this definition is equivalent to the condition

$$(0, l) \in \mathfrak{B}_{\text{full}} \Rightarrow l = 0$$

For a hybrid representation $N(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})w = M(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})l$, asking for observability is thus equivalent with asking for $M(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})$ to be an injective map.

Theorem 2.3.2 *Let $\mathfrak{B}_{\text{full}}$ be the linear differential behavior given by*

$$\mathfrak{B}_{\text{full}} = \{(w, l) \in \mathfrak{C}^\infty \mid N(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})w = M(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})l \text{ holds}\}$$

with $M \in \mathbb{R}^{\bullet \times 1}[\xi_1, \ldots, \xi_n]$. Then the following are equivalent:

1. *l is observable from w*
2. *$< M^T >= \mathbb{R}^1[\xi_1, \ldots, \xi_n]$*
3. *$\text{rank}(M(\lambda_1, \ldots, \lambda_n)) = l$ for all $\lambda_i \in \mathbb{C}$*

Condition 2 of the above theorem gives us the possibility of verifying observability.

In the Behavioral Toolbox:

In [1], an algorithm for checking observability is deduced (algorithm 90). Short description of the algorithm: the idea is to extract rows from M of degree zero by looking at the matrix M_{hc}, which consists of the coefficients of entries (polynomials) of M corresponding to the highest degree of every row. If the rows of M_{hc} are not independent, select the corresponding row of highest degree of the rows of M and replace it by the previously found linear combination of these rows. This degree reduction corresponds to multiplying with a unimodular matrix and hence the behavior does not change, see [5] and theorem 2.1.1. If the rows of degree zero span $\mathbb{R}^l$ then condition 2 of theorem 2.3.2 holds. For a detailed description of the algorithm, the reader is referred to [1].
BTB uses the m-file *iobservability.m* as the interfacing file with Matlab. This file primarily uses the m-file *observability.m*

2.3.2 Controllability

We introduce a definition, see [5] and [1], of controllability which only relies on the system's trajectories, and not on specific properties of special variables chosen to represent it. This in contrast to the classical definition of controllability involving state variables.

Denote by $\bar{U}$ the closure of a set U.

Definition 2.3.3 *A behavior $\mathfrak{B}$ with $\mathbb{I} = \mathbb{R}^n$ is controllable if for all $w_1, w_2 \in \mathfrak{B}$ and open subsets $U_1, U_2 \subset \mathbb{R}^n$ with $\bar{U}_1 \cap \bar{U}_2 = \emptyset$, there exists $w \in \mathfrak{B}$ such that $w = w_1$ on U_1 and $w = w_2$ on U_2.*

So a behavior is controllable if solutions are *patchable*.
In particular, if the index set $\mathbb{I}$ is equal to $\mathbb{R}$, one can easily show that the above condition is equivalent with the following: for all $w_1, w_2 \in \mathfrak{B}$ and $t_1, t_2 \in \mathbb{R}$, $t_1 < t_2$, there exists $w \in \mathfrak{B}$ such that

$$w(t) = \begin{cases} w_1(t) & t \leq t_1 \\ w_2(t) & t \geq t_2 \end{cases}$$

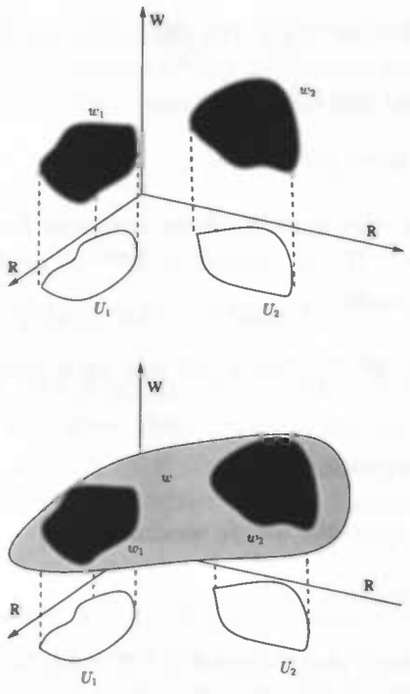

Figure 2.1: N-D controllability

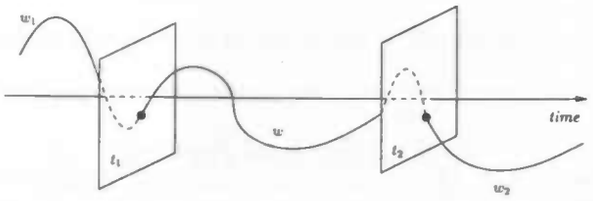

Figure 2.2: 1-D controllability

In the general case that $\mathbb{I} = \mathbb{R}^n$, the following theorem relates the controllability of $\mathfrak{B} = \ker(R(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}))$ with properties of the syzygy module of R:

Theorem 2.3.4 *Let* $R \in \mathbb{R}^{\bullet \times w}[\xi_1, \ldots, \xi_n]$ *and* $M \in \mathbb{R}^{w \times \bullet}[\xi_1, \ldots, \xi_n]$ *be such that* $\mathrm{SYZ}(R) =< M >$. *Moreover, let* $R_1 \in \mathbb{R}^{\bullet \times w}[\xi_1, \ldots, \xi_n]$ *be such that* $\mathrm{SYZ}(M^T) =< R_1^T >$. *Then*

$$\mathfrak{B} = \ker(R(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})) \text{ is controllable } \Leftrightarrow < R^T > = < R_1^T >$$

In the special case that $n = 1$, there holds:

Theorem 2.3.5 *Given* $R \in \mathbb{R}^{\bullet \times w}[\xi]$. *The following are equivalent:*

1. $\mathfrak{B} = \ker(R(\frac{\partial}{\partial t}))$ *is controllable*
2. $\text{rank}(R(\lambda))$ *is the same for all* $\lambda \in \mathbb{C}$
3. *If* N *is any polynomial matrix whose columns form a minimal generating set for the module* $< R^T >$, *then* $N \in \mathbb{R}^{\bullet \times c}[\xi]$ *and* $\text{rank}(N(\lambda)) = c$ *for any* $\lambda \in \mathbb{C}$ *where* $c = \text{rank}(R)$

Theorem 2.3.6 *Let* $R \in \mathbb{R}^{p \times w}[\xi]$ *be a full row rank polynomial matrix. The following are equivalent:*

1. $\mathfrak{B} = \ker(R(\frac{\partial}{\partial t}))$ *is controllable*
2. *The columns of* R *span the whole module* $\mathbb{R}^p[\xi]$
3. $\text{rank}(R(\lambda)) = p$ *for all* $\lambda \in \mathbb{C}$

Comparing this theorem and theorem 2.3.2, ones sees that the conditions for controllability are dual to the condition for observability only in case R is of full row rank. Obviously, you cannot observe "too many" variables (M does not have full column rank, excluded in theorem 2.3.2), whereas "too many" equations (R does not have full row rank, excluded in theorem 2.3.6) may still define a controllable behavior.

In the Behavioral Toolbox:

This duality implies that we can use the observability algorithm in case R is of full row rank. In general, observability of R^T checks condition 2 of theorem 2.3.6 and outputs a degree reduced matrix R whose columns generate the same module as the original R. If true comes out, then we are done. If not, then, as already discussed, the behavior could still be controllable. The algorithm extracts constant columns of R until they span the proper space or until no more degree lowering is possible. In the last case the columns of nonzero degree are indepent of the other columns. In general, one cannot expect that these obtained constant columns are independent. If they indeed are not independent, replace these columns with columns that are independent and that span the same space. The m-file *reduce.m* does the computation. Since the columns of degree more then zero are already independent of the other columns, the number of columns of the transformed matrix R is equal to its rank. In case the rank equals the row dimension of R, then by theorem 2.3.6, the test performed by the observability algorithm is necessary and

sufficient for controllability. Otherwise, we apply condition 3 of theorem 2.3.5 to check controllability; use the observability algorithm again with the transformed R as input. BTB uses the m-file *controllability.m* and interface file *icontrollability.m* .

2.4 Inputs, Outputs and States

Classically, most models of a physical system are given in input/output form or input/state/output form. However, these description are not a very natural thing to start with. First of all, models from physics do not occur in either one of these forms. Secondly, interconnected systems are not described by input/output pairing.

Of course these concepts are very useful, for instance for simulation. But these structures are too restrictive as a starting point in a more general framework. Input/output representations or input/state/output representation show up naturally *a posteriori* instead of *a priori*.

2.4.1 Inputs and Outputs

Equations that usually appear in textbooks are of the form:

$$Q(\frac{d}{dt})u = P(\frac{d}{dt})y$$

with P and Q polynomial matrices such that $P^{-1}Q$, the transfer matrix of the system, is a matrix of proper rational functions i.e. the degree of Q is at most the degree of P. In this case, u is called the input, yielding an output y. We begin with the concept of *free* variables of a differential system:

Definition 2.4.1 *Let $\Sigma = (\mathbb{I}, \mathbb{W}, \mathfrak{B})$ be a differential system. Variables u are said to be free if there exists a permutation matrix Π such that for all $u \in \mathfrak{C}^\infty(\mathbb{R}^n, \mathbb{R}^u)$ there exist a $y \in \mathfrak{C}^\infty(\mathbb{R}^n, \mathbb{R}^y)$ yielding $w = \Pi(u, y) \in \mathfrak{B}$. Variables u are called maximally free in $\mathfrak{B}$ if y does not contain any further free components.*

This definition also holds for distributions.

Lemma 2.4.2 *Let $\mathfrak{B} = \ker([Q(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}) \quad - P(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})])$. Let $w = (u, y)$ be partitioned according to the Q and P matrix. Then u is free if and only if*

$$\mathrm{SYZ}(P^T) = \mathrm{SYZ}([Q - P]^T)$$

However, which variables among the variables actually can be regarded as maximally free is not unique. In fact, many possible subsets of the w's can play this role. What is unique is the *number* of maximally free variables. The following theorem tells us what this number is.

Theorem 2.4.3 *Let* $R \in \mathbb{R}^{\bullet \times \mathtt{w}}[\xi_1, \ldots, \xi_n]$. *The cardinality of a set of maximally free variables in* $\mathfrak{B} = \ker(R(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}))$ *equals* $\mathtt{w} - \mathrm{rank}(R)$

This can be interpreted as follows: the number of maximally free variables equals the number of variables ($\mathtt{w}$) minus the number of independent constrains imposed ($\mathrm{rank}(R)$). The following definition leads to inputs and outputs in the classical sense:

Definition 2.4.4 *Variables* u *are said to be smoothly free in* $\mathfrak{B}$ *if for all* $u \in \mathfrak{C}^k(\mathbb{R}^n, \mathbb{R}^{\mathtt{u}})$ *there exist* $y \in \mathfrak{C}^k(\mathbb{R}^n, \mathbb{R}^{\mathtt{y}})$ *such that* $w = \Pi(u, y) \in \mathfrak{B}$ *with* Π *a permutation matrix. They are maximally smoothly free if* y *contains no further smoothly free components.*

Now the theorem that establishes the link with the classical concept of inputs:

Theorem 2.4.5 *Let* $\mathfrak{B} = \ker(R(\frac{d}{dt}))$ *be described by*

$$Q(\frac{d}{dt})u = P(\frac{d}{dt})y$$

with $w = (u, y)^T$ *and* $R = [Q - P]$ *of full row rank. Then* u *is maximally smoothly free if and only if* P *is a nonsingular submatrix such that* $P^{-1}Q$ *is a matrix of proper rational functions.*

In the Behavioral Toolbox:

Theorem 2.4.5 is used. Firsts R is row reduced using the m-file *prowred.m* of The Polynomial Toolbox, where after the matrix is sorted by descending column degree. Then the matrix R_{hc} is formed which consists of the coeffients of R corresponding to the columns of highest degree of R. Subsequently, a minor of R_{hc} is obtained with the use of the m-file *minor.m* which start looking at the first columns. Now stack the corresponding columns of R, defining a matrix P, and stack the remaining columns of R, defining a matrix Q. This guarantees that $P^{-1}Q$ is a matrix consisting of proper rational functions. The m-file that BTB uses is *io.m* with interface file *iio.m*

2.4.2 State Representations

Define the concatenation at t_0, $\wedge_{t_0}$, as

$$f \wedge_{t_0} g = \begin{cases} f(t) \text{ for } t < t_0 \\ g(t) \text{ for } t \geq t_0 \end{cases}$$

We begin by giving an abstract definition of state:

Definition 2.4.6 *Let $\Sigma_L = (\mathbb{R}, \mathbb{W}, \mathbb{X}, \mathfrak{B}_{\text{full}})$ be a dynamical system with latent variables $x \in \mathbb{X}$ and manifest variables $w \in \mathbb{W}$. Σ_L is called* a state system *if $(w_1, x_1), (w_2, x_2) \in \mathfrak{B}_{\text{full}}$ and $x_1(t_0) = x_2(t_0)$ imply $(w_1, x_1) \wedge_{t_0} (w_2, x_2) \in \mathfrak{B}_{\text{full}}$. The variable x is then called the* state.

