Model Generation for the Verification of Automatically Generated Mechatronic Control Software

Maarten Foeken, Mark Voskuijl, Andrés Álvarez Cabrera, and Michel van Tooren

***Abstract*—The development of embedded control software for mechatronic systems is mainly a non-automated process, requiring the intervention of a human programmer. A project has been started with the intention to develop a set of prototype tools and a framework with which an interdisciplinary product development team can automatically generate control software for mechatronic systems. This paper will discuss the development of a *Control Model Generator* as part of this project, which is envisioned to be able to generate system dynamics models at various levels of detail, which can be used to verify the automatically generated code at software level. Using SysML and Modelica, a model based view can be maintained throughout the model generation process.**

I. Introduction

Modern mechanical systems are in many cases multi-disciplinary products. Computers control industrial machines, information devices, aircraft, office equipments et cetera.

The development of such mechatronic products requires the collaboration of mechanical designers, electronic system engineers, aerodynamic engineers, and software engineers. A significant feature of software is that it is considered flexible and easy to modify. While the mechanical design has to be frozen at a certain point of the product development, software development is often requested for last-minute changes. Whereas software development for mechatronic systems in industry benefits from advances of tools and supporting systems, they still suffer from problems as:

1) Lack of integration: Tools are only integrated within their own domain, and not across domains, including mechanical, electronics, sequence control, and servo control design.

2) Lack of physics: Software development tools cannot reflect the physical world in which mechatronic systems are operating, as they only focus on the software domain using a black box approach for the physical system.

The authors gratefully acknowledge the support of the Dutch Innovation Oriented Research Program 'Integrated Product Creation and Realization (IOP-IPCR)' of the Dutch Ministry of Economic Affairs.

M. J. Foeken is Ph.D. researcher with the Faculty of Aerospace Engineering, Delft University of Technology, 2629HS Delft, The Netherlands (phone: +31 15 278 2088; e-mail: m.j.foeken@tudelft.nl).

M. Voskuijl is assistant professor with the Faculty of Aerospace Engineering, Delft University of Technology, 2629HS Delft, The Netherlands (e-mail: m.voskuijl@tudelft.nl).

A.A. Álvarez Cabrera is Ph.D. researcher with the Faculty of Mechanical, Maritime and Materials Engineering, Delft University of Technology, 2628CD Delft, The Netherlands (e-mail: a.a.alvarezcabrera@tudelft.nl)

M. J. L. van Tooren is professor with the Faculty of Aerospace Engineering, Delft University of Technology, 2629HS Delft, The Netherlands (e-mail: m.j.l.vantooren@tudelft.nl).

3) Irregular situations: Software development has to deal with irregular operation modes and abnormal situations, as well as regular modes like initialization, shutdown, maintenance and calibration.

4) Service functions: Software development also has to deal with service functions to support the operation of the system, like interface handling, communication and data collection.

5) Lack of verification: Verification of control software is poorly supported by current tools.

6) Lack of automation: As a result of the above mentioned problems, there is no automatic control software generation to support the software development process. The result is that the quality of the developed software cannot be guaranteed due to the intervention of the human programmer, and secondly, the process productivity is not high.

To attack these problems, a project named "Automatic Generation of Control Software for Mechatronic Systems" was started to develop a set of prototype tools and an integration framework with which an interdisciplinary product development team can automatically generate control software for mechatronic systems. Fig. 1 shows the framework with the set of eight tools that will be developed within the project, each represented as a white block.

The project envisions the use of a *Functional Model* as input to the control code generation process, specifying required functionality of the system being developed. The *Function Modeling* tool will create a formal representation of the functions, which will be used to generate the necessary behavior based on qualitative reasoning methods [1]. On the other hand, the function model will enable the *Mechatronic Feature Modeling* tool to generate the product definition by using mechatronic features, or function performers [2]. The output of these two latter tools is combined into the *Quantitative Behavior Generation* subsystem, which results in a formal description of the quantitative behavior of the system.

Next, this behavior description and the mechatronic feature model serve as an input for the control code and control model generation process, combined with data from the mechanical embodiment and electrical system design tools. Data from analysis tools such as finite element method or computational fluid dynamics (CFD) solvers could also be required. In Fig. 1, these existing commercial software tools, like e.g. CATIA, are represented by dashed-line blocks. Finally, the generated code can be verified at software and hardware level, using either the generated control models or the prototype hardware, respectively.

