Journal of Universal Computer Science, vol. 16, no. 17 (2010), 2415-2434
submitted: 15/2/10, accepted: 30/8/10, appeared: 1/9/10 © J.UCS

UML Behavior Models of Real-Time Embedded Software for Model-Driven Architecture

Jinhyun Kim and Jin-Young Choi
(Korea University, Seoul, S.Korea
{jhkim,choi}@formal.korea.ac.kr)

Inhye Kang
(University of Seoul, Seoul, S.Korea
inhye@uos.ac.kr)

Insup Lee
(University of Pennsylvania, Philadelphia, USA
lee@cis.upenn.edu)

Abstract: Model-Driven Architecture (MDA) presents a set of layered models to separate design concerns from platform concerns. The model executability for each model element is still challenging although MDA is currently able to cope with most syntactic and transformation definition issues. Moreover, the importance of rigorous specification and verification of the system is increasing, as the embedded software is more widely used for systems closely related to our life. Thus, this paper suggests behavior modeling views characterizing Platform-Independent Model (PIM) and Platform-Specific Model (PSM) behaviors and formal and verifiable models for them. In this, the PIM behavior is given from the view of the functionality of the software in Statecharts, whereas the PSM behavior is modeled from the view of a timed and resource-constrained behavior in TRoS, an extension of Statecharts in respect of time and resource constraints. Moreover, we provide an efficient way where PIM in Statecharts is transformed into PSM in TRoS. Using our approach, PIM and PSM behavior are captured in formal semantics for rigorous analysis in terms of system behavior, and the PSM behavior in TRoS is effectively and consistently obtained from the PIM behavior in Statecharts. We present a case study, in which safety-critical software for a railway control system is developed to show the feasibility of our approach.

Key Words: MDA, Behavior Models, Formal Models, Statecharts, Real-time Embedded Software

Category: D.2, D.2.1, D.2.10

1 Introduction

Model-Driven Development (MDD) is a development methodology that promotes a variety of models to reason a problem domain and design a solution in the solution domain[Beydeda and Volker 2005]. Model-Driven Architecture (MDA) supported by UML is widely discussed in the software industry, as a way to increase the quality, efficiency, and predictability of large-scale and high-quality software development[MDA]. One of the hallmarks of MDA is the notion that the world of modeling can be neatly separated into different areas of concern, typically called "separation of concerns." MDA introduces a specific set of modeling concerns, such as Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM), and Implementation Specific Model (ISM), and it defines rules for:

- Automating many steps needed to convert one model representation to another,
- Tracing between model elements,
- Analyzing important characteristics of the models.

Although MDA is currently able to cope with most syntactic and transformation definition issues, model executability is still challenging[Mohagheghi and Dehlen 2008]. One of the main obstacles is the lack of adequate models for the behavior of software and of mechanisms to integrate behavioral models with structural models and with other behavioral models[Riccobene and Scandurra 2009]. Furthermore, the transformation from one representation to another is another obstacle to be tackled for industrial adoption of MDA. In short, a current critical issue in MDA in terms of modeling software behaviors is to provide an effective specification and validation framework able to represent the meaning or semantics of behaviors of each modeling element, analyze each element of behavioral models thoroughly, and transform one model representation to another.

In this paper, we address the issues of providing executability to PIM and PSM models with Statecharts and one of its timely extensions, TRoS[Kim et al. 2010]. Our approach to PIM and PSM behavior models has the following goals:

- Provide behavior modeling views characterizing PIM and PSM behaviors,
- Provide an efficient way to transform PIM behaviors to PSM behaviors ,
- Support rigorous analysis methods for both the behavior models.

To achieve these goals, we propose here a functional behavior view independent from platform-related concerns as behavior modeling view for PSM and a time and resource-constrained functional behavior as behavior modeling view for PSM. The PIM and PSM behaviors are modeled in respectively Statecharts and TRoS because TRoS has the ability to capture the timed and resource-constrained behavior of the system and be transformed into Statecharts by annotating time and resource-related information to Statecharts. Moreover, they are all supported by formal and informal analysis tools, such as simulation, model checking and bi-simulation, to give formal proofs for the correctness of their behavior.

The advantages of our MDA approach are as follows: First, Statecharts and TRoS for PIM and PSM are formally defined in semantics to rigorously define and verify the real-time embedded software. Second, PIM in TRoS is easily obtained from Statecharts since by adding time and resource-related information to PSM in Statecharts. Third, the analysis for PIM and PSM behaviors is performed by various analyzing tools, such as model simulation and formal verifications for TRoS and Statecharts.

In this paper, the application of our approach to modeling behavior of the software is illustrated through modeling a real-time embedded software system, called Distance Control Module (DCM). DCM is a part of railway interlocking control system, a safety-critical system that needs to be proven correct in the safety. The system is first modeled in Statecharts, and then it is transformed into TRoS by a transformation procedure we propose here. Each model in Statecharts and TRoS was checked by simulation, model checking, and bi-simulation to verify the correctness of their modeled behavior. Consequently, we could find flaws inherent in the behavioral model of PIM and provide a formal proof for the correctness of the timed PSM behavior.

The remainder of this paper is organized as follows: First, we discuss related work in Section 2. In Section 3, behavior modeling views for PIM and PSM to capture the real-time embedded software in MDA will be suggested, and specification and analysis methods will be explained. We explain our experience of modeling and analyzing real-time embedded software system to illustrate our approach in Section 4. Finally, we will conclude this paper in Section 5.

2 Related Works

UML profiles for RTES (Real Time and Embedded Systems) and MARTE (Modeling and Analysis of Real-Time and Embedded systems) are proposed as foundations for the model-based description of real time and embedded systems[MARTE]. However, most UML models supported by MARTE cannot support the formality of specification for rigorous verification methods.