This definition captures the intuition of state variables as being the memory of the system. It summarizes all the information of the past of the system which we need to know in order to establish its future behavior.

Theorem 2.4.7 *An ordinary differential system $N(\frac{d}{dt})w = M(\frac{d}{dt})x$ with latent variables x and manifest variables w is a state system if and only if there exists real matrices E, F, H such that the behavior $\mathfrak{B}_{\text{full}}$ is represented by*

$$E\frac{d}{dt}x + Fx + Hw = 0 \tag{2.1}$$

Note that equations 2.1 are of first order in x and of order zero in w.

Definition 2.4.8 *Let $\mathfrak{B} = \ker(R(\frac{d}{dt}))$. A polynomial matrix X is said to be a state map for $\mathfrak{B}$ if the latent variable system*

$$R(\frac{d}{dt})w = 0$$
$$X(\frac{d}{dt})w = x$$

is a state system

The following result shows that every kernel representation admits a state map. First we define the *shift-and-cut* operator $\sigma : \mathbb{R}^{p\times q}[\xi] \to \mathbb{R}^{p\times q}[\xi]$ as

$$P_0 + P_1\xi + \cdots + P_L\xi^L \mapsto P_1 + P_2\xi + \cdots + P_L\xi^{L-1}$$

Theorem 2.4.9 *Let $R \in \mathbb{R}^{p\times w}[\xi]$ with $\deg(R) = L$. Then the polynomial matrix*

$$\Xi_R = \begin{bmatrix} \sigma(R) \\ \vdots \\ \sigma^L(R) \end{bmatrix}$$

induces a state map for $\mathfrak{B} = \ker(R(\frac{d}{dt}))$.

Of course we can use the results from section 2.4.1, in order to obtain the following:

Theorem 2.4.10 *Let* $\mathfrak{B} = \ker(R(\frac{d}{dt}))$. *Then there exists real matrices* A, B, C, D *and a permutation matrix* Π *such that* $\Pi(u, y)^T \in \mathfrak{B}$,

$$\begin{aligned} \frac{d}{dt}x &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{2.2}$$

defines a state system and variables u *are smoothly free. Equations (2.2) are called an input/state/output representation for* $\mathfrak{B}$.

However, a state representation is not unique. In fact, see [3], for a kernel behavior $\mathfrak{B}$ with kernel matrix R, all polynomial matrices X such that $\Xi_R = AX + BR$ for a real matrix A and a polynomial matrix B, induces a state map for $\mathfrak{B}$. What is unique, is the smallest *number of state variables* associated to $\mathfrak{B}$. This number, called *dynamic order* or *McMillan degree* of $\mathfrak{B}$, is denoted by $n(\mathfrak{B})$. State maps with exactly $n(\mathfrak{B})$ rows are called *minimal* state maps. In fact:

Proposition 2.4.11 *Let* $\mathfrak{B} = \ker(R(\frac{d}{dt}))$ *with* R *row reduced. Then the polynomial matrix* X *that consists of the nonzero rows of* Ξ_R *induces a minimal state map for* $\mathfrak{B}$.

In the Behavioral Toolbox:

A minimal input/state/output representation is computed, using proposition 2.4.11. First, R is row reduced and a state map X is constructed from Ξ_R. Now theorem 2.4.10 implies that:

$$\begin{bmatrix} R & 0 \\ X & -I \end{bmatrix} \sim \begin{bmatrix} I & -D & -C \\ 0 & -B & I\xi - A \end{bmatrix} \tag{2.3}$$

for real matrices A, B, C and D, where $R_1 \sim R_2$ means that there exists a unimodular matrix U such that $R_1 = UR_2$. This property is exploited in the m-file *iso.m* with interface file *iiso.m*.

2.5 Simulation

Obviously, one wants to compute trajectories of the behavior of the dynamical system being modeled. In this section we will discuss behaviors of the form

$$\mathfrak{B}_{\text{full}} = \{(w, f) \in \mathfrak{D}'(\mathbb{R}, \mathbb{R}^{\mathbf{w}+\mathbf{f}}) \quad | \quad G(\frac{d}{dt})w = M(\frac{d}{dt})f\} \tag{2.4}$$

We think of f as a vector-valued distribution called *forcing functions* or *external functions*, while w is a trajectory to be computed. Together with (2.4), we also consider initial conditions

$$(S(\frac{d}{dt})w)(t_0) = a \tag{2.5}$$

where S is a polynomial matrix and a a real vector.

Definition 2.5.1 *Let* $\mathfrak{B}_{\text{full}} = \{(w, f) \in \mathfrak{D}'(\mathbb{R}, \mathbb{R}^{\mathbf{w}+\mathbf{f}}) \mid G(\frac{d}{dt})w = M(\frac{d}{dt})f\}$. *Behavior* $\mathfrak{B}_{\text{full}}$ *is solvable for a given* $f \in \mathfrak{D}'(\mathbb{R}, \mathbb{R}^{\mathbf{f}})$ *if there exists a* $w \in \mathfrak{D}'(\mathbb{R}, \mathbb{R}^{\mathbf{w}})$ *such that* $(w, f) \in \mathfrak{B}_{\text{full}}$. *If this holds for any* $f \in \mathfrak{D}'(\mathbb{R}, \mathbb{R}^{\mathbf{f}})$, *then* $\mathfrak{B}_{\text{full}}$ *is solvable.*

This definition can be recast in the language of latent variable representations. Consider $\mathfrak{B}_{\text{full}}$ as a full behavior with latent variables w and manifest variables f, and $\mathfrak{B}$ is the manifest behavior. Then solvability for a given f simply means $f \in \mathfrak{B}$, while $\mathfrak{B}_{\text{full}}$ is solvable is equivalent to $\mathfrak{B} = \mathfrak{D}'(\mathbb{R}, \mathbb{R}^{\mathbf{f}})$. This directly (theorem 2.2.2) leads to

Corollary 2.5.2 $\mathfrak{B}_{\text{full}}$ *as defined in (2.4) is solvable for a given* $f \in \mathfrak{D}'(\mathbb{R}, \mathbb{R}^{\mathbf{f}})$ *if and only if*

$$n \in \text{SYZ}(G^T) \Rightarrow n^T(\frac{d}{dt})M(\frac{d}{dt})f = 0$$

$\mathfrak{B}_{\text{full}}$ *is solvable if and only if*

$$n \in \text{SYZ}(G^T) \Rightarrow n^T M = 0$$

In the Behavioral Toolbox:

From the above corollary it follows that if G of full row rank, then solvability is assured for any f. It also leads to the following:

$$\begin{bmatrix} G & M \end{bmatrix} \sim \begin{bmatrix} G' & M' \\ 0 & M_s \end{bmatrix}$$

with G' of full row rank. Now $\mathfrak{B}_{\text{full}}$ is solvable for given f iff $M_s(\frac{d}{dt})f = 0$ and solvable iff $M_s = 0$. If we write $M_s = M_{s0} + \cdot + M_{sd}\xi^d$ then $M_s(\frac{d}{dt})f = 0 \Leftrightarrow [M_{s0} \ \cdots \ M_{sd}][f \ \cdots \ f^{(d)}]^T = 0$. These derivatives are calculated within BTB and are send to the Matlab working space. Caution, if the step size for the sampled time is not small enough, it can lead to a false answer!!!

Now suppose that $\mathfrak{B}_{\text{full}}$ is solvable for a given f. How smooth is a trajectory w such that $(w, f) \in \mathfrak{B}_{\text{full}}$? This relationship leads to the *index* of a behavior.

Definition 2.5.3 *Let $\mathfrak{B}_{\text{full}}$ be defined as in 2.4. Define*

$$\mathfrak{J} = \{(j_1, \ldots, j_{\mathtt{f}}) \quad | \quad \text{for all } f \text{ with } f_i \in \mathfrak{C}^{k+j_i} \text{ there exists } w \in \mathfrak{C}^k \text{ such that } (w, f) \in \mathfrak{B}_{\text{full}}\}$$

where $\mathfrak{C}^k$ for $k < 0$ is defined as the set of all distributions whose $|k|$-th primitive is a continuous function. Let $\succeq$ be the partial ordering $(\alpha_1, \ldots, \alpha_{\mathtt{f}}) \succeq (\beta_1, \ldots, \beta_{\mathtt{f}}) \Leftrightarrow \alpha_i \leq \beta_i, i = 1, \ldots, \mathtt{f}$. Define the multi-index μ as the smallest element of $\mathfrak{J}$ with respect to such a partial ordering, and the index $\nu = \max\{\mu_i \mid i = 1, \ldots, \mathtt{f}\}$.

The multi-index establishes the minimal differentiability requirement on each component of f which assures that a sufficient differentiable trajectory w can be found. Indeed, this multi-index is well defined:

Theorem 2.5.4 *Let $\mathfrak{B}_{\text{full}}$ be defined as in (2.4) with $G \in \mathbb{R}^{p \times \mathtt{w}}[\xi]$ of full row rank. Let P be a square submatrix of G of maximal determinantal degree. Let δ_{ij} be the difference between the degree of the numerator and the denominator of $(P^{-1}M)_{ij}$. Then the multi-index is given by $(\mu_1, \ldots, \mu_{\mathtt{f}})$ with $\mu_j = max_i \delta_{ij}$.*

In the Behavioral Toolbox:

Theorem 2.5.4 is implemented in the m-file *index.m* which calculates the index of a behavior of the form (2.4). The interface file is *iindex.m*.

How to check if there is a solution w of the behavior of the form (2.4), solvable for a given f, that satisfies $(S(\frac{d}{dt})w)(t_0) = a$?

In the Behavioral Toolbox:

Since we already have checked the solvability condition, we now have a representation $G(\frac{d}{dt})w = M(\frac{d}{dt})f$ with G of full row rank. Assume also that G is square so that $\det(G) \neq 0$, since there will be no unique solution if G is not square. We can write $M = GQ + R$ with R such that $R = 0$ or $G^{-1}R$ is strictly proper. This is done with the use of the m-file *pdiv.m*. In case G is of degree zero, $R = 0$ and the initial conditions $(Sw)(t_0) = a$ yields $(SQf)(t_0) = a$. A degenerated state space representation i.e. without state variables is constructed and simulated with the command *lsim*.
Now consider the case that G is not of degree zero. Since

$$\begin{array}{rcl} Gw & = & Mf \\ & = & GQf + Rf \end{array} \Rightarrow G\underbrace{(w - Qf)}_{:=e} = Rf$$

and $G^{-1}R$ proper, we have a state system

$$\frac{d}{dt}x = Ax + Bf \tag{2.6}$$

$$e = Cx + Df \tag{2.7}$$

This last equation is equivalent to $w = Cx + (Q + D)f$. Therefore,

$$(Sw)(t_0) = a \Leftrightarrow (SCx)(t_0) + (S(Q + D)f)(t_0) = a.$$

Denote $S(Q + D) = L$. We can rewrite SCx with the use of (2.6). Suppose $S(\xi) = S_0 + S_1\xi + \cdots + S_s\xi^s$ and $L = L_0 + \cdots + L_l\xi^l$. If s $= 0$ then $Sw = S_0x + [L_0 \ \cdots \ L_l][f \ \cdots \ f^{(l)}]^T|_{t=t_0}$. Otherwise, since

$$(\frac{d}{dt})^n x = A^n x + A^{n-1}Bf + \cdots + ABf^{(n-2)} + Bf^{(n-1)}$$

it follows that

$$\begin{array}{rcl} SCw|_{t=t_0} & = & S_0Cx + \cdots + S_sC(\frac{d}{dt})^s x|_{t=t_0} + Lf|_{t=t_0} \\ & = & [S_0 \ \cdots \ S_s][C \ \ CA \ \ \cdots \ \ CA^s]^T x|_{t=t_0} \ + \\ & & [S_1 \ \cdots \ S_s] \begin{bmatrix} C & 0 & \cdots & 0 \\ CA & C & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ CA^{s-1} & CA^{s-2} & \cdots & C \end{bmatrix} \begin{bmatrix} Bf \\ \vdots \\ Bf^{(s-1)} \end{bmatrix} \Big|_{t=t_0} \\ & & + \ [L_0 \ \cdots \ L_l][f \ \cdots \ f^{(l)}]^T|_{t=t_0} \\ & = & a \end{array}$$

Rewrite this to an equation of the type $Px_0 = b$ where $x_0 = x(t_0)$. Write $P = UXV$, the SVD-decomposition of P, yielding

$$Px_0 = b \Leftrightarrow U^{-1}Px_0 = U^{-1}b$$

After removing the last zero equations, we have obtained an equation for x_0:

$$\tilde{P}\, x_0 = \tilde{b}$$

with $\tilde{P}$ of full row rank. If rank($[\tilde{P} \quad \tilde{b}]$) > rank($\tilde{P}$), no solution exists i.e. the initial conditions are not well-posed. Otherwise we take $x_0 = \tilde{P}^T (\tilde{P}\tilde{P}^T)^{-1} \tilde{b}$. If $Q = Q_0 + \cdots + Q_q\xi^q$, then we have a state system

$$\begin{aligned} \frac{d}{dt}x &= Ax + [B\ 0 \cdots\ 0]df \\ w &= Cx + [Q_0\ \cdots\ Q_q]df \end{aligned}$$

where $df = [f\ \ f^{(1)}\ \ \cdots\ \ f^{(q)}]^T$. With the use of matlab's command *lsim*, a solution w of this state space representation is computed with $x(t_0) = x_0$ as initial condition.