This paper will discuss the development of a *Control Model Generator*, as part of the framework depicted in Fig. 1. This tool will be able to generate system dynamics models at various levels of detail that can be used to verify the developed code at software level, before transferring the application to the actual hardware for machine based verification.

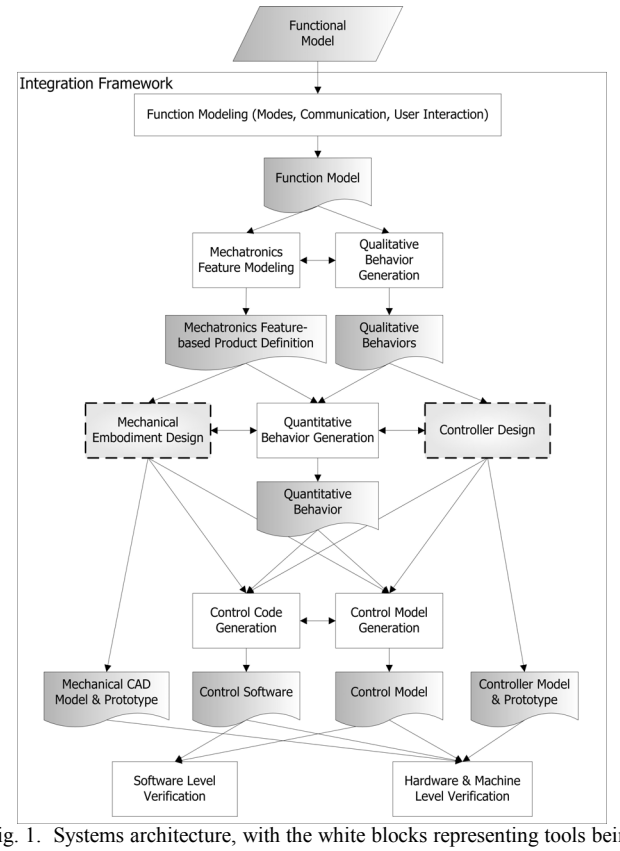

Fig. 1. Systems architecture, with the white blocks representing tools being developed in the project. Dashed-line blocks correspond to existing, commercial software tools.

The next section will introduce the methods and tools, followed by an overview of current work related to system model generation, keeping in mind the integration in the design process. Section IV will further elaborate on the applied methods by taking a look at an example test case.

II. Control Model Generation

Following the framework given in Fig. 1, the generation of a system dynamics model requires the integration of information from the mechanical, electronic, aerodynamic, and controller design domains, among others, into the product definition description, resulting in a single high-level system description. From this, system dynamics models at various levels-of-detail could be generated, taking into account different physical phenomena as required.

Before developing the transformation methods to come from high-level to dynamics model, suitable languages for both ends of the transformation have to be identified. The main challenge lies then in defining a modeling structure for both languages, taking into account that interfacing with other parts in the framework is required, but most of all in developing a generation/transformation method. The transformation process will require information from the high-level description, which will depend on the required level of detail, and which will have to be defined in one of the other preceding tools.

For this, the process of integrating information from various tools into a single model description is an important topic in the research project. An introduction to the approach of using a high level model of the system based on a functional description to represent the top-level conceptual hierarchy is given in [3].

The next subsections will describe the modeling languages and methods related to the model generation process.

A. High Level Description Language

As briefly discussed in the introduction, part of the input to the *Control Model Generator* is given as a quantitative behavior description. In view of the project, this will include a structural view on the system being designed, based on the mechatronic feature product definition, and as such a language able to describe the system structure is required.

For this, we have opted for the Systems Modeling Language (SysML) [4]. SysML is a domain specific customization, or profile, of the Unified Modeling Language (UML), and is a visual modeling language supporting the specification, analysis, design, verification and validation of a wide range of systems. UML was originally developed as a modeling language for software development [5], and as such it lacks features required for modeling other types of systems. For example, UML does not support continuous functions and flow between elements. On the other hand, the typical class-object relation one can find in software is not that common in other systems, and therefore in SysML the static element is called a block, and not a class. However, blocks can still have instances, such that the class-object concept is still supported, which is important in an object-oriented automatic generation process.