Until now, the research on a behavioral model for PSM still remains at its early stage. Recently, Riccobene et al. in [Riccobene and Scandurra 2009] provided a behavior modeling framework of MDA for PIM using State Machine [McNeile and Roubtsova 2009] that possesses all the characteristics of preciseness, abstraction, refinement, executability, and metamodel-based definition. However, their approach is limited to providing the executability to PIM structural models. Furthermore, as more safety or mission critical software such as automobile, nuclear power plant, and avionics, becomes increasingly prevalent, the more the use of precise semantics and rigorous analyzing methods, such as formal methods, is necessary. In the context of formal behavioral models for MDA, Snook et al. in [Snook and Butler 2006] introduced executability to structural models via UML-B, a profile of UML that bridges from class diagrams, informal UML structural models, to B, formal behavior specifications. However it did not take into account "separation concerns."

Statecharts is a widely used behavior modeling and specification language. Many extensions and variations of Statecharts have been proposed to compensate for the lack of expressiveness of Statecharts for timed behavior. Schulz et al. in [Schulz et al. 2000], and Holger and Sven in [Giese and Burmester 2003] proposed many other real-time extensions of Statecharts. Particularly, Eshuis in [Eshuis and Wieringa 2000, Eshuis et al. 2002] addressed the requirements-level Statecharts for UML to define the behavior of software from the view of functionality.

Timed automata[Alur and Dill 1994] and time petri-net[Merlin 1974] and so on were also proposed to represent and analyze the timed behavior of the system. Timed automata describes a time-constrained behavior of the system by specifying timing constraints on transitions and invariants on locations, supported by UPPAAL, etc., as a verification tool[Larsen and Pettersson 2000]. Time petri-net, an extension of petri-net, introduces the notion of temporal constraints, which restrict timely triggering of a transition. Those methods take into account a timing constraint for the system behavior, but not a resource constraint restricting the system behavior. Moreover, defining a resource-constrained behavior of software bound to a platform and transforming the functional behavior of the system into platform-concerned behavior of software through those description languages still remains unresolved. Meanwhile, the timing constraint presented in TRoS centers around a time-consuming action that is subject to the availability of shared resources, thus the problems that can be detected by TRoS used in our approach include a problem between the interaction between processes or executions, such as deadlock or shared resource contamination. Moreover, the transformation from Statecharts to TRoS is not difficult. Hence, we use Statecharts and TRoS to model the behavior of real-time embedded software that is restricted by the limitation of resources.

3 Our Approach

The main target system of our approach is real-time embedded software that is composed of application software and platform, such as real-time operating system. The scope of our approach is limited to the development and analysis of models for application software and platforms operating the application software. Figure 1 shows our approach to behavioral models of MDA for real-time

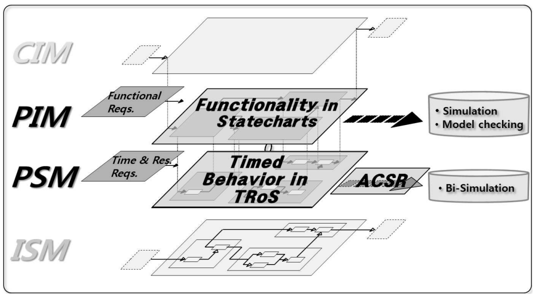


Figure 1: Our Approach to Behavior Models for MDA

embedded software. The behavior model of the software, we consider here, centers around PIM and PSM behaviors. First, CIM in Figure 1 defines the purpose and boundary of the target software. The focus of CIM is on basic software requirements under the environment, but not on the detail of the structure and processing of the system are hidden or yet undetermined. The separation concern between PIM and PSM behavior is about the consideration of a platform-oriented constraint. Thus, the PIM behavior presents the functionalities of the software in behavioral manners, being independent from platform constraints, whereas the PSM behavior focuses on a platform-concerned behavior, particularly, a time and resource-constrained functional behavior of system bound to a platform. Our point of view to the PSM behavior is based on the fact that the software behavior may be changed by timing constraints and the availability of shared and limit resources[Lee et al. 1994]. Finally, ISM specifies all the information needed to implement the PSM structural and behavioral models into a software system[Miller and Mukerji 2003].

3.1 Modeling Behaviors of PIM and PSM

(1) PIM Behavior Model and Analysis: The platform independent viewpoint focuses on the operation of a system, while hiding the details necessary for a particular platform[Miller and Mukerji 2003]. In the same context, the software behavior in PIM model centers around software's functionalities for the purpose of software, being independent from platforms. Regardless of the platforms, the software must generate correct results according to inputs. Thus, the functional behavior model of software also pinpoints the correct relation of inputs and outputs in behavioral manners. In our approach, the PIM behavior model specified in Statecharts defines relations of inputs and outputs based on system states. In our approach, two analysis methods, simulation and model checking, are used to check the correctness of functional behaviors in Statecharts for PIM behavior models. The simulation is used to check whether the system in the model reaches a desired state during executing its behavior and interacting with the environment. Model checking is a well-known formal verification method. It checks if there are no undesirable states while traversing all the states. In our approach, it is used to search an undesired situation that the system may reside in.

