Design Specification of Cyber-Physical Systems: Towards a Domain-Specific Modeling Language based on Simulink, Eclipse Modeling Framework, and Giotto

Muhammad Umer Tariq, Jacques Florence, and Marilyn Wolf

Georgia Institute of Technology,
Atlanta, USA
{m.umer.tariq,jacques.florence,marilyn.wolf}@gatech.edu

Abstract. In this paper, we propose a domain-specific modeling language for specifying the design of cyber-physical systems. The proposed domain-specific modeling language can capture the control, computing, and communication aspects of a cyber-physical system design in an integrated manner. The concrete syntax of the proposed domain-specific modeling language has been implemented as an extension of standard blocks available in Simulink. The meta-model for the proposed domain-specific modeling language has been defined using the Ecore meta-modeling language, which was originally developed as a part of Eclipse Modeling Framework project. We have also implemented an initial version of a parser that converts Simulink models, employing the proposed domain-specific modeling language, into corresponding Eclipse Modeling Framework instance models, which can then serve as input to model transformation tools. The proposed domain-specific modeling language builds on the concepts introduced by Giotto programming paradigm for platform-independent specification of control law, implemented by the controller in a cyber-physical system.

Keywords: Model-driven development, cyber-physical systems, domain-specific modeling language

1 Introduction

Embedded control systems consist of a physical plant being controlled by a feedback control algorithm, executing on an embedded computing platform. Typical examples of embedded control systems include automotive and avionics. Development of embedded control systems typically involves the following steps: requirements specification, control design and analysis, embedded design and analysis, and implementation. Model-driven development paradigm has been

successfully applied to the field of embedded control systems through the definition of appropriate domain-specific modeling languages (DSMLs) for different development steps of embedded control systems and appropriate model transformations among these DSMLs. Figure 1 shows the DSML and analysis tools used for each development step of embedded control system.

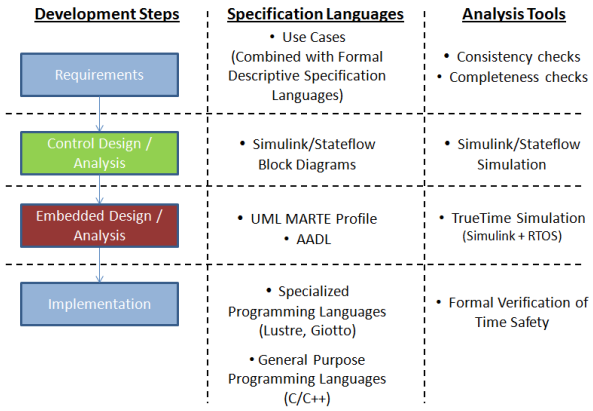

Fig. 1. Embedded control systems: development steps, specification languages, and analysis tools.

Last couple of decades have witnessed a dramatic decrease in the cost of sensing, communication, and computation technologies. This phenomenon has enabled the development of a new breed of embedded control systems that are much larger in scale, resulting in non-guaranteed wide-area communication subsystems. Moreover, these systems have longer lifespans and are more open in nature, resulting in the need for online reconfiguration and maintenance. Some examples of this new breed of embedded control systems are smart electric grids, smart irrigation networks, and vehicular networks. Design techniques and tools available in the domain of traditional embedded control systems cannot be directly applied to this new breed of embedded control systems. The field of cyber-physical systems aims to address these issues by rethinking the abstractions involved in the development of embedded control systems. In particular, the field of cyber-physical systems emphasizes the co-design of control, communication, and computation aspects of a system. As a result, the development steps for cyber-physical systems are requirements specification, cyber-physical system design (incorporating the co-design of control and computation), and implementation.

Figure 2 shows the development steps of a cyber-physical system (CPS) and analysis tools required at each of these steps. However, in order to develop a cyber-physical system according to model-driven development paradigm, the DSMLs used to specify the design of embedded control systems are not sufficient. An appropriate DSML for CPS design specification must be able to capture the

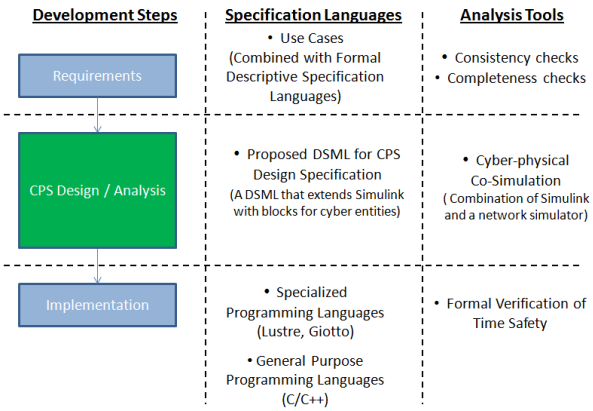

Fig. 2. Cyber-physical systems: development steps, specification languages, and analysis tools.

control, communication, and computation aspects of a CPS design. Moreover, an appropriate DSML for CPS design specification must independently convey the characteristics of a CPS computing platform as well as the control law implemented on that computing platform. The meta-model for this DSML can then be used to define model transformations that convert a CPS design specification into appropriate input for analysis tools (such as simulation and formal verification tools) as well as executable code for implementation platforms.

