Integration of Modeling and Verification for System Model Based on KARMA Language

Jie Ding
dingj@zkhoneycomb.com
Beijing Institute of Technology
Haidian, Beijing, China

Michel Reniers
m.a.reniers@tue.nl
Eindhoven University of Technology
Eindhoven, The Netherlands

Jinzhi Lu*
jinzhi.lu@epfl.ch
EPFL - Ecole Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

Guoxin Wang*
wangguoxin@bit.edu.cn
Beijing Institute of Technology
Haidian, Beijing, China

Lei Feng
lfeng@kth.se
KTH - Royal Institute of Technology
Stockholm, Sweden

Dimitris Kiritsis
dimitris.kiritsis@epfl.ch
EPFL - Ecole Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

Abstract

Model-based systems engineering (MBSE) enables to verify the system performance using system behavior models, which can identify design faults that do not meet the stakeholders' requirements as early as possible, thus reducing the R&D cost and error risks. Currently, different domain engineers make use of different modeling languages to create their own behavior models. Different behavior models are verified by different approaches. It is difficult to adopt a unified integrated platform to support the modeling and verification of heterogeneous behavior models during the conceptual design phase. This paper proposes a unified modeling and verification approach supporting system formalisms and verification. The KARMA language is used to support the unified formalisms across MBSE models and dynamic simulations for different domain specific models. In order to describe the behavior model more precisely and to facilitate verification, the syntax of hybrid automata is integrated into KARMA. We implemented behavior models and their verification in MetaGraph, a multi-architecture modeling tool. Finally, the effectiveness of the proposed approach is validated by two cases: 1) the scenario of booking railway tickets using BPMN models; 2) the behavior performance simulation of unmanned vehicles using a SysML state machine diagram.

DSM '21, October 18, 2021, Chicago, IL, USA

ACM ISBN 978-1-4503-9106-1/21/10…$15.00
https://doi.org/10.1145/3486603.3486775

CCS Concepts: • **Computing methodologies** → **Model verification and validation**; • **Software and its engineering** → *Integration frameworks.*

Keywords: behavior model, modeling and verification, model-based systems engineering, KARMA language

ACM Reference Format:
Jie Ding, Michel Reniers, Jinzhi Lu, Guoxin Wang, Lei Feng, and Dimitris Kiritsis. 2021. Integration of Modeling and Verification for System Model Based on KARMA Language. In *Proceedings of the 18th ACM SIGPLAN International Workshop on Domain-Specific Modeling (DSM '21), October 18, 2021, Chicago, IL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3486603.3486775

1 Introduction

With the rapid growth of system complexity, Model-Based Systems Engineering has been increasingly used in the development process of complex systems. MBSE makes uses of formal models to support system requirements, design, analysis, verification and validation activities involved in the entire life cycle of the system [1]. The abstract system model is used as a single source of truth to guide the subsequent development process, thereby increasing the efficiency of the R&D process.

The early-designed system model enables to represent the solution of the developed system across the entire development process, so the correctness of the system model is very important, because it decides if the developed solution can satisfy the system requirements [2; 3]. System behavior models are used to describe the dynamic characteristics of the system in the early-design phase. The verification of behavior models can prove whether the functional and performance requirements of the system are met, thus reducing the errors in the early-design phase.

The development process of complex systems is a multi-domain process. When using an MBSE method to model complex systems, models are created from different perspectives, which correspond to different domain specific knowledge

and different stakeholders [4]. Behavior models in different domains are created by different modeling languages. For example, BPMN (Business Process Model and Notation) is used to describe the behavior between the system and users from the perspective of business process, and SysML state machine diagrams are used to describe the behavior of subsystems or components within a system. There are different methods and tools for verifying behavior models in different domains. It is difficult to adopt a unified integration platform to support the modeling and verification of heterogeneous behavior models constructed by different modeling languages.

Meta-modeling approaches are widely used to formalize modeling languages with the help of meta models. The GOPPRR (Graph-Object-Port-Property-Relationship-Role) approach is a meta-modeling approach which enables to describe several domain-specific languages [5]. The GOPPRR-E (Graph-Object-Point-Property-Relationship-Role-Extension) approach is proposed in the basis of GOPPRR, which constraint rules used to construct meta models is added [6]. The semantic modeling language KARMA [7] based on GOPPRR-E is developed to uniformly describe different domain models involved in the development of the complex system. However, KARMA lacks the ability to support the verification of system models currently.

This paper focus on a unified modeling and verification approach to integrate multi domain system behavior models. This approach makes use of the KARMA language based on GOPPRR-E meta-modeling approach to uniformly describe behavior models in different domains. Based on the unified description, hybrid automata are used to extend KARMA for supporting dynamic verification. Firstly, the mappings between the KARMA system behavior model and the hybrid automata model are established. On this basis, the syntax of hybrid automata is integrated into KARMA to support a more detailed description of the system behavior. Then a semi-automatic process is implemented for verifying the system behavior by a hybrid automata solver. First, a simulation compiler is used to compile the system models with descriptions of system behaviors based on hybrid automata into a .sim file, as the input to the hybrid automata solver. After the execution process by the solver, simulation results are generated. The contribution in this paper is to propose a semantic approach to integrate system model formalisms and dynamics based on KARMA and hybrid automata theory, which provides a solution for verifying the system behavior among heterogeneous modeling languages.

The rest of this paper is organized as follows. We discuss the related work in Section 2. In Section 3, we specifically introduce the unified modeling and verification approach for MBSE, including the unified expression of the behavior model based on GOPPRR-E meta-modeling approach and the behavior verification based on hybrid automata. Moreover, the implementation process of verification is introduced specifically. Then Section 4 describes two cases to clarify and verify the proposed approach. Finally, we offer the conclusions in Section 5.

