



# Exploring MDE techniques for engineering simulation models

Zhi Zhu<sup>1</sup> · Yonglin Lei<sup>1</sup> · Qun Li<sup>1</sup> · Yifan Zhu<sup>1</sup>

Published online: 1 January 2020  
© Springer Science+Business Media, LLC, part of Springer Nature 2020

## Abstract

A recent approach to tackle the ever increasing complexity of simulation system is model-driven engineering (MDE). However, it is mostly used to produce simulation tools, and seldom can perform formal analysis. Consequently, this raises issues like poor qualities of product, and falls short of non-functional requirements such as extensibility, maintainability, and reuse. In general, many of the success of MDE projects depend on the descriptive power of modeling languages and how conceptual models are implemented. Hence, this paper presents contributions in two main aspects: customizing domain specific language by metamodeling and enhancing model continuity by formalizing model transformations. A military application is used as a motivating example to illustrate the whole process by transforming the conceptual models into other more precise formalisms until they reach final executable models.

**Keywords** MDE · Metamodeling · Model transformation · Model continuity

## 1 Introduction

Traditional military simulation models are usually represented by UML which has not precise and unambiguous semantics defined using a mixture of OCL (Object Constraint Language) and informal text, or the semantics of simulation models are left to model interpreters or simulators which are defined by general-purpose programming languages, which is clearly unacceptable for formal analysis [1]. Meanwhile, although the syntax of current domain specific modeling languages (DSML) are formally described with a lot of general metamodeling tools like UML Profile [2], EMF [3], and GME [4] etc., the semantics are left toward other less than desirable means [5]. All of these

accompanying with the lack of formal model transformations contribute to difficult formal analysis at a model level. Hence, it is a real challenge to describe simulation models formally, and to improve the model continuity that exist between different models in different development stages at different levels of abstraction [6], so as to reuse existing model assets and simulation services to a great degree.

This paper proposes a set of formal theories of model transformations for engineering model continuity, transforming models represented by various modeling languages into other formalisms that have precise definitions of semantics until they reach final executable simulation models [7]. A motivating example named group fire control channel system (GFCCS) is used through this paper, commencing with its customization of DSML and transforming its conceptual models represented in this DSML to final executable simulation models. After that, a military simulation system in support of engineering modeling and composable simulating is capable of integrating those executable simulation models and reusing them for multiple simulation applications.

## 2 Background

Inconsistent terminology in the model-driven engineering (MDE) context [8] means it is necessary to define basic meanings of important frequently used terms to provide a

---

✉ Zhi Zhu  
zhuzhi@nudt.edu.cn  
Yonglin Lei  
yllei@nudt.edu.cn  
Qun Li  
liqun@nudt.edu.cn  
Yifan Zhu  
yfzhu@nudt.edu.cn

<sup>1</sup> Department of Military Modeling and Simulation, College of Systems Engineering, National University of Defense Technology, Changsha 410073, People's Republic of China

common understanding. Many of these terms are used alternatively in specific contexts, but providing their definitions and/or subtle distinctions is helpful to understand the methodologies, techniques, and tools used in model-driven development. For different modeling goals, there exist three typical issues, i.e. model composability, model heterogeneity, and model continuity.

Firstly, model composability [9, 10] concentrates on the syntactical matching and semantic relations between different simulation models. Unlike the other two issues, it is usually discussed in a MDE context and emphasizes the integration of multiple simulation models to form an effective and meaningful simulation application [11, 12].

Secondly, model heterogeneity comes from the joint use of several DSMLs dedicated to particular domains or applications. In many cases, it refers to the syntactical incompatibility between different used DSMLs during the language customization process and has four sources in general [13, 14]. Firstly, the different technical or application domains involved in a simulation system under design require different model specifications, modeling formalisms, or simulation protocols. Secondly, the different levels of abstraction need suitable modeling techniques. Thirdly, a simulation system is always studied from different points of view, and lastly different stages of a development cycle may use different languages for different activities.

Thirdly, model continuity refers to the generation of an approximate morphism relation between different phases of a development process [15]. In general, model continuity is obtained if the initial and intermediate models are effectively consumed in the later steps of a development process and the modeling relation is preserved. In a sense, model continuity is similar to the consistency between the source and target models, involving the syntactical correctness of target model, the completeness of source model consumed in model transformation, and the semantic relations preserved in target model [16].

