Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A multi-paradigm decision modeling framework for combat system effectiveness measurement based on domain-specific modeling*

Xiao-bo LI†[1,2], Yong-lin LEI[1], Hans VANGHELUWE[2], Wei-ping WANG[1], Qun LI[1]

([1]*Institute of Simulation Engineering, College of Information Systems and Management, National University of Defense Technology, Changsha 410073, China*)

([2]*Department of Mathematics and Computer Science, University of Antwerp, Antwerp 2020, Belgium*)

†E-mail: lixiaobo.nudt@gmail.com

Received Dec. 22, 2012; Revision accepted Mar. 21, 2013; Crosschecked Apr. 16, 2013

Abstract: Decision modeling is an essential part of the combat system effectiveness simulation (CoSES), which needs to cope with the cognitive quality, diversity, flexibility, and higher abstraction of decision making. In this paper, a multi-paradigm decision modeling framework is proposed to support decision modeling at three levels of abstraction based on domain-specific modeling (DSM). This framework designs a domain-specific modeling language (DSML) for decision modeling to raise the abstraction level of modeling, transforms the domain-specific models to formalism-based models to enable formal analysis and early verification and validation, and implements the semantics of the DSML based on a Python scripts framework which incorporates the decision model into the whole simulation system. The case study shows that the proposed approach incorporates domain expertise and facilitates domain modeler's participation in CoSES to formulate the problem using DSML in the problem domain, and enables formal analysis and automatic implementation of the decision model in the solution domain.

Key words: Multi-paradigm modeling (MPM), Decision modeling, Domain-specific modeling (DSM), Effectiveness measurement, Model transformation

doi:10.1631/jzus.C1200374 **Document code:** A **CLC number:** TP391.9

1 Introduction

Because of the increasing structural and behavioral complexity of combat systems, modeling and simulation (M&S) is being widely used in combat system effectiveness measurement (CoSEM) (Davis and Bigelow, 1998). Combat modeling contains two aspects: the physical aspect that relates to the internal physical mechanisms in the physical and information domains, and the intelligent aspect that relates to the tactics of how to use the combat systems under certain circumstances. In most cases, physics modeling (i.e., modeling the physical aspect) becomes the modeler's main concern, and decision modeling (i.e., modeling the intelligent aspect) is usually technically affiliated to physics modeling. However, the operations of the combat systems (i.e., tactics) are as important as their physical attributes. They should be appropriately modeled since they have distinct characteristics compared to the physical aspect: firstly, decision making belongs to the cognitive domain which concentrates on logical reasoning rather than physical mechanisms; secondly, different combat systems have a diversity of decision making modes; thirdly, tactics evolution is inevitable with the change of time and situation; fourthly, decision modeling studies how combat personnel employs combat systems, and decision models are at a higher abstraction level than physical domain models.

* Project (Nos. 61273198, 91024015, 61074107, 60974073, 60974074, and 71031007) supported by the National Natural Science Foundation of China

We have studied CoSEM for many years and have built a simulation model portability standard 2 (SMP2)-based M&S platform called Sim2000 for the combat system effectiveness simulation (CoSES) (Li *et al*., 2010). In the initial phase of our research, decision modeling was based on the same technical space as physics modeling (unified modeling language (UML) diagrams for design and C++ code for implementation); i.e., decision parts are hard-coded inside the combat simulation systems and closely coupled with the physical parts. This mixture leads to several disadvantages, such as inflexibility and difficulty in understanding. Motivated by the technical separation idea of the 'tactics manager' (Son *et al*., 2010), we introduced a Python script framework to build decision scripts that are separated from physical domain models, and we proposed a domain-specific decision modeling approach that supports decision modeling at the problem domain level to exploit the tactics expertise and promote the efficiency (Li *et al*., 2013).

Multi-paradigm modeling (MPM) addresses complex system design and implementation issues combining the following orthogonal dimensions: model abstraction, multi-formalism modeling, and metamodelling. MPM highlights the importance of model transformation to link models of different formalisms at different abstraction levels (Mosterman and Vangheluwe, 2004). The MPM method is suitable for dealing with the tough demands of decision modeling: (1) Multi-formalism modeling can use a series of formalisms to appropriately model the diversity of decision making modes and analyze the decision model with formal methods. (2) Metamodeling can be used to construct the metamodel of the decision modeling language. (3) Model transformation facilitates automatic implementation of decision models and promotes efficiency.

In this paper, to cope with the cognitive quality, diversity, flexibility, and higher abstraction of decision making, and to solve the problem encountered in our application, we combine the domain-specific modeling (DSM) with MPM and propose a DSM-based MPM framework to support the decision modeling at three levels of abstraction, namely conceptual modeling at the domain-specific level, simulation model design and analysis at the formalism level, and model implementation at the code level.

This research mainly focuses on the combat platform decision modeling at the engagement level and contributes to the M&S community in three aspects: firstly, it designs a domain-specific modeling language (DSML) for decision modeling based on the domain analysis, and implements its semantics by model transformation to modeling formalisms and automatic generation of Python scripts; secondly, it constructs a modeling framework to support decision modeling at three abstraction levels and build the links among them; finally, MPM is empowered to support modeling at the problem domain-specific level with DSM.

2 Related works

This section concentrates on works related to the decision modeling at three levels of abstraction.

2.1 Decision modeling for combat simulation systems at the implementation level

The implementation solutions of decision modeling for combat simulation systems can be divided into four categories: built-in modeling, rule library based modeling, table-based modeling, and script-based modeling.

The built-in method models the decision process based on the same technical space as in physics modeling. For example, Seo *et al*. (2011) built an underwater warfare simulator including the decision model (named the controller) based on discrete event system specification (DEVS). The advantage of this method is that the whole simulation system is based on one technical space, and thus it is technically easy to implement and efficient to execute at runtime; as discussed in Section 1, the disadvantage is the inconvenience of making changes on decision models.

The rule library based method builds a rule library and dynamically inquires the library and finds the matched rule at runtime. The rules can be revised at runtime and the revisions take effect immediately, and thus it is efficient for modeling decision alternatives. However, the rules are scattered and the decision process description is obscure using a rule library, so this method is restrictively used when the problem space is explicit (rules are explicit) and the scale of the system is relatively small.

The table-based method (Son *et al*., 2010) uses text files (.txt) for tactics description. In the text files, C language-based data types are used for the decision condition types. The simulation model will call the decision table file when it needs to select a tactics alternative under a certain condition. This method relies on the interpreter inside the simulation model to parse the text content correctly. Though it is in an independent technical space, this method lacks flexibility.

The script-based method implements the decision process using scripts, which are dynamically interpreted when called by the simulation system. Besides flexibility, script languages can support description of decision process (Son *et al*., 2010; Son and Kim, 2012a). Actually, the script-based and the rule library based methods can be used together; a common example is the Extended Air Defense Simulation (EADSIM) (US Army Space and Missile Defense Command, 2012).

As listed in Table 1, each of these methods has its own advantages and disadvantages, which is suitable for certain situations. As far as the platform decision modeling at the engagement level to support CoSEM under the background of system of systems (SoS) combat is concerned, the whole simulation system is large-scaled, involving a series of equipment models, and the problem space is not explicit since different system effectiveness simulations relate to different configurations of combat SoS and distinct combat scenarios. The script-based method is better suited to the research purpose of our work for the following reasons: the built-in and table-based methods lack flexibility; the rule library method fails to describe the decision process and it is infeasible to build a rule library for all the platforms in various scenarios under the SoS combat background.

Table 1 Comparison of implementation methods for decision models

Method	Flexibility	Diversity	Technical separation	Abstraction level	Process description
Built-in	Weak	Weak	No	Low	Strong
Rule library	Strong	Medium	Yes	Low	Weak
Table-based	Medium	Medium	Yes	Low	Medium
Script-based	Strong	Strong	Yes	Low	Strong

2.2 Decision modeling for combat simulation systems at the formalism level

Formalisms in M&S can be defined to consist of two parts: model specification and execution algorithm. The model specification is a mathematical theory describing the kinds of structure and behavior that can be described with it. The execution algorithm specifies an algorithm that can correctly execute any model described in accordance with the model specification (Sarjoughian and Zeigler, 2000). A modeling formalism is usually an abstraction of certain structural and behavioral characteristics of systems and can be used to describe diverse systems that possess the same characteristics across different application domains.

A diversity of modeling formalisms are used for decision modeling. Since decision making has the cognitive quality, logic-related modeling formalisms are widely used to describe the decision logic and infer the rule-set; for example, Son and Kim (2012a) presented a fuzzy logic based approach to model torpedo evasion tactics, and Liu and Shi (2010) proposed an object-oriented Bayesian networks based decision modeling approach to study unmanned combat air vehicle teams air-to-ground attack decision making. To combine the decision process with system state and event, formalisms can be used for decision modeling; for example, DEVS can be used to model the decision of the controller in the underwater platform model (Seo *et al*., 2011) and maneuver control model (Son and Kim, 2012b), and Petri Net can be used to model the fleet cooperation decision-making system (Liu, 2007).

Though agent-based modeling (ABM) is a modeling paradigm rather than a concrete formalism, ABM provides inherent decision modeling support since it is tailored for cognitive modeling (Sokolowski, 2003), and ABM can be instantiated to concrete formalism in the application domains; e.g., Sarjoughian and Huang (2005) gave an example to use a specific type of ABM formalism, reactive action planning, to model the decision part of the whole simulation model.

On the one hand, formalism-based decision modeling has several advantages: (1) It raises the abstraction level from the code implementation level to the formalism specification level, so that the

models are more easily understood and manipulated by humans, which is especially important since decision modeling needs to cope with the cognitive quality. (2) The mathematical foundation of the formalisms supports formal analysis and early verification & validation of decision models, and the execution algorithms of the formalisms enable dynamic execution of the models and even simulation when tools are available.