2.6 Things for future releases of BTB

Of course there is much more developed in behavioral systems theory. Further more, due to time limitation, there are a number of things that should be supplemented or extended.

- A state map for the manifest behavior is calculated by first calculating the kernel representation of the manifest behavior by elimination. It is also possible to calculate this state map with the use of the hybrid representation.
- A linear time-invariant differential system is controllable if and only if it admits an image representation. Computing image representations would be a valuable extension of the program.
- For nonlinear systems, we wish to compute the stationary points and linearize the system in one of these points.
- The more general theory for ND-kernel behaviors should be implemented as well. This requires the use of *Gröbner bases*, see [1] for more information.
- Implement *stabilizability* of a kernel behavior and *detectability* of a full kernel behavior. For more information about these topics, the reader is referred to [5].
- As discussed in section 2.4.1, an input/output partition of a behavior is not unique. The program picks just one of them and this partition may not be the partition one is looking for.
- A state map for the behavior can be computed. But this state map can not be used for the initial conditions in the simulation automatically.
- Equations has to be typed in with the characters # for variables and @ for parameters. This is not very desirable.
- In the simulation window, it would be convenient if the user can see the impuls response, etc at an earlier stage.
- Eventually, the program should be autonomous i.e. does not need Matlab for its computations.
- A consequence of the limitation of the language JAVA, where BTB is written in, and the time limitation is that the user interface visualizes the model of an interconnected not very clear. Particularly, the use of a scratch of paper and pencil to assist the user is almost inevitable. A way to omit this

adventitious circumstance is, for instance, the use of icons for the modules and connect them by drag and drop principles.

Bibliography

[1] T. Cotroneo. *Algorithms in Systems Theory.* PhD thesis, Rijksuniversiteit Groningen, 2001.

[2] R. Morelli. *Java, Java, Java Object-Oriented Problem Solving.* New Jersey, Prentince Hall, 2000.

[3] P. Rapisarda. *Linear Differential Systems.* PhD thesis, Rijksuniversiteit Groningen, 1998.

[4] J.C. Willems. The behavioral approach to modeling and control of dynamical systems. *Proceedings of the Sixth Conference on Process Control, Tucson, AZ*, to appear.

[5] J.W. Polderman & J.C. Willems. *Introduction to Mathematical Systems Theory: A Behavioral Approach.* Springer Verlag, 1998.

[6] P.H. Winston. *Onto Java.* Addison-wesly, 1998.

Chapter 3

Documentation

How to use the Behavioral Toolbox?

3.1 System Requirements

In order to be able to run BTB (short for Behavioral Toolbox), you must have:

- OS: Win95/98/ME/NT4/NT5/2000 or some Unix based system such as Linux
- Matlab version at least 5 should be installed with a license of the control toolbox
- Java version 1.3 is also required. It can be obtained for free at: http://java.sun.com

Remark: If you have no license for the control toolbox, the only thing that doesn't work in BTB is displaying trajectories in a figure.

3.2 Installation and getting started

To install the program on the system (be sure that you have write permission), type in the following (or something similar):

java -jar install.jar

Follow the instructions on the screen. Every user with read acces to the specified program location can now setup BTB as follows:

java -jar BTB.jar

and fill in the blanc fields. Your settings will be stored in a file ".BTB.ini" on the root of your account. Now you are able to run BTB:

java -jar BTB.jar

3.3 What is BTB

There are many simulation programs in use, but almost all based on regarding systems as signal processors, or using Bond-graph methodology (for instance Matlab's Simulink, Spice, Modelica and MTT). Our goal was to create a simulation tool based on "The behavioral approach" (e.g. see [1] and [3]) for more details) and modeling interconnected systems with the terminal approach (see chapter 1). Our main interest are linear systems, this doesn't mean that nonlinear systems cannot be considered, but the possibilities are more limited.

The program is written in the language JAVA. The main reason is that JAVA is an OO language (see appendix A.1). For heavy calculations we use Matlab. The desktop of Matlab will not appear so the user isn't aware of its use.

In the next figure an example is given of the main window of BTB, which contains a model of an electrical system modeled in the Behavioral Toolbox. How to model such a system is explained in section 3.4.
In figure 3.1, you see information of the system being modeled:

Name	The name of the inserted module, provided by the user
Type	Type of the inserted module, i.e. Resistor, Capacitor, Spring, Vessel
Free terminals	The terminals of the module which are not connected
Connected terminals	The terminals of the module which are connected with other terminals
Connection architecture	The connection architecture of the whole system; in the picture you can see that terminal 1 is connected with terminal 4 and terminal 2 with 6

Also some properties of the system are shown in a TextPane below the lists. This is discussed in section 3.6.4.

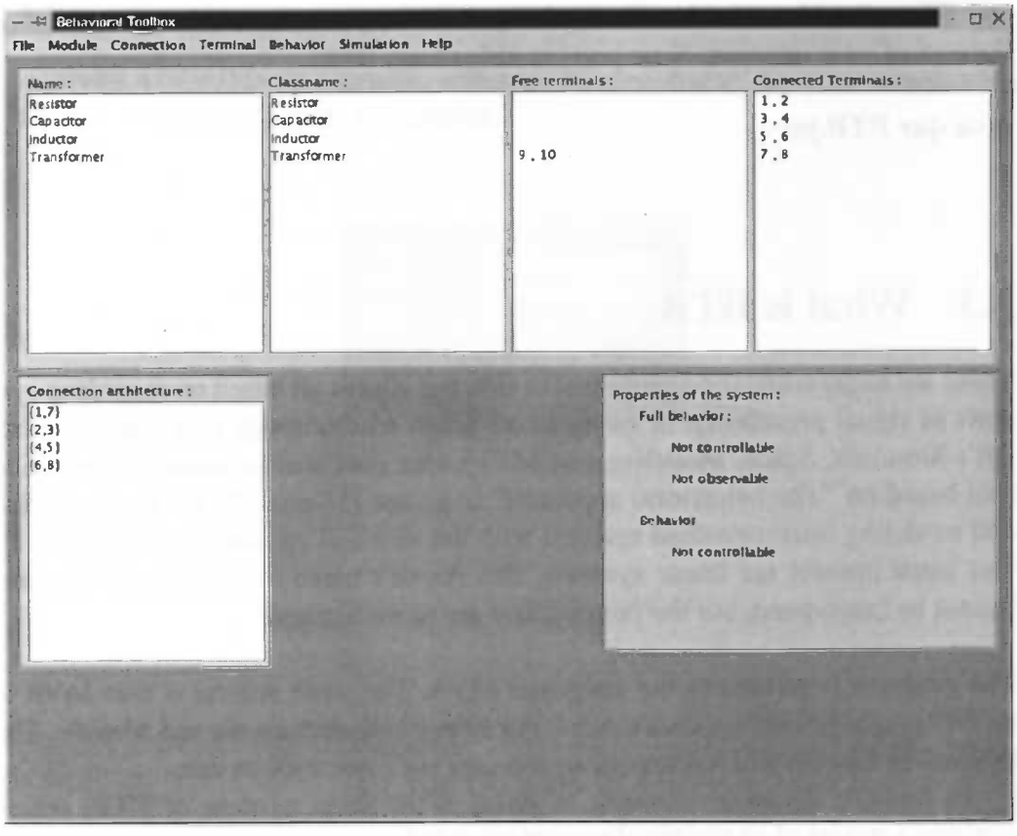

Figure 3.1: Example of a system in the Behavioral Toolbox

3.4 Modeling a system

To model a dynamical system, you insert the modules and connect the terminals of those modules. We already implemented a number of standard modules, but it is also possible to model a system with a new module kind of module. For that, you have to create a new module class. This should be done with the following steps:

1. Create a new type of terminal (if necessary).
2. Create a connection for that type of terminal (if necessary).
3. Create a new module class.

A logical way to model a system with the BTB is to

1. Insert modules in your system.

2. Connect terminals.

Conversely, you can also undo these actions:

1. Disconnect terminals.
2. Remove modules from your system.

How to do this is explained in the next seven subsections.

3.4.1 Create a new type of terminal

To create a new type of terminal, select in the menubar in the main window **Terminal > Create new type of terminal.**

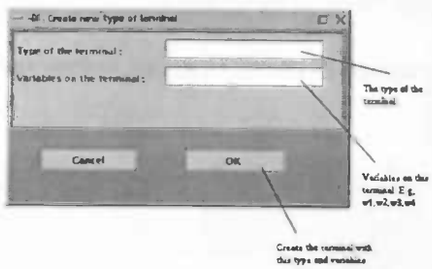

Figure 3.2: Create a terminal

Specify the type of the terminal, which amounts to associating the set of corresponding variables with this terminal. In the second textfield, the variables has to be typed in. When ready, press "OK" to create this new type of terminal.

3.4.2 Create a connection

As discussed in chapter 1, connecting terminals imposes some additional restrictions called interconnection constrains. In the program, a connection carries these restrictions. In order to connect a new type of terminal, a new type of connection has to be created first. Select **Connection > Create new connection** in the menubar in the main window.

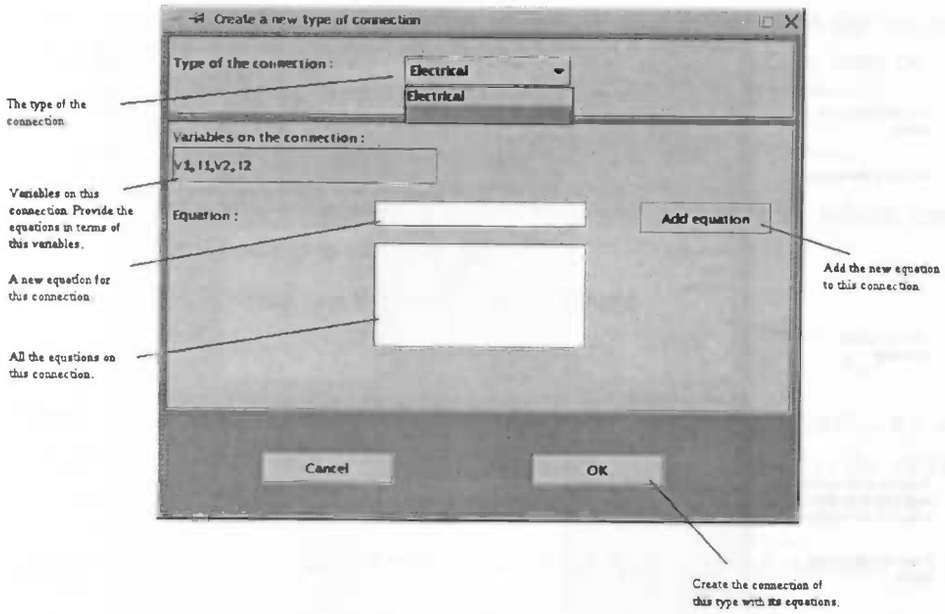

Figure 3.3: Create a connection

Select a type for this connection in the choiselist. The connection constrains has to be in terms of the variables that are given. If you don't use these variables, errors occur later on. For an equation we use the following convention:

1. Write the equation in the form f(x) = 0.
2. Put before each variable a # character.

E.g. #w1 + #w2 = 0. Here w1 and w2 are the variables on the terminals.

Press the "OK" button to create the connection.

3.4.3 Create a module class

To create a new module class, select **Module** > **Create new module** in the main window.

