

A MULTI-DOMAIN FUNCTIONAL DEPENDENCY MODELING TOOL
BASED ON EXTENDED HYBRID BOND GRAPHS

By

Zsolt Lattmann

Thesis

Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

May, 2010

Nashville, Tennessee

Approved:

Professor Gabor Karsai

Professor Gautam Biswas

ACKNOWLEDGEMENT

This work was supported under the DRAFTS Program, Airbus UK LTD on “Dependability and Risks Assessment Frameworks Tool Sets”. Their support is acknowledged.

I would like to thank my advisor, Professor Gabor Karsai for his support, instruction, and advice during this project.

I would like to thank Professor Gautam Biswas for his help.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENT.....	ii
TABLE OF CONTENTS	iii
LIST OF TABLES.....	iv
LIST OF FIGURES.....	v
Chapter	
I. INTRODUCTION.....	1
<i>Problem statement</i>	3
II. BACKGROUND.....	5
<i>Bond graphs</i>	5
Basics	5
Modulated elements and switched junctions.....	11
Vector bond graphs.....	13
Hierarchical design.....	13
Differential equations from bond graphs	13
<i>Model Integrated Computing</i>	14
III. DESIGN OF THE MODELING LANGUAGE.....	17
IV. DESIGN OF THE MODEL INTERPRETER	23
V. CASE STUDY	28
VI. SUMMARY AND CONCLUSIONS.....	32
BIBLIOGRAPHY	34

LIST OF TABLES

Table	Page
II.1: Power and energy variables in different domains.....	5
III.2: Legal and illegal signal connections	22
IV.3: Simulink models of the bond graph elements.....	27

LIST OF FIGURES

Figure	Page
I.1: Hierarchical design structure	2
II.2: The two different causal stroke	6
II.3: Relationship between the power and energy variables	6
II.4: Causal form of sources.....	7
II.5: Resistor’s causal form and characteristics.....	8
II.6: Causal form and characteristics of capacitors	9
II.7: Causal form and characteristics of inertias.....	9
II.8: Transformer causal graph	10
II.9: Gyrator causality graph.....	10
II.10: Junction types and their equations.....	11
II.11: Modulation signal is connected to bond graph elements	11
II.12: Example for modulated transformer	12
II.13: ON state with equations, OFF state with equations.....	12
II.14: Example for creating differential equation from bond graph	14
II.15: Meta-modeling example [7]	15
III.16: FDM tool	17
III.17: Meta-model of FunSketcher	18
III.18: State machine model of the junctions.....	20
IV.19: Power propagation example.	24
V.20: System diagram	28
V.21: Bond graph of the system in GME.....	29
V.22: Generated Matlab Simulink model.....	30
V.23: Step response of the system	31

CHAPTER I

INTRODUCTION

Modeling physical systems is necessary to understand how they work and how the compositions of physical components (i.e., complex physical systems) work. Modeling is used for prediction, design, and operation. Simple components or simple physical systems can be modeled with modeling languages that have modeling elements on the order of 10. For instance, bond graphs can be used for modeling physical systems and they have only nine basic elements. However, complex physical systems can have many interconnected components (e.g. a landing gear system of airplanes has 80-100 elements), which are also physical systems. The number of components (including physical, hardware, and software components) rapidly grows in a particular subsystem during the design time. This means that number of interconnections and the complexity also grows during subsystem design.

Therefore, modelers need support for hierarchical modeling because using only the atomic elements of the modeling language would result in a very large number of atomic elements and their connections on the same level. This is beyond the scalability boundary of graphical languages; normally the accepted limit is 50 icons per screen. Using hierarchy, modelers can define smaller components with ports. Ports are gateways between two levels in the hierarchy. Models that contain components, these also have ports, and ports are called subsystems. Subsystems can be connected to other elements through their ports, or ports in a higher level subsystem that contains the composition of the elements. The hierarchical structure of an example system is shown in Figure I.1.

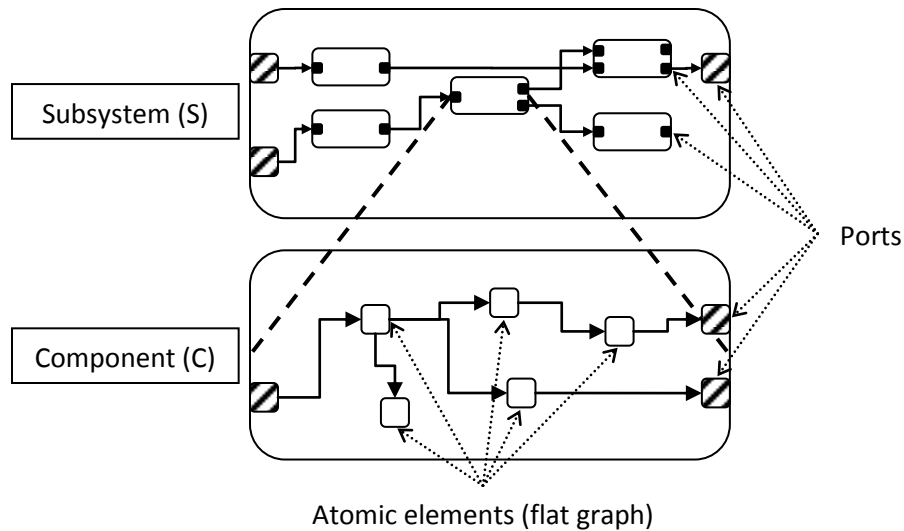


Figure I.1: Hierarchical design structure

When modelers design a complex physical system, they use different physical domains: hydraulic, electrical, mechanical or thermal. Each domain has different elements and components with descriptions of different physical laws (such as Kirchoff's Law, Newton's Law, Pascal's Law). Modelers must create well-defined mathematical models for each domain. For instance, the mathematical model of an electrical capacitor can be written as Equation 1, where the voltage on the capacitor can be computed at every instant of time by the integrating of the current on the capacitor, then dividing it with the capacitor value and adding the initial voltage.

$$v(t) = \frac{1}{C} \int_{t_0}^t i(\tau) d\tau + v(t_0) \quad (1.)$$

Using those models, the modeler obtains a set of equations. However, when the system has certain kind of switching elements, electrical switch or hydraulic valve, the equations might depend on time. A better approach is to use those mathematical models as atomic elements of the system and create the composition of them. Each component has its mathematical description, which is used to calculate the states and the independent variables of the system. The modeler can visualize the composition of elements and components with a visual layout designer (such as Generic Modeling Environment GME

[4]). If the modeler specifies the initial conditions and parameters of each atomic element or component, the system can be simulated and analyzed.

According to the requirements above, the basis of the visual domain specific modeling language that we have chosen is an extended hybrid bond graph modeling language. Bond graphs are a domain independent modeling formalism [1][8]; any physical domain can be modeled using the same elements and connections, but with different domain-dependent physical meaning. When modelers create a bond graph model for a specific component, they also create the visual representation of the mathematical equations of the component because bond graphs can be rewritten into differential equations. However, hybrid bond graphs are not sufficient for modeling complex physical systems, for instance, sensing and actuation is not possible in classical bond graph terminology. Therefore, we extended our language to include sensing and actuation together, with hierarchy, as previously described, data processing elements, and controllers. Hierarchy can provide for visualizing different details on different levels. Given an engineering tool, models can be visualized, edited, created, and managed graphically.

Problem statement

1. In this thesis we will describe a Domain Specific Modeling Language (DSML), which can be used to create and edit physical models and their controlling functions in different domains. Our DSML is called Functional Dependency Model (FDM), and it is based on extended hybrid bond graphs.
2. Bond graphs and the connected elements in the bond graphs have predefined syntax and semantics. Syntactic and semantic concepts determine the design rules. Design rules define the legal compositions of the elements and rules for the values of specific attributes. Modelers can create only syntactically and semantically correct models. Therefore, the design rules shall be validated by a constraint checker and/or a model interpreter.

3. Another design rule is that the bond graph shall be correct regarding the causality directions. The interpreter generates a causality graph to show the causality directions and the causality problems to the modeler. The interpreter uses the power propagation between the components to perform causality checks.
4. If the model seems correct to the interpreter and the constraint checker, the interpreter shall create a MATLAB script to build a Simulink [10] model according to the FDM model and its hierarchy.

CHAPTER II

BACKGROUND

Bond graphs

Basics

The discussion below follows the book on System Dynamics [1]. Bond graphs provide a modeling formalism based on basic elements with ports and interconnections between ports. Interconnections represent power flows between the elements, which elements can be of type 1-port, 2-port, and n-port. Ports can be connected together via connections called bonds. The bond connections are lines that end with a half arrow. The power sources are always on the beginning of arrows and the power sinks are always the end of arrows. Each bond represents two variables: the generalized effort and flow variable. The power is the product of the effort and the flow variables, which have different physical meaning and represent different quantities in different domains. For example, in the electrical domain the effort can be the voltage and the flow can be the current. Table II.1 contains the identification of the effort and flow variables in several domains.