On the other hand, formalism-based decision modeling cannot meet all the requirements for effective simulation of the combat system: (1) The formalism-based notations are still unfamiliar to tactics modelers. (2) Formalisms merely describe certain aspects of the modeled subjects and abstract formalism-related characteristics of diverse problem domains, and thus may neglect other aspects of the modeled subject in a specific problem domain.

Principally decision modeling at the formalism level is independent of technical solutions, and the formalism-based decision models can be implemented based on any of the aforementioned three ways. However, the inherent characteristics of the formalisms may impact the implementation choice; e.g., the rule library method is not effective and natural to describe the process information of a DEVS-based decision model.

2.3 Domain-specific simulation modeling: raising the abstraction level of decision modeling

From Table 1, we can see that the script-based method meets most of the requirements of platform decision modeling except that the abstraction level is also low, and formalism-based models can describe only the decision process with formalism-related concepts. Raising the abstraction level of modeling is of critical importance to enable modelers from the problem domain (tactics modelers in this case) to participate in decision modeling and offer appropriate modeling concepts which promote interoperability and understandability. Currently, UML is a commonly used tool to provide higher level descriptions of conceptual models and UML profile mechanism enables user customization for their problem domains. However, UML is not designed for simulation modeling and provides mainly software modeling concepts.

DSM is proposed as a new software development paradigm in the software engineering community to raise the level of abstraction beyond current general programming languages, to increase the productivity and to enable the domain experts to take part in the software development by specifying the solution directly using problem domain concepts (France and Rumpe, 2005). According to the 3D conceptual model for service-oriented simulation (Wang *et al*., 2009), M&S activities are closely related to the developments of software engineering and systems engineering, and computer simulation models technically are also software models; thus, simulation modeling can also adopt a DSM approach (Li *et al*., 2011).

Currently, M&S researchers usually study DSM for M&S from three aspects of concern: problem-oriented solution by constructing DSML or domain-specific modeling environment (DSME), generic tool extension to provide domain support, and adaptation of software engineering (especially model driven engineering) techniques (Li and Shen, 2010) to enable DSM for M&S system development (Li *et al*., 2011). Though domain extension (e.g., simulation building blocks methodology (Verbraeck and Valentin, 2008)) and DEVSJAVA extension (Ferayorni and Sarjoughian, 2007) are meaningful to empower existing tools, these efforts are limited by their existing technical constraints and paradigm limitation. Using DSM tools for simulation modeling exhibits promising potentials, since these tools are built based on DSM methodology and state-of-the-art technologies. Among all DSM platforms, the generic modeling environment (GME) is considered as the centerpiece modeling technology for the model integrated computing (Davis, 2003) and is better suited to the demands of decision modeling. GME is an open-sourced tool which provides infrastructure supports for metamodelling, DSME generation, and interpreter construction; thus, it is widely used in DSM practices for M&S. For example, it is used to develop an integration model for high-level architecture (HLA) systems (Hemingway *et al*., 2012) and to create domain-specific language (DSL) and transform the domain-specific models to the DEVS formalism (Mittal and Douglass, 2011). Especially, Neema *et al*. (2009) used the GME metamodeling tool-suite to create a model-based integration approach that allows for a rapid synthesis of different command and control models in complex HLA-based simulation environments.

3 Domain-specific modeling based multi-paradigm decision modeling framework for combat system effectiveness measurement

Based on the discussion of related works, we conclude that to cope with the characteristics of combat platform decision making as proposed in Section 1, the decision modeling for CoSES should meet the following requirements:

1. The framework should promote technical separation of models in the physical domain and models in the cognitive domain to support flexible modeling.

2. The framework should support decision modeling at the domain-specific level to provide DSML for tactics modelers.

3. The framework should provide a framework to incorporate the expertise of decision making experts, software engineers, and simulation experts, and to enable their cooperation.

To meet these requirements, we propose a DSM-based multi-paradigm decision modeling framework for CoSES to support the decision modeling at three levels of abstraction in a separate technical space. This framework is incorporated in a CoSES system. Fig. 1 presents an overview of the model framework of the whole simulation system. The simulation system is composed of two parts: physical domain model and cognitive domain model. The former part is implemented using Visual Studio in C++, which models the combat system's structure and behavior in the physical domain and information domain. The latter part describes the tactics and decision making process in the cognitive domain. These two parts interact via a C++/Python interface that provides situation information of the physical domain to the cognitive domain and transfers orders from the cognitive domain to the physical domain.

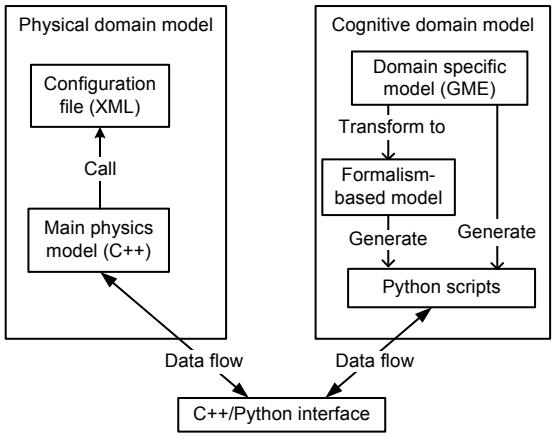

Fig. 1 A multi-paradigm modeling (MPM) framework for combat system effectiveness measurement (CoSEM)

There are three layers of decision models, namely the DSM layer, the formalism layer, and the script layer. The DSM layer provides a GME-based DSME tailored for decision modeling. The DSME is generated from a DSML metamodel composed of situation description and decision making concepts. Tactics modelers build domain-specific decision models using the DSML in DSME, which can be transformed to either formalism-based models or Python scripts. The formalism layer acts as not only the semantic anchoring base, but also a modeling environment. A tactics modeler familiar with the formalisms can build formalism-based decision models and perform formal analysis and verification & validation in the formalism-specific modeling environment. At the bottom layer, scripts in Python executed by the simulation system can be generated from either domain-specific or formalism-based models.

The DSM-based decision modeling framework can certainly meet the first and second requirements. It also provides a cooperation framework to incorporate domain expertise in DSML engineering for decision modeling, M&S expertise of M&S formalism and environment, and software engineering techniques to implement model transformation and environment construction. Decision modelings at the three layers and their relationships are explicitly studied in Sections 4–6.

The physical domain model has two components: the configuration file in XML which sets the value of the parameters and experimental factors, and the main model in C++ which describes the entity structure and dynamic behavior. When the simulation starts, the main model reads the configuration file and initializes the parameters with corresponding values read from the file. The configuration file is flexible and revisions take effect immediately without recompilation, which can be used to investigate how the values of the parameters affect system effectiveness. The main model describes the entity structure and the fundamental behaviors, and it should be more stable as it is hard coded and needs to be rebuilt if there is any modification.

4 Domain-specific modeling based decision modeling

DSM-based decision modeling refers to modeling at the problem domain level with domain-specific language tailored for decision modeling, and the key is to design a DSML for decision modeling, and specify its semantics. A DSML contains three parts: abstract syntax, concrete syntax, and semantics. This section mainly discusses metamodeling the abstract syntax of the DSML based on a problem domain analysis and the metamodel-based generation of a DSME using GME. The semantics of the DSML is discussed in Sections 5 and 6.

4.1 Problem domain analysis of decision modeling for CoSES

As discussed in Section 1, decision modeling in CoSEM concentrates on combat platform decision modeling at the engagement level. Thus, the decision making method itself is outside our research scope, and we mainly study how to model the tactics of main combat platforms, such as warship, fighter plane, and submarine. These combat platforms are usually equipped with sensors, weapons, and jammers, and can execute various tasks. For simplicity, the intelligent behaviors of sensors and weapons are not explicitly researched.

Essentially platform tactics are a series of 'IF…THEN…' specifications to specify how the platform behaves 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. 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.

Fig. 2 gives an overview of the two spaces. In the situation space, there are three dimensions (task, status and capability, and target and threat). In each dimension, there are a set of situational conditions which need to be evaluated, while for each condition evaluation, there are three kinds of decision choices (movement management, counterwork management, and sensor management) in the order space. In principle, the number of the items combination in six dimensions is huge, and modeling the combinations one by one is taxing and infeasible in the background of the SoS combat process. Actually, some of the combinations are invalid and the sizes of the two spaces can be reduced. The situation space can be divided into three groups:

1. Tasks: The platform executes a series of tasks and the two spaces can be divided as a number of task-guided activities which constitute the backbone of the situation space.

2. Phases: The combat process of the platform can be separated by certain phases, in which certain state variables of the platform remain invariant. These phases are usually identified according to main physical mechanisms or typical behavioral patterns. For example, when a helicopter is executing an anti-submarine task, the phase can be takeoff, flying to waypoint, hovering, tracking target, or landing.