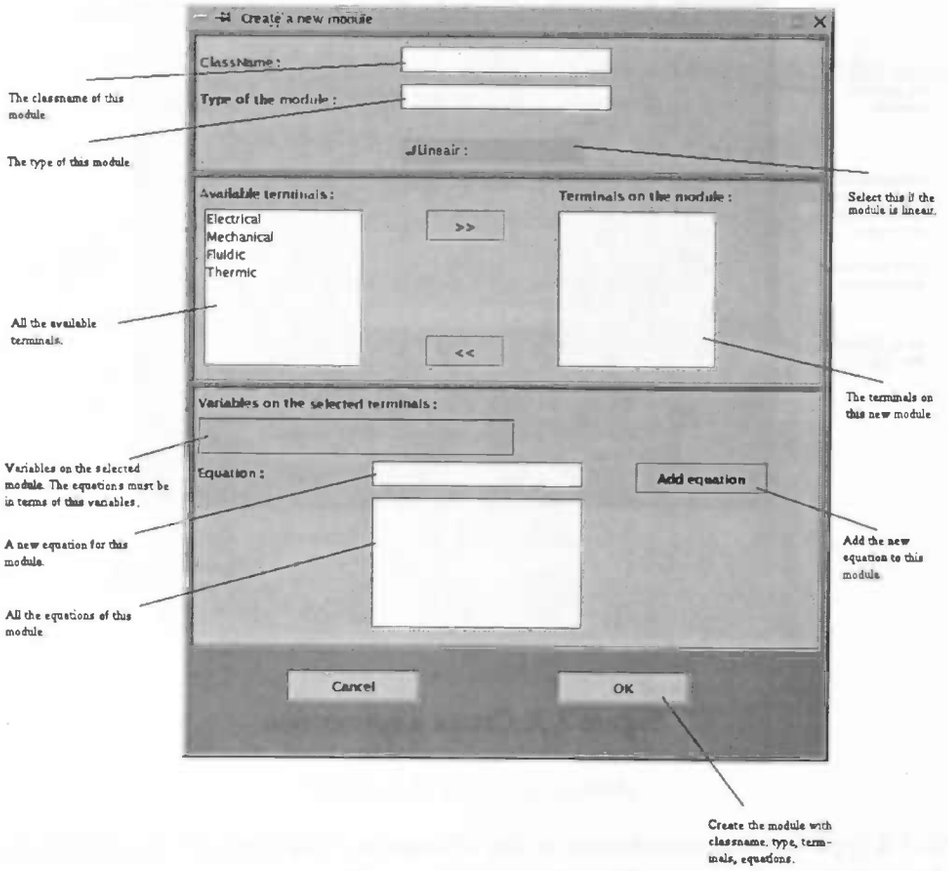

Figure 3.4: Create a module

Now do the following steps:

1. Provide the class name of the module. This is the name that you will henceforth use for this module class. For example, you could use 'Resistor' for an electrical resistor or 'Spring' for a mechanical spring.

2. Provide the type of the module. The type is identified with the name for the family of classes, where this module class belongs to. For example, a resistor belongs to the family of *electrical* modules.

3. Add a number of terminals to the module. (If the terminal is not available, create it first (section 3.4.1)

4. Provide the constitutive equations of the module in terms of the variables that are given. If you don't use these variables, errors occur later on. We use the following convention for an equation:

   (a) Write the equations in the form f(x) = 0.

   (b) Put # before the variables and @ before the parameters, which are the physical constants of the module.

   (c) For derivatives use the D-operator. Where

   $$D(i_1, ..., i_n)f(x_1, ..., x_n) = \frac{\partial^{i_1}}{\partial {x_1}^{i_1}} \cdots \frac{\partial^{i_n}}{\partial {x_n}^{i_n}} f(x_1, ..., x_n)$$

   For example, the equation for a vessel is given by $@S_t * D(1)\#H_t - \#f = 0$. Here H_t is the variable which denotes the height in the tank, f is the variable which denotes the flow and the parameter S_t is the surface of the tank.

5. Press the "Add equation" button to add the equation to the module.

6. Repeat step 4 and 5 for all the equations.

7. Press the linear checkbox if the equations of the module are linear.

8. Press the "OK" button to create the module.

3.4.4 Insert a module in the system

To insert a module into the system, select **Module** > **Insert module**. If the module is not available, created it first (section 3.4.3).

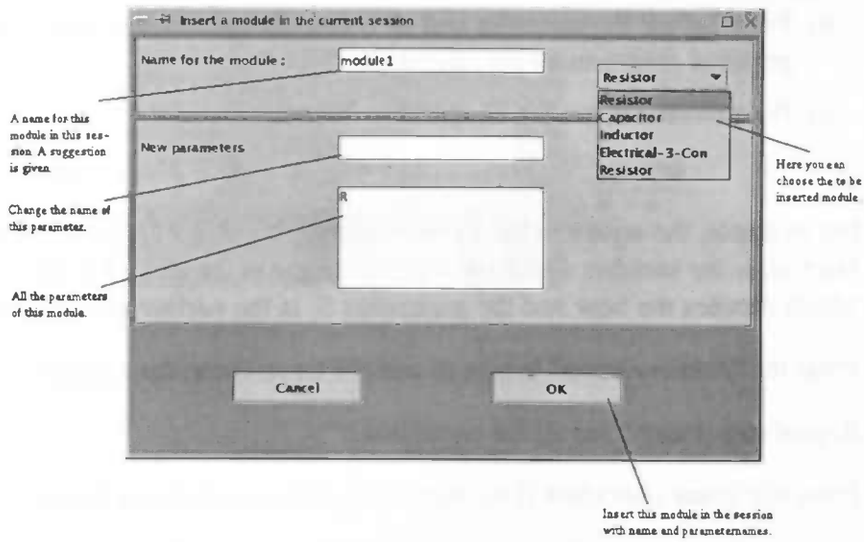

Figure 3.5: Insert a module

Select the type and the class name of the module you want to insert. A suggestion for a name of this module - only in this session - is already given, but you are free to change it. Also the symbolic parameter can be changed as well, so there will be no parameters with the same identifier in the current session. Press the "OK" button to insert this module.

3.4.5 Connect two terminals

Terminals that are not already connected (free terminals) can be connected, but only if they are of adapted type (see chapter 1). Not all terminals have to be involved in the interconnections. Those terminals which are not involved are the so called *external* terminals. The terminals entered in the connection architecture have the role of *internal* terminals (see again chapter 1). To connect terminals, select **Terminal** > **Connect** in the main window.

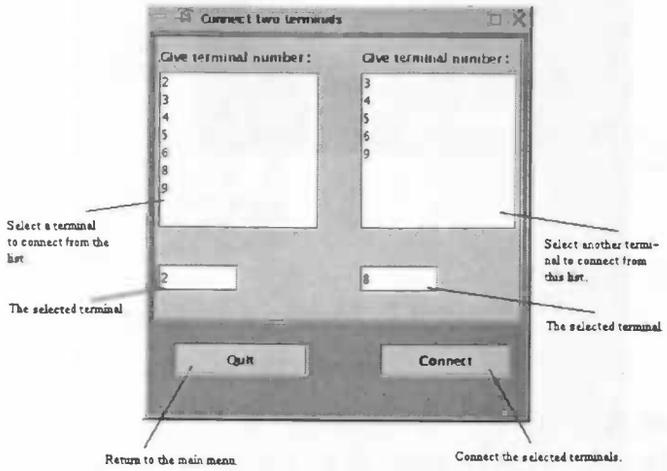

Figure 3.6: Connect terminals

Select a terminal in the left list. Doing so, only the free terminals of the adapted type as the selected terminal remains in the right list. Now select a terminal in the right list and press the "connect" button. Repeat this action for all the connections you want to make. To close the window press the "quit" button. When a connection is made, the variables on the connected terminals will be assigned latent. To change this assignment, see 3.6.2.

3.4.6 Disconnect two terminals

Terminals can be connected, but also the reversal of this is possible. Select the connected pair in the list showing the connection architecture. Select **Terminal** > **Disconnect two terminals** and the terminals will be disconnected. Another way to do this is, is to select the connected pair, press the right mouse button and select **Disconnect**.

3.4.7 Remove a module from your system

A module can be removed from the system. If you want to do this, select the name of the to be removed module. Press your right mouse button and select **Remove from session**. Another way to do this: select **Module** > **Remove from session**.

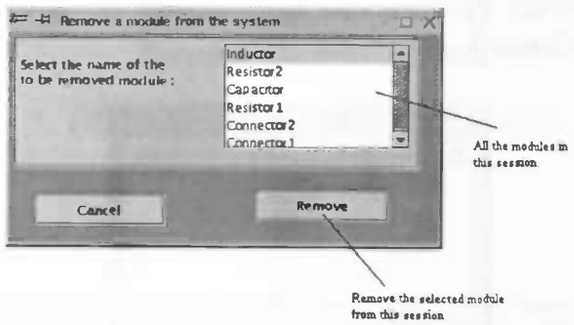

Figure 3.7: Remove a module

Select the name of the to be removed module in the list and press "Remove". The module will be removed (of course all its connected terminals are disconnected and removed as well).

3.5 Information and corrections

In the previous section we discussed how to model an interconnected system and how to change it by (dis)connecting terminals and removing modules. But a mistake is easily made. For example when creating modules and connections. But how do we find out that we made mistakes?

In this section we explain *how* to find out what the mistakes are and how to *undo* this mistakes. We consider the next four items:

1. Show information of the modules.
2. Show information of the connection constrains.
3. Change a module class.
4. Change a connection.

3.5.1 Information of the module

If you want to see information of a certain module, select the name of the module, press the right mouse button and select **Info**.

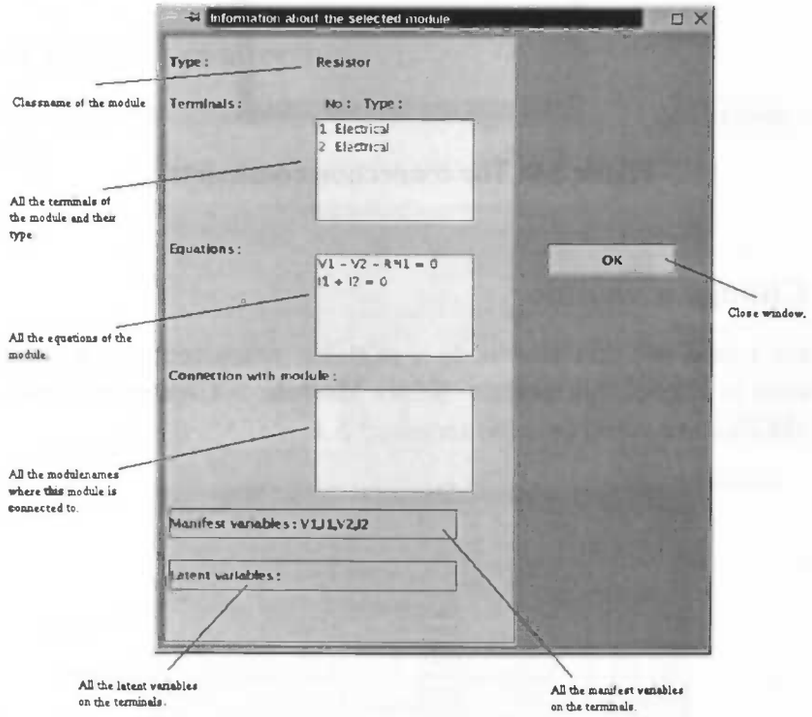

Figure 3.8: Information about the module

Here you see information of the class name, type, linearity, and the equations of the module. But also information of the variables and connections of this module is provided.

3.5.2 Information of the connection constraints

For information of the connection constrains, select the connection in the list with the connection architecture, press the right button of your mouse and select **Connection constraints**.

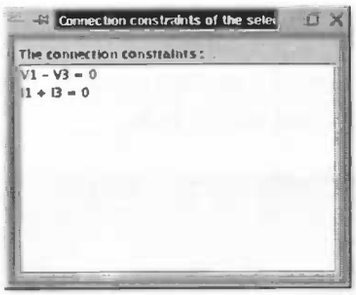

Figure 3.9: The connection constraints

3.5.3 Change a module

If you have found out that you made a mistake, when creating a module, you obvious want to change this module. Select **Module** > **Change module**. To find out what the mistake could be, read section 3.5.1.

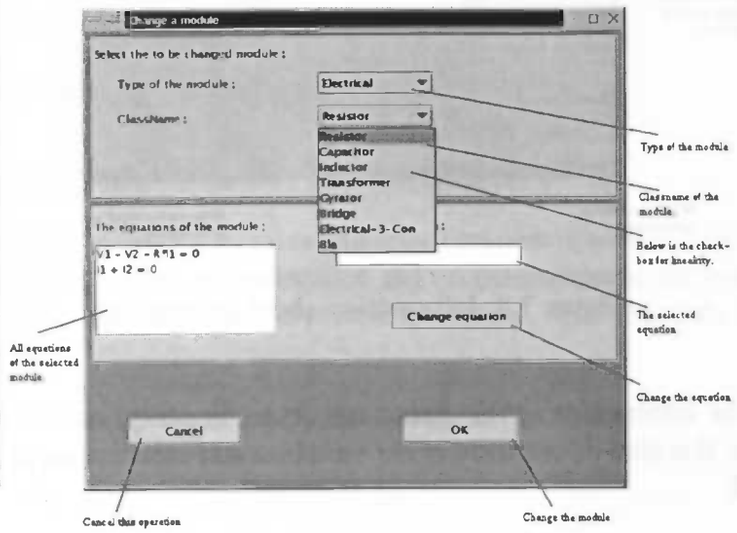

Figure 3.10: Change module

After selecting the module - the one you would change - in the choice box, all the attributes (like type, linearity and equations) of the selected module are shown and you are able to change them. Note that you have to provide the equations as in 3.4.3! When ready press the "OK" button.

