

Component-based Modeling of Dynamic Systems using Heterogeneous Composition

Zsolt Lattmann
Vanderbilt University
Nashville, TN, USA
lattmann@isis.vanderbilt.edu

Adam Nagel
Vanderbilt University
Nashville, TN, USA
adam@isis.vanderbilt.edu

Tihamer Levendovszky
Vanderbilt University
Nashville, TN, USA
tihamer@isis.vanderbilt.edu

Ted Bapty
Vanderbilt University
Nashville, TN, USA
bapty@isis.vanderbilt.edu

Sandeep Neema
Vanderbilt University
Nashville, TN, USA
sandeep@isis.vanderbilt.edu

Gabor Karsai
Vanderbilt University
Nashville, TN, USA
gabor@isis.vanderbilt.edu

ABSTRACT

Cyber-Physical Systems (CPS) are composed of computational and physical components, which includes various types of physical phenomena such as electrical and mechanical domains. Many modeling paradigms exist to model the static properties and dynamic behavior of such components. However, there is no unified modeling framework to compose components that use different paradigms and/or tools. In this paper, we present the syntax and semantics of such an integration language and its component-based design, where components can embed models from different tools, formalisms, and paradigms such as Bond Graphs and Modelica models. Our framework is built around common set of interface concepts to support heterogeneous composition and interchangeability among modeling paradigms.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*

1. INTRODUCTION

Cyber-Physical Systems (CPS) are heterogeneous systems composed of computational and physical components, which often appear in safety-critical applications such as avionics, nuclear power plants, and life-critical control systems. Complex electro-mechanical systems, such as next generation combat vehicles, also fall into this category. The latter is the primary focus of DARPA's Adaptive Vehicle Make (AVM), a portfolio of programs that address revolutionary approaches to the design, verification, and manufacturing of complex defense systems and vehicles. This paper addresses the composition of complex electro-mechanical components of CPS with the following characteristics: (i) the composition includes causal and acausal components, (ii) physical variables are shared across component boundaries, (iii)

component behavior models are represented using different paradigms, and (iv) the models are supplied by different tools. Our solution also supports interchangeability of components with the same functionality.

This paper is organized as follows. We will present background information in Section 2 and motivate our work in Section 3. Section 4 presents our solutions to the problems detailed above, featuring the syntax and the semantics of the integration language. Finally, conclusions are drawn in Section 6.

2. BACKGROUND

In order to describe the composition of CPS, we must clarify the concept of causality. From a simplified point of view, a system is considered *acausal* if all forms of its equation model features an algebraic loop. Otherwise, it is considered a *causal* system. A more precise definition can be found in [14]. According to its terminology, the class of systems considered by this paper is nonanticipating. Typically, computational systems are causal, while physical systems may be acausal.

2.1 Simulink

Simulink [13] is an environment for Model-Based Design and Simulation and includes a variety of libraries. Simulink models use causal signal flow diagrams that have input and output ports. Each connection is a directional, causal connection between a source and a destination port. The underlying simulation engine for Simulink is provided by MATLAB, which appears in the syntactical constructs as well.

2.2 Acausal Modeling Paradigms

Control design involves two distinct paradigms: the discrete specification of the controller and a continuous processes governed by the laws of physics. While a discrete controller can naturally be modeled as signal flows, the key to modeling physics is the use of an acausal modeling framework [14]. Using causal models (e.g. signal data flows) to represent interactions between components that share physical variables can be complex. Typically, acausal physics models have power ports, which represent a simultaneous, bidirectional energy exchange between components [11] [9]. A well-formed model in an acausal framework represents a

well-formed set of dynamic equations. Acausal models typically must interface with causal models to represent the integration of a controller function into a physical system. This requires carefully directed variable sharing between cyber and physical system components (e.g. through sensors and actuators). This is one of the key issues of this paper. In the following, we discuss the two most important acausal modeling paradigms.

