

# Multi-Domain Physical System Modeling and Control Based on Meta-Modeling and Graph Rewriting

Sagar Sen and Hans Vangheluwe

**Abstract**—A methodology is presented which enables the specification and synthesis of software tools to aid in plant and controller modeling for multi-domain (electrical, mechanical, hydraulic, and thermal) physical systems. The methodology is based on meta-modeling and graph rewriting. The plant is modeled in a domain-specific formalism called the Real World Visual Model (RWVM). Such a model is successively transformed to an Idealized Physical Model (IPM), to an Acausal Bond Graph (ABG), and finally to a Causal Bond Graph (CBG). A Modelica ([www.modelica.org](http://www.modelica.org)) model, consisting of a Causal (algebraic and differential equation) Block Diagram (CBD), is generated from the CBG. All transformations are explicitly modeled using Graph Grammars. A PID controller model, specified in Modelica as a CBD is subsequently integrated with the plant model. AToM<sup>3</sup> ([atom3.cs.mcgill.ca](http://atom3.cs.mcgill.ca)), A Tool for Multi-formalism and Meta Modeling is used to meta-model and synthesize visual modeling environments for the RWVM, IPM, ABG, and CBG formalisms as well as for transformations between them. The entire process of modeling, transformation, and simulation is demonstrated by means of a hoisting device example. Our methodology drastically reduces development time (of the modeling tool an indirectly of the domain-specific models), integrates model checking via Bond Graph causal analysis, and facilitates management and reuse of meta-knowledge by explicitly modeling formalisms and transformations.

## I. INTRODUCTION

Modeling of a multi-domain (electrical, mechanical, hydraulic, and thermal) lumped-parameter physical system and its controller is becoming challenging with the increasing complexity of such systems. Modeling techniques have come a long way from the traditional approach of writing down Ordinary Differential Equations (ODE) and Differential Algebraic Equations (DAE) to the construction of visual models that are modular, hierarchical, and more recently domain-specific, possibly encompassing multiple formalisms. Widely used modeling tools such as Dymola [7] (based on Modelica [6]) and MATLAB Simulink/SimMechanics [4], [5] are also based on visual languages. In these

Sagar Sen is with School of Computer Science McGill University, 3480 University Street, Montreal, Canada H3A2A7 [sagar.sen@cs.mcgill.ca](mailto:sagar.sen@cs.mcgill.ca)

Hans Vangheluwe is with School of Computer Science McGill University, 3480 University Street, Montreal, Canada H3A2A7 [hv@cs.mcgill.ca](mailto:hv@cs.mcgill.ca)

tools the domain-specific knowledge is hard-coded in the formalism and is not represented explicitly in the form of a model (meta-model). The development effort required to craft such a tool far exceeds that of our approach where the software for the modeling environment is automatically synthesized from the meta-model. In Modelica [6], the model is specified in an object-oriented language but it directly translates to differential-algebraic equations. In our approach, Graph Grammar [1] rules facilitate rapid model design and transformation as described in Section III. Model transformation is normally coded in a textual programming language in popular modeling tools even when the model itself is graphical. Our approach is implemented in the tool AToM<sup>3</sup> [8].

## II. META-MODELING

We specify the abstract syntax of modeling formalisms using Unified Modeling Language (UML) class diagram (CD) models called meta-models (model of a class of models). UML CD are in turn meta-modeled using UML CD itself or Entity-Relationship Diagrams (ERD). This is shown in Fig. 1. The expressive power of meta-modeling formalisms UML CD and ERD is enhanced through constraints specified in the Object Constraint Language (OCL) or Python (used in AToM<sup>3</sup>). Constraints are set on attribute values and relationship cardinalities depending on the formalism.

We specify, using UML CD, the meta-models for three different visual formalisms. We also use an object-oriented textual formalism, (Modelica), to represent models. The language itself is meta-modeled using Backus-Naur style grammar. Finally, the simulation result models are expressed in a Trajectory formalism. The meta-models for visual formalisms RWVM, IPM, and BG (contains ABG and CBG) are given in Fig. 2 (a), (b), and (c) respectively. We synthesize a dedicated visual modeling environment for each of these formalism to express models of our running example, the hoisting device. A hoisting device, as described by Broenink [2], is a multi-domain engineered physical system. It comprises four components: electrical mains, electromotor, cabledrum and a