3.5.4 Change a connection

If a connection is not correct, you want to change it. Select **Connection > Change connection**.

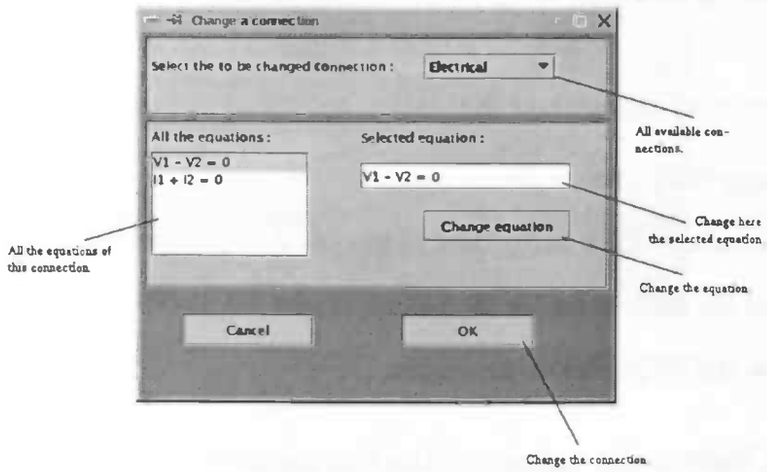

Figure 3.11: Change connection

Subsequently, after selecting the type of the connection you want to change, you can edit the equations of the connection. For notation see 3.4.2.

3.6 Behavior

A typical user is interested in the behavior of the modeled system (equations, kernel representation, input/output representation, etc.). To compute these representations, the program uses Matlab (see chapter 2 for more details). A consequence is that parameters must have numerical values before computing these representations.

If you are interested in some variables in particular, which might be on a connected terminal and thus assigned latent as a default, you have to assign them manifest.

In this section we explain how to :

1. Change the values of the parameters.
2. Do manifest/latent assignment of the variables on the terminals.
3. Show the (full) behavior (equations, kernel, IO, ISO).
4. Show the properties of the system.

3.6.1 Change the values of the parameters

Before computing some of the representations, parameters need to have numerical values. If you want to see a representation and the parameters or still symbolic you will be alerted to change them. To change the values, select **Module** > **Change parameters**.

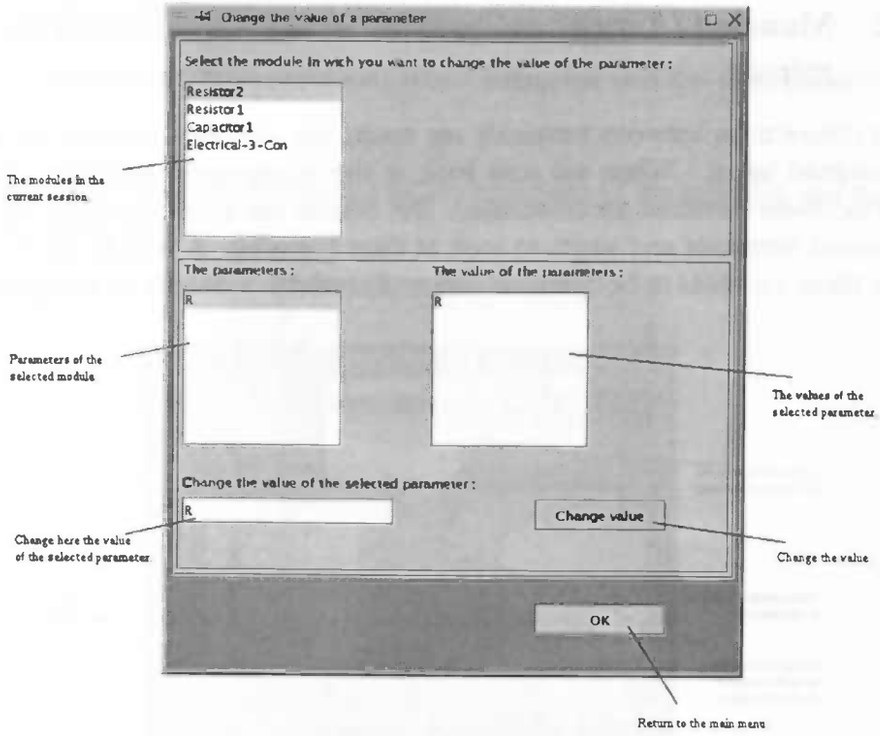

Figure 3.12: Change parameters

Select the name of the module with the symbolic parameters. Select the parameter and change the value in the textfield. In the right list the value of this parameters is shown. Repeat this for all parameters and press "OK". At another time you are able to give this parameter another (symbolic) value in the same way.

3.6.2 Manifest / Latent assignment of the variables on the terminals

When connections between terminals are made, the variables on these terminals are assigned latent. When we now look at the representations of the external behavior, these variables are eliminated. But maybe one is interested in variables on internal terminals and wants to look at their behavior. A way to do this is to assign these variables to be manifest. Select **Terminal > Man/Lat assignment**.

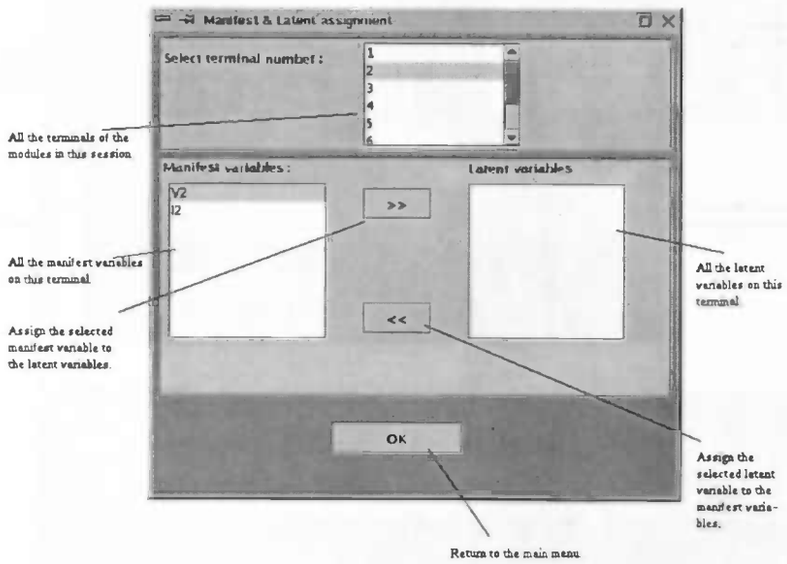

Figure 3.13: Manifest / Latent assignment

Select the terminal with the variables you are interested in. Select the variable in the right list (these are the latent ones) and push the >> button to to assign this variable manifest. To assign variables latent select them in the right list and press the << button.

3.6.3 Show Behavior of the system

To show the desired representation, select **Behavior** and the representation you want to see:

All equations. These equations are all the equations of the modules and the connection constrains.

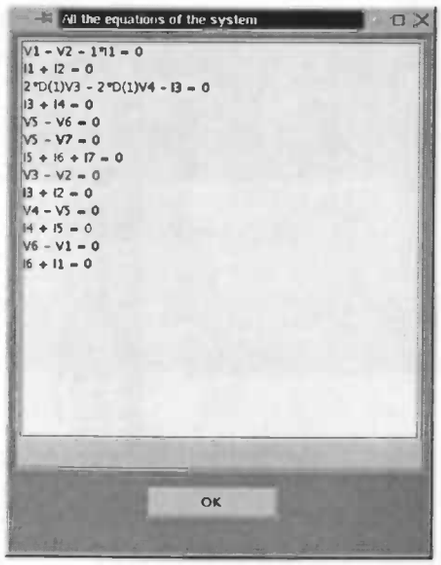

Figure 3.14: Equations of the system

Kernel representation.
This is the kernel representation $R(\frac{d}{dt})w + M(\frac{d}{dt})l = 0$ of the behavior with w the manifest and l the latent variables. Where

$$R(\tfrac{d}{dt}) = R_0 + R_1\xi + ... + R_n\xi^n \text{ and } M(\tfrac{d}{dt}) = M_0 + M_1\xi + ... + M_m\xi^m$$

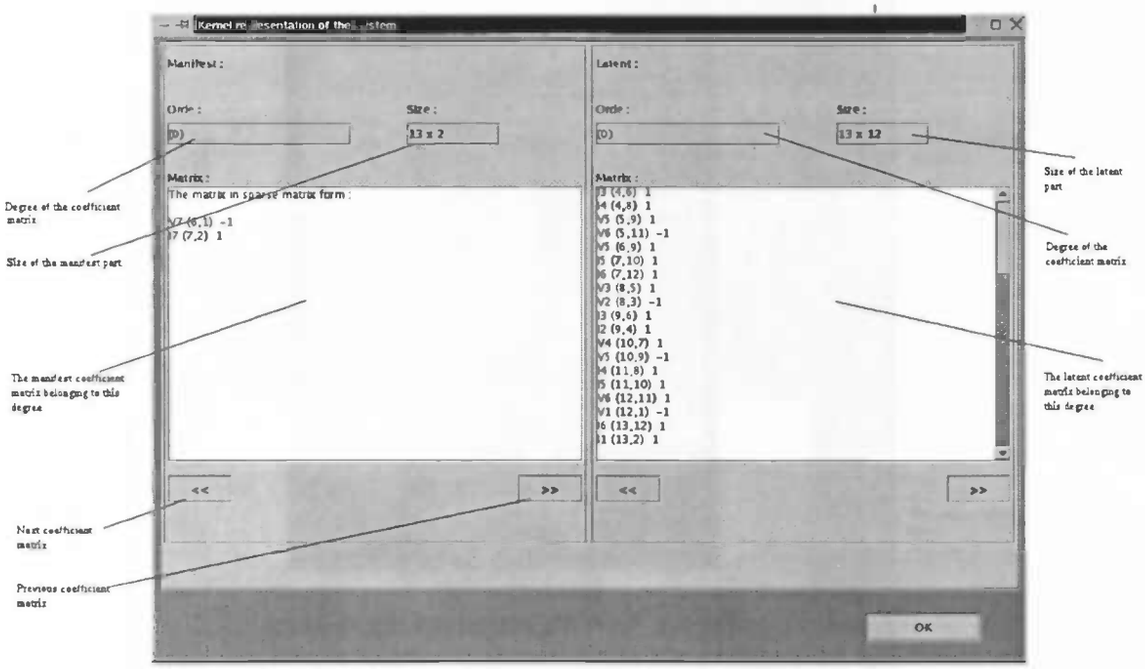

Figure 3.15: Kernel representation

At the left you find the R_i's and at the right the M_i's. Also information about the size and the order (order = i) is given. When a coefficient matrix becomes to large it is shown in sparse matrix form i.e. the positions of nonzero matrix entries are given together with the entry itself.

Minimal kernel representation.
This is the same window as above, but now with a row reduced kernel.

Eliminated kernel representation.
The kernel representation $R(\frac{d}{dt})w = 0$ of the external behavior with w the manifest variables. How to read this window we refer to kernel representation.

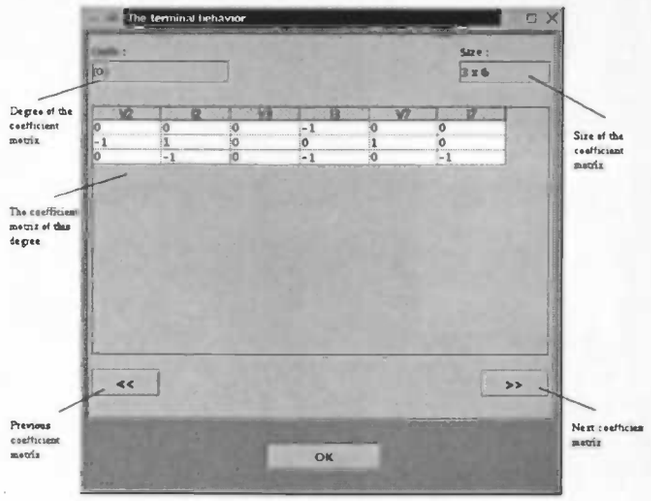

Figure 3.16: Kernel of eliminated system

Eliminated equations.
These are the equations of the external behavior.

Figure 3.17: Eliminated equations

IO representation.
An input/output representation $P(\frac{d}{dt})y = Q(\frac{d}{dt})u$ of the full behavior.