(2) PSM Behavior Model and Analysis: The platform specific viewpoint includes platform-constraints into software functional behaviors in the platform independent viewpoint. A PSM expands the specification of PIM with details that specify how that system uses a particular type of platform[Miller and Mukerji 2003]. Thus, PSM structural models complement PIM structure models with data-related information. A PSM behavior model, we propose here, focuses on software behavior that is restricted by a platform-given constraint. The correctness of the real-time embedded software is not only sensitive to time but also subject to resource availability because the software behavior may be delayed due to a lack of necessary resource. Generally speaking, most resources in the software are managed by the platform that synchronizes software processes based on the status of resources and scheduling policy. Thus, the platform-concerned behavior model for PSM, in our approach, centers on software behaviors bounded to a time and resource constraint imposed by a platform. The PSM behavior model is modeled in TRoS. This is suited to model a platform-concerned software behavior for the following reasons: First, the TRoS model includes both functional and timed and resource-constrained behavior of a system since it is extended from Statecharts that can capture a functional behavior of the system. Second, the transformation from PIM and PSM is performed efficiently because TRoS is extended from Statecharts by only adding time and resource-related information to Statecharts. Third, the analysis of timed behaviors in TRoS is supported by existing formal verification tools; the time and resource-related information in TRoS is represented in the form of a timed action adopted from ACSR (Algebra of Communicating and Shared Resources)[Lee et al. 1994], the information in a timed action can be transformed into an ACSR model to prove the correctness of the timed behavior of system, and the timed behavior in ACSR can be checked by bi-simulation in VERSA[Duncan et al 1995], a formal verification tool for ACSR, based on the notion of equivalence in CCS[Milner 1989].

3.1.1 Statecharts

Statecharts is a popular behavior modeling languages[Eshuis and Wieringa 2000, Eshuis et al. 2002, Harel 1987, Harel and Naamad 1996]. It is becoming more prevalent for the development of safety or mission critical systems and software, as it is supported by formal verification tools, such as model checking. Recently, it plays a role as a standard specification tool for the automobile industry. Most constructs in Statecharts are based on finite-state automata, thus most engineers can define the system behavior easily. The syntax of Statecharts consists of states, transitions, events, guards, and actions; a state can be one of an OR-State, an AND-state, and a Basic state. A transition from one state to another state is taken if transition conditions for the transition hold. A transition condition is expressed in the form of $E[C]$, where E is an event expression and C a guard expression. An action is performed along with a sequence of operations when a transition is taken.

3.1.2 TRoS

We overview TRoS here, but we do not detail the syntax and semantics of TRoS presented in [Kim et al. 2010]. TRoS is a time and resource-constrained extension of Statecharts from the view of constraints with which platforms restrict the software behavior. TRoS is an extension of Statecharts as follows:

TRoS = Statecharts + Timed Actions

The behavior of TRoS is composed of functional behaviors and timed behaviors; the functional behavior is represented using the constructs of Statecharts while the timed behavior is modeled using the construct of timed action. A system in TRoS consists of a set of TRoSs that run in parallel and share limited resources. The use of resource is represented by the notion of timed action, and the communication between parallel TRoS components is supported by instantaneous events. A timed action denotes a timed and prioritized use of a resource[Lee et al. 1994, Lee et al. 2006]. It is assumed that an execution of a timed action takes one or more time units in respect of a global clock and consumes a set of resources during that time. Moreover, it is subject to resource availability. The notion of timed action is supported by a priority-based scheduling mechanism given in the semantics, based on the semantics of ACSR. Thus, the competition for a resource is arbitrated according to priorities of competing timed actions. The availability of every resource is checked every time unit, and the available resource is assigned to only one process with the highest priority. The situation where two or more processes with the same priority access to the same resource is not permitted in TRoS. Unlike timed action, the execution of an event action is instantaneous and never consumes any resources and time.

In addition to the constructs of Statecharts, TRoS presents a timed action node that implies a timed action. A timed action node is in the form of oval, and a timed action node includes one or more timed action expressions representing a series of timed actions. The syntax of timed action expression is as follows:

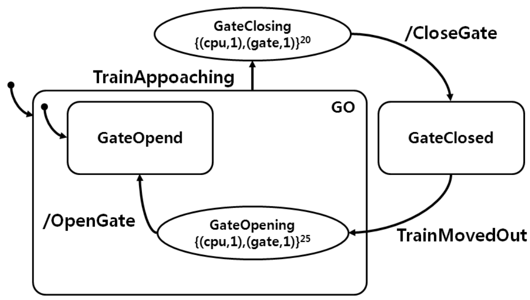


Figure 2: Railroad Crossing Gate Controller in TRoS

$$\begin{array}{lll} A & ::= & \{S\}^n \ \ | < S >^n \\ S & ::= & \epsilon \ | \ (r, ve), S \end{array}$$

r represents a resource id, and two natural numbers, ve and n, are used to respectively represent a priority for resource and execution time to pass during the timed action. A timed action with a high priority number has priority over timed actions with low priority numbers. In addition, a timed action enclosed by "<>" denotes an non-preemptive timed action that cannot be preempted by any other timed actions.

Figure 2 shows an example of TRoS model, a gate controller for railroad crossing, composed of two timed action nodes and two basic state nodes. The two basic state nodes represent that the system is in idle status, waiting for a train's approach or pass by. Meanwhile, the two timed action nodes represent the execution of two operations of closing and opening the gate. The gate controller perceives the train's approach by receiving the event *TrainApproaching* when a train approaches the gate. Then, it starts closing the gate, and the gate closing consumes 20 time-unit using two resources, *cpu* and *gate*. During the train's pass by, the gate controller stays in the state *GateClosed*, looking forward to the event *TrainMovedOut* denoting the train's leaving. The gate controller starts opening the gate, if it receives the event *TrainMovedOut* denoting the train's leaving. Then it uses two resources, *cpu* and *gate*, for 25 time-units to execute opening the gate. When the opening ends, the controller notifies completion of gate opening by sending the event *OpenGate*. The approach of a train can take place even during opening the gate. Thus the OR-state node *GO* encloses two states, *GateOpened* and *GateOpening*, and the exiting transition from the state *GO* is linked to the state *GateClosing* to represent repeating the gate closing during gate opening.