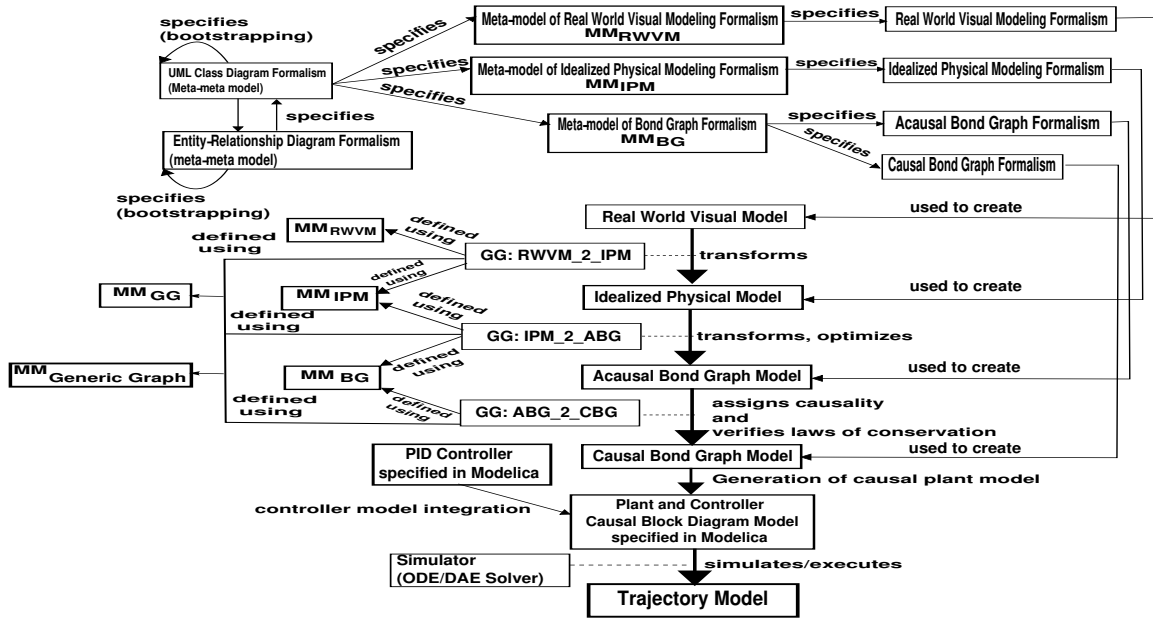


Fig. 1. Methodology Overview

load arranged in the configuration shown in Fig. 2 (a). The electrical mains represent the electrical domain of the hoisting device. The electromotor, which represents the rotational mechanical domain, converts electrical to rotational mechanical energy. The cabledrum transforms the rotational mechanical energy to linear mechanical energy to lift the load along a rope. Therefore, the hoisting device example combines three engineering domains: electrical, linear mechanical, and rotational mechanical. The roles of textual formalism ,Modelica, and the Trajectory formalism (meta-modeled using UML CD) to encode model behaviour are shown in Fig. 2 (d) and (e) respectively.

### III. GRAPH TRANSFORMATIONS

The *transformation* of models is a crucial element in all model-based endeavours. As models, meta-models, and meta-meta-models are all in essence attributed, typed graphs, we can transform them by means of graph rewriting. The rewriting is specified in the form of Graph Grammar [1] models. These are a generalization, for graphs, of Chomsky grammars. Graph Grammars are composed of an ordered collection of rules. Each rule consists of Left Hand Side (LHS) and Right Hand Side (RHS) graphs. Rules are evaluated against an input graph, called the host graph. If a matching is found between the LHS of a rule and a sub-graph of the host graph, then the rule can be applied. When a rule is applied, the matching subgraph of the host graph is replaced by the RHS of the rule. Rules can have applicability conditions, as well as actions to be performed when the rule is