In this paper, we propose a DSML for CPS design specification. The proposed DSML is capable of capturing the control, communication, and computation aspects of a CPS design, as it provides blocks to represent physical system, cyber system, and cyber-physical interface involved in a cyber-physical system. The proposed DSML builds on the concepts introduced by Giotto programming paradigm [5] in order to specify the control law, executed by the embedded controller of CPS, in a platform-independent manner. Since Simulink (combined with auxiliary Stateflow and Simscape blocks) has become a defacto standard in the domain of embedded control system, we chose to implement the concrete syntax of the proposed DSML as an extension of standard blocks available in Simulink. The abstract syntax of the proposed DSML has been defined using an Ecore-based meta-model. Ecore meta-modeling language was originally developed as a part of Eclipse Modeling Framework (EMF) project. We have also implemented an initial version of a parser that converts Simulink models, employing the proposed DSML, into corresponding EMF instance models, which can then serve as input to model transformation tools. Figure 3 shows the envisioned model-driven development toolset for CPS based on the proposed DSML.

The rest of the paper is organized as follows. In Section 2, we present a brief review of the component technologies involved in the development of DSML, proposed in this paper. In Section 3, we present the details of the proposed

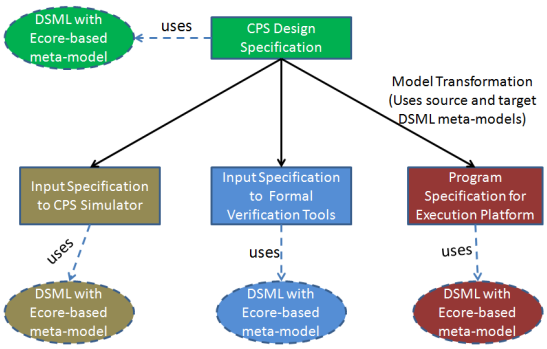

Fig. 3. Role of proposed DSML in the envisioned model-driven toolset for cyber-physical systems.

DSML for CPS design specification. In Section 4, we outline some related work. In Section 5, we present the conclusion.

2 Component Technologies

In this section, we briefly review the technologies involved in the definition and implementation of the proposed DSML for CPS design specification.

2.1 Simulink

Simulink, developed by MathWorks, is a simulation and model-based design tool that provides a graphical editor for specifying a model as a set of hierarchical block diagrams. Simulink is often used in conjunction with some auxiliary tools that provide specialized types of blocks to be used in Simulink block diagram. Two important examples of such auxiliary tools are Stateflow and Simscape. Stateflow allows the users to model decision logic based on the state machine and flow chart formalisms. Simscape provides fundamental building blocks from various domains (such as electrical, mechanical, and hydraulic) that can be combined to model a physical plant.

Simulink has become a defacto standard in the field of embedded control systems. It is not only used to simulate and analyze the proposed designs, but various model transformations have been defined to generate executable code from Simulink models for various embedded platforms. Therefore, we have chosen to develop the concrete syntax of the proposed DSML for CPS design specification as an extension of the standard blocks available in Simulink toolset.

2.2 Eclipse Modeling Framework

Eclipse is an open-source software project, aimed at providing a platform that can be reused for the development of integrated development environments

(IDEs). Eclipse is divided into numerous top-level projects such as Eclipse Project, Modeling Project, Tools Project, and Technology Project [1]. Eclipse Project is the core project that provides a generic framework for tools development and a Java IDE, built using this generic tools development framework. The Modeling Project has served as the focal point for the implementation of model-driven development technologies under the Eclipse project [2].

At the core of the Eclipse Modeling Project is Eclipse Modeling Framework (EMF) and all the other model-driven development technologies (such as model-to-model transformations and model-to-text transformations) have been built on top of EMF. At it core, EMF is a framework for defining a model and generating Java code from that model. As a part of EMF, Ecore modeling language has been defined that can be used to specify the model from which EMF generates a set of Java classes and interfaces. However, in the model-driven development context, an Ecore model can also serve as the meta-model of a DSML. We have used an Ecore-based meta-model to specify the abstract syntax of our proposed DSML for CPS design specification.

