Research Article

Model Based Control System Design Using SysML, Simulink, and Computer Algebra System

**Takashi Sakairi,¹ Eldad Palachi,² Chaim Cohen,² Yoichi Hatsutori,¹
Junya Shimizu,¹ and Hisashi Miyashita¹**

¹ IBM Research-Tokyo, IBM Japan, NBF Toyosu Canal Front Building 6-52, Toyosu 5-chome, Koto-ku, Tokyo 135-8511, Japan

² Rational, IBM Israel, Park TMR, P.O. Box 2523, Rehovot 76123, Israel

Correspondence should be addressed to Takashi Sakairi; sakairi@jp.ibm.com

Received 7 October 2012; Revised 16 June 2013; Accepted 1 July 2013

Academic Editor: Derong Liu

Copyright © 2013 Takashi Sakairi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The Systems Modeling Language (SysML) is a standard, general-purpose, modeling language for model-based systems engineering (MBSE). SysML supports the specification, analysis, and design of a broad range of complex systems such as control systems. The authors demonstrate how they can integrate a SysML modeling tool (IBM Rational Rhapsody) with a proprietary simulation tool (MathWorks Simulink) and a Computer Algebra System (CAS) to validate system specification. The integration with Simulink enables users to perform systems engineering process in a SysML model, while designing continuous control algorithms and plant behavior in Simulink, and to validate the behavior by simulating the overall composition in Simulink. The integration with a CAS enables the evaluation of mathematical constraints defined in SysML parametric diagrams. The authors also show the overall approach using a Dual Clutch Transmission (DCT) and a Cruise Control System as examples.

1. Introduction

The Systems Modeling Language (SysML) has been under development since 2001 [1] by the International Council of Systems Engineering (INCOSE) and the Object Management Group (OMG) organizations, as a standard modeling language for model-based systems engineering (MBSE). The need for a standard modeling language originates from the fact that interdisciplinary complex systems, which require high-quality efficient systems engineering activities, are being developed by different teams, sometimes across companies in different countries, using a variety of tools and methods.

SysML extends a subset of the Unified Modeling Language (UML) [2, 3] targeted mainly for software engineering. This acknowledges the need for continuity from systems engineering to software engineering in the development of software intensive systems.

SysML supports the specification, analysis, design, and verification of a broad range of complex systems such as control systems. Part of the control systems specification includes dynamic (time-dependent) continuous behavior of the control algorithms and the physical environment;

these systems interact with the plant. In addition, SysML introduces a new kind of diagram called a parametric diagram used to define mathematical relations among attributes of the system using equations and inequalities. It is possible to define dynamic behavior of the system using time derivative function [4]. Using differential equations allows one to fully specify dynamic behavior; however, they are inadequate to design the reactive and transformational behaviors of the designed system parts.

One of the most effective methods of verifying a system is simulation. To simulate a system that consists of hardware and software, we have to define both discrete-time behavior and continuous-time behavior. To simulate a combination of discrete-time and continuous-time elements, we have to use a simulation tool such as Simulink or Modelica [5]. Although a simulation tool is well suited for defining a detailed behavior of dynamic system, it is not well suited for conducting the overall systems engineering processes such as requirement analysis and architectural analysis. To develop a complex system, we have to perform systems engineering processes before defining detailed behavior. Various researchers proposed integration of a SysML tool with a simulation tool to

perform both systems engineering process and simulation. Some researchers integrate SysML with Simulink [6–9], and others integrate SysML with Modelica [10].

SysML also supports engineering analysis using parametric diagrams [11, 12]. Parametric diagrams are used to specify constraints and associate them with the design elements as an integral part of the overall engineering model. If these constraints are well defined, an analysis tool can be developed to perform various kinds of analysis such as determining whether the values of the various systems attributes satisfy constraints. Until several years ago, no SysML modeling tool supports analysis using parametric diagrams, so very few SysML users draw parametric diagrams [13]. But now, several commercial SysML tools support analysis using parametric diagrams [4, 14]. Such tools call a Computer Algebra System (CAS) to solve simultaneous equations defined in parametric diagrams.