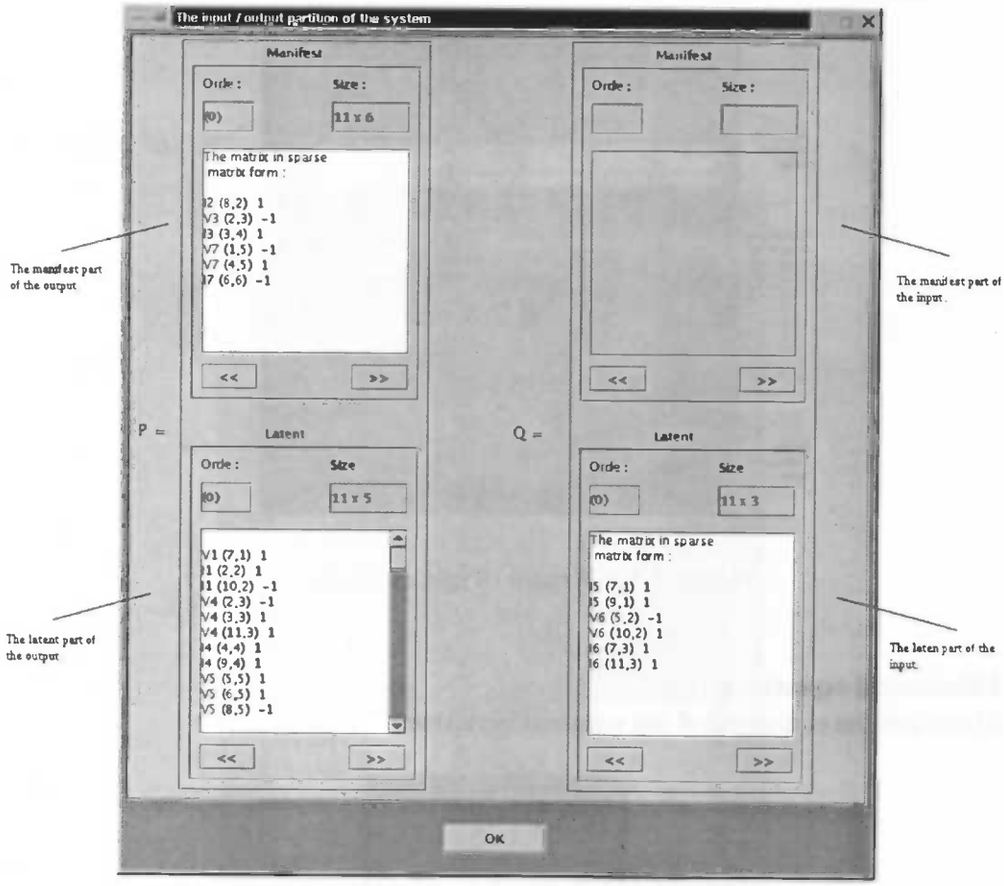

Figure 3.18: I/O representation

External IO representation.
An input/output representation $P(\frac{d}{dt})y = Q(\frac{d}{dt})u$ of the external behavior.

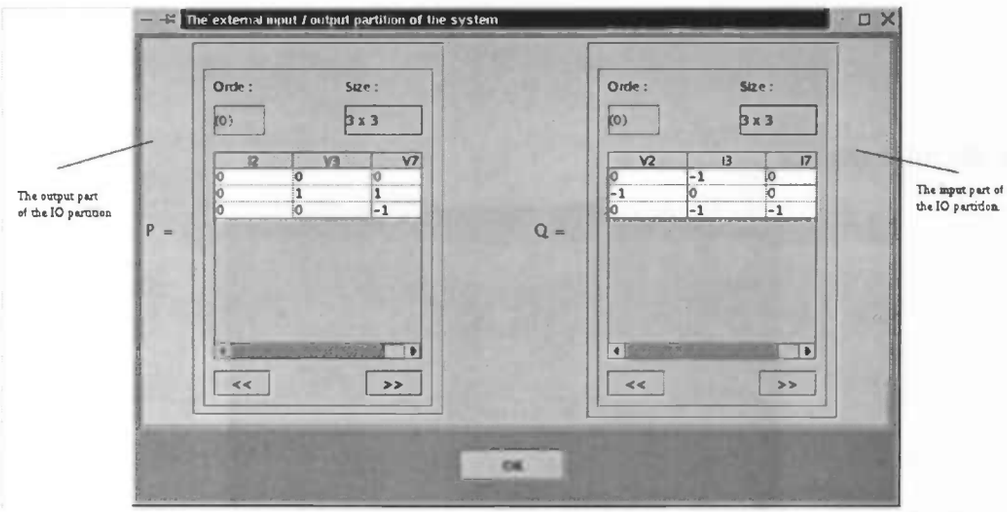

Figure 3.19: External I/O representation

ISO representation.
An Input / State / Output representation

$$\frac{d}{dt}x = Ax + Bu$$
$$y = Cx + Du$$

of the full behavior.

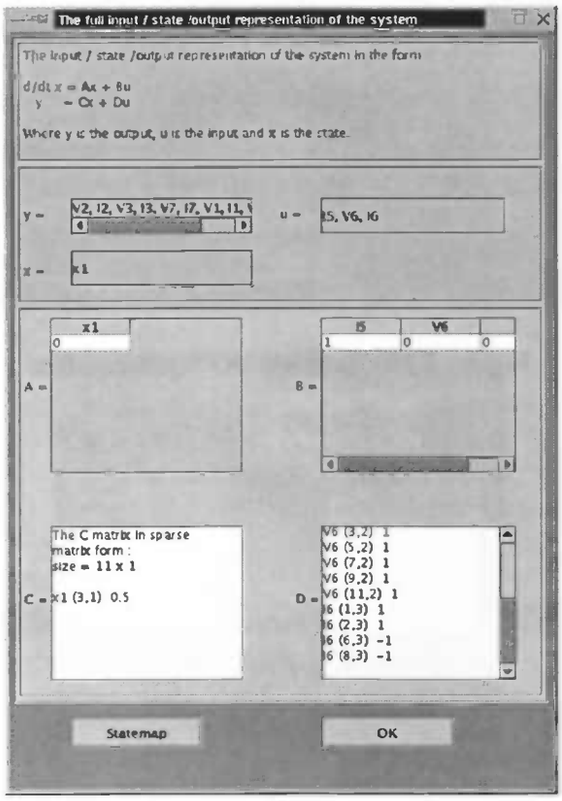

Figure 3.20: I/S/O representation

Press the 'OK' button for the state map:

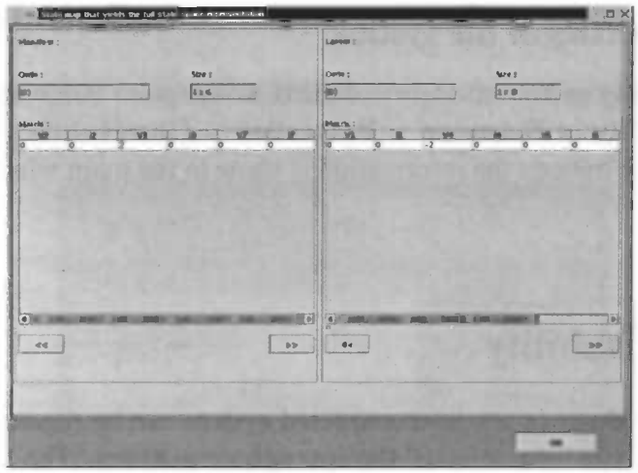

Figure 3.21: State map

The window for an **external ISO representation** is the same as for the ISO representation. Only the state map is different:

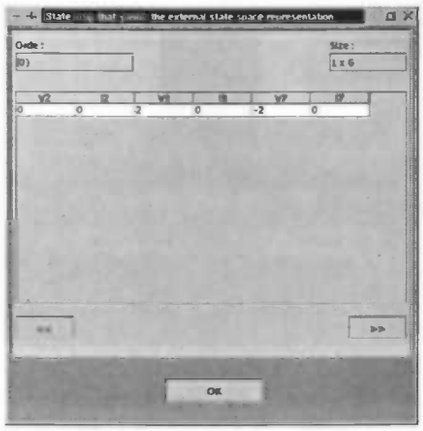

Figure 3.22: State map for external behavior

3.6.4 Properties of the system

For controllability and observability of the behavior go to **Behavior** > **Properties** > **Controllability** or **Behavior** > **Properties** > **Observability**. After you selected one of the options the information is show in the main window, see section 3.2.

3.7 Re-Usability

A couple of modules of the interconnected system can be regarded as a module itself. Suppose you have selected the appropriate modules. The terminals of this new module are those terminals of the selected modules, that are not connected with terminals of a selected module. The equations for this module are the equations of the external behavior, where the variables on its terminals are considered as manifest. Select **File** > **Re-Usability**

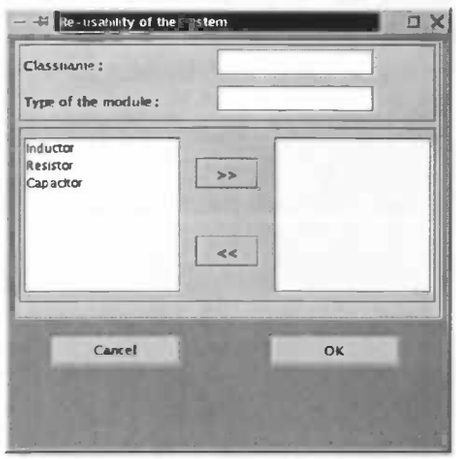

Figure 3.23: Re-usability

Provide a new class name and a type for this module. Select the modules from the current session, which will be the building blocks for the new module, and press "OK".

3.8 Simulation of the external behavior

If you want to simulate the external behavior $R(\frac{d}{dt})w = 0$, select **Simulation** > **Simulate**

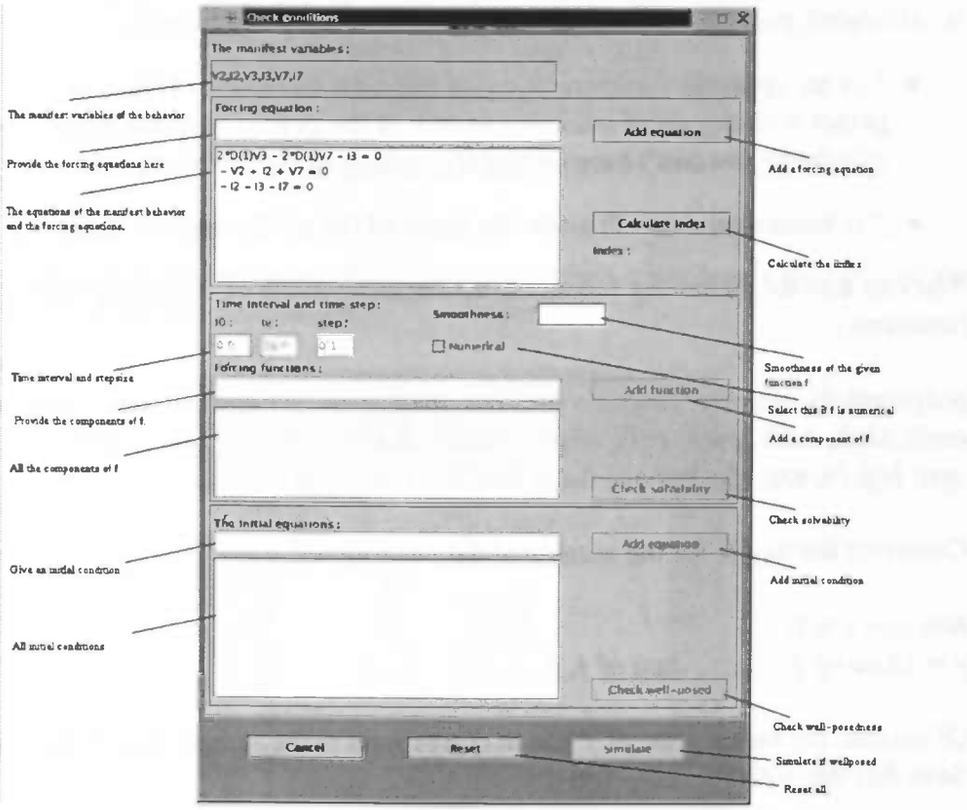

Figure 3.24: Conditions for simulation

Specify forcing equations (these are additional equations, see also chapter 2)

$$\tilde{R}(\tfrac{d}{dt})w = M(\tfrac{d}{dt})f$$

Use notation as in 3.4.3 and press "Calculate Index". Subsequently, after the index is calculated, provide the forcing functions f. We consider 2 cases:

- f is an symbolic function: Provide the time interval and the function f together with degree of smoothness of f. If the degree of smoothness is equal to infinity you don't have to specify it, or type 'Inf' in the textfield.
- f is numerical data : Provide the name of the m-file with the data of f.

You can use the following functions, and combinations of them, for the forcing functions :

polynomials, *sin, cos, tan, sec, csc, cot, asin, acos, atan, asec, acsc, acot, sinh, cosh, tanh, sech, csch, coth, asinh, acosh, atanh, asech, acsch, acoth, neg, abs, sqrt, log, ln, exp,* $\frac{d}{dx}$. Provide these functions in terms of x.