2.3 Giotto

Typical development process for an embedded control system requires a collaboration between control engineer and software engineer. First, a control engineer models the physical plant, derives the feedback control law, and validates the controller design through mathematical analysis and simulation. Then, a software engineer decomposes the computational activities of a feedback controller into time-constrained software tasks, develops code for these tasks in a traditional programming language (such as C), and assigns priorities to these tasks so that they could meet their timing constraints while being scheduled by an RTOS scheduler. Giotto is a specialized programming language for embedded control systems that aims to bridge the communication gap between a control engineer and a software engineer by providing an intermediate level of abstraction between control design and software implementation [5].

Giotto program provides a platform-independent description of feedback controller design in terms of time triggered sensor readings, actuator updates, task invocations, and mode transitions. Then, a Giotto compiler compiles the Giotto program onto a specific computing platform, preserving the functionality as well as the timing behavior, specified by Giotto program. Any CPS design specification must independently convey the computing platform characteristics as well as the controller design that needs to be implemented on that computing platform so that this design specification can then serve as an input to appropriate CPS analysis tools. Therefore, our proposed DSML builds on the concepts introduced by Giotto programming paradigm for platform-independent specification of the control law, implemented by the controller in a cyber-physical system. Figure 4 reviews the major concepts involved in a Giotto program [5].

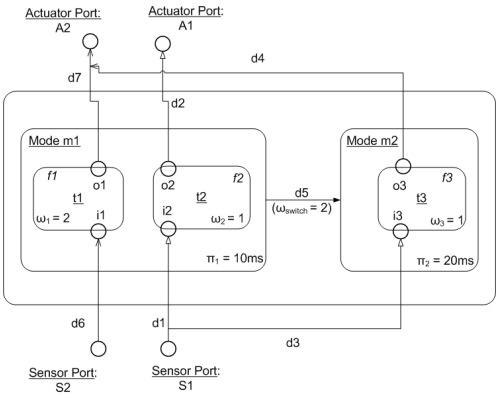

Fig. 4. Major programming elements of Giotto programming language [5].

3 Proposed Domain-Specific Modeling Language

In this section, we present the details of the proposed DSML for CPS design specification. The individual elements of the proposed DSML can be divided into three categories: *physical system blocks*, *cyber system blocks*, and *cyber-physical interface blocks*. *CompoundPhysicalPlant*, *AtomicPhysicalPlant*, *PhysicalSystemParameter* and *PhysicalLink* blocks belong to the *physical system blocks*. *Sensor* and *Actuator* blocks make up the *cyber-physical interface blocks*. *ComputingNode*, *NetworkLink*, *NetworkRouter*, *NodePlatform*, *NodeApplication*, *Mode*, *Task*, *ModeSwitchLogic*, *SensorPort*, *ActuatorPort*, *InputMsgPort*, and *OutputMsgPort* make up the *cyber system blocks*.

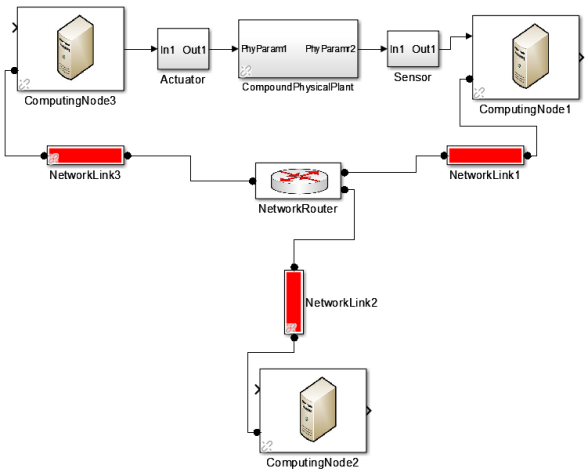

Fig. 5. A CPS design, specified as a Simulink model with the proposed DSML.

Physical system component of a CPS design can be specified by a set of *AtomicPhysicalPlant* blocks connected to each other through *PhysicalLink* blocks. *PhysicalLink* blocks represent "flow of energy" between components of a physical plant. A set of *AtomicPhysicalPlant* and *PhysicalLink* blocks can also be grouped together into a *CompuondPhysicalPlant* block. Moreover, *PhysicalSystemParameter* blocks are used to identify the elements of a physical plant that are to be sensed and actuated upon by the cyber system. Cyber-physical interface of a CPS design is captured by a set of *Sensor* and *Actuator* blocks. Each *Sensor* and *Actuator* block is connected to a *PhysicalSystemParameter* block.