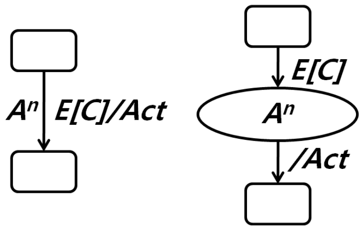


Figure 3: Syntactic Sugar for Time and Resource-related Information

3.1.3 TRoS Construction from Statecharts

In our approach, the PSM behavior model in TRoS is extended from the exiting PIM behavior model in Statecharts. The construction of TRoS is performed using the following procedure:

1. Annotate time and resource-relating information using timed action expression directly on some transitions of Statecharts including an action that needs time and resources for its execution,
2. Replace the time and resource-related information with a timed action node including timed action expressions,
3. Split the transition tagged with the time and resource-related information into two transitions: a guard transition entering into the timed action node, and an action transition exiting from the timed action node,
4. Label the existing guard and condition expression on the guard transition and the existing action expression on the action transition.

Figure 3 presents a model of Statecharts annotated with syntactic sugar in the form of timed action expression representing time and resource-related information and its corresponding model of TRoS. For a Statecharts, the timed action A^n is annotated on a transition labeled with $E[C]/Act$. The information in the timed action is transformed with a timed action node in TRoS. The expression $E[C]/Act$ in Statecharts is split into two parts, $E[C]$ and Act, by "/" in TRoS. Then, the event and guard expression $E[C]$ is labeled on the entering transition to the timed action node while the action expression Act is labeled on the exiting transition from the timed action node. This implies that the timed action A is serially executed for the amount n of time units after the event E occurs and the guard $[C]$ holds. In completing the timed actions, the action Act is executed instantaneously. In this way, the syntactic sugar facilitates constructing

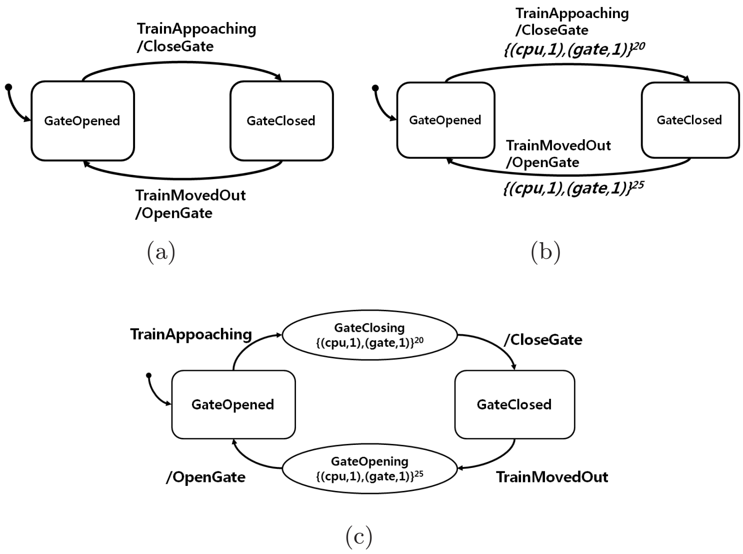


Figure 4: Railroad Crossing Gate Controller's Behavior Model

TRoS from Statecharts, a functional behavior model, only by annotating time and resource-related information.

Figure 4 shows an example of extending a Statecharts model to a TRoS model. Figure 4 (a) depicts the gate controller in Statecharts. The construction of TRoS starts with annotating time and resource-related information in timed action expression upon transitions of Statecharts. In Figure 4 (b), some time and resource-related information, such as $\{(cpu, 1), (gate, 1)\}^{20}$and $\{(cpu, 1), (gate, 1)\}^{25}$, is annotated to transitions relating to the action *CloseGate* and *OpenGate*. Next, the time and resource-related information in Statecharts is replaced with timed action nodes including the timed action expressions. After that, the timed action-annotated transition is split into two transitions; a guard transition entering into the timed action node and an action transition exiting from the timed action node. The guard and condition expressions, *TrainApproaching* and *TrainMovedOut*, are labeled on the entering transitions into respectively *GateClosing* and *GateOpening*, and the action expressions, *CloseGate* and *OpenGate* are labeled on the exiting transition from respectively *GateClosing* and *GateOpening* into *GateClosed* and *GateOpened*. Figure 4 (c) shows the gate controller of TRoS after constructing from the Statecharts in Figure 4 (a). For this transformation, the annotation in the first step cannot help being performed manually by the user because there is no way to determine the exact execution time for an operation on a platform and to identify operations needing shared resources. Thus, the execution time of operations is assumed by the user. However, we believe that the remaining transformation steps can be carried out by means of systematical way we propose here.

3.2 Analysis for MDA Behavior Models

The analysis of PIM and PSM behavior models is performed using the following methods.

(1) Positive analysis: this analysis demonstrates that a designed system performs a necessary task or action and that a design is met by verifying it against a requirement[Bailey and Martin 2010], i.e., from the positive view of functional requirement, it checks whether the designed system presents desired behaviors/functionalities required in the functional requirement. For instance, the gate controller (GC) mentioned earlier must close the gate ($down\_gate$) if the gate is open ($gate\_up$) and a train approaches the crossing ($train\_approach$). The requirement specification might be as follows:

$$GC \vdash (gate\_up \wedge train\_approach) \rightarrow down\_gate$$

In our approach, the positive analysis is performed by simulation; scenario to be performed is derived from functional requirements, and the designed system is simulated one by one according to the scenario to check if the outcomes for given inputs conform to the expected outcomes.