applied. Some graph rewriting systems have control mechanisms to determine the order in which rules are checked. We use our tool AToM<sup>3</sup> to specify graph grammar (GG) rules. Graph Grammars can transform models between formalisms (specifying denotational semantics), structurally optimize models, and also can be used to specify operational semantics *i.e.*, simulations. Formalism transformation rules require a source formalism meta-model, a target formalism meta-model and the Generic Graph formalism. The Generic Graph edges and vertices can connect between objects of any formalism allowing a rule to associate related entities of different formalisms. Objects from the source formalism can be preserved via these generic links to uniquely identify their position at any stage. These source objects, relationships and generic links can be removed using other rules leading to a model purely in the target formalism. Rule execution is constrained by pre-conditions. Post actions are executed after the execution of a rule. Rules are also used to copy/specify attributes to a model in the target formalism. An  $\langle ANY \rangle$  tag for an object X, for instance, in the LHS indicates that the GG rule will execute for any value of the attribute of the object X. The  $\langle COPIED \rangle$  or  $\langle SPECIFIED \rangle$  tags in the RHS of a rule indicate that some attribute from the LHS object is copied or specified respectively to an object in RHS. Every object in the LHS and RHS of a rule is identified by a unique label (a positive integer is annotating the object). We present three sets of graph grammar rules (referred to in Fig. 1): RWVM to IPM (RWVM\_2\_IPM), IPM to ABG (IPM\_2\_ABG), and



TABLE I  
HOISTING DEVICE RELATED GRAPH GRAMMAR RULES FOR  
TRANSFORMATION RWVM<sub>2</sub>IPM

Order	Rule Name	Description
1	Mains <sub>2</sub> IPM	RWVM of electrical mains is transformed toIPMM of electrical mains
2	Motor <sub>2</sub> IPM	RWVM of a motor is transformed toIPMM components of the rotational mechanical domain
3	CableDrum <sub>2</sub> IPM	RWVM of the cable drum is transformed toIPMM of the cable drum
4	Load <sub>2</sub> IPM	RWVM load is transformed intoIPMM of load

ABG to CBG (ABG<sub>2</sub>CBG).

#### A. Graph Grammar Rules: RWVM to IPM

The Real World Visual Modeling formalism is domain-specific. We specify the RWVM for a hoisting device. Objects in the RWVM represent only high-level entities that comprise a hoisting device such as electrical mains, motor, and cable drum based on domain knowledge and required detail. Mapping a RWVM to an IPM can be done in one or more ways. There is no fixed set of rules and in general human input will be required. One such mapping is given by four graph grammar rules in Table I.

#### B. Graph Grammar Rules: IPM to ABG

There is a one-to-one transformation from an idealized physical model to an acausal bond graph. The set of GG rules with their execution order is given in Table II. Specifying the transformation a brief description of the steps is given below:

**Step 1 (Identifying Efforts):** The rules `identify_efforts_{E2E, RM2RM, RM2LM, E2RM, LM2LM}` insert a bond graph junction for pertinent IPM links. A 0-junction is associated with all `E2{E, RM}` relationships. The rules `identify_efforts_{RM2LM, LM2LM}` are for the mechanical domain and result in the insertion of a 1-junction. An example is given in Fig. 3 (a).

**Step 2 (Finding Effort/Flow Differences):** The rules `effort_differences_{R2E, C2E, I2E, AC2E}` insert a 0 to 1 junction as shown in Fig. 3 (b) are applicable to the components in the electrical domain. The rules `flow_differences_{R2RMFrmE, R2RM, I2RM}` are executed across mechanical domain components for which a 1 to 0 junction is inserted. A flow difference example is shown in Fig. 3 (d).

**Step 3 (Inserting Bond Graph Elements):** After creating the initial junction structure we insert Bond Graph elements at the appropriate

positions. The network formed by a generic link that connects an IPM element with its associated junction structure (created in Step 1 and 2) is unique. This property helps in the deterministic execution of these rules at the correct locations. The rules `insert_SE2E, R2E, I2E, GY2E, TF2RM2RM, R2RM, L2RM, SE2LM, I2LM1` are executed in this step. An example, `insert_SE2E`, for the electrical domain is given in Fig. 3 (c). Also, in Fig. 3 (h) the example `insert_R2RM` illustrates an insertion rule for the mechanical domain.

**Step 4 (Deleting IPM Elements and Generic Link):** Now that we have the initial Bond Graph structure we can remove the IPM elements and the generic links using some deletion rules. The LHS is the IPM element or connection or a generic link and the RHS of the rule is Empty. The rules `delete_E2E, E2RM, RM2RM, RM2LM, LM2LM, R2E, C2E, I2E, AC2E, R2RM, I2RM, AC2E, TF2RM2LM, Earth2E, generic_link, I2LM` are executed at this step. An example is given in Fig. 3 (d).