### 3 Model transformations

Model transformation is a process that takes a source model in a specific form as an input and outputs another form of the target model according to a set of predefined rules. This process does not build new models from scratch, but reuse existing information when conducting a model transformation. A formal model transformation requires that the models involved in the transformation are represented clearly by well-defined modeling languages that have accurate syntax and unambiguous semantics. Furthermore, it requires that the transformation rules are written by a well-defined transformation language to ensure the transformation is conducted under a well-

defined transformation template [17]. To ensure model continuity, the target model should preserve as much as possible the initial model information and modeling relations that are embedded in the source model [18].

#### 3.1 The basic model transformation mode

Figure 1 shows the basic mode of model transformation. In this mode, each node at a certain layer conforms to or is an instance of the node at a higher layer. The middle column is the transformation mechanism that inputs the left source nodes and outputs the right target nodes. For example, the transformation engine is an instance of the transformation template which is further an instance of the transformation language, which means the transformation template is written by the transformation language and prescribes the internal mechanism of the transformation engine. This engine inputs the source model which is an instance of the source metamodel and outputs the target model which is an instance of the target metamodel, and both metamodels are respectively taken as the inputs and outputs of the transformation template [19, 20].

According to the concrete form of target model, model transformation has two typical categories: model to model (M2M) and model to text (M2T). In practice, M2T transformation is also called code generation when the text is in the form of source code. In general, a model-driven development process contains a sequence of M2M transformations and a final code generation. In addition, model transformation is endogenous when the source metamodel is similar to the target metamodel, and exogenous when they are different [21].

#### 3.2 MDA based model transformations

MDA (model-driven architecture) introduces three model development roles, and two transform mechanism types [22]. Using these MDA models, i.e. conceptual independent (CIM), platform independent (PIM), and platform specific models (PSM), developers can be classified into

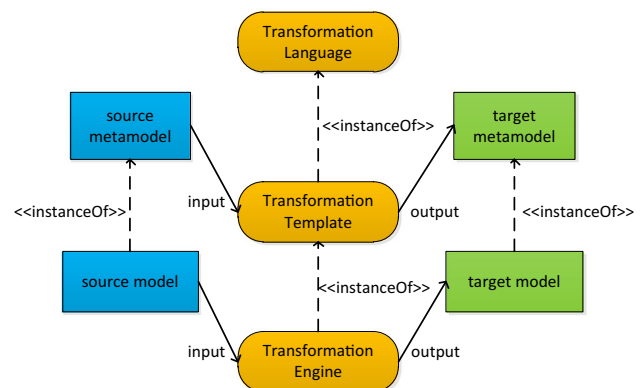


Fig. 1 The basic model transformation mode

comparable roles, i.e. conceptual and simulation modelers, and simulation programmers, respectively, where later stages only can commence when developers for the former stages reach a consensus on an artifact. For example, once the problem owner and the conceptual modeler agree on a conceptual model, the simulation modeler can transform it into a formal model. In addition, M2M and M2T model transformation mechanisms are used as a bridge to reduce the gap between these roles. We adopt the formal MDA process as depicted in Definition 1 [15].

**Definition 1** AMDA process is defined as  $mda = \{n, MML, ML, MO, SL, pl, MTP, STP, MT, SM, TO\}$

- $n = 3(CIM, PIM, PSM)$ ,
- $MML = \{ll_0, ll_1, ll_2\}$  is an ordered set of metamodeling languages,
- $ML = \{l_0(mm_{CIM}), l_1(mm_{PIM}), l_2(mm_{PSM})\}$  such that  $confromTo(mm_{CIM}, ll_0)$ ,  $confromTo(mm_{PIM}, ll_1)$ ,  $confromTo(mm_{PSM}, ll_2)$ ,

This means metamodels must conform to their corresponding metamodeling languages.

