

STRUCTURAL, BEHAVIORAL AND FUNCTIONAL MODELING OF  
CYBER-PHYSICAL SYSTEMS

By

Tamas Szarka

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

Computer Science

August, 2011

Nashville, Tennessee

Approved:

Professor Gautam Biswas

Professor Gabor Karsai

*Revised for Electronic Submission*

## ACKNOWLEDGEMENTS

This research was supported by Airbus UK Limited, under the “Dependability and Risks Assessment Framework Tool Sets” program as a project called “Behavioral Dependency Modeling” (BDM). Their support is acknowledged. I would like to specifically thank Sanjiv Sharma and Christopher Dean for their instructions and comments during the series of teleconferences.

I would like to say thanks to my advisor, Professor Gautam Biswas for all his support, the instructions and advice he provided me with during this project. I would also like to say thanks to Professor Gabor Karsai for his advice on the project and contributions to the teleconferences with Airbus. Finally I want to say thanks to Zsolt Lattmann, previous developer of the system, for his ideas, insightful questions and the rigorous testing of the model interpreter that helped a lot during the development of the tool.

# TABLE OF CONTENTS

ACKNOWLEDGEMENTS .....	ii
LIST OF TABLES .....	v
LIST OF FIGURES .....	vi
Chapter	
I. INTRODUCTION .....	1
Motivation .....	1
Problem statement .....	3
Organization of the thesis .....	4
II. BACKGROUND.....	5
Modeling hybrid systems.....	5
Hybrid bond graphs .....	6
Introducing bond graphs .....	6
Bond graph components .....	8
Modulation and signals.....	9
Causality .....	10
Causality assignment procedure .....	13
Hybrid behavior.....	13
Example bond graph.....	16
Grafcet notation .....	18
Standard Grafcet.....	18
Signal-Action Graphs .....	20
Hoare logic .....	21
III. THE GENERIC MODELING ENVIRONMENT .....	23
Model integrated computing.....	23
MetaGME .....	24
IV. DESCRIPTION OF THE SYSTEM.....	27
Previous work.....	27
Overview .....	28
The BDM paradigm.....	29
Structure and hierarchy.....	29
Signal and power flow .....	31
ControlNodes.....	32

Other features of the paradigm .....	33
The simulation model in Matlab®.....	34
Overview .....	34
The Simulink model interpreter.....	35
HBG-Simulink mapping.....	36
Hybrid switching structures.....	39
Grafcet-SimEvents mapping.....	42
V. CASE STUDY – THE REVERSE OSMOSIS SYSTEM.....	45
The Water Recovery System .....	45
Operation of the Reverse Osmosis system .....	46
Bond graph model of the RO system.....	47
Simulation.....	50
VI. DISCUSSION & CONCLUSIONS.....	54
Summary.....	54
Other tools .....	55
Future work .....	55
APPENDIX.....	57
A. LIST OF MODELING ELEMENTS IN THE BDM FRAMEWORK .....	57
B. PARAMETERS OF THE RO SYSTEM .....	64
C. CAUSAL BOND GRAPH OF THE RO SYSTEM IN THE THREE MODES .....	65
D. DOOR SYSTEM SAG CALCULI EXAMPLE .....	66
REFERENCES .....	67

## LIST OF TABLES

Table 1. Domain-specific equivalents of bond-graph variables.....	7
Table 2. Chart of bond graph elements .....	12
Table 3. Bond-equations of the suspension system .....	17
Table 4. Simulink structure of regular bond graph elements .....	38
Table 5. SimEvents subsystems for Grafcet entities .....	43

## LIST OF FIGURES

Figure 1. Bond graph with modulated resistor (MR) and effort source (MSe).....	10
Figure 2. Hybrid bond graph and schematic of a tank system .....	15
Figure 3. Effect of switching on causality .....	15
Figure 4. Example mechanical system.....	16
Figure 5. Example of Grafcet and Petri net notation .....	19
Figure 6. ‘AND’ type token-junction and distribution in Grafcet .....	19
Figure 7. Interaction of Grafcet and HBG .....	21
Figure 8. Relationship of Grafcet and Hoare triples .....	22
Figure 9. Multigraph architecture .....	23
Figure 10. MetaGME UML class diagram .....	26
Figure 11. Structure of the BDM environment .....	28
Figure 12. The Four Variable Model .....	30
Figure 13. Example of the ControlNode elements.....	33
Figure 14. Internal structure of a hybrid, non-fixed junction.....	41
Figure 15. Simulink structure of a resistor with changing causality.....	42
Figure 16. The NASA Water Recovery System .....	45
Figure 17. Schematic of the RO system.....	46
Figure 18. Bond graph model of the RO system.....	48
Figure 19. Grafcet model of the three modes of operation .....	50
Figure 20. GME model of the RO system .....	51
Figure 21. Mode switches, conductivity and hydraulic displacement values .....	52
Figure 22. Pressures, output flow and membrane resistance values .....	52

## CHAPTER I

### INTRODUCTION

#### Motivation

Modeling and simulation of cyber-physical systems is a key component for designing dependable next-generation automotive, avionics and space systems, power plants and manufacturing processes. Cyber-physical systems (CPS) are typically large-scale, distributed control systems, where computational devices are tightly coupled with physical processes [1]. Therefore, they are inherently concurrent and often enormously complicated. CPSs are prevalent in industry, and the degree of their complexity has reached a level, where component-based modeling and effective simulation approaches are essential to analyzing their behavior and predicting their performance under different operating conditions. The design process of these complex systems includes building preliminary mathematical behavior models to create simulation artifacts, and carry out virtual testing and verification, before the physical design commences.

This thesis discusses the implementation of a visual modeling environment that supports component-based modeling and simulation of cyber-physical systems. The system is built on the Generic Modeling Environment (GME), developed at ISIS, Vanderbilt University [2] and a set of Matlab™ toolboxes [3]. GME provides a flexible environment, implementing the Multigraph architecture for model-based development and analysis [4]. The workflow begins with the software engineer, who defines the modeling language, allowing domain experts to build the necessary models in the domain, while model interpreters whose development is facilitated by the GME environment completely automate the process of generating simulation models.

Cyber-physical systems include a set of physical processes and the associated computational components, which monitor and control those processes and their interactions, initiate actuation mechanisms and facilitate human interaction. Therefore, a cyber-physical system includes a large number of hybrid subsystems, whose state evolves in both continuous and discrete time. CPSs are often distributed in physical space, such as the various subsystems that make up an aircraft system, which necessitates that the system models constructed include properties, such as time-delay and power-dissipation occurring during the interaction. Today, the typical size-scale of cyber-physical systems – especially in the avionics industry – requires non-standard approaches to design and analysis.

From a systems-engineering point of view, cyber-physical system models are often considered as a composition of three layers, often called the Function, Behavior and Structure (FBS) representation of the system. Models of the physical assembly defines the *structural* description of the system, the *behavior* of this structure can be simulated under a variety of circumstances to verify system functionality and properties, and the well-defined view of the expectations of the system are captured as system *functions*. Typically the goal of the system design is to translate these functional requirements into system models [5]. For a modeling tool that is designed to aid complex cyber-physical system development, it is essential to capture and relate the Structural, Behavioral and Functional aspects of the system.

Our modeling tool employs a component-based approach to capture the functional, behavioral and structural layers of the cyber-physical system. Component based models preserve the hierarchy in the system, keeping models human-readable and supporting the reuse of previously factored components, while Function-Behavior-Structure (FBS) models allow for simulating the behavior and verify high-level functional properties too. Finally, such a modeling environment has to provide clean semantics to integrate physical models from various physical domains, such as hydraulic, electrical, mechanical, and thermal subsystems.



## Problem statement

In this thesis, we introduce a Domain Specific Modeling Language (DSML) to model cyber-physical systems, including the notions of abstract and concrete syntax. A DSML is a set of modeling elements and abstract rules, designed to describe artifacts of a particular problem domain. The definition of the DSML also contains the visual syntax of the modeling language.

The language is implemented as a modeling paradigm in the Generic Modeling Environment tool, and is made up of two components: (1) the hybrid bond graph language and (2) an extended version of the Grafcet notation [6]. After developing the DSML specifics, we formally define the mapping between the semantics of hybrid bond graphs and Matlab Simulink® models [7] to provide a simulation environment for studying system behaviors. The Simulink signal-flow representation is designed to allow efficient reconfigurations as the mode changes occur. A mapping between the Grafcet notation and Matlab SimEvents® [8] block diagrams is also presented.

As part of the system, we developed two model-interpreters. One interpreter is built to generate Matlab simulation artifacts from hybrid models defined in GME, and the other to extract functional requirements defined within the Grafcet language in a form of Hoare triples [9]. Both interpreters are implemented in C++ and use the Universal Data Model [10] interface to access the model. When generating a Simulink signal-flow graph from the hybrid bond graph, we have to add the notion of causality to the model. This happens in the Matlab environment, based on the “Sequential Causality Assignment Procedure” (SCAP) [11]. In a hybrid bond graph, as the reconfigurations occur along with the discrete mode changes, the correct causality should be efficiently maintained in the system. This is performed by a library of Matlab functions.

## Organization of the thesis

The following chapters are organized as follows. Chapter 2 gives background information on the hybrid bond graph modeling language, describing its syntax and semantics, the causality assignment procedure and the hybrid behavior. It also introduces the Grafcet notation, which is used to model the computation elements for the CPS models. Chapter 3 gives an overview on the Generic Modeling Environment, and Chapter 4 discusses the BDM Sketcher tool<sup>1</sup>, the related GME paradigm and Matlab libraries, and other implementational details. Chapter 5 presents a case study; the Reverse Osmosis system, part of the Advanced Water Recovery system built at NASA [12,13], and Chapter 6 summarizes the results and conclusions.

---

<sup>1</sup> The names “BDM” and “BDM Sketcher” come from Airbus UK Limited. The abbreviation means “Behavioral Dependency Modeling”.

## CHAPTER II

### BACKGROUND

#### Modeling hybrid systems

Systems are considered hybrid, if their state variables can evolve in both continuous time and at discrete time points. While physical systems are always continuous by nature, from a modeling point of view, some of their behaviors which include fast nonlinearities may be simplified as discrete changes at points in time. For example, a bouncing ball is usually modeled with a set of equations describing free fall in a gravitational field, with the collision with the ground being abstracted and represented as a discrete event. The change in the ball's behavior may be described by one or more equations that are only valid at the point of collision. This simplification or approximation abstracts away the complex energy transfers from kinetic to potential to kinetic that happen during an elastic collision. An equation based on the coefficient of restitution simplifies the resulting simulation model for the system.

Cyber-physical systems, where components with their behavior regulated by the laws of physics are coupled with computational devices that are modeled to operate in a discrete manner, are naturally hybrid in their behavior. A standard approach to model these kinds of systems is the hybrid automaton model of computation [14]. The different modes of operation are represented as states of the automata model. In each state, continuous behavior evolution is defined by a set of differential algebraic equations. Discrete changes that occur at points in time are modeled as transitions between states; they are triggered by evaluation of guard conditions associated with a state transition function. Transitions can be extended with an arbitrary function to recompute the state variables when the transition occurs; this is called the hybrid reset function [15].

The bond graph modeling language represents a domain-independent formalism to model physical systems. It abstracts away the specific details of particular physical domains, providing a common notation for defining cross-domain systems using a power- and energy-based formalism. The standard bond graph language contains only 9 different modeling elements and their connections [16]. This is discussed in greater detail in the next section. An extended version of bond graphs is the hybrid bond graph language, which can be used to capture the discrete mode-changes by defining reconfiguration conditions that apply to the junction elements of the bond graph model.

The physical layer modeling is based on the hybrid bond graph language, while computational units which trigger mode switches in the hybrid bond graph are modeled with an extended version of the Graftet language [6]. These two concepts are integrated in an environment called the “BDM Sketcher”.

## Hybrid bond graphs

### Introducing bond graphs

The concept of bond graphs was developed in 1959 at Massachusetts Institute of Technology by Henry Paynter, and was further developed by his students: Karnopp, Margolis, Rosenberg and others. The introduction below is based on the fourth edition of their book on modeling mechatronic systems [16], which is still the most comprehensive reference to bond graphs.

The bond graph formalism is a graphical modeling language to describe cross-domain physical systems. It generalizes components of various physical domains, providing a domain-independent notation for system modeling, consisting of 9 types of atomic elements: energy storage elements: the capacitor (C) and inertia (I), a dissipative element, the resistor (R), sources of effort (Se) and flow (Sf), the transformer (TF) and gyrator (GY) and two types of junctions (0, 1). Bond graphs are energy based; energy transfer between the elements of the system is defined by connections,

called bonds. They capture the power exchange between components, and each bond is represented with two generalized variables, the *effort* ( $e$ ) and *flow* ( $f$ ), such that  $effort \times flow = power$  ( $P$ ) = rate of energy transfer between the connected components. Therefore, the integral of the exchanged power corresponds to the *energy* ( $E$ ) exchange between components on the two sides of the bond.

$$P(t) = e(t) \cdot f(t) \qquad E(t) \equiv \int^t P(t) dt = \int^t e(t) \cdot f(t) dt$$

Table 1 lists the effort and flow variables in the different domains. There are two additional variables called energy variables: the *displacement* ( $q$ ), corresponding to potential energy and *momentum* ( $p$ ), corresponding to kinetic energy. They are associated with the integral of flow and effort respectively, and they define the formal state variables of the system. Table 1 also presents their domain-specific versions. Formally they are:

$$q(t) \equiv \int^{t_0} f(t) dt = q_0 + \int_{t_0}^t f(t) dt \qquad p(t) \equiv \int^{t_0} e(t) dt = p_0 + \int_{t_0}^t e(t) dt$$

**Table 1. Domain-specific equivalents of bond-graph variables**

Domain \ Variable	Effort ( $e$ )	Flow ( $f$ )	Displacement ( $q$ )	Momentum ( $p$ )
<b>Mechanical translation</b>	Force, $F$	Velocity, $V$	Displacement, $X$	Momentum, $P$
<b>Mechanical rotation</b>	Torque, $\tau$	Angular velocity, $\omega$	Angle, $\theta$	Angular momentum, $p_\tau$
<b>Hydraulic</b>	Pressure, $P$	Volumetric flow, $Q$	Volume, $V$	Pressure momentum, $p_P$
<b>Electrical</b>	Voltage, $e$	Current, $i$	Charge, $q$	Flux linkage variable, $\lambda$
<b>Thermal</b>	Temperature, $T$	Heat flow rate, $\dot{q}$	Entropy, $S$	-

The thermal domain is different from the others in that there is no inertia element defined in it, and the product of effort and flow is not power. (The heat flow rate,  $\dot{q}$  defines power, as flow of energy in the thermal domain). However, to maintain uniformity of representation and allow for multi-domain modeling that includes the thermal domain, a pseudo bond graph representation is adopted, where temperature,  $T$  represents the effort variable, and flow rate,  $\dot{q}$  is the flow variable.

### Bond graph components

The bond graph modeling language describes the power interaction of subsystems that are considered atomic. Places where a subsystem can be interconnected with other subsystems are called ports. The primary bond-graph formalism consists of 5 one-port elements, 2 two-port elements and 2 junction types to define their connections. One-ports are the *effort source* ( $Se$ ), *flow source* ( $Sf$ ), *capacitor* ( $C$ ), *inertia* ( $I$ ) and *resistor* ( $R$ ); two-port elements are the *transformers* ( $TF$ ) and *gyrators* ( $GY$ ). Other multiport elements include the *common effort-* ( $0$ ) and *common flow junctions* ( $1$ ). In general, junctions are ‘n-ports’; they can interconnect arbitrary number of components. One-port and two-port elements are again generalized versions of subsystems in real physical domains. Source elements model ideal effort and flow sources (i.e.  $Se = e(t)$ , or  $Sf = f(t)$ ). Their output does not depend on the rest of the bond graph. The capacitor and the inertia are called storage elements, and they maintain the energy variables, and therefore, the state of the system. The resistor is used to model energy dissipation. We have two types of two-ports. Both of these conserve power: the transformer relates the effort to the effort and the flow to flow between its two bonds, while the gyrator relates flow to effort and vice versa. Junctions define the interconnection of the port-elements; one- and two-port elements can only be connected to junctions, while junctions can also be connected to other junctions. Common effort and common flow junctions are also called 0-junctions and 1-junctions respectively. 0-junctions force a common effort value on each adjacent bond, while 1-junctions correspond to a common

flow connection. They can be thought of as parallel and serial connections in the electrical domain: the analogies of the 0- and 1-junction are Kirchhoff's first and second laws, respectively.

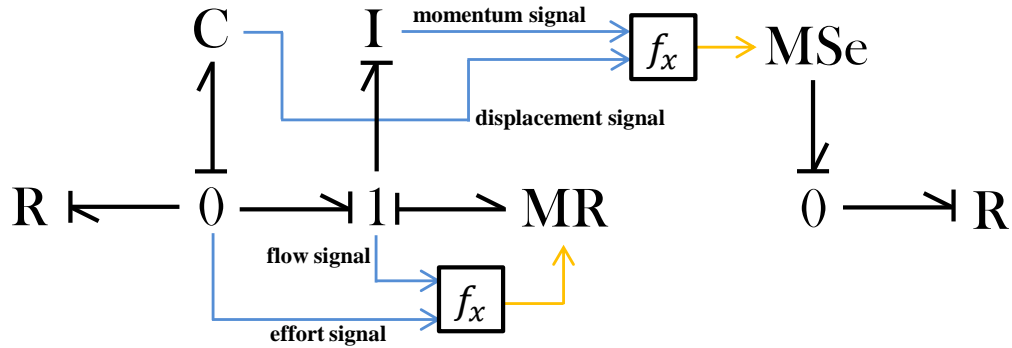
Bonds are directional; they are represented with a half-arrow, and the direction of the arrow denotes the positive direction of power exchange between the components. A negative value of power implies that the direction of power flow is opposite to the direction of the bond. Associated with the nine primitive bond graph elements are constituent equations that define the relationship between effort and flow for the particular bond.