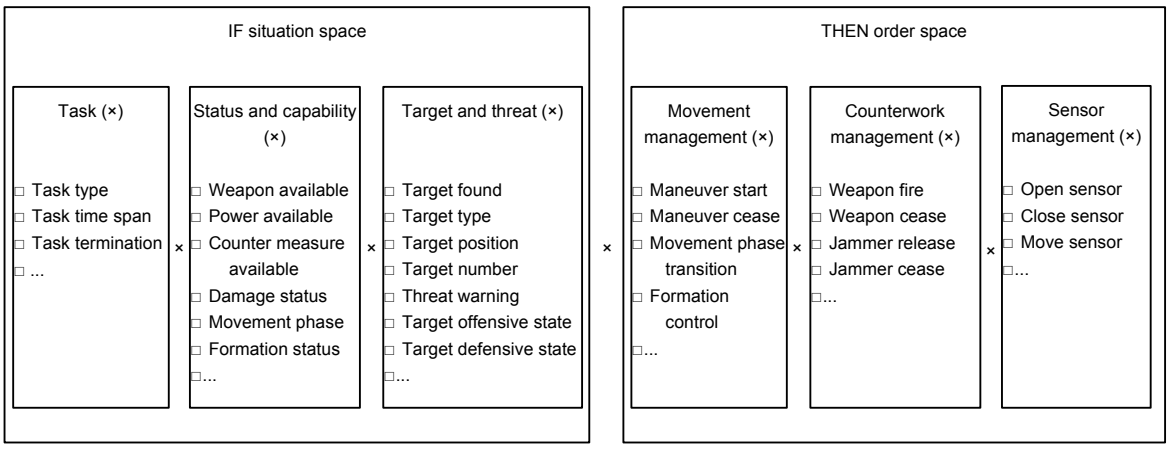

Fig. 2 Situation space and order space of decision making

3. Conditions: Conditions refer to a set of evaluation items of the situation (including information of both the allies and the rivals), which act as the stimulus for the tactic action of the platform.

The decision making process of combat platforms can be described using the three concepts as follows: the platform performs certain tasks (in parallel or serially), each of which can be described as a decision process. The process is associated with certain phases and decision making is performed to respond to the change of the situation based on the current task and the phase. As shown in Fig. 3, the decision modeling scheme can be simplified by reorganizing all items in the two spaces as follows. The platform tactics is a set of task-guided decision activities (from Task1 to Task*N*); each task is composed of certain platform phases, e.g., Phase1 and Phase2 in Task1. The phase can be transited to itself or other phases (either in the same or different tasks) when a condition (e.g., C1 connecting Phase1) connecting this phase is evaluated to be true; and, the decision action corresponding to the condition (e.g., A1 for C1 in Task1) is executed if the condition is true. Transition to phases in other tasks means the termination of the current task and beginning of another new task.

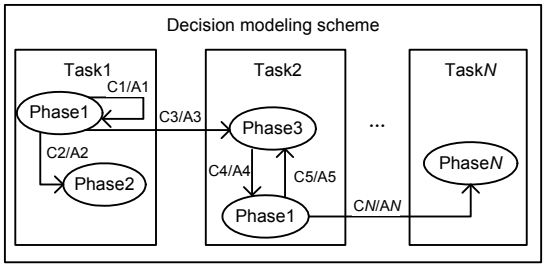

Fig. 3 An example of the decision modeling scheme

4.2 Metamodeling the domain-specific modeling language (DSML) for decision modeling

The core of DSM-based decision modeling is to design a DSML which captures the fundamental concepts and relationships of decision modeling. 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 (Sprinkle *et al*., 2011). Based on the aforementioned problem domain analysis, we build a decision modeling DSML metamodel (Fig. 4) using the metamodeling language of GME (MetaGME), which elaborates the metamodel proposed in Li *et al*. (2012).

As illustrated in Fig. 3, the core concepts for decision modeling are task, phase, condition, and tactic action. The task is a tactical objective that the platform attempts to achieve by a sequence of tactic actions based on a series of platform phases. Typical tasks are patrolling an area, tracking a target, destroying a specified target, evading a threat, and so on. A 'Task' should have at least two actions: one initial action and one end action (for conciseness, we attribute these two fake actions to tactic actions). The initial action is the starting point of performing a task, and the end action means the platform finishes the current task. Tasks can contain sub-tasks and are connected by TaskSequence. The action is an instantaneous decision order, which is triggered by two kinds of situations. The first is an action sequence (ActionSequence), namely by finishing the former action which this action connects to. The second is when the value of a decision condition is evaluated to be true, which has a link whose destination is the action. The action has an enumerated attribute ActionType to specify the type of the action. Typical action types contain sensor management, weapon management, movement management, jammer management, and so on. A Phase is a typical state of the platform, and the transition is activated when the PhaseTransCon is evaluated to be true. The Phase has an enumerable attribute called PhaseType, and the typical phase type is usually specified according to its movement modes, damage status, or available weapons and energy; for example, based on movement modes, the PhaseType can be categorized as waypoint phase, track phase, evasion phase, follow phase, and so on. For length limitation, we do not list the concrete types of Phase, EventTrigger, Variable, LogicalOperator, and MathOperator of the metamodel.

The condition is the primal concept to describe the situation space, which consists of at least one guard expression, and can have logical operators (include AND, OR, XOR, and NOT). Guard expressions are connected by LogicalOperators to calculate a Boolean value for the condition. The guard expression comprises at least one value, and can have mathematical operators (e.g., <). There are three kinds of values: constant, variable, and event trigger. The variable stands for the status of the situation space,

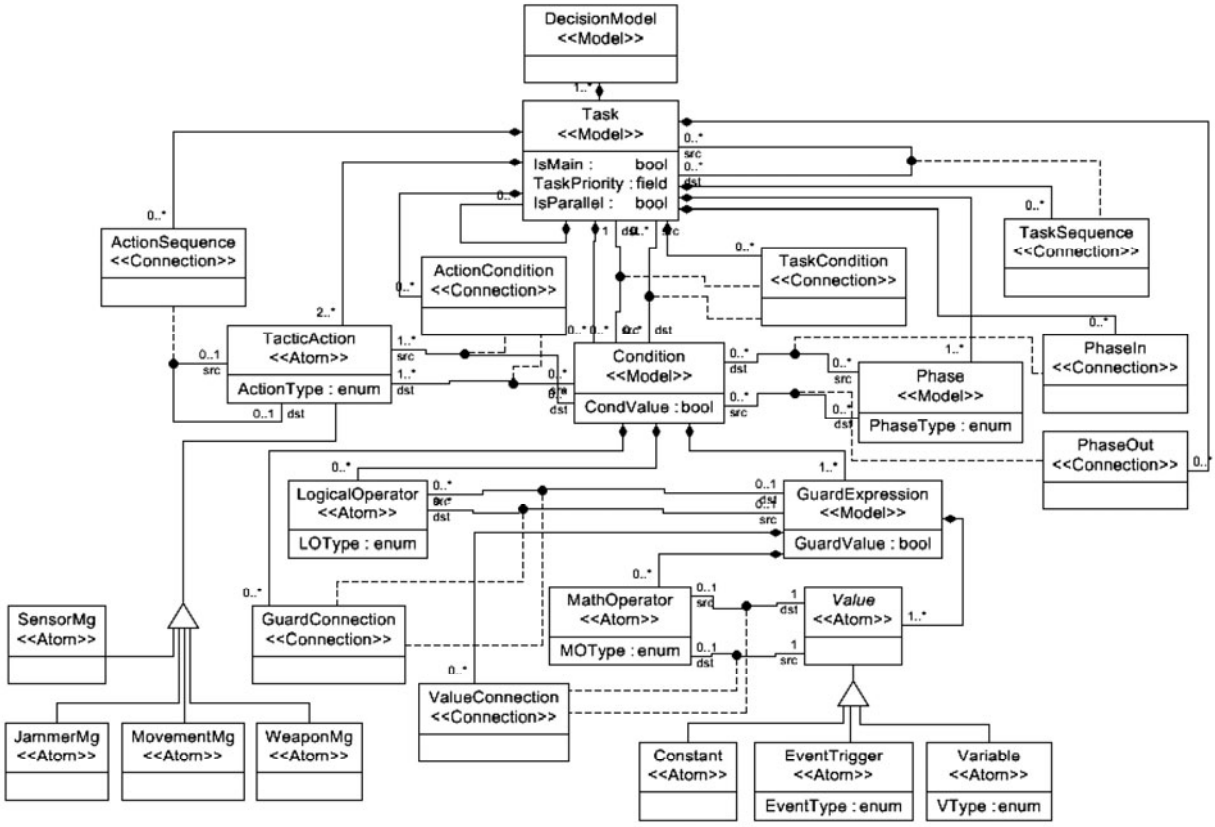

Fig. 4 Metamodel of the domain-specific modeling language (DSML) for decision modeling

and the constant is used to make comparison against the value of the variable. The event trigger is used to monitor whether an event has occurred, and its value turns true when the event occurs. The event trigger has an EventType attribute to enumerate all supported event types of the DSML, e.g., target detected signal.

GME provides a Constraint aspect to specify the constraints of the metamodel based on the object constraint language (OCL) to enhance the well-formedness of the metamodel. We can use this function to make the metamodel more precise. For example, for the constraint that each decision model should have only one main task, a constraint can be added to Task and the OCL equation is

let tasks=self.parts(Task) in tasks→size>0,
implies tasks→select(s:Task|s.IsMain)→size=1.

Similarly, we can add more constraints, such as the size of Value should be one bigger than the size of MathOperator in each GuardExpression.

