Softw Syst Model (2017) 16:1201–1222
DOI 10.1007/s10270-015-0513-x

REGULAR PAPER

A model framework-based domain-specific composable modeling method for combat system effectiveness simulation

Xiao-bo Li[1,2] · Feng Yang[1] · Yong-lin Lei[1] · Wei-ping Wang[1] · Yi-fan Zhu[1]

Received: 18 September 2014 / Revised: 14 November 2015 / Accepted: 21 December 2015 / Published online: 25 January 2016

Abstract Combat system effectiveness simulation (CoSES) plays an irreplaceable role in the effectiveness measurement of combat systems. According to decades of research and practice, composable modeling and multi-domain modeling are recognized as two major modeling requirements in CoSES. Current effectiveness simulation researches attempt to cope with the structural and behavioral complexity of CoSES based on a unified technological space, and they are limited to their existing modeling paradigms and fail to meet these two requirements. In this work, we propose a model framework-based domain-specific composable modeling method to solve this problem. This method builds a common model framework using application invariant knowledge for CoSES, and designs domain-specific modeling infrastructures for subdomains as corresponding extension points of the framework to support the modeling of application variant knowledge. Therefore, this method supports domain-specific modeling in multiple subdomains and the composition of subsystem models across different subdomains based on the model framework. The case study shows that this method raises the modeling abstraction level, supports generative modeling, and promotes model reuse and composability.

Keywords Modeling and simulation · Composable modeling · Domain-specific modeling · Simulation model framework · System effectiveness simulation

Communicated by Dr. Sebastien Gerard.

✉ Feng Yang
forestyoung@nudt.edu.cn

Xiao-bo Li
lixiaobo@nudt.edu.cn

Yong-lin Lei
yllei@nudt.edu.cn

Wei-ping Wang
wangwp@nudt.edu.cn

Yi-fan Zhu
yfzhu@nudt.edu.cn

1 College of Information Systems and Management, National University of Defense Technology, Changsha 410073, Hunan, People's Republic of China

2 Department of Mathematics and Computer Science, University of Antwerp, 2020 Antwerp, Belgium

1 Introduction

System effectiveness is the measurement of the system capability to fulfill specified tasks under certain conditions, and it is a critical indicator for the quality of system product from the aspect of system utilization. In particular in the military, combat system effectiveness has become one of the most important guidelines for the demonstration and overall design of various equipments (e.g., missiles, warships). Combat system effectiveness evaluation involves combat system-of-systems (SoS) and personnel from two or more combat sides, and contains complex counterwork process. Since traditional analytical methods and physical experimentation are limited to deal with such complexity, simulation-based system effectiveness evaluation is proposed recently [1]. Based on measures of performance (MoP), combat system effectiveness simulation (CoSES) evaluates the combat system effectiveness in the counterwork process to fulfill certain tasks via simulation. Typical CoSES needs to fulfill the following functions: (1) modeling physical combat systems, combat decision rules and the combat environment; (2) exhibiting the dynamic counterwork process; and (3) evaluating the influence of MoP (or tactics) variations of certain

weapon equipments on the overall effectiveness of combat systems.

Combat system modeling and simulation (M&S) is researched at a series of abstraction levels from top to down: the theater/campaign level, the mission/battle level, the engagement level, and the engineering level [2]. System effectiveness is explicitly studied at the engagement level to evaluate system attribute or tactics alternatives under the background of SoS counterwork. So it is higher than the engineering level which concentrates on the evaluation of performance parameters of combat systems, and lower than the mission level which aims to study the influence of scale and structure variation of combat SoS on its capability. At the engagement level, combat systems are modeled conforming to principles of functional modeling [3] to achieve conciseness and efficiency. For example, a radar system is modeled using power, bandwidth, and signal-to-noise ratio to calculate the probability of discovery, identification, tracking, and interference in a typical sensor functional process, while at the engineering level it is modeled from the aspect of signal generation, transmission, reflection, attenuation, and reception.

Models lie in the heart of the simulation, and modeling is a critical issue in M&S activity. In this work, we concentrate on modeling issues and identify two prominent modeling requirements in CoSES: composable modeling and multi-domain modeling.

(1) *Composable modeling*

Combat systems usually need to fulfill multiple combat tasks. Thus combat system effectiveness evaluation should be conducted based on the comprehensive result on performing multiple heterogeneous tasks. For example, a nuclear submarine should possess the capability of anti-ship, anti-shore, anti-air, and strategic strike, so the effectiveness evaluation of a nuclear submarine needs to construct multiple corresponding simulation applications for each capability. A *simulation application* is a specific configuration of the simulation system to solve a given application problem, and it includes a specific set of simulation scenarios, simulation model instances and design of experiment files. These simulation applications can be developed independently; however, many models can be reused in different scenarios of these applications, such as the submarine model, the sonar model, and the torpedo model. So how to incorporate and reuse existing models for submarine effectiveness simulation is of vital importance to increase the productivity and efficiency in the development of simulation applications. More generally, we need to support composable modeling for the multi-application effectiveness simulation of other kinds of equipment (e.g., warships, fighter planes, and tanks).

Model frameworks are used in many M&S practices to support composable modeling. A *model framework* (also called *model architecture*) is a specification of the types of the simulation models and their interaction relationships in a typical subject domain. The model framework can support composable modeling from the following two aspects: Firstly, in the model design phase, it architects the basic entity types and relationship patterns of all simulation models and lays the foundation for detailed submodel design; secondly, it integrates all the simulation models in the model integration phase and assemble them into a simulation application. The model framework specifies the basic structural aspect of the overall simulation model architecture, and this structural specification remains invariant for multiple simulation applications.

(2) *Multi-domain modeling*

Effectiveness simulation models involves many subject domains, which include detection domain (e.g., radar, infrared, magnetic, acoustic detection), firepower domain (e.g., missiles, bombs, torpedoes, mines), platform domain (e.g., ships, aircrafts, vehicles, satellites), communication domain (e.g., voice, digital communication), cognitive domain (e.g., command and control, planning), and environmental domain (e.g., atmosphere, geography, marine, space). On the one hand, models from different domains differ evidently in behavioral semantics and should be described using appropriate formalisms accordingly. For example, models in the firepower domain usually can be described as multi-phase continuous systems; models in detection domain can be depicted as reactive discrete time systems; combat platforms can be modeled as cognition-driven continuous system; and cognitive domain can be characterized by various paradigms such as rules, events, activity and process. Effectiveness simulation needs to support multi-formalism modeling to combine these models of heterogeneous semantics. On the other hand, users and modelers from various domains are involved in effectiveness simulation to ensure simulation credibility and extensibility. It is unfeasible to adopt a general modeling method in the face of various domains, and thus domain-specific modeling methods and environments tailored for domain-specific characteristics should be provided. Then domain users and modelers can use these domain-specific infrastructures to create, modify, and extend models conveniently.

The key to address the multi-domain modeling problem is to construct domain-specific modeling infrastructure based on domain knowledge. According to the knowledge engineering in M&S, domain knowledge in CoSES can be divided into *application invariant knowledge* (AIK) and *application variable knowledge* (AVK). AIK refers to domain knowledge common to the domain and not specific to a particular simulation application, whereas AVK is the part of domain knowledge specific to one or several applications but not qualified to the domain level.

As will be discussed in detail in the following section, current effectiveness simulation research attempts to cope with

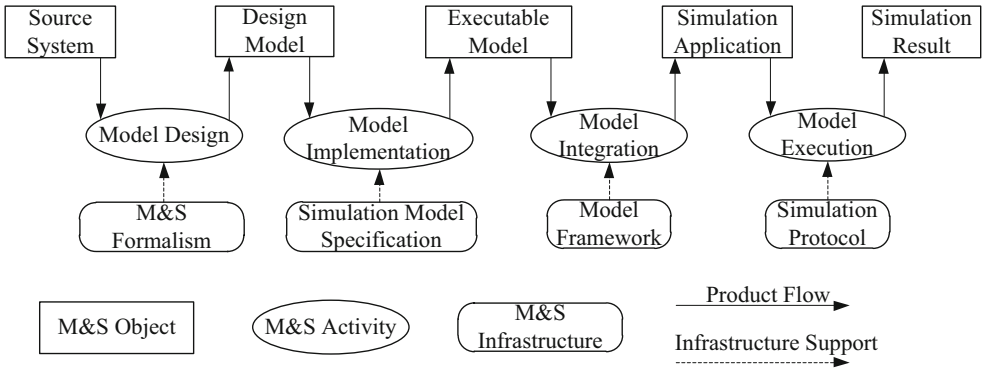


Fig. 1 A model development and execution lifecycle and its infrastructure support for CoSES

the composable modeling and multi-domain characteristics of effectiveness simulation systems solely based on a unified technological space. These methods are limited, and they fail to meet both of the two modeling requirements sufficiently.

According to our understanding and experience on CoSES research, a model framework for CoSES plays a central role in meeting these two modeling requirements. On the one hand, to meet the composable modeling requirement, a common model framework for various CoSES applications is built based on the AIK extracted from all CoSES subdomains and a domain-specific modeling language (DSML) for each subdomain is constructed using the relevant AIK. On the other hand, to meet the multi-domain modeling requirement, the model framework provides corresponding extension points or mechanisms which are used to integrate AVK for different specific simulation applications, and domain modelers use DSML to construct domain-specific model based on the AVK.

We had proposed a domain-specific modeling (DSM)-based multi-paradigm modeling methodology for complex systems [4]. This methodology can be used as a guideline for effectiveness simulation model development since CoSES is also one kind of complex simulation systems. Moreover, we had built DSM infrastructure for decision modeling in combat systems [5,6]. Based on the aforementioned efforts, we propose a domain-specific composable modeling method which supports the description of domain-specific structure and behavioral patterns using a unified model framework.

The rest of the paper is organized as follows. Section 2 discusses four kinds of existing typical M&S methods for effectiveness simulation. Section 3 presents the proposed approach and its technical implementation after a brief introduction to DSM and the model framework for effectiveness simulation. Based on the model framework, Sect. 4 investigates domain-specific composable modeling in three typical subject domains. Section 5 illustrates the application of the method with a case study, and Sect. 6 concludes this research and proposes future work.

2 Related work on M&S for CoSES

Effectiveness simulation has been widely used in requirement analysis, overall design, development and training of combat systems. A diversity of methods, standards, and platforms have been proposed and developed for these purposes. According to the modeling and simulation lifecycle [7], model development and execution can be divided into four phases (as shown in Fig. 1): model design, model implementation, model integration and model execution. Based on the technical characteristics of each phase, current M&S research on combat system effectiveness measurement can be categorized as the following four kinds: formalism-based model design, simulation model specification-based implementation, model framework-based model integration, and simulation protocol-based model execution. Actually each method includes all four phases. However, these researches are usually based on one unified technological space which mainly aims to cope with a certain phase of the whole process.

2.1 Unified formalism-based method

Formalisms in M&S are usually defined to consist of two parts: model specification and execution algorithm [8]. Model specification is a mathematical theory describing the kinds of structure and behavior that are in accordance with it, and execution algorithm specifies an algorithm that can correctly execute any model that is described in accordance with the model specification [8]. A modeling formalism is usually an abstraction of certain structural and behavioral characteristics of systems and can be used to describe diverse systems which possess the same characteristics across different application domains. Domain modelers can develop modeling environment and simulation engine based on certain formalism to meet specific domain requirements.