2 Related Work

In this section, we describe several methods to verify behavior models of different domains in existing research, including formal verification, simulation, and executable models.

Formal verification is based on complex formula reasoning to realize the verification of the system model. A behavior model is verified by transforming the model into the input language of an existing formal verification tool. Samir et al. [8] transformed SysML activity diagrams into models for the probabilistic symbolic model checker PRISM to realize the verification of the SysML activity diagrams. Hongli et al. [9] integrated the model checker NuSMV into a SysML modeling tool. The SysML model is transformed into a NuSMV model to realize the analysis and verification of system security. Edward, Messaoud, C. Dechsupa et al. [10–12] used an approach of transforming SysML and BPMN behavior models into Petri net models to formally verify the behavior of a system.

Simulation is an important method for system verification. Some researchers combine modeling languages with system simulation languages to realize the automatic verification for system behavior. Modelica is a language which support for hierarchical physical modeling and simulation. Johnson et al. [13] created meta models for Modelica based on the extension mechanism of SysML to support the verification of the system model. They use a transformation method based on TGG (Triple Graph Grammars) to establish the mappings between SysML and Modelica, and then realize the automatic transformation from SysML to Modelica. Cao et al. [14] proposed a unified behavior modeling language based on SysML, which integrated the dynamic characteristics of physical control systems. Then, the model described by this language is executed by a system simulation tool to realize the automatic simulation of this model. Verification based on simulation is usually implemented with highly customized model transformations. If system models are built by different modeling languages, it is difficult to adopt a unified transformation method to integrate these models into other simulation tools.

Modeling languages such as SysML and UML can well express the structure and behavior of the system, but they lack precise notations to describe system behaviors so that the model can be dynamically executed. By extending the syntax of SysML and UML, system behaviors can be modeled more precisely, thus supporting the verification of the behavior model. Object Management Group (OMG) proposed ALF [15] to analyze class diagram, activity diagram and state machine diagram. SparxSystem added syntax to the SysML state machine diagram in the form of configuration box, and

verified the SysML behavior model through the simulation of state machine. The method which enables models to be dynamically executed enables to build and verify models in one platform. But it usually extends syntax for one modeling language which is not universal for other modeling languages.

From literature review, we find that a lot of research has been done on the verification of system behavior built by modeling languages such as SysML and UML. However, behavior models in different domains are built with different modeling languages and verified by different methods and tools. There is a lack of a user-friendly method and platform to integrate modeling and verification of multi-domain behavior models, which is the main motivation of this paper.

3 Integrate Modeling and Verification of System Behaviors

3.1 Unified Formalism of Multi-domain System Behavior Models

To realize the unified verification of the multi-domain behavior model, the KARMA language based on GOPPRR-E meta-modeling approach is used to uniformly formalize the heterogeneous behavior models. Then the GOPPRR-E meta-modeling approach is used to express the concepts of BPMN and SysML state machine diagrams, which will be used in Section 4 for case study.

3.1.1 GOPPRR-E Meta-modeling Approach Formalizing System Behavior Models. The MOF [16] framework is a meta-modeling framework proposed by OMG, including four hierarchical structures: M3-M0 layers. As shown in Figure 1, upper-layer elements are abstractions of lower-layer elements, and lower-layer elements are instances of upper-layer elements. Each layer of MOF has the following meanings:

- M3 refers to meta–meta models which are basic elements of the constructed model compositions.
- M2 represents the meta model layer, which is used to define models in M1 layer. Modeling languages, such as SysML is defined based on meta models.
- M1 represents the model layer. Models can describe real systems in an abstract way. Models are instances of meta models and enable to support system development.
- M0 refers to the perspective of real systems, which is abstractly described by models in M1 layer. For example, we use BPMN models to describe a system from the perspective of business process.

In the MOF framework, meta-meta models in M3 layer are the foundation of the entire modeling framework. The GOPPRR-E meta-modeling approach used in this paper has six meta-meta models: Graph, Object, Point, Property, Relationship and Role. An instance of a GOPPRR meta-meta model is a GOPPRR meta model, such as the Graph meta model. An instance of a GOPPRR meta model is called a GOPPRR instance, such as the Object instance. Specially, an instance of a Graph meta model is called a model in this paper. In addition, extra concepts (Extension) are used to describe additional constraint rules used to construct meta models. Descriptions of the six meta-meta models and the Extension are shown in Table 1.

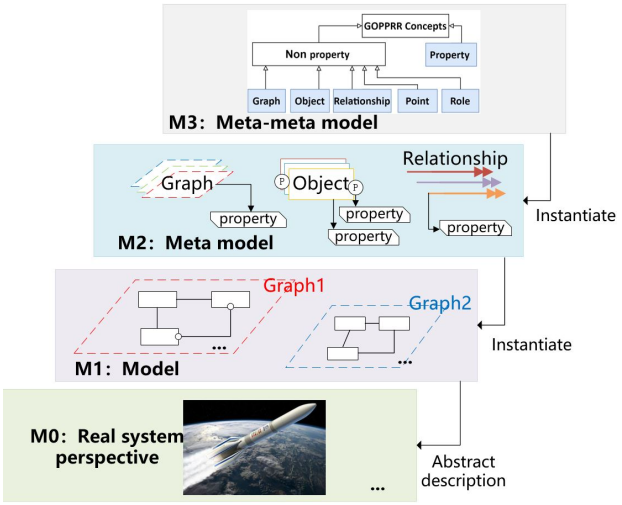

Figure 1. M3-M0 Meta-modeling Framework

Table 1. Descriptions of GOPPRR-E

Meta-meta model	Description
Graph	Graph refers to a collection of Objects and their Relationships.
Object	Object refers to one entity in a Graph.
Point	Point refers to one port in Objects.
Relationship	Relationship refers to the association between Objects, connected to Objects through Roles. One Relationship instance contains two Role instances.
Role	Role is used to define the connection rules pertaining to the relevant Relationship. Roles bind with Objects or Points in Objects.
Property	Property refers to the attribute of the five other meta–meta models.
Extension	Extension refers to the additional constraint rules used to construct meta models. Connector is one type of constraint. It refers to one binding between one Point or Object and one Role in one side of the Relationship.