### Modulation and signals

Every one-port element has a parameter, and both of the two-ports have a modulus associated with the function that relates the pairs of efforts and flows. For linear systems, these parameters assume constant values. To model components with nonlinear behaviors, we introduce the notion of modulated elements, where the parameter value can be made a function of one or more signal inputs to the component. Signal paths are distinct from power connections in the bond graph; they represent transfer of information as opposed to bonds that represent transfer of energy. Signal paths are denoted with lines ending with regular arrows. As an approximation, we assume that zero power is required to transmit signals. Signals can be made external, i.e. they originate from outside the bond graph model, or they can be internal, in which case they transmit the current value of any particular bond graph variable from one part of the graph to another. The notion of modulated components adds seven more elements to the bond graph language, since every one- and two-port element has its modulated version ( $MS_e$ ,  $MS_f$ ,  $MC$ ,  $MI$ ,  $MR$ ,  $MTF$ ,  $MGY$ ).

Figure 1 demonstrates the bond graph notation, including the notion of signals, which carry effort, flow and energy variables to modulate bond graph elements. In this example, the modulated source value is a function of the displacement variable from capacitor  $C$ , and momentum variable from inertia,  $I$ . Similarly, the resistance value is modulated by a function

whose input parameters are the common effort of the 0-junction, and the common flow of the 1-junction.



**Figure 1. Bond graph with modulated resistor (MR) and effort source (MSe)**

### Causality

It is important, that when we model physical systems, components cannot be thought as something that performs a function on a set of input-variables, generating the corresponding output-variables. In reality, the behavior of each component is concurrent; the relationships are described by equations, and their causal structure cannot always be determined [17]. However, some of the bond graph elements impose causality constraints on the system model, which is propagated, and results in a consistent causal assignment to bonds in the bond graph model. Bonds are assigned a causal direction, and this may or may not be independent of its power direction. The causal direction is indicated by a perpendicular stroke to one end of the bond, meaning that the effort value is imposed from the side of the bond without the causal stroke to the side with the causal stroke. Once causality is assigned to a bond, the direction of effort and flow is known, and it generates a response in the opposite direction.

To derive a block diagram from a bond graph with regular, directed signals, we have to come up with a consistent causality assignment to every bond. To perform simulation efficiently on the



bond graph model, we convert it into a signal-flow block-diagram, which determines the sequence of computations necessary to establish system behavior over time.

The state variables of the system are maintained in the storage elements, where we consider only integral causality. Integral causality means that energy-variables are expressed as the integral of the input effort or flow, instead of using derivatives. An energy storage element forced to accept derivative causality would mean that its energy variable is dependent on another state-variable; therefore, it could be eliminated from the state equations based on the rest of the equations. By assuming integral causality, the equations describing the storages remain independent from the rest of the bond graph, and a single state equation will be associated with each storage element. Therefore, computationally the capacitor accepts flow as input, and generates an effort value, whereas the inertia accepts effort as input, and produces flow.

The causality of source elements is also constrained; the effort source imposes effort, while the flow source imposes flow onto its adjacent bond. The causality of the resistor can be assigned arbitrarily. The transformer and the gyrator both have two possible combinations of causality assigned on their bonds, based on their equations, as shown in Table 2. Junctions mean a common effort or flow value in the associated points of the system: one of their bonds imposes this effort or flow on them, which bond is called the *determining bond*. This incoming effort or flow is then transmitted on every other bond as an output, while the responded flows or efforts coming in via those bonds are summed within the junction (with the power directions taken into account), and that will be the output of the determining bond. Therefore, the determining bond of 0-junctions has the causal stroke on the side closer to the junction, while every other bond has it on the further side, and this is true the other way around for 1-junctions. Table 2 shows the regular bond graph elements with their possible causal assignments, equations and block-diagram representation.

Table 2. Chart of bond graph elements

	Sign	Causality	Equation	Block diagram
Source of effort	Se	Se : u $\longrightarrow$	$e = u$	
Source of flow	Sf	Sf : u $\longleftarrow$	$f = u$	
Capacitor	C	$\longleftarrow$ C : c	$e = \frac{1}{c} \int f dt$	
Inertia	I	$\longrightarrow$ I : i	$f = \frac{1}{i} \int e dt$	
Resistor	R	$\longleftarrow$ R : r	$e = rf$	
		$\longrightarrow$ R : r	$f = e/r$	
Transformer	TF	$\xrightarrow{1} \text{TF:m} \xrightarrow{2}$	$e_2 = e_1/m$ $f_1 = f_2/m$	
		$\longleftarrow \text{TF:m} \longleftarrow$	$e_1 = me_2$ $f_2 = mf_1$	
Gyrator	GY	$\longleftarrow \text{GY:m} \xrightarrow{2}$	$e_1 = mf_2$ $e_2 = mf_1$	
		$\longrightarrow \text{GY:m} \longleftarrow$	$f_1 = e_2/m$ $f_2 = e_1/m$	
0-junction	0	$\xrightarrow{1} 0 \xrightarrow{2}$ Determ. bond $\downarrow 3$	$e_1 = e_2 = e_3$ $f_1 = f_2 + f_3$	
1-junction	1	$\longleftarrow 1 \longleftarrow$ Determ. bond $\downarrow 3$	$f_1 = f_2 = f_3$ $e_1 = e_2 + e_3$	

## Causality assignment procedure

Based on the causality restrictions of the elements discussed above, a bond graph can be augmented with causality information using an incremental algorithm. One such algorithm is called “SCAP”, which stands for Sequential Causality Assignment Procedure [16,11]. The SCAP algorithm plays a pivotal role in the conversion from bond graphs to block diagrams. An important part of the procedure is the propagation of the causality, where we take advantage of the fact, that a particular causality assignment of a bond usually constraints the assignment of a set of other bonds. For example, setting a bond to be the determining bond of its neighboring junction causes every other bond adjacent to that junction to be set to the opposite direction; or setting the causality on one side of a two-port determines the assignment of the bond on the other side. These effects of assigning causality on bonds are propagated all through the bond graph after every step of the algorithm. Below is the outline of “SCAP”:

1. While source with unassigned bond exists
  - a. Pick an unassigned source and assign the required causality to its bond
  - b. Propagate the effects via junctions and two-ports where possible
2. While storage with unassigned bond exists
  - a. Pick an unassigned storage and assign integral causality to its bond
  - b. Propagate the effects via junctions and two-ports where possible
3. While resistor with unassigned bond exists
  - a. Pick an unassigned resistor and assign the causality to its bond arbitrarily
  - b. Propagate the effects via junctions and two-ports where possible
4. While there is unassigned bond left (they must be between junctions or two-ports)
  - a. Pick one of these bonds, and assign its causality arbitrarily
  - b. Propagate the effects via junctions and two-ports where possible

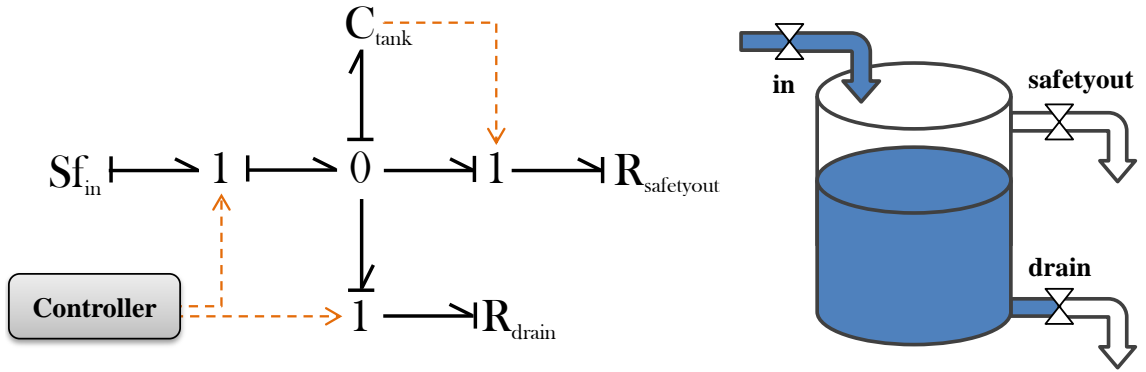
## Hybrid behavior

Regular bond graphs have the same expressiveness as a set of differential equations that describe continuous system behavior. To incorporate hybrid behavior in the bond graph modeling language, we introduce the notion of controlled (or switching) junctions [18]. This means that

junctions may switch on and off to model discontinuous behavior: when a 0-junction is switched off, it functions as a source of effort with a zero value, while a closed 1-junction becomes a flow source with zero parameter. Note that a zero effort or flow on a bond implies that the bond is inactive because it does not transmit power. The on-off configuration of the set of controlled junctions in a component corresponds to a discrete state of the related hybrid automaton.

Controlled junctions define two logical expressions: the ON-condition and OFF-condition. Junctions have their initial state defined, and when they are in the ON state and their OFF-condition becomes true, a switching is triggered, while in the OFF state they turn back on as soon as the ON-condition evaluates to true. These switching conditions must be defined carefully to avoid Zeno executions, when an infinite number of transitions occur in finite amount of time [19].

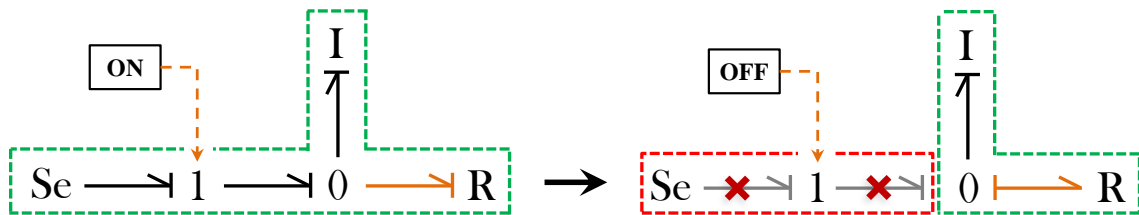
The switching behavior, just like the modulation of one- and two-port elements, relies on signals. A discrete transition can be triggered by explicit control signals originating from outside the graph, or they can be defined based on local conditions. These two types are called *controlled* and *autonomous* switching functions, respectively. For example, in a water-heating system, where a controller maintains a cycle of filling the tank, heating the water, and emptying the tank, a series of controlled switches occur according to a predetermined schedule to set the valves of the input tap and the drain. An example of an autonomous switch is opening a safety outlet on the tank if the fluid level reaches a level where a risk of overflow arises. The example is illustrated in Figure 2.



**Figure 2. Hybrid bond graph and schematic of a tank system**

The three controlled junctions correspond to the three valves in the schematic. Two of these junctions are controlled by an external logic, while the safety valve is controlled by a simple function defined of the displacement variable (i.e., height of the liquid) in the capacitor.

An important issue emerging from the notion of controlled junctions is that causality of bonds may change when the bond graph model configuration changes, because junctions may switch on and off. Every state-configuration of the set of switching junctions in the graph corresponds to a different model configuration, which may imply a different causality configuration too. We cannot ensure that a particular bond will have the same causal direction in every discrete state. An example of changing causality is shown on Figure 3:



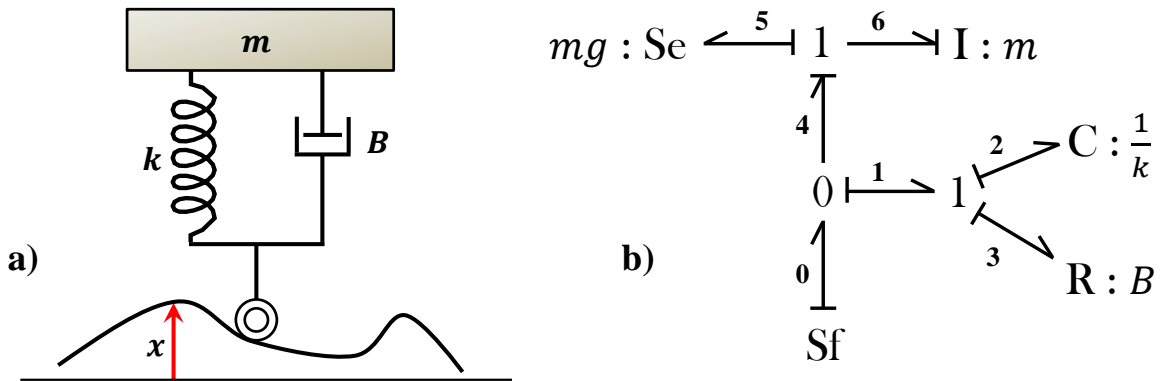
**Figure 3. Effect of switching on causality**

The determining bond of the 0-junctions is removed when the 1-junction is turned off. As a result, the bond, incident on the resistor must take on the role of the determining bond, and its causality direction changes. As described in the previous sections, different causality assignments of bonds

map to different signal-flow diagram structures, therefore, the generated hybrid simulation model should employ reconfigurable components and an online version of the SCAP algorithm.

### Example bond graph

A bond graph example from the mechanical domain is shown in Figure 4. The schematic of a simple vehicle-suspension system is on the left, including a mass attached to a wheel via a spring and a damping element [20]. Figure 4 also presents the causal bond graph of the model.



**Figure 4. Example mechanical system**  
**a) Physical model; b) Causal bond graph model**

The wheel of the modeled vehicle rolls on the ground, and the unevenness of the ground results in vertical motion input to the wheel. This is modeled in terms of the bond graph as a source of flow ( $Sf$ ), function of time, while the gravitational force is a source of effort ( $Se$ ) in the opposite direction, with parameter  $mg$ , where  $m$  is the weight of the mass, and  $g$  is the gravitational constant. The direction of the power exchange between the system and the environment is captured by the power directions of source elements' bonds. The spring is modeled with a capacitor ( $C$ ) because of its ability to store potential energy, and the capacitance is the reciprocal of the spring constant. Masses are capable of storing kinetic energy, and they are associated with inertia ( $I$ ) elements in bond graphs, and the inductance parameter is the weight of the mass. The damping element dissipates power, and should be modeled with a resistor ( $R$ ), with the damping

parameter,  $B$ . The spring and the damping element are connected via a 1-junction, because of their “common flow” relationship.

The causality of the bond graph is assigned using the SCAP algorithm. Flow and effort sources determine the causality of their bonds, but these assignments have no causal effects to propagate through junctions in this case. The next step assigns the integral causality to the inertia, and this determines the causality of the rest of the bond graph. On Figure 4b, bonds are numbered to associate the related effort and flow variables with them. Below we present the equations of the system, and derive the state-space description with the energy variables as state-variables. In the equations  $e_x$  denotes the effort on bond  $x$ , while  $f_y$  denotes the flow on bond  $y$ .

**Table 3. Bond-equations of the suspension system**

One-ports	Storages	Junctions
<b>Sf:</b> $f_0 = v(t)$	<b>I:</b> $e_6 = \dot{f}_6 m$	<b>0-junction:</b> $e_0 = e_1 = e_4$ $f_0 - f_1 - f_4 = 0$
<b>Se:</b> $e_5 = mg$	<b>C:</b> $f_2 = \dot{e}_2 \frac{1}{k}$	<b>1-junction<sub>1</sub>:</b> $e_1 - e_2 - e_3 = 0$ $f_1 = f_2 = f_3$
<b>R:</b> $f_3 = e_3 \frac{1}{B}$		<b>1-junction<sub>2</sub>:</b> $e_4 - e_5 - e_6 = 0$ $f_4 = f_5 = f_6$

The state-variables are  $f_6$  and  $e_2$ . The state space equations can be derived by eliminating the other effort and flow variables. The resulting state-space equations are presented below.

$$\begin{bmatrix} \dot{f}_6 \\ \dot{e}_2 \end{bmatrix} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\frac{B}{m} & \frac{1}{m} \\ -k & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{B}{m} & -\frac{1}{m} \\ k & 0 \end{bmatrix} \begin{bmatrix} v(t) \\ mg \end{bmatrix}$$

These equations can be used to simulate the behavior of the suspension system. The BDM Sketcher tool eliminates the procedure of deriving state-space equations by directly converting bond graphs into simulation models.

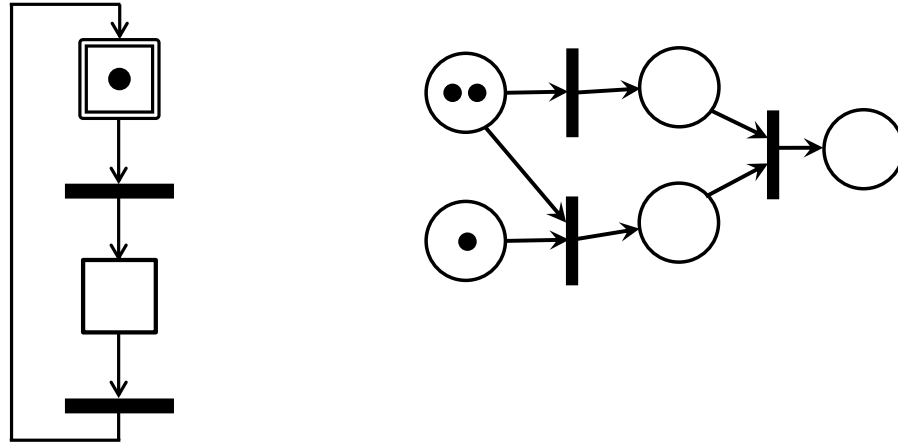
## Grafcet notation

### Standard Grafcet

The BDM Sketcher environment uses an extended version of the *Grafcet* model of computation [6], to model controller elements and software components associated with hybrid bond graphs. The standard Grafcet formalism was developed in the mid '70s, and was standardized in 1988, with the primary purpose of modeling the logic of process controllers. Grafcet is a discrete event modeling language motivated by Petri nets. There are two types of modeling elements in standard Grafcet: *steps* and *transitions*. Steps correspond to the notion of places in the Petri net notation, while transitions have the same purpose and notation as Petri nets. The Grafcet graph is a directed bipartite graph like a Petri net: transitions can only be connected to steps and vice versa.