The unified formalism-based method aims to solve composable modeling from the aspect of model design. This method is usually based on a unified formalism which

can describe various behavioral patterns, and employs the extension capability of that formalism to integrate other formalisms. Typical unified formalisms are discrete event system specification (DEVS) [9] and Modelica [10]. DEVS can describe discrete event behavior, discrete time behavior and continuous time behavior. Modelica can describe continuous time behavior, event and state behavior of discrete systems in a causal or non-causal way. DEVS is the foundation of M&S theory and has been widely accepted by M&S community. DEVS is proven to support both theoretical and practical research of combat system effectiveness simulation, typical examples are JointMEASURE [11] developed by Lockheed Martin and simulation-based defense system analysis performed by the research team led by Professor Kim in South Korea [12]. Modelica is constructed and standardized by the joint efforts of the academia and industry, and it has become the unified M&S formalism for engineering physical systems.

This method supports model composition based on the unified formalism at the semantics level. However, no formalism is able to describe all behavior patterns appropriately and to incorporate that the AIK is not the design goal of formalisms.

2.2 Unified model specification-based method

This method provides a unified model specification for the description and implementation of multi-domain simulation models to enable composable modeling. Models from different domains should conform to a canonical model specification, such as Base Object Model (BOM) [13] and Simulation Model Portability standards 2 (SMP2) [14].

BOM is proposed to raise the abstraction level of high-level architecture (HLA) [15]-based model specification and becomes the actual model specification for HLA-based simulation system development. It introduces modeling concepts, such as entity and event, and describes the object behaviors via state machine. However, it has several disadvantages as a model specification: It lacks support for hierarchical model composition; its description of state machine-based behavior is conceptual, not formal; and it is closely coupled to and technically affiliated to HLA, and thus it encounters the same problem as HLA (as discussed in Sect. 2.4).

SMP2 is proposed by Europe Space Agency to promote model portability, reuse, and interoperability, which complies to model-driven architecture (MDA) principles by the usage of platform-independent model and platform-specific model in the simulation model development process. SMP2 standard has been successfully used in various application domains as the simulation model specification to develop large complex simulation systems, and it has proven to promote technical interoperability. However, SMP2 lacks behavioral modeling capability and mainly specifies the software implementation of simulation models. It is difficult to describe the characteristics of multi-domain and multi-subsystem for effectiveness simulation using SMP2.

The model specification (BOM or SMP2) essentially is specification for technical implementation of simulation models and is not application domain oriented. The model specification-based method can be used to represent models of various application domains in theory; however, it inevitably lacks description of AIK and cannot support domain-specific modeling directly. The M&S practice shows that the model specification-based method needs to describe the AIK for each development process of effectiveness simulation applications, and thus the workload is heavy, and this method lacks attraction to effectiveness simulation researchers.

2.3 Unified model framework-based method

The main components of complex simulation systems include simulators and various kinds of simulation models. The relationship between models and simulators are usually specified by simulation protocols (e.g., abstract simulator protocol of DEVS [9]). Another important relationship is the interfaces and interaction patterns among these simulation models. The M&S community proposes the method of model frameworks (or model architectures) to specify the types of the simulation models and their interaction relationship.

Model frameworks are built based on abstracting AIK from different simulation applications using M&S experience in the problem domain. M&S practitioners gradually figure out basic kinds of simulation models and their main interaction relationships in CoSES through years of practices. Though normally these frameworks are not described using formal methods, the AIK is purified and built into them. Moreover, the model framework provides extension points which are used to integrate AVK for different specific simulation applications.

Model frameworks are usually used in CoSES at the engagement level or higher levels since simulation applications of these levels contain models from various domains which have complex interactions. The model framework is of vital importance to integrate model and promote development efficiency. Typical examples of simulation systems using model frameworks are Extended Air Defense Simulation (EADSim) at the engagement level for air defense simulation analysis [16], and JMASS [17] at the mission level. However, current research exhibits the following disadvantages for existing model frameworks. Firstly, on the one hand, model frameworks lack platform-independent description; on the other hand, canonical and unified model specification has not been adopted for the implementation of model frameworks; these two aspects hamper the extensibility and composability of the model framework. Secondly,

the behavioral part of the model framework is mostly implemented as black boxes, so model reuse is not promoted and it is not flexible to revise the behavioral model. Thirdly, model frameworks lack domain-specific mechanisms (e.g., common modeling libraries and modules) for behavioral modeling in subdomains.

2.4 Unified simulation protocol-based method

The main purpose of this method is to support the interconnection and interoperability of different simulation systems, promote the reusability of simulation resources, and enable the composable development of simulation applications via a unified simulation protocol. The most prominent simulation protocol is HLA [15] proposed by Defense Modeling and Simulation Office (DMSO) in 2000 (followed by HLA Evolved in 2010 [18]). HLA is a general-purpose architecture for distributed computer simulation systems, which enables simulations systems interact with each other regardless of the computing platforms based on run-time infrastructure (RTI). HLA is widely used in interconnection of training simulators and simulation systems in the military field. It provides the model specification called federation object models (FOMs) and simulation object models (SOMs) which specify the object data flow and interaction message.

Essentially HLA is an interface model specification and lacks the description of model structure and behavior, and thus HLA lacks support for semantics-based model composition. Moreover, HLA-based method provides domain modeling support via integration of other tools and platforms. This integration causes problems such as difficulty for maintaining and inefficiency for the simulation run.

2.5 Summary

The categorization of four groups is neither absolute nor complete, and there is a trend to combine them for specific application purposes, for example, DEVS/HLA [8,19]. Table 1 presents an overview of the characteristics of these methods. These four methods contain mainstream M&S efforts based on typical technological spaces for effectiveness simulation, and try to cope with the multi-subsystem, multi-domain and multi-application scenario characteristics of effectiveness simulation by extending the existing technological spaces. Thus they are limited as far as the two modeling requirements are concerned, and a new method combining these efforts should be explored.

3 A model framework-based domain-specific composable modeling method

The literature review shows that a model framework is of vital importance to promote development efficiency and model reuse in different simulation applications. And the model framework can act as the integration infrastructure to compose models from multiple subdomains. However, as pointed out in Sect. 2.3, existing model frameworks mostly lack formal specification and DSM support. In this section, we propose a SMP2-compliant model framework-based domain-specific composable modeling method to meet the two modeling requirements. Firstly, we analyze how to exploit DSM for effectiveness simulation and identify that the composition of domain-specific models is a critical issue in CoSES; then we introduce a unified model framework which contains two subframeworks based on a domain partition of combat system behaviors; finally we propose a domain-specific composable modeling method which uses the model framework to compose diverse domain-specific models, and discuss the technical implementation of the method.

3.1 Domain-specific modeling for effectiveness simulation

Domain-specific modeling is proposed as a new software development paradigm in software engineering community. DSM mainly aims to raise the modeling abstraction level beyond current general programming languages by using DSML, to increase the productivity and enable the domain experts to take part in the software development by specifying the solution directly using problem domain concepts [20,21]. Computer simulation models technically are software models too, and thus can also adopt a DSM approach [22]. Currently M&S researchers usually study DSM for M&S from the following three aspects of concern: problem-oriented solution by constructing DSML or domain-specific

Table 1 A comparison of M&S methods for CoSES

	Simulation protocol	Model framework	Simulation model specification	M&S formalism
AIK incorporation	No support	Support	No support	No support
DSM support	None	Partly	None	Partly
Model composition and reuse	No support	Support	Support	Support
Model maintenance and extension	Hard	Normal	Easy	Easy
Behavioral description	No support	No support	Partly support	Support

modeling environment (DSME), generic tool extension to provide domain support, and adaptation of software engineering (especially model-driven engineering) techniques to enable DSM for M&S system development (more details of this categorization can be found in [22]).

Unlike the four technique-oriented methods discussed in Sect. 2, DSM adopts an application-oriented philosophy to provide an integrative solution tailored for a specific application domain. This application-oriented philosophy is supported by the comprehensive usage of existing modeling techniques and domain knowledge. Thus DSM provides a feasible technical solution to meet the requirement of multi-domain modeling. There have been plenty of research and practice for each subject domain in CoSES, and large amounts of models have been accumulated in each domain. This lays a sound foundation for domain engineering and enables DSM in each domain by extracting domain concepts for DSML construction from domain knowledge and developing code generators for the implementation of DSML semantics. DSM provides domain modelers with familiar modeling concepts and supports automatic generation of executable simulation models (or code) from domain-specific models, and thus it can raise the modeling abstraction level, promote model reusability, and increase productivity in each subject domain.

CoSES is a typical kind of complex systems which contain different subsystems across diverse subject domains. M&S of such complex systems needs to compose models from different subject domains to enable integrative simulation of the whole system. Different domains of complex systems are modeled by multiple DSMLs, so the syntax interoperation and even semantic composition of these DSMLs should be solved appropriately to enable model-based analysis and integrative simulation. Though DSML composition is intensively researched in software engineering field from the perspective of software system development, these research concentrate on solving the multi-view (aspect) modeling problem of software systems [22–24]. It should be noticed that software aspects usually refer to data structure, user interface, behavioral description and so on [23,24]. While simulation system development needs to model in broader subject domains not limited to computer science, and needs to cope with more complex situations for mechanism abstraction, model implementation and simulation execution.

3.2 A unified effectiveness simulation model framework

The construction of a model framework needs to not only understand and employ the domain knowledge of CoSES from the aspect of the knowledge engineering, but also grasp the technical implementation of the simulation models from the aspect of the domain engineering. Constructing a model framework should answer the following questions: (1) Which part of the knowledge is AVK? (2) How to extract the AIK and compose them into a model framework (3) Which part is AIK? And how to design extension points for subdomains and enable the domain modelers to use the AVK for DSM? and (4) How to generate implementation code of simulation models automatically from the model framework to promote development efficiency?

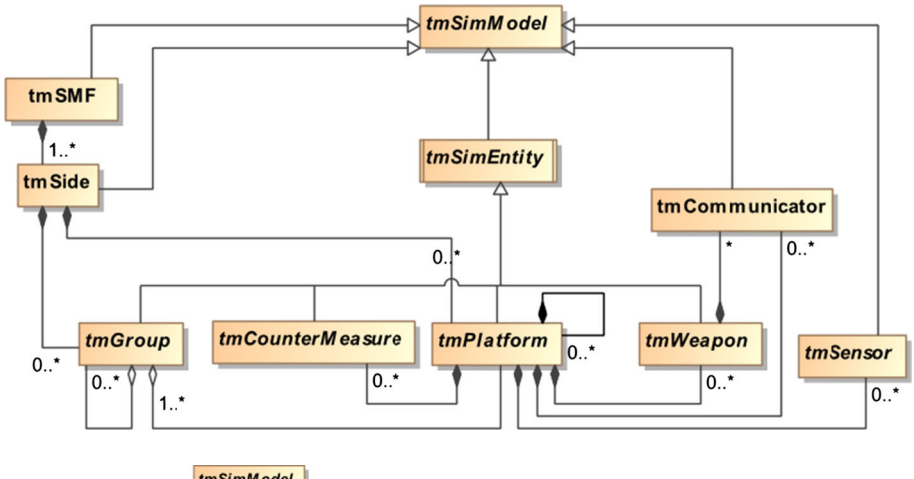
The behaviors of combat systems are usually conducted in the physical domain, the cognitive domain, the information domain and the social domain [25]. We do not study behaviors in the social domain in this research. Thus effectiveness simulation behavioral models are divided into two categories: combat equipment models which mainly describe behaviors in the physical and the information domain (physical behavior for short), and cognitive decision models which describe cognitive behaviors. Combat equipment models are built based on the physical and informational principles which remain invariant with the change of combat mission or situation. Moreover, the interfaces and relationships among these models are explicit and relatively stable. In contrast, cognitive decision models are more flexible since tactic decisions vary with the type of decision makers, combat missions, and theater situations. So these two kinds of models need to be built based on separate model frameworks using different modeling methods. In this subsection we introduce two submodel frameworks for physical domain models and cognitive domain models.

Since the cognitive model framework is built based on the physical model framework and finally seamlessly integrated into it, these two frameworks compose one unified model framework.