In this paper, first we describe our tool that integrates a SysML modeling tool (IBM Rational Rhapsody) with a simulation tool (MathWorks Simulink). Second we explain our tool that evaluates constraints defined in parametric diagrams. Then we show systems engineering processes for designing a Dual Clutch Transmission (DCT) [15, 16] as an example of control systems using our tools. We also show a Cruise Control System example [17] to explain how to interact with users while simulating using a panel diagram. Finally we conclude the paper.

2. Our Tool to Integrate SysML with Simulink

2.1. Existing Integration. Rhapsody is a UML/SysML modeling tool that has many capabilities including model execution of SysML models. Before developing our new tool, Rhapsody already had two integration points with Simulink.

The first integration point is importing Simulink components into Rhapsody (Figure 1). Some parts of Rhapsody model are designated as Simulink blocks: their detailed behavior is described using Simulink block diagrams. The user generates C/C++ code from the corresponding Simulink models by using RealTime Workshop Embedded Coder. Each Simulink block in the Rhapsody model contains references to the Simulink model and the generated code. Rhapsody can deduce the Simulink interface in the form of SysML atomic flow ports by reading the Simulink model and in addition it can compile and link the generated code into an executable program for simulating the entire model using Rhapsody's simulation framework. Liu and Cao used this integration point to design a flight control system [18]. One problem of this integration point is that we cannot use variable step solvers. Using fixed step solvers, simulation result is not accurate if we use large step size; it takes long time if we use small step size to get accurate result.

The second integration point is creating an S-function using Rhapsody (Figure 2). An S-function is a mechanism for extending the capabilities of Simulink by introducing blocks implemented in C/C++. This enables users to model a part of system using Rhapsody and generates an S-Function block from the SysML block stereotyped «S-FunctionBlock». The generated S-Function can be used in the Simulink model.

FIGURE 1: Importing a Simulink component.

FIGURE 2: Generating an S-Function.

One problem of this integration point is that it is difficult for users to use Rhapsody as a systems' engineering tool. They can only model a part of a system not a whole system using SysML. To use Rhapsody as a system engineering tool, this integration point is not acceptable.

2.2. New Integration. We developed our new integration tool to address the above problems (Figure 3). First users design a system using Rhapsody describing various aspects of the system such as requirements, use cases, structures, and behaviors. To specify behaviors, they can use SysML diagrams such as state machine diagrams, and they can also use Simulink models by referencing them as SysML blocks stereotyped «SimulinkBlock». Second, users generate a Simulink model from the Rhapsody model. Behaviors defined by SysML diagrams are converted to a single S-Function, and behaviors defined by Simulink models are converted to ModelReference blocks. A ModelReference block is a Simulink block that refers to a Simulink submodel. Third, users simulate the generated Simulink model. They can use variable step solvers if needed. They can see the Rhapsody's animated diagrams/views to understand the generated S-Function's behavior.

This new integration tool solves the problems described in Section 2.1: (1) we can use variable step solvers, so we can get accurate simulation result without using fixed small step size; (2) we can use Rhapsody as a systems' engineering tool. If users want to use Rhapsody as a control program development tool, they want to use the first integration point, because Rhapsody can generate C code from behavior defined by UML/SysML diagrams such as state machine diagrams. But if users want to use Rhapsody as a designing tool, new integration tools are better than the existing integration points described in Section 2.1.

FIGURE 3: New integration.

Our tool can also help engineers to find a set of appropriate values of system parameters by running multiple simulations automatically. Simulink users often use MATLAB variables in Simulink models to represent system parameters. We enhanced our integration tool to maintain system parameters in SysML and synchronize them with MATLAB variables in Simulink model. Users can modify system parameters' value and simulate the generated Simulink models using the modified values.

Defining behaviors using SysML diagrams is optional. Users can define all behaviors using Simulink if they want. In that case, no S-function is generated. Although Simulink is a good tool to simulate dynamic systems, it does not have a capability to support systems engineering processes. Users can use Rhapsody as a systems' engineering process support tool for Simulink.

This new integration tool is shipped as a component of Rhapsody, so several researchers use this tool. Wang and Baras use this tool for designing wireless sensor networks [19], and Garro and Tundis use this tool for analyzing reliability [20].