- $MO = \{CIM, PIM, PSM\}$  such that  $CIM$  is the initial model,  $PSM$  is the final model, and  $instanceOf(CIM) = mm_{CIM}$ ,  $instanceOf(PIM) = mm_{PIM}$ ,  $instanceOf(PSM) = mm_{PSM}$ ,

This means models must conform to their corresponding metamodels.

- $SL$  is a set of model transformation languages,
- $pl$  is a programming language with simulation capabilities,
- $MTP = \{p_{CIM}, p_{PIM}, p_{PSM}\}$  such that  $p_{CIM} = \{l_0(mm_{CIM}), l_1(mm_{PIM}), r_0\}$ ,  $p_{PIM} = \{l_1(mm_{PIM}), l_2(mm_{PSM}), r_1\}$ ,  $p_{PSM} = \{l_2(mm_{PSM}), pl, r_2\}$ ,

This represents model transformation patterns that a source language to a target language through some rules.

- $STP$  is a set of other supplementary formal model transformation patterns,  $MT = \{transformTo(CIM, p_{CIM}) = PIM, transformTo(PIM, p_{PIM}) = PSM, transformTo(CIM, p_{PSM}) = SM\}$ ,

This means model transformations that a source model to a target model using some patterns.

- $SM$  is the final executable simulation model,
- $TO$  is a set of tools to ease the activities.

Above definition is suitable for general model development base on the MDA principles. Given this definition, we can conclude a process for the GFCCS development as Definition 2, which will be illustrated by later sections. In the GFCCS process, we take the GFCCS DSML as the conceptual modeling language to describe CIM, P-DEVS [23] as the modeling formalism to define PIM, and JAVA as the programming language to build PSM. Hence, the GFCCS process involves the following types of meta-models and model transformations.

- The CIM metamodel is GFCCS metamodel.
- The PIM metamodel is P-DEVS metamodel.
- The PSM metamodel is JAVA metamodel.
- The CIM–PIM transformation is GFCCS to P-DEVS transformation.
- The PIM-PSM transformation is P-DEVS to JAVA transformation.
- The PSM-SM transformation is JAVA to java code transformation.

**Definition 2** A GFCCS simulation modeling process is defined as  $gfccs = \{n, MML, ML, MO, SL, pl, MTP, STP, MT, SM, TO\}_{instance}$

- $n = 3(CIM, PIM, PSM)$ ,
- $MML = \{Ecore, Ecore, Ecore\}$  is an ordered set of metamodeling languages,
- $ML = \{l_0(mm_{GFCCS}), l_1(mm_{DEVS}), l_2(mm_{JAVA})\}$  such that

$confromTo(mm_{GFCCS}, Ecore)$ ,  $confromTo(mm_{DEVS}, Ecore)$ ,  $confromTo(mm_{JAVA}, Ecore)$ ,

- $MO = \{CIM, PIM, PSM\}$ , and  $instanceOf(CIM) = mm_{GFCCS}$ ,  $instanceOf(PIM) = mm_{DEVS}$ ,  $instanceOf(PSM) = mm_{JAVA}$ ,

- $SL = \{ATL, Acceleo\}$  is a set of model transformation languages,
- $pl = JAVA$  is the final programming language
- $MTP = \{p_{CIM}, p_{PIM}, p_{PSM}\}$  such that  $p_{CIM} = \{l_0(mm_{GFCCS}), l_1(mm_{DEVS}), gfccs2devs.atl\}$ ,  $p_{PIM} = \{l_1(mm_{DEVS}), l_2(mm_{JAVA}), devs2java.atl\}$ ,  $p_{PSM} = \{l_2(mm_{JAVA}), JAVA, java2code.mtl\}$ ,

- $STP = \emptyset$  dictates there exists no other supplementary formal model transformation patterns,  $MT = \{$

$transformTo(CIM, p_{CIM}) = PIM,$

- $transformTo(PIM, p_{PIM}) = PSM,$

$transformTo(CIM, p_{PSM}) = SM$

$\},$

- *SM* is the final executable simulation model,
- $TO = \{ATL, Acceleo, EMF, GMF, Eclipse\_IDE\}$ .