Table II.1: Power and energy variables in different domains

		Generalized variables			
		Effort, e	Flow, f	Momentum, $p=fe$	Displacement, $q=ff$
Domain	Mechanical translation	Force, F	Velocity, V	Momentum, P	Displacement, X
	Mechanical rotation	Torque, τ	Angular velocity, ω	Angular momentum, p_τ	Angle, θ
	Hydraulic	Pressure, P	Volume flow, Q	Pressure momentum, p_p	Volume, V
	Electrical	Voltage, e	Current, i	Flux linkage variable, λ	Charge, q
	Thermal	Temperature, T	Entropy flow, f_s		Entropy, S

The direction of effort and flow is indicated by the causal stroke (see Figure II.2) of the connection. Effort is always above or on the left hand side of a connection. Flow is always below or on the right hand side of a connection. The causal stroke sign can be on either side of the connection. Figure II.2 shows the causal strokes in different cases and the direction of effort and flow.

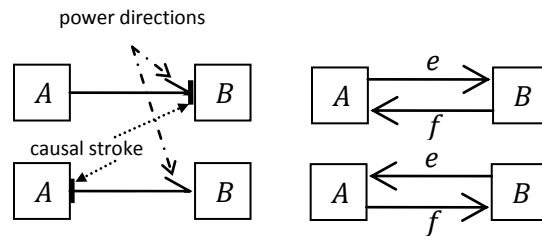


Figure II.2: The two different causal stroke

Bond graphs show the power directions, but they are the same as the energy directions (see Figure II.2). Figure II.3 summarizes the power and energy variables and their mathematical relationship between them, which is true at every instant of time. The variables are also referred to as (generalized) effort (e), flow (f), momentum (p), and displacement (q).

$$\begin{array}{ccc}
 e & \xrightarrow{\int dt} & p \\
 \xleftarrow{\dot{p} = e} & & \\
 f & \xrightarrow{\int dt} & q \\
 \xleftarrow{\dot{q} = f} & &
 \end{array}$$

Figure II.3: Relationship between the power and energy variables

Bond graph elements are described next, including 1-port elements (such as sources, resistors, capacitors, and inertias), 2-port elements (such as transformers and gyrators), and junctions.

Simple 1-port elements generate power, dissipate energy or store energy. Power generation is represented by a power source, which can be either source of effort or source of flow. A resistor

element dissipates energy. Storage elements are (generalized) capacitance and inductance elements that can store either effort or flow.

In the bond graph terminology, there are two types of power sources: source of effort (S_e) and source of flow (S_f). The power sources produce constant efforts or constant flows. The produced flow and effort does not depend on other elements of the system, assuming ideal sources. For example, a car battery can be modeled as a source of effort using bond graph language. The voltage (i.e. effort) of the battery is a constant 12 V and does not change when the load is changing. This assumption is not satisfied for real systems. The effort sources can be voltage supplies, pressure sources, and force sources such as gravity. The flow sources can be current supplies, hydraulic flow sources, and velocity sources such as a wall, where $v = 0 \frac{m}{s}$. The causal direction is predefined on source elements (see Figure II.4).

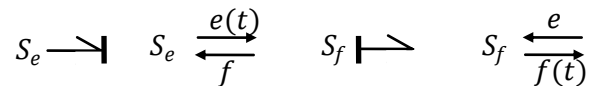


Figure II.4: Causal form of sources

Resistors represent a characteristic relationship between flow (f) and effort (e) variables. The characteristics of the resistors can be linear or nonlinear. In a special case, when the resistor is linear, the resistor's value is called resistance and its inverse is called conductance. Power flows into the resistor and the resistor dissipates energy according to its characteristics. Using the power convention, the half arrow is pointing towards the (passive) resistors. Parameters of resistors are the electrical resistance in the electrical domain or mechanical damper coefficient in the mechanical domain. The hydraulic resistor can be a pipe, which is defined with nonlinear parameterized equations; pipes cannot be described with a single resistor parameter. The causal stroke can point in either direction on a resistor (see Figure II.5).

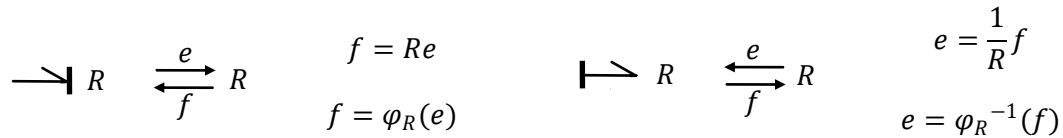


Figure II.5: Resistor's causal form and characteristics

Ideal storage elements can store energy without loss. In other words, they are the 'memory' of the system. The effort storage element is called (1-port) capacitor or compliance and the flow storage element is referred to as (1-port) inertia in bond graph terminology. Causality determines the direction of the effort and the flow. Therefore, storage elements have two different types of causality: integral causality and derivative causality. In physical systems, the preferred one is the integral causality. When a storage element has an integral causality, the system has an independent variable or state of the system. When a storage element has derivative causality, the system does not have any other independent variables or states.

Capacitors represent a characteristic relationship between the effort (e) and the displacement (q). Also, this element has two different types, linear and nonlinear, according to the relationship between the effort and displacement. 1-port capacitors include electrical capacitors, springs, torsion bars, hydraulic accumulators, and gravity tanks. Figure II.6 shows the causal strokes and characteristic equation of capacitors.

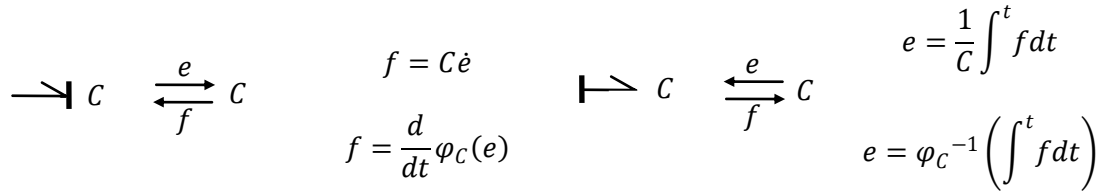


Figure II.6: Causal form and characteristics of capacitors

Inertias represent a characteristic relationship between the flow (f) and the momentum (p) (see Figure II.7). Also, this element has two different types, linear and nonlinear, according to the relationship between the flow and momentum. 1-port inertias can be electrical inductances, masses, and fluid inertias.

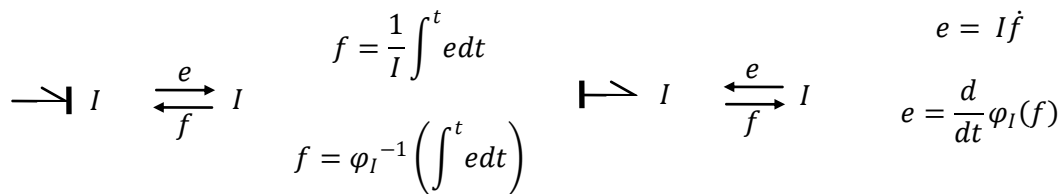


Figure II.7: Causal form and characteristics of inertias

Ideal 2-port elements are used for power conversion between the input and the output ports of the element. This conversion is ideal: the element does not have power loss. Equation 2 must be satisfied for every instant of time, according to the rule of the power conversion without power loss. There are two types of 2-port elements in bond graph terminology: transformer and gyrator.

$$e_1(t)f_1(t) = e_2(t)f_2(t) \quad (2.)$$

Transformers can be electrical transformers, rigid levers, gear pairs and hydraulic rams. Transformers have a parameter m , which is called the transformer modulus. Equation 3 shows the constitutive law between effort and flow. The subscript notations on the equations are used to differentiate between the two sides of the transformer (see Figure II.8).

$$e_1 = me_2, \quad f_2 = mf_1 \quad (3.)$$



Figure II.8: Transformer causal graph

Gyrators usually convert power between different domains and they can be electrical gyrators, mechanical gyrators (converting power from mechanical rotation domain to mechanical displacement domain, for example), and voice coil transducers (from mechanical displacement domain to electrical domain). Gyrators have a parameter r , which is called gyrator modulus. Equation 4 shows the constitutive law between effort and flow. The subscript notation is used to differentiate between the two sides of the gyrator (see Figure II.9).

$$e_1 = rf_2, \quad e_2 = rf_1 \quad (4.)$$



Figure II.9: Gyrator causality graph

Bond graphs have two different junction types that make it possible to connect bond graph elements, and elements can be connected only through junctions. The first type of junctions is the 0-junction, which has two constraints: efforts are the same on all the connected bonds, and the summation of the flows on those bonds is zero. For example, 0-junction represents parallel connection in electrical domain. The second type of junctions is the 1-junction, which has two constraints: flows are the same on all connected bonds and the summation of the efforts is zero. For instance, 1-junction represents serial connection in electrical domain. Junctions have 2 or more connected bonds.