4.3 DSME for decision modeling at the DSM level

The metamodel designs only the abstract syntax, and thus we implement the concrete syntax of metamodel elements by attaching domain familiar icons to them. Then, a DSME is automatically generated from the metamodel using the icons as shown in Fig. 5. The DSME is a graphical editor with domain-specific concepts, which are familiar to the tactics modelers, and it is not error-prone since the abstract syntax and constraints guide the user to model legally and help them to locate mistakes using its 'Check' function. The modelers do not need to type too much since the main work is to select the graphical elements, drag them to the canvas, specify their attributes, and link the elements together. The DSME in GME provides hierarchical modeling views according to the structural relationship of the model elements. For example, if you double click the WaypointAssigned icon, a new modeling view will pop up with condition modeling elements

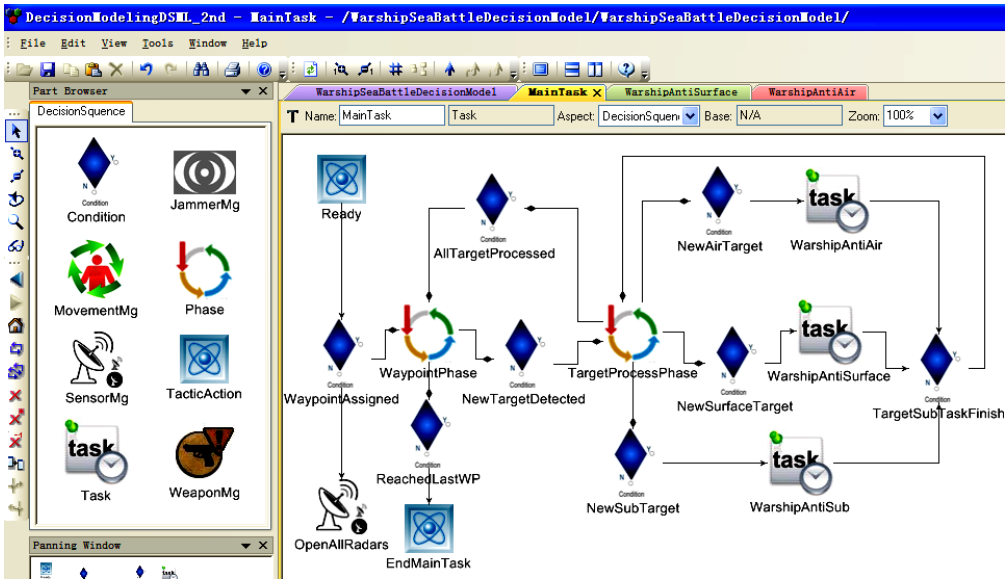

Fig. 5 A domain-specific decision modeling environment in generic modeling environment (GME)

(GuardExpression and LogicalOperator). The models are saved as XML files, which support file interchange between GME and other tools.

5 Formalism-enabled decision modeling

5.1 Typical process of formalism-based decision modeling and analysis

Though domain-specific decision models can describe the decision making process using domain-related concepts, they are neither formal nor executable. To support formal analysis and verification and validation of the domain-specific models, we propose the following process to use classical formalisms for the decision modeling and analysis of combat platforms:

1. Conceptual analysis of the platform decision process and characteristics

Different platforms bear different tasks, which have diverse decision processes and unique characteristics. Though these processes and characteristics can all be described with the domain concepts of DSML, they cannot be formally analyzed or dynamically executed since DSML itself has no explicit semantics. We need to recognize the patterns and modes of the decision process, and conclude which kind of characteristics they possess. For example, the air combat decision of a fighter plane is different from the sea battle task of a warship. The decision rules of the fighter plane is complex and the state space of the fighter is large since the maneuverability of the fighter plane is better than that of the warship, so the focus of decision modeling is how to build a logically correct rule set based on the state of the plane. However, in the sea battle of a warship, the main concern of decision modeling is how to describe the task concurrency since the warship needs to concurrently cope with different kinds of targets and threats such as airplanes, submarines, and warships.

2. Formalism selection

After conceptual analysis, we need to choose the formalism(s) to study the basic process and characteristics of the decision. Table 2 lists the formalism choice for certain analysis purposes. Table 2 is not complete and can be extended if necessary. For example, for fighter plane decision, we can choose statecharts as the formalism to study its large state space; and for sea battle decision of the warship, Petri Net is a suitable choice to study the concurrency of multiple tasks. If the decision process is too complex and requires being studied based on multi-formalism, then a set of formalisms can be chosen.

3. Transformation from domain-specific models to formalism-based models

Once the formalisms are chosen, the domain-specific models based on DSML need to be transformed to the chosen formalisms. There are three

Table 2 Formalism choices based on the analysis purpose

Analysis purpose	Formalism choice
Probability	Bayesian network
Rule completeness, consistency, and redundancy	Predict logic, Temporal logic, Petri Net
State reachability	Statecharts, Petri Net
Concurrency	Petri Net
Time	Timed Petri Net, DEVS, Temporal logic

choices for the transformation: (1) Build modeling environments for the chosen formalisms using GME, and then use the graph rewriting and transformation language (GReAT) (Balasubramanian *et al*., 2006), the transformation tool of GME, to transform the domain-specific models to formalism-based models. (2) If the formalism-based tool can import XML-format models, we firstly store the domain-specific models as XML, then do an XML-based transformation to convert the source models to destination models, and finally import the XML models into the modeling environment. (3) If the formalism has a sound mathematical foundation, then transform the domain-specific models to the mathematical representation of the formalism. These three solutions can be used in combination to meet various requirements.

4. Formalism-based static check of the rule set

If the chosen formalism(s) supports the rule set analysis, then the rule set can be checked. This is important especially when the rule set is large and complex. Normally the logic-based formalisms are suitable to infer and verify certain characteristics of the decision rules using formal analysis methods, such as the check for the completeness, consistence, and redundancy of the rule set; the check is usually based on mathematical analysis and does not require dynamic execution. For example, decision rules for the fighter plane can be checked using the logic predicate. Certain tools of the underlying formalisms can be used to perform static check. If any problem is detected, we then need to modify the domain-specific models accordingly and go to step 3.

5. Formalism-based dynamic analysis of the decision process

If nothing unusual is found in the static analysis and the formalism supports dynamic execution, then the dynamic analysis of the decision process is required to verify the decision model. The dynamic analysis is usually performed based on a tool which implements the semantics of the formalism; for example, the coloured Petri Net (CPN) tools (Ratzer *et al*., 2003) can simulate the model and exhibit the decision process. If the process is not the same as the conceptual model, then we should locate the deviations, correct them accordingly in the domain-specific model, and go to step 3.

6. Generation of decision scripts from formalism-based models

Normally, we use only the formalism-based model to perform formal analysis and early verification and validation, and do not need to transform formalism-based models to decision scripts. However, in some cases the transformation is useful: the first is when we care only about the characteristics of the domain-specific model and these characteristics can be sufficiently described by the formalisms, and then the transformation can substitute the transformation of the domain-specific models to the scripts; the second is when we build decision models directly using formalisms and the simulation system is compatible with only decision scripts.

5.2 Formalism-based modeling environments using GME

In Section 5.1, we have discussed three methods of transformation from domain-specific models to formalism-specific models in step 3 of the typical process. There are usually tools available for formalism-specific modeling, and many of them support XML-format import and export; for example, CPN tools for Petri Net and DEVSML for DEVS (Mittal *et al*., 2007). Method 2 is applicable in this case since we can store domain-specific models as XML and perform XML-based transformations. Method 3 is also applicable when the models are not so complex and the transformation relationship from domain-specific models to formalism's mathematical representation is obvious and easy to implement.

In this section, we concentrate on method 1 for the following reasons. Firstly, GME provides a powerful technical space for metamodeling the formalism, metamodel-based modeling environment generation, and metamodel-based model transformation. Secondly, after we transform the domain-specific models to formalism-based models, methods

2 and 3 are easier to implement. For example, when we store the GME-based Petri Net models as XML files, it is easier to import them into CNP tools compared to using method 2 independently. We choose Petri Net as the example to illustrate how to build a formalism-based modeling environment using GME. The metamodel of basic Petri Net is illustrated in Fig. 6, which includes fundamental concepts of Petri Net. From the metamodel, a Petri Net modeling environment can be automatically generated (Fig. 7). The execution semantics is the same as that of the basic Petri Net (Murata, 1989).

5.3 Semantic anchoring from decision modeling DSML to formalisms using GReAT

Semantic anchoring is proposed to achieve semantics of DSMLs via transforming them to a set of 'semantic units', and a tool suit is built to support the semantic anchoring (Chen *et al*., 2005). In this tool suit, GReAT is a graph transformation language and a toolset to specify and execute model transformation based on mapping relationship between metamodels, which is used in conjunction with GME. We illustrate how to transform the decision modeling DSML to Petri Net using GReAT. The metamodel mapping relationship between the DSML and Petri Net is listed in Table 3.

The mapping relationship only partially maps the DSML metamodel to Petri Net since Petri Net mainly describes state-transition behaviors. Most of the TacticActions and the ActionConditions irrelevant to phase transition are unnecessary for mapping to Petri Net, and the details of the conditions (GuardExpressions and LogicalOperators) are also omitted. The Task is a hierarchical organization of the Phases and Conditions, including one Initial TacticAction and one Final TacticAction, and thus the organization relationship is also transformed by transforming the two fake actions to place.