### 3.3 Criteria for evaluating model continuity

Model transformation is an automated process of modifying and creating one or several target models from one or several source models. The aim of model transformation is to save effort and reduce information loss as much as possible by automating model building and modification where possible [24, 25]. The key to designing a successful model transformation is a set of formal transformation rules to improve model continuity. Although there is no general guidance to define a good model transformation, we can evaluate model continuity according to the following criteria.

1. *Correctness* A model transformation is syntactically correct if the target model conforms to the target metamodel specification [26], and semantically correct if the target model contains information as much as possible from the source model [27].
2. *Completeness* A model transformation is complete if the target model has a corresponding element for each element in the source model.
3. *Uniqueness* A model transformation is unique if there are no two identical elements in the generated target model [28].
4. *Determinism* A model transformation is determinate if it produces a uniquely defined target model output for each specific source model input [29].

## 4 Customizing a DSML based on metamodeling

### 4.1 Metamodeling based on EMF

Metamodeling is an important mean to design DSMLs [30, 31], especially for EMF usually has close relationships with a set of OMG standards, like UML, MOF, XMI, and MDA, etc. Firstly, UML is widely used to capture various concerns of a certain system by an object-oriented method, emphasizing multi-view to describe the structure, behavior, function, and deployment, etc. While, EMF as a way of defining metamodels is only concerned with one aspect of a system, i.e. class structure. Secondly, EMF/Ecore focuses on the tool sets not the metadata warehouse management, thus avoiding some of the complex issues such as data structure, package relationships, and associations compared to MOF. Thirdly, XMI is a widely accepted serializing standard which is not only used as the format for serializing

EMF models, but also suitable for serializing the meta-model, i.e. Ecore itself. This method is very different with the UML profiling mechanism because it defines meta-models from scratch without considering the UML rules [32]. Hence, it has the potential for the most direct and succinct expression of domain concepts. Furthermore, it has a collection of supporting tools (e.g. GEF and GMF) thanks to the Eclipse open source architecture. Recently, some researches also have identified the need of domain specific metamodeling to avoid the general metamodeling facilities like UML and EMF [33].

Using EMF, we take the GFCCS as a motivating example to illustrate how to design and develop a DSML for this domain. This language is able to directly use the domain concepts and relationships to build a model. For this purpose, it is necessary for a language engineer to have a good background of GFCCS. Many of today’s GFCCS often combine multiple subsystems. A group of air defense, for example, combines the command and control center, sensor, weapon, and a variety of countermeasures.

Figure 2 shows the metamodel of GFCCS. This metamodel consists of a basic diagram node named GroupFireControlSystem and two mutually related nodes named Node and Connection respectively. The Node derives a set of domain concepts such as Group, GroupNode, Weapon, Target, and Channel, which are connected by specific relationships. For example, two groups can share common information by the relation tagged as COPShare that represents common operation picture (COP) [34]. A group can have one or multiple members and zero or multiple weapons which can also be equipped by a group member. A group member may be disjoint with or affiliated by itself, and can control zero or multiple fire control channels. Each channel may be mutually exclusive with itself. It embraces two dynamic entity lists, i.e. weaponList and targetList. These two lists are used to