Cyber aspects of a CPS design include the network topology of computing nodes, platform characteristics of each computing node, and the application software executing on each computing node. These aspects can be captured by connecting a set of *ComputingNode*, *NetworkRouter*, and *NetworkLink* blocks. Each *ComputingNode* block must include a *NodePlatform* and *NodeApplication* block. *NodePlatform* block in turn includes *Processor*, *RTOS*, *Middleware*, and *ApplicationFramework* blocks, which allow it the capture the platform characteristics of a computing node. The *NodeApplication* block includes a set of Giotto-inspired blocks that can specify the feedback control algorithm executing on that computing node in a platform independent manner.

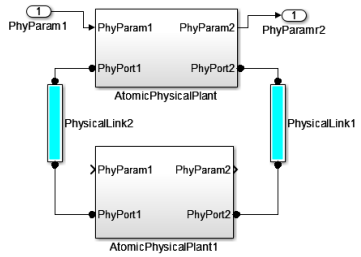

Fig. 6. Internal details of *CompoundPhysicalPlant* block in Figure 5.

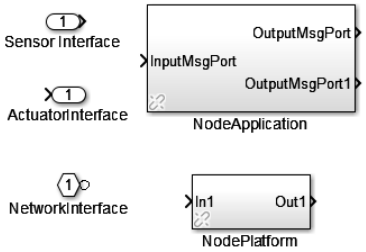

Fig. 7. Internal details of *ComputingNode* block in Figure 5.

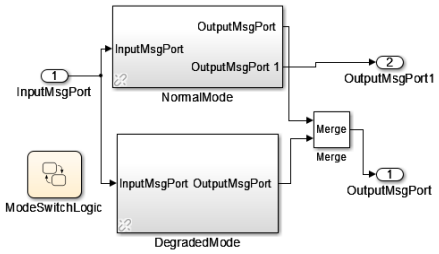

Fig. 8. Internal details of *NodeApplication* block in Figure 7.

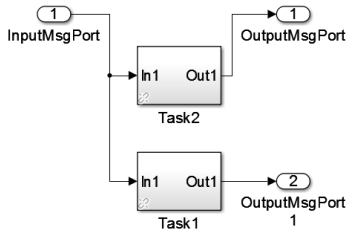

Fig. 9. Internal details of *Mode* block in Figure 8.

Concrete syntax of the proposed DSML has been implemented as an extension to standard blocks available in Simulink. In particular, a new Simulink library has been developed that provides a Simulink block for each of the proposed DSML element. We have also used Simulink's mask interface capability to provide each new Simulink block with a custom look, and a dialog box for entering element-specific parameters, such as the bandwidth and delay of *NetworkLink* block. Figure 5 shows a Simulink model that specifies a CPS design using the proposed DSML. Figure 6 shows the internal details of *CompoundPhysicalPlant* block. Figure 7 shows the internal details of a *ComputingNode* block, which contains a *NodeApplication* and a *NodePlatform* block. Figure 8 shows the internal details of *NodeApplication* block, which consists of a set of *Mode* blocks and a *ModeSwitchLogic* block. Figure 9 shows the internal details of *Mode* block, which contains a set of *Task* blocks. Figure 10 shows the simplified version of the Ecore-based meta-model for the proposed DSML.

4 Related Work

Integration of Simulink with Giotto has also been pursued in [6]. However, in this paper, we address a larger issue of developing an appropriate DSML for CPS design specification. Concepts introduced by Giotto are only used for one component of the proposed DSML (i.e. the platform-independent specification of a control algorithm executing on a computing node of CPS). Moreover, we also extend the concepts of traditional Giotto paradigm by introducing the concepts

of *InputMsgPort* and *OutputMsgPort* to enable mode transitions based on the violation of QoS constraints on the communication among computing nodes.