(2) Negative analysis: this analysis demonstrates that bugs do not exist[Bailey and Martin 2010] and exposes any flaws in the implementation of a requirement; from the negative view of functional requirement, it checks if the designed system should never present undesirable behaviors/functionalities. For instance, the gate controller must not close the gate down ($\neg down\_gate$) if the gate is open ($gate\_up$) and no train approaches the crossing ($\neg train\_approach$), and the requirement specification might be as follows:

$$GC \vdash (gate\_up \wedge \neg train\_approach) \rightarrow \neg down\_gate$$

This specifies a prohibited action that the gate controller should never take in its execution. This analysis is performed by model checking. Unfortunately, it is impossible to identify all the undesirable cases from the system, thus we select some critically undesirable cases to analyze the system.

(3) Timing analysis: this analysis demonstrates that the system responds on time. It checks if all the functionalities completes its own execution by deadline. For instance, the gate controller must complete closing the gate in 25 seconds, and the requirement specification might be as follows:

$$GC \vdash (gate\_up \wedge train\_approach) \rightarrow_{\leq 25} down\_gate$$

The timing analysis is performed her by bi-simulation introduced in [Lee et al. 1994], which checks if a timed behavior of a system produces no deadlock and bi-simulates a timely infinite idle system. If the system does not results in deadlock and bi-simulates the idle system, it is proved that all the processes in the system are schedulable.

The positive and negative analysis are used to prove the correctness of data-flow that the PIM behavior in Statecharts takes, whereas the timing analysis is used to prove the correctness of control-flow that the PSM behavior in TRoS controls i.e. schedulability.

3.3 Discussion for our MDA Behavior Models

Note a certain time or resource constraint makes a functionality of the software in PIM change to satisfy the purpose of the system when analyzing the timed behavior of it in PSM. For instance, the gate controller in Figure 4 (c) does not care for the situation when a train approaches the crossing during opening the gate. The reason is due to the model that assumes the execution of opening and closing the gate is instantaneous. Thus, the original functionalities of the system needs to be changed to consider a time consUMLtion of executing operations and resource limitations. For this reason, some PIM behaviors may need to be corrected to meet the purpose of the system through analyzing the timed behavior of PSM models. Figure 2 is an enhanced gate controller of Figure 4 (c) that reacts to another train's approach during gate opening.

Particularly, the parallelism of some parallel PIM components may not be preserved when they are transformed into some current PSM behaviors. It is because any parallel executions may be serialized due the limitation of resources in a platform. Thus, the PSM behavior may need to be correct to meet not only a platform-given time and resource-constraint but also functional requirements during the timing analysis of PSM. However, we do not further discuss verification of the PSM behavior in terms of functionality here.

4 Case Study: Distance Control Module of Railway Interlocking Control System

Now, the development of a real-time embedded software system is illustrated here to explain our approach to behavior models for MDA. The real-time software, we provide here, is a part of the railway interlocking control system, called Distance Control Module (DCM).

The development of DCM applies IEC 61508[IEC 61508], a standard for safety critical embedded systems, to the development of the Korean railway control system. DCM is a part of the interlocking control system, called the mockup

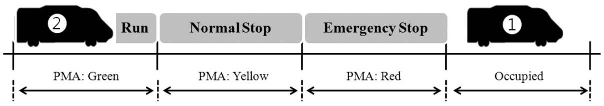


Figure 5: An Example of Permissive Moving Authority (PMA) Setting

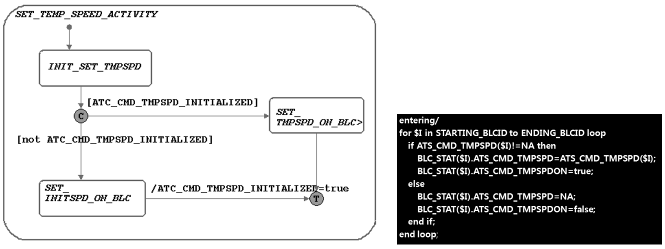
system. The mockup system consists of a central control system, control devices around railways, and trains simulated by Automatic Train Protection (ATP). The control center is composed of two main controls; Automatic Train Supervision (ATS) administrating the comprehensive railway control, and Control Route Distance (CRD) responsible for the comprehensive safety control. In particular, CRD is composed of the Distance Control Module (DCM) and Inter-locking Control Module (ICM). DCM supervises the movement of train to maintain a safe train interval, and ICM controls the route and direction of the moving train.

Figure 5 shows an example of controlling the movement of trains based on PMA that is the DCM's main processing data. The railway, as shown in Figure 5, is composed of a series of sections, called blocks, and every block yields a relative permission for every train. Figure 5 shows the status of PMA values set to four blocks where train 2 is going to run on. DCM must compute the value to PMA for every block to prevent two or more trains from moving on the same block at the same time. A value of PMA is one of Green, Yellow, and Red denoting respectively normal or limited-speed running, normal stop, and no entrance or emergency stop. The computation of the PMA value is based on information about status of railways, position and running speed of trains, and commands of ATS regarding a block close/open and temporary speed.

4.1 DCM Models

The model of DCM is divided into two aspects; structural aspect and behavioral aspect. The DCM structural model is given in Activity-charts of STATEMATE, similar to the UML class-diagram and defines the functional structure of DCM and data-flow between functional components, whereas the DCM behavioral model is defined in Statecharts and describes the overall functional behavior of the DCM functionality.

