A Prototype of Model-Based Design Tool and its Application in the Development Process of Electronic Control Unit

李荣清 周榕 李广鑫 和卫民 张晓谦 谷德权

1 Introduction

In the embedded world, physical systems always accompany computing systems. This demands that physical systems become an essential part of the system design and embedded software must operate concurrently with the physical systems. However, it puts forward a serious challenge in system design as the embedded systems grow much larger with complexity and heterogeneity today. Model-Based Design (MBD) is an efficient and effective way to develop those sophisticated embedded systems. Rather than relying on physical prototypes, MBD uses system models as executable specifications throughout the development. It allows system-level and component-level design and simulation, automatic code generation, and other necessary analysis by using particular tools. In particular, Simulink is a model-based development tool widely used in the industry and academia for the design of embedded system applications.

In order to manage the complexity and heterogeneity, the design approach should be based on the use of formal methods to describe the behavior of the system at a high level of abstraction. A model of computation (MoC) is a mathematical model, which defines how computation takes place in the system, and the MoC gives semantics to the behaviors of each model component. Ptolemy II is an infrastructure for experimenting with concurrent real-time embedded system design techniques. It provides an environment for modeling, simulation, and design of systems through the assembly of concurrent components, allowing a heterogeneous mixture of different models of computation. It also focuses on visual, module-oriented programming with an emphasis on multiple component interaction semantics. The executable model is precisely controlled via so-called directors, which are MoCs in nature. Ptolemy II supports some commonly used MoCs for modeling embedded systems including Synchronous Dataflow (SDF), Discrete Event (DE), Continuous Time (CT), and Finite State Machine (FSM).

There are several modeling languages that have appeared
in the past ten years with more or less success. Each model-based software tool is constructed for some specific objectives and has its own Domain Specific Modeling Language (DSML) [12]. Each DSML offers designers modeling concepts and notations that are tailored to characteristics of their application domains. The Generic Modeling Environment (GME) [13] is a Model-Integrated Computing (MIC) [14] based configurable tool for creating domain specific modeling, and performing model analysis and transformation. The configuration is achieved by using metamodels to specify the abstract and concrete syntax of a modeling paradigm, or to define DSML. The GME metamodel is based on Unified Modeling Language (UML) class diagrams, and Object Constraint Language (OCL) can be used to specify constraints. The modeling paradigm contains all the abstract and concrete syntax information of the domain, including which concepts will be used to construct models, what relationships may exist among those concepts, how the concepts may be organized and viewed by the modeler, and what rules govern the construction of models. GME also provides C++ and Java interfaces for writing plug-in components to traverse, manipulate, and interpret models.

In the GME framework, a design tool named MoDAL is developed at SIAT CAS to enable the design of heterogeneous systems in a hierarchical environment. MoDAL is designed for forming a toolchain among Simulink, Ptolemy II and ReachLab [15] to perform system-level model design, integration and analysis. By hierarchically organizing MoC for defining the interactions, models can be made heterogeneous by having them designed in MoDAL or imported from Simulink. The model file can be exported as a MoML [16] file, which is an XML file used in Ptolemy II.

Several attempts have been made in the industry and academia for capturing Simulink models for performing system-level model integration and analysis. In [17], a mapping between Simulink models and UML models is investigated. Model transformation is used in [18] for transforming Simulink/Stateflow models into hybrid automata models.

The rest of this paper is organized as follows: in section II the background of the paper is introduced. The design tool MoDAL is presented in section III. In section IV an Electronic Control Unit (ECU) model is used to show how MoDAL can be used in the development process. Model-in-the-loop and software-in-the-loop simulations of the ECU model are shown in section V. The conclusions and future works are presented in section VI.

2 Background

MIC is a framework designed to meet essential needs of embedded software development. MIC focuses on the formal representation, composition, analysis, and manipulation of models during the design process. MIC places models in the center of the entire life-cycle of systems, including specification, design, development, verification, integration, and maintenance. In MIC, in order to meet the specific needs of an application domain, a Domain Specific Modeling Languages (DSML) can be defined for capturing the concepts, relationships, integrity constraints, and semantics of the application domain and allows users to program declaratively through model construction. Formally, a DSML can be defined as a 5-tuple.