2.2.1 Hybrid Bond Graph Modeling

Bond Graphs [9] are a physics-based, domain-independent graphical notation for describing the behavior of components and systems which can be modeled using differential algebraic equations. Bond graphs generically model the energy exchange between different types of energy storage and conversion components, analogous to a circuit diagram in the electrical domain. Bond graphs are composed of the following primitive elements: source of effort (Se), source of flow (Sf), resistor (R), capacitor (C), inertia (I), transformer (TF), gyrator (GY), one-junction (1), and zero-junction (0). These primitive elements are connected through junctions, which correspond to either common flow (one-junction) or common effort (zero-junction). For example, in electrical circuits, one-junctions (common flow) represent series connections and zero-junctions (common effort) represent parallel connections. The connections between the primitive Bond Graph elements and the junctions are called bonds, each of which represents an effort and a flow variable. The product of the effort and flow variables is the power flowing between the connected elements.

In our previous work, we have extended Bond Graphs in multiple ways to include modulated elements, domain-specific power ports, and hierarchical modeling support [9]. Domain-specific power ports (e.g. electrical power port) connect quantities in one component with another, and each includes two variables: a domain specific effort (e.g. voltage) and a domain specific flow (e.g. current). Power ports can be connected to either a one-junction or a zero-junction only. Bond Graphs easily and uniformly represent electrical, rotational, translational, thermal, and other types of power domains. Input signals are either control parameters (e.g. modulate an effort or a flow source) or directly influence the system behavior through functions on the physical variables (i.e. determine the parameter value of a modulated element). The Hybrid Bond Graph Language (HBGL) includes the ability to resolve causality and create a Simulink model from a Bond Graph model [9]. HBGL also supports domain specific power ports for valid component composition.

2.2.2 Modelica

Modelica is a modeling language for dynamic systems that is equation-based and uses signals to express physical constraints imposed by physical connections in the system [11] [3]. Modelica is an object-oriented mathematical modeling approach to systems modeling. The building blocks of the models are stereotyped classes, of which the most important constructs are models, blocks, and connectors. Models can describe hybrid models, which are composed of discrete and continuous variables. Blocks are similar to models with a restriction that they can only expose those connectors that are tagged as *input* or *output*. Connectors are ports representing causal/acausal signal variables. The behavior of the

building blocks is defined by equations. Modelica does not strive for the uniformity of representation that Bond Graphs provide, but provides a library of standard components for each physical modeling domain called Modelica Standard Library (MSL). Also, Modelica simplifies connecting physical variables by its interconnection model. Interconnections among components are made using connections (i.e. connect statements) between connectors, which directly represent physical connections (e.g. attaching a wire to a pin of an electronic device), enabling the compositional definition of system behaviors. Each connector that represents a physical interface has the same number of flow and potential variables. For instance, an electrical pin connector has a voltage (potential) and current (flow) variable. For a well-formed model, Modelica compilers translate all of the model subsystems and connections into equations suitable for simulation or analysis. Unlike Bond Graphs, the Modelica language is an international standard that has well-supported commercial tools. Modelica is an open-source language and has some level of open-source compiler support as well as an open-source standard library (MSL).

3. MOTIVATION

In order to drive our focus, outlined in Section 1, we needed a component-based modeling framework of such systems, where component models were coming from different tools. We illustrate this through a case study using a simple electro-mechanical system. A schematic of the system is shown in Figure 1. The system is modeled using Modelica blocks and the simulation can be executed using a Modelica tool. The example uses three different domains: (i) electrical (*stepVoltage*, *resistor*, *inductor* and *emf*), (ii) mechanical rotational (*emf*, *inertia*, *damper* and *idealRollingWheel*), and (iii) mechanical translational (*idealRollingWheel* and *mass*).

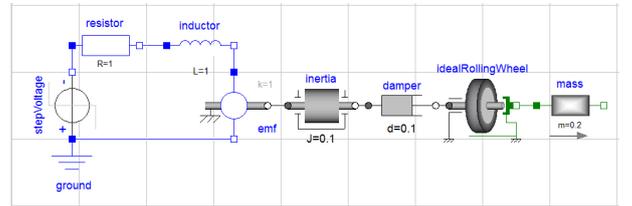


Figure 1: Schematic diagram of the case study

To model further components of the same system (e.g. *resistor* and *idealRollingWheel*), we would like to use the formalisms of another modeling paradigm, namely Bond Graphs. In order to support this setting in a design environment, we need the following: (i) a common, consistent modeling framework that can interface to models that are based on various formalisms and paradigms, (ii) a composition approach that is able to integrate and simulate the system as a whole, and (iii) the ability to adapt the system models to widely used tools in order to be able to simulate the composed system.