The main functionalities of DCM are achieved by the following eight functional components: **MANAGEMENT BLC STAT** computes block status, based on block-related information from ATS, ATP, and ICM. **BLC OPEN CLOSE** deals with the command of block open/close from ATS and commands **MANAGEMENT BLC STAT** to apply the change of block status to PMA for the relating blocks. **SET TEMP SPEED** deals with the temporary speed command of ATS for a block and commands. **MANAGEMENT BLC STAT**



(a) Set Temp Speed (b) Action Specification

Figure 6: PIM Behavior Models of DCM

to apply the temporal speed to the block. **MONITOR TRAIN DIRECTION** compares the current direction of a train to the scheduled direction, and if two directions are different, it notifies **MANAGEMENT BLC STAT** of the difference to make the related train cease movement. If the problem is resolved, it enables the train to move to the released block. **VERIFYING TRAIN POSITION** locates every train based on information from trains and reports the information of train-occupied blocks to **MANAGEMENT BLC STAT** to apply the change of block status. **TRAIN DISTANCE CONTROL** delivers PMA values to ATP, a train simulator, to control the movement of trains. **DISPLAY DCM STAT** displays the status of the system including trains, railways, and various commands from ATS and other controllers. **Scheduling** controls the executions of the seven functions to achieve the functionality of DCM, based on time and system conditions.

4.1.1 PIM Behavior Models

The platform independent behavior model of PIM for DCM describes computations of the system status and control variable values based on the current system status, input events and data. A behavior of DCM is composed of reading input data, processing control values based on the data, and delivering the computed control values to other functions.

Figure 6 (a) depicts the behavior of the function **Set Temp Speed** in Statecharts. Some behaviors in Figure 6 (b) are defined in an action definition language, such as Action Specification Languages in xUML[Mellor and Balcer 2002]: the array *ATS_CMD_TMPSPD* contains a temporal speed command from ATS. Once a temporary speed command is given through *ATS_CMD_TMPSPD*, the block to be restricted by the temporal speed is identified, and then the temporal speed is applied by setting the speed value to the corresponding block status in the array *BLC_STAT*.

Task	WCE (time unit)	Priority (ACSR)	HW Resource	Shared Resource
BLC_OPEN_CLOSE	2	9	cpu	ATS_CMD_BLC_CLOSE, BLC_STAT
SET_TEMP_SPEED	2	8		ATS_CMD_BLC_TMPSPD, BLC_STAT
MANAGEMENT_BLC_STAT	4	4		BLC_STAT
MONITOR_TRAIN_DIRECTION	2	5		BLC_STAT, TRN_STAT, TTRNID, TRN_CONN
VERIFY_TRAIN_POSITION	11	6		BLC_STAT, TRN_STAT, TTRNID, TTRNSCHED, DCM_TRN_CONN_OK, TRN_CONN
TRAIN_DISTANCE_CONTROL	6	3		BLC_STAT, TRN_STAT, TTRNID, TTRNSCHED, DCM_TRN_CMD
DISPLAY_DCM_STAT	4	1		

Table 1: Time and Resource Constraints

4.2 PSM Behavior Models

The platform-specific behavior of PSM for DCM is designed based on a given time and resource constraint as shown in Table 1, which enumerates the assumed worst-execution time, necessary hardware and software resources and priorities for each function in DCM. Based on these time and resource constraints, the platform specific behaviors of DCM are defined in TRoS by complementing some transitions of Statecharts with time and resource-related information in syntactic sugar.

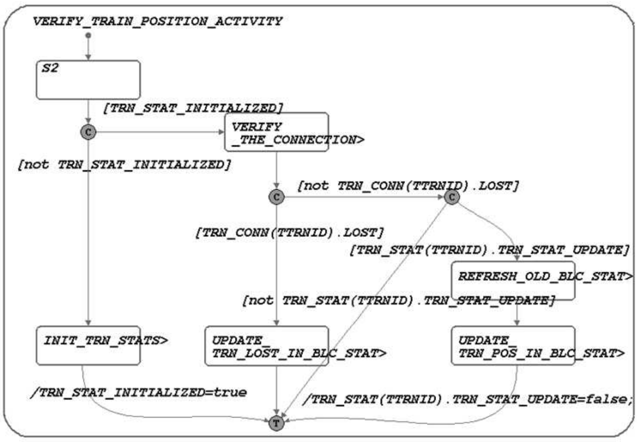
Figure 7 shows both the PIM and PSM behavioral model of **Verify Train Position**; the PIM behavior in Figure 7 (a) is defined with the constructs of Statecharts, whereas the PSM behaviors in Figure 7 (b) add time and resource-related information in syntactic sugar to some transitions of PIM behaviors. For instance, the entering transition into the state *VERIFY_THE_CONNECTION* is described in the PSM model to use a series of resources composed of *cpu*, *BLS_STAT*, *DCM_TRN_CONN_OK*, *TRN_CONN*, and *TTRNID* at the priority 6 for 1 time-unit. The resources to be used are selected among resources and the priority for those resources are also given by the constraint proposed in Table 1. The sum of the execution time of **Verify Train Position** cannot exceed the assumed worst-case execution time in Table 1. In this way, the PSM behavior models can be constructed, based on time and resource constraints.

4.3 Analysis for DCM

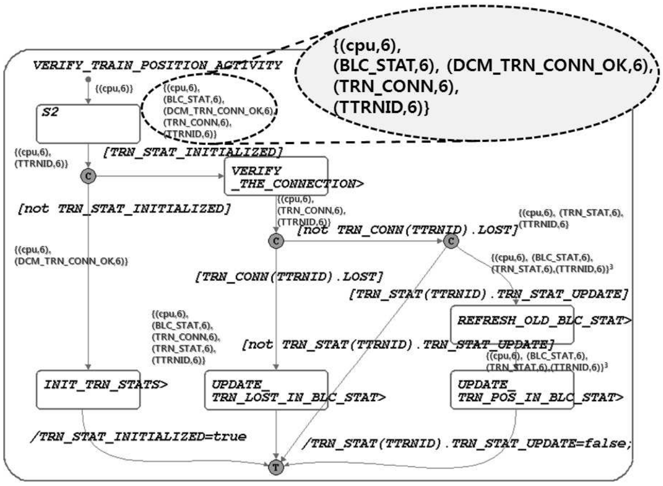
In analyzing DCM, the PIM behavior models of DCM are analyzed by positive and negative analysis using simulation and model checking while the PSM behavior models are analyzed by timing analysis using bi-simulation.