Definition 2.1 (Domain Specific Modeling Language, DSML) A DSML is a 5-tuple \( L = \langle A, C, C, MS, MC \rangle \), where \( A \) is an abstract syntax, defining the concepts, relations, model-composition principles, and integrity constraints of domain modeling languages; \( C \) is a concrete syntax, defining the specific notations (graphical, textual, or mixed) used to describe models; \( S \) is a semantic domain, usually defined in some formal, mathematical framework, in terms of which the meaning of the models is explained; \( M_C: A \rightarrow S \) is a semantic mapping, which relates syntactic concepts to those of the semantic domain; \( M_M: C \rightarrow A \) is a syntactic mapping, which assigns syntactic constructions to elements in the abstract syntax.

GME is a MIC-based configurable toolkit for creating domain specific modeling and program synthesis environments. In GME, the language used for defining components of a DSML is called metalanguage and the formal specification of a DSML is called metamodel. The specification of the abstract syntax \( A \) of a DSML requires a metalanguage that can express concepts, relationships, and integrity constraints. The concrete syntax \( C \) is the model itself, and it is represented as a graph. The syntactic mapping \( M_M \) describes the mapping from concrete syntax to abstract syntax, that is, the abstract syntax defines all the elements that will be presented in the graphic model. The specification of semantic domain \( S \) and the semantic mapping \( M_S \) is more complicated, because models might have different interpretations; therefore a DSML might have several semantic domains and semantic mappings associated with it. For example, the structural semantics of a modeling language describes the meaning of the models in terms of the structure of model instances: all the possible sets of components and their relationships, which are consistent with rules, are defined by the abstract syntax. Accordingly, the semantic domain for structural semantics is defined by a set-valued semantics. The behavioral semantics may describe the evolution of the state of the modeled artifact along some time models. Hence, the behavioral semantics is formally captured by a mathematical framework representing the appropriate form of dynamics.

3 Modal

In order to enable designers to model embedded systems with a rich set of MoCs supported by Ptolemy II while
having the metamodeling architecture implemented in GME for supporting model transformation and interpretation, a GME-based tool called MoDAL is developed. In this section, we introduce classes defined in Ptolemy II [19]. By using these classes, MoDAL is constructed as a DSML for constructing heterogeneous system models.

A. Classes in Ptolemy II

Ptolemy II is a tool for designing concurrent real-time embedded systems. It provides an environment for modeling, simulation, and design of systems through the assembly of concurrent components, allowing a heterogeneous mixture of different models of computation. It also focuses on visual, module-oriented programming with an emphasis on multiple component interaction semantics.

In Figure 1, some of the key classes in Ptolemy II are shown as a UML static structure diagram. They define the Ptolemy II abstract syntax and abstract semantics. Most of the classes generalize NamedObj, which in addition to being nameable can have a list of attributes associated with it. Attributes themselves are instances of NamedObj. Entity, Port, and Relation are three key classes that extend NamedObj. These classes define the primitives of the abstract syntax supported by Ptolemy II. ComponentPort, ComponentRelation, and ComponentEntity extend these classes by adding support for clustered graphs. CompositeEntity extends ComponentEntity and represents an aggregation of instances of ComponentEntity and ComponentRelation. The Executable interface defines objects that can be executed. The Actor interface extends this with the capability of transporting data through ports. AtomicActor and CompositeActor are concrete classes that implement this interface. The Executable and Actor interfaces are key to the Ptolemy II abstract semantics. An executable Ptolemy II model consists of a top level CompositeActor with an instance of Director and an instance of Manager associated with it. The manager provides overall control of the execution. The director implements a semantics of a model of computation to govern the execution of actors contained by the CompositeActor.

Director is the base class for directors that implement models of computation. Each such director is associated with a domain. In Ptolemy II, there are directors that implement continuous-time modeling, process networks, synchronous dataflow, discrete-event modeling, communicating sequential processes and finite state machines.

B. Construction of MoDAL

The metamodel of MoDAL is based on some key classes of Ptolemy II. The class diagram of MoDAL metamodel implemented in GME is shown in Figure 2.