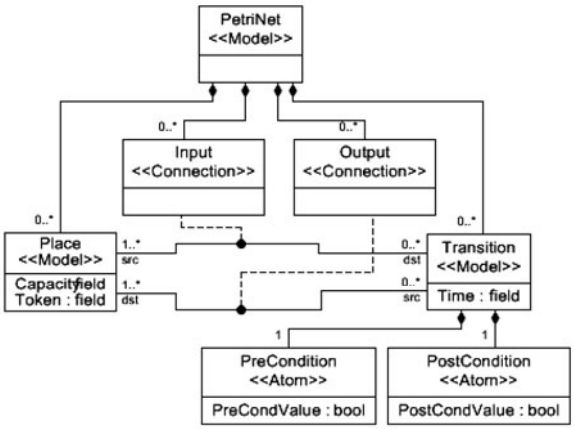

Fig. 6 A MetaGME-based basic Petri Net metamodel

Table 3 Metamodel mapping relationship between the DSML and Petri Net

DSML	Petri Net
DecisionModel	PetriNet
TacticAction (initial)	Place
TacticAction (final)	Place
Phase	Place
Condition (partly)	Transition
PhaseIn	Input
PhaseOut	Output
MaxTargetProcessNum (Attribute of TargetProcessPhase)	Token

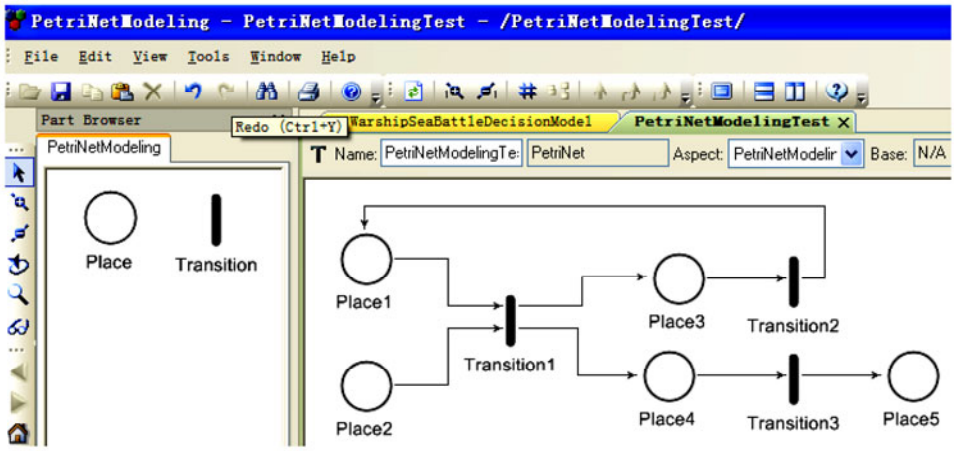

Fig. 7 A Petri Net modeling environment generated using GME

Based on the mapping relationship, we build a transformation model using GReAT (Fig. 8). The transformation model is composed of five rules. The first rule is to transform the main task of the decision model to a Petri Net model; the second rule transforms the initial state and the end state of a task to places, since the phases in different tasks are connected by them; the third rule transforms all the phases to places; the fourth rule transforms phase transition conditions to transitions; and, the fifth rule recognizes the connections (PhaseIn and PhaseOut) between the phases and conditions, and transforms them to links (Input and Output) of places and transitions. Fig. 9 presents the first rule, which creates a PetriNet model for the DecisionModel and sets the name of the PetriNet model to the name of the DecisionModel by AttributeMapping. For length limitation, we omit details of other rules.

6 Script-based implementation of decision models

6.1 Script-based domain-specific modeling language (DSML) semantics implementation

Simulation is the dynamic execution of models along a time base, and thus the DSML should have explicit execution semantics to support effectiveness simulation. From the metamodel, we can see that a decision model is composed of a set of tasks, and each task is actually a decision activity which comprises a series of actions and the conditions or sequences connecting them. The execution logic of the DSML is as follows. At the beginning of the simulation, the platform model performs the main task. When one task terminates, the platform will perform other task(s) by sequence following the TaskSequence line. Within each task, firstly the initial action is executed, and then the decision process goes along the connection. If the connection is an ActionSequence, the action(s) at the destination of the connection is (are) performed unconditionally and the decision process will go on immediately after the performance. If the connection is an ActionCondition or PhaseTransCon, the condition(s) at the destination of the connection is (are) evaluated by calculation of the GuardExpression and the logical operators; the decision process will stop and only stop at the condition that the attribute CondValue is false, and will go on along the connection otherwise. The task terminates at the end action and will go on along the TaskSequence connection if it exists.

Though we have explained the descriptive semantics in a natural language, the DSML requires execution semantics that can be carried out by the computer. The GME provides a builder object network (BON) based framework for interpreter construction using Visual Studio (Davis, 2003). In our case, we employ the framework to build a code

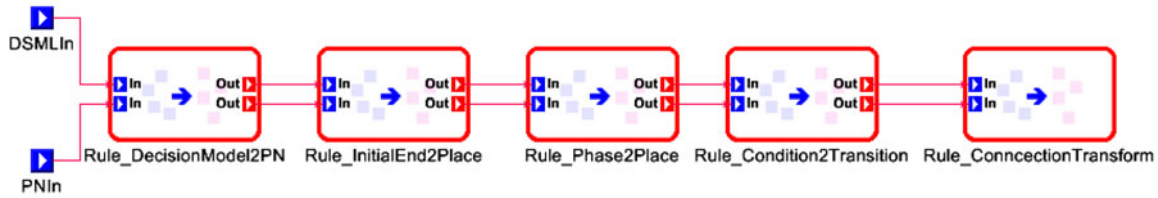

Fig. 8 A transformation model from DSML to Petri Net using GReAT

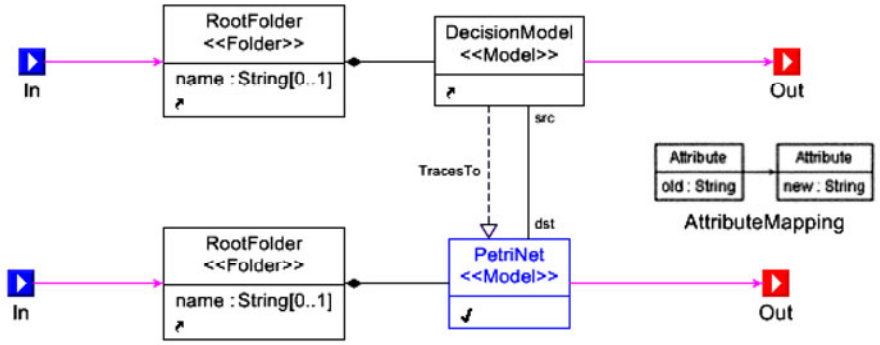

Fig. 9 Transformation rule from DecisionModel to Petri Net

generator that generates Python scripts from the domain-specific models. The code generator traverses the graphic model and generates Python code using the following algorithm:

1. Begin with the root (DecisionModel) and create a Python script file.

2. Define a MainTask function, find the only main taskm, and write the MainTask execution line. Then put the MainTask node generation to a Task generation queue.

3. Traverse the Task generation queue, call the Task generation algorithm to generate a Python function for each Task, and remove the generated Task from the queue. Go to step 4 when the queue is empty.

4. Traverse the TacticAction generation queue, call the TacticAction generation algorithm to generate a Python function for each TacticAction, and remove the generated TacticAction from the queue. Go to step 5 when the queue is empty.

5. Traverse the Phase generation queue, call the Phase generation algorithm to generate a Python function for each Phase, and remove the generated Phase from the queue. Go to step 6 when the queue is empty.

6. Traverse the Condition generation queue, call the Condition generation algorithm to generate a Python function for each Condition, and remove the generated Condition from the queue. Go to step 7 when the queue is empty.

7. Generate other implementation-specific code to complete the file.

The Task generation algorithm is as follows:

1. Begin with the initial TacticAction (Action for short), write a TacticAction execution code line, and put this node generation to a TacticAction generation queue.

2. Follow the output line(s) (ActionSequence, ActionCondition line, or TaskSequence line) to the next node(s).

3. Write a node execution code line for current node(s), and put this node(s) generation to the corresponding generation queue according to its node type (Action, Condition, or Task). If the node is a Condition node, the next node execution line is in the function domain of the Condition node execution line.

4. If the output line exists, go to step 2; otherwise, finish the task generation.

Action generation function generates python code according to the ActionType. There are two categories of Actions: the first is the fake action, namely initial Action and end Action; for these two action nodes, the Python implementation is empty. The second is the tactic action node, and we need to transform the action to a Python implementation (a function). For example, if it is an order to open all radars, the code generated is as follows:

```
def openAllRadars(PlatformInfo)
    PlatformInfo.openAllRadars()
```

A Condition is generated as an IF expression in python code. LogicOperators are generated as corresponding Python logic operators and thus are MathOperators. Variable is generated to corresponding python code according to its VariableType. For example, for the variable Alt which means current altitude of the platform, the python code is PlatformInfo. getAlt(). EventTriggers are also transformed to python code according to their EventTypes. After the IF expression is generated, the node of which the output line connects is within the function domain.

Phase generation function is to generate a Python function to set the phase of the platform after the phase transition. The function is as follows:

```
def setPhase(PlatformInfo)
    PlatformInfo.setPhase(PhaseType)
```

GME provides infrastructure support for building interpreters (in this research, code generator). We use the BON interpreter wizard to automatically generate the framework code and implement the abovementioned algorithms in function InvokeEx() in C++ using Visual Studio (Davis, 2003). The implementation includes three parts: the first is node traverse, done by the functions provided by the framework, the second is the code generation algorithms for each kind of node, and the third is other implementation specific code (e.g., reference to other files). After the coding, we build the code as a DLL file, and register the DLL as an interpreter component for decision modeling DSME to GME platform to generate Python code from the graphic decision models.

6.2 Script-based implementation framework for platform decision models

Though Python scripts discussed above can describe the decision behavior, a framework to incorporate them into the simulation system should be provided. Fig. 10 briefly describes the script-based framework of the simulation system to fully support the implementation of the decision model, which has Task, Event, Timer, and TargetTactic mechanisms (Li *et al*., 2013). Each kind of combat platform model (tmPlatform) has one main task script (MainTask) which specifies the basic decision process, and the platform model will call it 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, MemeoryVariable, and ScriptEvent. Timer mechanism and ScriptEvent mechanism are managed by tmPlatfom and are valid in all scripts of the platform model. MemeoryVariable is managed by tcTarget and is valid in the corresponding TargetTactic script.

The Python script based decision model interacts with the physical domain model via the Python/C++ interface. 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), platform current status (e.g., position and velocity), target, and ally information. For example, in Section 6.1, the python function PlatformInfo.getAlt() transforms to C++ function through the corresponding interface. On the other hand, the physical domain model receives orders from the decision model: movement, subsystem (weapon, sensor, jammer) management, and formation control. For example, in Section 6.1, the python function openAllRadars() transforms to C++ function through the corresponding interface. There is also interface support for the decision model to use Timer, MemeoryVariable, and ScriptEvent mechanisms.