Construct the m-file for the numerical data as follows:

function y = f;
y = [data of f_1 *; ..... ; data of* f_n*];*

Of course, the sample rate of f has to correspond to the sample rate of the time. Save this file with the name f.m

After giving the forcing functions $f_1, ..., f_n$ press "Check solvability". If the system is solvable for these forcing functions, provide the initial conditions

$$S(\tfrac{d}{dt})w(0) = a$$

as equations (notation as in 3.4.3) in terms of the w variables and press "Check well-Posed". If the initial conditions are well-posed press "Simulate".

The following window appears:

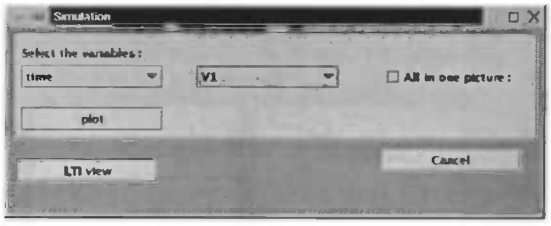

Figure 3.25: Simulation

Select the variables and press "Plot". Matlab returns a window with a plot of the trajectory of the selected variables.

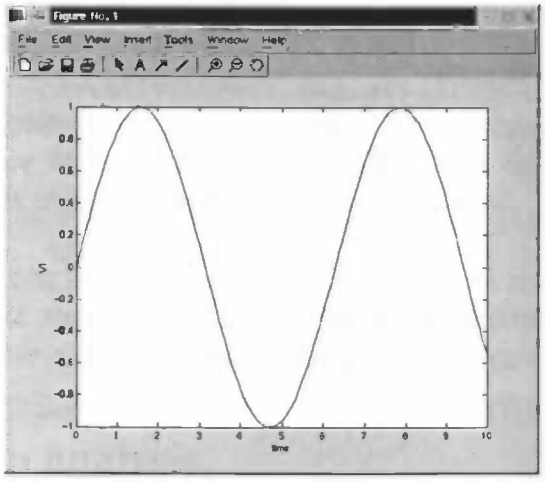

Figure 3.26: Plot of a trajectory

In this window, properties can be edited and the figure can be saved in numerous formats.

Press "LTI View" for step response, impuls response, Bode, Nyquist plot, etc...

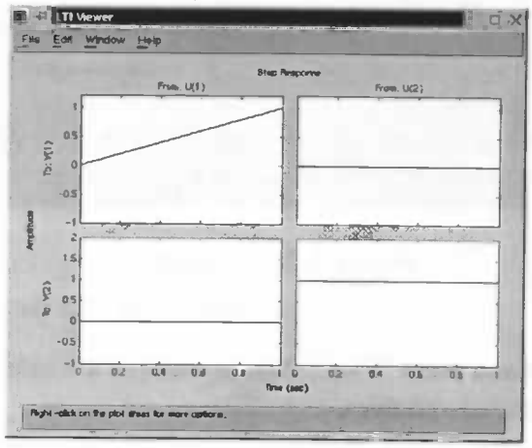

Figure 3.27: LTI View

Chapter 4

The program

There are many simulation programs in use, but almost all based regarding systems as signal processors, or using Bond-graph methodology (for instance Matlab's Simulink, Spice, Modelica and MTT). Our goal was to create a simulation tool based on "The behavioral approach" (e.g. see [1] and [3]) for more details) and modeling interconnected systems with the terminal approach (see chapter 1). Our main interest are linear systems, this doesn't mean that nonlinear systems cannot be considered, but the possibilities are more limited.

The program is written in the language JAVA. The main reason is that JAVA is an OO language (see appendix A.1). For heavy calculations we use Matlab 6. The desktop of Matlab will not appear so the user isn't aware of its use.

4.1 Program analysis.

4.1.1 Specification of the program.

Since the program is responsible for constructing an interconnected system, representing the behavior and simulating the system, it should be able to perform the following tasks:

Construct an interconnected system. First of all, the program must deal with creating terminals, modules and connections. When done, modules must be inserted in the current session and connections between terminals must be made, but also the possibility to remove a module and to disconnect terminals has to be available.

Represent the behavior. When constructed a system, the following representa-

tions for the system should be available: equations, (minimal) kernel, eliminated kernel, (external) I/O and (external) I/S/O. Also the properties like controllability and observability should be implemented.

Simulate the system. After providing some forcing equations and initial conditions, solvability and well-posedness need to be checked. When solvable and well-posed, Bode-plot, Nyquist-plot, step response etc. must be available.

4.1.2 Description of the structure.

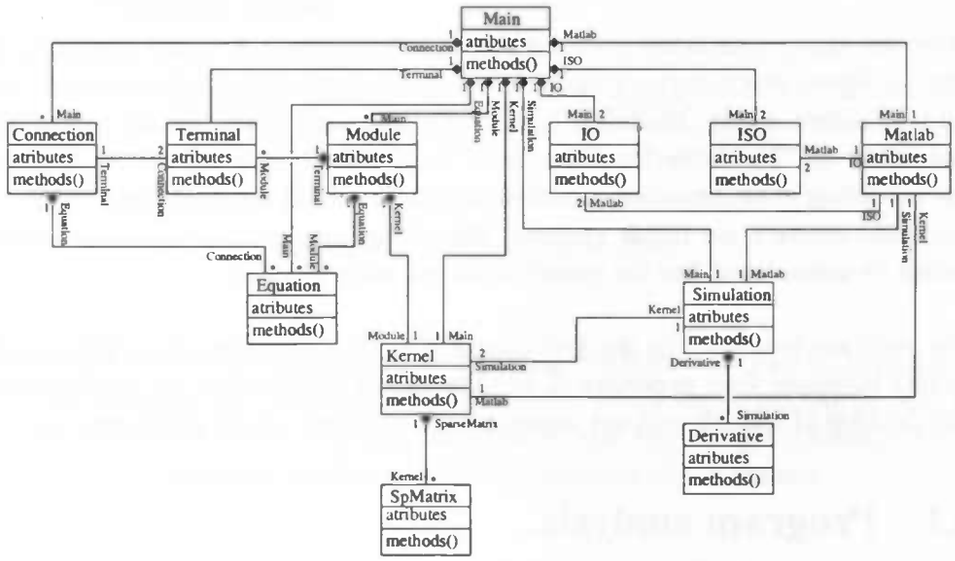

Figure 4.1: Structure of the program in UML notation

For a description of the attributes and methods see **Help > Class description** in the program .

4.2 Program design.

In the previous section we saw the structure and the specifications of the program together with the description of the methods and attributes of the classes. In this section we explain in what way the program works, using collaborations diagrams (see Appendix A).

4.2.1 Building an interconnected system.

Create a new type of terminal: As mentioned in chapter 1, a module has terminals of different types. Before one can create a module its terminal properties, type and variables, must be defined. From the defined type and variables we make separate terminals, which can be added to the module whenever the module will be created. In this way we only have to define the terminal properties once, and not every time a new module is created.

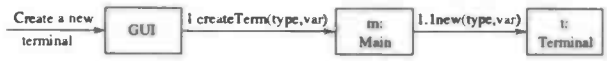

Figure 4.2: Create a terminal

Create a new equation: Every module and connection behavior is described by equations. From the provided equations in the GUI, equations are made with pointers to the module or connection.

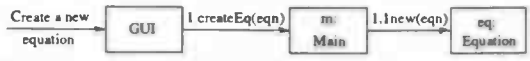

Figure 4.3: Create an equation

Create a new module class: Given terminals and equations together with type and linearity, a module can be created. From the equations and the variables on the terminals, a kernel representation for this module is made. Note that not only the main object has a kernel representation, but also every separate module (if the behavior of the module is linear).

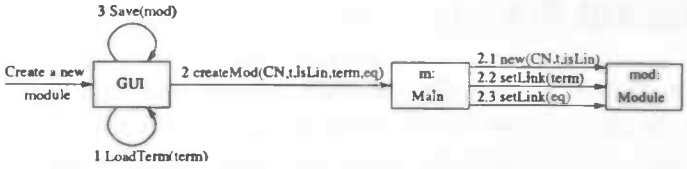

Figure 4.4: Create a module

Create a new type of connection: To connect terminals, the connection between two terminals must be defined, using the connection constraints. Given the provided equations from the GUI and the type of the to be connected terminals, a new type of connection is created.

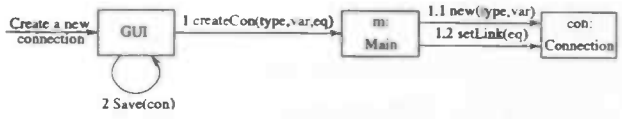

Figure 4.5: Create a connection

Insert modules in the system: After defining modules they can be inserted in the current session, by providing the class name and a name for the module in the current session. After inserting the module, the kernel representation of the system will be updated with the constitutive equations of the module.

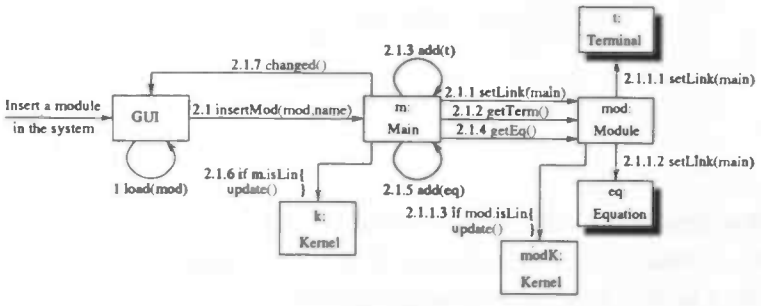

Figure 4.6: Insert a module

Connect two terminals: When inserted one or more modules, their terminals can be connected, if they are of adapted type. Afterwards the kernel representation has to be updated, since connecting terminals imposes additional restrictions to the variables.

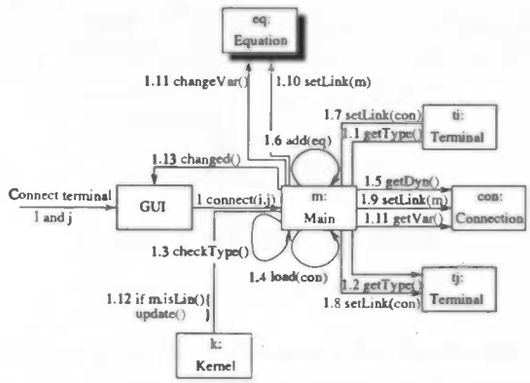

Figure 4.7: Connect two terminals

Disconnect two terminals: After connecting two terminals there is a possibility to undo this by disconnecting them.

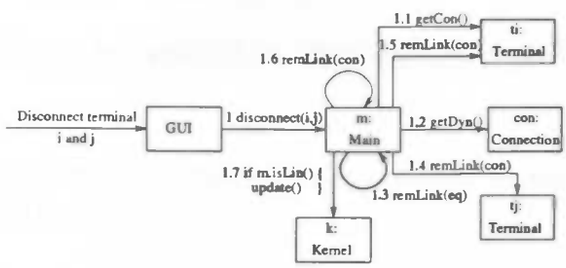

Figure 4.8: Disconnect two terminals

Remove a module from the system: When inserting modules you are able to remove a certain module from your session.

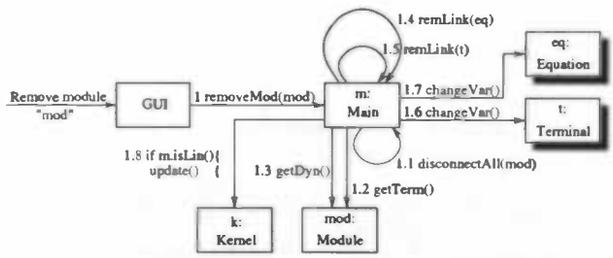

Figure 4.9: Remove a module

Disconnect all the terminals of a module: When a module object is removed form the current session, all its terminal objects will be disconnected.

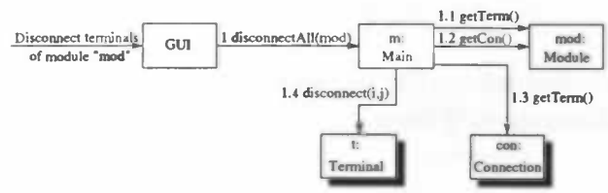

Figure 4.10: Disconnect all terminals of a module

4.2.2 Behavior of the interconnected system.

The collaboration diagrams for showing the representations and calculation of the representations are given below.

Show all the equations: These are all the equations of the modules and connections. The parameters doesn't have to be numerical values in this representation, because of the fact that we don't need Matlab for calculating this representation.

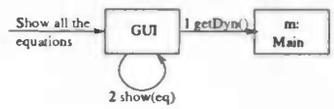

Figure 4.11: Show equations

Show the kernel representation: This is the kernel representation of the full behavior defined by $R(\frac{d}{dt})w = 0$. Also in this representation parameters doesn't have to be numerical values.

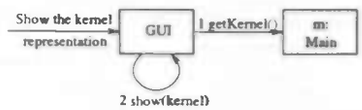

Figure 4.12: Show kernel representation

From now on parameters need numerical values because we use Matlab m-files to calculate the desired representations.