3.2.1 The physical model framework

To construct the physical model framework, we need to answer the aforementioned four questions. Firstly, the AIK in the physical domain includes the types of the equipments, the hierarchical structural relationship of the equipments, the performance attributes or parameters of each kind of equipment, and the interaction relationship patterns among the equipments.

Secondly, based on years of CoSES experiences of our research group [26], we explicitly describe these knowledge using unified modeling language (UML) and build a physical model framework. Figure 2 presents the top-level view of the physical model framework, which describes the main structure of the physical part of CoSES as follows. A simulation model framework (*tmSMF*) contains at least one force side (*tmSide*) which comprises force groups (*tmGroup*) and combat platforms (*tmPlatform*). The platform (*tmPlatform*) is at the heart of this framework, which can be aggregated into force groups (*tmGroup*) or directly belongs to force sides (*tmSide*), and it is the physical carrier for weapons



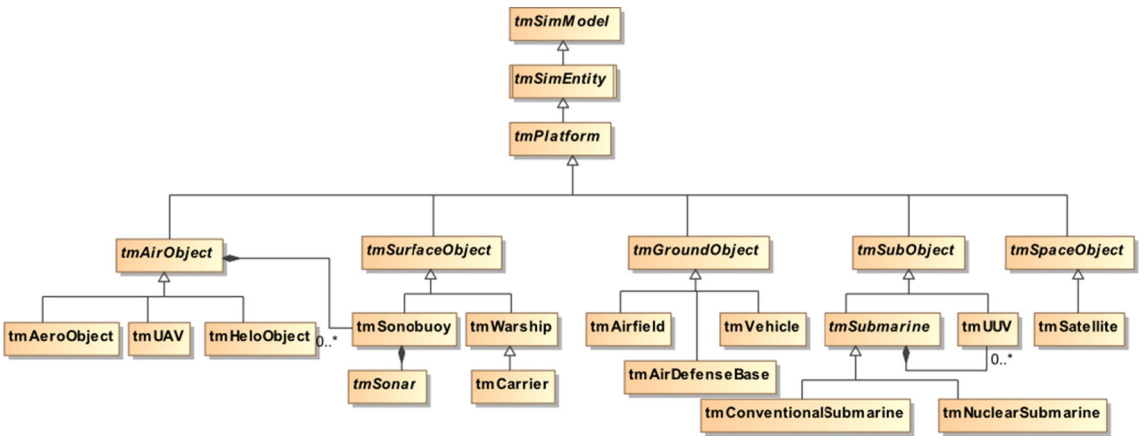


Fig. 2 The physical model framework for CoSES: top-level view

Fig. 3 A detailed view of the physical model framework: the combat platform model family

(*tmWeapon*), sensors (*tmSensor*), communicators (*tmCommunicator*), and countermeasures (*tmCountermeasure*). All equipment models inherit directly or indirectly from a virtual type model called "*tmSimModel*", which is a basic specification for all simulation models. Four kinds of equipment models (*tmGroup*, *tmCountmeasure*, *tmPlatform*, and *tmWeapon*) are modeled as moving entities, and thus they inherit from the model "*tmSimEntity*".

The abstract classes of the framework are further elaborated. For length limitation only the detail of combat platform models is presented in Fig. 3 as a typical example. Other kinds of equipments are elaborated in a similar way. Figure 3 shows typical combat platforms from the space, the air, the surface, the ground, and the underwater. For example, the airborne platforms (*tmAirObject*) are categorized as three kinds: fixed-wing aeroplanes (*tmAeroObject*), unmanned aerial vehicles (*tmUAV*), and helicopters (*tmHeloObject*). A platform can contain other platforms from the same category or another category, e.g., airborne platforms can be equipped with several sonobuoys (*tmSonobuoy*). A number of attributes and operations are added in each model to specify its basic structural and behavioral aspects, and they are hidden in the figure for conciseness.

Thirdly, the AVK of the framework comprises the values of the attributes, the concrete physical behavior of the equipment, the specific interaction relationships between equipments, and the structural formation of the forces. The framework only provides implementation interfaces as extension points for specific simulation applications. And the modelers can use their AVK to concretize the simulation models based on the model framework. For example, the values and the operation of the equipments are not specified in the model framework, and the modelers need to set the values and implement the operations for specific simulation applications.

Fourthly, a computer simulation model is a special kind of software, and it should be executed by a specific simulator. Thus simulation models should describe the simulator interface and other simulation-related aspects, and this description is often guided by a specific model specification. Although UML is a powerful language for the design of the model framework and it can explain the model framework in an explicit and well-understood manner, it is not a specialized simulation model specification. To implement the model framework and support automatic code generation, we need to transform the UML-based description of the physical

model framework to a SMP2-compliant model (its name is "catalogue" according to SMP2 standard [14]), and then executable code is automatically generated from that catalogue. As mentioned in Sect. 2.2, SMP2 is a MDA-compliant simulation model standard which promotes model composability and supports automatic platform-specific code generation from its platform-independent model. Thus we choose SMP2 as the implementation standard for the model framework. Using the model transformation (MT) techniques presented in [27], we transform the model framework into a SMP2 catalogue model which is a platform-independent model (as presented in the left part of Fig. 4).

The left part of Fig. 4 presents the top-level view of the catalogue, which describes the model components, their inheritance and reference relationship, and their interfaces of the SMP2-compliant CoSES model framework. The catalogue uses namespace as its categorization mechanism and the model framework can be divided into two groups of namespaces. The first group is transformed from the physical model framework as shown in Fig. 2, and this group includes namespace *CounterMeasure* (corresponding to *tmCounterMeasure* in Fig. 2), namespace Communicator (*tmCommunicator*), namespace Group (*tmGroup*), namespace Platform (*tmPlatform*), namespace Sensor (*tmSensor*), namespace Weapon (*tmWeapon*), and namespace *BaseModel* (*tmSMF*, *tmSide*, *tmSimModel* and *tmSimEntity*). The second group is transformed form the attributes and interaction relationships that are hidden in Fig. 2, and this group includes namespace *CommonDataType* and namespace Interaction. The namespace *CommonDataType* specifies all data types of the simulation models. The namespace Interaction specifies the event types and the interface types of the simulation models. Figure 4 also shows a detailed view of namespace platform which corresponds to the platform model family presented in Fig. 3.

Currently SMP2 supports C++ code generation [14], and the framework code is automatically generated from the catalogue as shown in the right part of Fig. 4. For example, the user chooses the model *tmPlatform* and click on the code generation, and then a series of C++ code is generated automatically. Other parts of the model framework are implemented as C++ code in the same way. Finally all components of the model framework are built as dynamic linked library (DLL) files, which can be executed by the SMP2 simulator. The generated code for each model is the skeleton of model implementation. The right part of Fig. 4 only shows part of the code skeleton for length limitation. The generated code contains the following parts: SMP2-compliant framework code, references of other files, model parameters (e.g., *CruiseSpeed*), model operations (e.g., *GetCruiseSpeed*()),

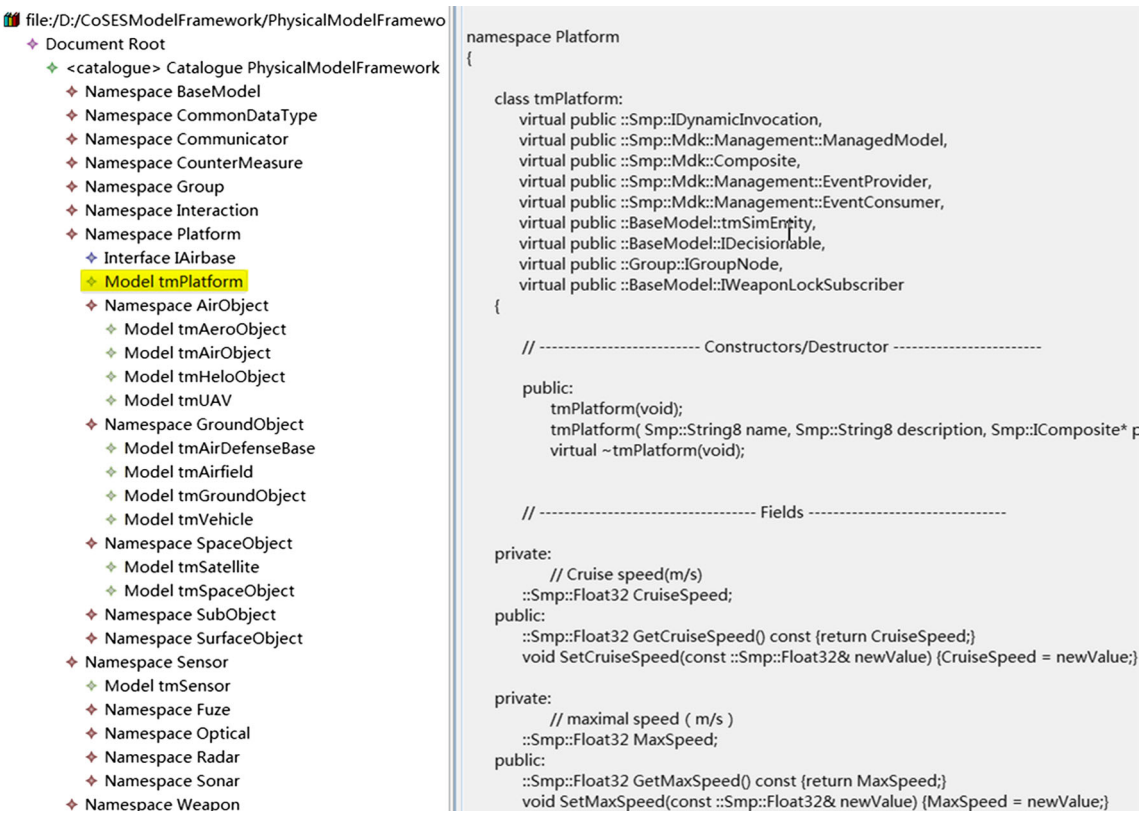


Fig. 4 An overview of SMP2-compliant model framework and a code generation example of model *tmPlatform*

structural container relationship (e.g., the platform model contain a weapon list which can integrate weapon models), interfaces between models, and interfaces for the simulator. As discussed in the third question, the modelers need to finally implement the model by specifying its concrete structure and behavior based on this code skeleton.

3.2.2 The cognitive model framework

The construction of the cognitive model framework also needs to answer the aforementioned four questions. Actually we had explicitly studied decision modeling in our previous work [5,6]. In this paper we reorganize the relevant work and present it in a new manner to answer the four questions. In this work, behavior modeling in the cognitive domain is mainly referred to as decision modeling which concentrates on combat platform decision modeling at the engagement level, and we mainly study how to model the tactics of main combat platforms.

Firstly, the problem analysis was performed on the cognitive behavior of combat platforms [5]. We found platform tactics essentially are a series of "IF…THEN…" specifications to specify how the platform should behave under certain circumstances. IF stands for the situation space of all conditions where the platform confronts, and THEN stands for the order space of decision choices which the platform should perform according to the concrete situation. So the AIK for decision modeling contains the items of the situation space and the order space (e.g., typical tasks and phases of the platform), and the combination rules and patterns of these items to form tactic rules.