The semantic modeling language KARMA is developed based on the GOPPRR-E meta-modeling approach, and the model described by KARMA can be called as a KARMA model which is a Graph instance. As shown in Figure 2, the syntax structure of KARMA is designed according to GOPPRR-E meta-modeling approach. It uses the *MetaModelClass* to describe GOPPRR meta models in M2. And it uses the *ModelClass* to describe GOPPRR instances in M1. For example, *MetaObject* and *ObjectInstance* in Figure 2 can describe Object meta models and Object instances respectively. One project described by KARMA can contain several different modeling languages with different meta models and models. Therefore, we use KARMA to support the unified expression of multi-domain behavior models.

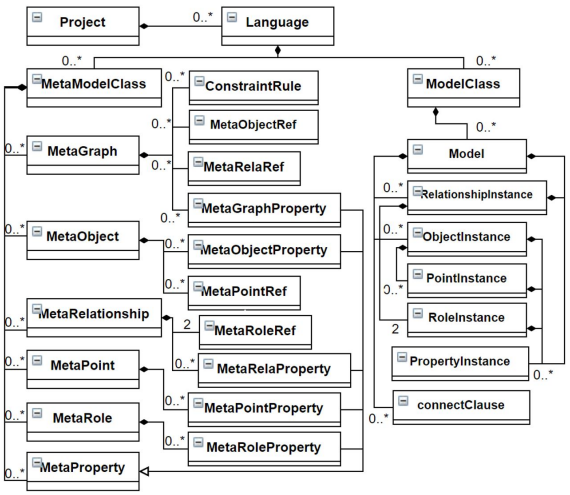

Figure 2. The Syntax Structure of KARMA Language

Based on KARMA, a multi architecture modeling tool based on Eclipse Sirius named [1] is developed. MetaGraph supports GOPPRR meta model development by using tool configurations. The meta model is instantiated as a graphical model in its modeling environment using drag-and-drop. The graphical model will be synchronized automatically to the textual syntax described by KARMA.

3.1.2 GOPPRR-E Meta-modeling Approach to Express SysML and BPMN Models. In this section, we use KARMA to express the SysML state machine (STM) diagram and BPMN diagram which will be used in Section4. The STM diagram is dynamic behavioral diagram which composed of a finite set of states and transitions between these states. The BPMN diagram is a graphical representation for specifying business processes. As shown in Table 2, when KARMA is used to express STM and BPMN diagrams, different GOPPRR meta models and constraint rules are developed by KARMA to support different concepts in STM and BPMN.

[1]MetaGraph is a multi-architecture modeling tool developed by Z.K.F.C: http://www.zkhoneycomb.com/.

Table 2. KARMA Meta Models for STM and BPMN Concepts

GOPPRR Meta Models	Concepts in STM	Concepts in BPMN
Graph meta model	STM meta model	BPMN meta model
Object meta model	Simple state, start state, final state, choice node.	Process, task, start event, end event, exclusive gateway, parallel gateway, swim lane.
Relationship meta model	Transition	Sequence flow and message flow.
Role meta model	The direction of transitions	The direction of sequence flow and message flow.
Property meta model	Do behavior in simple state, trigger, guard and effect in transition.	Conditions in sequence flow and message flow.
Constraint rules	Both sides of a transition can be connected to a state, etc.	Both sides of a sequence flow can be connected to a process, etc.

For example, we can use Object meta models to express concepts such as simple state, start state in STM and process, task, start event in BPMN. And Relationship meta models is used to describe concepts such as transition, sequence flow. These meta models are developed based on KARMA by adding semantic concepts of STM and BPMN based on the six meta-meta models.

$$MM- > F_i(MM, DSS)- > M \quad (1)$$

As shown in Formula 1, *MM* represents the meta-meta model, *DSS* represents the domain specific semantics. The F_i is a hypothetical function which is used to instantiate meta-meta model. By inputting the *MM* and *DSS* into F_i, the meta-meta model can be instantiated as a meta model. For example, the Object meta-meta model is instantiated as a state Object meta model by adding "State" semantics. In addition, some constraint rules can be developed for Object meta models and Relationship meta models. For example, the process Object meta model can be connected to one side of the sequence flow Relationship meta model.

3.2 Hybrid Automata Supporting System Behavior Verification

Although KARMA language based on GOPPRR-E meta modeling approach enables to support the unified expression of multi-domain behavior models, it lacks syntax to express the behavior precise enough to support for the dynamic verification. To address this problem, the KARMA syntax

is extended to formalize hybrid automata in order to describe system behaviors more precisely. As shown in Figure 3, firstly, the mappings between elements in KARMA model and hybrid automata are established. Then the syntax for hybrid automata is integrated into KARMA.