**Step 5 (Optimizing Bond Graph):** The Bond Graph obtained in Step 4 needs to be simplified due to extra junctions created in Steps 1 and 2 to facilitate the transformation. The rules `optimize_JJJ, SJJ, JJR, JJGY, JGES, JJTF, SRJGY, GYRJTF` are some of all possible optimizations (see [2] for all others). Two examples are given in Fig. 3 (e) and (f).

#### C. Graph Grammar Rules: ABG to CBG

Causality assignment for Bond Graphs is an algorithmic procedure [2]. We implement causality assignment as a set of graph grammar rules. The rules are given in Table III. The basic steps involved are as follows.

**Step 1 (Fixed Causality):** Fixed causality is assigned at the sources. An effort source (SE) has by definition its effort variable going out as a signal output. And hence it has an outward causal stroke. This causality is called effort-out causality or effort causality. Likewise, a flow source (SF) gives rise to a flow-out causality or flow causality. The rules `FC2SE2ZJ, SF2OJ` are executed in this step. An example is given in Fig. 4 (a).

#### Step 2 (Constrained Causality):

Causality assignment of certain bonds imposes a causal constraint on other bonds. For a transformer element one of the connected bonds has an effort out causality, while the other has a flow out causality. Similarly, in a gyrator element both the incoming bond and the outgoing bonds have the same causality *i.e.*, either effort-out or flow-out. An example of this type of causality constraint is given in Fig. 4 (c). The causal condition at a