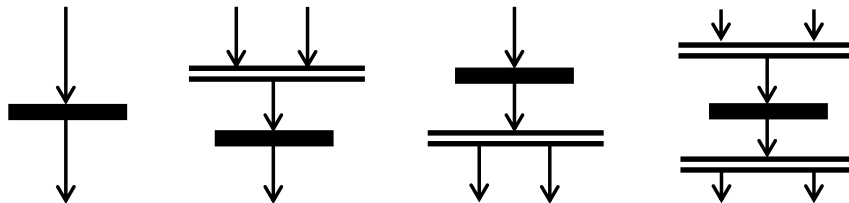
In Petri nets, we associate tokens with places, and the token distribution – also called *marking* – denotes the actual state of the model. Places and transitions are connected via directed arcs, and a transition is enabled to fire when every incoming arc has a token to consume in its source place. When the firing happens, the incoming arcs consume those tokens from the origin, and another token is produced via every outgoing edge. This works the same way in Grafcet; however a Petri net place may contain an arbitrary number of tokens, the Grafcet step is either contains a single token or it is empty. The presence of the single token determines whether that step is active or not. Grafcet steps are represented with squares on the chart, and there is no initial token-marking; only a single initial step is selected, which contains a token in the beginning. The initial step is denoted with a double square. The initial step of the model is usually drawn on the top of the chart, and the layout is designed such that tokens flow downwards.





**Figure 5. Example of Grafcet and Petri net notation**

Another major difference between Petri nets and Grafcet is the expressiveness of transitions: while Petri net transitions always require a token on every input arc and produce a token on every output arc, Grafcet transitions can be more elaborate. By default, if a step receives multiple input arcs, those enable the transition in an ‘OR’ logical relationship: having one token in one of the previous nodes enables the firing. Multiple output arcs introduce nondeterminism, as the incoming token is transferred to only one of the successor steps. To require an enabling token on every input arc, or to distribute tokens via every output arc, a different notation is used involving a double bar before or after the transition bar, respectively. From left to right, Figure 6 displays a simple transition, an ‘AND’ junction, an ‘AND’ distribution and their combined version.



**Figure 6. ‘AND’ type token-junction and distribution in Grafcet**

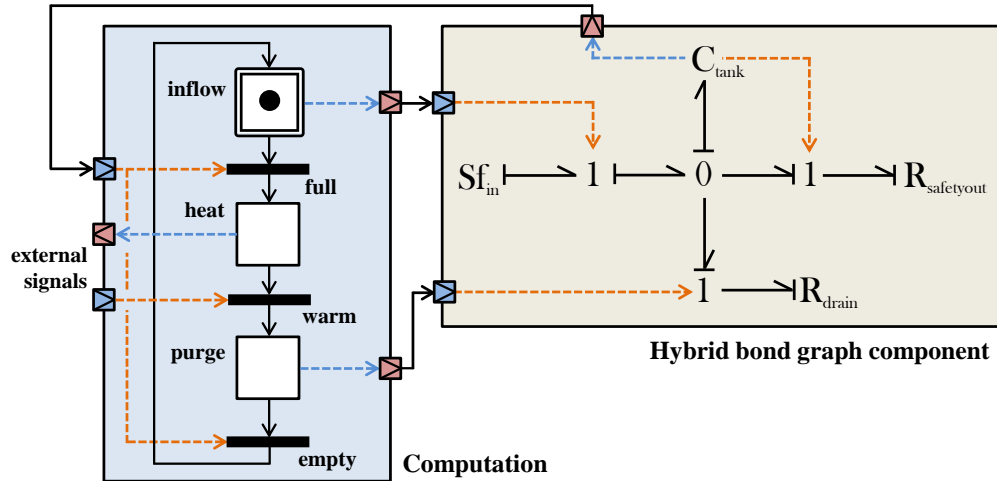
The Grafcet chart can be augmented with logical expressions. Steps are associated with actions, and transitions have a so called receptivity, which is a boolean function defined on external

signals or the current token distribution. The true evaluation of the receptivity condition is another requirement for a transition to be enabled, in addition to the token availability in their preceding steps.

### Signal-Action Graphs

The Grafcet notation was chosen by Airbus Ltd to model the functional level of the Cyber-Physical System (CPS). They developed an extended version of Grafcet called the Signal-Action Graph (SAG) [21]. The standard Grafcet refers to macro-states and hierarchical Grafcet components, but the SAG notation has a stronger support for hierarchy. Compound steps are called modes and phases. In Signal-Action Graphs the transition concept of Grafcet is generalized, the token distribution after the firing can occur based on ‘AND’, ‘OR’ or ‘XOR’ logical functions, while multiple token paths can also be joined based on any arbitrary predefined token distribution on the set of preceding steps before the transition. Elements that define these logical functions are called *junctions* in SAGs, which is different from the notion of junction in bond graphs.

Grafcet plays a dual role in the BDM modeling language. They capture functional level requirements and procedures of the CPS, and the associated signals and actions of the nodes facilitate verification of various properties of the process, such as state reachability and other safety-properties. On the other hand, Grafcet also functions as modeling language for computation blocks, and allows generating hybrid simulation models from the combined modeling language incorporating hybrid bond graphs along with Grafcet.



**Figure 7. Interaction of Grafcet and HBG**

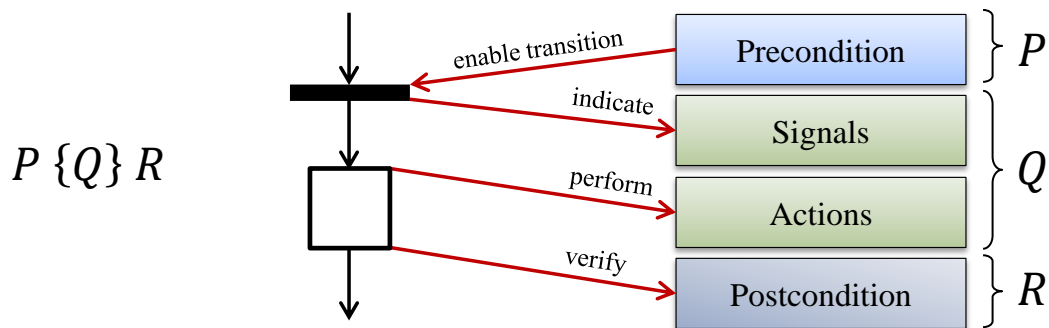
Figure 7 elaborates on the example of the water-heating tank in Figure 2. For the sake of simplicity, only the hydraulic domain is modeled in the HBG component, the temperature of the water is considered an external input signal for the Computation logics. The three Steps in the Grafcet model are (1) the filling phase, when water flows in, so the “inflow” valve is open and the “drain” is closed, (2) the heating phase, when both controlled valves are closed until the water temperature reaches the desired level, and (3) the purging mode, when the “drain” valve opens, and the warm water flows out. The safety outlet is controlled autonomously. The switching conditions of controlled junctions and the enabling conditions of Grafcet transitions are defined in their attributes based on their input signals.

### Hoare logic

As discussed in the previous chapter, both the standard and the extended version of the Grafcet notation incorporate logical expressions and external actions. In the standard version, receptivity conditions provide an additional requirement to enable transitions, while actions are associated with the activation of steps. In the Signal-Action Graph notation, this approach is extended. Every transition has a related set of signals, and a precondition expression. If the precondition evaluates

to true and the sufficient amount of tokens are available in the preceding steps, then the transition fires, those signals are indicated, and the next step – or set of steps – becomes active. Steps usually have a set of associated actions, which are performed on entry to that step. The effects of these actions are captured in a postcondition, associated with the step.

This series of actions and signal indications along with the pre- and postconditions can be expressed using Hoare logic [9]. Hoare logic was developed in the late ‘60s for program verification purposes. In the BDM Sketcher environment, it is possible to extract the requirements captured by Grafcet signals and actions in the form of Hoare triples. A Hoare triple consists of three expressions:  $P \{Q\} R$ , where  $P$  denotes a precondition, which is a logical expression that must evaluate to true before command  $Q$  can be executed. The triple expresses that if  $P$  is true, then by executing  $Q$ ,  $R$  will be always true.  $R$  is called the result or postcondition. After extracting the functional requirements of the Signal-Action Graph into Hoare triples, it is possible to perform a variety of verification operations. Preconditions associated with the transitions and the postconditions associated with the steps are mapped to  $P$  and  $R$ , respectively, while the combination of the indicated signals and the performed actions correspond to  $Q$ . The relation between Hoare triples and Grafcet is shown in Figure 8. A related example is also presented in Appendix D.



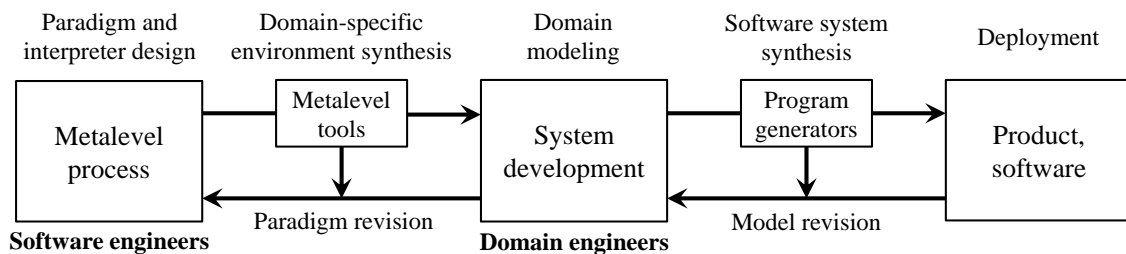
**Figure 8. Relationship of Grafcet and Hoare triples**

## CHAPTER III

### THE GENERIC MODELING ENVIRONMENT

#### Model integrated computing

The BDM sketcher tool relies on a metamodeling environment, called the Generic Modeling Environment (GME) [2] developed at ISIS in Vanderbilt University. GME is a part of a software toolset related to the concept of Model Integrated Computing [22]. GME enables developing Domain Specific Modeling Languages (DSMLs) and manipulating domain models. It employs a visual interface to aid the design of metamodels, including the abstract and concrete syntax, and then these custom modeling languages are encapsulated in a so called *paradigm*. Using the paradigm, GME allows domain experts to build their own models corresponding to the syntax of the defined language. These domain models can be manipulated through different software interfaces using model interpreters, and the final software artifacts can be automatically generated. This approach is an implementation of the Multigraph architecture, which was also developed at ISIS, Vanderbilt University [4]. The key ideas are illustrated on Figure 9.



**Figure 9. Multigraph architecture<sup>2</sup>**

There are two main activities of software development in the Multigraph architecture. One is the metalevel process, including the paradigm definition and implementation of the related software

<sup>2</sup> Figure 9 is a version of a figure taken from [22]

to manipulate models and generate output. These are the model interpreters. The actual system development is the process of building models which comply with the paradigm using the modeling environment. The first step involves software engineers, while the second involves domain experts. The paradigm is refined based on the feedback from domain experts, and then it is frozen, and used to develop domain-specific models repeatedly. The domain models are refined using the experience with the actual product.

These concepts are applied in the BDM Sketcher. The purpose of the work related to this thesis was to implement the first step of the procedure, discussed above. A paradigm for defining cyber-physical systems was created along with two model-interpreters, to generate Matlab simulation models and export the constraints defined within the models.

## MetaGME

The definition of the domain specific modeling language in GME follows the same pattern as the creation of domain models: there is a predefined paradigm that describes the metamodel for creating domain specific modeling languages, called MetaGME. While the expectation from a domain specific language is to be as specific as possible, restricting the expressiveness of the language as much as possible to prevent building invalid models, the requirement of the MetaGME language is to be as general as possible.

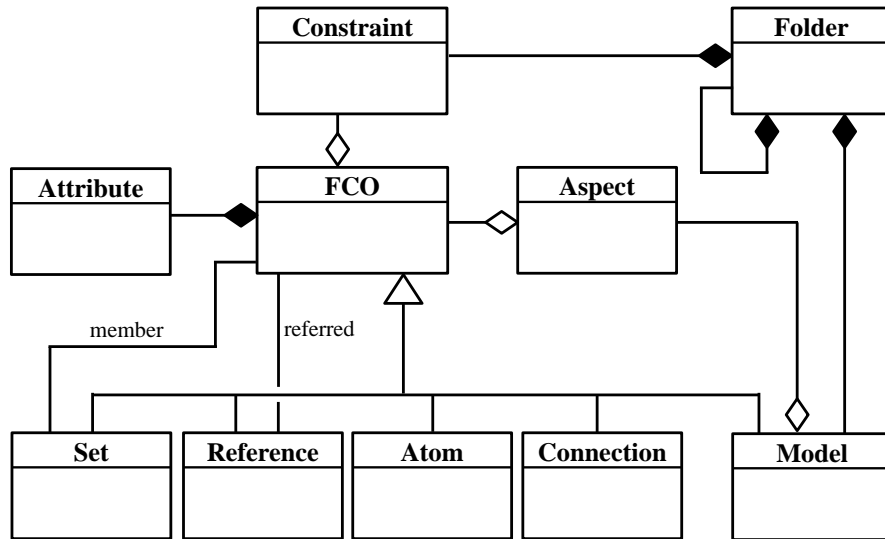
The MetaGME modeling language is defined using Unified Modeling Language (UML) as the metamodel. UML complies with the Meta Object Facility (MOF) language, and MOF is self-describing [23]. MetaGME employs various concepts to provide a most expressive domain-definition language. *Atoms* represent atomic elements and *Models* are compound types. Inheritance can be used between identical classes and *FCO* elements. The FCO abbreviation stands for First Class Object, and it is designed to allow defining of inheritance between different classes, such as to provide a common ancestor for an Atom and a Model. FCOs are abstract by

definition. The modeling elements are organized into a tree structure: every instance is contained by a Model. The root-level modeling element in MetaGME is called *ParadigmSheet*. *ParadigmSheets* can be organized into *Folders*.

*Sets* are weak containers; they may contain any number of elements from heterogeneous classes, assuming they are in the same parent Model. This containment corresponds to aggregation in UML terminology while the containment in a Model is composition. It is also possible to use *Reference* types, which can point to any instance of a chosen class in the domain model. For example, to make the modeling process easier, this concept is implemented at the MetaGME level; there are *Proxy* elements for every modeling class to allow referring to any instance of that class across the hierarchy.

Various types of *Attributes* can be connected to every modeling class, to store the type-specific fields of element instances. Using the *Connector* element, it is possible to add an *Association* between any elements, not just between instances of the same class. Attributes and visibility of connections are defined using *Association classes*. An important concept is the notion of *Aspects*. *Aspects* can contain a subset of the defined modeling elements, and every Model has an associated set of *Aspects*. They are used as a filter: when working on the domain model in GME, one *Aspect* is activated at a time for a Model, so that only those elements which are contained in the chosen *Aspect* become visible.

The MetaGME paradigm is extended with constraints, to express additional requirements that cannot be defined using the syntax of the paradigm. Typical examples are validation of *Attributes* or enforcing complex cardinality requirements. Constraints can be expressed using the standard OCL language [24]. Figure 10 presents a simplified version of the MetaGME paradigm structure, expressed with a UML class diagram.



**Figure 10. MetaGME UML class diagram**



## CHAPTER IV

### DESCRIPTION OF THE SYSTEM

#### Previous work

The BDM Sketcher tool was developed as the second phase of a research project for Airbus UK Ltd. In the first phase, a regular bond graph paradigm was built along with a model-interpreter to generate Matlab Simulink simulation models. The bond graph modeling language was augmented with additional features to accommodate the needs of the domain experts for the Airbus projects.

In this second phase of the project, the functionality of the bond graph language was extended to hybrid bond graphs, and an extended version of the Grafcet modeling language was incorporated to model the discrete behaviors of the system. Grafcet is used to model the computational elements that control the discrete junction switching across the hybrid bond graph. As discussed in the previous chapter, the Simulink signal-flow graph is built from the bond graph by adding causality information generated using the SCAP algorithm. Previously, this was done in the interpreter: the causality assignments were derived, and a static block diagram was generated. However, in the hybrid bond graph, the causality direction of bonds might change when switching occurs; therefore, performing the SCAP in the interpreter was not sufficient. The incorporation of controlled junctions necessitated implementing the SCAP algorithm in the Matlab environment, to be invoked during the simulation to reconfigure junctions' inner structure based on the current causality scenario. Because of this, most of the block-structures in the previous Simulink® bond graph library were also re-implemented to include the reconfiguration logic.

## Overview

The BDM environment works as follows. The modeling language was created as a GME paradigm, which includes the hybrid bond graph (HBG from now on) and Grafcet language. The domain models are cyber-physical systems represented with these notations. There are two model interpreters associated with the paradigm: one to generate simulation models, and the other to extract the logical constraints from the Grafcet subsystems in the format of Hoare triples. The structure of the BDM environment, including GME and Matlab is visible on Figure 11.