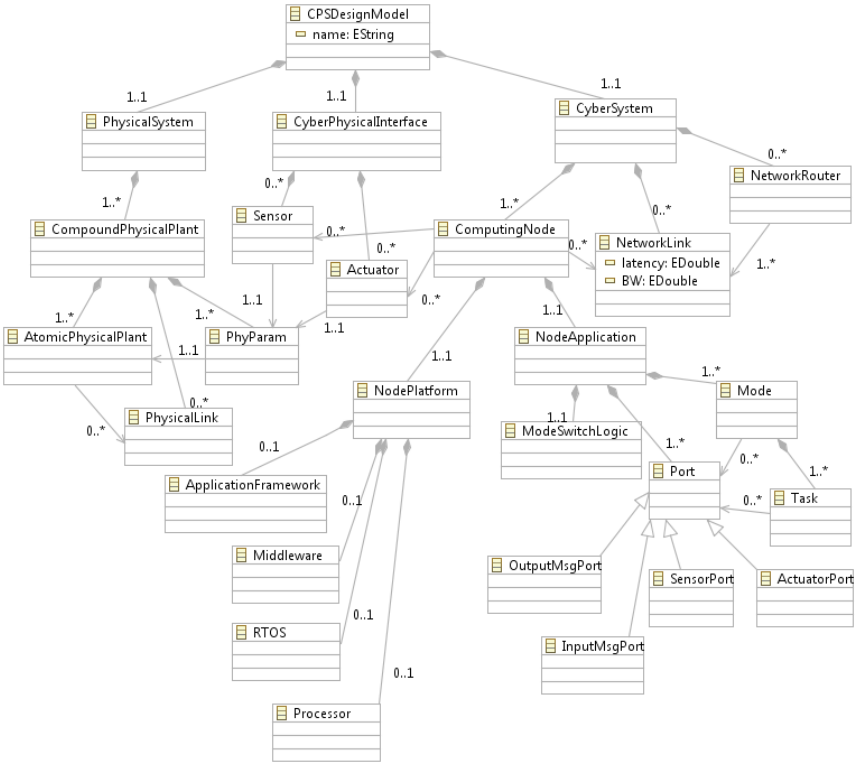

Fig. 10. Ecore-based meta-model of proposed DSML.

In [3], EMF-based integration of Simulink and EAST-ADL (a modeling language for automotive embedded systems) has been presented. The approach presented in [3] is targeted to the domain of automotive system. However, we propose to extend this approach by defining Ecore-based meta-model for specification languages required at each development step of a cyber-physical system. In this paper, we proposed a DSML for CPS design specification and developed its meta-model using Ecore.

Generic Modeling Environment (GME) is a popular domain-specific modeling environment [4]. GME supports its own meta-modeling language, MetaGME, and a graphical concrete syntax of DSMLs. The DSML, proposed in this paper, could have been implemented using GME as well. However, we chose to implement the proposed DSML using a combination of EMF and Simulink, because Simulink has become a defacto standard in the field of embedded control systems and the choice of EMF allows us to leverage all the model-based development

tools that have been developed on top of EMF. It must be pointed out that a transformation between MetaGME and Ecore has been reported in literature [7]. As a result, the DSML presented in this paper could be integrated with GME in the future.

5 Conclusion

In this paper, we have proposed a DSML for CPS design specification that is capable of capturing the control, computing, and communication aspects of a CPS design. The proposed DSML leverages the concepts of Giotto programming paradigm to ensure that the CPS design specification independently conveys the CPS computing platform characteristics as well as the control law that needs to be implemented on that computing platform. The abstract syntax of the proposed DSML has been defined through an Ecore-based meta-model. The concrete syntax of the proposed DSML has been implemented as an extension of standard blocks available in Simulink. We have also implemented an initial version of a parser that converts Simulink models, employing the proposed DSML, into corresponding EMF instance models, which can then serve as input to EMF-based model transformation tools.

The individual elements of the DSML, proposed in this paper, will be refined further in the future based on the experience obtained from different case studies. However, the proposed DSML with the choice of EMF for meta-modeling and Simulink for concrete syntax can serve as the basis of integrating various emerging facets of CPS research such as CPS simulation tools, CPS formal verification tools and CPS implementation platforms.

References

1. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional, Boston (2008)
2. Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language Toolkit. Addison-Wesley Professional, Boston (2009)
3. Biehl, M., Sjostedt, C.J., Torngren, M.: A Modular Tool Integration Approach - Experiences from Two Case Studies. In: 3rd Workshop on Model-Driven Tool and Process Integration at the European Conference on Modelling Foundations and Applications (2010)
4. Karsai, G., Sztipanovits, J., Ledeczi, A., Bapty, T.: Model-integrated Development of Embedded Software. Proceedings of the IEEE, 91(1), 145-164 (2003)
5. Henzinger, T.A., Horowitz, B., Kirsch, C.M.: Giotto: A Time-Triggered Language for Embedded Programming. Proceedings of the IEEE, 91(1), 84-99 (2003)
6. Pree, W., Stieglbauer, G., Templ, J.: Simulink Integration of Giotto/TDL. In: Automotive SoftwareConnected Services in Mobile Networks, pp. 137-154. Springer Berlin Heidelberg (2006)
7. Emerson, M., Sztipanovits, J.: Implementing a MOF-based Metamodeling Environment using Graph Transformations. In: Proceedings of OOPSLA Workshop on Domain-Specific Modeling, pp. 83-92 (2004)