2.3. Mechanism of New Integration. We explain the mechanism of our new integration tool in this section. Figure 4 shows relationships among five SysML blocks. CruiseControlSystem block has pf, vd, c, and s as parts, and the types of the parts are PFeedback, VehicleDynamics, Controller, and Speedometer, respectively. CruiseControlSystem block is applied `<<StructuredSimulinkBlock>>` stereotype; this means internal structure of this block will be converted to a Simulink model. PFeedback block and VehicleDynamics blocks are applied `<<SimulinkBlock>>` stereotype; this means these blocks refer to Simulink submodels. Icons indicating state transition are shown in CruiseControlSystem block, Controller block, and Speedometer block; this means these blocks' behavior is defined in state machine diagrams. Controller block has a reference to CruiseControlSystem block's parts; this means Controller block can access the part.

Figure 5 shows an internal structure of CruiseControlSystem block. Arrows surrounding small box are flow ports. Flow ports specify the input and output items that may flow between blocks. Lines connecting flow ports represent data flow.

Our tool generates an S-Function block `SFunctionBlockCruiseControlSystem` from CruiseControlSystem block (Figure 6) by the following steps.

FIGURE 4: Block definition diagram of a cruise control system.

FIGURE 5: Internal block diagram of CruiseControlSystem block.

FIGURE 6: Internal block diagram of generated S-Function block.

- (1) Remove parts pf and vd, because they refer to Simulink submodels.
- (2) Add flow ports pf_desiredSpeed and vd_actualSpeed.
- (3) Add lines between pf_desiredSpeed and desiredSpeed, and between vd_actualSpeed and actualSpeed.
- (4) Apply `<<S-FunctionBlock>>` stereotype to the generated block.

Rhapsody has a capability to generate an S-function from an S-Function block as described in Section 2.1.

Then our tool generates a Simulink model (Figure 7) from the internal block diagram of CruiseControlSystem block (Figure 5). SysML parts vd and pf are converted to Simulink ModelReference blocks vd and pf, respectively. The

FIGURE 7: Generated Simulink model.

SFunctionBlockCruiseContrlSystem block is converted to the Simulink S-Function block `rhp_s_function`.

3. Our Tool to Evaluate Parametric Diagrams

We developed a tool to evaluate parametric diagrams, Parametric Constraint Evaluator (PCE). This section briefly explains PCE. The details of PCE can be found in [4].

The main SysML model elements used for parametric analysis are constraint blocks, constraint properties, constraint parameters, and binding connectors. Constraint blocks specify a set of equations and inequalities. The parameters used by the expressions of the constraints are explicitly specified as constraint parameters owned by the constraint block. The constraint parameters are a specialized type of UML/SysML property and as such they have names, types, and multiplicities. A constraint property is an instantiation or usage of a constraint block in a specific context, allowing the same constraint block, defining a set of equations, to be used in different contexts with different values. Constraint properties are connected via the constraint parameters by binding connectors, either to other constraint properties via constraint parameters or to other UML/SysML property uses in the design model. A binding connector between properties means that the values of the properties at each end of the connector are equal.

The language for specifying the constraints expression is not defined by SysML, so tool vendors have to define the language. PCE calls CAS to evaluate constraints. PCE supports two kinds of CAS, Maxima and MATLAB Symbolic Math Toolbox. Each CAS uses its own language. We chose a subset of the Modelica language [5] as a CAS independent constraint language. The parameters of the constraints are intentionally not directional in SysML [1, 11]. For example, an expression " $f = m * a$ " contains three variables. Our tool can calculate f from m and a , m from f and a , and a from f and m .

PCE is shipped as a component of Rhapsody. Bleakley, Whitfield, and Lapping propose to use this tool to perform trade study [21].

FIGURE 8: Upper level requirements.

4. Case Study 1

4.1. Dual Clutch Transmission. We developed a model of a Dual Clutch Transmission (DCT) using Rhapsody, calculated a system parameter by evaluating a parametric diagram, and validated the model by simulating it using Simulink. A DCT is a kind of automotive transmission and a recent trend of the powertrain control. A DCT has two independent subgear-boxes each connected to the engine via its own clutch. One subgearbox contains the odd gears (1, 3, 5...) and the other the even gears (2, 4, 6...). By dividing the gears through the dual clutch, the DCT becomes fully power shiftable [15].