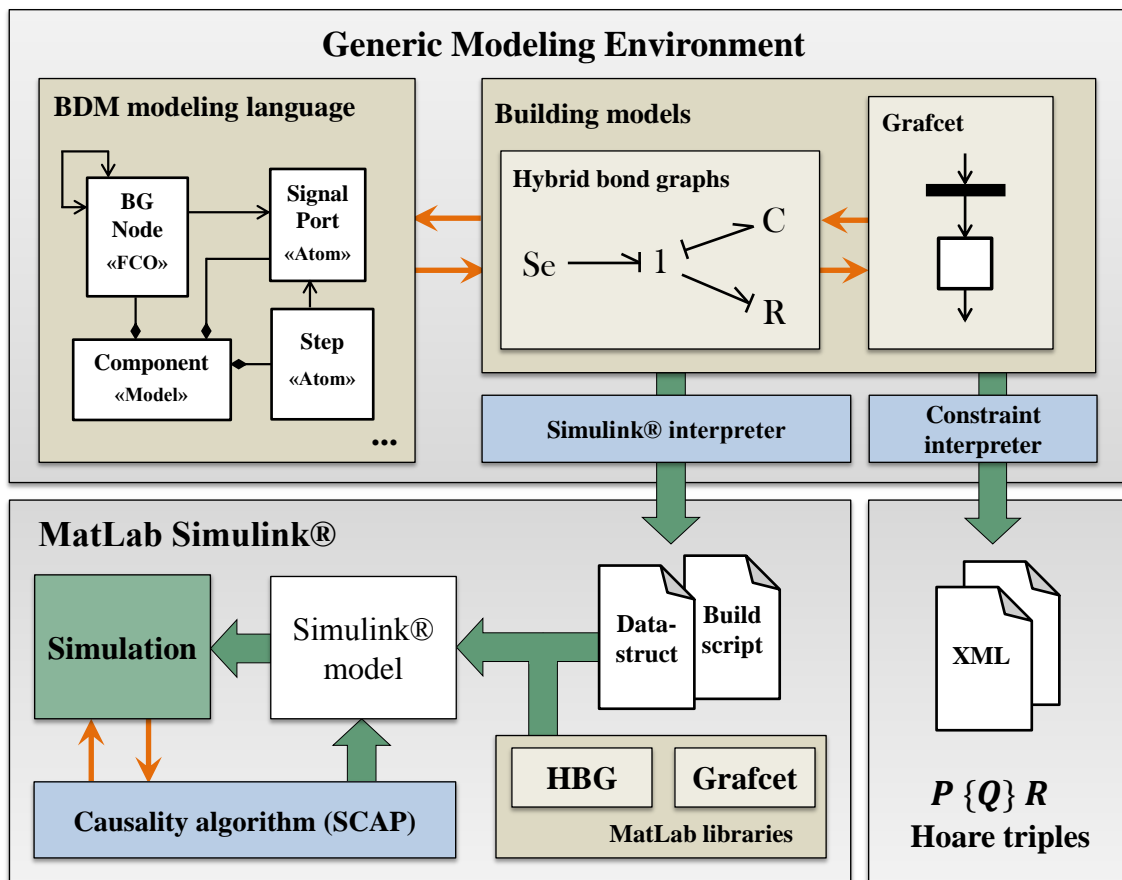


Figure 11. Structure of the BDM environment

The *SimulinkInterpreter* follows the same approach to generate simulation artifacts as the one described in [25]. It flattens the hierarchical bond graph and converts it into an intermediate representation that contains all the necessary information to perform the causality assignment

algorithm. Two outputs are generated; one is the *datastructure script*, which represents the structure of the bond graph converted into Matlab variables – this is the intermediate representation – and the other output is the *buildscript*, which is again in Matlab format. The *buildscript* runs the *datastructure script* and invokes the SCAP algorithm from the Matlab library on the stored HBG structure to create the initial causality assignment. The rest of the *buildscript* consists of a series function calls, called *buildfunctions*. Buildfunctions are also defined in the HBG Matlab library; they are used to assemble predefined Simulink® subsystems corresponding to different parts of the HBG, customizing them according to the function parameters. The *buildscript* also creates the SimEvents® structures from the Grafcet library, adds the connections and opens the finalized simulation model. During the simulation, whenever a discrete switching occurs, the SCAP algorithm is invoked to compute the new causality assignment on each bond, and the junctions are reconfigured accordingly.

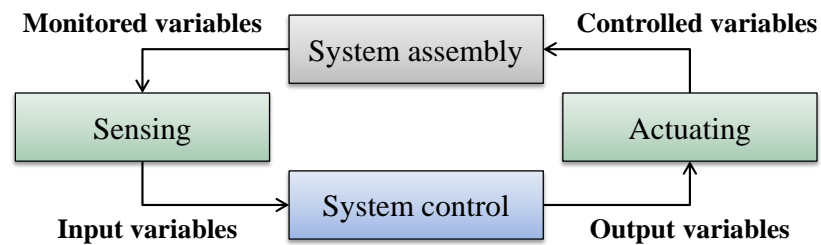
When running the *ConstraintInterpreter*, it traverses the selected Computation elements, and extracts the constraints associated with the signals, actions, pre- and post-conditions in the SAG model. The export format is a simple XML representation.

## The BDM paradigm

### Structure and hierarchy

The BDM modeling language is a composite metamodel incorporating a hybrid bond graph language and an extended version of the Grafcet notation called Signal-Action Graphs. The purpose of the language is to build and maintain domain models represented with these formalisms, and automatically generate the simulation artifacts for them. The HBG and Grafcet language is augmented with signals, and has a strong support for hierarchy. Grafcet elements can be added in distinct containers called *Computation*, and the interaction between the discrete and continuous components is defined by signals traveling across the hierarchy. There are several other types of container elements used for different purposes, and the organization of these

elements is based on the idea of the Four Variable Model approach [26], which is a system model used in the Software Cost Reduction (SCR) method of requirements definition [27]. We model a subsystem of a cyber-physical system with four components, performing operations on four sets of variables. The four components are the *system assembly*, the *actuating element*, the *sensing element* and the *system control* component. The system assembly has a set of state variables, driven by the laws of physics, representing the current motion, position, and other values in the physical system. A subset of interest is the set of *monitored variables* of the sensing component. Through the analog-digital mappings in the sensors of the sensing subsystem, these variables become the *input variables* for the system control component, which performs the necessary computations and generates a set of *output variables* that influence the system. Output variables are then transmitted to the actuating component, to be mapped back to physical variables in the system assembly, and they are called the *controlled variables*. Figure 12 visualizes the idea below.



**Figure 12. The Four Variable Model**

Because the idea of the Four Variable Model is credited to David Parnas, systems which are modeled as operations on these four sets of variables are often called Parnas systems. An essential container-type, part of the BDM hierarchy is called ParnasSystem, which implements the Four Variable Model. ParnasSystems can be organized into ParnasSystemSets to model the interactions of these Four Variable subsystems, and ports can be added in these Models (Models are compound type elements in MetaGME – discussed in the previous chapter). Ports are atomic

elements that are visible from the parent of the container, enabling connections to be added across the hierarchy levels. Inside the ParnasSystem, the modeling element called Computation contains a Grafcet model, while the other three components of the Four Variable Model can be realized using general-purpose elements, which contain hybrid bond graphs and the related signal-manipulation elements.

### Signal and power flow

Following up from the previous chapter, the bond graph language is augmented with signals. Therefore, it is important to distinguish signal paths from the power-exchange paths. However, in the Simulink® simulation model, as the signal-flow graph is generated, this separation will be lost. Signals are used to modulate the M-versions of the bond graph elements, to control junctions, or to carry any variable from one component to another. The modeling language employs different port-types for power-flow and signal-flow. Signal ports are directed, we have input and output signals. Ports connecting junctions are direction-independent, and are called PowerPorts. They are used to extend bond-connections between bond graph junctions placed into different containers in the hierarchy. A set of junctions can be connected to each PowerPort at and those ports can be connected with a bond on the upper levels – or they can be propagated further by using additional PowerPorts. A bond which is extended using one or more PowerPorts is called power-path, and will be mapped to multiple bonds: one between each junction-pair on different ends or branches of the path. PowerPorts are domain specific for the five physical domains: mechanical displacement, mechanical rotation, electrical, hydraulic and thermal. The reason for this is that domain conversions are intended to be modeled with a dedicated component employing the adequate two-port element to model the multiplicative factor of the domain conversion.

There can be several invalid configurations for both signal ports and PowerPorts or power paths; the interpreter checks the configuration before generating the simulation model, and error

messages or warnings are passed back to the modeling environment to notify the modeler if invalid configurations are found.

Signals can traverse through the hierarchy, including the Computation elements, and the Grafcet hierarchy inside them. When a Grafcet Step is active, the originating signal has a boolean *true* value, while the incoming boolean signals – if connected to a Transition – function as enabling signals.

### ControlNodes

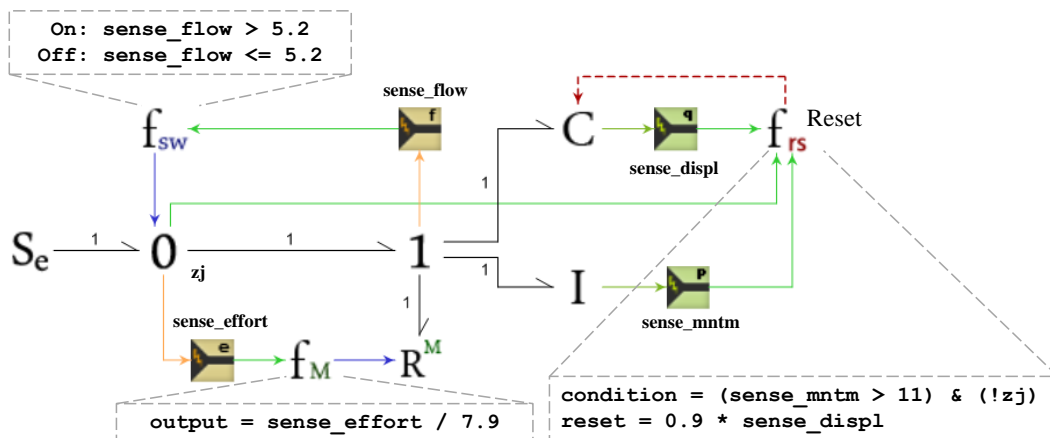
There are two types of elements in the BDM language to define real-valued and boolean-valued functions of signals. These are the *Modulation* and the *Switching* elements, respectively; they are both descendants of the abstract *ControlNode* element in the metamodel.

ControlNodes may take inputs from arbitrary signals originating from a port, a Constant, junctions, and Sensing elements. Signals coming from junctions denote whether the junction is turned on or off. This is always true for non-switching junctions. Sensing elements are visually distinct modeling elements that denote the conversion from the effort or flow variables of junctions, or energy variables of storage elements to the corresponding signals (see Appendix A for details). Then a real-valued or boolean function can be defined in their attribute using standard Matlab syntax; then the parameter of modulated one- and two-ports can be replaced with the value of the connected Modulation element. Junctions are considered controlled, when a Switching element is connected to them. In the junctions' attributes, the modeler defines the on- and off conditions as logical functions of one or more Switching inputs. Another purpose of the Switching elements is to define enabling functions for Grafcet transitions outside the Computation element.

The third element derived from the ControlNode is the *Reset* node, which implements the hybrid reset function. It is used to reset the energy variables of the connected Capacitors or Inertias. The

Reset element may have the same type of signal inputs as the Modulation or the Switching, and it defines a boolean and a real-valued function: the boolean function triggers the reset event while the real-valued function defines the new value of the energy variable.

On Figure 13, the example shows a simple RLC circuit with a modulated resistor, and a source that can be turned off by controlling its adjacent 0-junction. The actual resistance of the resistor depends on the Modulation element, which defines a function on the effort value imposed on the 0-junction. The conditions of turning on and off the same junction are defined in the Switching element, and are based on the flow value in the circuit, measured at the 1-junction. The charge stored in the capacitor is set to a new value anytime when the flux on the inductor exceeds a value, and the 0-junction is turned off.



**Figure 13. Example of the ControlNode elements**

Other features of the paradigm

The BDM paradigm contains lots of additional functionality in addition to the standard hybrid bond graph language and the Grafcet notation. Some of these were discussed in the previous sections, and we refer to the rest here. A detailed list of the modeling elements is included in Appendix A.

The paradigm includes elements to perform mathematical functions on signals, such as delay, time integral, and time derivative. The ControlFunction elements allows the modeler to write arbitrary Matlab code in the block manipulating a set of input signals to produce a set of output signals. A previously built Simulink model can be referred in a modeling element, the connections can be made via ports and it will be copied into the final system. The Pack and UnPack block can be used to multiplex and de-multiplex signal groups, while there are monitor elements which translate to Simulink Scopes to observe signal values.

## The simulation model in Matlab®

### Overview

The BDM Sketcher provides a metamodel for building cyber-physical system models. The *SimulinkInterpreter* is a model interpreter associated with the paradigm, and it is able to generate textual script-files to automatically assemble simulation models. These scripts rely on the BDM Matlab library. Models are defined in the domain language, and all the modeler has to do is to run the interpreter and the generated buildscript. The BDM Matlab library defines the mapping between the hybrid bond graph elements, and Simulink, and the Grafcet Signal-Action Graphs and SimEvents. To construct the models, both of these toolboxes are accessed via the Simulink API [28].

Simulink® is a complex cross-domain simulation platform within the Matlab environment, where the modeler can define signal-flow diagrams from blocks, implementing mathematical functions, and the framework allows performing various simulations and analyses. Simulink is an environment primarily for continuous time simulation; however, it integrates with a large set of other Matlab toolboxes, which can be used to model discrete behavior. For each hybrid bond graph element, a composite Simulink block is generated by the related *buildfunction*. It assembles the block, based on a set of parameters passed to it from the buildscript, from which it is invoked.



SimEvents® is a discrete event system modeling language included in the Matlab environment as a toolbox, and its models can be incorporated into Simulink models. SimEvents operates with tokens and provides several blocks to build queuing systems and perform time analyses. Among the Matlab toolboxes, the semantics of SimEvents is the closest to Grafcet models, because the notion of token is preserved. For example, the logic of distributing tokens to multiple token paths can be done dynamically, instead of allocating static states for each combination, which is the way to implement this in Stateflow.

### The Simulink model interpreter

The Simulink model interpreter processes a single model or a set of models in GME, and generates two output files for each of them. These are the *datastructure script*, and the *buildscript*. The first task of the interpreter is to traverse the hierarchy of containers and unite the fragmented bond graph segments, which are connected via PowerPorts. The traversal is a simple Depth-First Search (DFS), because internal containers in the ParnasSystems are hierarchical. First, the interpreter converts the power paths to simple bonds, and then creates an internal object-oriented representation of the bond nodes with a map in each node, identifying adjacent nodes. This flattened representation is then exported to the datastructure script in the format of Matlab variables. This intermediate representation contains all the properties of bond graph nodes which are necessary to perform the SCAP algorithm. To generate the simulation model, the user runs the buildscript, and the first task in the buildscript is invoking this datastructure script to feed the flattened bond graph into the Matlab workspace, to run the SCAP algorithm and create an initial causal assignment. After exporting the datastructure script, the interpreter generates the buildscript. First, a skeleton of the containers is constructed. Because power paths and signal paths are both mapped onto the same ordered set of Simulink ports for each container, port-numbers are carefully assigned, and these ports are added. Computation elements are also processed in the first traversal cycle and SimEvents blocks are generated inside them. In a second

traversal cycle, the hybrid bond graph elements and other specific elements are added. Connections are collected into an intermediate set during both traversal cycles, and they are added only after every node is processed. One of the greatest challenges of developing the Simulink interpreter was to find the correct order of processing the different modeling elements, and to serialize the tasks of the buildscript to incrementally build the model, and avoid dependency conflicts in its execution.

### HBG-Simulink mapping

The BDM Sketcher tool converts hybrid bond graphs to Simulink signal-flow graphs. During the conversion, atomic bond graph elements become composite Simulink subsystems. The parameters of these subsystems are taken from GME, and they are masked, so that they can be refined easily from the Matlab GUI.

Having the causality assigned to each bond in the bond graph, it is straightforward to generate a static block diagram based on the mathematical functions implemented by bond graph elements. However, for hybrid bond graphs, we have to take into account the changing causality of bonds. This requires additional logic in certain components, and the causality information of bonds has to be maintained during the simulation. As the next section describes, we handle the changing causality by generalizing the structure of junctions, two-ports and resistors; the rest of the elements have fixed causality. The modified elements implement every required routing behavior, and use the appropriate one at any point in time, based on the current causality.

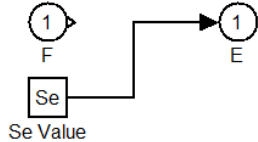
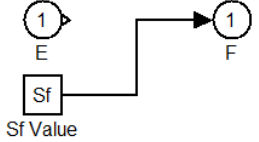
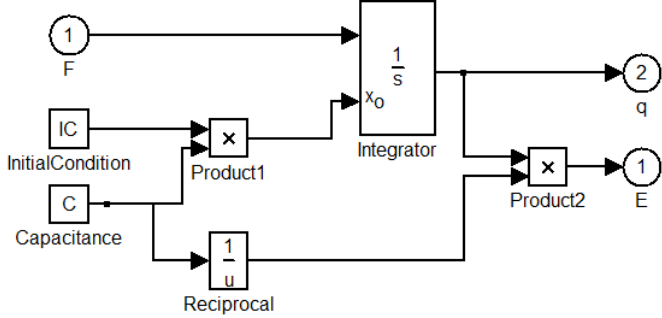
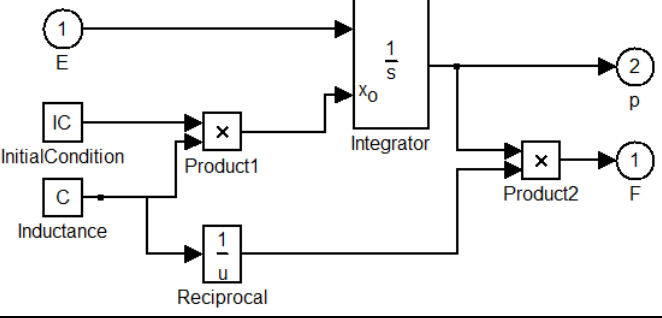
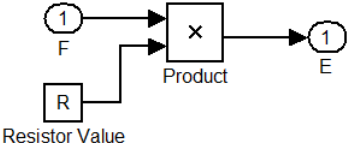
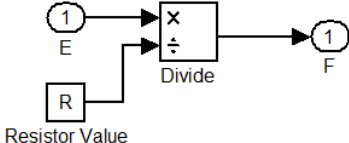
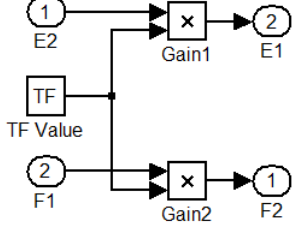
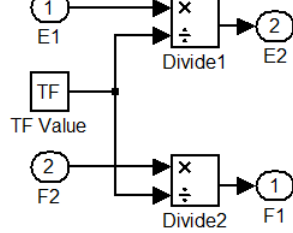
The difference between regular and modulated elements in Simulink is that regular elements have a “Constant” Simulink block inside them, determining their parameter (or modulus), while modulated elements take those as an input, via a port. Modulated elements may cause algebraic loops when the modulation function depends on the output of the modulated block. This usually slows down the operation of Matlab solvers significantly. When the duration of the simulation is

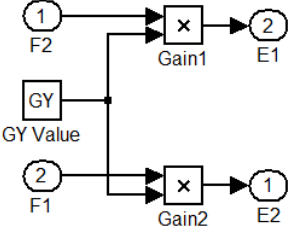
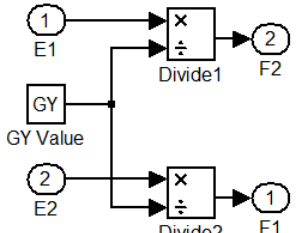
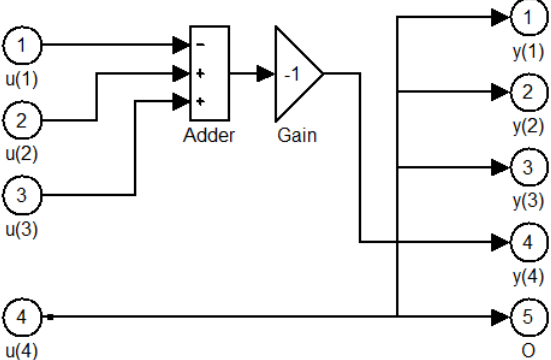
critical, a unit delay can be used in modulated elements as an approximation to break up the algebraic loop. The initial value for the delay element can be chosen heuristically, and during the first few sampling hits, the simulation approximates the real value. Using an appropriately small maximum sampling step size eliminates the negative effects of the heuristic initialization of the delay.