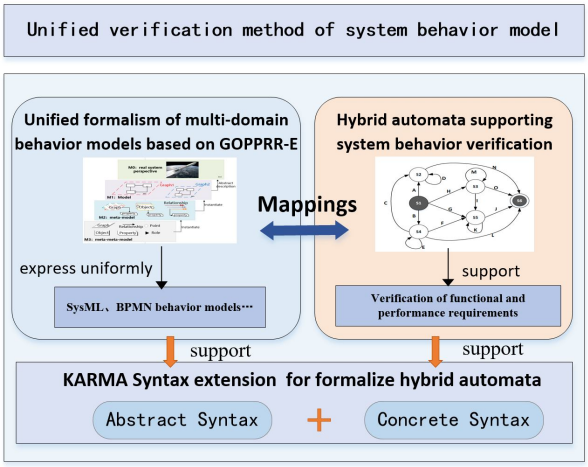

Figure 3. Framework of Unified Verification Method for Behavior Model Based on Hybrid Automata

3.2.1 Hybrid Automata Theory for Verifying System behavior.

System behavior is the characteristics of state variables in the system that change with time and events [17]. System behavior contains both continuous variables and discrete events. Hybrid automata (HA) is a finite state machine with continuously evolving variables and discrete jump transitions, which can describe the behavior of a system. In this paper, we adopt the HA theory to verify system behavior models.

HA is a mathematical model that represents a finite number of states and transitions between these states [18]. HA can be expressed as an 8-tuple, whose specific description is shown as in Formula 2:

$$HA = (Loc, Edg, X, Init, Inv, Flow, Jump, Event) \quad (2)$$

- Loc is a finite set of discrete states, $Loc = \{l_1, l_2, \ldots, l_m\}$.
- Edg is a set of transitions between discrete states.
- X is a finite set of variables. $X = \{x_1, x_2, \ldots, x_n\}$.
- Init is a set of initial states. All behaviors of a hybrid automata HA must start from an initial state.
- Inv is a set of invariants which set a value constraint for each state.
- Flow is a set of predicates each of which represents the continuous evolution of the hybrid system in one state. Equations are usually used to describe the Flow.
- Jump is a set of functions assigned to each edge that provides discrete events when a HA moves from one state to another. A jump function consists of a guard and an assignment *x' = r(x)*.
- Event is a set of discrete events.

When modeling a complex system, the HA model is used to describe the system behavior. Then the correctness of the described system behavior can be verified through hybrid automata simulation. In this paper, we make use of the CIF specification [19] which is used for hybrid automata simulation to support the verification of the behavior model described by KARMA.

3.2.2 Integration of KARMA Syntax and Hybrid Automata Syntax.

To verify behavior models, we integrate the syntax of hybrid automata into KARMA. First, we establish the mappings between the KARMA model and the hybrid automata HA.

In graph theory, the HA model can be represented by a dynamic directed graph: G = V, E, where V is the finite set of nodes and E is the set of edges directed from node i to node j [20]. V can represent the Loc of hybrid automata, and E can represent all transitions (Edg) of hybrid automata. Some elements in KARMA models have similar expressiveness to elements in directed graph G. For example, a KARMA model can be considered as a directed graph. Object instances in a KARMA model can be used to represent nodes V in a directed graph. And Relationship instances in a KARMA model can be used to describe edges in a directed graph. Therefore, the KARMA model can also support the formalism of hybrid automata. Models can be used to formalize hybrid automata. Object instances can describe the Loc, and Relationship instances can describe the Edg. The specific mappings between elements of hybrid automata HA, G (directed Graph), KARMA model and is established as shown in Table 3.

Table 3. The Mappings between Elements in Directed Graph, KARMA Model and Hybrid Automata

HA Elements	Directed Graph Elements	KARMA Model Elements
Hybrid Automata	Directed Graph	Model
Loc and Init	V (nodes)	Object instance
Edg	E (edges)	Relationship instance
directions of Edg	directions of E	Role instance
X	variables in G	Property instance
Event	events in G	Property instance
Inv	invariants in V	Property instance
Flow flow	equations in V	Property instance
Jump	Jump in E	Property instance

According to the mappings between the elements in HA and the elements in KARMA model, we established the mappings between elements in HA and elements in the KARMA STM, BPMN model as shown in Table 4.

Table 4. The Mappings between HA and KARAM STM Model, BPMN Model(Part)

HA Elements	KARAM STM Model Elements	KARMA BPMN Model Elements
HA	STM model.	BPMN model
X, Event, Inv	Property instance.	Property instance.
Loc	State Object instance.	Process, gateway Object instance, etc.
Initial State	Start state Object instance.	Start and end event Object instance.
Flow	Do behavior in state Object instance.	_____
Edg	Transition Relationship instance.	Sequence flow Relationship instance.
Guard	Trigger and guard in transition Relationship instance.	Constraint in sequence flow Relationship instance.
Assignment x' = r(x) .	Effect in transition Relationship instance.	_____

Then the syntax of hybrid automata is defined and added into KARMA language specification based on the mappings established in Table 3. The abstract concepts are shown in Figure 4, the extended syntax is labeled by dark gray block. Three declarations, *variableDeclare*, *eventDeclare*, and *equationDeclare*, are added to the syntax *Model* in KARMA Model, which are used to describe variables, events and flows in hybrid automata respectively.

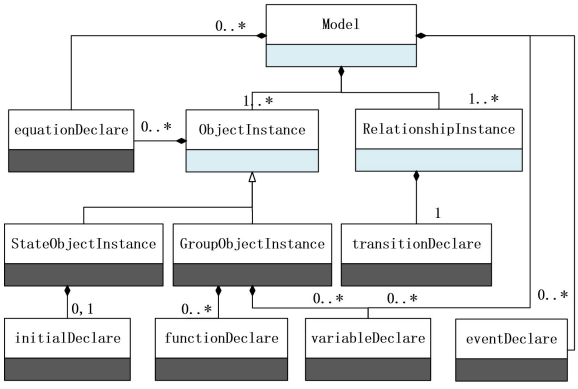

Figure 4. Abstract Concepts for the Behavior Verification Section of the KARMA Language

In addition, we divide the *ObjectInstance* into State type and Group type, which is described by *StateObjectInstance* and *GroupObjectInstance* in Figure 4 respectively. Object instances of State type can be used as Loc in hybrid automata, and equations can be defined in *ObjectInstance*. Object instances of Group type are public, which means variables and functions defined in these Object instances can be used by other Object instances. The specific description of the abstract concepts in Figure 4 is shown in Table 5.