Show minimal kernel representation: This is the row reduced kernel representation $\tilde{R}(\frac{d}{dt})w = 0$ of the full behavior, with

$$U(\xi)R(\xi) = \begin{bmatrix} \tilde{R}(\xi) \\ 0 \end{bmatrix}$$

and $\tilde{R}(\xi)$ of full row rank and $U(\xi)$ an unimodular matrix.

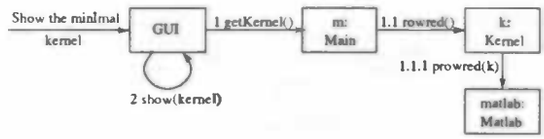

Figure 4.13: Show minimal/row reduced kernel representation

Show eliminated kernel representation: This is the *row reduced* kernel representation $R(\frac{d}{dt})w = 0$ of the manifest behavior, . We use the Matlab file *eliminate.m* to calculate this representation (compare section 2.2)

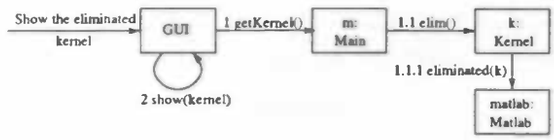

Figure 4.14: Show eliminated kernel representation

Show the equation of the eliminated system: From the kernel representation of the manifest behavior we build equations for the manifest behavior.

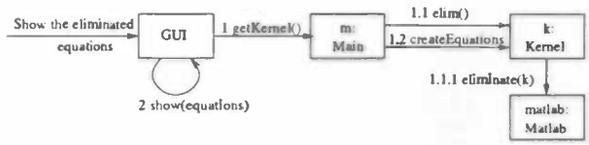

Figure 4.15: Show equations of eliminated kernel representation

Show I/O representation: As discussed in section 2.4.1 the behavioral equations $R(\frac{d}{dt})w = 0$ of the full behavior can be written as $P(\frac{d}{dt})y = Q(\frac{d}{dt})u$ the input/output form of the system, where w can be partitioned as

$$w = \begin{bmatrix} u \\ y \end{bmatrix}$$

We use Matlab for calculating P,Q and the I/O partition.

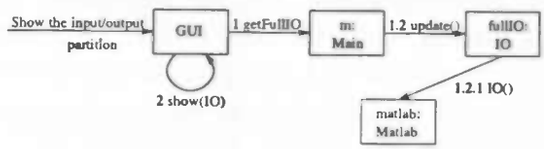

Figure 4.16: Show IO representation

Show external I/O representation: We saw that the full behavior can be written in input/output form, but this can also be done for the manifest behavior.

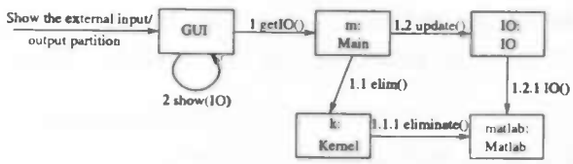

Figure 4.17: Show external IO representation

Show I/S/O representation: Every behavior described by $R(\frac{d}{dt})w = 0$ admits an i/s/o representation of the form

$$\frac{d}{dt}x = Ax + Bu$$
$$y = Cx + Du$$

with u the input, x the state and y the output (section 2.4.2).

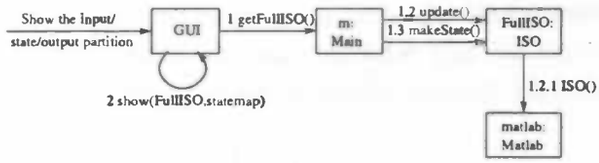

Figure 4.18: Show ISO representation

Show external I/S/O representation: Also for the external behavior an i/s/o representation can be calculated.

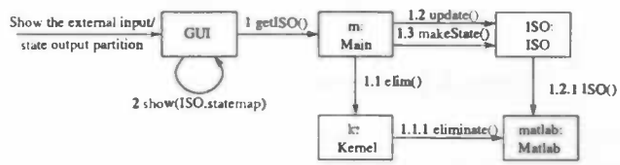

Figure 4.19: Show external ISO representation

Provide properties of the system: When you build a dynamical system you might be interested in some properties of this system. We implemented three properties namely:

1. Controllability of the full behavior.
2. Controllability of the manifest behavior.
3. Observability of the full behavior.

See also section 2.3.2 and section 2.3.1.

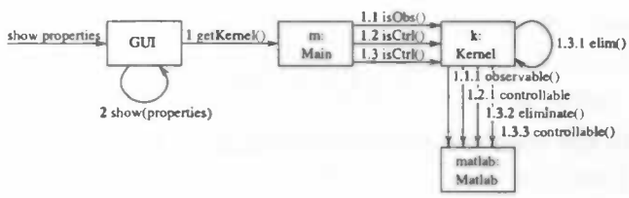

Figure 4.20: Properties

In the future some properties as stability and detectability will be implemented.

4.2.3 Simulation.

If you want to simulate the behavior $R(\frac{d}{dt})w = 0$, you need to specify the forcing equations

$$\tilde{R}(\frac{d}{dt})w = M(\frac{d}{dt})f$$

together with the initial conditions

$$S(\frac{d}{dt})w(0) = a$$

Given the forcing functions f Matlab calculates the index of the the behavior, this index relates the smoothness of the given f to the smoothness of the solution w. If the provided f is smooth enough Matlab checks if the behavior

$$\begin{bmatrix} R(\frac{d}{dt})w \\ \tilde{R}(\frac{d}{dt})w \end{bmatrix} = \begin{bmatrix} 0 \\ M(\frac{d}{dt})f \end{bmatrix}$$

is solvable for this f. If so well-posedness of the initial conditions $S(\frac{d}{dt})w(0) = a$ is calculated. When the initial conditions are well-posed we are able to simulate the trajectories w. See also section 2.5

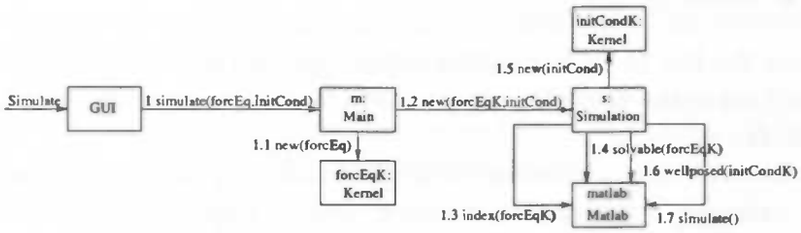

Figure 4.21: Simulation

Appendix A

OOP and the Universal Modeling Language

In this section we explain the *Object Oriented Programming* approach using *UML*, short for *Universal Modeling Language*. We define *objects*, which are essential elements in OOP, and also *classes* and *inheritance*.

A.1 Definitions

Objects are the key to understanding object-oriented technology. You can look around and see many examples of real-world objects: a car, a desk, the television set, a bicycle.
These real-world objects share two characteristics: they all have *state* and *behavior*. For example, dogs have state (name, colour, breed, hungry) and behavior (barking, fetching, and wagging tail). Bicycles have state (current gear, current pedal cadence, two wheels, number of gears) and behavior (braking, accelerating, slowing down, changing gears). But also a desk has (trivial) behavior: doing nothing.
Software objects are modeled after real-world objects in that they too have state and behavior. A software object maintains its state in one or more variables called *attributes*. A variable is an item of data named by an identifier. A software object implements its behavior with *methods*. A method is a function (subroutine) associated with an object. You can communicate with an object by sending it a message, a so called *method call*. There is no other communication possible. This is illustrated in figure A.1. This leads to the following definition:

Definition A.1.1 *An object is a software bundle of variables and related methods.*

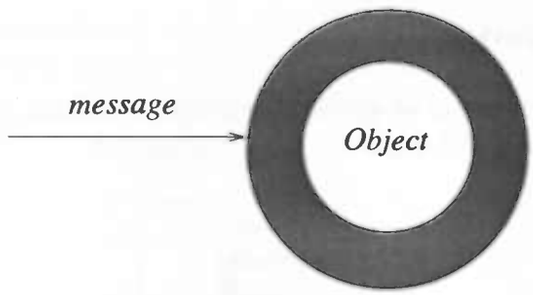

Figure A.1: Method call to object

You could see objects as *agents*, each of them with its own speciality. The agent is ordered to do something, the method call, and he carries out his task possibly by asking things to other agents as well.

If you think about it, some objects have things in common. In chapter 1, for instance, we encountered *module classes*. The members of this class share a couple of things: they all have terminals, behavior, etc. But for every module individually, these terminals and behavior may differ. In general, objects of a certain class possesses the same methods but a different state, although each member of a class can have additional methods and additional attributes. This gives rise to the following definition:

Definition A.1.2 *A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind.*

This can be more generalized. Classes itself may also have things in common. For example, we could group electrical modules, 1D-mechanical modules, etc, inducing an *electrical module class*, a *1D-mechanical module class, etc*. But they all are modules. We call these classes *subclasses*.

An example. Take the class bicycles. Each bicycle object has two wheels, pedals etc. But some bikes have gears. The bikes with gears form a subclass of the class bicycles. A gear bike has attributes that a bike without a gear doesn't have. For instance, the current gear attribute. The method change gear is also not present for a general bike.
We say that members of a subclass *inherit* the attributes and methods. The notion of inheritance plays an important role in Object Oriented Programming.

A.2 UML notation

UML stands for *Universal Modeling Language*, a visualization for OO-analysis and OO-design where OO stands for *Object Oriented*.

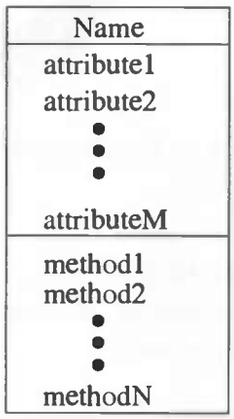

Figure A.2: Visualization of a class

A class is visualized by a rectangle which is divided into three sections. The first section contains the name of the class. The second section contains the attribute identifiers and the third section the identifiers of methods. Most literature mix up the concept of attribute identifier and attribute. The same holds for methods. Subclasses are visualized as follows:

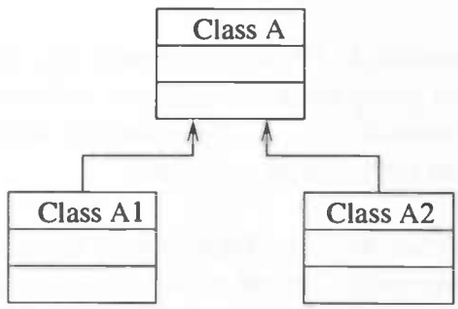

Figure A.3: Gen-Spec structure

Class A1 and A2 are subclasses of class A. They inherit all attributes and methods defined in class A and they have additinal attributes and/or methods.

There is also this notion of whole-part relation. This simply means that an object is part of another object. For example, terminals are part of a module. UML-notation:

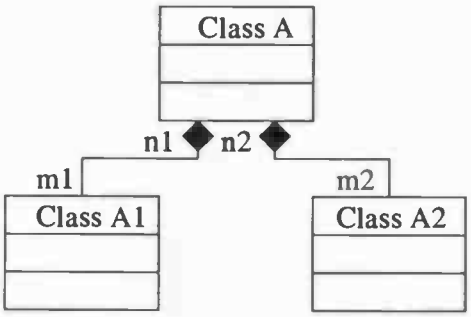

Figure A.4: whole-part relation

The symbols in this figure reflects the following: $m1$ objects of class A1 are part of $n1$ objects of class A, etc. This symbols are, for instance, nonnegative integers. But also other symbols are used: * means zero or more, + means 1 or more.

We could ask ourselves how these object interact with eachother. In *UML* this is depicted with *collabotation-diagrams*. Such a diagram shows the evolution of a message call to an object, this is called an *input event*. This object itself can send a message to another object, and so on.

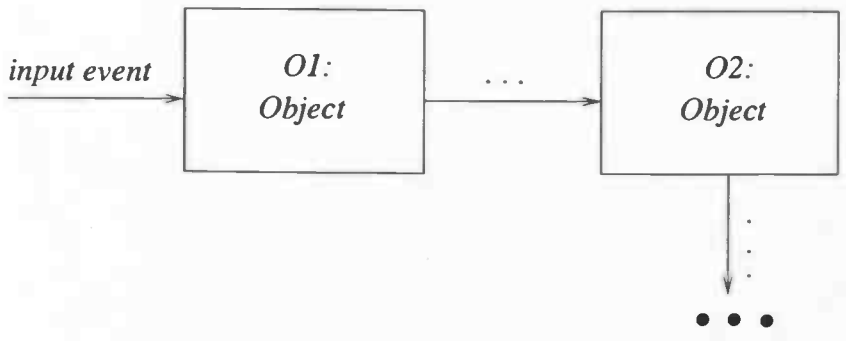

Figure A.5: collaboration diagram