$$\begin{array}{ccc}
\frac{e_1}{f_1} & 0 & \frac{e_3}{f_3} \\
e_2 \Big| f_2 & &
\end{array}
\quad
\begin{array}{l}
e_1 = e_2 = e_3 \\
f_1 + f_2 + f_3 = 0
\end{array}
\quad
\begin{array}{ccc}
\frac{e_1}{f_1} & 1 & \frac{e_3}{f_3} \\
e_2 \Big| f_2 & &
\end{array}
\quad
\begin{array}{l}
f_1 = f_2 = f_3 \\
e_1 + e_2 + e_3 = 0
\end{array}$$

Figure II.10: Junction types and their equations

Modulated elements and switched junctions

Some of the bond graph elements, called modulated elements, can be modulated with a signal, and are used to model non-linear elements. Signals represent a flow of information without any energy meaning. Signals can modify the parameter values of the elements or can switch junctions. Bond graphs, that contain switched junctions, are called hybrid bond graphs [14]. The bond graph terminology assumes that power is not needed for transferring, creating, and using signals. Signals are powerless data links, which can carry any kind of data: numbers, Boolean values, vectors and matrices. The modulated elements have the letter *M* in front of the name of the elements (i.e. MS_e , MS_f , MTF and MGY). Some examples for the modulated bond graph elements are shown in Figure II.11.

$$\begin{array}{cccc}
\begin{array}{c} e(t) \downarrow \\ MS_e \end{array} & \begin{array}{c} f(t) \downarrow \\ MS_f \end{array} & \begin{array}{c} m(t) \downarrow \\ MTF \end{array} & \begin{array}{c} r(t) \downarrow \\ MGY \end{array} \\
\frac{e_1 = e(t)}{f_1} & \frac{e_1}{f_1 = f(t)} & \frac{e_1}{f_1} \quad \begin{array}{l} e_2 = m(t)e_1 \\ f_2 = \frac{1}{m(t)}f_1 \end{array} & \frac{e_1}{f_1} \quad \begin{array}{l} e_2 = r(t)f_1 \\ f_2 = \frac{1}{r(t)}e_1 \end{array}
\end{array}$$

Figure II.11: Modulation signal is connected to bond graph elements

The modulated effort source (MS_e) can be, for instance, a sinusoidal voltage in the electrical domain. Modulated transformer (MTF) is usually kinematic linkage or geometric transformation in the mechanical domain. A simple example is a rotating arm around a pivot, which is on one side of the arm (see Figure II.12). The input is the force on the end of the arm, and the output is the torque around the pivot. In this case, the mechanical displacement domain is the input and the mechanical rotation domain is the output of the system. This system can be modeled as a MTF according to the domain transformation and the relationships between the force/torque and the velocity/angular velocity.

Assume that the force is vertical. The parameter value of the transformer r depends on the causality and r is equal to either $l \cos \theta$ or $\frac{1}{l \cos \theta}$, where l is the length of the rod and θ is the angle between the horizontal axis and the arm.

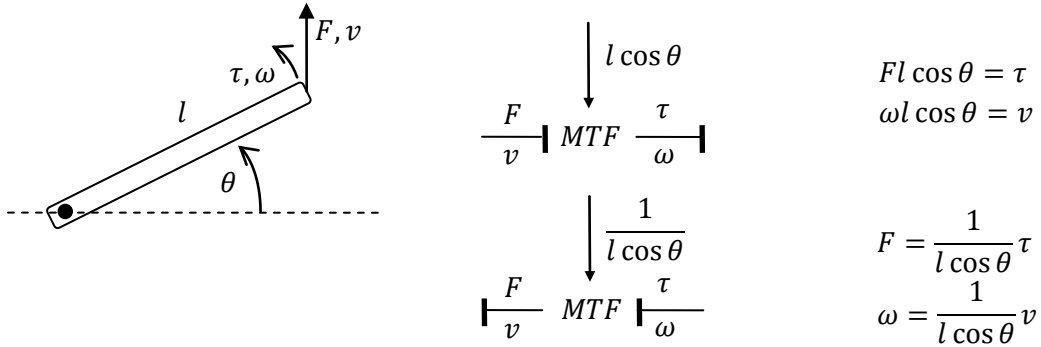


Figure II.12: Example for modulated transformer

Signals must carry a Boolean value (i.e. TRUE or FALSE) when they are connected to a junction element, called the switched junction [2]. When the connected signal carries a TRUE value the junction is in the ON state and works as described above. When the connected signal carries a FALSE value the junction is in the OFF state. Therefore, the effort and the flow variables must have zero values on all connected bonds as shown on Figure II.13.

	<i>ON State</i>	<i>OFF State</i>
$\frac{e_1}{f_1} \quad 0 \quad \frac{e_3}{f_3}$ $e_2 \mid f_2$	$e_1 = e_2 = e_3$ $f_1 + f_2 + f_3 = 0$	$e_1 = e_2 = e_3 = 0$
$\frac{e_1}{f_1} \quad 1 \quad \frac{e_3}{f_3}$ $e_2 \mid f_2$	$f_1 = f_2 = f_3$ $e_1 + e_2 + e_3 = 0$	$f_1 = f_2 = f_3 = 0$

Figure II.13: ON state with equations, OFF state with equations

Vector bond graphs

Vector bond graphs [11] have multidimensional bonds, and these bonds carry as many variables between elements as many ports they have. Vector bond graphs are an extension of the bond graph terminology. Since bonds have multiple variables the one-port elements are defined as a field. Therefore, vector bond graphs have different fields: R-field, I-field, C-field, IC-field, multiport TF, multiport GY and junction structures. Parameters of multiport bond graph elements are vectors or matrices. Those elements and their parameters simplify the modeling of multidimensional dynamics.

Hierarchical design

Complex physical systems would have to be described by very large graphs if we used only flat bond graphs. This is a scalability problem that always appears when the modeling language supports only flat graphs. By extending the bond graphs with hierarchy, the graphs can be reduced to fewer, simpler pieces and their connections. This way, the components can be connected to each other through their ports. Hierarchical modeling promotes reuse: when the modelers create components, for example a real voltage supply with internal resistor, another modeler can use that. Combining several components takes less time than creating and editing one large flat model for the same system. Therefore, following the hierarchical approach, system models are more comprehensible for other engineers. Using the hierarchical design, engineers can create reusable components for further modeling.

Differential equations from bond graphs

Bond graph models can be transformed to differential equations only after the causality algorithm [1] has been completed on the graph. The causality algorithm determines the physical causal directions (i.e. causal strokes) on all the bonds. The algorithm will be described in the section on the design of the model interpreter. Storage elements, that have integral causality, indicate the state variables of the differential equations. The relationships between the state variables are described by the characteristic

equation of the elements. Figure II.14 shows a simple example how one can create the differential equations from the bond graph notations.

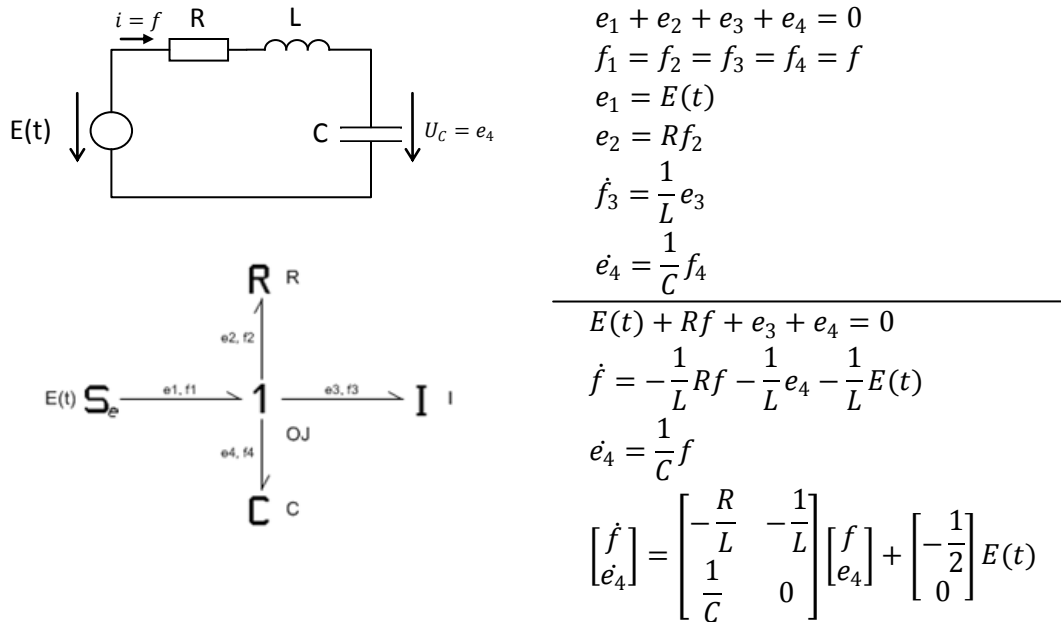


Figure II.14: Example for creating differential equation from bond graph

Model Integrated Computing

Model Integrated Computing (MIC) [6] is used for designing well-defined software systems, and it has three core components: (1) Domain Specific Modeling Languages, (2) the meta-programmable MIC tool suite based on the Unified Modeling Language (UML) class notations [9] and (3) a framework to verify, analyze, and transform models during the design process.