Take Figure 5 as an example, there are two Object instances (*idle* and *moving*) and one Relationship instance (*transition1*) described by KARMA with extended syntax for system behaviors. Each Object instance is described by a *StateObjectInstance* module "*State Object identifier ... end identifier*". And the Relationship instance transition1 between two Object instances is described by a *RelationshipInstance* module "*Relationship identifier ... end identifier*". The Object instance *idle* is the ***initial*** state, which is identified by the keyword initial. When in this state, the behavior "*can-drive = true*" described by an equation will be executed. A transition module is declared in *transition1*, which means the system state will change from *idle* to *moving* when the event *start* occurs.

```
State Object idle refer object_State
   Property property_name1 ="idle" refer metaProperty_name;
   initial;
   equation can_drive = true;
end object_idle;

State Object moving refer object_State
   Property property_name2 ="moving" refer metaProperty_name;
end object_moving;

Relationship transition1 refer relationship_Transition
   Role fromRole refer incoming;
   end fromRole

   Role toRole refer outcoming
   end toRole

   transition{
         event start;
         goto moving;
   }
   ......
end transition1;
```

Figure 5. An Example of KARMA Behavior Model with Syntax of Hybrid Automata (Part)

3.3 Implementation of Behavior Model Verification

Based on the extended KARMA language, the verification of behavior models is realized in MetaGraph. As shown in Figure 6, the KARMA behavior model is built graphically in MetaGraph. Users can add syntax for system behavior to the graphic model using the related configuration window. Then the syntax for system behavior is automatically synchronized to the textual KARMA model. Then a simulation compiler (SC) is used to transform the KARMA model to an executable code for the hybrid automata solver (HAS), which is a solver based on CIF specification to simulating the hybrid automata.

The SC is composed of a syntax parser and an interpreter. The parser is developed based on another tool for language recognition (ANTLR) using a g4 file that is used to define

Table 5. The Syntax of KARMA for Behavior Verification

Description	Concrete syntax
Model is used to define KARMA models based on Graph meta models.	**partial Model** ID **instanceof** metaID …… **end** ID
StateObjectInstance is used to define State type Object instances by the keyword **State**.	**State Object** ID **refer** metaID …… **end** ID
GroupObjectInstance is used to declare Group type Object instances by the keyword **Group**.	**Group Object** ID **refer** metaID …… **end** ID
initialDeclare is used to identify the initial State by the keyword **initial**.	initial;
variableDeclare is used to declare variables of discrete, continuous, constant and algebraic types. Data types such as Int, Real, String, Boolean, Set and Distribution are supported.	**discrete Boolean** a = true; **continuous Real** b = 1.0; **algebraic Int** c = funA(arg); **constant Int** d = 0;
eventDeclare refers to declare events by the keyword event.	event eventName;
RelationshipInstance is used to define Relationship instances.	**Relationship** ID **refer** metaID …… **end** ID

transitionDeclare is used to declare transitions of Relationship instances which contains guards, effects and the new state which is transit to.	**transition** {
	[**event** identifier;]? %guard
	[**when**(condition);]? %guard
	[**do** assignment;]? %effect
	goto stateName; %new state
	};

equationDeclare is used to declare Flows which can describe behaviors. It can be used in Object and Graph instances.	**equation** can_drive = true

functionDeclare is used to declare functions which can solve complex logical and mathematical calculations.	**func** returnType funcName(datatype parameters) {
	< funcBody>
	}

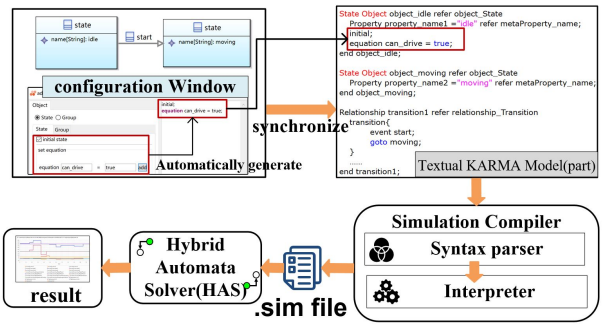

Figure 6. The Verification Process of Behavior Models

grammar following the KARMA syntax [21]. The KARMA behavior model with syntax of system behaviors is analyzed lexically and syntactically by the parser after which an abstract syntax tree is generated. Then the interpreter accesses the nodes of the abstract syntax tree and extracts the information of the key words in KARMA behavior model, the behavior model is automatically compiled to generate a *.sim* file which is the input file of HAS. Finally, the verification results are generated after simulation execution.

4 Case Study

To evaluate the proposed approach, this section takes two examples: 1) a BPMN model for describing a railway ticket booking process as a case to verify whether the behavior model can meet the requirement of ticket booking, 2) a STM model for describing a process of unmanned vehicle driving is taken as a case to verify whether the behavior model meets the system performance requirements.

4.1 Verification of the BPMN Behavior Model

The railway ticket booking system is a system that queries ticket information through the network and then supports ticket booking. There are two roles for users to booking tickets: student and normal users. Both roles can query the ticket they need by inputting the query option, and confirm whether to buy or not, otherwise they will reselect. When the booking is confirmed, users need to pay online. According to the booking process, a BPMN behavior model described by KARMA language is created in MetaGraph.