The tactics manager proposed by Son *et al*. (2010) describes scripting language based implementation of the decision models, which is closely related to this script-based implementation framework. However, the two script-based frameworks have the following significant differences:

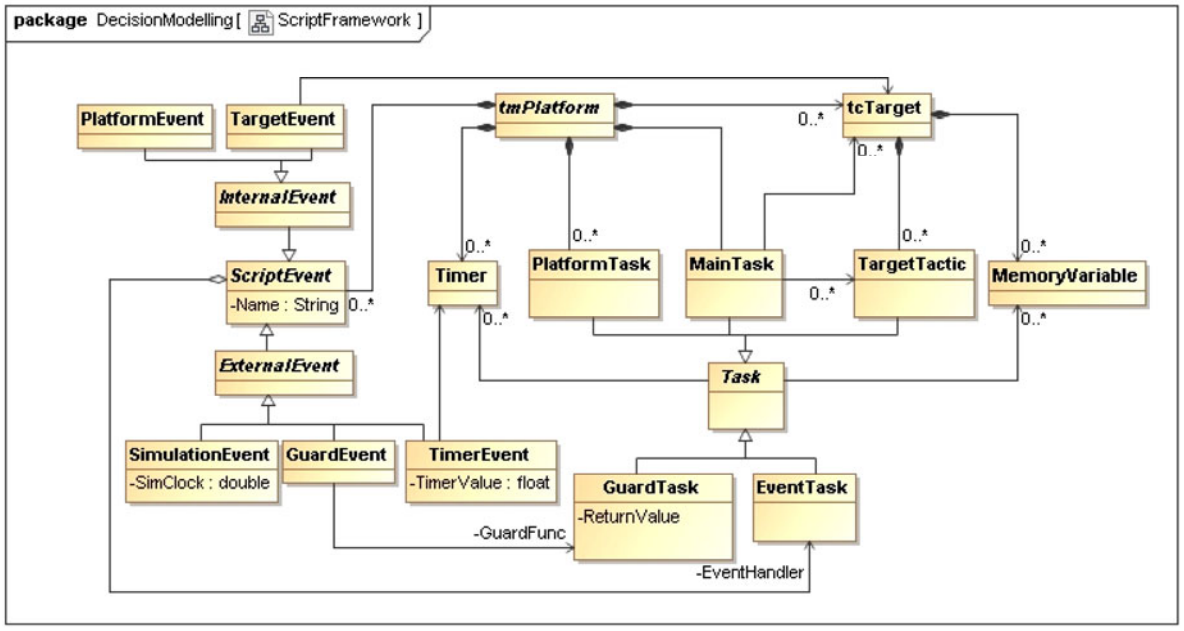

Fig. 10 A script-based platform decision modeling framework

1. The tactics manager mainly models the decision choices for tactics selections. Our script framework not only models tactics selection, but also models the command and control of all affiliate equipments.

2. The tactics manager mainly uses the decision phase mechanism for its model design and implementation. Our script framework not only supports the phase mechanism, but also supports the target mechanism (arrange a tactics for a specific target), event mechanism (arrange a tactics for a specific event), and timer mechanism.

3. The tactics manager is invocated by the simulation model when there are tactic selections to determine. While there is a decision time step for our script framework, and the simulation model calls the decision model in each decision time step.

4. The tactics manager reads the values of state variables of the platform model for tactics determination. While our script framework uses not only the platform state variables as the input, but also platform parameters, target, and ally information.

5. The tactics manager is built for the simulation of a target motion analysis between a submarine and a surface ship; it may also be applicable for other one-to-one warfare decision modeling, but more efforts are needed to support many-to-many warfare decision modeling. On the contrary, our script framework supports the platform decision modeling under the background of SoS warfare.

7 Case study: warship sea battle decision modeling

In this section, we study the decision modeling of warship sea battle to illustrate the usage and applicability of the proposed modeling framework. Firstly, how to model the tactical decision at the domain-specific level and generate Python scripts from the domain-specific models is presented; then, how formalisms can be used to support the formal analysis and verification and validation of domain-specific models is studied.

7.1 Domain-specific modeling of warship decision process

A warship in a sea battle 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 submarines and torpedoes underwater. The warship needs to cope with complex situations, survive under these multiple threats, and achieve the combat goals. Thus, the decision making process is complicated and needs to be modeled correctly.

The basic scenario is as follows. There are two opposed sides in the theater. The red side has a warship with sensors (including radars and sonars) and weapons, whose main task is to navigate along a set of specified waypoints 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 a set of waypoints is assigned to the warship and the warship starts moving towards the first waypoint, and the platforms of the blue side are patrolling along the specified areas around the waypoints.

The top level decision process of the blue warship is illustrated in Fig. 5. The warship is in the Waypoint phase when there are waypoints assigned to it; it will transit to the TargetProcess phase if there is any target detected; for each new target, the warship arranges a task to cope with the target according to its type (air, surface or underwater); the warship will go back to the Waypoint phase 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.

Figs. 11 and 12 describe the warship anti-air (WarshipAntiAir task) and anti-sub tactics (WarshipAntiSub task), respectively. These two decision processes are basically the same except that the tactic actions are tactics-specific. The WarshipAntiAir task is not shown. It shares similar decision logic from the aspect of phase transition: when the target is not threatening (far enough from the warship), the warship is in TargetProcess to travel to the waypoint while keeping a certain distance from the target; and if the target becomes a threat, the warship transits to EvasionPhase to take emergency actions to prevent the situation from being worse; once the target gets out of the dangerous scope, the phase goes back to the

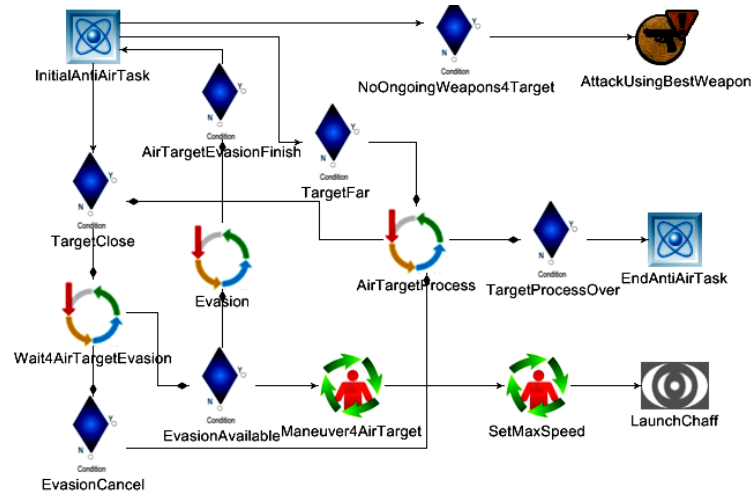

Fig. 11 Decision process of the warship anti-air task

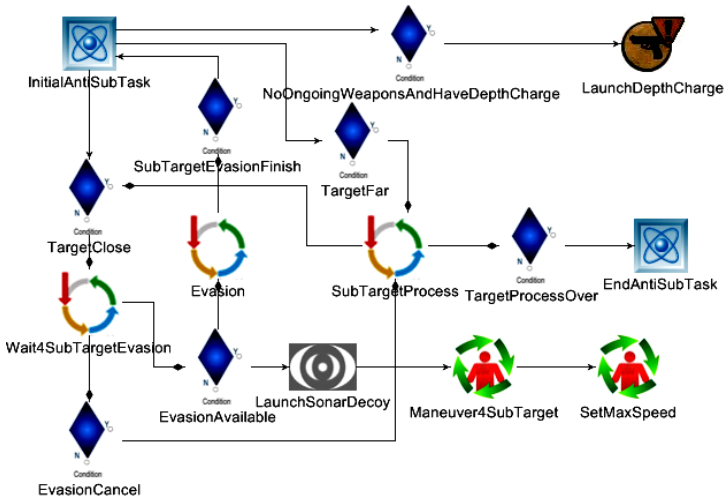

Fig. 12 Decision process of the warship anti-sub task

TargetProcess phase again. The decision model is relatively simple and can execute only a single task for each kind of target.

To test the decision model, the basic scenario is concretized to a series of simulation scenarios that are used to run multiple Monte-Carlo simulations under typical instances of the situation space. To validate the DSM-based framework and script-based implementation, we implement the decision model using three approaches. The first is a domain-specific model in Fig. 5, which is translated to python code automatically (Fig. 13), the second is a manually coded Python script model, and the third is a C++-based model closely coupled with the physical part of the warship model. For each scenario, we run the simulations using all three models and record the model behaviors. Then, we compare the behaviors and the results, and find that all the three models exhibit the same behaviors. However, the efforts to build different forms of decision modeling are not the same: the first method needs manual coding in C++ and recompilation; the second method needs manual python coding; and, the third needs only to revise the graphical model by dragging new icons or deleting old ones, and connecting or disconnecting the links.

7.2 Petri Net based analysis of the decision model

Using the transformation rules specified in Fig. 8, we transform the warship sea battle decision model to a Petri Net model (Fig. 14). Since there are three evasion places in the model which refer to the same place, we modify it using one Evasion place instead while preserving all the links of them. The Petri Net model contains 17 places, which are indexed in Table 4. We set the initial token number of place 1

*WarshipSeaBattleDecisionModel.py - E:\Modeling\DSM\DecisionDSML\Gene...