Modelers can specify Domain Specific Modeling Languages (DSML) [6] for their problem domains. These languages are not abstract; they are well-defined modeling languages using domain specific notations. DSMLs can be defined by a meta-language, which is based on UML and is supported by the Generic Modeling Environment [4].

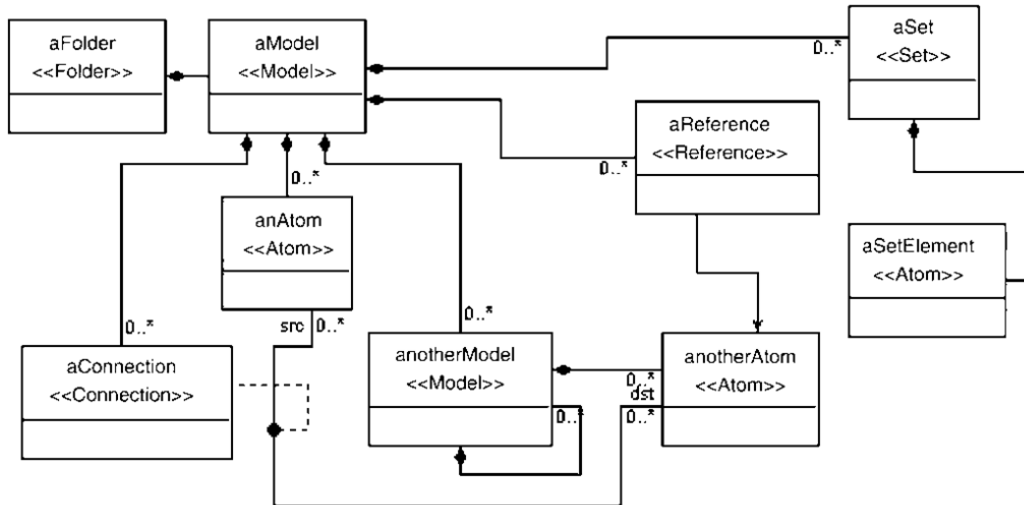


Figure II.15: Meta-modeling example [7]

The meta-modeling language for GME is called MetaGME. The MetaGME [13] language has several predefined concepts: models, atoms, sets, aspects, connections, attributes, and constraints. The modelers can describe their DSML with the composition of these concepts. A simple example for a meta-model is shown in Figure II.15. *Models* can contain other components, this containment allows the user to create hierarchical models, and they might have ports and connections. *Atoms* represent the atomic elements of DSMLs, which are shown with icons and have attributes. *Sets* are containers similar to models, but they group elements on the same level of the hierarchy.

In this and several other cases, the modeler may need to use different aspects for a model. Aspects are used to define the visualization of the models through projections. The model elements (models, atoms, etc.) can be connected to each other via connections. Connections are visualized as lines with or without arrows. The atoms, connections, sets, and models might have attributes, which can be set by the modeler. Attributes can be of type Boolean, enumeration, and field attribute (string, integer, and double). Regarding the attributes, the connections and the composition of the models, the language might have several design rules. Design rules are predefined syntactic and semantic restrictions for the user-built models, which must be satisfied by the model. One way to implement those rules is to use

constraints. Constraints can be written on Object Constraint Language (OCL) [12]. GME has a constraint checker, which evaluates constraints for the domain-specific model. When a constraint evaluates to false GME flags this as a problem.

GME contains the models in a tree structure, where one can reach every element of the model with starting from a special element called the root element. There are many algorithms that can traverse tree structures, for instance, Depth First Search (DFS) and Breadth First Search (BFS) algorithms.

ISIS has developed other tools, including Universal Data Model (UDM) [3], which is used to access the user built model. Using UDM, the developers of DSMLs can generate a C++ class representation for their modeling language. Then they can write a C++ program, which processes the user built models based in the particular DSML. The C++ program can create, delete, merge, transfer and generate models using UDM interface.

Such program is referred to as a model interpreter. Model interpreters can be used to process and verify GME models and check design rules on those models. The interpreter can generate any other files, for example, an other GME model based on another DSML, source code, script file, configuration file, input file for another software, etc. If the input and the output of the interpreter are graphs from different domains, this is called graph transformation. Another ISIS tool, GReAT [5] is used for this purpose.

CHAPTER III

DESIGN OF THE MODELING LANGUAGE

The purpose of this thesis is creating a DSML for modeling complex physical systems, and a model interpreter that processes the user-built models. The block diagram of the tool is shown in Figure III.16. The FDM tool carries out three main steps: design a model (based on the FDM meta-model), generate output files (using the FDM model interpreter), and create the Simulink model using the generated output files.

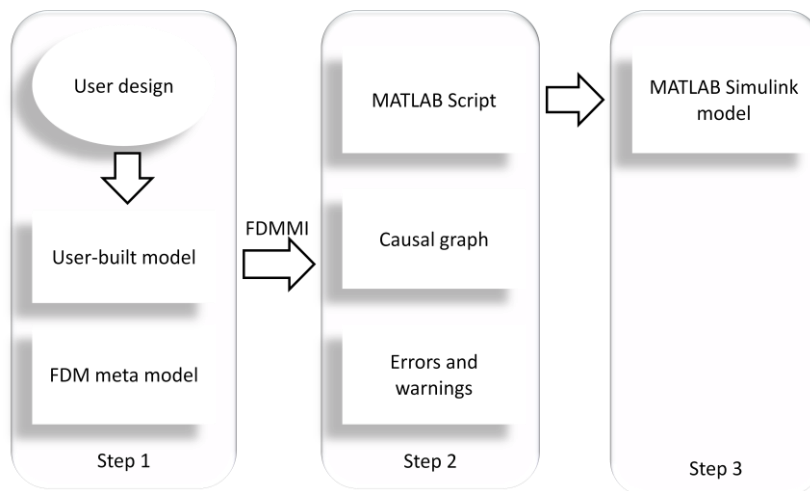


Figure III.16: FDM tool

FDM is a domain specific modeling language with functional and physical aspects. The modeling elements of the FDM are as follows. *Systems* have only functional components on the top level. Top level building blocks can be *Components*, *Power links*, *Information links*, *Groups*, *Ungroups*, *Packs*, *Unpacks*, and *Controllers*. This top level has different aspects: functional (i.e. *FunSketcher*) and physical domain specific aspects (i.e. electrical, mechanical displacement, mechanical rotation, thermal, and hydraulic). A part of the FDM meta-model is shown in Figure III.17.