We chose a DCT as an example since we have to consider (1) whole automotive system, (2) both hardware and software, and (3) many requirements to differentiate the performance in both the functional aspects and the nonfunctional aspects.

4.2. Outline of the Design Process. We performed the following steps to design the DCT:

- (1) requirement analysis,

FIGURE 9: Functional requirements.

- (2) system components definition,
- (3) simulation context definition,
- (4) parametrics definition,
- (5) parametrics evaluation,
- (6) plant behavior definition,
- (7) controller behavior definition,
- (8) connection definition,
- (9) test cases definition,
- (10) simulation.

4.3. Requirement Analysis. First, we sorted out requirements from upper-level systems or concepts by describing them in a requirement diagram (Figure 8). A requirement is a SysML model element that specifies a capability or condition that must be satisfied. A requirement diagram is a diagram to show requirements, relationship among requirements, and other model elements [1]. Upper level requirements are requirements from the stakeholders. We specified categories and priorities for each upper level requirement. Figure 8 shows four upper level requirements: efficient Shift change, safety, fuel consumption, and realizing “Driving Smarter.”

Next, we derived functional requirements in a requirement diagram (Figure 9). DeriveReq relationships are used in Figure 9. A DeriveReq relationship is a dependency between two requirements in a client requirement that can be derived from the supplier requirement [1].

4.4. System Components Definition. We expanded the target system into components in a Block Definition Diagram

FIGURE 10: System components.

(BDD) (Figure 10). Blocks are modular units of system description. BDD defines features of blocks and relationships between blocks [1]. Figure 10 shows four blocks: DualClutchTransmission, MechanicalParts, ShiftController, and ClutchSignalGenerator. DualClutchTransmission block has three parts, and types of the parts are MechanicalParts, ShiftController, and ClutchSignalGenerator.

We assigned functional requirements defined in the previous subsection to subsystem components in a requirement diagram (Figure 11). DeriveReq and Satisfy relationships are used in Figure 11. A Satisfy relationship is a dependency

FIGURE 11: Assign functional requirements to subsystem components.

between a requirement and a model element that fulfills the requirement [1].

Based on the subsystem components and their assigned requirements, we described the specific subsystem elements of blocks consisting of the DCT. The configuration of the mechanical parts would be restricted by functional requirements. Some parts may be defined by a specific vendor's product that has a fixed specification. Other parts may be described by the performance characteristics by mathematical equations in parametric diagrams.

4.5. Simulation Context Definition. We defined a context for simulation using a BDD (Figure 12). SimulationContext is a block to define a context for a simulation. The block is applied `«StructuredSimulinkBlock»` stereotype to indicate that it will be converted to a Simulink model. Plant, Engine, ClutchSignalGenerator, ExternalTorque, and Driver blocks are applied `«SimulinkBlock»` stereotype to indicate that behaviors of the blocks are modeled as Simulink submodels. Behaviors of ShiftController, WarningSignalGenerator, and TestDriver blocks are defined in state machine diagrams. Plant, Engine, and Driver blocks have attributes. These attributes are synchronized MATLAB variables in Simulink blocks. MassConstraint is a constraint block. A constraint block includes the constraint and the parameters of the constraint. A constraint block defines generic forms of constraints that can be used in multiple contexts.

4.6. Parametrics Definition. We defined relations among attributes using a parametric diagram (Figure 13). A parametric diagram describes the usage of the constraints. Three SimulationContext block's parts and one constraint property are shown in this diagram. This diagram defines the relation that the sum of the body mass, the engine mass, and the driver mass is equal to the total mass.

4.7. Parametrics Evaluation. We can evaluate the parametric diagram using Rhapsody Parametric Constraint Evaluator (PCE) [4]. Figure 14 shows a constraint view window, a user interface of PCE.

The values of the driver mass, the engine mass, and the body mass are known. When we click the "Evaluate" button, PCE calculates the value of total mass. We can synchronize the attribute values on this window with the SysML model by clicking the "Update Model" button.

4.8. Plant Behavior Definition. We defined behaviors of plant models using Simulink submodels and referred the Simulink submodels from a SysML model.

SysML's flow ports specify the input and output items that may flow between a block and its environment. Simulink has also concept of ports—Inport and Outport blocks. We have to synchronize interfaces between a SysML block applied `«SimulinkBlock»` stereotype and a Simulink submodel referred from the SysML block.