4.3.1 Positive Analysis using Simulation

We performed the positive analysis for DCM using simulation. In this simulation, the analyzer can enter input events and values into the simulated system

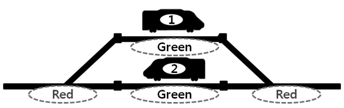


(a) PIM Behaviors

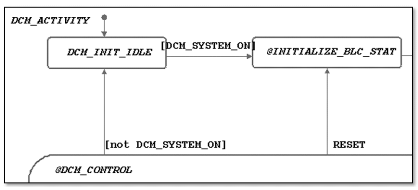


(b) PSM Behaviors

Figure 7: PIM and PSM for Verify Train Position



(a)



(b)

Figure 8: Errors in DCM Model Behaviors

and check if the output events and values after an execution of the system are desired against the functional requirement. Through this simulation, some flaws in functional requirements of DCM are found. One of them is an incompleteness error caused by the following the functional requirement:

> **"DCM should set to Red the value of PMA for the first rear block where a train has left"**

Unfortunately, the functional requirement may result in a deadlock situation described in Figure 8.(a): After two trains run across, they would proceed simultaneously to the rear block that each other each other has left. However, DCM changes each other's rear block to Red based on the requirement; thus, they both can no longer move ahead because the blocks into which they are proceed to are set to Red, implying no entrance.

4.3.2 Negative Analysis using Model Checking

The negative analysis for DCM was performed by model checking. One property to be proven is that there is no non-determinism in the DCM execution. The non-determinism implies two and more executions are not determined to be executed due to the multiple satisfactions of conditions for the executions.

Figure 8.(b) shows a non-determinism situation; if the occurrence of event *RESET* and the satisfaction of guard [*not DCM_SYSTEM_ON*] occur at the same time, two transitions into two different the states, *DCM_INIT_IDLE* and *INITIALIZE_BLC_STAT*, are enabled simultaneously. This non-deterministic situation is solved by prioritizing their executions.

4.3.3 Timing Analysis using Bi-Simulation

The timing analysis of DCM checks if every functional component of DCM completes by its deadline. The constrained behavior of the system impacted by two important parameters, the assumed worst-case execution and shared resources, are the focused of this analysis. The action subject to the availability of resource may be delayed if the necessary resource is not available due to other's actions using the resource. The worst case execution time for an action is assumed by the assumed execution time plus delay time due to unavailable resource.

To analyze the timed behavior of DCM, only timed actions are extracted from TRoS into ACSR, which can be verified by formal verification tools, VERSA. Consequently, we could obtain the result of timing verification for DCM as shown in Figure 9, where it is checked if the designed timed behavior model of DCM denoted by ***DCM***, is bi-similar with an infinite idle process denoted by ***IDLE***. The result is "***'true (by prioritized strong equivalence)***", which means that

Figure 9: The Result of DCM's Timing Verification

all the timed behaviors of DCM have no resource collision and missing deadline. Thus, we could conclude that all the executions of DCM functions complete on time, i.e., they are schedulable unless there is an interruption to the execution of DCM.

5 Conclusions

MDA should be equipped with the ability to convert from one to another, trace between models, and analyze important characteristics. Behavioral models for PIM and PSM are also called for to meet the rules. This paper presented a software behavior modeling and analysis framework that satisfies those rules for MDA. Some contributions of our approach to MDA can be as follows:

– Suggest formal behavior models of PIM and PSM easily and efficiently extensible from one to another,

– Suggest time and shared resource as parameters characterizing PSM behaviors extending PSM behaviors,

– Provide analysis views for PIM and PSM behaviors.

In MDA, the platform-related concerns identify the difference between PIM and PSM. However, it is not easy to define their behavioral characteristics. Thus, we suggested that the PIM behavior focuses on the functionality of software independent from a platform while the PSM behavior extends the functionality of software in terms of platform-given constraints characterized by two parameters, time and shared resources. Thus, we proposed that the PSM behavior is defined in TRoS and extended from the PIM behaviors in Statecharts. One reason why we use TRoS and Statecharts for PIM and PSM behavior models is because TRoS has the ability to describe a timed behavior of the system with the notion of timed action for the description of real-time system. Another reason is that TRoS has the ability to be transformed from Statecharts only by adding time and resource-related information to the behavior of Statecharts, and the transformation satisfies the principle of MDA that MDA provides a transformation rule from PIM to PSM. In our approach, the specifications in PIM and PSM are supported by various analysis methods, such as simulation, model checking and bi-simulation to provide formal proof for the development of safety-critical software. Thus, our approach, we think, is suited for the development of safety-critical embedded software using MDA approach.

However, we think the PSM behavior also needs to be analyzed in terms of the functionality because the timing and resource constraints may change the behavior of software. Therefore, the analysis of PSM in terms of timing and functional correctness can be further research.

Acknowledgment

The authors thank the anonymous referees and the associate editor, who reviewed the paper and helped to improve it. This research was partially supported by NIPA under the program of Software Engineering Technologies Development, by KOSEF R11-2008-007-03002-0, and by NSF CNS-093239 and NSF CNS 0834524.

References

[Alur and Dill 1994] Alur, R.,Dill, D, L.: “A Theory of Timed Automata”; Theor. Comput. Sci. 126, 2 Elsevier Scie., Essex, UK (1994), 183-235.