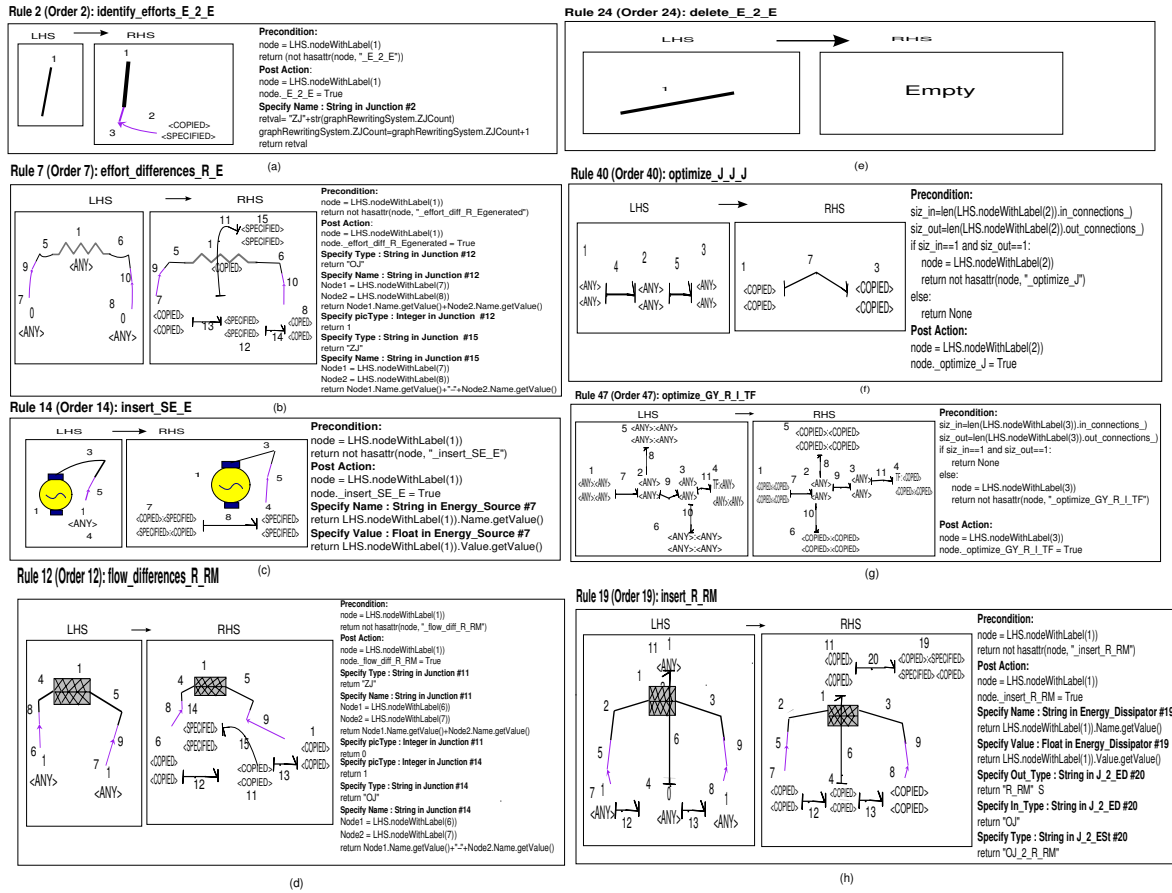


Fig. 3. Typical Graph Grammar Rules in IPM2ABG

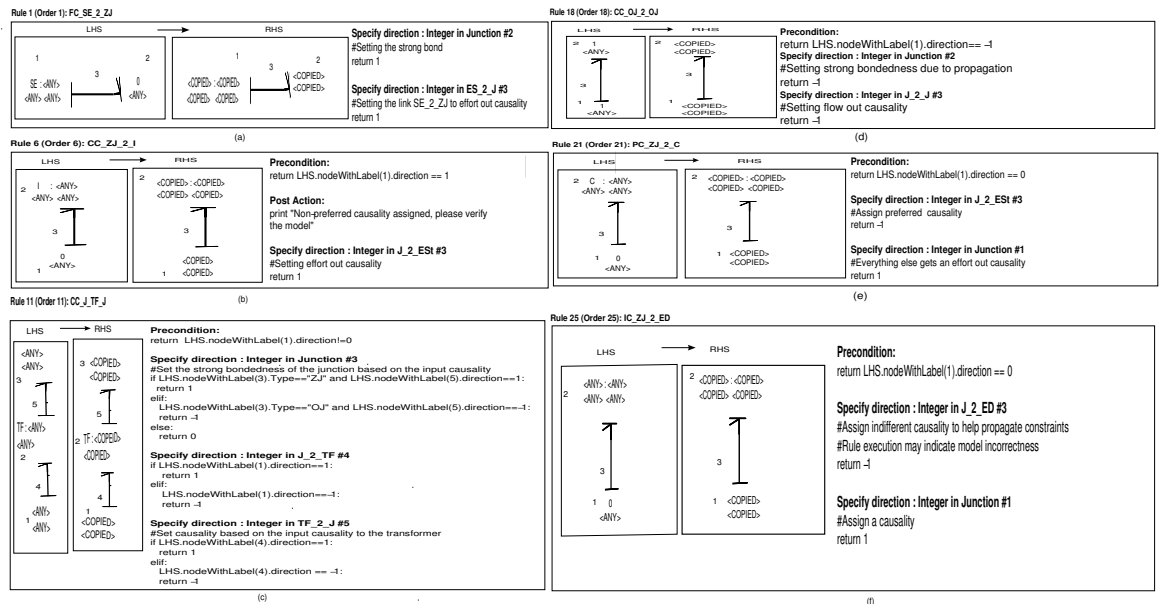


Fig. 4. Typical Graph Grammar Rules in ABG2\_CBG

TABLE II  
GRAPH GRAMMAR RULES FOR IPM\_2\_ABG

No.	Rule Name	No.	Rule Name
1	delete_earth	25	delete_E_2_RM
2	identify_efforts_E_2_E	26	delete_RM_2_RM
3	identify_efforts_RM_2_RM	27	delete_RM_2_LM
4	identify_efforts_RM_2_LM	28	delete_LM_2_LM
5	identify_efforts_E_2_RM	29	delete_R_E
6	identify_efforts_LM_2_LM	30	delete_C_E
7	effort_differences_R_E	31	delete_I_E
8	effort_differences_C_E	32	delete_AC_E
9	effort_differences_I_E	33	delete_R_RM
10	effort_differences_AC_E	34	delete_LRM
11	flow_differences_R_RM_FmE	35	delete_AC_E
12	flow_differences_R_RM	36	delete_TF_RM_2_LM
13	flow_differences_LRM	37	delete_Earth_E
14	insert_SE_E	38	delete_generic_link
15	insert_R_E	39	delete_LLM
16	insert_I_E	40	optimize_J_J_J
17	insert_GY_E	41	optimize_S_J_J
18	insert_TF_RM_2_RM	42	optimize_J_J_R
19	insert_R_RM	43	optimize_J_J_GY
20	insert_R_RM_1	44	optimize_J_J_ES
21	insert_LRM	45	optimize_J_J_TF
22	insert_SE_LM	46	optimize_S_R_I_GY
23	insert_LLM_1	47	optimize_GY_R_I_TF
24	delete_E_2_E		

1-junction is the dual form of the causal condition at the 0-junction. All flows are equal, thus exactly one bond will bring in the flow, implying that exactly one bond has the causal stroke away from the 1-junction (see Fig.4 (d) for an example rule). These conditions give rise to several other causality constraints that are specified using Graph Grammars (see the rule in Fig. 4(b)). The rules in Table III starting with the prefix CC\_ are executed, in the given order, to propagate causality due to constraints. If differential causality is assigned due to constraint propagation, an error is reported indicating a problem with the physical model.