In the following, we illustrate the objective with an example for the expected solution. Regarding the case study, this means that we create the same system along with its connections shown in Figure 2. The puzzle pieces represent components, which are Modelica models. The expected solution along with its simulation results is shown in Figure 2.

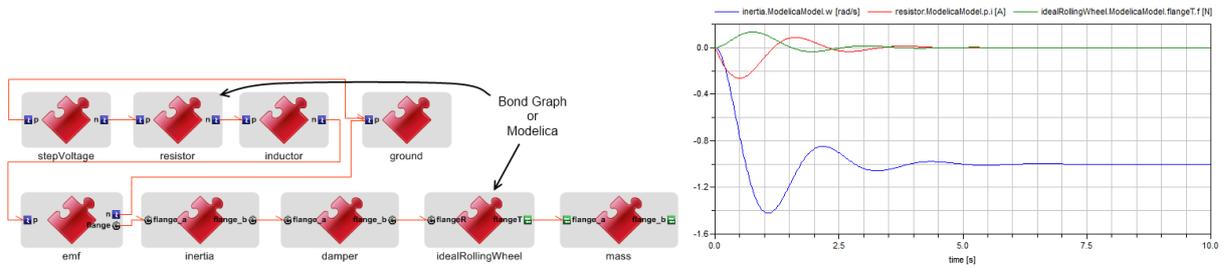


Figure 2: CyPhyML diagram and results

The plot shows the simulation results of: (1) the angular velocity of the *inertia*, (2) the force on the translational interface of the *idealRollingWheel*, and (3) the current on the positive electrical pin of the *resistor*.

We have changed the underlying model types from Modelica to Bond Graphs for the *resistor* and the *idealRollingWheel* components. The generated simulation results are identical with respect to the variables mentioned above and as it is shown in Figure 2. We used a simple example, where the models of the components are easy to develop either using Modelica or Bond Graphs. A more practical example is a drive train model, where the engine and transmission components can be modeled using Modelica, and the final drive and the load components can be modeled as Bond Graphs. The integration language supports heterogeneous composition of components implemented using Bond Graphs and Modelica. Thus, it should characterize interfaces in terms of the commonalities between the supported modeling paradigms. We will present the integration language and its semantics.

4. APPROACH

We chose a Domain Specific Modeling Language (DSML) to implement the integration language. DSMLs are flexible to support iterative refinement and can capture multiple paradigms as well as complex electro-mechanical system domains. We used the Generic Modeling Environment (GME), a meta-programmable editor, for creating this domain-specific integration language [7]. For the AVM program, a Cyber-Physical Modeling Language (CyPhyML) was created for a diverse set of design tasks. This paper describes the subset of the CyPhyML meta-model (language) and explains in detail the concepts relevant to the behavior of dynamic systems. CyPhyML, as an integration meta-model, supports two different paradigms with respect to modeling dynamic systems: Hybrid Bond Graph Language (HBGL) and Modelica.

4.1 Modeling language

CyPhyML has different modeling aspects and one of them is the dynamics aspect of components. In CyPhyML components different behavior models can provide a common set of interfaces, and can be composed through them. Also, The modeling framework supports hierarchical composition of reusable CyPhyML components.

4.1.1 Interfaces of CyPhyML components

A *Component* in CyPhyML is an atomic building block. A CyPhyML component is defined by its interfaces: (i) parameters, (ii) signal ports, and (iii) power ports. Each component includes a *behavior model*, a mathematical abstraction of a real-world physical system which represents the component's dynamic behavior. A behavior model can be specified using the HBGL or Modelica. HBGL models can be drawn within a component using CyPhyML, while Modelica models are incorporated by referring to an external Modelica model. These references in CyPhyML replicate the parameters, signal ports, and power ports of Modelica models, but do not include the internals or Modelica code. These external models may come from the Modelica Standard Library or from a user-defined library. Both behavioral model