Secondly, since the items of the order space correspond to those of the situation space, the pivotal problem of decision modeling is how to describe the situation space. Moreover, the cognitive model is different from the physical model. The cognitive model should be designed and implemented using another technological space. Based on the problem analysis and the characteristics of decision modeling, we designed the script-based cognitive model framework of the simulation system (as shown in Fig. 5). Each kind of combat platform models (*tmPlatform*) has one main task script (*MainTask*) which specifies the basic decision process, and the platform model will call the main task script at each decision time step via *StepDecision* function of *tmPlatform*. *tmPlatform* and *MainTask* are navigable to each other, and *MainTask* can visit corresponding interface functions and variables. The target information is stored in a target list (*tcTarget*). When a new target is detected, *MainTask* will arrange a *TargetTactic* to cope with the new target via an interface function *SetTask()* of tcPlatform. At each decision time step *tmPlatform* will call *MainTask* to traverse *tcTarget* and perform the *TargetTactic* for each target. The script-based framework provides three mechanisms to describe the timing information, decision status and condition of decision models: *Timer*, *MemoryVariable* and *ScriptEvent*. *Timer* mechanism and *ScriptEvent* mechanism are managed by *tmPlatfom* and valid in all scripts of the platform model. *MemoryVariable* is managed by *tcTarget* and valid in the corresponding *TargetTactic* script. The *ScriptEvent* is inherited by the *ExternalEvent* which contains three kinds of events: *SimulationEvent*, *GuardEvent* and *TimerEvent*.

Thirdly, the model framework only provides the types of items of the situation space and the order space. AVK comprises the knowledge to combine the items from the situation space and the order space based on AIK to form specific tactical rules in a certain combat scenario. Thus the modelers can use their AVK to set the values for each item (if there is any attribute), specify the combination of each rule, and assemble these rules to form a specific decision model.

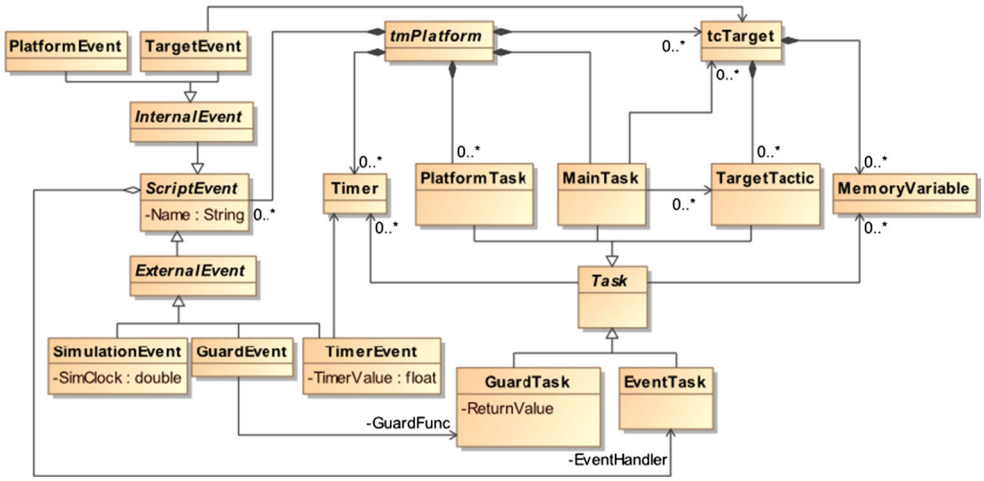


Fig. 5 A script-based cognitive model framework [6]

Fourthly, the model framework is implemented using python and these decision models can be interpreted by Python interpreters directly.

3.2.3 The unification of the physical model framework and the cognitive model framework

The cognitive model framework is developed based on the physical model framework and these two frameworks are combined together by pre-built interfaces. At the code level, we implement these interfaces as a series of Python/C++ interface functions, which enable the interaction of decision models and physical domain models. On the one hand, the decision model mainly inquires three categories of information from the physical domain model: platform parameters (including parameters of subsystems of the platform, e.g., weapon range), current platform status (e.g., position, velocity), the information of targets and the ally. On the other hand, the physical domain model receives orders from the decision model: movement, subsystem (including weapon, sensor, communicator, and jammer) management and formation control. There is also interface support for the decision model to use *Timer*, *MemoryVariable* and *ScriptEvent* mechanisms.

For each simulation run, the implementation of the unification of the physical model and the cognitive model is as follows. Firstly, the C++-based physical model calls the Python-based cognitive model through the function *StepDecision* and other interfaces as discussed in Sect. 3.2.3. Then the Python interpreter executes the cognitive model (Python scripts) to calculate the decision process. This calculation relies on the aforementioned pre-built interfaces. The interpretation of the python scripts only execute the code directly supported by the interpreter, while other parts are actually executed through C++ code implemented in the physical model through the pre-built interfaces. That is to say, the atomic decision action units are implemented in C++ physical model, and the decision model mainly describe the decision logic process, set the condition for action units and organize them in sequence.

3.3 A domain-specific composable modeling method and its technical implementation solution

As discussed in the two subsections above, DSM can effectively supports multi-domain modeling, and the model framework provides the infrastructure for reusing models in different simulation applications and composing models of different subsystems across multiple domains. In this subsection, we firstly propose a domain-specific composable modeling method based on the unified effectiveness simulation model framework and our previous work on modeling methods for complex simulation systems [4], and then present a technical implementation solution of the method.

3.3.1 The domain-specific composable modeling method

We had proposed a DSM-based multi-paradigm modeling methodology to enable formal and automated model development lifecycle for complex simulation systems in [4]. This methodology comprises the following four phases: The first phase is DSM-based conceptual modeling, and this phase mainly uses DSM techniques to construct a domain-specific and formal conceptual model; the second phase is model framework-based model architecting and integration, and this phase mainly employs the model framework for overall model design and integration for a simulation application; the third phase is M&S formalism-based model design and formal analysis, and this phase uses formalisms to support domain-specific model design and analysis; and the fourth phase is SMP2/C++ transformation based model implementation, and this phase generate C++ code from the model framework using a C++ code generator for SMP2. Since CoSES is a typical kind of complex simulation systems, we propose a model framework-based domain-specific composable modeling method (as shown in Fig. 6) based on this methodology and comprehensive utilization of the research fruits of current methods.

The proposed method comprises four layers: the model framework layer (supports the second phase of the methodology [4]), the behavior representation layer (supports the fourth phase), the simulation modeling formalism layer (support the third phase), and the DSML layer (support the first phase). The first three layers are mainly used to model the combat behaviors at three abstraction levels, and these behavioral models are based on the entity structure and interaction relationship specified by the model framework layer.

(1) As discussed in Sect. 3.2, the SMP2-compliant model framework layer comprises two parts which are connected by a cognitive decision interface framework. The interface framework is technically comprises of a set of Python/C++ transformation functions. The model framework layer plays a dual role in the whole model development process: On the one hand, the framework is the overall design scheme for the whole simulation system and it provides the top-level architecture (structural constraints for each DSML as presented in Fig. 6) for the design of subsystem models and subdomain models; on the other hand, the framework acts as the integration infrastructure to integrate subsystem models and subdomain models, and the behavioral code generated by these models can be embedded into the structural code generated by the model framework.

(2) The behavioral representation layer, i.e., the model implementation layer, uses C++ to implement SMP2-compliant physical domain models and Python to implement cognitive domain models. The physical domain model is implemented using C++ since physical behaviors are relatively stable and can be hard-coded in C++. The decision script framework is implemented using Python since decision behavior is flexible and model revisions take effect immediately without rebuilding. Eventually, all Python code are transformed to C++ code for integrative simulation in SMP2 environment.

(3) The formalism layer utilizes common M&S formalisms to provide behavioral semantics for domain-specific models. Figure 6 lists a series of common formalisms to describe certain aspects of the behaviors of different combat equipments. Generally speaking, formalisms such as statecharts [28], Modelica [29], and block diagram [30] can be used to describe state transition and movement aspects in the physical domain, while event graph, activity diagram, and Petri Nets are used in cognitive domain for to model event triggers of tactic decisions, decision activities and processes. However, the listed formalisms are neither complete nor absolute, and other formalisms like DEVS, Markov Chains can be also used. The formalism layer not only provides support for model design, but also helps the modelers to analyze the models across different domains using formal methods and enables early verification and validation.

(4) The DSML layer provides DSME and DSML for typical CoSES domains such as missiles, radars, warships and so on. A DSML metamodel uses the corresponding structural information in the CoSES model framework to design its structural aspect and chooses appropriate M&S formalisms based on its domain behavioral patterns to design its behavioral aspect. For example, the platform decision DSML can use the Command and Control (C2) interface in the framework to design structural coupling elements and use Petri Nets to model its decision process [5]. This layer provides the modeler with domain-specific languages and environments, which facilitate problem description and conceptual modeling at a higher abstraction level. Moreover, combined with its semantic anchoring mechanism and code generation infrastructure, this layer supports formal representation and automatic implementation of conceptual models.

These four layers conforms to the four phases of the multi-paradigm modeling methodology [4]. This method provides formal representations for products of all model development phases, thus it enables model reuse and composable simulation at four layers. The transformation from higher-level models to lower-level implementations is automated by model transformation and code generation. So this method promotes development efficiency of effectiveness simulation applications and supports flexible, high-level and generative modeling.

This method adopts the principle of model architecture-based overall model design, multi-paradigm-based domain-specific detailed model design, formalism-based formal model analysis, automated model implementation, model framework-based model implementation and composable simulation. Thus it provides an integrative solution for composable and multi-domain modeling. As discussed in Sect. 2.5, this method has integrated the research fruits of M&S formalisms, simulation model specification and model framework to support model design, implementation and integration, respectively. According to Fig. 1, only simula-

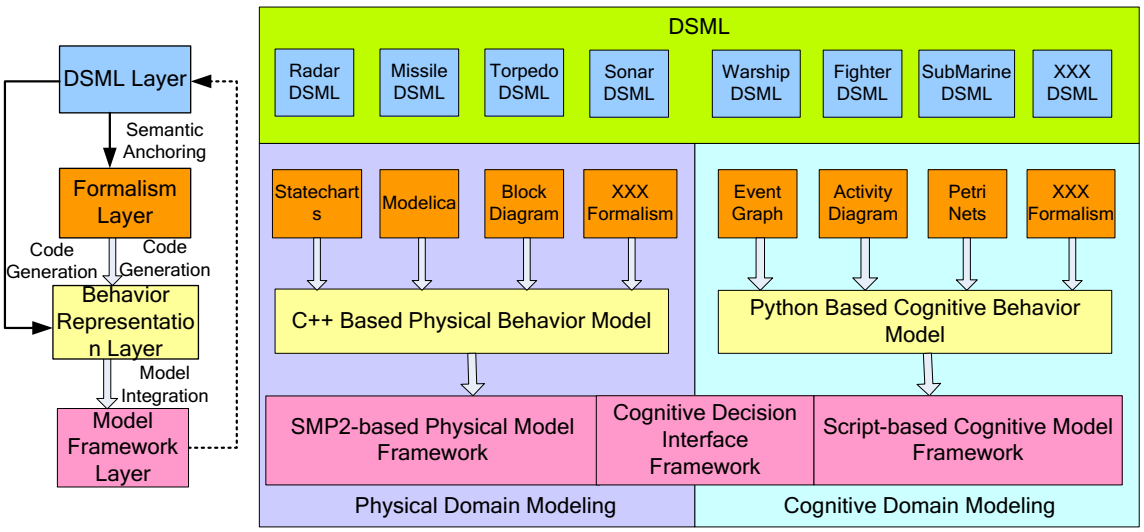


Fig. 6 A model framework-based domain-specific composable modeling method for CoSES (adapted from [4])

tion protocol is not researched in this work, the reason for this is that distributed simulation is outside the research scope of this work.

3.3.2 Technical implementation

The proposed modeling method provides a collaborative platform for different stakeholders of effectiveness simulation application, and to use this method for effectiveness simulation needs the joint efforts from domain experts, domain modelers, M&S experts, formalism modelers, and software engineers. The technical implementation solution for the proposed method is presented in Fig. 7, including the following essential steps.