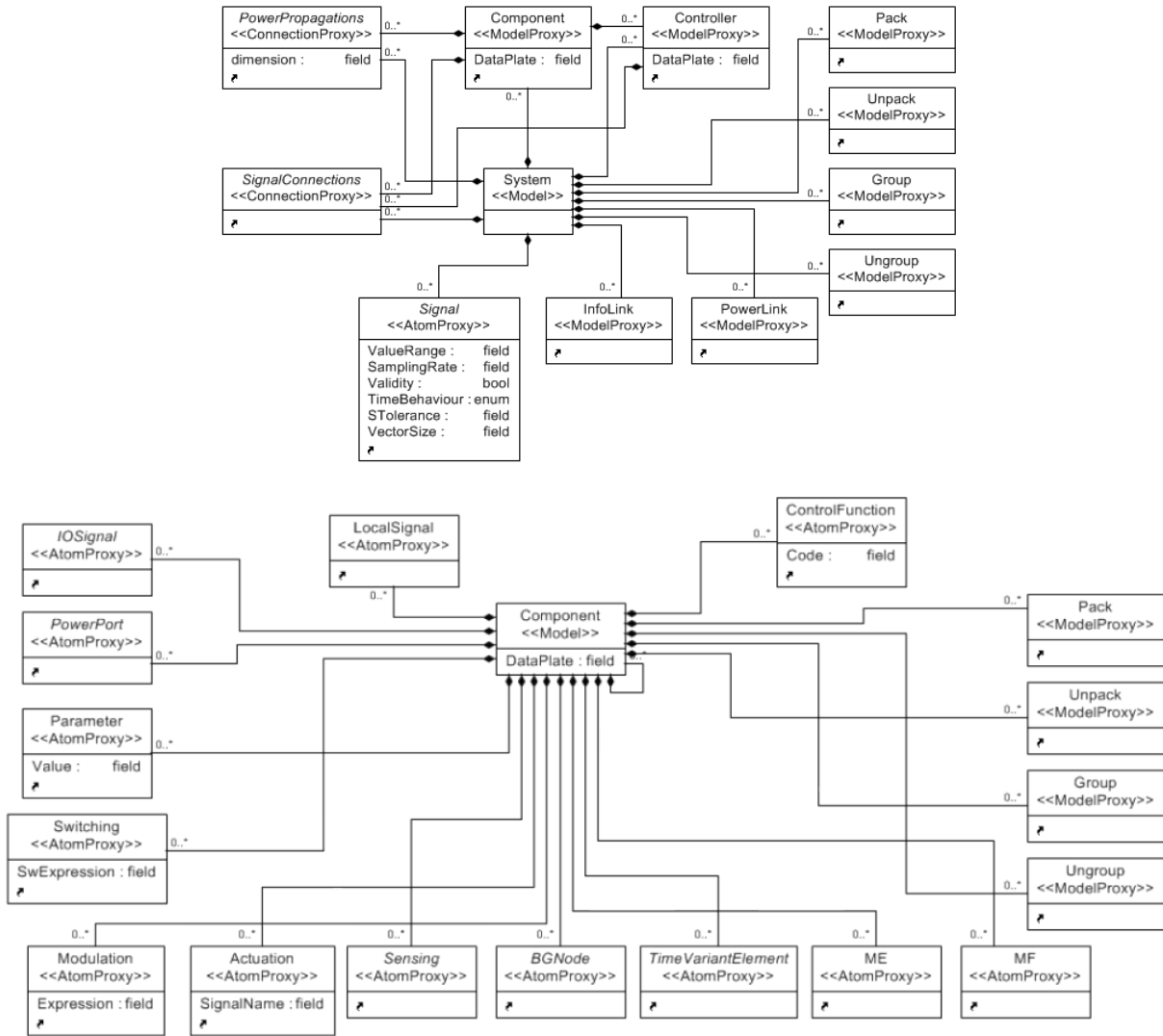


Figure III.17: Meta-model of FunSketcher

On the physical level, the modelers can create well-defined mathematical models for physical systems using our extended hybrid bond graph notations as described on page 11. Modelers can use additional elements to modify, measure, sense, and perform computations on bond graph variables and signals. The modulation function element can be connected to any modulated bond graph node, and it represents the internal modulation of the system. Internal modulation happens when a parameter of the system depends on a state or a variable of the system. For example, pipes can be modeled as a non linear resistor and the resistor is value depends on the fluid flow. The switching function element can be

connected to any switched junction, and it represents the internal switching of the system. For example, when a tank has a hole at height h and the fluid level reaches that point than a junction will be turned on by a switching function.

Sensing elements (i.e. `detect effort`: De and `detect flow`: Df) can `read` the generalized bond variables (i.e. effort and flow) on the junctions and they represent measured or sensed variables. De can be connected to a common effort junction (0-junction) to sense the (common) effort and Df can be connected to a common flow junction (1-junction) to sense the (common) flow. Modelers have also support for debugging their models and visualizing bond variables using monitor elements (such as monitor flow MF and monitor effort ME). MF can be connected to a common flow (i.e. 1-junction) element and ME can be connected to a common effort (0-junction) element.

FDM language supports three operators (i.e. time variant elements): *integrator*, *differentiator*, and *delay* elements for performing mathematical operations on signals. Sensing elements can measure physical quantities, but the modelers may need to use the integral or derivative of those quantities. For instance, the effort is the force and the flow is the velocity in mechanical displacement domain, and the time integral of the velocity (i.e. the generalized flow variable) is the distance.

1-port and 2-port elements have three attributes: value, minimum tolerance, and maximum tolerance. The value represents the nominal value of the element. The minimum and maximum tolerance is a percentage range around (below and above) the nominal value. These values are given by the manufacturers. The storage elements have one additional value, which is the initial value. For example, a hydraulic tank can be modeled as a capacitor. Assume that the hydraulic tank has some fluid at the starting point of the simulation. Then the initial value of the capacitor should be set up according to the initial fluid height in the tank.

The switched junctions have state machines with two states and two transitions, shown in Figure III.18. Each junction has three attributes: *InitialState*, *OffCondition*, and *OnCondition*. The *InitialState* has two different possible values *ON* (default) and *OFF*. This attribute represents the state of the junction at the beginning of the simulation. When the junction is *ON*, then the junction establishes a connection among all bonds connected to it. According to the type of the junction, all the efforts or all the flows are equal on all bonds. When the junction is *OFF*, the bonds are disconnected. When the *OnCondition* becomes true during the simulation and the actual state of the junction is *OFF* then the junction will switch its state from *OFF* to *ON*. When the *OffCondition* becomes true during the simulation and the actual state of the junction is *ON* then the junction will switch its state from *ON* to *OFF*.

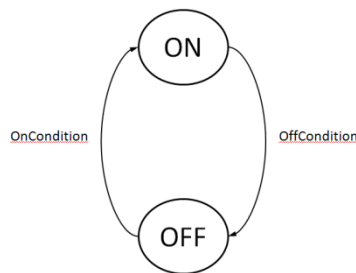


Figure III.18: State machine model of the junctions

Power ports are physical domain specific elements that are used to connect bond graph elements (i.e. junctions) to each other on different levels of the hierarchy. Power ports can be electrical, mechanical rotation, mechanical displacement, hydraulic, and thermal; they have domain specific attributes, which can be used to define minimum and maximum values for the power variables (i.e. generalized effort and flow). For instance, electrical power ports have minimum and maximum voltages and currents. Hydraulic power ports have minimum and maximum pressures and hydraulic flows. The connections between the power ports and the bond connections have a dimension attribute, which

represents the number of effort and flow pairs on that connection. This attribute allows the modeler to create vector bond graphs.

Power links represent real power connections between components, and they can model energy loss. *Power links* contain *Group* and *Ungroup* elements. *Group* elements are used to combine power ports together and create power groups. Power groups represent vector bonds. *Ungroup* elements are used for splitting combined power connections into separate bonds. These elements must have at least one power port element. The power ports must derive from the same domain. The top level model (i.e. *System*) and *Component* contain *Controllers*. *Controllers* are used for modifying signals and creating control signals. *Controllers* contain *signals*, *Simulink models*, mathematical operators (i.e. time variant elements: *integrator*, *differentiator*, *delay*), *control functions*, *Packs* and *Unpacks*, which are needed to create or use existing controller (such as Simulink model) logic. *Pack* elements are used for combining signal lines together and creating signal groups. *Unpack* elements are used for splitting signal connections. These elements must contain at least one input or output signal element.

FDM language has several design rules such as unique names, port connection restrictions, domain specific restrictions. Design rules can be implemented in OCL or in the model interpreter. The difference between the two methods is that the OCL expression is evaluated at design time and the interpreter is executed during the model processing.

All of the elements of the model must have unique names; this design rule is implemented in OCL. Another design rule is that 1-port elements in bond graphs must have only one bond connection. Therefore, input signal ports must have one input connection. However, between the input and output ports there are 16 different connections using the hierarchy concept. We use Table 1 for designing constraint rules of signal connections. The table contains all possible cases of the signal connections

between input and output ports. Cross sign (✗) represents the illegal connections and check sign (✓) represents legal connections.

Table III.2: Legal and illegal signal connections

			Destination			
			Same level		Child	
			Output	Input	Output	Input
Source	Same level	Output	✗	✗	✗	✗
		Input	✓	✗	✗	✓
	Child	Output	✓	✗	✗	✓
		Input	✗	✗	✗	✗

The minimum value must be less than or equal the maximum value on each power port. Power ports can be connected to each other if the power port type of the source and the destination is the same. Another design rule is the dimension checking on *Group* and *Ungroup* elements. The output vector dimension of a *Group* must be equal to the sum of the input vector dimensions. Therefore, the input vector dimension of *Ungroup* must be equal to the sum of the output vector dimensions.

A *Control function* is a computational block, and it is used to perform calculations with signals. The input and the output of a control function must be signals. The input and the output variables are determined by the direction of the connections between the control function and the signals. The control function elements have a string attribute, which must contain an executable code in the form of a script using the syntax of Embedded MATLAB Functions (EMF). The function header, which is the first line of the script, must have EMF syntax. The listed names of input/output variables on the function header must match the names of the connected input/output signals' names must be identical.

CHAPTER IV

DESIGN OF THE MODEL INTERPRETER

One of the duties of the FDM Model Interpreter (FDMMI) is to check the design rules (i.e., those of the abstract syntax) that must be satisfied by the model. The model interpreter marks the conflicts in the models. After the interpreter has processed the GME models, it will list all the errors and warnings in the GME console. These messages are shown with a hyperlink to the elements causing the design rule conflict. Modelers must fix those problems to generate semantically correct models.