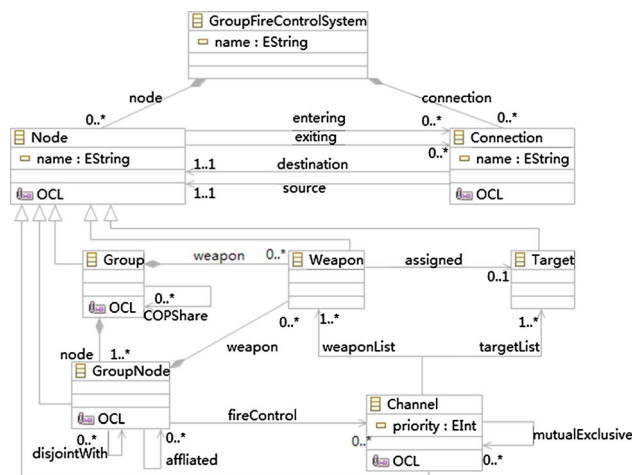


Fig. 2 The GFCCS metamodel

**Table 1** GFCCS domain specific constraints using OCL

OCL static semantics	Descriptions
context <i>Group</i> inv: <i>hasNotDisjointGroupNodes</i> <i>self.node-&gt;forAll(n1,n2 n1.disjointWith-&gt;select(dis dis.name=n2.name)-&gt;isEmpty()) and n2.disjointWith-&gt;select(dis dis.name=n1.name-&gt;isEmpty())</i>	A group can never own two disjoint members
context <i>GroupNode</i> inv: <i>hasNotDisjointChannels</i> <i>self.fireControl-&gt;forAll(c1,c2 c1.mutualExclusive-&gt;select(dis.name=c2.name)-&gt;isEmpty()) and c2.mutualExclusive-&gt;select(dis dis.name=c1.name-&gt;isEmpty())</i>	A group node can never own two mutual exclusive fire control channels
context <i>GroupNode</i> inv: <i>notDisjointWithItself</i> <i>self.disjointWith-&gt;select(dis dis.name=self.name)-&gt;isEmpty()</i>	No group node can be disjoint with itself
context <i>GroupNode</i> inv: <i>notDisjointWithItsDownLevelNode</i> <i>self.affiliated-&gt;forAll(n self.disjointWith-&gt;select(dis dis.name=n.name)-&gt;isEmpty())</i>	No group node can be disjoint with its senior node
context <i>GroupNode</i> inv: <i>notDisjointChannelsOfDownLevelNodeToItself</i> <i>self.affiliated-&gt;forAll(n self.fireControl-&gt;forAll(c1 n.fireControl-&gt;forAll(c2 c1.mutualExclusive-&gt;select(dis dis.name=c2.name)-&gt;isEmpty())))</i>	Two mutual fire control channels may never be assigned to the same group node and no junior node is allowed to possess a fire control channel that is mutual exclusive to the fire control channel assigned to one of its senior nodes
context <i>Channel</i> inv: <i>channelHasContents</i> <i>not weapon.oclIsUndefined() implies target-&gt;notEmpty() and not target.oclIsUndefined() implies weapon-&gt;notEmpty</i>	No fire control channel owns solely a weapon or a target

manage weapons and targets that are alive or may be already ruined [35, 36].

In fact, except the abstract syntax as described above, there are other aspects need to be detailed for a well-defined metamodel [37]. Table 1 defines the static semantics of GFCCS metamodel, written by OCL [38], explaining how those elements of the abstract syntax model can be organized as a valid GFCCS metamodel.

## 4.2 Graphical definitions of GFCCS using GMF

This subsection presents a guidance of the definition, mapping, and generation of a graphical editor for GFCCS using GMF. According to the GMF dashboard, one can define the domain model, domain gen model, tooling model, graphical model, mapping model, and gmf gen model step by step, then generate the diagram editor. Due to the domain model and gen model belong to the field of abstract syntax definition as already given before, so we describe the definition of concrete syntax as follows [39].

1. *Tooling model* As usual, the tooling model definition provides six ways to define a tool palette of a graphical

editor, including creation tool, standard tool, generic tool, tool group, palette separator, and image. In GFCCS, we created a tool for each element of the domain model except the abstract element Node, and bundled a representative image for each element.

2. *Graphical model* It defines the concrete display of modeling elements that will be used in the graphical editing environment. In general, GMF provides default display based on the domain model, but one usually needs to define the figure gallery, figure descriptor, and polyline decoration in practice. In GFCCS, we set the Group and GroupNode as compartments to be able to contain other elements, for example, Group can contain GroupNode and Weapon, and GroupNode can contain Weapon. Additionally, we set the Channel as a scalable polygon, adding template points (0, 0), (40, 0), (40, 30), (30, 30), (30, 40), (40, 30), (30, 40), (0, 40).
3. *Mapping model* When the domain model, tooling model, and graphical model are ready, it is necessary to map them into a whole. Usually, we need to select the corresponding tooling nodes and diagram nodes for each node mapping, and provide the correct

compartment figure for each compartment node. In the properties of Channel node, for example, we select the Node Channel (Channel Figure) for the diagram node, and the Creation Tool Channel for the tooling node.

4. *GMF gen model* If the mapping model is defined correctly, it can generate correct gmf gen model without much modifications. In many cases, it is possible to modify some parameters, such as the fixed background, the list layout, and the suffix of a diagram project.