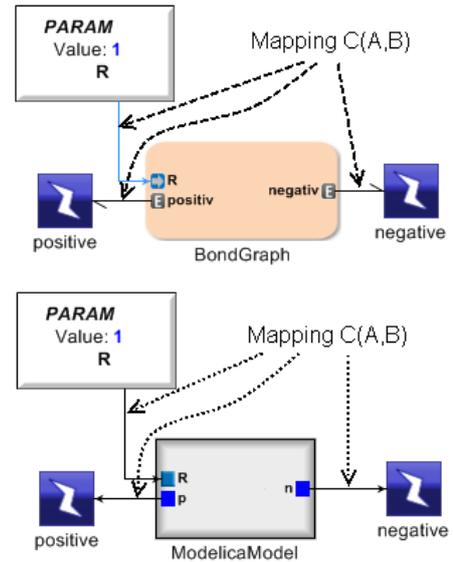


Figure 3: Component model using Bond Graph or Modelica

interfaces should be mapped to the CyPhyML component interfaces shown in Figure 3. Figure 3 depicts two components representing the same behavior: the component on the top uses HBGL and the component on the bottom uses Modelica.

4.1.2 Internals of CyPhyML components

CyPhyML components use their own interface definition to support composition between components and to provide common structure among different modeling paradigms. Fig-

Figure 3 shows an example for the mapping between the CyPhyML component-level interfaces and behavioral model interfaces. The mapping is defined by a connection $C(A, B)$, where C represents the connection; A is the source element; and B is the destination element. The mapping and the connections, which are described below are only the internal connections that wire the component up to its interfaces. Examples are given for Figure 3. The semantics of the connections are as follows. For parameters (real numbers), $C(A, B)$ means $B := A$ (e.g. Eq. 1) during the model instantiation. This mapping represents a causal assignment between A and B , where A drives B . For signal ports, $C(A, B)$ means the signal value of B is equal to the signal value of A during the entire simulation. It is also a causal connection and A drives B .

$$\begin{aligned} \text{BondGraph}.R &:= R \\ \text{ModelicaModel}.R &:= R \end{aligned} \quad (1)$$

While power ports are acausal connections and they contain multiple variables at the same time, the mapping needs to be defined through equations using variables of the power ports. Modelica electrical pin maps to CyPhy electrical power port: $C(A, B)$ is going to be resolved as Eq. 2 (e.g. Eq. 3).

$$\begin{aligned} A.\text{voltage} &= B.\text{voltage} \\ A.\text{current} + B.\text{current} &= 0 \end{aligned} \quad (2)$$

$$\begin{aligned} \text{ModelicaModel}.p.v &= \text{positive.voltage} \\ \text{ModelicaModel}.p.i + \text{positive.current} &= 0 \end{aligned} \quad (3)$$

Bond graph electrical port maps to CyPhy electrical power port: $C(A, B)$ is going to be resolved as shown in Eq. 4 (e.g. Eq. 5).

$$\begin{aligned} A.\text{effort} &= B.\text{voltage} \\ A.\text{flow} + B.\text{current} &= 0 \end{aligned} \quad (4)$$

$$\begin{aligned} \text{BondGraph}.positive.\text{effort} &= \text{positive.voltage} \\ \text{BondGraph}.positive.\text{flow} + \text{positive.current} &= 0 \end{aligned} \quad (5)$$

Modelica mechanical translational flange maps to CyPhy translational power port: $C(A, B)$ is going to be resolved as shown in Eq. 6.

$$\begin{aligned} A.\text{position} &= B.\text{position} \\ A.\text{force} + B.\text{force} &= 0 \end{aligned} \quad (6)$$

Bond graph mechanical translational port maps to CyPhy translational power port: $C(A, B)$ is going to be resolved as shown in Eq. 7.

$$\begin{aligned} A.\text{flow} &= \text{der}(B.\text{position}) \\ A.\text{effort} + B.\text{force} &= 0 \end{aligned} \quad (7)$$

Modelica mechanical rotational flange maps to CyPhy rotational power port: $C(A, B)$ is going to be resolved as shown in Eq. 8.

$$\begin{aligned} A.\text{angle} &= B.\text{angle} \\ A.\text{torque} + B.\text{torque} &= 0 \end{aligned} \quad (8)$$

Bond graph mechanical rotational port maps to CyPhy rotational power port: $C(A, B)$ is going to be resolved as shown in Eq. 9.

$$\begin{aligned} A.\text{flow} &= \text{der}(B.\text{angle}) \\ A.\text{effort} + B.\text{torque} &= 0 \end{aligned} \quad (9)$$

The structure of CyPhyML is rich enough to define multiple behavior models for the same component e.g. on different fidelity or abstraction level. The current state of the tools does not support this concept, but CyPhyML does. Related part of the simplified conceptual meta-model of CyPhyML is shown in Figure 4.