There are two ways to synchronize them. If users want to use an existing Simulink model, our tool generates flow ports and attributes to a SysML model from the existing Simulink model. If users want to create a new Simulink model, our tool generates a template Simulink model that contains Inport blocks, Outport blocks, and MATLAB variables from a SysML model. They can define the behavior of the plant by modifying the generated template Simulink model. Values calculated by PCE are also synchronized with MATLAB variables used in Simulink models.

4.9. Controller Behavior Definition. After describing the blocks related with an environment under the given functional requirements, we considered a control strategy whose behavior would also be described in the ShiftController block as a state machine diagram (Figure 15). The state machine represents behavior as the state history of an object in terms of its transitions and states [1].

A shift controller receives inputs from various sensors like the throttle position, current vehicle speed, current gear position, and engine speed. Based on these inputs and a command from a driver, the controller makes a decision of an upshift, downshift, or gear position holding. The threshold of each transition is adjusted to make the engine speed stable by trial and error method.

FIGURE 12: Simulation context.

FIGURE 13: Parametric diagram to define relationships between attributes.

States shown in Figure 15 represent the following meanings.

Neutral: no gear is engaged.

FirstGear_SecondGear: the first gear is engaged and the second gear is prepared.

SecondGear_FirstGear: the second gear is engaged and the first gear is prepared.

SecondGear_ThirdGear: the second gear is engaged and the third gear is prepared.

GenerateWarning_1, GenerateWarning_2: an incorrect command is received.

Events shown in Figure 15 represent the following meanings:

chUpcommand: upshift request,

chDowncommand: downshift request,

tm(100): wait 100 milliseconds.

The following attributes are defined in ShiftController block:

odd1, odd2, odd3: flags to represent the first, third, and fifth gear are selected or not;

even1, even2, even3: flags to represent the second, fourth, and sixth gear are selected or not;

odd: flag to represent the odd subgearbox is engaged or not;

even: flag to represent the even subgearbox is engaged or not.

Each attribute has a set method such as setEven.

The initial state is Neutral, and the first and second gears are selected by setEven1(1) and setOdd1(1). When the controller receives chUpcommand, the odd subgearbox is engaged by setOdd(1).

FIGURE 16: Internal structure of the simulation context.

FIGURE 17: Verification items.

FIGURE 18: Test case.

FIGURE 19: Generated Simulink model.

FIGURE 20: Simulation result.

After waiting 4000 milliseconds, the state is changed to SecondGear, and upcommand attribute is changed.

After waiting 1000 milliseconds, the state is changed to ThirdGear, and upcommand attribute is changed.

4.12. Simulation. First, we generated a Simulink model from the simulation context block (Figure 19). The layout of the generated Simulink model is almost same as the IBD of the simulation context block. Blocks applied «SimulinkBlock» stereotype are converted to a Simulink ModelReference

blocks. A Simulink Model block is a block that refers to a Simulink submodel. Behaviors defined in a SysML model are converted to an S-Function block.

Then we performed simulation using Simulink. We could see the simulation result using Simulink's capability such as a Scope block (Figure 20).

As defined in Figure 18, TestDriver sent three chUpcommand events to ShiftController at 1000, 5000, and 6000 millisecond, respectively. The state of the ShiftController was in "Neutral" state as an initial state. The state was

FIGURE 21: Animation of an S-Fuction's behavior.

FIGURE 22: Part of generated sequence diagram.

shifted from “Neutral” to “FirstGear_SecondGear” by the first chUpcommand at 1000 millisecond. The ShiftController sent the command to the odd-clutch in the plant and an impulse torque due to the engagement of the odd-clutch was generated at 1000 millisecond as shown in Figure 20. A similar impulse torque due to the second chUpcommand was confirmed at 5000 millisecond in Figure 20. In the ShiftController, the state was transited from “FirstGear_SecondGear” to “SecondGear_FirstGear” by the second command. On the other hand, there is no impulse torque at 6000 millisecond in Figure 20. In the state of “SecondGear_FirstGear,” the drive line was engaged to the second gear and the first gear was prepared as a next gear. Because the third gear was not prepared, the third chUpcommand generated on the TestDriver at 6000 millisecond is ignored in the ShiftController. As a result, the requirement defined in AN2 “Verify the case of gear selection when error occurs in shift operation.” was confirmed.