### 4.3 The GFCCS DSME

Figure 3 shows a simple example of building an engagement scenario using the GFCCS DSML. On the tool palette, this domain specific modeling environment (DSME) contains a set of buttons decorated with professional denotations, including the basic language elements such as GroupFireControlSystem, Group, GroupNode, Channel, Weapon, and Target as well as various relationships. Using this environment, it is possible for domain experts to use these language elements intuitively and friendly. For example, one can draw an arrow from Channel List 1 to Enemy Fighter 1 BLUE only by ChannelTargetList, disabling the use of other relationship buttons.

In the editor, we create a scenario of many to many combat between two opposite sides RED and BLUE. The RED side consists of two defense groups, i.e. Defense

System RED and Remote Surveillance System RED, which refers to the local defense system (e.g. air defense base) and the remote surveillance system (e.g. satellite). The local system is composed of a warship platform, a ground-to-air missile base, and a helicopter platform, which are armed with two surface-to-air missiles, a ground-to-air missile, and a homing torpedo, respectively. The BLUE side consists of three coming threats which are denoted by Enemy Fighter 1 BLUE, Enemy Fighter 1 BLUE, and Enemy Submarine BLUE. In addition, there exist two lists of fire control channel which are denoted by Channel List 1 and Channel List 2, respectively. The former list is managed by the warship platform and has a weapon-target pair, i.e. Surface2AirMissile1-Enemy Fighter 1 BLUE. The latter list is managed by the helicopter platform and has two weapon-target pairs.

## 5 GFCCS implementations

### 5.1 The rules of GFCCS to P-DEVS to JAVA transformations

The matching rules of GFCCS to P-DEVS transformation is defined by using the GFCCS metamodel and P-DEVS metamodel, as detailed in Table 2. The basic diagram node GourpFireControlSystem matches with the DEVSMODEL element. The compartmental nodes like Group and GroupNode match with coupled components named