FDMMI traverses the user-built model once using Depth First Search algorithm. The interpreter collects and stores all the information that it needs to generate the output model of the system, and stores it in a special temporary data structure (such as the causal directions of the bonds). This data structure contains the source and destination of the bonds, and a causal direction attribute. Then the model is processed, and the interpreter determines the causal directions on each bond.

Power ports can be connected to other power ports, which come from the same domain, through power connections. Power ports also can be connected to junctions. The path between the connected junctions through power connections called power propagation. Figure IV.19 shows examples for power propagations and their representation with additional bond connections. The pseudo code of the implemented algorithm is shown below, which processes all power propagations in the model and creates a flat bond graph for the SCAP algorithm.

```

AddDestJunctionAndPropagate(PowerPorts)
for PowerPort in PowerPorts
  dstJunctions += PowerPort.GetJunctionConnections()
  AddDestJunctionAndPropagate(PowerPort.GetPowerPortConnections())

```

```

for thisPowerPort in allPowerPorts
  If (thisPowerPort has only source power port connection)
    srcPowerPorts += thisPowerPort
  for srcPowerPort in srcPowerPorts
    srcJunctions = srcPowerPort.GetJunctionConnections()
    AddDestJunctionAndPropagate(srcPowerPort.GetPowerPortConnections())
  ConnectWithBonds(srcJunctions, dstJunctions) //connect all src-s to all dst-s

```

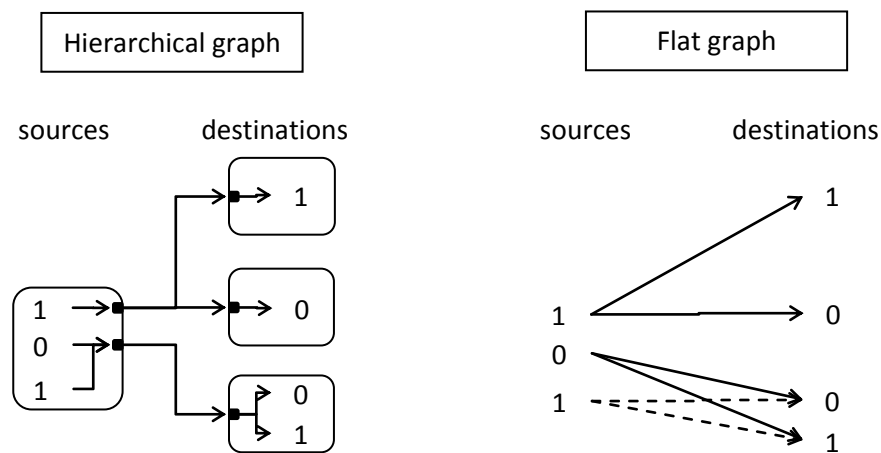


Figure IV.19: Power propagation example.

After the interpreter produced the flat bond graph in a special data structure, it can determine the causal strokes on the bonds. We use the Sequential Causality Assignment Procedure (SCAP) algorithm for determining the causal strokes on the bonds. Processing the bond graph using SCAP is needed to create either the mathematical equations or the block diagram of the system. When the modelers have one of them, they can simulate and analyze the system behavior using some kind of software tool (e.g. MATLAB Simulink).

```

Function Propagate(thisjunction)
  Set causal direction on bonds for this junction and 2 port elements if needed
  If (causality changed) Propagate(thisjunction.GetConnectedJunctions())

Algorithm (SCAP) pseudo code
// step #1
While ((srcElement = pick a source element) != null) // source elements: Se and Sf
  setcausality(srcElement.bond)
  Propagate(srcElement.GetConnectedJunction())
// step #2
While ((srcElement = pick a storage element) != null) // storage elements: C and I
  setcausality(srcElement.bond) // integral causality
  Propagate(srcElement.GetConnectedJunction())
// step #3
While ((srcElement = pick a resistor element) != null) // resistor element: R
  setcausality(srcElement.bond) // integral causality
  Propagate(srcElement.GetConnectedJunction())
// step #4
While (allbonds.causality != SET)
  Pick a bond and set the causality
  Propagate(srcElement.GetConnectedJunctions())

```

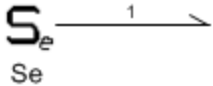
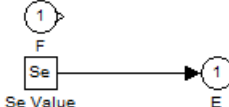
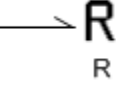
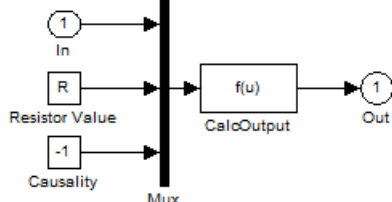
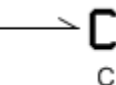
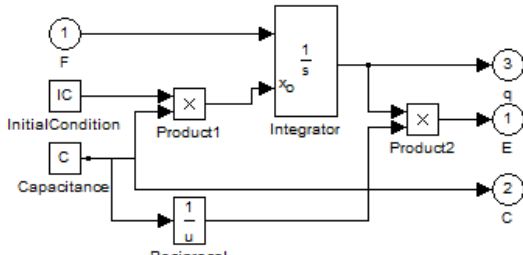
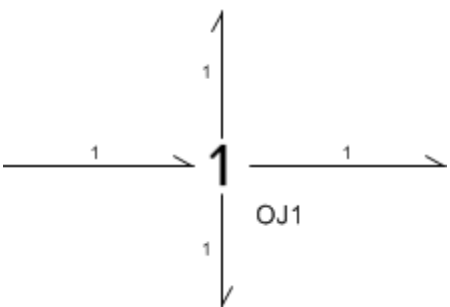
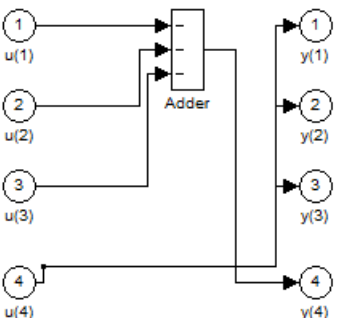
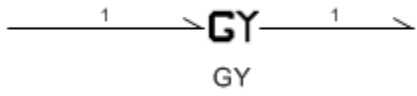
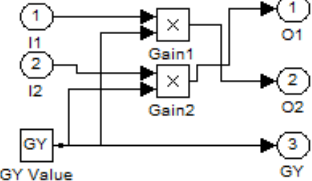
FDMMI generates MATLAB script that will construct a Simulink model according to the GME model. On the generated Simulink model the hierarchy and the layout of the subsystems is the same as in GME. Each model in GME can be mapped to a Simulink model or a special subsystem (except power ports) during the graph traversal. These subsystems and sub-models are connected together (except power connections and bonds) according to the connections in GME. The bond and power connections cannot be produced in one graph traversal. However, the model interpreter stores the source, the destination, and the causal direction of each bond. When the SCAP algorithm has the result, the model interpreter can generate the remaining connections between the bond graph elements and the necessary power ports in the Simulink model.

After the Simulink model is generated, the modeler can extend it with additional elements that can be connected to the top-level models. Top-level models can be connected together through input and output ports. Those additional elements are usually various sources (inputs), and the outputs (scopes), which allow the modeler to analyze the system behavior using different initial conditions and sources.

When the modelers work with a large project that contains many systems, and are interested in analyzing only a subset of the systems at a time, the FDMMI supports this feature. If the FDMMI processes only one or some systems, it takes less time than to interpret the whole project.

Table IV.3 contains some of the bond graph elements and their generated Simulink models. The source of flow is similar to source of effort, but it has effort input and flow output. The resistor element has one input that can be either the flow or the effort, and the *CalcOutput* function calculates either the effort or the flow according to the current causality. The inertia element is similar to the capacitor element, but the input of the inertia is effort, assuming integral causality, and the output is the flow. The 0- and 1-junction structures are the same, the difference is in the meaning of the inputs and outputs. The gyrator element contains two gain elements that are either multiplication or division, depending on the causality. The transformer element is similar to the gyrator element.

Table IV.3: Simulink models of the bond graph elements

Bond graph element in GME	Generated Simulink subsystem
	
	
	
	
	

CHAPTER V

CASE STUDY

In this section, we will describe an example that shows the FDM language and the FDMMI in action. We will build the bond graph model of a system and generate the corresponding MATLAB model using FDMMI tool. Our example system can be analyzed and simulated by MATLAB's simulation engine. The input source of the system is an (electrical) voltage source and the output of the system is (mechanical) rotation speed.