(1) Based on domain knowledge and M&S experience a SMP2-compliant model framework (a catalogue file) is constructed as the backbone of CoSES system, which was discussed in Sect. 3.2. The framework construction relies on the joint efforts from the M&S experts and the domain experts of combat equipments to identify the AIK and the AVK in CoSES domain. As also mentioned in Sect. 3.2, C++ code is generated from the model framework to implement the basic entity hierarchy and interaction interfaces of the whole simulation system.
(2) According to language engineering of DSML, it comprises abstract syntax (including constraints), concrete syntax, and semantics [31–33]. The metamodel of a DSML is designed based on two knowledge sources: The first is the domain knowledge from domain experts, and the second is the structural information (includes entity formation, attributes, parameters and interaction interfaces) from the model framework. Moreover, the behavioral aspect of the metamodel is designed using one or more simulation modeling formalisms suggested by M&S experts according to their experience and expertise. Then the concrete syntax is attached to each element of the metamodel, and a DSME is generated based on the built metamodel using metamodeling platforms such as AToM^3 [34] and generic modeling environment (GME) [35].
(3) Domain modelers build conceptual models with DSML in DSME. These models are comprised of domain-specific entities and relationships which are familiar to the problem owners. Since all DSMLs are based on the same framework, the domain-specific models built using these DSMLs have the common foundation for model composition. The semantics of DSML is achieved either by semantic anchoring which transforms domain-specific models to formalism-based models based on the metamodel mapping relationship, or by code generation which generates executable code from domain-specific models using code generators developed by software engineers.
(4) Modelers who are familiar with certain M&S formalisms can design simulation models based on selected for-

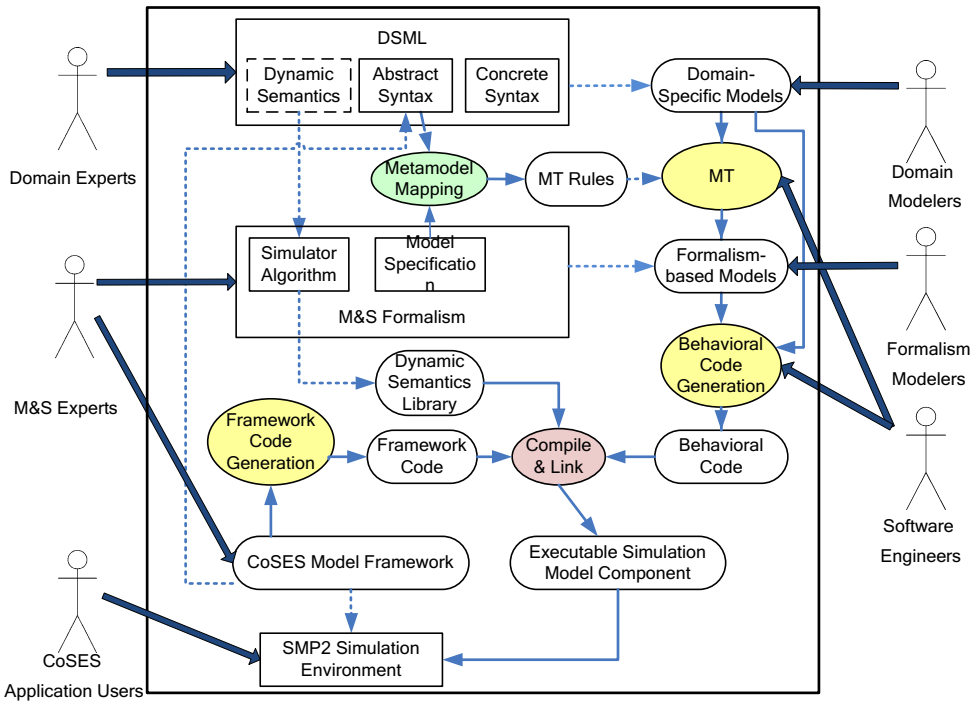


Fig. 7 A technical implementation solution for the composable modeling method

malisms using their supporting tools (for example, CPN tools for Petri Nets). Then formal analysis and early verification and validation can be performed on formalism-based models which can be either built by the modelers or transformed from domain-specific models. We can transform domain-specific models from different domains to the same formalism and perform joint formal analysis and achieve semantic composability of these models at the formalism level. Usually, the M&S experts and software engineers have developed code generators for common formalism-compliant platforms which generate behavioral code from formalism-based models.

(5) The behavioral code generated from the formalism-based models or domain-specific models is combined with the framework code generated from the SMP2-compliant model framework, and these two kinds of code are built together as executable simulation model components. Then these components are stored in an executable simulation model component library. Since the behavioral aspects of the models are based on the structural parts specified by the model framework, these two kinds of code are seamlessly integrated into one executable model.

(6) Effectiveness simulation application users select the components and compose them into an assembly file based on the catalogue file. Also they need to construct concrete simulation scenarios and design of experiment files to configure the simulation system. Finally they run the application specified by the three aforementioned files in the integrative SMP2 effectiveness simulation environment, collect and analyze the simulation results.

4 Domain-specific composable modeling in typical CoSES domains

As stated in Sect. 3.1, the composition of domain-specific models from different subject domains is a critical issue for M&S of complex systems. According to the levels of the conceptual interoperability model [36], higher levels of interoperability are more demanding for model development and execution lifecycle. The proposed method supports conceptual interoperability for the following reasons: Firstly, the SMP2/C++ implementation of domain-specific models provides the syntactic interoperability; secondly, the structural aspect of DSMLs is designed based on the unified model framework which provides the structural semantics; thirdly, the behavioral aspect of DSMLs is constructed based on the formalisms which provide explicit behavioral semantics; fourthly, based on the three kinds of interoperability support, DSM-based conceptual modeling building domain-specific models in different subject domains can achieve conceptual interoperability.

According to the top-level view of the model framework as presented in Fig. 2, typical subject domains in CoSES contain the combat platform, the C2 system, the sensor, the weapon, the communicator, and the countermeasure. The combat platform model is a fundamental element in the model framework since it is the physical carrier of other affiliate equipment models and the C2 system model. From the technical aspect of composable modeling, we can categorize these domains into three groups: the platform domain, the affiliate equipment domains (including sensors, weapons, countermeasures) and the decision domain.

The combat platform model is the backbone of the CoSES model framework which integrates affiliate equipment models and its decision model. However, the structural complexity of the model framework is not dealt with by DSM-based behavioral modeling, but by a code generator for CoSES catalogue. This code generator generates framework code such as simulator interfaces, interfaces for submodels and other infrastructure implementation code from the model *tmPlatform* presented in Fig. 4. Thus the main task left for DSM is to model the dynamic behavior based on the generated structure.

In this section, we concentrate on using DSM method for platform modeling, decision modeling and missile modeling (as a typical example for the modeling of affiliate equipments) for length limitation. Firstly, the general DSM process for CoSES domains is analyzed, and then DSM researches in these three subdomains are explicitly presented following the general process.

4.1 DSM process using GME tool suite

Since DSM tools are built based on DSM methodology and state-of-the-art technologies, they have exhibited promising potentials for simulation modeling. Among all DSM platforms, GME is considered as the centerpiece modeling technology for Model Integrated Computing (MIC) [35]. GME is an open-sourced DSM tool suite with infrastructural support for metamodeling, DSME generation and interpreter construction. In this work we choose GME as the DSM platform. A DSML usually contains abstract syntax, concrete syntax and semantics, and thus DSM for CoSES using GME usually includes the following steps:

(1) *Metamodeling the DSML abstract syntax based on domain analysis and the model framework* Currently metamodeling is the mainstream method to specify the abstract syntax of a DSML since it is concise, formal and supports model-driven development of modeling languages [37]. The metamodeling language of GME is MetaGME, which can be transformed to Meta-Object Facility (MOF) [38]. Metamodeling should be based

on domain analysis to extract typical behavior patterns which can be formally described by behavioral formalisms. Unlike DSM in software engineering field, DSM for CoSES mainly model the behavior, so metamodeling depends on the interfaces and entity elements specified by the SMP2 catalogue and the cognitive modeling framework.

(2) *Metamodel-based DSME generation* Then a DSME is generated based on the DSML metamodel using GME meta-interpreter and the DSML designers attach domain-specific icons to the elements of the metamodel to specify the concrete syntax, which promotes the user friendliness for domain modelers.

(3) *Denotational Semantics implementation by model transformation* Since the DSML is designed based on M&S formalisms, its semantics can be implemented by model transformation transforming the domain-specific models to formalism-based models. Semantic anchoring is proposed to achieve DSML semantics via transforming them to a set of "semantic units", and a tool suit is built to support semantic anchoring [39]. In this tool suit, the Graph Rewriting and Transformation Language (GReAT) [40] is a graph transformation language and a toolset to specify and execute model transformation based on mapping relationship between metamodels. GReAT is incorporated into the GME tool suite. This step is optional and only necessary if we choose to implement the DSML semantics based on certain formalisms or perform formalism-based analysis.

(4) *Operational semantics implementation by code generation* GME provides a Builder Object Network (BON)-based framework for interpreter construction using Visual Studio. The users can use the BON interpreter wizard to automatically generate the framework code, and implement the code generation algorithms in function *InvokeEx*() by C++ coding in Visual Studio [35]. The framework code provides the GME interface implementation, fundamental code generator methods, and other implementation supports, and thus the users can concentrate on only the implementation of code generation rules. After the coding, the C++ project is built as a DLL file, and then the DLL is registered as an interpreter component to GME platform. This registered interpreter is used as a code generator for domain-specific models.

4.2 DSM in the platform decision domain

We had explicitly studied DSM in platform decision domain in the work presented in [5]. In this article we briefly introduce these four steps and how they are incorporated into the proposed composable modeling method. (1) *DSML Metamodeling*. A domain analysis is performed to explore the decision modeling, and the decision model is divided into a condition space and an order space. The two spaces have three dimensions each and the number of combinations of these six dimensions is quite huge. Thus we proposed a decision modeling scheme using a phase-based reorganization of the two spaces. And a metamodel following this scheme is constructed using GME, which imports structural information from the model framework. (2) *Metamodel-based DSME generation*. After attaching domain-familiar icons to the modeling elements of the metamodel, a DSME is automatically generated which is similar to Fig. 9. (3) *Denotational semantics implementation by model transformation*. A series of formalisms are introduced to provide denotational semantics for the decision DSML, and these formalisms support formal analysis methods such as temporal analysis, rule reasoning, probability analysis, state analysis and concurrency analysis. Then Petri Net is chosen as the example formalism to illustrate how to transform decision DSML to formalisms using GReAT. Formal analysis using the Petri Net decision model is presented in the case study. (4) *Operational semantics implementation by code generation.* Using the BON infrastructure provided by GME, a code generator is built to generate Python scripts from domain-specific decision models. And the Python scripts are incorporated into the CoSES system via the cognitive decision interface framework as shown in Fig. 6. Please refer to [5] for more details on DSM in decision domain.

4.3 DSM in the missile domain

DSM in the missile domain models physical behavior of the missile which is different from cognitive behavior of platform decision. Thus we concentrate on the metamodeling, DSME generation, and C++ code generation, and model transformation to formalisms is omitted.