In this section, we present the Simulink versions of non-modulated bond graph elements with *fixed* causality. Fixed causality means that all the adjacent bonds of the element have the same causality direction across every possible discrete state. The next section explains the implementation of the switching logics. Table 4 presents the Simulink block structures.

Every Simulink subsystem implements the block structure, presented earlier in Table 2 (Chapter 2). The resistor with changing causality, junctions and two-port elements are discussed in the next section. We should note that in modulated storage elements, we had to ensure that the law of conservation of energy was not violated. This requires using a block structure, where modulating the capacitance and inductance only affects the output flow and effort values, respectively, and the amount of energy stored as displacement and momentum remains the same.

**Table 4. Simulink structure of regular bond graph elements**

Name	GME icon	Simulink structure	
Sources	$S_e$ $S_f$		
Capacitor	$C$		
Inertia	$I$		
Resistor	$R$	 <p style="text-align: center;"><b>flow input</b></p>	 <p style="text-align: center;"><b>effort input</b></p>
Transformer	$TF$	 <p style="text-align: center;"><b>effort direction is the opposite of power flow direction</b></p>	 <p style="text-align: center;"><b>effort direction is the same as the power flow direction</b></p>

Gyrator	GY	 <p><b>effort is an output on both sides of the gyrator</b></p>	 <p><b>effort is an input on both sides of the gyrator</b></p>
Junctions	<p style="text-align: center; font-size: 2em;">0</p> <p style="text-align: center; font-size: 2em;">1</p>	 <p style="text-align: center;"><b>Example 0-junction</b></p> <p style="text-align: center;"><b>4 bonds: one in, three out; the determ. bond is an outbond</b></p>	
Controlled junction	<p style="text-align: center;">↓</p> <p style="text-align: center; font-size: 2em;">0</p>	<p><i>Controlled junctions are extended with switches to impose zero output on all their bonds. They also store the current value of the guard conditions in global data-storage elements to facilitate the causality update. The structure is shown on Figure 14.</i></p>	

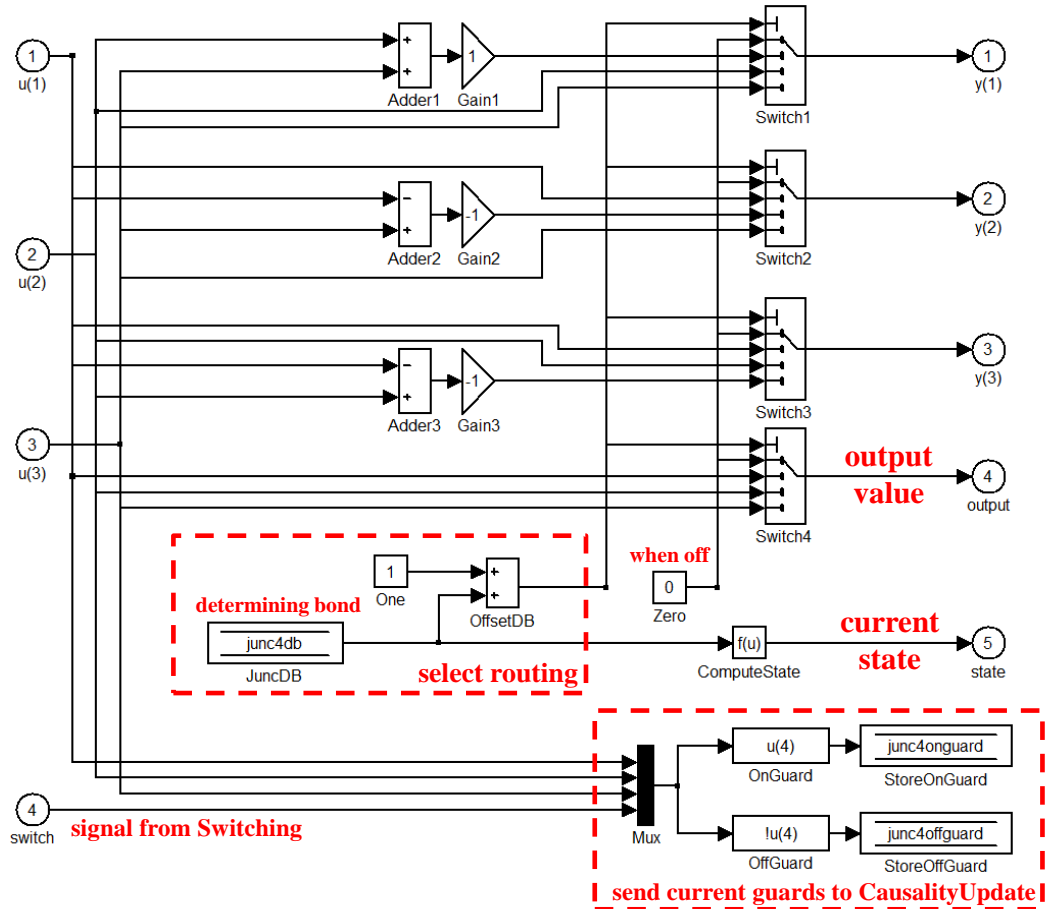
### Hybrid switching structures

As discussed in the previous chapter, the signal-flow structure of the hybrid bond graph changes with every discrete mode transition if the causality direction of one or more bond changes. These scenarios are enumerated during the construction of the model, and the required switching mechanisms are added to the block-structure during the build process. As mentioned, the flattened bond-graph is loaded into the Matlab workspace, and then the SCAP algorithm is invoked to derive the initial configuration. This algorithm computes the causality assignment for every combination of the states of controlled junctions. As mentioned in the previous section, a hybrid bond graph element is called *fixed* if the causality of its adjacent bond(s) remains

unchanged in every possible discrete state. We should note that both controlled and regular junctions can have fixed or variable causality. The procedure assigns a determining bond to every junction, and it creates a list of the fixed junctions [25].

The discrete reconfigurations are handled by the *CausalityUpdate* procedure, which is implemented as a set of Matlab functions. This is invoked using a Level-2 Matlab S-function [29]. As discussed in Chapter 2, the causality of the source and storage elements are always fixed because of the integral causality assumption [16]; therefore, the change of bond directions affects the junctions, the resistor, and the two-port elements only. To maintain the current causality information across mode changes, data-storage blocks are allocated to store the identifier of the determining bond for each non-fixed junction during the simulation. Controlled junctions store their evaluated guard conditions in another set of data-storage blocks in every sampling step, and then the *CausalityUpdate* method is called to determine whether a discrete switch has occurred or not based on these guard conditions. When a switch occurs, the SCAP is invoked to update the data-storages with the current determining bond identifiers. Based on this information, the appropriate signal routing structure can be applied in the non-fixed elements.

Port directions cannot change in Simulink, therefore, the effort-flow roles of ports change instead. Non-fixed junctions implement every causality scenario, and the active routing structure is selected with multiport switches, controlled by the data storage blocks, containing the identifier of the current determining bonds. The internal structure of a hybrid, non-fixed junction with three bonds is displayed on Figure 14 as an example.

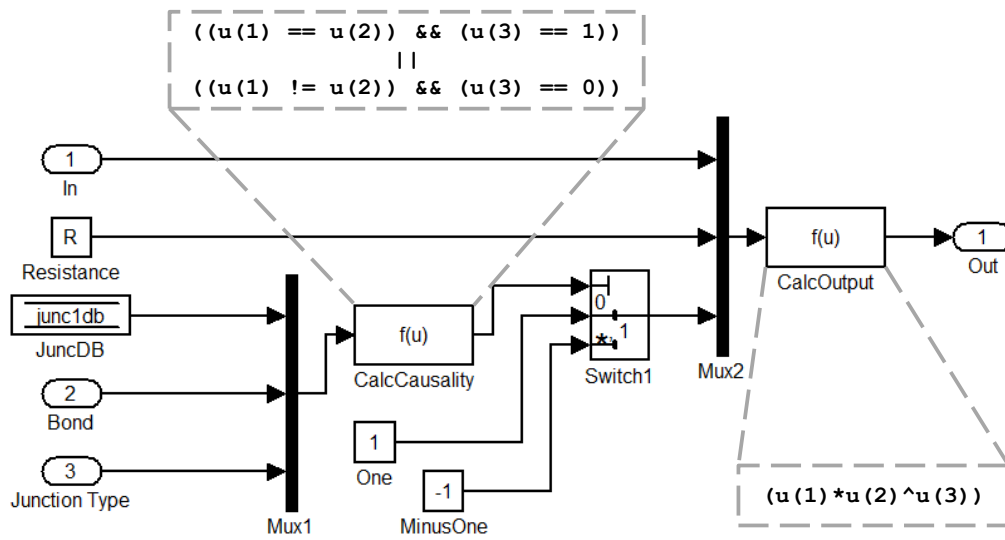


**Figure 14. Internal structure of a hybrid, non-fixed junction**

When controlled junctions turn off, they ignore inputs, and force zero output on all of their bonds. The section outlined on the bottom right of Figure 14 performs the evaluation of the guard conditions to store them for the CausalityUpdate method to access. Junctions have an extra output port, where the common value of the junction is routed out to facilitate the use of signals. The common value is the effort of the determining bond for 0-junctions and the flow of the determining bond for 1-junctions. Controlled junctions have one more additional port, where the output denotes their current on-off state.

In resistors and the two-port elements, the causal assignment of adjacent bonds only determine whether the resistance parameter or the modulation factor is applied to the signal as division or

multiplication. This is again decided based on the values read from these data-storage blocks with the current causality information. The structure on Figure 15 implements the varying causal interpretation and the corresponding behavior for a resistor. The same idea applies to the two-port elements, TFs and GYs. The “CalcCausality” block determines which operation should be applied based on the (1) identifier of the current determining bond for the adjacent junction, (2) identifier of the bond in the adjacent junction’s bond-map on which the current element is connected and (3) the type of the adjacent junction. Then the appropriate operation is applied in the “CalcOutput” block.



**Figure 15. Simulink structure of a resistor with changing causality**


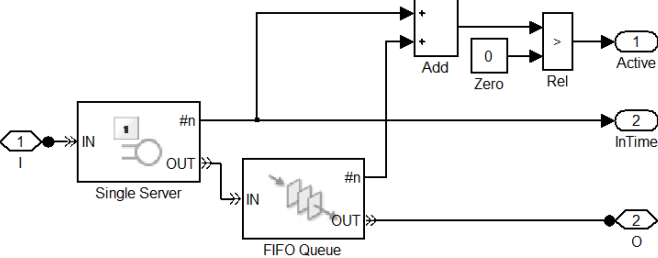

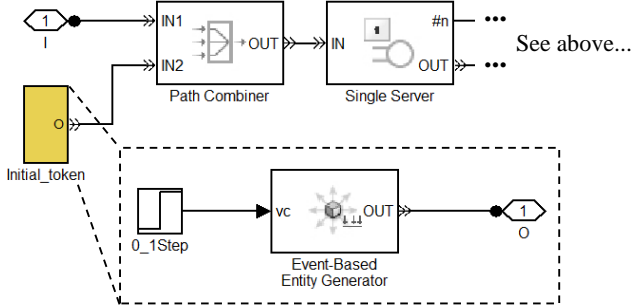

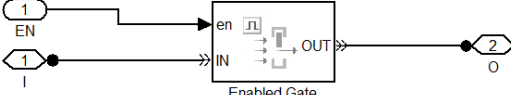
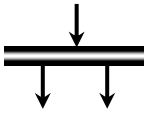
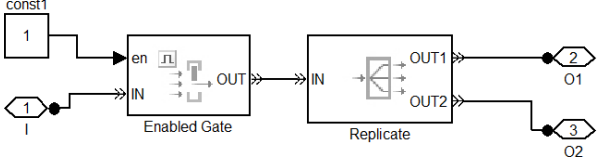
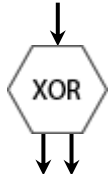
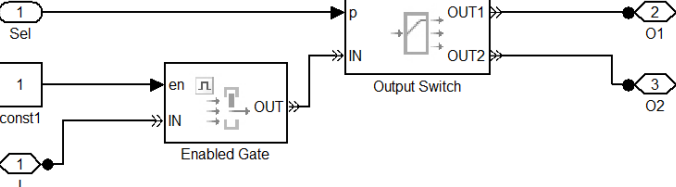
### Grafcet-SimEvents mapping

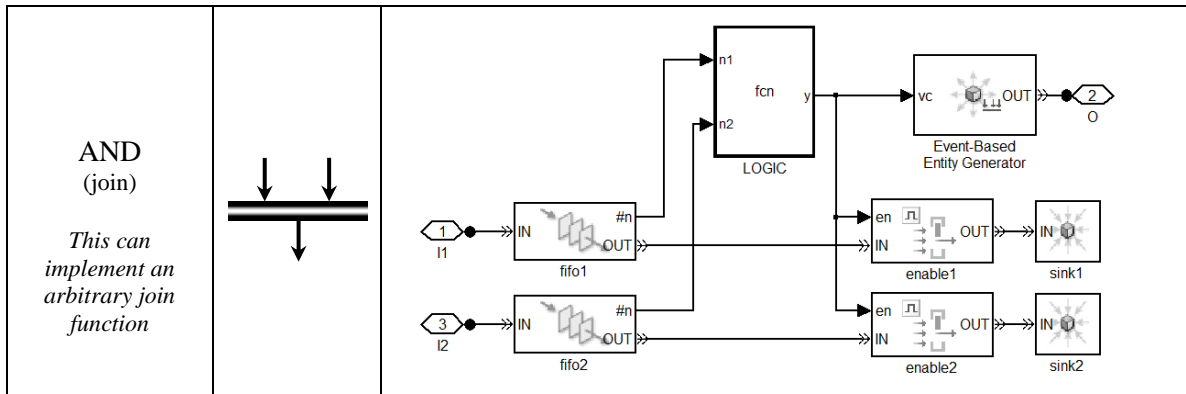
The “Computation” element of the BDM modeling language contains Grafcet diagrams to model the discrete modes of the Cyber-Physical System. When the simulation model is generated, Grafcet models are mapped onto SimEvents subsystems. The interface to the continuous part of the model includes signals, routed into the SimEvents system to enable Transitions; and signals routed out from the Computation to trigger mode switches. The interpreter flattens the hierarchical Grafcet model, and generates a new layout for the subsystem. Because Grafcet elements require less customization than bond graph elements, we employ a block-library of



prebuilt SimEvents blocks instead of buildscripts. Table 5 shows the SimEvents block diagrams generated from the GME Grafcet elements.

**Table 5. SimEvents subsystems for Grafcet entities**

Name	GME icon	Simulink structure
AtomicStep		
Initial AtomicStep		
Transition		
AND (distribution)		
XOR (distribution)		



A minimum execution time must be provided for each Grafcet Step in the model. This facilitates time-triggered execution of the Computation model when using transitions that are enabled by default. To implement this in SimEvents, the “Step” block contains a Server element that has an associated processing time. After the time is up, the token is emitted and stored in a temporary Queue (still inside the “Step” block), until the following Transition enables and the token proceeds. A step is considered active, if the internal Server *OR* the Queue contains a token. When transitions are enabled by default, tokens spend zero time in Queues. The “InitialStep” block includes additional elements to generate a single token initially. Transitions are implemented with an “Enabling Gate” block in SimEvents, which prohibit or enable tokens to pass through based on a boolean control signal. The modeling language implements two types of distribution junctions, the ‘AND’ and the ‘XOR’. The ‘AND’ junction replicates the incoming token and distributes a new token on each outgoing path, while the ‘XOR’ block uses an “Output switch” to determine the path on which the incoming tokens proceeds. Joining multiple token paths is done with the same modeling element as the ‘AND’ distribution; the interpreter determines the required SimEvents block based on the number of input and output arcs. The “Join” subsystem contains a Queue element for each incoming path to store the tokens temporarily. The “LOGIC” block defines an arbitrary boolean function based on the number of tokens in each queue, and when this function evaluates to true, tokens are deleted and a new token is emitted via the output.