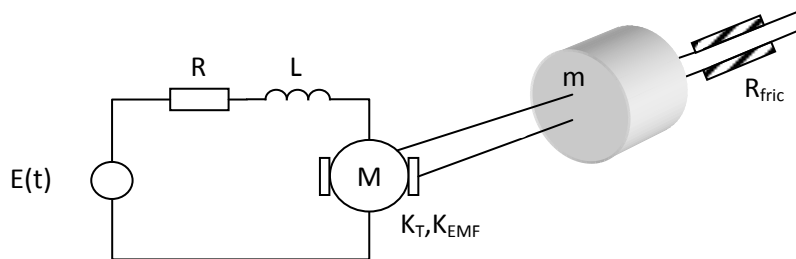


Figure V.20: System diagram

The example contains two different physical domains: electrical domain and mechanical rotation domain. The transformation of the quantities between the domains is made by an electro/mechanical device: a motor. This motor can be modeled as a gyrator in bond graph terminology. The electrical part contains an ideal voltage source (i.e. source of effort), a resistor (i.e. resistor), and an inductance (i.e. inertia). All electrical elements are connected to each other with serial (i.e. 1- junction) connections. The mechanical part of the system has two elements: a load mass (i.e. inertia) and a friction element (i.e. resistor).

The final bond graph of the system is shown in Figure V.21. The input source can be connected to the *In* input signal, which is the input voltage of the system. The *Electrical Source* produces voltage on its

output. This electrical output is connected to the input of the motor. The motor is modeled as a gyrator ($K_T=0.8 \frac{Nm}{A}$, $K_{EMF}=0.8 \frac{Vs}{rad}$, $r=0.8$), a resistor ($R_{arm}=8 \Omega$), and an inductance ($L_{arm}=400 \text{ mH}$). The current of the motor can be 'read' by the I_{arm} 'detect' flow element. The output I signal represents the current of the motor. The output of the motor is in the mechanical rotation domain. The Mechanical Load can be connected to the Motor. The load contains a mechanical friction element ($R_{fric}=0.2 \frac{Nms}{rad}$) and a mass ($m=0.4 \frac{Nm s^2}{rad}$). The angular velocity of the rod can be 'read' by the Dfw detect flow element. The output signal (w) represents the ω .

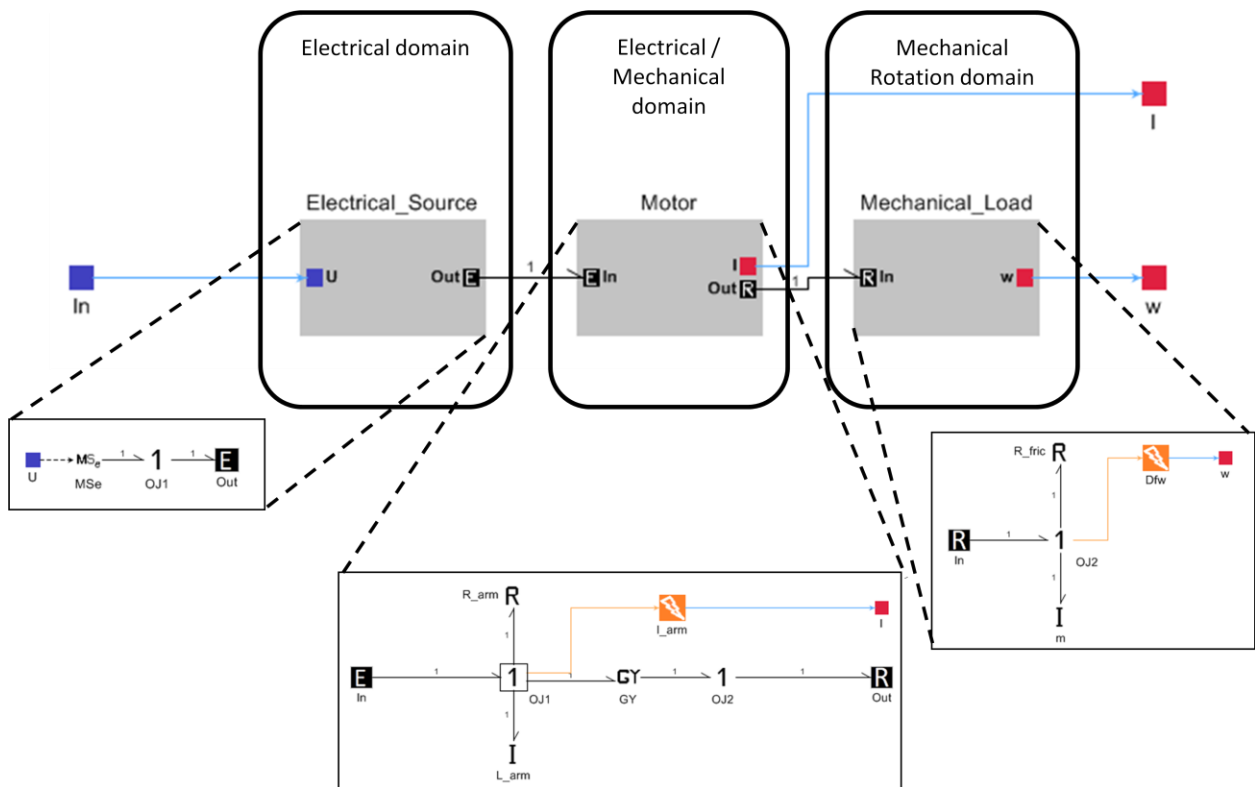


Figure V.21: Bond graph of the system in GME

The model of the system is given by a bond graph and we can use the FDM model interpreter to transform the FDM model to a Simulink model. The parameter values of the elements can be set up in GME and the generated Simulink model will reflect those values. This is done in two steps. The first step

is to run the interpreter and the second one is to execute the generated MATLAB script, which generates the Simulink model. The generated Simulink model is shown in Figure V.22.

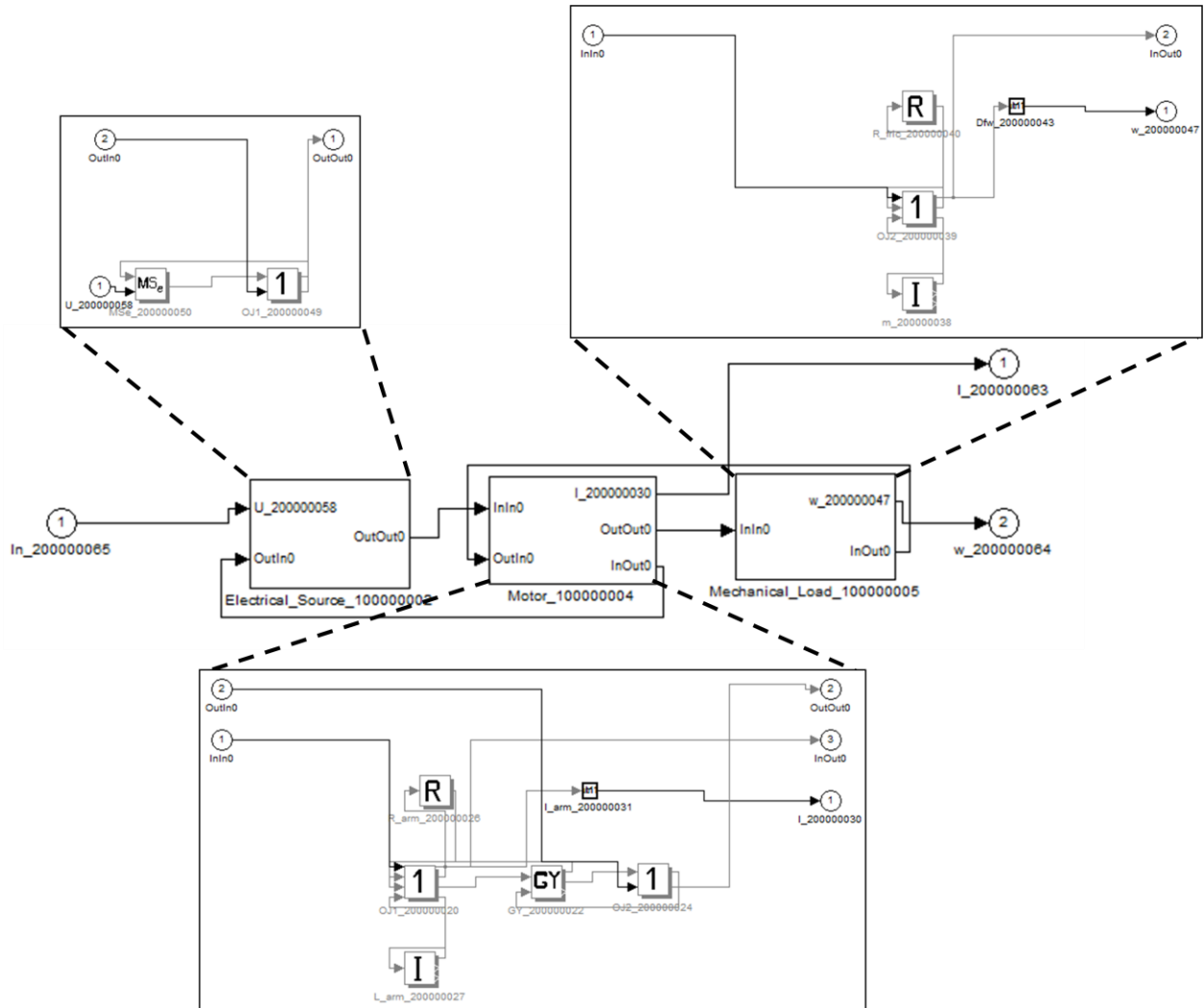


Figure V.22: Generated Matlab Simulink model

The input of the system is a step function with 0 V initial value at $t=0$ s and 24 V final value at $t=1$ s. Scopes are connected to the output signals for plotting the step response of the system. The plots show the current of the *Motor* and the angular velocity of the rod in Figure V.23.