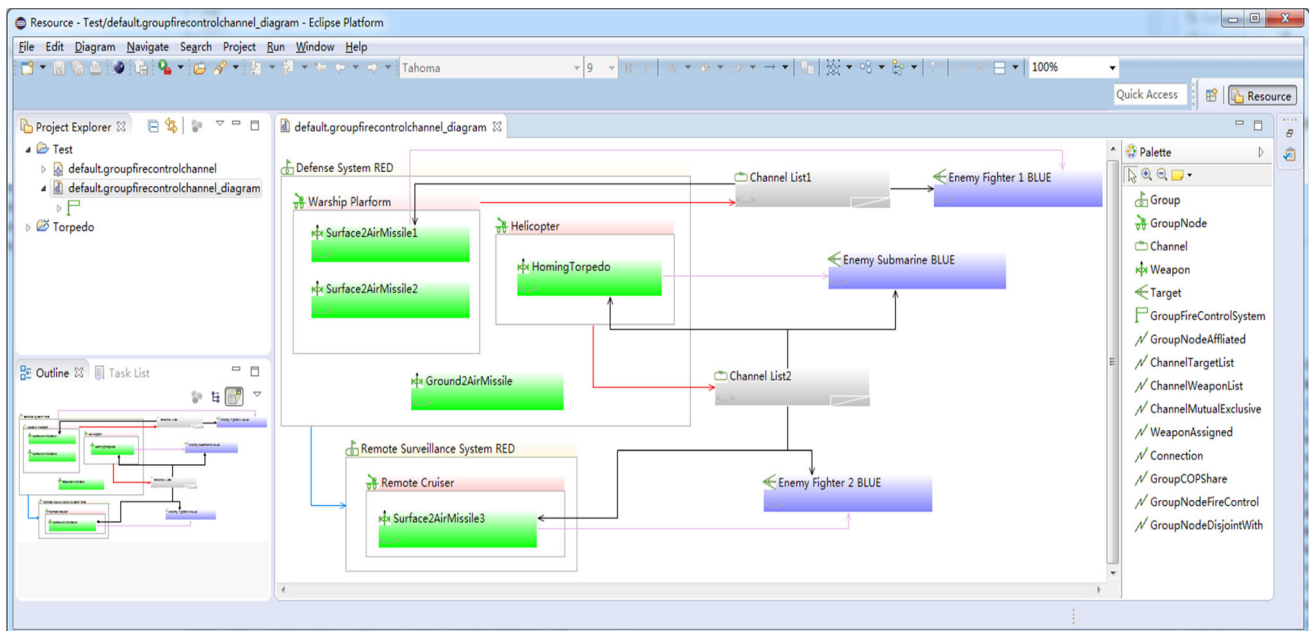


Fig. 3 The GFCCS DSME

DEVSCoupledComp. The connections that connect the modeling elements of the same layer are transformed into internal transitions named DEVSOutToIn\_ICConnection, while those across different layers are transformed into external transitions and output functions with a set of ports.

The matching rules of P-DEVS to JAVA transformation is defined by using the P-DEVS metamodel and JAVA metamodel, as detailed in Table 3. The basic element DEVSModel matches with a JAVA package named JAVAPackage which includes javaClasses, javaConstructors, and javaExpressions. Both the coupled component DEVSCoupledComp and the atomic component DEVSA-atomicComp match with JAVA classes which include JAVAVariables, javaConstructors, and javaExpressions. DEVSInputPort, DEVSOutputPort, and StateVariable are all transformed into JAVAVariables. The connections like DEVSOutToIn\_ICConnection, DEVSInToIn\_EICConnection, DEVSOutToOut\_EOConnection as well as Expression are transformed into JAVAExpressions. The remaining functions like DeltaIntFunction, DeltaExtFunction, LambdaFunction, TimeAdvanceFunction, DeltaConFunction are all transformed into JAVAMethods.

Such transformation rules are written in ATL, as proposed in Definition 2. ATL, the Atlas Transformation Language, is a model transformation language specified as

both a metamodel and a textual concrete syntax. In the MDE field, ATL provides developers with a means to specify the way to produce a number of target models from a set of source models. An ATL transformation program is composed of rules that define how source model elements are matched and navigated to create and initialize the elements of the target models. Besides, ATL Integrated Development Environment (IDE) provides a number of standard development tools (syntax highlighting, debugger, etc.) that aim to ease the design of ATL transformations. The ATL development environment also offers a number of additional facilities dedicated to models and metamodels handling. These features include a simple textual notation dedicated to the specification of metamodels, but also a number of standard bridges between common textual syntaxes and their corresponding model representations.

## 5.2 Automatic code generation

Based on ATL, the transformations of GFCCS to P-DEVS to JAVA instances contains two projects named GFCCS2P-DEVS and P-DEVS2JAVA, each of which includes three packages, i.e. Metamodels, Models, and TransformationEngine. In the edit area includes three instances for the GFCCS, P-DEVS, and JAVA metamodel

**Table 2** The matching rules of GFCCS to P-DEVS transformation

GFCCS metamodel	P-DEVS metamodel
GroupFireControlSystem	DEVSModel
Group	DEVSCoupledComp
GroupNode	DEVSCoupledComp
Channel	DEVSAAtomicComp
Weapon	DEVSAAtomicComp
Target	DEVSAAtomicComp
COPShare	DEVSOutToIn_ICConnection
disjointWith	DEVSOutToIn_ICConnection
affiliated	DEVSOutToIn_ICConnection
fireControl	DEVSOutToOut_ICConnection + Source.out: DEVSOutputPort + Target.in: DEVSOutputPort +SourceParents.EOCPorts: DEVSOutputPort
mutualExclusive	DEVSOutToIn_ICConnection
targetList	DEVSOutToIn_ICConnection
assigned	DEVSOutToOut_ICConnection + Source.out: DEVSOutputPort + Target.in: DEVSOutputPort +SourceParents.EOCPorts: DEVSOutputPort
weaponList	DEVSInToIn_ICConnection + Source.out: DEVSInputPort + Target.in: DEVSInputPort +SourceParents.EICPorts: DEVSInputPort