The so-called “four pillars” of SysML consist of modeling (a) system structure, (b) requirements, (c) behavior and (d) parametrics [4]. These four groups more or less correspond to the 3 major groups of diagram types that are available in SysML, as given in Fig. 2. The parametrics are incorporated in the structure diagram group.

For our approach, the main focus will be on the group of structure diagrams, which provide the capability to give multiple views on the system, using either the Block Definition Diagram (BDD) or the Internal Block Diagram (IBD) to represent system hierarchy or structure, respectively. With respect to UML, these structure diagrams offer more capabilities, e.g. due to the support of item flow in the IBD. The use of these diagrams is illustrated in Section IV.

For others goals of the research project, not only the system structure in terms of basic system elements and their interaction, but also the behavior description and the requirements could be modeled using SysML. The formal

definition of the model syntax and structure that will be used in the project is a major point of interest, as it influences the integration framework.

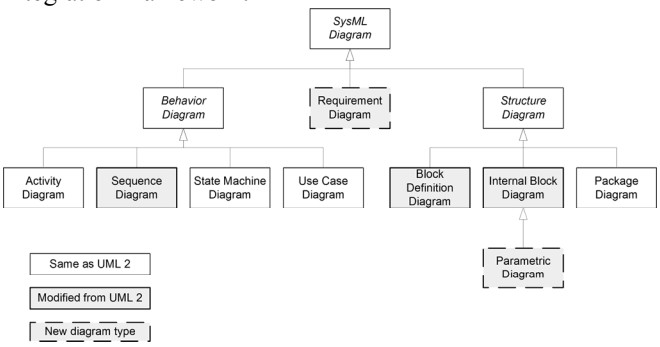

Fig. 2. SysML diagram types [4].

UML and SysML both support the use of the interoperability standard XMI, an XML-based format which can be used to transfer model data between tools. However, the interoperability between UML/SysML modeling tools is far from realized, due to differences in how the visual aspects of the diagrams are saved in XMI, resulting in diagrams that can only be viewed in a single tool. On the other hand, the model semantics are portable between most tools.

However, for our current research we are only interested in the interoperability between design tools in different domains, and not between various SysML tools. For that, an implementation resulting in a valid and well-formed XMI representation can be used to transfer model semantics by applying standard XML processing techniques.

B. System Dynamics Modeling

The de-facto standard for controller design and system dynamics modeling is Matlab/Simulink of The MathWorks [6]. However, after evaluating two modeling approaches, one purely based on mathematical equations, the other applying a more model-based view by partly using the SimMechanics toolbox, and assessing the possibilities of generating Simulink models with Matlab scripts for the test case, it was decided to also take a look at Modelica [7]. As Simulink is not a real modeling language, it is less suitable for applying object-oriented modeling concepts, like extendibility. Furthermore, the proprietary format prohibits gaining insight in the underlying code.

Although a model-based modeling approach is supported by some of the toolboxes, like SimMechanics, those only cover part of the physical domain, being mainly the mechanical (SimMechanics), electrical (SimElectronics) and hydraulical domain (SimHydraulics). However, domains that are not covered still require the system behavior to be modeled by mathematical equations, and as such a single modeling approach can not be supported.

The modeling language Modelica has been designed to model large, complex and hybrid physical systems and is based on differential and algebraic equations. It supports non-causal and object-oriented modeling techniques, and as such stimulates the reuse of modeling knowledge. Although the language is text-based, the models can also be presented to the user as schematic block diagrams, each block representing a (sub)system element.

To be able to simulate Modelica models a dedicated Modelica simulation tool is required, either commercial or free. This tool interprets the model, build-up from non-causal elements, and converts it to a full system of equations. Tools like Dymola [8] or MathModelica [9] also include a graphical editor to model the system using the schematic diagram representation.

C. System Dynamics Model Generation

The control model generation includes the transformation from system description to dynamics model. To be able to perform this transformation, a knowledge-based method will be developed, integrating knowledge about various aspects of the transformation process. This method for translating, or mapping, components from system description to system dynamics level will be the third focal point of this research.

After interpreting the high level model data, each system element, or combination of system elements must be translated to a corresponding system dynamics model element. Furthermore, the input/output types of system elements must be considered to prevent connecting incompatible elements. Note that this should also be checked at the system description level, where the flow between components is defined. Finally, the generation process should be able to setup tool dependent variables and environment parameters, requiring information about simulation tool specific options.