**Step 3 (Preferred Causality):** At storage elements, the causality determines whether an integration or a differentiation occurs. Integration or differentiation is performed on the flow or the effort coming into the element. In the case of the capacitor the effort is integrated in physical reality. The flow is integrated in the case of an inductor. Therefore, integral causality is the preferred causality due to the physics of the phenomenon. This implies that a C-element has an effort-out causality and an I-element has a flow-out causality as its preference. The Graph Grammar to assign preferred causality to the bond between a 0-junction and an inductor is given in Fig. 4(e).

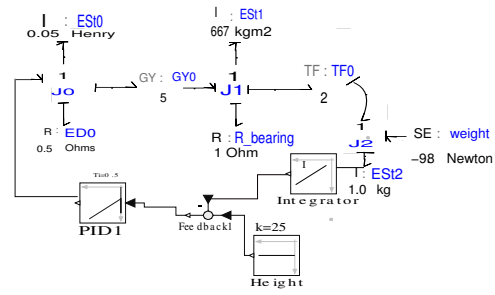


Fig. 5. Multi-formalism model of Plant (Bond Graph) and Controller (Causal Block Diagram)

**Step 4 (Indifferent Causality):** Indifferent causality is assigned, when there exist no causality constraints. For the linear resistor it does not matter which of the port variables is the output. There is no difference in causality if the current(flow) is the incoming variable and the voltage (effort) is the outgoing variable and vice versa. An example rule is presented in Fig. 4(f). The rules IC\_ZJ\_2\_R, OJ\_2\_R are finally executed in this step.

#### D. Simulatable Model Generation for Plant and Controller

The plant system is now in the CBG formalism. We generate Modelica models using François Cellier's Bond Graph library [3]. The Bond Graph library is based on the causal block construct in the standard Modelica language. A PID controller, from the Modelica standard library, which is also made of Causal Blocks, is connected to the plant system at this level of abstraction. For our hoisting device example we use the controller to lift the load to a desired height by controlling input voltage. In Fig. 5 we present a schematic that shows how the CBG is controlled by the PID controller.

## IV. SIMULATION RESULTS

The hoisting device plant model is first simulated without a controller and then a controller is attached that has the task of bringing the load to a specific height. Realistic parameter values are given to the components of the Bond Graph model of the hoisting device. The electrical domain has resistance  $R_{el} = 0.5\Omega$ , inductance  $L = 0.05H$ , and the electromotor gyration has a ratio of  $m = 5$ . In the rotational mechanical domain bearing resistance is  $R_{bearing} = 1\Omega$ , and its rotational inertia  $J = 667Nm/rad$ . The cable drum transformer is given the ratio  $n = 2$ . In the translational mechanical domain the mass of the load is  $m = 1kg$  and the effort/force source due to gravity is  $weight = -9.8N$ .

A pulsed voltage (0-110V) signal is applied as input as shown in the upper plot of Fig. 2 (e). The

TABLE III  
GRAPH GRAMMAR RULES FOR IPM<sub>2</sub>\_ABG

No.	Rule Name	Description	No.	Rule Name	Description
<b>Fixed Causalities</b>					
1	FC_SE_2_ZJ	Effort source to 0-junction	13	CC_OJ_2_C	1-junction to Capacitor
2	FC_SF_2_OJ	Flow source to 1-junction	14	CC_OJ_2_I	1-junction to Inductor
<b>Constrained Causalities</b>					
3	CC_ZJ_2_R	0-junction to Resistor	15	CC_OJ_2_ZJ	1-junction to 0-junction
4	CC_ZJ_2_C	0-junction to Capacitor	16	CC_OJ_2_OJ	1- junction to 1-junction
5	CC_ZJ_2_I	0-junction to inductor	17	CC_OJ_2_TF	1-junction to Transformer
6	CC_ZJ_2_ZJ	0-junction to 0-junction	18	CC_OJ_2_GY	1-junction to Gyrator
7	CC_ZJ_2_OJ	0-junction to 1-junction	<b>Preferred Causalities</b>		
8	CC_ZJ_2_TF	0-junction to Transformer	19	PC_ZJ_2_C	0-junction to Capacitor
9	CC_ZJ_2_GY	0-junction to Gyrator	20	PC_OJ_2_C	1-junction to Capacitor
10	CC_J_TF_J	0/1-junction to Transformer to 0/1-junction	21	PC_ZJ_2_I	0-junction to Inductor
11	CC_J_GY_J	0/1-junction to Gyrator to 0/1-junction	22	PC_OJ_2_I	1-junction to Inductor
<b>Indifferent Causalities</b>					
12	CC_OJ_2_R	1-junction to Resistor	23	IC_ZJ_2_R	0-junction to Resistor
			24	IC_OJ_2_R	1-junction to Resistor