In Ptolemy II, a hybrid automaton can be modeled by using ModalModel as the top level, which contains an MoC block named CT, and other entities representing the ordinary differential equations for describing the dynamics. Inside the ModalModel, an FSM is represented as the second level by using States and Transitions. A Transition has two attributes, that is, guardExpression and setAction, which correspond to the guard and reset used in hybrid automata. A State contains
several components to depict the continuous dynamics of a hybrid automaton, such as Expression, Integrator. In particular, an attribute named RefinementClass of State describes the MoC used in the refinement of this state.

In GME, the concept of First Class Object (FCO) is introduced to enable objects that are inherently different to be able to inherit from a common base class. Modeling concepts, including Model, Atom, Set, Connection and Reference, are defined as FCO. As shown in Figure 2, the top level of a MoDAL model is a CompositeEntity, which is a type of Model in GME. A CompositeEntity is similar to the CompositeActor in Ptolemy II and its main purpose is to provide a hierarchical organization of models. It contains MoCs, AtomicActor, Port, DirectLink, IndirectLink, TypedIORelation. It can also contain another CompositeEntity. Each CompositeEntity in MoDAL contains an MoC, like the director in Ptolemy II, governing the execution and interactions of components within the CompositeEntity.

A ModalModel, which inherits from CompositeEntity, is also a type of Model in GME. The ModalModel is a special composite actor which is used to represent a hybrid automaton. States and Transitions only can be contained in ModalModel which could represent the discrete modes and discrete jumps of a hybrid automaton. Another entity named AtomicActor is also a type of Model in GME. The AtomicActor serves as a primitive functional unit for models. It cannot be directly used for modeling because it is declared as an abstract class. The extending concrete classes, including Integrator, Sampler, TimedDelay, ZeroOrderHold, and Expression, which are used to implement the actual operations. The composition of different actors in composite actors could represent different signal processes in different systems. Currently, there are a few actors implemented in MoDAL metamodel which can manipulate the continuous or discrete signals. Other actors could be added into MoDAL metamodel if needed.

MoDAL metamodel contains Port and Relation. Port is a type of Atom in GME and is declared as an abstract class, the extending classes named input and output can be directly used for modeling. Input and output ports are used to represent the signal interfaces for composite actors and atomic actors. Relation, a type of FCO in GME, is used to connect the signal interfaces between different composite actors or different atomic actors. There are two kinds of connections between ports and relations, DirectLink and IndirectLink. The DirectLink can only connect ports at the same level while the IndirectLink can connect ports across different levels of hierarchies through TypeIORelation. A special implementation of Relation called Transition is used for representing the discrete jumps between states in ModalModel.

Furthermore, GME provides a mechanism to impose constraints on system models by using OCL. The constraints can be imposed on FCO including Model, Atom, Set, Connection and Reference. In order to perform static analysis of models in MoDAL, a set of constraints is defined as shown in Figure 3. For example, in each level of a system model, the number of MoC must be one and only one, and we can use OCL to set this constraint on CompositeEntity as: let MoCNum = self.parts(MoCs) → size in(MoCNum == 1). Other constraints are specified in OCL according to the needs in the system design.

By interpreting the metamodel in GME, a hierarchical modeling environment MoDAL is constructed. Models can be constructed heterogeneously by having them designed in MoDAL. On the other hand, models of Simulink can be imported into MoDAL via an adapter. The interactions among models can be organized hierarchically by properly mixing various MoCs. In the
current version, MoDAL supports SDF, CT, and FSM. A translator for converting a MoDAL model into a MoML file for Ptolemy II is built upon C++ interfaces provided by GME.

4 Case Study

In this section, an Engine Timing with Closed Loop Control model from Simulink demo is used as an example to show how MoDAL can be used in the development process of Electronic Control Unit for enabling model representation and code generation.

A. The ECU Model

In the ECU model from Simulink demo, Triggered Subsystems are used to model engine timing including a feedback controller. As shown in Figure 4, the valve timing subsystem generates a signal that triggers the Controller and Compression subsystems twice per revolution of the crank (calculated from the crank speed). A speed set point (in rpm) from the Step block drives the simulation. The Controller calculates the necessary throttle angle based on the desired rpm and the crank speed is calculated by the vehicle dynamics. After running the simulation, the Scopes show the engine speed and the throttle angle compared to the load torque.