D. Integration

As the mechanical design will determine part of the behavior of the mechatronic system, it is necessary to incorporate data of the design in the system model, including information about mass, stiffness, and dimensions. Data from other design and analysis tools, like for example a electronics design tool or a CFD solver, can be included in a similar way.

The overall goal of the research project is to automatically generate and validate control software, which means that all parts of the framework should fit together in the end. The integration of information from various design domains into a single high-level representation is a major topic in this project, and will therefore be explored separately [3]. A first step to achieving this goal has been taken by using SysML as a base language for the system description, which enables the use of a common visual language for requirement, behavior and function descriptions. Furthermore, the XMI representation enables the exchange and manipulation of model data.

The integration with the generated control software as in Fig. 3 depends on the tools and platforms used for the controller development. In terms of existing software tools multiple options on both Linux and Windows platforms are available to integrate Modelica models with controller design tools as Matlab/Simulink and Scilab/Scicos [6], [8], [10]-[12].

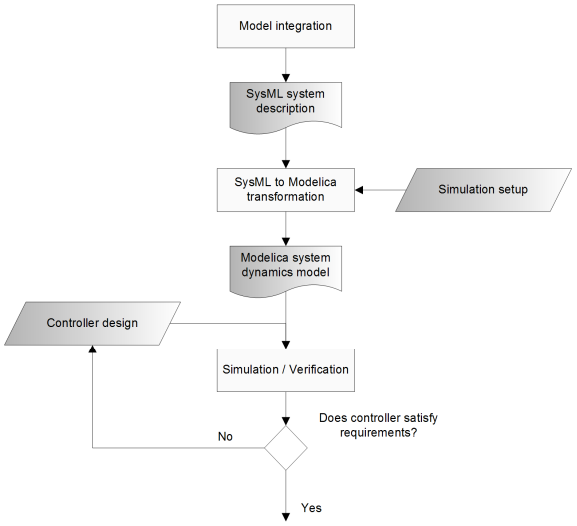

Fig. 3. Control model generation work flow and integration into the software generation framework.

III. LITERATURE

A. Automatic System Dynamics Model Generation

The generation of system dynamics models is being researched by focusing on various aspects of the automation process. Reference [13] introduces the concept of *Composable Objects*, combining form (CAD) and behavior into a single object. The interaction between components is port-based and reconfigurable, such that components at different levels of detail are interchangeable. The idea is used to automatically generate system dynamic equations from the linear graph representation [14].

The use of an ontology for ports for automatic model composition is being discussed in [15]. With the ontology one can represent and verify compatibility between the ports in a connection, and reason to select interaction models automatically. Often, these interaction models depend on the parameters of both subsystems involved. Reference [16] focuses on developing algorithms to assemble bond-graph representations of system components into a global bond-graph. This result is automatically processed to define the required level of detail, by taking into account the required dynamics, e.g. specified by the bandwidth. The problem of simplifying complex models is also discussed in [17], where the proposed semi-automatic abstraction method helps viewing systems at different levels of abstraction.

The application of knowledge engineering for the development of conceptual simulation models is discussed in [18], focusing on how to capture, represent and organize the knowledge required for conceptual simulation modeling.

A project aiming at applying artificial intelligence and knowledge engineering for supporting users engaged in modeling and simulation tasks, focusing on offering search services on top of a knowledge base supported by a domain ontology is presented in [19]. The knowledge captured in the knowledge base focuses on the modeling activity, and not on the models themselves.

Reference [20] discusses the use of knowledge based engineering methods to integrate discipline specific design and analysis tools. Using an object-oriented modeling approach, the use of "high level primitives" classes is introduced, enabling the automatic generation of aircraft configurations based on specific requirements. The generated model contains information about shape, structure, materials, mass, dimensions, and cost, which can subsequently be used by the integrated analysis tools.

B. SysML and System Dynamics Modeling

The use of SysML in combination with system dynamics modeling languages is gaining increased attention, resulting in various combined applications. Reference [21] presents a formal approach to modeling continuous system dynamics in SysML, by introducing a bi-directional mapping between SysML and Modelica. This concept is further explored in support of model-based system engineering in [22]. Part of the suggested approach links system-level models in SysML to components representing the system dynamics, but the representation of equations in SysML is also supported.