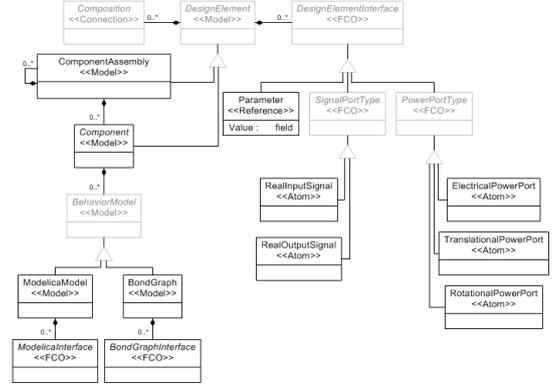


Figure 4: CyPhyML meta-model part

4.1.3 External of CyPhyML components

A subsystem or a system that contains components and other subsystems is called Component Assembly in CyPhyML. Component Assemblies have the same interfaces as components. There are two key differences between components and component assemblies: (i) Component assemblies can contain other components and other component assemblies; (ii) behavior of component assemblies is defined implicitly through the composition of its child objects, but behavior of components is defined explicitly in its behavior model. Feature (i) makes building hierarchical systems and subsystems possible. Feature (ii) hides the behavioral modeling language specific concepts, and users use a single abstract concept to build system models. Since component assemblies and components have the same type of interfaces as well as composition rules, those elements are interchangeable. Switching between components and component assemblies helps system designers refine their systems and subsystems further. For instance, assume that we would like to design a drive train (Component Assembly), and our system breakdown structure contains a power pack (Component) and a drive line (Component). Each component has an associated behavior model, but if one would like to refine the system-subsystem breakdown structure, the power pack could be decomposed into a Component Assembly, with an engine (Component) and a transmission (Component). In this case, the dynamic behavior model for the power pack is given by the composition of two components: engine and transmission. *oka*

4.1.4 Composition

Composition of Components, Component Assemblies, and internals of Component Assemblies in CyPhyML is supported through ports of various types. Each port type has one or more variables based on its type. The composition rules are different for: (i) parameter ports, (ii) signal ports (causal), and (iii) power ports (acausal) from mul-

multiple physics domains. Figure 5 depicts an abstract conceptual model of CyPhyML between ports. CyPhyML has many more connection types to enforce compatibility between ports on the meta-model level, and there are some composition restrictions that are encoded as constraints using the Object Constraint Language. The semantics of com-

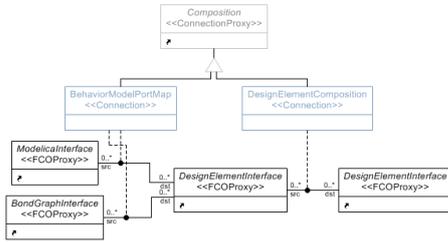


Figure 5: CyPhyML meta-model composition

position is as follows: Parameters are variables that can be connected to subsystem level parameters or to parameters of the behavior model. The directional connection between parameters means that they pass their values to the subsystem level during model instantiation. Their value is time-independent; therefore, they do not change during the dynamics simulation. Signal ports are causal input/output ports, which are used to perform some computation in the signal domain or pass signal values from controllers to actuators in the dynamic system. Their value is time-dependent during the dynamic simulation. To propagate parameters and signals among systems and subsystems takes no power. If modelers are interested in power loss on those connections, they need to use power interfaces to model power losses accordingly. There are different types of power ports such as electrical, mechanical translational, and mechanical rotational. Electrical power ports share potential and current (flow) variables, mechanical translational power ports share position and force (flow) variables, and mechanical rotational power ports share angle and torque (flow) variables. When multiple power ports are connected together, the algebraic sum of the flow variables is equal to zero and the potential variables are equal [14]. Since the composition is defined using acausal power interfaces, components can be power sinks (loads) or power sources (drivers) at any time, as determined by the power (i.e. energy) distribution in the system.