We could see the generated S-Function’s execution using Rhapsody’s animation capability (Figure 21). The highlighted state is the current state, and it is changed during the simulation. Figure 21 shows the animated state machine diagram captured at simulation time was 6000 milliseconds. ShiftController received chUpcommand event at SecondGear_FirstGear state. The state means the second gear is engaged and the third gear is not prepared, so the chUpcommand event did not cause shiftup.

We could also see the interactions during the simulation using generated sequence diagrams (Figures 22 and 23). A sequence diagram describes the flow of control between parts of a system [1].

We could find the GenerateWarning_1 state in ShiftController block. So Requirement ID AN2 was verified.

5. Case Study 2

5.1. Cruise Control System. Lussier explained the first integration point of Rhapsody and Simulink described in Section 2.1, and he used a cruise control system as an example [17]. We modified this model to use our new tool. The purpose of this case study is to show how to interact with users while simulating using a panel diagram.

A cruise control system is a system to maintain vehicle speed smoothly. It observes the actual speed, reasons the desired speed, and transmits the desired speed to the physical system at the driver’s command.

A driver can maintain speed and set speed (Figure 24).

Figure 4 shows the block definition diagram of a cruise control system, and Figure 5 shows the internal block diagram of the CruiseControlSystem block.

PFfeedback block refers to a Simulink model. It is a PI controller to use to vary the thrust applied to accelerate the vehicle based on the delta between the actual speed and the desired speed. VehicleDynamics block also refers to a Simulink model. It calculates from thrust to actual speed. Controller block is responsible for implementing the cruise control function. Speedometer is a sensor.

Figure 25 shows a state machine of Controller block. It uses several events:

reqCruise: a request to start/stop cruise mode

FIGURE 23: Another part of generated sequence diagram.

FIGURE 24: Use case diagram.

reqSet: a request to set the current actual speed as the desired speed

reqAccelerate: a request to increase the desired speed

reqDecelerate: a request to decrease the desired speed

reqBrake: a request to brake

reqResume: a request to set the desired speed to the previous value.

5.2. Simulation with a Panel Diagram. It is possible to define test cases using SysML diagrams (Figure 18) and to see the simulation result using plots (Figure 20). But this section explains another method. Figure 26 shows a panel diagram. A panel diagram is a diagram to place graphical elements, and users can see values, change values, and send events using them. In this panel diagram, users can see the speed value, see and change the accelerator value, and send events (reqCruise, reqSet, reqAccelerate, reqDecelerate, reqBrake, and reqResume).

Figure 27 shows the animation of a state machine diagram of Controller block and a panel diagram. Users can see state transitions on the animated state machine diagram, and they can also see the values of the measured speed and accelerator, and send events to the Controller block on the panel diagram.

FIGURE 25: A state machine diagram of Controller block.

FIGURE 26: A panel diagram.

FIGURE 27: Animation of a state machine diagram of Controller block and a panel diagram.

6. Conclusions

In this paper, we described our tools to integrate SysML with Simulink and to evaluate parametric diagrams. Next we showed systems engineering processes for designing a DCT using our tools. Users can design a control system using SysML and perform systems engineering processes such as requirement analysis on Rhapsody. They can evaluate relationship among system parameters by using PCE. They can define controller's behavior using SysML state machine diagrams and define plant's behavior using Simulink. They can generate a Simulink model from a SysML model, so they can simulate the model using Simulink. In addition, users can calculate system parameters by evaluating parametric diagrams and use the calculated parameter values in Simulink models. Finally we showed how to interact with users while simulating using a panel diagram.

IBM, the IBM logo, ibm.com, Rational, and Rhapsody are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/us/en/copytrade.shtml>.