A different approach on the integration of SysML and Modelica has been introduced in [23], where a UML profile for Modelica based on the SysML profile, called ModelicaML, is proposed. Several diagram types of SysML are reused, while some are extended and new types are added. With this approach, a Modelica model can be graphically represented using the UML/SysML notation.

A summary on how UML and Matlab/Simulink can be associated with each other, and the future role of SysML for systems engineering in this association is given in [24]. The two options for coupling are either co-simulation or integration based on a common underlying executable language. The possibilities for converting one to the other are not being discussed.

Fig. 4. "Insight" quadrotor UAV, used as test case [25].

IV. APPLICATION TO A TEST CASE

To further explain the use of a *Control Model Generator* under the scope of the project framework, the "Insight" will be used as an example mechatronic application. Shown in Fig. 4, the "Insight" is a quadrotor UAV developed at the Faculty of Aerospace Engineering of TU Delft to perform indoors surveillance missions. The aircraft weighs 72 g, has a diameter of 30 cm and an endurance of 20 minutes while providing live streaming video [25].

The system description generated from the mechatronic

feature-based product definition and the quantitative behavior description gives a hierarchical and/or structural view on the system, to which extra mechanical design and analysis data is added. Fig. 5 shows a partial description of the quadrotor UAV, in which the hierarchy of system, subsystems and elements is defined. Parameters defining behavior obtained from other design and analysis tools integrated in the framework, like mass, inertia, lift and drag coefficients and relative positions, are attributes of the blocks, whereas input and output ports are also related to the block. Another view representing the system structure would define the connections between the basic system components, as in Fig. 6. In both figures, the electronics subsystem is not further detailed, but includes power supply, I/O ports, embedded computer components, and the control software among others.

For each of the basic quadrotor components, a mathematical equation describing the (dynamic) behavior can be derived. For the dc-motor – gearbox – rotor combination, the number of revolutions per minute of the engine (rpm), Ω_m, is a function of the input voltage u, and further depends on a set of 5 motor specific parameters (motor torque constant k_m, internal motor resistance R_m, propulsion system inertia J_t, and motor efficiency η), one gear specific parameter (gear ratio r), and one rotor related parameter (drag coefficient d) by [26]:

$$\dot{\Omega}_m = -\frac{1}{\tau}\Omega_m - \frac{d}{\eta r^3 J_t}\Omega_m^2 + \frac{1}{k_m \tau}u$$
$$\frac{1}{\tau} = \frac{k_m^2}{R_m J_t} \qquad (1)$$

For more detail, each of the elements can be further decomposed in more basic elements, each governed by their own set of equations.

The thrust T and torque Q generated by a single rotor can be expressed as a simple function of the engine rpm via:

$$T = b \cdot \Omega_m^2$$
$$Q = d \cdot \Omega_m^2 \qquad (2)$$

with b and d being the rotor thrust and drag coefficient, respectively, of which the values defined by the rotor blade geometry and air density can be calculated using an aerodynamics solver. However, if more detail is required these basic equations can be replaced by more elaborate versions, taking into account e.g. a non-constant air density and forward flight speed. In total, the equations of motion of the entire system are a function of the combined rotor forces and torques, gyroscopic effects, aerodynamic forces and gravity.

The entire set of equations required to simulate the dynamic behavior of the quadrotor is a combination of the behavior of the separate components making up the system, and as such shows that a component-based mapping is a viable model-based approach for generating system dynamics models. Of course, a direct, one-to-one mapping does not suffice for cases where elements need to be combined before any behavior can be generated, or where elements can relate to multiple physical domains. In those situations, the transformation method must be able to derive information about the system or element context.