4.2 Simulation

4.2.1 Bond Graphs to Simulink

Our first approach supports Hybrid Bond Graph Language and Simulink blocks. The benefit of Simulink is its wide range of library elements, but Simulink itself does not support acausal modeling. We built a HBG library for Simulink in order to perform a smoother translation between CyPhyML models and Simulink. CyPhyML supports acausal models, but the translation has a disadvantage: All Simulink blocks are causal elements, which means that inputs and outputs must be identified during model construction time. Using HBGL, input and output relationships are given after we run a procedure called the Sequential Causal Assignment Procedure (SCAP). The Simulink HBG library is generic enough to perform the causality updates during simulation. The problem is that the Simulink model must capture all

possible signal paths that can occur during the simulation, and the model's complexity increases drastically. In addition to the size of the model, the solver has an additional overhead because the causality must be recomputed and updated during the simulation.

Another problem that we encountered is that each element from HBGL has an associated element in the Simulink HBG library, but for storage elements we used only integral blocks. The number of integral block in a system gives the number of independent state variables. Therefore, if the original model had dependent state variables, we could not map it to a valid Simulink model using this approach. Thus, we were looking for a modeling language which supports acausal modeling, open-source libraries, and has some open-source tools/solvers which can execute the simulations of the composed system models. The modeling language should also resolve the dependent state-variables and their initialization problem.

4.2.2 CyPhyML models to Modelica

To overcome this limitation, we turned towards Modelica, which supports acausal modeling, open-source libraries, and has some open-source tools/solvers which can execute the simulations of the composed system models. These tools support optimization techniques and also resolve the dependent state-variables and their initialization problem. The modeling approach described below supports acausal system capture via Modelica Standard Library power ports, with Modelica Standard Library semantics as well as Bond Graphs.

One of the target languages of CyPhyML is Modelica, which means that a translator can generate equations or instances of library elements that use equations (e.g. a Bond Graph library) as well as connect statements for connections. The translator uses interfaces from Modelica Standard Library 3.2 and elements from the Bond Graph library for Modelica if bond graphs are present in the model. Modelica does not have any notion of CyPhyML elements, and the translator works from a semantically rich domain to a semantically poor domain. For future purposes and clarity, the generated Modelica models are marked with information from the source domain using Modelica class inheritance. For instance, each generated model that was a component in CyPhyML extends a base class from ISIS.Icons.Component in the Modelica model. After the CyPhyML models have been translated into Modelica models, the hierarchical structure of the generated models resembles the original CyPhyML model's hierarchy. This helps users to navigate their model the same way they do in CyPhyML. The variable tree structure of simulation results look the same, and it is easy to find the appropriate subsystems and plot variables using any Modelica tool.

CyPhyML uses Test Benches to establish a simulation or test case for a system. In general, a system itself is not enough to perform a simulation; it requires a context, which contains test drivers and/or environments that interact with the system. Test Benches are described in [10] and are out of the scope of this paper, but were used to generate the simulation results for the examples.

5. RELATED WORK

Modeling of real-time and embedded systems is not new; for example the tools AToM3 [1], Fujaba [6] and Ptolemy [2] have demonstrated the feasibility of the approach. In comparison to our work, we focus on different paradigms, and our approach is more centered on composition as well as the integration of CPS. Ptolemy [2] is an actor-oriented modeling framework that supports heterogeneous model composition using various Models of Computation (MoC). However, the actors appears to have clearly distinguished input and output ports through which different actors interact with each other, as determined by the MoC. Outputs depend on inputs and the state of the actor, hence the inputs are always independent variables - thus actors are functional (if we count the actor state as an 'input' to the function). In our approach, the components establish a mathematical constraint (expressed as an equation) among the values represented by the ports of the component which is always true, and where ports are directionless. Model-based composition has been specified by several Domain Specific Modeling approaches. For example, [8] and [4] summarize the main results in Model-Based Design of CPS. As far as heterogeneous systems are concerned, [5] provides a thorough overview of the existing modeling approaches. Our focus is on the composition of heterogeneous components, instead a general approach we provide a solution for modeling complex electro-mechanical systems. The concept of interchangeable components is a novel result of our approach. It is also possible to use SysML for specifying the composition rules of CPS components. Paredis et al [12] present a transformation specification to integrate Modelica models with SysML models. However, we not only provide a modeling language with syntax and semantics, but a tool chain for automatic transformations as well. Another difference is that SysML4Modelica contains the Modelica blocks and its equations as well. Our tools abstract away the specifics of Modelica and focus only on the details required to perform the integration of physical components. Furthermore, our solution is able to support multiple paradigms, such as Bond Graphs.