(1) *Metamodeling the DSML abstract syntax*

The key to missile behavior modeling is the differentiation of its movement phases and the kinematic equations for each phase. Figure 8 shows a top-level view of the DSML metamodel for missile behavior modeling. The main purpose of this DSML is to specify its movement phase based on statecharts formalism [28] and kinematic equations which describe the kinematic characteristics for each phase based on the causal block diagram formalism [30]. Unlike statecharts, the movement phase is enumerated and the user can only select among the specified phases (e.g., *ReadyToLaunchP*, *TargetGuidanceP* phase). Kinematic equations modeling is the same case, since the kinds of kinematic variables (*KinematicVariable*) and the parameter (*Parameter*) are also tailored for the platform. The *PhaseTransition* element is similar to the *Condition* element in the platform decision DSML, and it uses *LogicalOperator* and *GuardExperssions* to specify phase transitions, and it uses *ExternalInstruction*

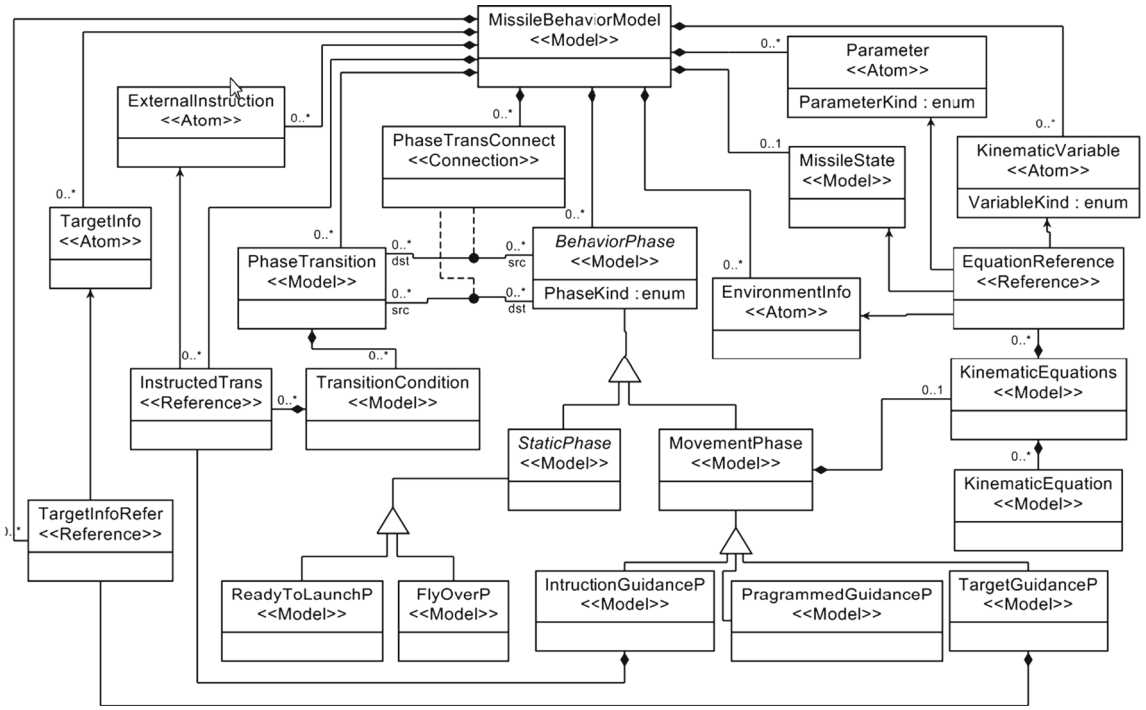


Fig. 8 A metamodel of the DSML for missile behavior modeling (*top-level view*)

as one kind of *EventTriggers* for *GuardExpression*. For conciseness the detail of *PhaseTransition* element is omitted. The DSML imports two kinds of structural elements from the catalogue. The first is interfaces, including *TargetInfo* as the interface to acquire target information, *ExternalInstruction* as the interface to receive instructions from the platform, and *EnvironmentInfo* to acquire environment information. The second is internal structure of the missile model and they are described as fields in the *tmMissile* model of the catalogue. These attributes include *MissileState*, *Parameter*, *KinematicVariable*.

(2) *DSME generation*

After metamodeling, we design the concrete syntax by attaching domain-familiar icons to the modeling elements. Then from the metamodel the DSME is automatically generated as shown in Fig. 9. The DSME is a graphical editor with domain-specific concepts which are familiar to missile modelers. It is not error-prone since the abstract syntax and constraints guide the user to model legally and help them to locate mistakes by using its "Check" function. The modelers construct the missile behavior model by dragging icons, connecting them, selecting the enumerated attribute, and setting the attribute values. In this DSME, no manual coding is necessary. The GME-based DSME provides hierarchical modeling views according to the structural relationship of the model elements. For example, if you double click the *LaunchOrder* (an instance of *PhaseTransition*) icon in Fig. 9, a new modeling view will pop up with *PhaseTransition* modeling elements (*Conditions of Transitions*). The models can be saved as Extensible Markup Language (XML) files to support file interchange between GME and other tools.

(3) *C++ Code generation*

Missile behavior model implementation relies heavily on structural code, which is automatically generated from the model *tmMissile* of the catalogue. The structural code has implemented most of the missile model, except the *StepMotion*() function which specifies its behavior for each simulation time step. So the code generator should generate a "*MissileBehavior.h"* C++ header file and a "MissileBehavior.cpp" C++ source file to implement domain-specific behavior models. The code generation contains the following steps: (1) Build a header file and a C++ source file (.cpp file); (2) Generate the reference code in each file to make appropriate references to the framework code files and define the variables and functions; (3) Generate code to implement the phase transition behavior in the *StepMotion*() function. (4) For each movement phase, generate the code to implement the kinematic equations. The interfaces, *MissileState*, *Parameter*, *KinematicVariable*, and *BehaviorPhase* have been all defined and implemented in the framework code, and thus the code generator should generate the same names as theirs in other files. After the code generation, the two generated files are grouped with other files, which are built into one DLL file as an executable missile model.

GME provides infrastructural support for building interpreters (in this research we mean code generator). We use

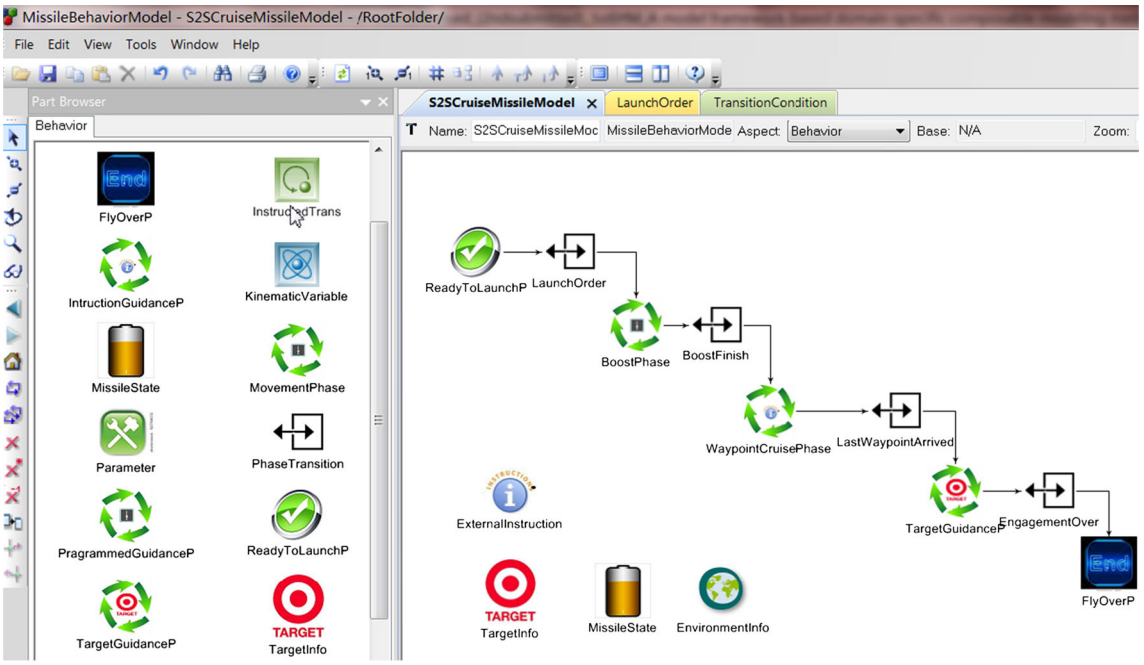


Fig. 9 A DSME for missile behavioral modeling using GME

the BON interpreter wizard to automatically generate the framework code and implement the abovementioned algorithms in function *InvokeEx()* in C++ using Visual Studio [35]. The implementation includes three parts: the first part is node traversal which is done by the functions provided by the framework, the second part is the code generation algorithms for each kind of nodes, and the third part is other implementation specific code (e.g., reference to other files). After the coding, we build the code as a DLL file, register the DLL as an interpreter component to GME. This interpreter generates C++ code automatically from the graphic missile models.

4.4 DSM in the platform domain

The key to combat behavior modeling is to model the behavior of the combat platform, since the platform is the physical carrier of all equipments and other combat behaviors are mostly coordinated or controlled by the platform. Platform behavior contains three parts: platform movement, platform decision, and affiliate equipment behavior. For combat platform model implementation, decision behavior is the most complicated part, which is independently described in platform decision model as discussed in Sect. 4.2; affiliate equipment behavior is implemented in affiliate equipment models; thus only movement description is left for the combat platform model.

In this subsection, we study DSM in the platform domain and concentrate on the metamodeling of the platform DSML. DSME generation and code generator implementation are similar to the Missile DSML, and they are omitted for length limitation. The top level view of the platform DSML metamodel is shown in Fig. 10. Its movement part is almost based on the same

formalism as the Missile DSML. However, the movement phases are more complicated than missiles since platforms need to fulfill multiple tasks. The move phase (*MovePhase*) is divided in to five groups: startup phases (*StartupP*), waypoint phases (*WaypointP*), target track phases (*TargetTrackP*), threat avoid phases (*ThreatAviodP*), and maneuver phases (*ManeuverP*). Moreover, the decision models (the model proxy called *PlatformDecisionModel*) and other affiliate equipment models (the model proxy called *SubEquipModel*) contained in the platform model influence the platform movement. If the platform model contains a decision model, the movement phase transition is usually controlled by the decision model (the *PlatformDecisionModel* can control phase transition since the *TransitionCondition* refer to *DecisionTrans*). Otherwise, the modeler should specify the transition conditions. Similar to the missile DSML, the platform DSML also imports two kinds of structural elements from the catalogue.

5 Case study

CoSES is usually used to evaluate system attribute alternatives (or tactics alternatives) under the background of SoS counterwork. Warship sea battle is a typical combat pattern which involves a diversity of combat entities. In the sea bat-

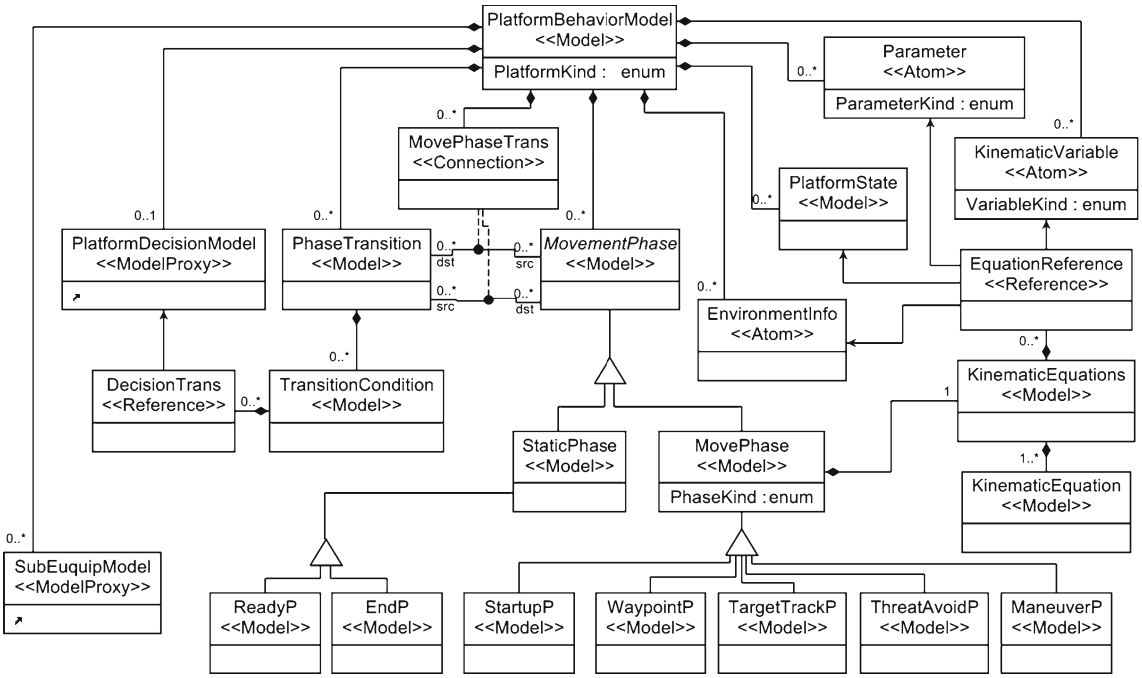


Fig. 10 A metamodel of the DSML for platform behavior modeling: *top-level view*

tle, the warship confronts a diversity of threats concurrently: in the air, there are the fighters and air-to-surface missiles; on the surface, there are enemy warships with artilleries and surface-to-surface missiles; and there are underwater submarines with torpedoes. The warship needs to cope with complex situations, survive under these multiple threats and achieve the combat goals. So how to measure the system effectiveness of the warship in typical combat scenarios is a tough issue.