As shown in Figure 7. The BPMN model contains Object instances of start event, end event, process, exclusive gateway and parallel gateway. The type of these object instances is State. The BPMN model also contains Relationship instances of sequence flow. As shown in Figure 8-A, some discrete variables and events are added to the BPMN model through the behavior configuration window for models. A Bernoulli distribution d is used to represent the probability of buying tickets as normal users. The parameter 0.5 of d means that the probability of normal users is 0.5, so the probability of

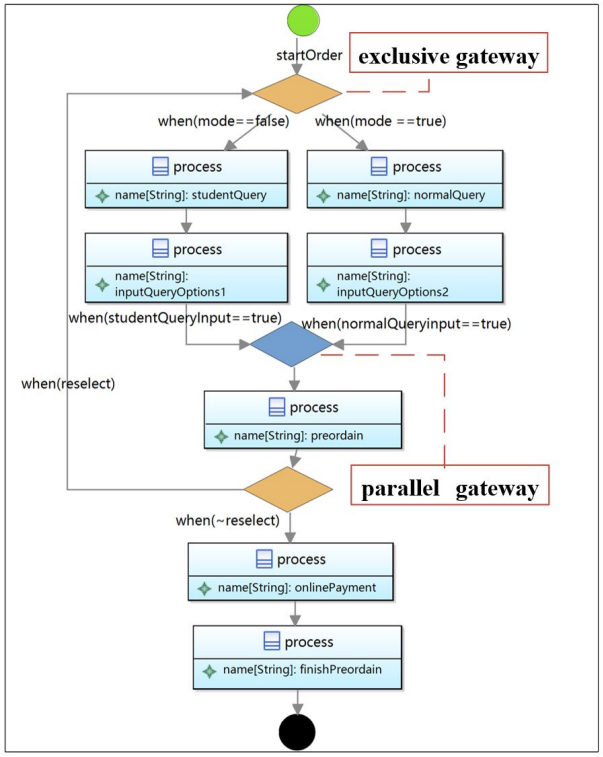

Figure 7. The Graphical BPMN Model for the Ticket Booking Process

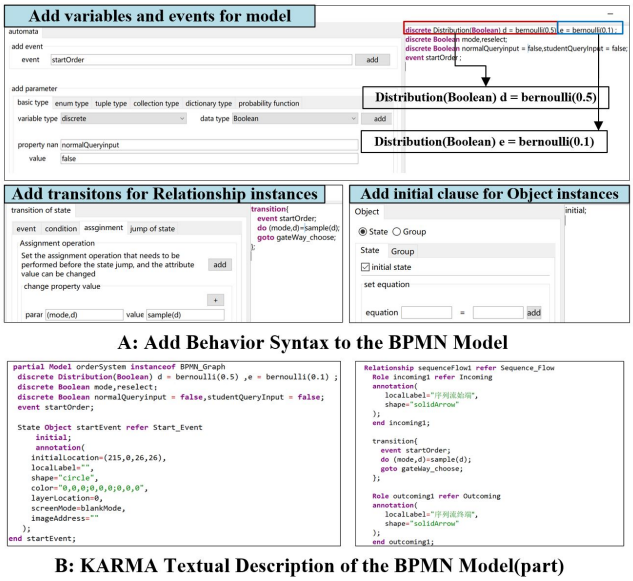

Figure 8. KARMA BPMN Model with Behavior Syntax

student users is "1-0.5", which is 0.5 too. The Bernoulli distribution *e* is used to represent the probability of users to reselect tickets. The parameter 0.1 of *e* means that the probability of reselection is 0.1, so the probability of confirming is "1-0.1", which is 0.9.

In addition, an initial declaration is added to the start event Object instance represented by a green circle in Figure 7 through the behavior configuration window for Object instances. The transition declarations are added to Relationship instances of the BPMN model. As shown in Figure 8-A, we add an event *guard*, an *effect* and a *goto* statement to one Relationship instance. The graphical BPMN behavior model with added behavior syntax will be automatically synchronized to the textual KARMA model, and a part of the KARMA model's textual description is shown in Figure 8-B.

Then, the BPMN behavior model is compiled by SC and the generated *.sim* file is executed by HAS. The verification results are shown in Figure 9, the system stops at the state *gateWay_aggregation* which is the parallel gateway in Figure 7. We found that the parallel gateway is improperly used through analysis of the BPMN model. Because there is an exclusive gateway above the parallel gateway, and the exclusive gateway in Figure 7 only selects one path, so only one path will arrive at the parallel gateway. However, the parallel gateway needs to wait for two paths to arrive to go to next state, so the system stops at *gateWay_aggregation* forever and the booking process cannot move on. Therefore, the behavior model is wrong and cannot meet the requirement of ticket booking.

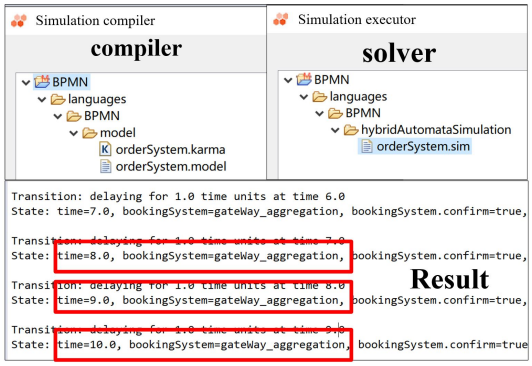

Figure 9. The Verification Result of BPMN Behavior Model

4.2 Verification of the STM Behavior Model

Unmanned vehicles are used for specific tasks such as detection and delivery. In this case, an unmanned vehicle (UA) needs to go to a place 350 meters away for a detection task. During the process of driving, the road is muddy from 150m to 250m, and the rest is flat. The UA will slow down when the road is muddy and accelerate when the road is flat. When approaching the detection destination, the UA brakes and stops. Table 6 shows the acceleration of the UA in different conditions. It is assumed that the UA starts up at the acceleration of 4 m/s2, reaches the initial speed of 12m/s after 3s. Then the UA will adjust its speed according to the

Table 6. Acceleration of the UA in Different Conditions

Condition	Acceleration	Description
startup	4 m/s2	The UV will stop accelerating when its speed reaches 12m/s
muddy	-1.5m/s2	The UV will stop decelerating when its speed reaches 8m/s
flat	2.5m/s2	The UV will stop accelerating when its speed reaches 16m/s
brake	-5m/s2	The UV will brake when approaching the destination

road condition during the next driving process. In this case, the performance requirement is that the unmanned vehicle should reach the detection destination within 30 seconds.