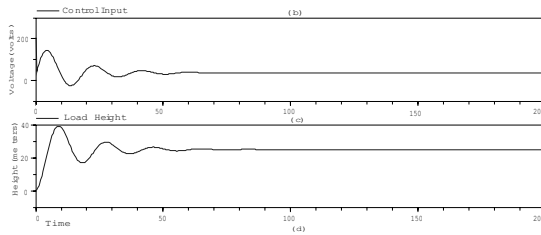


Fig. 6. Simulation Plots

height of the mass with respect to time is given in the lower plot of Fig. 2 (e). We then introduce the PID controller. The controller voltage input is shown in Fig. 6 (c). The task of the controller is to lift the load to a height of 25 m. The height of the load with respect to time is shown in Fig. 6 (d).

## V. CONCLUSIONS AND FUTURE WORK

We have implemented a framework to model multi-domain physical systems and their controllers. The system as such was modelled at a domain-specific level in the RWVM formalism. A PID controller was connected at the level of Causal Blocks since it is a suitable representation formalism. The RWVM of our running example hoisting device is automatically transformed using graph rewriting to a Modelica model that is simulated using a DAE solver. Meta-modeling allows us to synthesize domain-specific visual modelling environments. This numerically constrains the modeler to create only valid models therefore development errors. Furthermore, the explicit encoding of modeling knowledge as graph grammar rules (or model transformations in general) allows the synthesis of model transformations. This facilitates the creation of complex domain models without re-creating an-

alytical models (such as IPM, BGs, and Modelica code) normally constructed by hand.

In ongoing work [9] we extend our approach by exploring the design space of domain-specific models in pursuit of satisfying product requirements represented as fitness criteria. This extension will improve the process of synthesizing physically meaningful models directly from the requirements.

## VI. ACKNOWLEDGMENTS

We thank Denis Dube and Ximeng Sun of the MSDL, McGill University for their help with debugging and tool support.

## REFERENCES

- [1] Ehrig, H. and Engels, G. and Kreowski, H.-J. and Rozenberg, G., *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1-3*, World Scientific, 1999
- [2] Broenink J.F., *Introduction to Physical Systems Modelling with Bond Graphs*, <http://www.ce.utwente.nl/bnk/papers/BondGraphsV2.pdf>, 1999.
- [3] Cellier, F.,A.N., *The Modelica Bond Graph Library*, Proc. of the 4th International Modelica Conference, 2005.
- [4] SIMULINK. *Dynamic System Simulation for Matlab*. The MathWorks, Natick, MA, January 1997.
- [5] GD Wood, *Simulating Mechanical Systems in Simulink with SimMechanics*, DC Kennedy - The Mathworks report, 2003
- [6] Fritzson, P. and Engelson, V. *Modelica - A Unified Object-Oriented Language for System Modeling and Simulation*. In Proceedings of the 12th European Conference on Object-Oriented Programming (ECOOP'98 , Brussels, Belgium, Jul. 20-24), 1998.
- [7] Elmqvist, H., *Dymola - User's Manual*, Dynasim AB, Research Park Ideon, Lund, Sweden, 1994.
- [8] Juan de Lara, Hans Vangheluwe, *AToM<sup>3</sup>: A Tool for Multi-formalism and Meta-modelling*, Lecture Notes In Computer Science; Vol. 2306 pp. 174-188
- [9] Sagar Sen, *Model Driven Design Space Exploration of Lumped-Parameter Physical Systems*, MSc. Thesis, School of Computer Science, McGill University, Canada. In Progress.