The MATLAB allows for the modeler to simulate the system using different input functions such as sine, square, etc. The scopes and the input signals are not generated by the FDMMI, so the modelers shall add these elements to their systems.

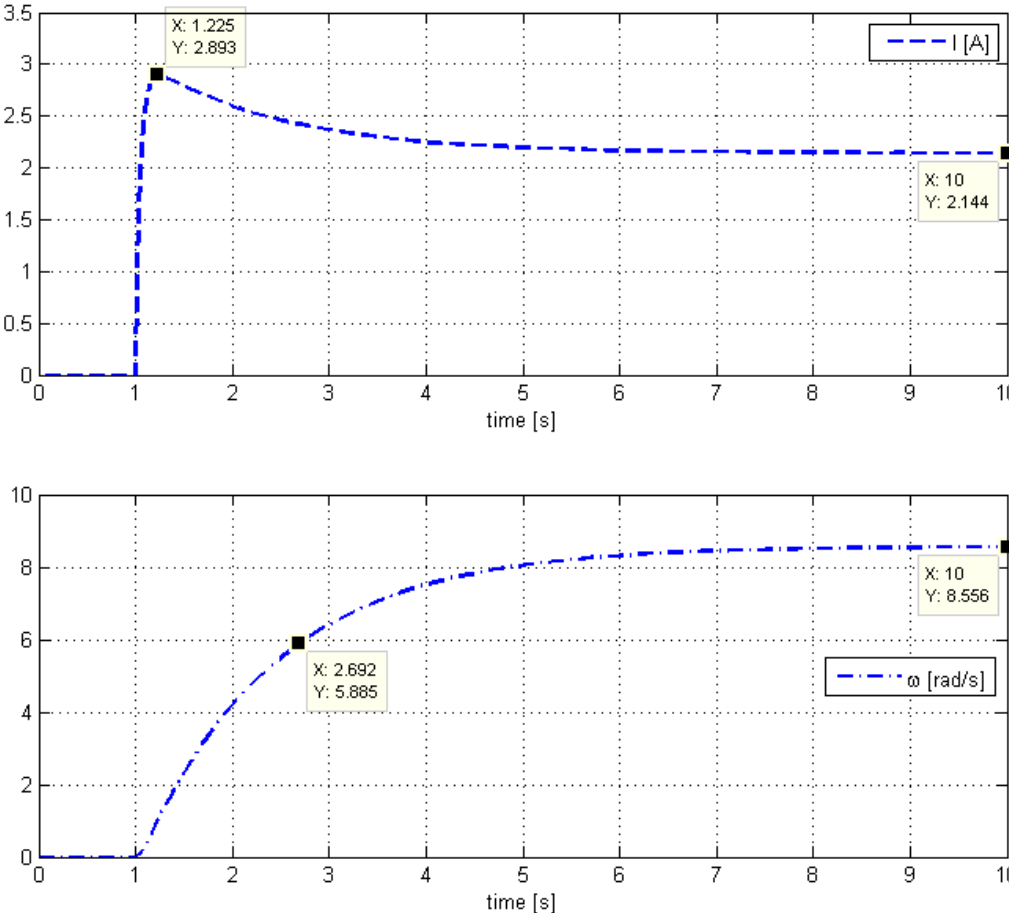


Figure V.23: Step response of the system

The overshoot of the current in the electrical side of the motor is $(2.893-2.144)/2.144 = 35\%$. The time constant of the mechanical side of the motor is $(5.885-1) s=4.885 s$.

CHAPTER VI

SUMMARY AND CONCLUSIONS

Summary

This thesis described two results: FDM, a DSML that can be used for modeling complex physical systems, and FDMMI, a model interpreter that generates Simulink models from the FDM models. The DSML is based on the hybrid bond graphs with some extensions: simple controller elements, sensors, actuation elements, modulation functions, switching functions, information links, and power links. The model interpreter checks some of the design rules, generates the causal graph of the system, and generates a MATLAB script, which can construct the Simulink model of the system. The modeler can simulate and analyze the system using the simulation engine of MATLAB.

Conclusion

When the models are large, using a graphical modeling language is more efficient than using only the mathematical equations of the system. These graphical modeling languages often support hierarchy and ports for easier modeling and understanding the system and its behavior. Usually, the modelers can choose a modeling language, which will be the basis of their DSML. However, that language should be extended other elements regarding the problem domain. Bond graphs provide a very common modeling technique in physical system modeling. The basic bond graph language does not support sensing, actuation, and mathematical operator elements. However, the physical systems have observed variables, control variables. Thus, we extended our language with explicit elements for sensing and actuation. On the other hand, transferring energy between the components usually has power loss. This

power loss linkage also can be modeled with bond graphs, but that power link element must be different than other component elements.

Future work

Modelers can use the tool in its current form, but there are some development opportunities. The meta-model can be extended with fault elements. The connections of fault elements can represent the fault injection point to the model, and the fault propagation through the system.

The model interpreter can also be extended to support the vector bond graphs and an optimal algorithm for the switching behavior of the junctions. The tool can be extended with other model interpreters. The model interpreters might generate models for other simulation engines (e.g. Modelica [15] and 20-Sim [16]).

BIBLIOGRAPHY

- [1] D. C. Karnopp, D. Margolis, and R. Rosenberg, *System Dynamics: Modeling and Simulation of Mechatronic Systems*, 4th ed. New Jersey: John Wiley & Sons Inc., 2006.
- [2] I. Roychoudhury, M. Daigle, G. Biswas, X. Koutsoukos, and P. J. Mosterman, "A method for efficient simulation of hybrid bond graphs," in *International Conference on Bond Graph Modeling and Simulation (ICBGM 2007)*, Jan. 2007, pp. 177-184.
- [3] Bakay A. and Magyari E. *The UDM Framework*, November 2005. Available with the UDM Framework, <http://repo.isis.vanderbilt.edu>.
- [4] Ledeczki, A., M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomasson, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The Generic Modeling Environment", *Workshop on Intelligent Signal Processing*, Budapest, Hungary, May, 2001.
- [5] Balasubramanian, D., A. Narayanan, C. van Buskirk, and G. Karsai, "The Graph Rewriting and Transformation Language: GReAT", *Electronic Communications of the EASST*, vol. 1, 2006.
- [6] Sztipanovits, J., and G. Karsai, "Model-Integrated Computing", *IEEE Computer*, vol. 30, pp. 110--112, April, 1997.
- [7] G. Karsai, et al., Model-driven architecture for embedded software: A synopsis and an example, *Science of Computer Programming (2008)*, doi:10.1016/j.scico.2008.05.006
- [8] Peter C. Breedveld, *MODELING AND SIMULATION OF DYNAMIC SYSTEMS USING BOND GRAPHS*, in *Control Systems, Robotics and Automation*, from Encyclopedia of Life Support Systems (EOLSS), Developed under the Auspices of the UNESCO, Eolss Publishers, Oxford ,UK, [<http://www.eolss.net>]
- [9] Pilone, Dan. *UML Pocket Reference*. Sebastopol, CA: O'Reilly Media Inc, 2003
- [10] MATLAB®/Simulink®, Available: <http://www.mathworks.com/products/simulink/>.
- [11] François E. Cellier and Dirk Zimmer, "Wrapping multi-bond graphs: A structured approach to modeling complex multi-body dynamics", *European Conference on Modeling and Simulation*, Germany, 2006
- [12] Object Management Group (OMG); *Object Constraint Language OMG Available Specification Version 2.0*, May 2006
- [13] Karsai, G., Maroti, M., Ledeczki, A., Gray, J., and Sztipanovits, J.: "Type Hierarchies and Composition in Modeling and Meta-Modeling Languages," *IEEE Transactions on Control Systems Technology*, Vol. 12, No. 2, pp. 263-276, March, 2004

- [14] Mosterman, P. J., Biswas, G.: *A theory of discontinuities in physical system models*, Journal of the Franklin Institute, Volume 335, Issue 3, April 1998, Pages 401-439
- [15] Modelica, Available: <http://www.modelica.org/>
- [16] 20-Sim, Available: <http://www.20sim.com/>