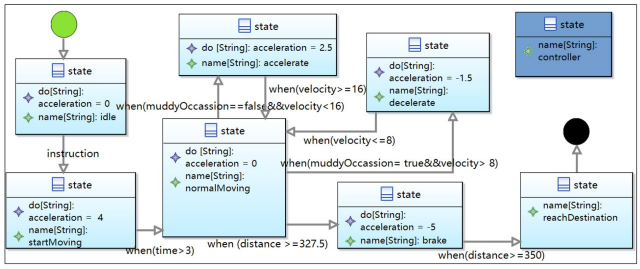

Figure 10. Thw Graphical STM Model of Unmanned Vehicle Driving

Based on the scenario above, a STM model for unmanned vehicle driving is created using KARMA as shown in Figure 10. Continuous variables, algebraic variables and events are added to the STM model through the behavior configuration window for models as shown in Figure 11-A. The algebraic variable *acceleration* is used to describe the acceleration of the UA in different conditions. The value of another algebraic variable *muddyoccasion* is assigned to a user-defined function *isMuddy(distance)* which declared in a Group Object instance called *controller*. The function *isMuddy* is used to judge whether the road is muddy or not according to the variable *distance*, which represents the driving distance of the unmanned vehicle. The variable *velocity* is used to represent the speed of the UA.

In addition, the behavior syntax is added to Object instances of State type through the behavior configuration window as shown in Figure 11-A. For example, an equation "*equation acceleration = 2.5*" is added to the state Object instance called *accelerate*, which means that the acceleration of the UA is 2.5m/s2 in this state. And a user-defined function *isMuddy(distance)* is added to an Object instance of Group type. Also, we add transitions for Relationship instances by the behavior configuration window as shown in Figure 11-A. For example, a transition that contains a guard "*when(time>3)*"

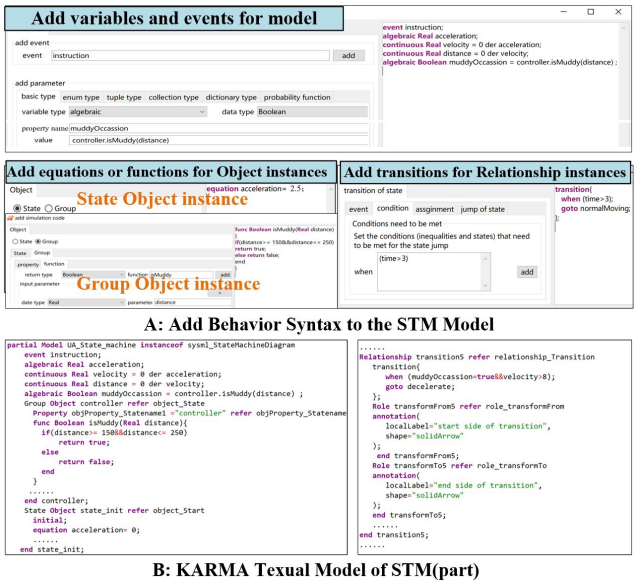

Figure 11. The KARMA STM Model with Behavior Syntax

and a *goto* clause "*goto normalMoving*" is added to a Relationship instance, which is used to describe that the UA will goto the state *normalMoving* after 3 seconds.

The STM model with more precise behavior syntax will be automatically synchronized to a KARMA textual model (part) as shown in Figure 11-B. Then the STM model is compiled by SC to generate a .sim file. The generated *.sim* file is executed by HAS and the simulation result is shown in Figure 12. From the figure, we find it takes 29.01 seconds to reach the detection destination, which meets the prescribed performance requirement (the UA should reach the detection destination within 30 seconds). Therefore, the STM model can meet the performance requirement.

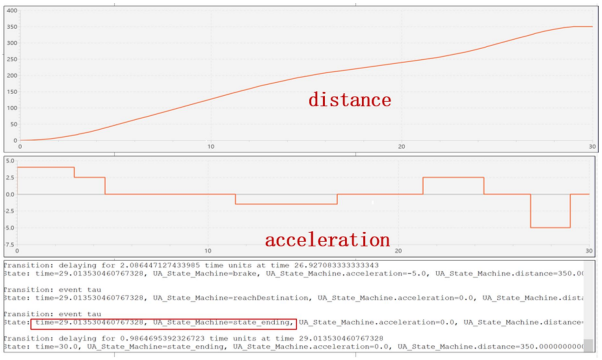

Figure 12. The Verification Result of STM Model

4.3 Discussion

In the case study, a STM model and a BPMN model are built in MetaGraph. The number of BPMN meta models and

STM meta models described by KARMA language are shown in Table 7. By adding the syntax of system behaviors to the instances of these meta models, the behavior models of BPMN and SysML can be verified based on the hybrid automata simulation.