File Edit Format Run Options Windows Help

```
from Common import *
from random import *

def InitDecision(PlatformInfo):
     PlatformInfo.Debug("Decision script initialized successfully.")
     PlatformInfo.SetPhase(ReadyPhase)

def MainTask(PlatformInfo):
    Action_Initial()
    if (Condition_WaypointAssigned(PlatformInfo)):
        Action_OpenAllRadars(PlatformInfo)
        setPhase_WaypointPhase(PlatformInfo)
    if (Condition_AllTargetProcessed(PlatformInfo)):
        setPhase_TargetProcessPhase(PlatformInfo)
    if (Condition_ReachedLastWP(PlatformInfo)):
        Action_EndMainTask(PlatformInfo)
    if (Condition_NewTargetDetected(PlatformInfo)):
        setPhase_TargetProcessPhase(PlatformInfo)
    if (Condition_NewAirTarget(PlatformInfo)):
        PlatformInfo.AddNewTask(WarshipAntiAir)
    if (Condition_NewSurfaceTarget(PlatformInfo)):
        PlatformInfo.AddNewTask(WarshipAntiSurface)
    if (Condition_NewSubTarget(PlatformInfo)):
        PlatformInfo.AddNewTask(WarshipAntiSub)
    if (Condition_TargetSubTaskFinish(PlatformInfo)):
        setPhase_TargetProcessPhase(PlatformInfo)

def Condition_WaypointAssigned(PlatformInfo):
    if (PlatformInfo.GetWaypointListLength() > 0):
        return True
    else:
        return False
```

Ln: 6 Col: 38

Fig. 13 A generated Python decision script file for warship sea battle

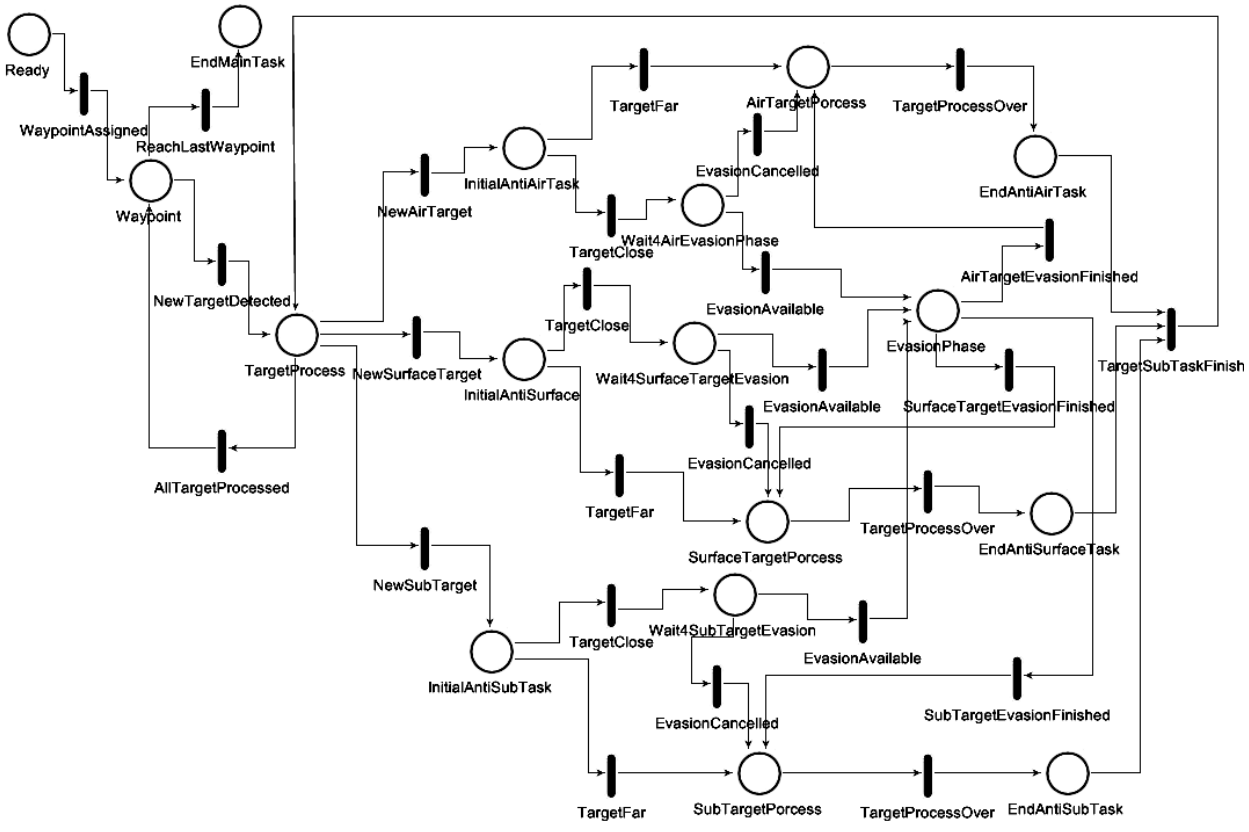

Fig. 14 A Petri Net based warship sea battle decision model

(Ready) to 1 in order to start the state transition and that of place 7 (EvasionPhase) is 2 since it is transformed from the MaxTargetProcessNum value of the TargetProcess phase (precisely speaking, the value 2 is smaller than the MaxTargetProcessNum value by 1 since when a new token flows into this place, the token value will be equal to that value). Thus, the initial marking of the Petri Net model is the same as listed in the second column in Table 4.

Based on the Petri Net decision model, we can perform various kinds of analyses, such as state reachability, boundedness, and liveness. Since the focus of this work is not to study formalism-based formal analysis methods, we concentrate on how to analyze certain characteristics of the decision process based on a typical decision process described in Table 4, which presents a series of markings of the decision model. The 14 markings in Table 4 represent a typical decision process for warship sea battle: the warship firstly travels along a set of specified waypoints (markings 01–02), then it encounters and processes a target in the air (markings 04–08), a target on the surface (markings 06–12), and an underwater target (markings 07–09). These targets are sequentially detected but concurrently processed. After all targets are processed, the warship goes back to the waypoint phase (markings 12–13), and the decision process is terminated when it arrives at the last waypoint (markings 13–14). Since concurrency and conflict are two of the core characteristics of Petri Net, we analyze the concurrency among the three target process tasks and the conflict of the EvasionPhases.

1. Boundedness analysis

Among all the places, only the place TargetProcess can have 3 tokens (i.e., its capacity is 3), and other places can have only 1 token at most (capacity is 1). This Petri Net is bounded since all of the transitions can neither produce new tokens nor diminish tokens. Therefore, the boundedness value of this Petri Net is 3.

2. Task concurrency analysis

The concurrency of the Petri Net exists only when the token number of place 3 is smaller than 2, which means currently there is more than 1 target being processed (markings 06–10). Especially during markings 07–09, there are three targets being processed. The warship needs to appropriately coordinate its resource usage and make decisions based on the comprehensive situation considering all the targets. So, this part of the decision model needs to be tested and verified intensively since the situation is complex and the detection sequence of different targets can have several combinations. Testing different tactics of concurrent task coordination is of vital importance for the warship sea battle effectiveness.

Table 4 An analysis of the warship decision process using the Petri Net decision model

Place	Token number													
	01	02	03	04	05	06	07	08	09	10	11	12	13	14
1. Ready	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2. Waypoint	0	1	0	0	0	0	0	0	0	0	0	0	0	0
3. TargetProcess	2	2	3	2	2	1	0	0	0	1	2	2	3	2
4. InitialAntiAirTask	0	0	0	1	0	0	0	0	0	0	0	0	0	0
5. AirTargetProcess	0	0	0	0	1	0	0	0	0	0	0	0	0	0
6. Wait4AirTargetEvasion	0	0	0	0	0	1	0	0	0	0	0	0	0	0
7. EvasionPhase	0	0	0	0	0	0	1	1	1	0	1	0	0	0
8. EndAntiAirTask	0	0	0	0	0	0	0	0	0	1	0	0	0	0
9. InitialAntiSurface	0	0	0	0	0	1	0	0	0	0	0	0	0	0
10. SurfaceTargetProcess	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11. Wait4SurfaceTargetEvasion	0	0	0	0	0	0	1	1	1	1	0	0	0	0
12. EndAntiSurfaceTask	0	0	0	0	0	0	0	0	0	0	0	1	0	0
13. InitialAntiSubTask	0	0	0	0	0	0	1	0	0	0	0	0	0	0
14. SubTargetProcess	0	0	0	0	0	0	0	1	0	0	0	0	0	0
15. Wait4SubTargetEvasion	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16. EndAntiSubTask	0	0	0	0	0	0	0	0	1	0	0	0	0	0
17. EndMainTask	0	0	0	0	0	0	0	0	0	0	0	0	0	1

3. EvasionPhase conflict analysis

Concurrent tasks inevitably lead to conflicts on resources. Place 7 (EvasionPhase) represents a warship phase to take emergency actions to avoid a certain target. From Figs. 11 and 12, we can see that the evasion needs to maneuver, set the maximum speed, and launch counterwork measures, so the evasion cannot be performed concurrently for multiple targets. In Table 4, place 11 in markings 07–10 has 1 token since the warship has to wait for the surface target evasion when it is evading from an air target. According to the execution semantics of Petri Net, the transition for place 11 cannot fire since the post condition cannot be satisfied; if it is fired, the token number of place 7 would exceed the place capacity. Evasion is an important tactics for the warship sea battle since it is directly related to its survival. Thus, this part needs to be reconsidered and carefully tested. For example, if the target that the warship is waiting to evade is much more dangerous than the target currently evading, then the decision model should be able to arrange evasion for the more dangerous target. Moreover, simulations should be designed accordingly to optimize the conflict solutions.

Though the Petri Net model cannot exhibit all aspects of the domain-specific decision model, it presents the decision model in a precise and formal way. Form the Petri Net model, it is clear to observe the task concurrency and EvasionPhase conflict; and, formal analysis can also be done to analyze certain characteristics of the whole decision model since Petri Net has a sound mathematical foundation. The next step is to integrate mature Petri Net tools to support various formal analyses by transforming GME-based Petri Net models to tool-specific models.