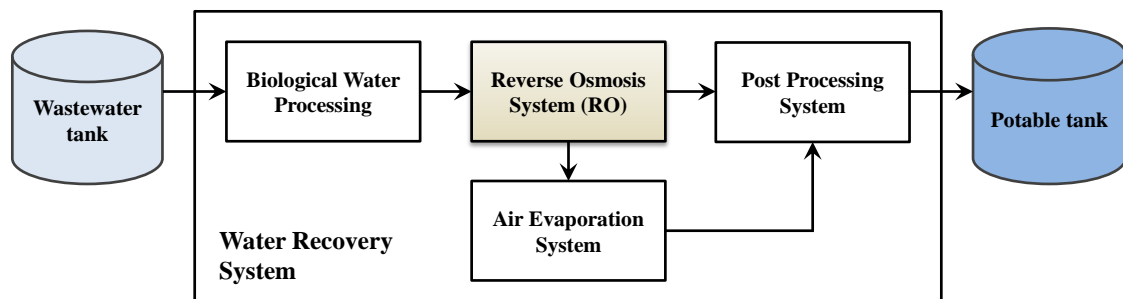
## CHAPTER V

### CASE STUDY – THE REVERSE OSMOSIS SYSTEM

In this chapter we present a case study to demonstrate the capabilities of the BDM Sketcher tool. The subject of the case study is a subsystem of the Advanced Life Support System (ALS), built by NASA, for long-duration manned missions [12]. A previous project has already been completed at the Institute of Software Integrated Systems, Vanderbilt University, to perform model-based diagnosis on the Water-Recovery subsystem of the ALS [30]. However, the previous project only implemented one mode of system operation. In this project, we build a more complete hybrid model with three modes of operation.

#### The Water Recovery System

The studied system is the Reverse Osmosis (RO) subsystem of the Water Recovery System part of the ALS. The purpose of the Water Recovery System is to recover potable water from wastewater. The RO subsystem operates on the effluent of the Biological Water Processing subsystem that has already removed the organic matter from the wastewater, and removes the inorganic impurities from the liquid.

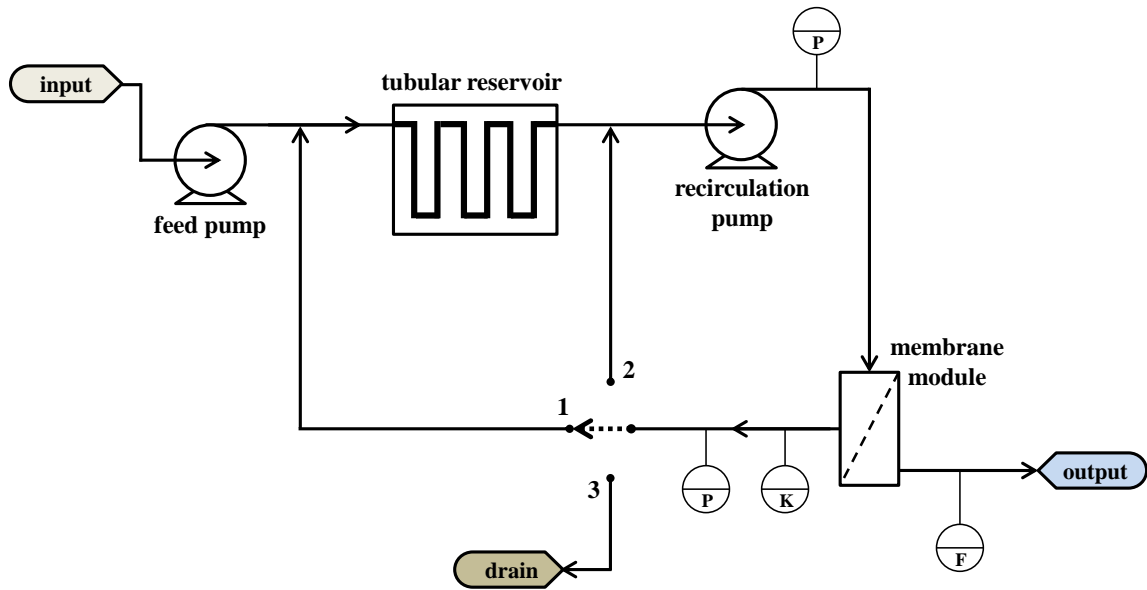


**Figure 16. The NASA Water Recovery System**

In the RO system, the water is circulated at high speed, and is pushed through a membrane. This process typically cleans about 85% of the water, which is then fed into the Post Processing system to apply Ultra-Violet treatment and produce potable water. The remaining 15% of the input cannot be cleaned efficiently within the RO system, because of the increased concentration of brine. This effluent is periodically purged into the Air Evaporation System, where it is continuously evaporated and condensed until the rest of the water is recovered.

### Operation of the Reverse Osmosis system

The RO system is a Cyber-Physical System; it has three modes of operation, and the discrete mode transitions are triggered by a controller, based on values monitored in the system. Figure 17 shows the schematic of the system.



**Figure 17. Schematic of the RO system**

The feed pump keeps pushing water extracted from the Biological Waste Processor (BWP) into the main RO loop. The recirculation pump boosts the liquid flow rate in the loop, thus facilitating pushing the water into the membrane module. Some of the water passes through the membrane, and on the other side, the filtered permeate leaves the system towards the Post Processing System.

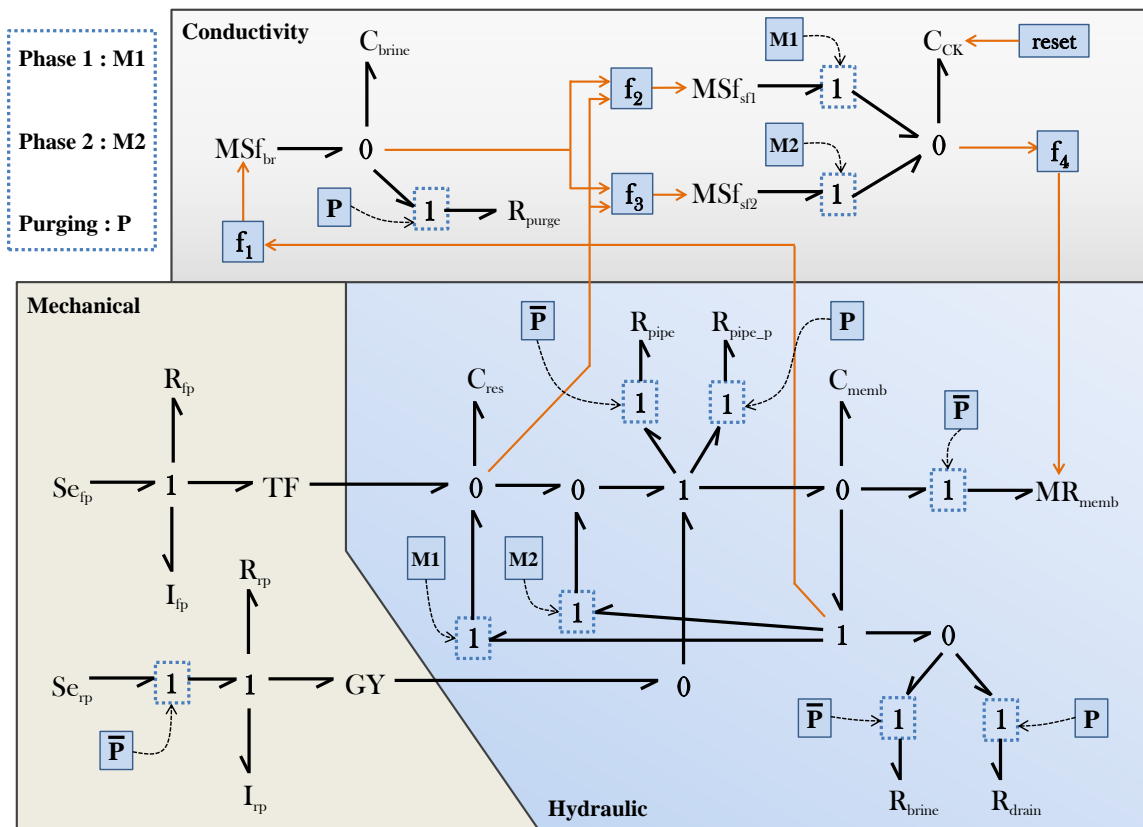
The rest of the water flows back through the loop. The tubular reservoir helps balance fluctuations in the flow through the loop. Inorganic impurities collect on the membrane, and also increase the concentration of impurities in the circulating water. In Mode 2, a valve, which controls the direction of the liquid flow in the back-flow pipe is turned to position 2 (see the schematic on Figure 17), so that the liquid, flowing back from the membrane is routed straight back to the recirculation pump, skipping the reservoir. This smaller circulation loop increases the pressure and flow rate, causing more water to permeate through the membrane, and the brine concentration in the system to increase more rapidly.

Because the throughput of the membrane degrades as the water gets dirtier, after some time, the concentration reaches a level where very little water can be pushed through the membrane. This causes a transition to the *Purge Mode*, where the recirculation pump is turned off, the routing valve is set to position 3 on the back-flow pipe, and the feed pump slowly pushes out the remainder of the liquid through the drain, into the Air Evaporation System.

### Bond graph model of the RO system

The hybrid bond graph model of the RO system structure was built during the previous work on the Water Recovery System, described in [30]. A few changes and refinements were applied to the bond graph, and we used a different set of parameters that were measured more recently at NASA Johnson Space Center. The detailed list of the parameters is available in Appendix B. The model involves two physical domains: (1) the mechanical rotation, and (2) the hydraulic domain, and a fictive domain with no direct physical representation, the *conductivity domain*. The pumps are modeled as ideal effort sources in the mechanical rotational domain, while the pipe system is modeled in the hydraulic domain, and the flow routing is implemented using a set of controlled junctions.

The *conductivity* ( $K$ ) is the measure of the concentration of impurities in the water. Because the current conductivity value affects the resistance of the membrane, we include a bond graph fragment to compute the conductivity value in the system. This segment of the bond graph does not have a direct physical representation; we associate the conductivity value with the effort value of a capacitor element (this captures the accumulation of impurities in the water), and construct the bond graph such that it corresponds to the equations governing the conductivity. The complete bond graph, including the three domains is shown on Figure 18.



**Figure 18. Bond graph model of the RO system**

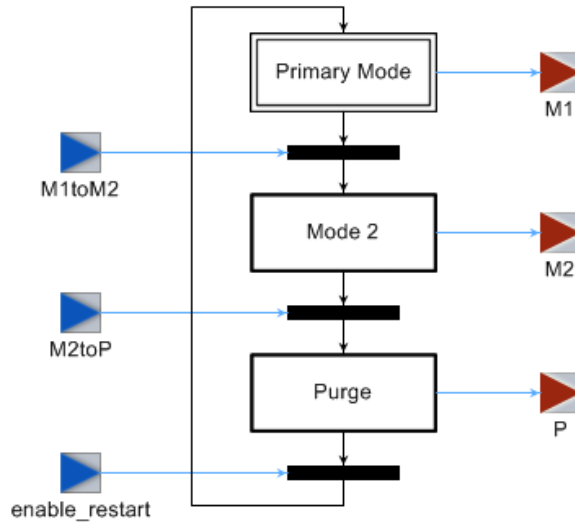
The two pumps are modeled in the mechanical rotation domain, as sources of effort ( $Se_{fp}$  and  $Se_{rp}$ ); they maintain a constant torque of the rotor. The rotational inertia of the rotor and the power dissipation associated with friction are modeled with inertias and resistors connected to the

sources with 1-junctions ( $I_{fp}$ ,  $I_{rp}$ ,  $R_{fp}$ ,  $R_{rp}$ ). The source element of the recirculation pump can be disconnected with a controlled 1-junction, but the resistor and the inertia still remains attached to the pipe, because the water flow still has to push the rotor of the pump when it is turned off. A transformer and a gyrator connect the rotational domain to the hydraulic domain. Hydraulic bond graphs are constructed such, that a 0-junction is associated with every location that has different pressure level, and the connections are defined by 1-junctions [16]. The tubular reservoir is modeled as a capacitor ( $C_{res}$ ), while the membrane module is modeled as a capacitor ( $C_{memb}$ ) and a modulated resistor ( $MR_{memb}$ ). The resistor is detached in the Purge Mode, because no fluid passes the membrane during that. The pipe between the membrane module and the reservoir is modeled with two resistors ( $R_{pipe}$ ,  $R_{pipe_p}$ ); one is used in the first two modes, and the other, having different resistance is activated in the Purge Mode. The membrane resistor is modulated by a signal, which is defined as a function of the current value of conductivity, which is computed in the conductivity domain. The back-flow pipe is modeled with two resistors: one has a larger resistance ( $R_{brine}$ ) and is used in Mode 1 and Mode 2 when the water circulates, and another is used when the water is purged from the RO system ( $R_{drain}$ ).

The conductivity domain includes three sources of flow; all of them are modulated with some function of signals coming from the hydraulic domain. These are the flow rates of the back-flow pipe and the input pipe of the membrane. The conductivity is measured as the effort value of a capacitor in the conductivity domain, and is transmitted back to the Pipe system as a signal to modulate the membrane resistor. The transition from the Purge Mode to Mode 1 is triggered when the membrane capacitor gets completely empty. This event also triggers the hybrid reset function associated with the capacitor on which the conductivity is measured: the displacement is reset to the conductivity of the incoming fluid.

Figure 18 is does not include the causality information, since it represent the causality changes across the three modes. The causal bond graphs are included in Appendix C. In Figure 18, the

conditions of the junction-switches are denoted by boxes connected to the junctions via dashed arrows. These boxes include a logical condition referring to modes names, in which the junction is on, or off, if a bar is visible above the mode name. We should note that in the BDM modeling language, junctions are also controlled with signals, originating from the Computation subsystem. The structure of the Computation subsystem is shown in Figure 19.



**Figure 19. Grafcet model of the three modes of operation**

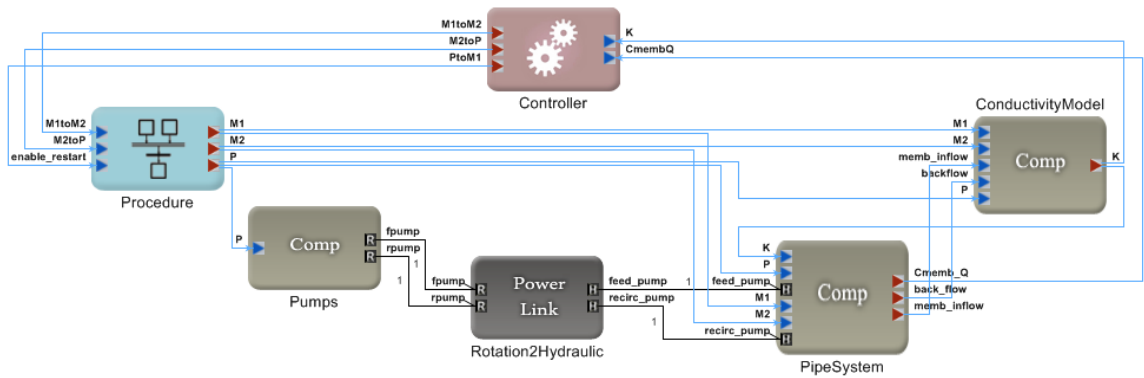
### Simulation

The RO system model was built in GME, using the BDM paradigm. The model interpreter generates the buildscript, and the simulation can be performed on the constructed model. We built two versions of the RO system: one is completely time triggered, and the other relies on monitoring signals. We present the monitored version here. The GME model is shown on Figure 20.

The Procedure component contains the Grafcet model, including the three Steps to model the three discrete behavior modes of the system. The Controller component generates the enabling signals for the Grafcet transitions, and the Grafcet subsystem produces the control signals for the switching junctions. The black connection paths are bonds, propagating between components via PowerPorts, while the blue lines are signal-connections. Each part of the bond graph, which

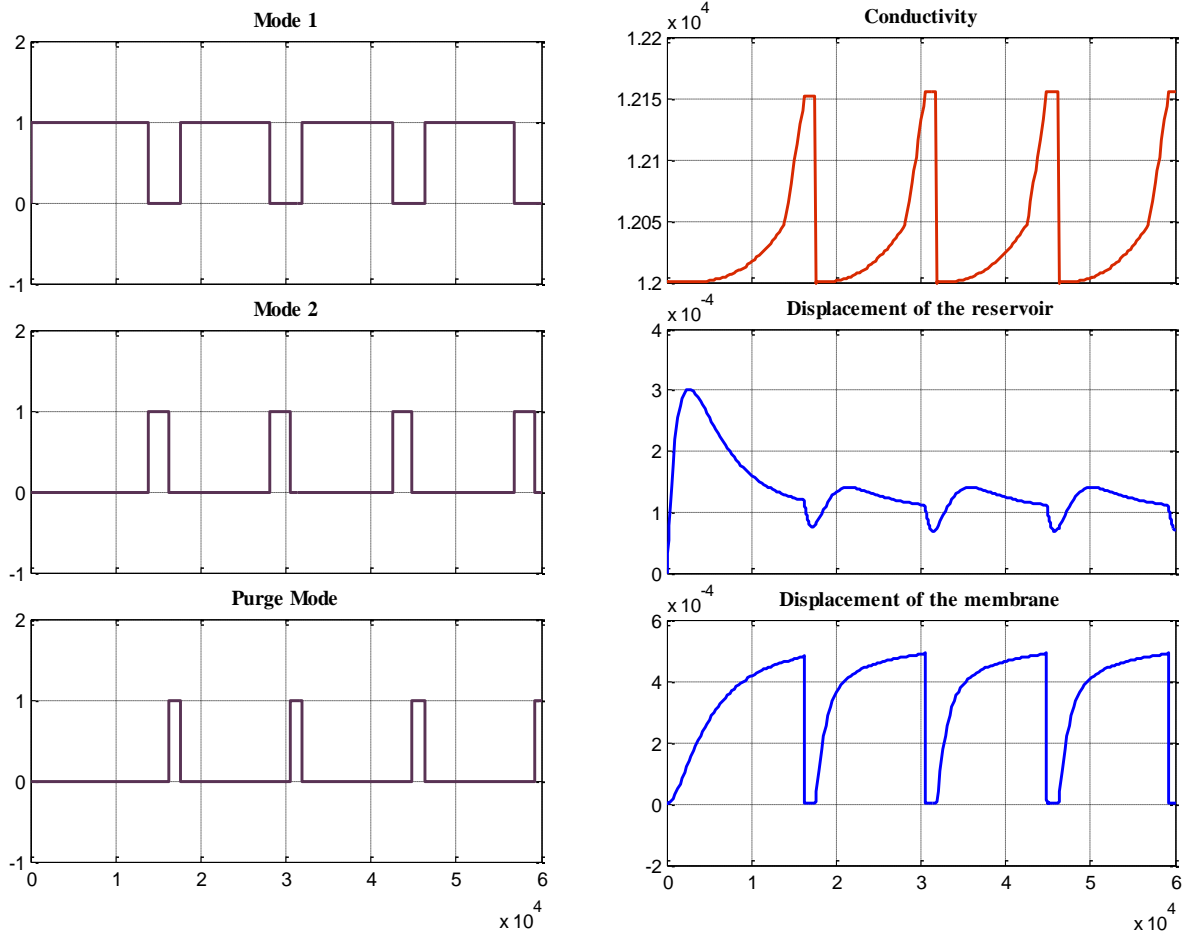


corresponds to a different domain, is built in a distinct Component. The PowerLink container represents the conversion between the mechanical rotation and hydraulic domain.