5.1 Basic scenario overview

The basic scenario is as follows. There are two opposed side in the theater: the Red side comprises a satellite and a warship with sensors (including radars and sonars) and weapons. The main task of the warship is to destroy the Blue warship and survive in the counterwork against the Blue side; the Blue side has a submarine, a fighter and a warship which are deployed near the waypoints to fight against the warship and equipped with sensors and anti-ship weapons. The initial status is that the red satellite sends a set of waypoints (the position of the target warship as the last waypoint) to the Red warship, and the Red warship starts moving toward the first waypoint, and the platforms of the Blue side are patrolling along the specified areas around the waypoints.

The warship effectiveness simulation involves lots of simulation models, such as platform models and decision models for warships, fighters, and submarines, and the affiliate equipment models of the platforms (e.g., the surveillance radar model, the surface-to-surface missile model). We had implemented all the aforementioned models using manual coding in the previous engineering practice. And we found manual coding is taxing and inefficient. In this section, we use the proposed modeling method to rebuild the warship platform model, the warship decision model, and surface-to-surface missile model.

5.2 DSM-based warship decision modeling [5]

The top-level decision process of the Blue warship is illustrated in Fig. 11 using the DSML presented in Sect. 4.2: the warship is in Waypoint phase after the waypoints are assigned to it during initialization; it will transit to *TargetProcessPhase* if there is any target detected; for each new target, the warship arranges a task to cope with this target according to its type (air, surface or underwater); the warship will go back to *WaypointPhase* again after all targets are processed; the warship terminates its sea battle process when it reaches the last waypoint. The task to cope with the corresponding target is concurrent and the warship can perform several heterogeneous tasks simultaneously. Figure 12 describes the warship anti-surface decision process (*WarshipAntiSurface* task in Fig. 11). The other two subtasks are presented in [5]. The three decision processes share a similar decision logic: When the target is not threatening (far enough from the warship), the warship is in *TargetProcessPhase* to travel to the waypoint while keeping a certain distance from the target, and if the target becomes a threat, the warship transits to *Evasion-*

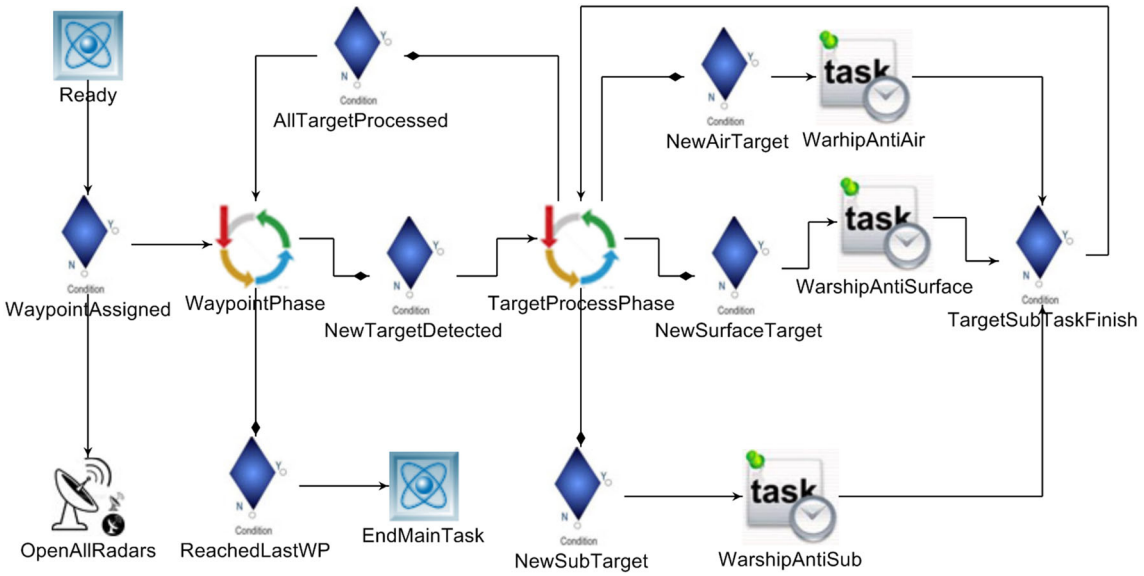


Fig. 11 The overall decision process model of warship sea battle (adapted from [5])

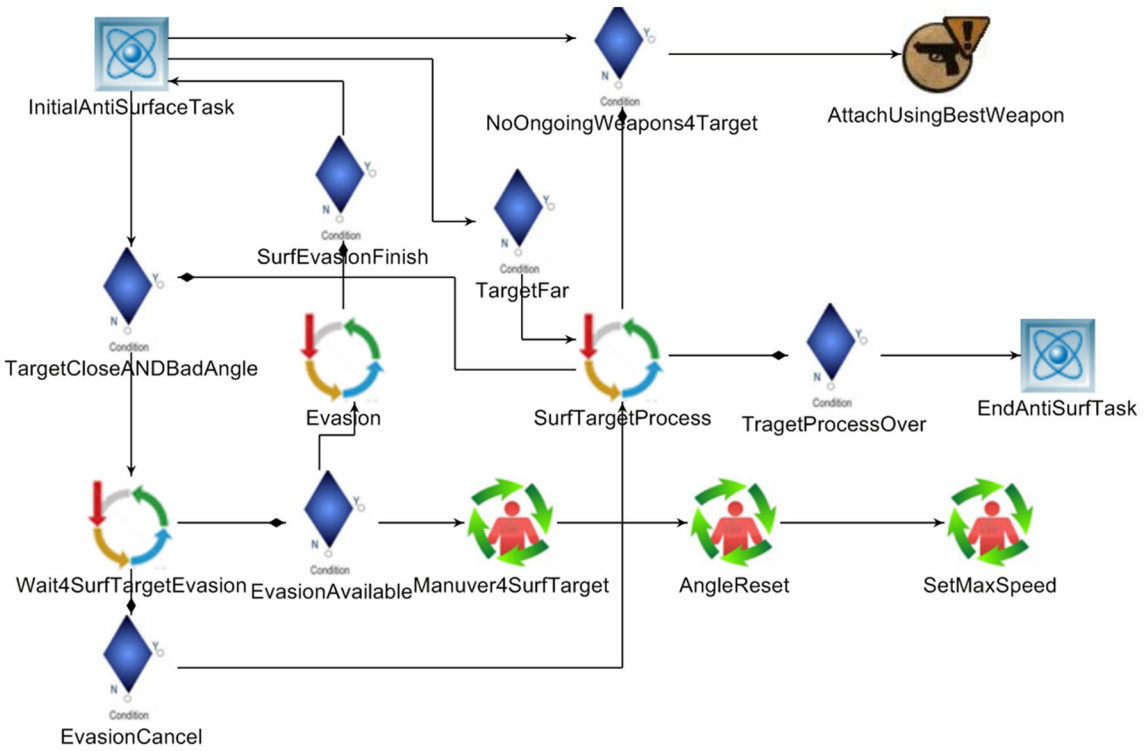


Fig. 12 The decision process model of warship anti-surface task

Phase to take emergency actions to prevent situation being worse; once the target gets out of the dangerous scope, the phase goes back to *TargetProcessPhase* again. The decision model is relatively simple and it can only execute a single task for each kind of target.

Our previous work presented in [5] had used transform the warship sea battle decision model to a Petri Net model. There are three Evasion places in the model which refer to the same place, so we modify it by using one *EvasionPhase* place instead while preserving all the links of them. Petri Net has a sound mathematical foundation and supports several formal analysis methods, and thus formal analysis and early verification and validation can be performed based on that Petri Net model. For example, the concurrency analysis of

Table 2 Simulation results using different models in typical scenarios

Experiment no.	1	2	3	4	5	6	7	8
Scenario	1	1	1	1	2	2	2	2
Strategy	S1	S1	S2	S2	S1	S1	S2	S2
Model	1	2	1	2	1	2	1	2
BWDamageRate	0.588	0.587	0.591	0.592	0.598	0.598	0.601	0.601
RWDamageRate	0.685	0.686	0.603	0.603	0.556	0.557	0.543	0.543
DamageRatio	0.858	0.855	0.980	0.981	1.075	1.073	1.106	1.106

the anti-air, the anti-sub, and the anti-surface tasks, and the conflict analysis of the *EvasionPhase* is performed using this Petri Net model (see [5] for details).

5.3 DSM-based behavior modeling of the surface-anti-surface cruise missile (SASCM)

Similar to the decision model, SASCM model is built using the DSM method. The behavior modeling for SASCM is not as tricky as decision modeling, since the movement phases are stable and the kinematic equations are explicit. The behavior model of SASCM is shown in Fig. 9, which includes five behavioral phases: *ReadyToLaunch* phase, *Boost* phase, *WaypointCruise* phase, *TargetGuidance* phase, and *FlyOver* phase. The kinematic equations are specified for each phase, and the transition conditions are also prescribed inside each transition. These phases and transition can refer to *MissileState*, *EnvironmentInfo*, *ExternalIntruction* and *TargetInfo*. A BON-based code generator is constructed (similar to the Python code generator) to generate C++ code, which is incorporated into the SASCM C++ project to build a DLL file.

5.4 Platform-centered composable modeling

Platform behavior modeling is easier than decision modeling, since movement phase transitions are controlled by the decision model and the platform movement model only needs to specify the kinematic equations for each phase. From the domain-specific model for the platform behavior a C++ header file and a source file is generated, which implement the *StepMotion()* function in the framework code, and these two files together with other framework code files are built as a DLL file for the warship platform model. The warship platform decision script is dynamically executed during the simulation through the *StepDecision()* function in the C++ platform model. Actually this platform-based composition method is useful for other simulation applications.

5.5 Scenario design and simulation analysis

As shown in Table 2, we design two typical scenarios, which set different distances between blue combat platforms, to study how the geographic distribution of enemy platforms affects effectiveness. Blue combat platforms are closer to one another in scenario 1 and farther in scenario 2. In each scenario we consider two strategies in the decision model of Red warship: The first strategy (S1) deals with the threats from enemy one by one following the threat grade, and the second strategy (S2) uses an algorithm for comprehensive evaluation of all threats and reacts accordingly based on the experiment results presented in [5]. For each strategy in each scenario, we run two series of 1000 Monte Carlo simulations with two kinds of models for warship platform model, warship decision model and SASCM model: The first is the manually coded executable models (*Model_1*) and the second is the generated code from domain-specific models (*Model_2*). The results collected from Monte Carlo simulations exhibit almost the same effectiveness values: the damage rate of the Blue warship (*BWDamageRate*) and the damage rate of the Red warship (*RWDamageRate*) using the two kinds of models. We also find that the overall effectiveness value in scenario 1 is relatively higher than that in scenario 2. This means that the geographic distribution of enemy platforms has a significant impact on combat effectiveness of Red warship. Another important finding is that S2 is obviously better than S1 in both two typical concrete scenarios.