[Bailey and Martin 2010] Bailey, B., Martin, G.: “ESL Models and their Application”; Springer-Verlag New York, Inc., (2010).

[Beydeda and Volker 2005] Beydeda, S., Book, M., Gruhn, V.: “Model-Driven Software Development”; Springer-Verlag New York, Inc.,(2005).

[Csertán et al. 2002] Csertán, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., Varró, D. : “VIATRA ” Visual Automated Transformations for Formal Verification and Validation of UML Models”; Proc. 17^{th} IEEE Int. Conf. Automated Software Engineering., IEEE Computer Society, Washington (2002), 267.

[Duncan et al 1995] Clarke, D., Lee, I., Xie, H-l.: “VERSA: A tool for the specification and analysis of resource-bound real-time systems”; Journal of Computer and Software Engineering, 3 (1995).

[Eshuis et al. 2002] Eshuis, R., Jansen, D,N., Wieringa, R. : “Requirements-Level Semantics and Model Checking of Object-Oriented Statecharts”; Requirements Engineering, Springer London, 7,4, (Dec. 2002), 243-263.

[Eshuis and Wieringa 2000] Eshuis, R., Wieringa, R. : “Requirements-level semantics for UML statecharts”; Proc. 4^{th} Int. Conf. Formal methods for Open Object-based Distributed Systems, Kluwer Academic Publishers, Norwell, (2000), 121-140.

[Giese and Burmester 2003] Giese, H., Burmester, S.: “Real-Time Statechart Semantics”; Tech. Rep. tr-ri-03-239, Lehrstuhl f , Paderborn,Germany, (June 2003), 1-32.

[Harel 1987] Harel, D.: “Statecharts: A Visual Formalism for Complex Systems”; Sci. Comput. Program., 8, 3, Elsevier North-Holland Inc., Amsterdam, (Month 1987), 231-274.

[Harel and Naamad 1996] Harel, D., Naamad, A.: "STATEMATE Semantics of Statecharts"; ACM Trans. Softw. Eng. Methodol., 5, 4, ACM Press, New York (1996), 293-333.

[IEC 61508] IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems.

[Kim et al. 2010] Kim, J, H., Kang, I., Lee, I., Choi, J-Y.: "Timed and Resource-Oriented Statecharts for Embedded Software"; Accepted to IEEE Transactions on Industrial Informatics, (2010).

[Larsen and Pettersson 2000] Larsen K.G., Pettersson, P.: "Timed and Hybrid Systems in Uppaal2k"; Modelling and Verifying Parallel Processes (2000).

[Lee et al. 1994] Lee, I., Br'emond-Gr'egoire, P., Gerber, R. : "A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems"; Proc. IEEE Spec. Issue on Real-Time Systems (Jan. 1994), 158-171.

[Lee et al. 2006] Lee, I., Philippou, A., Sokolsky, O.: "A Family of Resource-Bound Real-Time Process Algebras"; Proc. Workshop Essays on Algebraic Process Calculi (APC 25), Electronic Notes in TheoreticalComputer Science, 162, (2006), 221 - 226.

[McNeile and Roubtsova 2009] McNeile, A., Roubtsova, E.: "Composition semantics for executable and evolvable behavioral modeling in MDA"; BM-MDA '09: Proc. 1^{st} Workshop on Behaviour Modelling in Model-Driven Architecture, ACM, New York,(2009),1-8.

[MARTE] : "UML MARTE"; `http://www.omgwiki.org/marte/`.

[MDA] : "OMG Model Driven Architecture"; `http://www.omg.org/mda/`.

[Mellor and Balcer 2002] Mellor, S, J., Balcer, M. : "Executable UML: A Foundation for Model-Driven Architectures"; Addison-Wesley Longman Publishing Co., Inc., Boston, (2002).

[Merlin 1974] Merlin, P.M.: "A study of the recoverability of computing systems"; PhD thesis, Department of Information and Computer Science, University of California (1974).

[Miao et al. 2002] Miao, H., Liu, L., Li, L.: "Formalizing UML Models with Object-Z"; Proc. 4^{th} Inter. Conf. on Formal Engineering Methods, Springer-Verlag, London, (2002), 523-534.

[Miller and Mukerji 2003] Miller, J., Mukerji, J.: "MDA Guide Version 1.0.1"; Tech. Rep., Object Management Group (OMG)(2003).

[Milner 1989] Milner, R.: "Communication and Concurrency"; Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989).

[Mohagheghi and Dehlen 2008] Mohagheghi, P., Dehlen, V.: "Where Is the Proof? - A Review of Experiences from Applying MDE in Industry"; ECMDA-FA '08: Proc. 4^{th} European conference on Model Driven Architecture, Springer-Verlag, Berlin, Heidelberg, (2008), 432-443.

[Riccobene and Scandurra 2009] Riccobene, E., Scandurra, P.: "Weaving executability into UML class models at PIM level"; Proc. 1^{th} Workshop on Behaviour Modelling in Model-Driven Architecture, ACM, New York, (2009), 1-9.

[Schulz et al. 2000] Schulz,S., Ewing,T,C., Rozenblit,J,W.: "Discrete Event System Specification (DEVS) and StateMate StateCharts Equivalence for Embedded Systems Modeling"; Proc. Engineering of Computer Based Systems, 0, (2000), 308.

[Snook and Butler 2006] Snook, C., Butler, M.: "UML-B: Formal modeling and design aided by UML"; ACM Trans. Softw. Eng. Methodol. 15,1, ACM, New York (2006), 92-122.

[STATEMATE] : "IBM Statemate"; `http://www-01.ibm.com/software/awdtools/statemate/`.