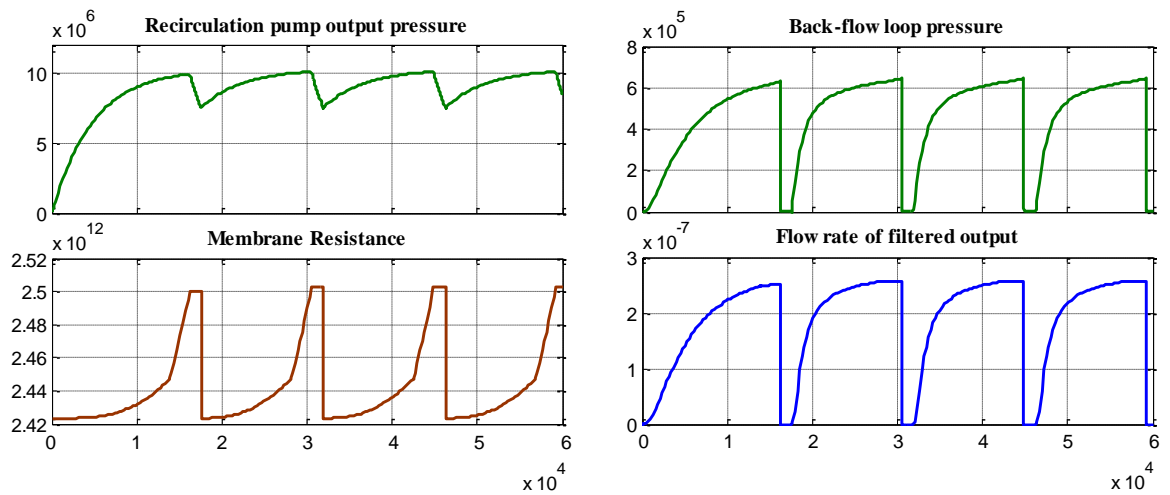


**Figure 20. GME model of the RO system**

During the simulation in Simulink, we monitored four physical variables, which are marked on the schematic, on Figure 17. These are (1) the pressure at the output of the recirculation pump, (2) the pressure of the back-flow pipe, (3) the flow rate of the filtered output of the membrane module, and (4) the conductivity of the water. In addition to these, the liquid level in the hydraulic capacitors (the tubular reservoir, and the membrane module), the mode-switch signals, and the computed membrane resistance were monitored. We started the simulation from the initial state: the pumps were turned off and the tanks were empty. Because of this, the first cycle is different from the rest, as the tubular reservoir had to fill up, and it does not get completely empty in the Purge Mode. We defined the mode switches based on the given parameters and simulated 4 full cycles, which resulted in a simulation length of 60000 seconds, more than 16 and a half hours. We present the simulation results below.



**Figure 21. Mode switches, conductivity and hydraulic displacement values**



**Figure 22. Pressures, output flow and membrane resistance values**

Figure 21 shows the mode changes triggered by the Grafset interpreter, the change of the conductivity, and the liquid level in the tubular reservoir and the membrane module. When the conductivity reaches 30% of the maximum allowed conductivity increment, Mode 2 activates, and the water keeps circulating faster until it reaches the maximum level, where the Purge Mode starts. In the Purge Mode, the recirculation pump turns off as visible on its output pressure, and the membrane empties almost instantaneously as the feed pump pushes cleaner water in the system. The tubular reservoir does not get empty during the purge, but the liquid level drops noticeably. Figure 22 shows the recirculation pump output pressure, the back-flow pressure, the permeate output flow rate, and the resistance of modulated membrane resistor. The permeate flow rate and the pressure in the back-flow loop keep increasing during the first two periods, and drop back in the purge mode. The pressure at the output of the recirculation pump converges to a steady state, but falls back in every purge mode, when the recirculation pump turns off, and only the feed pump continues to operate. The computed membrane resistance follows the change of the conductivity.

The Reverse Osmosis System was built at the NASA Johnson Space Center as part of the Water Recovery System. Our simulation results matched the expected values, derived from the test data measured during experiments on real system [30].

## CHAPTER VI

### DISCUSSION & CONCLUSIONS

#### Summary

Building preliminary behavior models for complex Cyber-Physical Systems is an essential step of developing vehicles, aircraft, and other large-scale systems, where computational devices are embedded in a physical environment. The bond graph formalism is an abstract energy-based mathematical description, which is perfect for modeling cross-domain physical systems. While bond graphs capture the system structure and the continuous-time behavior evolution of the physical system, hybrid bond graphs also provide a way to incorporate discrete mode changes in the model, applying the standard hybrid automaton model of computation. To perform hybrid simulation, we have to model the discrete phases, modes and states that the HBG models exhibit, and define a two-way interaction between the computation model and the physical structure of the system, to monitor state variables and control mode trajectories.

In this thesis, we introduced a composite modeling language based on hybrid bond graphs, where we employ an extended version of the Grafcet formalism to model the computation elements and control the switching behavior of the bond graph. Grafcet is a token-based discrete events system with Transitions and Steps, alike to Petri-nets. The hybrid bond graph domain is augmented with signals, so that the modeler is able to define the interaction between the computation and the physical subsystems. Such composite models encapsulate the three layers of Function, Behavior and Structure, and allow for generating hybrid simulation models to verify system properties.

The result of this work is a visual modeling environment, in which modelers are able to construct hybrid models by formally describing both the continuous and discrete state evolution. The

models preserve the hierarchy in the system, and capture the three levels of structure, behavior and function as well. Domain models are built using the visual interface of the Generic Modeling Environment, and the generation of hybrid simulation models is completely automated by the model-interpreter. The simulation models are built using the Simulink® and SimEvents® toolboxes of Matlab, and the Matlab environment allows for studying behavior and functional properties of the system. The modeling language incorporates several augmentations to aid the design process of Cyber-Physical Systems, and also provides a method to define functional constraints of the operation, using a calculi language based on the Hoare triples.

### Other tools

There are plenty of tools and modeling languages on the market that allow for creating cross-domain system models, and provide simulation methods for them. These modeling tools include Modelica [31], 20-sim [32], Ptolemy [33]. All of these tools have different purposes, but an abstract visual modeling environment that provides hybrid simulation does not exist yet.

While the Modelica library allows for simulation of models built with Modelica, it only provides a textual language for defining the models. 20-sim incorporates the bond graph modeling language, but 20-sim models are not hybrid, they cannot capture discrete behavior evolution. Ptolemy implements the hybrid automaton model of computation, and defines a simulation method to simulate heterogeneous models; however, it is not suitable to describe complex systems, since the required number of states of the Ptolemy model is exponential in the number of controlled junctions in the corresponding hybrid bond graph.

### Future work

The most important direction where this work should be continued is the constraint language, incorporated in the Grafset formalism (see Appendix D). Currently, the ConstraintInterpreter is able to extract the logical constraints expressed using the SAG calculi, and present them in the

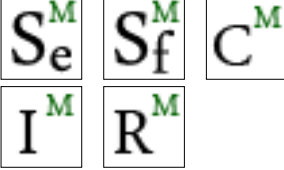

form of Hoare triples. A constraint checker module should be developed, to verify high-level safety properties in the system.



Another direction in which the development could continue is model-based diagnosis. Hybrid bond graph nodes could be extended with fault insertion points, to simulate fault-tolerant systems, where the fault-detection and fault-isolation logic is modeled in the computation domain.

## APPENDIX



### A. LIST OF MODELING ELEMENTS IN THE BDM FRAMEWORK

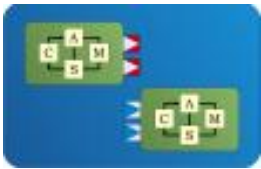
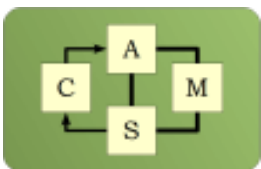
<i>Hybrid bond graph elements</i>	
<p><b>Zero junction, One junction</b></p> <p>These are the two types of bond graph junctions. ZeroJunctions are also called common effort-junctions while OneJunctions are called common-flow junctions. They can connect all the elements in the bond-graph. Junctions all have an OnCondition and an OffCondition attribute where their switching function can be defined – if they are meant to be controlled junctions. The conditions are defined with the standard Matlab syntax, and Switching inputs of the junction can be used as variables. The modeler can also specify the initial state of the junction.</p>	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 5px; width: 30px; text-align: center; font-weight: bold;">0</div> <div style="border: 1px solid black; padding: 5px; width: 30px; text-align: center; font-weight: bold;">1</div> </div>
<p><b>Sources (effort, flow)</b></p> <p>Power sources in the bond-graph. “Se” is a source of effort, while “Sf” represents the source of flow.</p> <p><i>The “Value” parameter must be filled out!</i></p>	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 5px; width: 30px; text-align: center; font-weight: bold;">S<sub>e</sub></div> <div style="border: 1px solid black; padding: 5px; width: 30px; text-align: center; font-weight: bold;">S<sub>f</sub></div> </div>
<p><b>Storage elements (capacitor, inductance)</b></p> <p>These elements represent the relationship between effort and displacement or flow and momentum in the bond graph respectively. They store the energy variables, which are the state-variables of the system.</p> <p><i>The “Value” parameter must be filled out!</i></p>	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 5px; width: 30px; text-align: center; font-weight: bold;">C</div> <div style="border: 1px solid black; padding: 5px; width: 30px; text-align: center; font-weight: bold;">I</div> </div>
<p><b>Resistor</b></p> <p>Dissipative element in the bond graph.</p> <p><i>The “Value” parameter must be filled out!</i></p>	<div style="border: 1px solid black; padding: 5px; width: 30px; text-align: center; font-weight: bold;">R</div>
<p><b>2-ports (transformer, gyrator)</b></p> <p>These elements can connect two junctions, establishing relationship between effort and effort or effort and flow on the two sides respectively.</p> <p><i>The “Value” parameter must be filled out!</i></p>	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 5px; width: 30px; text-align: center; font-weight: bold;">TF</div> <div style="border: 1px solid black; padding: 5px; width: 30px; text-align: center; font-weight: bold;">GY</div> </div>




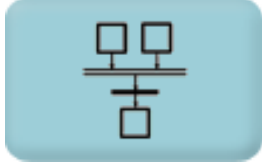

<p><b>Modulated sources and one-ports</b></p> <p>Modulated version of the source of effort, source of flow, capacitor, inertia and resistor elements. This means that their parameter field is not taken into account, but they must have an input signal to determine their parameter value. This signal connection may come from an input signal or an autonomous Modulation function element.</p>	
<p><b>Modulated two-ports</b></p> <p>Modulated version of the transformer and gyrator. They have the same features as the modulated one-ports above, but these elements can also have an input signal-connection from the Actuation modeling element. In this case, that signal will modulate the input power of the modeled actuator by connecting a source element on one side of the 2-port and the realization of the actuation function on the other side.</p>	


<i>HBG-related elements</i>	
<p><b>PowerPorts</b> (<i>Displacement, Electrical, Hydraulic, Rotation, Thermal</i>)</p> <p>These are domain-specific ports to establish bond-connections between junctions at different locations in the component-hierarchy. They can be connected to junctions or to another PowerPort of the same type. Because they are ports, they are visible from their container's parent element.</p>	
<p><b>Autonomous switching function</b></p> <p>The purpose of this element is to define the switching condition for controlled junctions. The Switching element can be connected to junctions to control them or to Grafset Transitions to directly define their enabling condition. The Switching accepts input connections from other junctions, local constant values, signals and sensing elements. In the "Expression" attribute a boolean function can be defined using the standard Matlab syntax; and its inputs can be referred to with their names.</p>	

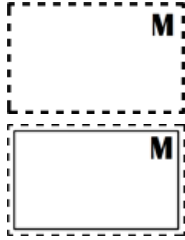


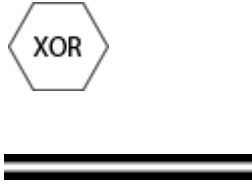


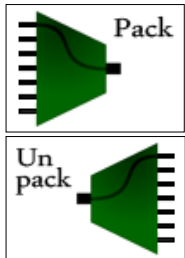
<p><b>Autonomous modulation function</b></p> <p>The Modulation element defines a function to control the actual value of Modulated bond-graph nodes. This element – like the Switching function above – can accept connections from other junctions, local constant values, signals and sensing elements. Its “Expression” attribute defines a real-value function, and the connected modulated one- and two-ports use that value instead of their parameters.</p>	
<p><b>Reset function</b></p> <p>The Reset function is a modeling element that implements the standard Reset function of the hybrid automaton on the energy variables of the connected storage whenever a discrete transition occurs. The reset is triggered by a boolean condition defined in the Reset block, (much like the Switching element), and the assigned value can also be defined as a function of any arbitrary input signal of the Reset block.</p>	


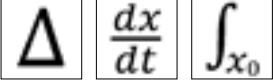



<i>Containers</i>	
<p><b>BDMsheet</b></p> <p>This is the root level container in the BDM paradigm. The RootFolder object can only contain these elements. These components will be mapped onto distinct Simulink models.</p>	<p><i>no visual representation</i></p>
<p><b>ParnasSystemSet</b></p> <p>The purpose of this element is to group ParnasSystems and connect them via signals.</p>	
<p><b>ParnasSystem</b></p> <p>Parnas system is the most important component of the BDM language: it models subsystems with autonomous computation, actuation, mechanics and sensing elements, implementing the Four Variable Model.</p>	



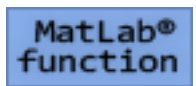

<p><b>Component</b></p> <p>This is a universal container, which is able to contain bond graphs and signal-related elements as well. A Component is recursively hierarchical so it can also contain other Components. It handles PowerPorts (for bonds) and SignalPorts as well.</p>	
<p><b>InfoLink</b></p> <p>This is a ParnasSystem-level element; it models the Signal-flow between other container types. It contains SignalPorts and signal paths only.</p>	
<p><b>PowerLink</b></p> <p>This is a ParnasSystem-level element as well. The PowerLink contains physical models (hybrid bond graphs) only, so the interface consists of the five types of PowerPorts.</p>	
<p><b>Computation</b></p> <p>This is the Grafcet container of the BDM Sketcher. This can be placed in any ParnasSystem, and its interface can be defined with SignalPorts. A hierarchical Grafcet model can be built inside.</p>	
<p><b>Controller</b></p> <p>This is the signal processing element of the framework. It can be used on the ParnasSystem-level or in Components. The Controller is also recursively hierarchic. Mathematical operators, ControlFunctions and these types are the children elements, and the interface is realized with SignalPorts.</p>	

<i>Grafcet elements</i>	
<p><b>Step and InitialStep</b></p> <p>These are the most basic elements of the Grafcet notation; they represent states (steps, phases) of the logical system. The component framed with double line is the initial step in each container.</p>	

<p><b>MacroStep and InitialMacroStep</b></p> <p>Macro-steps have the same purpose as their atomic version, but are modeled with “Model” elements in GME, so they can contain children elements. They realize the hierarchic nature of the Grafcet model.</p>	
<p><b>Transition</b></p> <p>The transition guards the token exchange along the arcs between steps. The actual firing can be triggered by input-signals; or they can generate output-signals when a token passes through a transition. They can be placed directly between two steps, or they can have Junctions (see below) both on their input and output side to establish relationship between multiple paths. The well-formedness rule in Grafcet is that between every two step, there has to be at least one Transition element.</p>	
<p><b>ExitMacro</b></p> <p>This element is used to denote when a token path ends in a MacroStep (or InitialMacroStep). Here, the token goes one level up.</p>	
<p><b>Junctions</b></p> <p>Junctions are defining logical relationship between their input or output arcs, coming either directly from a Step or from a Transition. In the model we have only implemented the XOR and “AND” junctions – according to the presentation. Introducing additional junction-types and the exact definition of these two are topic of further discussion.</p>	

<i>Signal-related elements</i>	
<p><b>Pack and Unpack element</b></p> <p>These two elements are created to group and ungroup signals. They are containers, so SignalPorts can be created inside them (input-signalports in the Pack and output-signalports in the UnPack), and then they can be connected through a signalport (ISignal, OSignal or LocalSignal). They map to multiplexers and demultiplexers in Simulink.</p>	