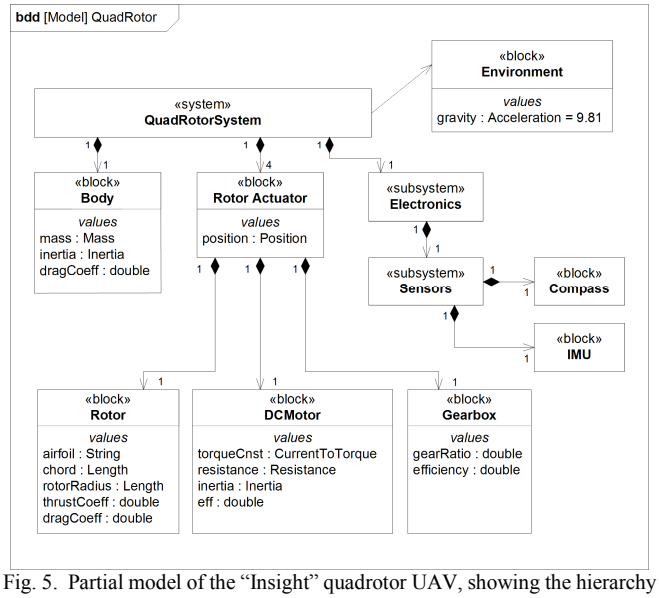

Fig. 5. Partial model of the "Insight" quadrotor UAV, showing the hierarchy of the system elements, using the SysML Block Definition Diagram.

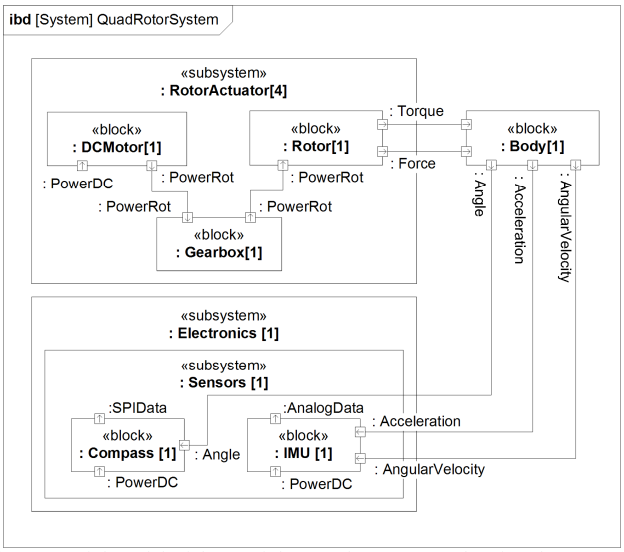

Fig. 6. Partial model of the "Insight" quadrotor UAV, showing the structure of the system, using the SysML Internal Block Diagram.

A challenge lies in having all element properties defined in the high-level description that are needed to generate the system dynamics model. For the test case, the dc-motor requires 6 parameters to fully define the behavior described by (1). Based on the available attribute data, the transformation should be able to identify the appropriate level of detail of the system dynamics equivalent, or return an error if parameters are missing.

V. Conclusion

Software based verification of control software for complex mechatronic systems requires complex system dynamics models. Under the scope of the "Automatic Generation of Control Software for Mechatronic Systems" project we have proposed the framework for a tool that can automatically generate control models from a high-level system description. Using SysML and Modelica as input and output languages for the generation process, respectively, a model-based view on the system can be maintained throughout this process. The high-level description will incorporate the product definition and data from design and analysis tools. The XMI format enables the use and exchange of the SysML model semantics using XML processing techniques.

The application of the approach to a quadrotor UAV shows that care must be taken when combining elements, not only in terms of interface compatibility but also in terms of parameter dependencies. It has also shown that with Matlab/Simulink and the included toolboxes it is not possible to use a single modeling approach for all system elements, making it less suitable to use in an automated model generation process.

Based on further test cases provided by the industrial partners in the project, the physical domains to be included in the modeling framework will be set. As a first step, we will define the structure for the high-level model description in SysML, deciding on what parts of the language will be used and defining new elements when necessary. After that, the transformation method to come from system description to system dynamics model will be explored.

Acknowledgment

M.J. Foeken thanks the Insight team for providing the data and models of the "Insight" quadrotor UAV.

References

[1] T. Kiriyama, T. Tomiyama, and H. Yoshikawa, "Qualitative reasoning in conceptual design with physical features," in *Recent Advances in Qualitative Physics*, Cambridge, Massachusetts, US: The MIT Press, 1993, pp. 375-386.

[2] I.F. Lutters-Weustink, D. Lutters, and F.J.A.M. van Houten, "Mechatronic features in product modeling, the link between geometric and functional modeling?," in *Proc. International Conference on Competitive Manufacturing (COMA'04)*, Stellenbosch, South Africa, 2004, pp. 125-130.