6. CONCLUSIONS

In this paper we presented the syntax and the semantics of an integration language that uses the multi-paradigm modeling approach. In this component-based integration language, component models can embed models from different tools, formalisms, and paradigms such as Bond Graphs and Modelica. While the integration language has well-defined interfaces and composition rules, components (using the same interface) implementing the same functionality are interchangeable. Moreover, components can be composed through their interfaces, which are parameters, signal ports (causal) and power ports (acausal). While both signal and power ports are supported, the composition is able to handle causal and acausal components. Connections between power ports make sharing state variables across component boundaries possible. We selected Modelica tools (compilers/solvers) to perform the simulation of the composed system design.

7. ACKNOWLEDGMENTS

This work was supported by DARPA under contracts FA8650-10-C-7082 and FA8650-10-C-7075.

8. REFERENCES

- [1] J. de Lara, H. Vangheluwe, and M. Alfonseca. Meta-modelling and graph grammars for multi-paradigm modelling in atom3. *Software and Systems Modeling*, 3:194–209, 2004. 10.1007/s10270-003-0047-5.
- [2] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127 – 144, jan 2003.
- [3] P. Fritzson and P. Bunus. Modelica - a general object-oriented language for continuous and discrete-event system modeling and simulation. In *Simulation Symposium, 2002. Proceedings. 35th Annual*, pages 365 – 380, april 2002.
- [4] H. Giese, B. Rumpe, B. Schätz, and J. Sztipanovits. Science and engineering of cyber-physical systems (dagstuhl seminar 11441). *Dagstuhl Reports*, 1(11):1–22, 2011.
- [5] C. Hardebolle and F. Boulanger. Exploring multi-paradigm modeling techniques. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 85:688–708, November 2009.
- [6] S. Henkler, J. Greenyer, M. Hirsch, W. Schafer, K. Alhawash, T. Eckardt, C. Heinzemann, R. Löffler, A. Seibel, and H. Giese. Synthesis of timed behavior from scenarios in the fujaba real-time tool suite. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 615 –618, may 2009.
- [7] V. U. ISIS. Generic Modeling Environment. <http://www.isis.vanderbilt.edu/projects/gme>.
- [8] S. Lacoste-Julien, H. Vangheluwe, J. de Lara, and P. Mosterman. Meta-modelling hybrid formalisms. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 65 –70, sept. 2004.
- [9] Z. Lattmann. A multi-domain functional dependency modeling tool based on extended hybrid bond graphs. Master's thesis, Vanderbilt University, 2010.
- [10] Z. Lattmann, A. Nagel, J. Scott, K. Smyth, J. Ceisel, C. vanBuskirk, J. Porter, S. Neema, T. Bapty, D. Mavris, and J. Szipanovits. Towards Automated Evaluation of Vehicle Dynamics in System-Level Designs. In *Proc. ASME International Design Engineering Technical Conf. & Computers and Information in Engineering Conf. (IDETC/CIE 2012)*, Aug 2012.
- [11] M. Otter, H. Elmqvist, and S. E. Mattsson. Multidomain Modeling with Modelica. In P. A. Fishwick, editor, *Handbook of Dynamic System Modelling*, chapter 36, pages 36.1 – 36.27. Chapman & Hall/CRC, 2007.
- [12] C. J. Paredis, Y. Bernard, R. M. Burkhart, H.-P. de Koning, S. Friedenthal, P. Fritzson, N. F. Rouquette, and W. Schamai. An Overview of the SysML-Modelica Transformation Specification. In *2010 INCOSE International Symposium*, July 2010.
- [13] The MathWorks, Inc. Simulink/Stateflow Tools. <http://www.mathworks.com>.
- [14] J. Willems. The behavioral approach to open and interconnected systems. *Control Systems, IEEE*, 27(6):46 –99, Dec 2007.