<p><b>Constant element</b></p> <p>This is element defines a value, and can be connected to Switching or Modulation elements (or even SignalPorts). This way, constants can be reused in multiple functions without duplicating the data in several places.</p>	
<p><b>Mathematical operations</b> (<i>delay, differentiate, integrate</i>)</p> <p>These elements represent mathematical operations that can be performed on signals. They can be used in Controllers and on the ParnasSystem-level.</p>	
<p><b>Signal ports (ISignal, OSignal, LocalSignal)</b> (<i>input-port, output-port, localport</i>)</p> <p>Input and output signalports are defining the interface of containers. These ports are visible from the parent element of the container (just as PowerPorts), and they accept input connections from-, and route output connections to one level higher than their origin.</p> <p>The LocalSignal is an abstract element, to connect elements, which are not connectable directly. For example we can use the LocalSignal to connect the Pack to an UnPack, to connect Sensing elements to ControlFunctions and so on.</p>	
<p><b>Sensing elements (De, Df, Dq, Dp)</b></p> <p>Sensing elements are abstract modeling elements that allow us to detect the gateway between the power and signal domain. We use this element to connect junctions and storages onto the signal flow with their actual <math>e</math>, <math>f</math>, <math>q</math> or <math>p</math> value. We can connect the De (detect effort) Sensing element to ZeroJunctions and the Df (detect flow) element to OneJunctions while Dq can be connected to Capacitors and the Dp can be connected to Inertias.</p>	
<p><b>Actuation</b></p> <p>This is the opposite of the Sensing elements; the Actuation element models the Signal-to-Power domain transformation. We can only connect this to Modulated Two-Port bond graph elements (MTF and MGY), and it represents a power-input to the system</p>	

<i>Other elements</i>	
<p><b>Simulink system</b></p> <p>Using this element, a Simulink model can be integrated into the BDM model, by specifying the .mdl file. The file should be in the same folder as the GME project-file, and the same number of input- and output ports should be added in GME as it is in the .mdl file.</p>	
<p><b>Parameter</b></p> <p>The Parameter element is a block-level parameter with its name and value attribute in Simulink. It can be used to specify block parameters for bond graph elements.</p>	
<p><b>ControlFunction</b></p> <p>The ControlFunction element is mapped onto an Embedded Matlab Function in Simulink. Its source can be specified in the “Code” attribute. The standard Matlab syntax can be used. The syntax for the function header is the following:</p> <pre>function [OSignalName1, OSignalName2] =     arbitraryName(ISignal1, ISignal2)</pre>	
<p><b>Monitor effort and flow</b></p> <p>These elements can only be connected to the appropriate junctions (e to ZJ, f to OJ), and they will be mapped to Simulink Scopes to monitor the actual value on the junction.</p>	

## B. PARAMETERS OF THE RO SYSTEM

### *Parameters of the physical system*

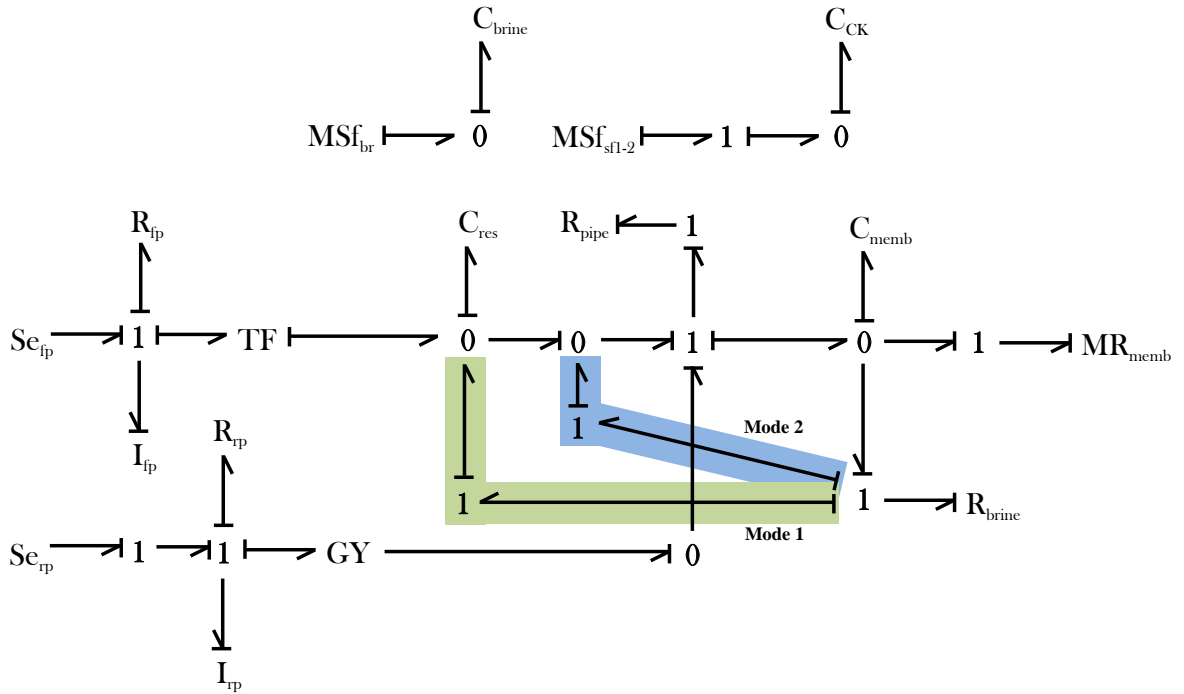
Parameter	Name	Value	Unit
Feed pump – Rotational mass	$I_{fp}$	$2.482e^{12}$	kg m <sup>2</sup>
Feed pump – Energy dissipation	$R_{fp}$	$3.2350e^{10}$	N m s
Feed pump – Rotor torque	$Se_{fp}$	11031	Nm
Recirculation pump – Rotational mass	$I_{rp}$	9600	kg m <sup>2</sup>
Recirculation pump – Energy dissipation	$R_{rp}$	2	N m s
Recirculation pump – Rotor torque	$Se_{rp}$	$2.0684e^7$	N m
Capacitance of the tubular reservoir	$C_{res}$	$4.351e^{-8}$	m <sup>5</sup> /N
Hydraulic resistance to memb. (M1, M2)	$R_{pipe}$	$2.8544e^{13}$	N s/m <sup>5</sup>
Hydraulic resistance to memb. (Purge)	$R_{pipe\_p}$	$2.8300e^{13}$	N s/m <sup>5</sup>
Capacitance of the membrane module	$C_{memb}$	$7.611e^{-10}$	m <sup>5</sup> /N
Hydraulic resistance of the back-flow pipe	$R_{brine}$	$9.1014e^{12}$	N s/m <sup>5</sup>
Hydraulic resistance of the drain pipe	$R_{drain}$	$6.3649e^9$	N s/m <sup>5</sup>

### *Modulation functions and parameters of the conductivity domain*

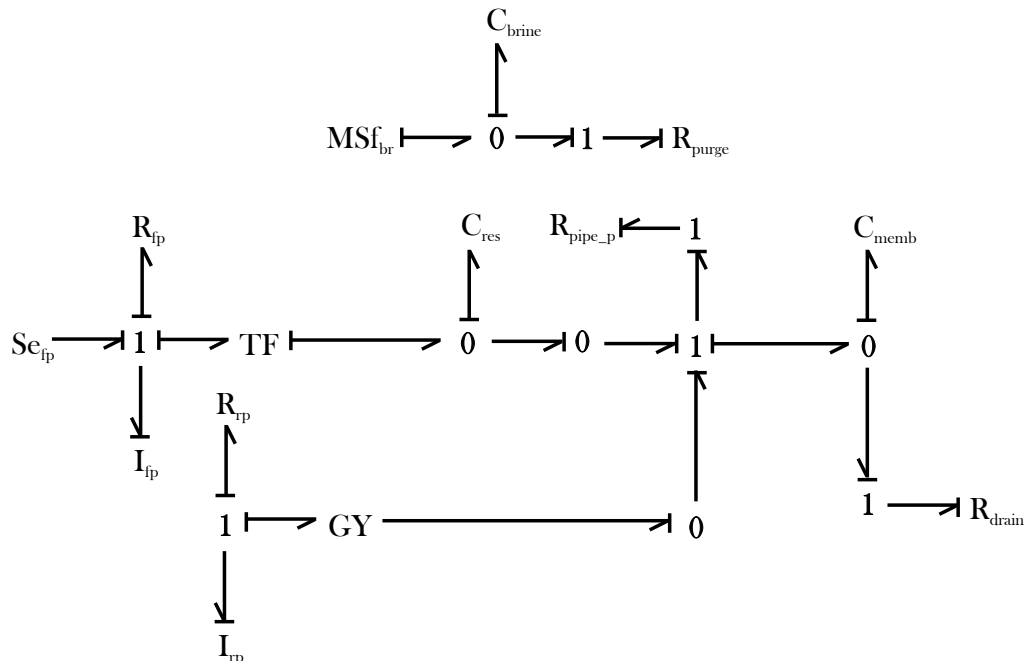
Brine flow source (F1)	$backflow\_rate/1.67e^{-8}$		
M1 flow source (F2)	$mmb\_inflow * ((backflow\_rate * 6) + 0.1)/1.67e^{-8}$		
M2 flow source (F3)	$mmb\_inflow * ((3.86 * backflow\_rate * 6) + 0.1)/1.67e^{-8}$		
Memb. resistance (F4)	$0.202 * (((K - 12000)/165 * 4.137e^{11}) + 29 * 4.137e^{11})$		
Brine capacitor	$C_{brine}$	$q_0 = 0$	$C = 2400$
Conductivity capacitor	$C_K$	$q_0 = 12000$	$C = 169500$
Resistance to purge the brine capacitor	$R_{purge}$	0.001	

### C. CAUSAL BOND GRAPH OF THE RO SYSTEM IN THE THREE MODES

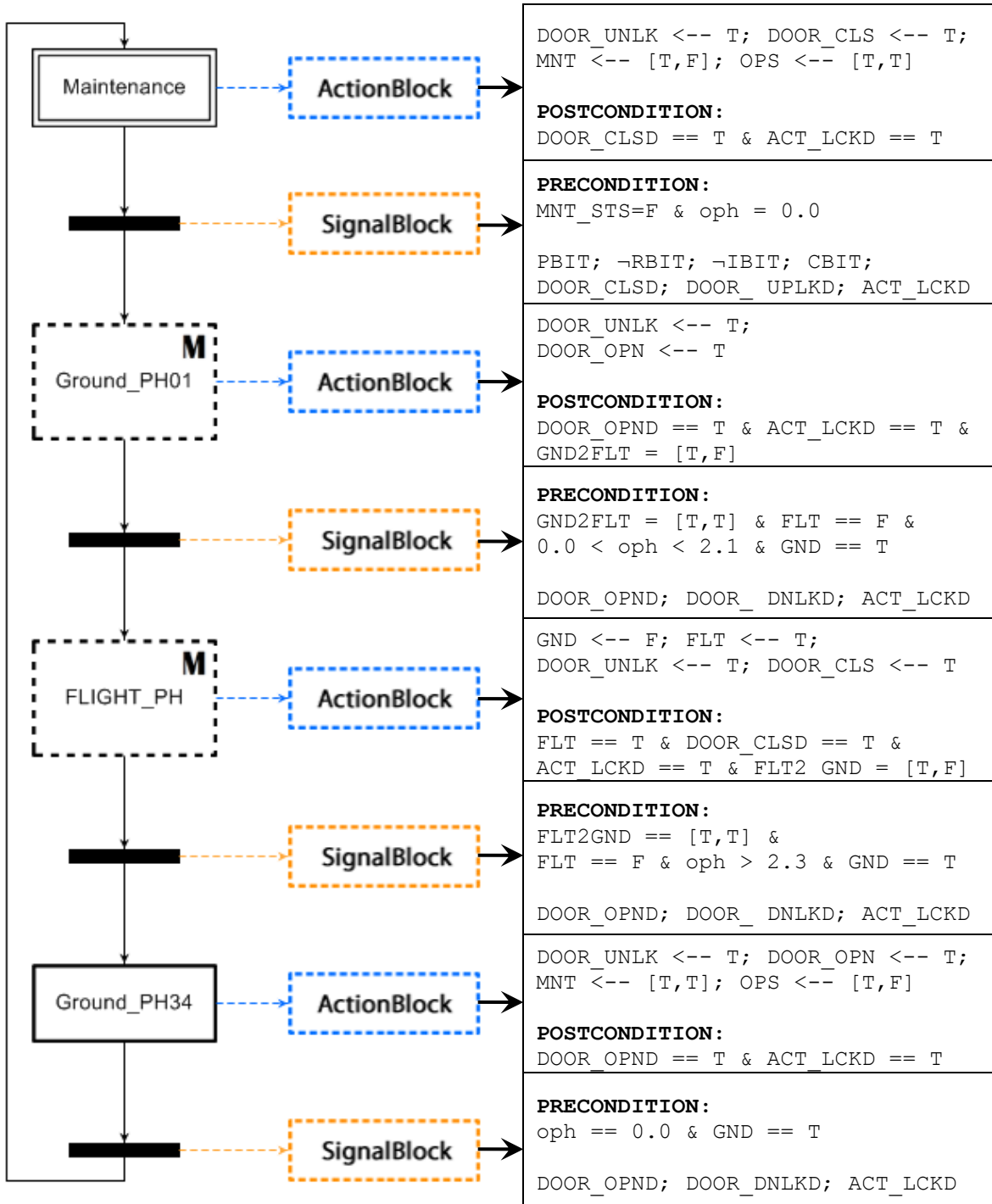
#### Mode 1 and Mode 2



#### Purge Mode



## D. DOOR SYSTEM SAG CALCULI EXAMPLE





## REFERENCES

- [1] Edward A. Lee, "Cyber Physical Systems: Design Challenges," in *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363-369.
- [2] Akos Ledeczki et al., "The Generic Modeling Environment," *Workshop on Intelligent Signal Processing*, May 2001.
- [3] Matlab®. [Online]. <http://www.mathworks.com/products/matlab/>
- [4] Janos Sztipanovits et al., "Multigraph: An Architecture for Model-Integrated Computing," *Proceedings of the IEEE*, pp. 361-368, November 1995.
- [5] Y. Umeda, H. Takeda, T. Tomiyama, and H. Yoshikawa, "Function, Behavior and Structure," 1990.
- [6] David Renè, "Grafcet: A Powerful Tool for Specification of Logic Controllers," *IEEE Transactions on control systems technology*, vol. 3, no. 3, pp. 253-268, September 1995.
- [7] Simulink®. [Online]. <http://www.mathworks.com/products/simulink/>
- [8] SimEvents®. [Online]. <http://www.mathworks.com/products/simevents/>
- [9] C. A. R. Hoare, "An axiomatic basis for computer programming," *CACM*, pp. 576-580, 1969.
- [10] Arpad Bakay and Endre Magyari, "The UDM framework," 2004.
- [11] Jan Top and Hans Akkermans, "Computational and Physical Causality," *Proceedings of the IJCAI*, pp. 1171-1176, 1991.
- [12] D. Kortenkamp and S. Bell, "BioSim: An Integrated simulation of an advanced life support system for intelligent control research," in *Proc. of the 7th Symp. on Artificial Intelligence, Robotics and Automation in Space*, 2003.
- [13] K. D. Pickering et al., "Early results of an integrated water recovery system test," in *Proc 29th Int Conf Environmental Sys*, 2001.
- [14] Rajeev Alur et al., "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, pp. 3-34, 1995.
- [15] Gautam Biswas and Sriram Narasimhan, "An approach to model-based diagnosis of hybrid systems," *Hybrid Systems: Computation and Control*, 2002.
- [16] Dean C. Karnopp, Donald L. Margolis, and Ronald C. Rosenberg, *System Dynamics: Modeling and Simulation of Mechatronic Systems.*: John Wiley & Sons, Inc., 2005.
- [17] Jan C. Willems, "The Behavioral Approach to Open and Interconnected Systems," *IEEE Control Systems Magazine*, pp. 46-99, December 2007.
- [18] P. J. Mosterman and Gautam Biswas, "A theory of discontinuities in physical system models," *J Franklin Institute*, vol. 335B, pp. 401-439, 1998.

- [19] Panos J. Antsaklis and Xenofon D. Koutsoukos, "Hybrid Systems: Review and Recent Progress," *Software-Enabled Control*, pp. 272-298, 2003.
- [20] Raul G. Longoria, Modeling of Physical Systems: Lecture summaries, 2006, <http://www.me.utexas.edu/~longoria/>.
- [21] Marcelin Fortes da Cruz and Sanjiv Sharma, Grafcet and SAG calculi, Proprietary documents of Airbus UK Ltd.
- [22] Janos Sztipanovits and Gabor Karsai, "Model-Integrated Computing," *IEEE Computer*, vol. 30, pp. 110-112, April 1997.
- [23] Object Management Group. Unified Modeling Language (UML) ®. [Online]. <http://www.uml.org/>
- [24] Object Management Group. Object Constraint Language (OCL) ®. [Online]. [http://www.omg.org/technology/documents/modeling\\_spec\\_catalog.htm#OCL](http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL)
- [25] Indranil Roychoudhury, Matthew J. Daigle, Gautam Biswas, and Xenofon Koutsoukos, "Efficient Simulation of hybrid systems: A hybrid bond graph approach," *Simulation: Transactions of the Society for Modeling and Simulation International*, pp. 467-498, 2010.
- [26] D. L. Parnas and J. Madey, "Functional documentation of computer systems," *Science of Computer Programming*, pp. 41-61, October 1995.
- [27] Constance Heitmeyer, R. D. Jeffords, and B. G. Labaw, "Automated Consistency Checking of Requirements Specifications," *ACM Transactions on Software Engineering Methodology*, vol. 5, no. 3, pp. 231-261, July 1996.
- [28] Simulink® API. [Online]. <http://www.mathworks.com/help/toolbox/simulink/slref/bq3cxmi.html>
- [29] Level-2 S-functions in Matlab®. [Online]. <http://www.mathworks.com/help/toolbox/simulink/slref/level2matlabsfunction.html>
- [30] Gautam Biswas, Eric-J. Manders, John Ramirez, Nagabhusan Mahadevan, and Sherif Abdelwahed, "Online Model-based Diagnosis to support autonomous operation of an Advanced Life Support System," *Habitat.: Int. J. Human Support Res.*, pp. 21-38, 2004.
- [31] Modelica®. [Online]. <https://www.modelica.org/>
- [32] 20-sim®. [Online]. <http://www.20sim.com>
- [33] UC Berkeley EECS Department. Ptolemy Project. [Online]. <http://ptolemy.eecs.berkeley.edu/>
- [34] Carl-Johan Sjöstedt, Doctoral Thesis: Modeling and Simulation of Physical Systems in a Mechatronical Context, 2009.