5.6 Discussions of the case study

The aforementioned modeling process is also applicable for other CoSES application development. The models are suitable to be reused and composed in other simulation applications, e.g., the warship model can be reused in warship-anti-submarine effectiveness simulation. For each CoSES application, application users select model components from the library and compose them via the model framework to construct a simulation application according to the simulation purpose. If the modeling assumption and constraints varies, only domain-specific models need to be

revised and the model implementations are generated automatically.

According to the description of Sects. 5.2–5.5, we can observe that the proposed method supports domain-specific modeling for typical CoSES domains and the models from different domains are composed based on the platform model. The domain-specific models are understandable to domain users, and executable code is automatically generated from these models. Actually, in the model design phase, the domain experts provide many useful suggestions on how to build a logically appropriate model since they can fully understand the semantics of the domain-specific model. However, the code generator needs to be improved since the modelers still need to add a few code manually to complete the model implementation.

6 Conclusions and future work

To meet the composable modeling requirement and multi-domain modeling requirement in effectiveness simulation, this paper proposed a model framework-based domain-specific modeling method. This method is based on the comprehensive utilization of current research fruits, domain knowledge and M&S experiences on CoSES. On the one hand, this method provides DSMEs to model AVK for subject domains in CoSES; on the other hand, this method uses AIK to construct a unified model framework to support the composition of subsystems models across different domains. Therefore, this method raises the modeling abstraction level, supports generative modeling and composable modeling, promotes model reuse and portability, and improves the development efficiency. The case study proved the applicability of the method. Although this method is proposed mainly to address the modeling issues in CoSES, it can also be applicable for the M&S of other complex systems.

The future work concentrates on the following aspects. Firstly, we have only constructed DSM facilities for three typical domains, thus DSM in other domains and how the models in other domains are composed should be studied, for example, the sensor domain and the countermeasure domain. Secondly, if one formalism is used to support DSM in diverse domains, then the formalism-based joint analysis of models across domains can be performed to verify and validate these models, e.g., state reachability analysis of a platform model, its decision model and affiliate equipment models. Thirdly, to promote the industrial application of the proposed approach, many technical issues should be handled appropriately, such as code generation efficiency (as discussed in Sect. 5.6), the graphic appearance of the DSME. Finally, M&S research on other kinds of complex systems (e.g., aeronautic systems, traffic networks) can be studied to explore the universal application of the proposed method.

Acknowledgements We specially thank the anonymous reviewers for their valuable suggestions and advices which greatly improve the quality of this paper. We thank our colleagues, Professor Qun Li and Dr. Chao Wang, for their contributions on SMP2 simulation system implementation. We are grateful to the suggestions and insights provided by Professor Hans Vangheluwe from University of Antwerp and Professor Pieter Mosterman from McGill University. The work presented in this paper is partly supported by the National Natural Science Foundation of China (Nos. 61273198 and 71401167).

References

1. Zimmerman, P.: DoD Modeling and Simulation Support to Acquisition. NDIA Modeling & Simulation Committee, February 21 (2013)
2. Davis, P.K., Bigelow, J.H.: Experiments in Multiresolution Modeling. RAND Corporation, Santa Monica, CA (1998)
3. Modarres, M., Cheon, S.W.: Function-centered modeling of engineering systems using the goal tree—success tree technique and functional primitives. Reliab. Eng. Syst. Saf. **64**(2), 181–200 (1999)
4. Li, X., Lei, Y., Wang, W., Wang, W., Zhu, Y.: A DSM-based multi-paradigm simulation modeling approach for complex systems. In: Winter Simulation Conference, pp. 1179–1190 (2013)
5. Li, X., Lei, Y., Vangheluwe, H., Wang, W., Li, Q.: A multi-paradigm decision modeling framework for combat system effectiveness measurement based on domain-specific modeling. J. Zhejiang Univ. C **14**(5), 311–331 (2013)
6. Li, X., Lei, Y., Vangheluwe, H., Wang, W., Li, Q.: Domain specific decision modelling and statistical analysis for combat system effectiveness simulation. J. Stat. Comput. Simul. **84**(6), 1261–1279 (2014)
7. Balci, O.: A life cycle for modeling and simulation. Simul. Trans. Soc. Model. Simul. Int. **88**(7), 870–883 (2012)
8. Sarjoughian, H., Zeigler, B.: DEVS and HLA: Complementary paradigms for modeling and simulation? Simul. Trans. Soc. Model. Simul. Int. **17**(4), 187–197 (2000)
9. Zeigler, B.P., Praehofer, H., Kim, T.G.: Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, 2nd edn. Academic Press, New York (2000)
10. Modelica Association: Modelica—A Unified Object-Oriented Language for Systems Modeling Language Specification Version 3.3 (2010). http://www.modelica.org/. Accessed 30 Apr 2013
11. Hall, S.B., Zeigler, B.P., Sarjoughian, H.S.: JointMEASURE: distributed simulation issues in a mission effectiveness analytic simulator. In: Proceedings of the Simulation Interoperability Workshop (1999)
12. Kwon, S.J., Seo, K., Kim, B., Kim, T.G.: Effectiveness analysis of anti-torpedo warfare simulation for evaluating mix strategies of decoys and jammers. In: Proceedings in Information and Communications Technology, pp. 385–393 (2012)
13. SISO Base Object Model Product Development Group: Base Object Model (BOM) Template Specification (2006)
14. European Space Agency: SMP 2.0 Handbook (Issue 1 Revision 2) EGOS-SIM-GEN-TN-0099 (2005)
15. IEEE-SA Standards Board: IEEE Std 1516-2000: IEEE standard for modeling and simulation (M&S) high level architecture (HLA)—framework and rules (2000)
16. US Army Space and Missile Defense Command, EADSIM Executive Summary (2013). www.eadsim.com/EADSIMExecSum.pdf/. Accessed 28 Apr 2013
17. JMASS Overview. https://www.jmass.wpafb.af.mil. Accessed 25 Apr 2013

18. IEEE-SA Standards Board: IEEE Standard for Modeling and Simulation High Level Architecture—Federate Interface Specification. IEEE (2010)
19. Zacharewicz, G., Frydman, C., Giambiasi, N.: G-DEVS/HLA environment for distributed simulations of workflows. Simulation **84**(5), 197–213 (2008)
20. France, R., Rumpe, B.: Domain specific modeling. Softw. Syst. Model. **4**(1), 1–3 (2005)
21. Kelly, S., Tolvanen, J.: Domain Specific Modeling: Enabling Full Code Generation, p. 437. Wiley-IEEE Computer Society Press, Hoboken (2008)
22. Li, X., Lei, Y., Vangheluwe, H., Wang, W., Li, Q.: Towards a DSM-based framework for the development of complex simulation systems. In: Summer Computer Simulation Conference, pp. 210–215 (2011)
23. Vallecillo, A.: On the combination of domain specific modeling languages. Eur. Conf. Model. Found. Appl. **6138**, 305–320 (2010)
24. Lochmann, H., Hessellund, A.: An integrated view on modeling with multiple domain-specific languages. In: Proceedings of the IASTED International Conference Software Engineering, pp. 1–10 (2009)
25. Alberts, D.S., Garstka, J.J., Stein, F.P.: Network Centric Warfare: Developing and Leveraging Information Superiority (2000)
26. Yang, F., Wang, W., Lei, Y.: Equipment Combat Effectiveness Simulation and Evaluation, 1st edn. Publishing House of Electronics Industry, Beijing (2010). (in Chinese)
27. Li, C.: UML-Based SMP2 Model Design and Integration. National University of Defense Technology (in Chinese) (2009)
28. Harel, D.: Statecharts: A visual formalism for complex systems. Sci. Comput. Program. **8**(3), 231–274 (1987)
29. Modelica Association: Modelica—A Unified Object-Oriented Language for Systems Modeling Language Specification Version 3.2 (2010)
30. Denckla, B., Mosterman, P.J.: Formalizing Causal Block Diagrams for Modeling a Class of Hybrid Dynamic Systems. In: Proceedings of the 44th IEEE Conference Decision and Control, pp. 4193–4198 (2005)
31. Clark, T., Evans, A., Kent, S.: Engineering Modelling Languages: A Precise Meta-Modelling Approach Lect. Notes Comput. Sci. Fundam. Approaches to. Softw. Eng. **2306**, 159–173 (2002)
32. Atkinson, C., Kühne, T.: Model-driven development—a metamodeling foundation. IEEE Softw. **20**(5), 36–41 (2003)
33. Vangheluwe, H., Sun, X., Bodden, E.: Domain-specific modelling with AToM3. In: Second International Conference on Software and Data Technologies (2007)
34. de Lara, J., Vangheluwe, H.: AToM3: A tool for multi-formalism and meta-modelling In: In European Joint Conference on Theory and Practice of Software (ETAPS), Fundamental Approaches to Software, pp. 174–188 (2002)
35. Vanderbilt University, GME Manual and User Guide- Generic Modeling Environment 10 (2010)
36. Tolk, A., Muguira, J.A.: The Levels of Conceptual Interoperability Model. In: 2003 Fall Simulation Interoperability Workshop, September (2003)
37. Sprinkle, J., Rumpe, B., Vangheluwe, H., Karsai, G.: Metamodelling: State of the Art and Research Challenges. In: Giese, H., Karsai, G., Lee, E.A., Rumpe, B., Schätz, B. (eds.) Model-Based Engineering of Embedded. Real-Time Systems, vol. 6100, pp. 57–76 (2010)
38. Emerson, M.J., Sztipanovits, J.: Implementing a MOF-Based Metamodeling Environment Using Graph Transformations In: 4th OOPSLA workshop on domain-specific modeling, pp. 83–92 (2004)
39. Chen, K., Sztipanovits, J., Neema, S.: Toward a Semantic Anchoring Infrastructure for Domain-Specific Modeling Languages. In: Fifth ACM International Conference on Embedded Software, pp. 35–43 (2005)
40. Balasubramanian, D., Narayanan, A., VanBuskirk, C., Karsai, G.: The graph rewriting and transformation language: GReAT. In: Third International Workshop on Graph Based Tools, vol. 1, pp. 1–8 (2006)

Xiao-bo Li is an Assistant Professor at National University of Defense Technology (NUDT), and also a PhD candidate at University of Antwerp. His research interest includes model-driven engineering techniques for M&S, simulation-based system design and demonstration, and domain-specific modeling. He holds a Ph.D. degree in Systems Engineering from National University of Defense Technology. His email address is lixiaobo@nudt.edu.cn and his website is http://msdl.cs.mcgill.ca/people/xiaobo.

Feng Yang is a Professor at National University of Defense Technology. His research interest focuses on Model Driven Systems Engineering, and simulation-based system design and demonstration. He holds a Ph.D. degree in Systems Engineering from National University of Defense Technology. His email address is forestyoung@nudt.edu.cn.

Yong-lin Lei is an Associate Professor at National University of Defense Technology. His research interests include complex system simulation, model composability, model-driven architecture, domain-specific modeling, and their applications in defense simulations. He holds a Ph.D. degree in Systems Engineering from National University of Defense Technology. His email address is yllei@nudt.edu.cn.

Wei-ping Wang is a Professor at National University of Defense Technology. His research interest focuses on system of systems engineering and simulation, and simulation-based system design and demonstration. He holds a Ph.D. degree in Systems Engineering from National University of Defense Technology. His email address is wangwp@nudt.edu.cn.

Yi-fan Zhu is a Professor at National University of Defense Technology. His research interest includes simulation-based system design and demonstration, and agent-based modeling and simulation. He holds a Ph.D. degree in Systems Engineering from National University of Defense Technology. His email address is yfzhu@nudt.edu.cn.