From the case study on the Petri Net based decision model, we find the following three functions of formalism-based decision models transformed from domain-specific models:

1. Early verification and validation can be done since modeling formalisms have unambiguous semantics and formal analysis methods. For example, more Petri Net based decision processes similar to Table 4 can be constructed for verification.

2. Formal analysis methods can help modelers with model testing. Decision models need to be tested via simulations. However, designing simulations for each decision rule is a trivial and taxing task. If more formal analysis has been performed, more insights about the decision model are gained, which will help us understand the important parts, the error-prone parts, and the difficulty parts of the model. Thus, the model testing can be done accordingly.

3. Formalism-based model can help us improve the decision model, identify the key points of the research problem, and set the purpose of simulations. For example, task concurrency analysis and EvasionPhase conflict analysis show that how to make an appropriate decision when targets appear concurrently is a tough issue in warship effectiveness simulation. We can add new tactics, such as two evasion tactics in synthesis based on the comprehensive statuses of the two concurrent dangerous targets, to raise the probability of survival. Simulations can be designed and performed accordingly to test the new tactics.

8 Conclusions and future work

In this paper, we analyzed the cognitive quality of decision modeling and proposed a DSM-based MPM framework to cope with diversity, agility, and higher abstraction level of decision modeling. The framework comprises decision model design and implementation at three abstraction levels. The upper level is the GME-based DSM layer providing domain related modeling concepts that are familiar to tactics modelers to raise the level of abstraction. The middle level is formalism-based environments for the formal analysis and verification and validation of the domain-specific model. The lower level is the Python script based modeling, which separates decision models from physical domain models. According to the MPM methodology, domain-specific models are transformed to formalism-based models using model transformation, and a code generator is built to automatically generate scripts at the implementation level from models of the upper and middle levels. The decision model is incorporated into the whole simulation system via C++/Python interface, which is a part of the script-based platform decision modeling framework. The case study illustrates the usage of the framework at multiple modeling levels of abstraction, and the results show that the DSM-based framework is competent in solving the decision modeling issues proposed in Section 1.

However, there are still more efforts required to improve the framework and enable its industrial length application. Firstly, the current metamodel of DSML is based on simulation modeling practice on common platforms under typical combat scenarios, and thus more platforms and combat scenarios need to be investigated to elaborate the metamodel. Moreover, ontology-based metamodeling techniques can be used to seize the fundamental concepts of the tactical decision domain and cope with diversity efficiently (Walter and Ebert, 2009). Secondly, only Petri Net is explicitly studied at the formalism level. To perform various kinds of formal analysis, more formalisms should be incorporated into the DSM-based framework, and how to enable DSM with multi-formalism modeling is of vital importance if the decision process becomes more complex. Thirdly, there exist many mature tools for the classical modeling formalisms, and the integration of these tools into this framework can provide effective tool supports for formalism-based modeling, analysis, and simulation; moreover, the generation of Python scripts from the formalism-based model should be studied. Finally, more large-scale case studies should be performed to promote the industrial length application of the framework.

References

Balasubramanian, D., Narayanan, A., van Buskirk, C., Karsai, G., 2006. The Graph Rewriting and Transformation Language: GReAT. Proc. 3rd Int. Workshop on Graph Based Tools, **1**:1-8.

Chen, K., Sztipanovits, J., Neema, S., 2005. Toward a Semantic Anchoring Infrastructure for Domain-Specific Modeling Languages. Proc. 5th ACM Int. Conf. on Embedded Software, p.35-43. [doi:10.1145/1086228.1086236]

Davis, J., 2003. GME: the Generic Modeling Environment. Proc. 18th Annual ACM SIGPLAN Conf. on Object-oriented Programming, Systems, Languages, and Applications, p.82-83.

Davis, P.K., Bigelow, J.H., 1998. Experiments in Multiresolution Modeling (MRM). RAND Co., Santa Monica, CA.

Ferayorni, A.E., Sarjoughian, H.S., 2007. Domain Driven Simulation Modeling for Software Design. Proc. 2007 Summer Computer Simulation Conf., p.297-304.

France, R., Rumpe, B., 2005. Domain specific modeling. *Softw. Syst. Model.*, **4**(1):1-3. [doi:10.1007/s10270-005-0078-1]

Hemingway, G., Neema, H., Nine, H., Sztipanovits, J., Karsai, G., 2012. Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach. *Simulation*, **88**(2):217-232. [doi:10.1177/0037549711401950]

Li, F., Shen, X., 2010. A component-based aircraft instrument rapid modeling tool. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **11**(11):911-918. [doi:10.1631/jzus.C1001010]

Li, Q., Lei, Y., Hou, H., Wang, W., 2010. Simulation Model Portability Standard 2 and Its Application. Publishing House of Electronics Industry, Beijing, China, p.344 (in Chinese).

Li, X., Lei, Y., Vangheluwe, H., Wang, W., Li, Q., 2011. Towards a DSM-based Framework for the Development of Complex Simulation Systems. Proc. 2011 Summer Computer Simulation Conf., p.210-215.

Li, X., Lei, Y., Vangheluwe, H., Wang, W., Li, Q., 2013. Domain specific decision modelling and statistical analysis for combat system effectiveness simulation. *J. Stat. Comput. Simul.*, in press. [doi:10.1080/00949655.2013.797421]

Liu, H., Shi, Z., 2010. Intelligent Decision-Making Modeling Based on Object-Oriented Bayesian Network. Proc. 3rd Int. Conf. on Information and Computing, p.300-303. [doi:10.1109/ICIC.2010.347]

Liu, L., 2007. Research on Modeling Fleet Cooperation Decision-Making System Based on Colored Petri Net. Master Thesis, Xidian University, Xi'an, China (in Chinese).

Mittal, S., Douglass, S.A., 2011. From Domain Specific Languages to DEVS Components: Application to Cognitive M&S. Proc. Symp. on Theory of Modeling & Simulation: DEVS Integrative M&S Symp., p.256-265.

Mittal, S., Risco-Martín, J.L., Zeigler, B.P., 2007. DEVSML: Automating DEVS Execution over SOA Towards Transparent Simulators. DEVS Integrative M&S Symp., p.1-9.

Mosterman, P.J., Vangheluwe, H., 2004. Computer automated multi-paradigm modeling: an introduction. *Simulation*, **80**(9):432-450. [doi:10.1177/0037549704050532]

Murata, T., 1989. Petri nets: properties, analysis and applications. *Proc. IEEE*, **77**(4):541-580. [doi:10.1109/5.24143]

Neema, H., Nine, H., Hemingway, G., Sztipanovits, J., Karsai, G., 2009. Rapid Synthesis of Multi-model Simulations for Computational Experiments in C2. Armed Forces Communications and Electronics Association-George Mason University Symp.

Ratzer, A.V., Wells, L., Lassen, H.M., Laursen, M., Qvortrup, J.F., Stissing, M.S., Westergaard, M., Christensen, S., Jensen, K., 2003. CPN tools for editing, simulating, and analysing coloured Petri nets. *LNCS*, **2679**:450-462. [doi:10.1007/3-540-44919-1_28]

Sarjoughian, H., Huang, D., 2005. A Multi-formalism Modeling Composability Framework: Agent and Discrete-Event Models. 9th IEEE Int. Symp. on Distributed Simulation and Real-Time Applications, p.249-256. [doi:10.1109/DISTRA.2005.4]

Sarjoughian, H.S., Zeigler, B.P., 2000. DEVS and HLA: complementary paradigms for modeling and simulation? *Simulation*, **17**(4):187-196.

Seo, K.M., Song, H.S., Kwon, S.J., Kim, T.G., 2011. Measurement of effectiveness for an anti-torpedo combat system using a discrete event systems specification-based

underwater warfare simulator. *J. Def. Model. Simul.*, **8**(3): 157-171. [doi:10.1177/1548512910390245]

Sokolowski, J.A., 2003. Enhanced decision modeling using multiagent system simulation. *Simulation*, **79**(4):232-242. [doi:10.1177/0037549703038886]

Son, M., Kim, T., 2012a. Torpedo evasion simulation of underwater vehicle using fuzzy-logic-based tactical decision making in script tactics manager. *Expert Syst. Appl.*, **39**(9): 7995-8012. [doi:10.1016/j.eswa.2012.01.113]

Son, M., Kim, T., 2012b. Maneuvering control simulation of underwater vehicle based on combined discrete-event and discrete-time modeling. *Expert Syst. Appl.*, **39**(17): 12992-13008. [doi:10.1016/j.eswa.2012.05.099]

Son, M.J., Cho, D.Y., Kim, T., Lee, K.Y., Nah, Y.I., 2010. Modeling and simulation of target motion analysis for a submarine using a script-based tactics manager. *Adv. Eng. Softw.*, **41**(3):506-516. [doi:10.1016/j.advengsoft.2009.10.009]

Sprinkle, J., Rumpe, B., Vangheluwe, H., Karsai, G., 2011. 3 Metamodelling. *LNCS*, **6100**:57-76. [doi:10.1007/978-3-642-16277-0_3]

US Army Space and Missile Defense Command, 2012. EADSIM Executive Summary. Available from http://www.eadsim.com/EADSIMExecSum.pdf/ [Accessed on Apr. 21, 2013].

Verbraeck, A., Valentin, E.C., 2008. Design Guidelines for Simulation Building Blocks. Proc. Winter Simulation Conf., p.923-932.

Walter, T., Ebert, J., 2009. Combining DSLs and Ontologies Using Metamodel Integration. Proc. Working Conf. of Domain-specific Languages, p.148-169.

Wang, W., Wang, W., Zander, J., Zhu, Y., 2009. Three-dimensional conceptual model for service-oriented simulation. *J. Zhejiang Univ.-Sci. A*, **10**(8):1075-1081. [doi:10.1631/jzus.A0920258]