References

- [1] OMG, *OMG Systems Modeling Language Version 1.2 (OMG SysML)*, 2010, <http://www.omg.org/spec/SysML/1.2/>.
- [2] OMG, *OMG Unified Modeling Language (OMG UML): Superstructure, Version 2.3*, 2010, <http://www.omg.org/spec/UML/2.3/Superstructure/>.
- [3] OMG, *OMG Unified Modeling Language (OMG UML): Infrastructure, Version 2.3*, 2010, <http://www.omg.org/spec/UML/2.3/Infrastructure/>.
- [4] E. Palachi and T. Sakairi, "Leveraging SysML parametric diagrams to perform trade studies and other quantitative analysis," *The Voice of the Systems*, vol. 9, pp. 18–29, 2012.
- [5] Modelica Association, *Modelica—A Unified Object-Oriented Language For Physical Systems Modeling, Language Specification Version 3.2*, 2010.
- [6] Y. Vanderperren and W. Dehaene, "From UML/SysML to Matlab/Simulink: current state and future perspectives," in *Proceedings of the Design, Automation and Test in Europe (DATE '06)*, p. 93, March 2006.
- [7] R. Kawahara, H. Nakamura, D. Dotan et al., "Verification of embedded system's specification using collaborative simulation of SysML and simulink models," in *Proceedings of the International Conference on Model-Based Systems Engineering (MBSE '09)*, pp. 21–28, March 2009.
- [8] A. Qamar, C. During, and J. Wikander, "Designing mechatronic systems, a model-based perspective, an attempt to achieve SysML-Matlab/Simulink model integration," in *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM '09)*, pp. 1306–1311, July 2009.
- [9] D. Phaoharuhansa and A. Shimada, "An approach to SysML and simulink based motion controller design for inverted pendulum robots," in *Proceedings of the 50th Annual Conference on Society of Instrument and Control Engineers (SICE '11)*, pp. 2190–2193, September 2011.
- [10] W. Schamai, P. Fritzson, C. Paredis, and A. Pop, "Towards unified system modeling and simulation with ModelicaML: modeling of executable behavior using graphical notations," in *Proceedings of the 7th Modelica Conference*, pp. 612–621, 2009.
- [11] C. Bock, "Systems engineering in the product lifecycle," *International Journal of Product Development*, vol. 2, no. 1-2, pp. 123–137, 2005.
- [12] R. S. Peak, R. M. Burkhart, S. A. Friedenthal, M. W. Wilson, M. Bajaj, and I. Kim, "Simulation-based design using SysML—part 1: a parametrics primer," in *Proceedings of the 17th Annual International Symposium of the International Council on Systems Engineering*, 2007.
- [13] M. Bone and R. Cloutier, "The current state of model based systems engineering: results from the OMG SysML request for information 2009," in *Proceedings of the 8th Conference on Systems Engineering Research*, 2010.
- [14] M. Bajaj, D. Zwemer, R. Peak, A. Phung, A. G. Scott, and M. Wilson, "SLIM: Collaborative model-based systems engineering workspace for next-generation complex systems," in *Proceedings of the IEEE Aerospace Conference (AERO '11)*, March 2011.
- [15] H. Naunheimer, B. Bertsche, J. Ryborz, and W. Novak, *Automotive Transmissions: Fundamentals, Selection, Design and Application*, Springer, 2nd edition, 2011.
- [16] M. Kulkarni, T. Shim, and Y. Zhang, "Shift dynamics and control of dual-clutch transmissions," *Mechanism and Machine Theory*, vol. 42, no. 2, pp. 168–182, 2007.
- [17] M. Lussier, "Using Matlab simulink with rational rhapsody," in *Proceedings of the IBM Rational Software Conference*, 2009.
- [18] X.-H. Liu and Y.-F. Cao, "Design of UAV flight control system virtual prototype using rhapsody and simulink," in *Proceedings of the International Conference on Computer Design and Applications (ICCD '10)*, vol. 3, pp. 34–38, June 2010.
- [19] B. Wang and J. S. Baras, "Integrated modeling and simulation framework for wireless sensor networks," in *Proceedings of the 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '12)*, 2012.
- [20] A. Garro and A. Tundis, "A model-based method for system reliability analysis," in *Proceedings of the Symposium on Theory of Modeling and Simulation—DEVS Integrative M&S Symposium*, 2012.
- [21] G. Bleakley, A. Whitfield, and A. Lapping, "Determining the right solution using SysML and model based systems engineering (MBSE) for trade studies," in *Proceedings of the 21st Annual International Symposium of the International Council on Systems Engineering*, 2011.