B. Work Flow

First, the subsystem shown in Figure 4 is imported into MoDAL with an adapter. Subsequent to importing the subsystem into MoDAL by using the adapter while preserving the information, the components and their interactions are shown in Figure 5. There are five components, corresponding to the five modules of the subsystem shown in Figure 4.

Different from the Simulink model, the ports are hidden in the components.

After obtaining the model in MoDAL environment, an interpreter is used to conduct transformation from MoDAL to Ptolemy II for this specific instant model. The transformed subsystem in Ptolemy II having all the information captured is shown in Figure 6.

Finally, the executable code segments are generated automatically under the guidance of the SDF Director in Ptolemy II. The code segments are shown in Figure 7.

Figure 5. The transformed subsystem in MoDAL

5 Ecu Simulation

There are sequential test stages to validate the correctness of the whole process. Simulation is the most primitive and fundamental means which can be applied to the generated code to demonstrate the effectiveness of the approach by using model-in-the-loop (MIL) and software-in-the-loop (SIL) simulations\[20\].

MIL simulation is the basic simulation to analyze the controller model along with the simulated system model. It runs on the host PC and captures the specified behaviors of the model. Since the computations are performed in PC with high precision, depending on precision of the floating point arithmetic and the word length of the machine, MIL simulations provide theoretical output values needed to control the plant. During the MIL execution, the model probably fail to execute, which indicates design faults in the model. This situation is likely to happen. Once the MIL test results are acceptable, system engineers will consider further
To perform SIL simulation, we can replace some parts (usually the controller) in the model with executable code which is generated by the code generator in the modeling and/or simulation environment. The simulation environment compiles the source code and runs the simulation on host PC. The results may differ from the MIL simulations, depending on the scheduling of the source code, the handling of numerical value and so on. SIL simulations mainly focus on software realization and the interaction between other system parts and the replaced part. This approach enables the detection of possible faults of syntax and semantics caused by the inadequate design choices during design time. It is important because software designers always seek to optimize the overall code size and the usage of storage space by selecting the minimum size that will satisfy the value ranges of the input and output variables estimated by the system engineers.

In this paper, SIL simulation is realized in Simulink for convenience by encapsulated the generated code in a S-function block, and the simulation result of SIL simulation is compared to that of MIL simulation. Comparison of the simulation results of MIL and SIL simulations is shown in Figure 8. The subfigures on the left side show the direct output of the controllers. In these subfigures, the dotted lines indicate the input values of load torque from the environment, and the solid lines indicate the output values of throttle degree, which are converted to commands driving the engine. After a sequence of transformation, the commands are finally turned into direct torque and have the engine run. The subfigures on the right side show the engine speeds of both systems. As shown in the subfigures, both MIL and SIL simulations are performed with the same result.

6 Conclusions

Embedded systems are demanded to be more sophisticated. In order to manage the complexity and heterogeneity of embedded systems, model-based design approach is widely used to raise the level of abstraction in the design process. Consequently, various tools have been designed for performing system-level modeling, integration, and analysis. Based on the concept of Model-Integrated Computing, we have developed MoDAL in the GME framework to enable the design of heterogeneous systems in a hierarchical environment. It can be used to form a toolchain among Simulink, Ptolemy II and some other model-based design tools. In MoDAL, models can be made heterogeneous by having them designed in MoDAL or imported from Simulink and the interactions among models can be organized hierarchically by properly mixing various Models of Computation. In the current version, MoDAL supports Synchronous Dataflow, Continuous Time, and Finite State Machine. The model file can be exported as a MoML file, which is an XML file used in Ptolemy II. In the case study, an Electronic Control Unit model is used to demonstrate how a model designed in Simulink can be imported into MoDAL and then exported to Ptolemy II for performing code generation. With the generated code from the model, model-in-the-loop simulation and software-in-the-loop simulation are performed with the same result. In the future, we will be working to show the correctness of the model transformation as well as the code generation mechanism by leveraging the existing theoretical results for some classes of Models of Computation.

7 Acknowledgements

This work is supported by the Ministry of Science and Technology, China, Grant No. 2009-EG120165 and the Shenzhen Institutes of Advanced Technology, the Chinese Academy of Sciences, Grant No. O9G003.
References