[3] A.A. Álvarez Cabrera, M.S. Erden, M.J. Foeken, and T. Tomiyama, "On high-level model integration for mechatronic systems control design," in *Proc. 2008 IEEE/ASME Intern. Conf. on Mechatronic and Embedded Systems and Applications*, Beijing, China, October 2008.

[4] Object Management Group, (2007, September). OMG Systems Modeling Language [Online]. Available: http://www.omgsysml.org

[5] Object Management Group (2007, November). OMG Unified Modeling Language [Online]. Available: http://www.uml.org

[6] The MathWorks, (2008, May). MATLAB and Simulink. [Online]. Available: http://www.mathworks.com

[7] Modelica Association (2008, May). Modelica and the Modelica Association. [Online]. Available: http://www.modelica.org

[8] Dynasim A.B., (2008, May). Dymola – Dynamic Modeling Laboratory. [Online]. Available: http://www.dynasim.se/index.htm

[9] MathCore, (2008, May). MathModelica System Designer. [Online]. Available: http://www.mathcore.com/products/mathmodelica

[10] Scicos (2008, May). Scicos Homepage. [Online]. Available: http://www.scicos.org

[11] Scilab (2008, May). Scilab Home Page. [Online]. Available: http://www.scilab.org

[12] M. Naja and R. Nikoukhah, "Modeling and simulation of differential equations in Scicos," in *Proc. 5th International Modelica Conference 2006*, Vienna, Austria, September 2006.

[13] C.J.J. Paredis, A. Diaz-Calderon, R. Sinha, and P.K. Khosla, "Composable models for simulation-based design," in *Engineering with Computers*, vol. 17, 2001, pp. 112-128.

[14] A. Diaz-Calderon, C.J.J. Paredis, and P.K. Khosla, "Automatic generation of system-level dynamic equations for mechatronic systems," Tech. Report, 1999.

[15] V.-C. Liang and C.J.J. Paredis, "A port ontology for automated model composition," in *Proc. 2003 Winter Simulation Conference*, 2003, pp. 613-622.

[16] J.L. Stein and L.S. Louca, "A component based modeling approach for system design: theory and implementation," in *Proc. Int. Conf. on Bond Graph Modeling and Simulation*, Las-Vegas, Nevada, USA, January 1995.

[17] K. Lee and P.A. Fishwick, "Semiautomated method for dynamic model abstraction," in *Enabling Technology for Simulation Science*, vol. 3038, June 1997, pp. 31-41.

[18] M. Zhou, Y.J. Son, and Z. Chen, "Knowledge representation for conceptual simulation modeling," in *Proc. 2004 Winter Simulation Conference*, 2003, pp. 450-458.

[19] M. Sevçenko, "Knowledge support for modeling and simulation," in *Proc. 7th International Conf. on Knowledge-Based Intelligent Information and Engineering Systems*, Oxford, UK, September 2003, pp. 99-103.

[20] G. La Rocca and M.J.L. van Tooren, "Enabling distributed multi-disciplinary design of complex products: a knowledge based engineering approach," *Journal of Design Research*, no. 3, vol. 5, 2007.

[21] T.A. Johnson, C.J.J. Paredis, J.M. Jobe, and R. Burkhart, "Modeling continuous system dynamics in SysML," in *Proc. 2007 International Mech. Eng. Congress and Exposition*, Seattle, Washington, USA, November 2007.

[22] T.A. Johnson, C.J.J. Paredis, and R. Burkhart, "Integrating models and simulation of continuous dynamics into SysML," in *Proc. 6th International Modelica Conference*, Bielefeld, Germany, March 2008.

[23] A. Pop, D. Akhvlediani, and P. Fritzson, "Towards unified systems modeling with the ModelicaML UML profile," in *Proc. International Workshop on Equation-Based Object-Oriented Languages and Tools*, Berlin, Germany, Linköping University Electronic Press, 2007.

[24] Y. Vanderperren and W. Dehaene, "From UML/SysML to Matlab/Simulink: current state and future perspectives," in *Proc. Design, Automation and Test in Europe (DATE)*, Munich, Germany, March 2006.

[25] Insight Team, "The Insight," Faculty of Aerospace Engineering, Delft University of Technology, Delft, The Netherlands, D.S.E. final report, 2007.

[26] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and Control of an Indoor Micro Quadrotor," in *Proc. IEEE International Conference on Robotics and Automation*, vol. 5, 2004, pp. 4393-4398.