Table 7. Number of Meta Models Used in Case Study

Modeling Language	Number of Types	
	Object meta models	Relationship meta models
BPMN	5 (process, start and end event, exclusive and parallel gateway)	1 (sequence flow)
SysML	3 (sample state, start state and final state).	1 (transition)

We found the railway ticket booking process cannot proceed normally by verifying the BPMN model. Therefore, the behavior model described by BPMN cannot meet the requirement of user ticket booking. Through the verification of the STM model of unmanned vehicle driving, we found the acceleration in Table 6 can meet the performance requirements, so the state machine model is correct. Through the case study, we found the proposed approach enables to support modeling and verification of behavior models using different modeling language specifications.

5 Conclusion

In the paper, we proposed a unified modeling and verification approach for multi-domain system behavior models. The multi-architecture modeling language KARMA based on GOPPRR-E meta modeling approach was used to uniformly express the heterogeneous behavior models. Based on the theory of hybrid automata, we extended the syntax of KARMA for describing system behaviors. Then the modeling and verification capabilities of KARMA are implemented in MetaGraph. Finally, through two cases of a STM model and a BPMN behavior model, it is illustrated that the proposed approach has the ability of integrated modeling and verification for different modeling languages.

References

[1] R. A. Luísa, F. J. Vasconcelos, B. Jaume, Model-based systems engineering: An emerging approach for modern systems, IEEE transactions on systems, man and cybernetics. 42 (1) (2012) 101–111. doi:10.1109/TSMCC.2011.2106495.

[2] B. Ronan, C. Mohammad, B. Jean-Michel, O. Iulian, Sysml models verification and validation in an industrial context: Challenges and experimentation, in: Modelling Foundations and Applications, Springer International Publishing, Cham, 2018, pp. 132–146. doi:10.1007/978-3-319-92997-2_9.

[3] A. Mounifah, N. Nan, S. Juha, Sysml modeling mistakes and their impacts on requirements, in: 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), 2019, pp. 14–23. doi:10.1109/REW.2019.00010.

[4] H. Wang, G. Wang, J. Lu, C. Ma, Ontology supporting model-based systems engineering based on a gopprr approach, in: WorldCIST, 2019, pp. 426–436. doi:10.1007/978-3-030-16181-1_40.

[5] S. Kelly, J.P.Tolvanen, Domain-Specific Modeling: Enabling Full Code Generation, Wiley, 2008.

[6] J. Lu, J. Ma, X. Zheng, G. Wang, K. Dimitris, Design ontology supporting model-based systems-engineering formalisms, IEEE Systems Journal 42 (1) (2021) 1–12. doi:10.1109/JSYST.2021.3106195.

[7] J. Lu, G. Wang, J. Ma, K. Dimitris, H. Zhang, T. Martin, General modeling language to support model-based systems engineering formalisms (part 1), in: INCOSE International Symposium, Vol. 30, 2020, pp. 323–338. doi:https://doi.org/10.1002/j.2334-5837.2020.00725.x.

[8] O. Samir, M. Otmane, D. Mourad, A formal verification framework for sysml activity diagrams, Expert Systems with Applications 41 (6) (2014) 2713–2728. doi:https://doi.org/10.1016/j.eswa.2013.10.064.

[9] H. Wang, D. Zhong, T. Zhao, F. Ren, Integrating model checking with sysml in complex system safety analysis, IEEE Access 7 (2019) 16561–16571. doi:10.1109/ACCESS.2019.2892745.

[10] E. Huang, L. F. McGinnis, S. W. Mitchel, Verifying sysml activity diagrams using formal transformation to petri nets, Systems Engineering 23 (1) (2020) 118–135. doi:https://doi.org/10.1002/sys.21524.

[11] M. Rahim, A. Hammad, M. Ioualalen, A methodology for verifying sysml requirements using activity diagrams, Innovations in Systems and Software Engineering 13 (1) (2017) 19–33. doi:https://doi.org/10.1007/s11334-016-0281-y.

[12] C. Dechsupa, W. Vatanawood, A. Thongtak, Transformation of the bpmn design model into a colored petri net using the partitioning approach, IEEE Access 6 (2018) 38421–38436. doi:10.1109/ACCESS.2018.2853669.

[13] J. Thomas, K. Aleksandr, P. Christiaan, Integrating models and simulations of continuous dynamics into sysml, Journal of Computing and Information Science in Engineering 38 (1) (2012) 122–129. doi:10.1115/1.4005452.

[14] Y. Cao, Y. Liu, H. Fan, B. Fan, Sysml-based uniform behavior modeling and automated mapping of design and simulation model for complex mechatronics, Computer-Aided Design 45 (3) (2013) 764–776. doi:https://doi.org/10.1016/j.cad.2012.05.001.

[15] O. M. Group, Action Language for Foundational UML (Alf), 2013.

[16] O. M. Group, Meta Object Facility (MOF) 2.0 Core Specification, 2003.

[17] D. Wang, H. Liang, J. Cu, Modeling and simulation of top-level design based on mbse, in: Journal of Physics: Conference Series, Vol. 1486, 2020, p. 072061. doi:10.1088/1742-6596/1486/7/072061.

[18] F. Goran, An introduction to hybrid automata, numerical simulation and reachability analysis, Formal Modeling and Verification of Cyber-Physical Systems (2015). doi:10.1007/978-3-658-09994-7_3.

[19] D. A. van Beek, W. J. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. M. van de Mortel-Fronczak, M. A. Reniers, Cif 3: Model-based engineering of supervisory controllers 8413 (2014) 575–580. doi:https://doi.org/10.1007/978-3-642-54862-8_48.

[20] Y. Chen, W. Li, Y. Guo, Y. Wu, Dynamic graph hybrid automata: A modeling method for traffic network (2015) 1396–1401doi:https://doi.org/10.1109/ITSC.2015.229.

[21] T. Parr, The Definitive ANTLR4 Reference, 2007.