**Table 3** The matching rules of P-DEVS to JAVA transformation

P-DEVS metamodel	JAVA metamodel
DEVSMODEL	JAVAPackage + javaClasses: JAVAClass +javaConstructors: JAVACONSTRUCTOR +javaExpressions: JAVAExpression
DEVSCoupledComp	JAVAClass +JAVAVariable +javaConstructors: JAVACONSTRUCTOR +javaExpressions: JAVAExpression
DEVSAAtomicComp	JAVAClass +JAVAVariable +javaConstructors: JAVACONSTRUCTOR +javaExpressions: JAVAExpression
DEVSIInputPort	JAVAVariable
DEVSOOutputPort	JAVAVariable
StateVariable	JAVAVariable
DEVSOOutToIn_ICCConnection	JAVAExpression
DEVSIInToIn_EICCConnection	JAVAExpression
DEVSOOutToOut_EOCCConnection	JAVAExpression
Expression	JAVAExpression
DeltaIntFunction	JAVAMethod
DeltaExtFunction	JAVAMethod
LambdaFunction	JAVAMethod
TimeAdvanceFunction	JAVAMethod
DeltaConFunction	JAVAMethod

respectively, and two ATL files for the GFCCS to P-DEVS and to JAVA transformations.

For the M2T transformation, it contains a source model and the generated code framework. Take the node Defense\_System\_RED as an example, the transformation model design should incorporate all the source model required information to satisfy completeness. The target code framework is automatically generated, but the concrete logic details must be manually implemented. In practice, not every concrete detail should be considered when designing M2T transformation models, because they may heavily burden the design phase.

Following the M2T transformation principles, the general source model is some instance models that must conform to a certain metamodel, and the target model can be text, e.g. java, C++, python, etc. Transformation model design is vital to implement the M2T transformation. This paper performed M2T transformation using Acceleo, a template based code generator incorporating a code generation editor with syntax highlighting, completion, real time error detection, and refactoring. The source model was a collection of JAVA instance models, represented by

an instance file named JAVAcase.xmi, and the target model was described in Java programming text.

### 5.3 Model continuity in GFCCS implementations

The process of GFCCS implementation showed that model continuity between different development stages is obtained when applying the formal transformation definition successfully. As stated earlier, it is possible to provide model continuity in a development process when transforming the initial and intermediate models, and preserving the modeling relations during the transformations. To evaluate model continuity, we already presented the criteria for model transformations before. According to these criteria, we describe how the model continuity is obtained for the process of GFCCS implementation when the formal transformation rules are applied. In fact, except these criteria as listed, there are some other non-functional requirements such as termination and readability satisfied. Also note that maintainability, scalability, reusability, evolvability, efficiency, etc. are partially supported since these requirements need more experiments for a better evaluation [40, 41].



The process of GFCCS implementation has two kinds of model transformation, with different expressions and output types. As the source input or target output of model transformations, the formal definition gives three model types: independent of computing details, independent of the computing platform, and specific to a particular computing platform, and two model transformation categories.

In the M2M category, the transformation focuses on the design of a set of formal rules to ensure model continuity when transforming CIM to PIM and to PSM. The transformation usually incorporates three steps.

1. All source concepts, relationships, and domain specific rules are transformed into particular target elements, connections, and domain specific constraints, respectively.
2. Compare all target elements, connections, and constraints to delete identical expressions.
3. Check the target model conforms to the target metamodel.

Completeness can be ensured in step (1) since all source elements are transformed, and a corresponding target element can be found for each source element. Step (2) is helpful and necessary to reduce target element redundancy, thus uniqueness is guaranteed. Syntactical correctness can be satisfied in step (3) since the target model will be expressed in a given formalism, and its semantic correctness will be evaluated in later stages of model transformation. Determinism is guaranteed implicitly in the model transformation editor that eases development and execution of ATL transformations [42].

In the M2T category, the transformation converts a source model into a text file, i.e. PSM to source code. If the text is in source code form, then the transformation is also called code generation, and the transformer is also called a code generator. The process of a M2T transformation is similar to that of a M2M transformation. The only difference is that step (2) in the M2M transformation can be skipped in a M2T transformation, since uniqueness has already been checked. Therefore, the three criteria listed above are achieved according steps (1) and (3). Similarly, determinism is satisfied because model transformation editors, such as Acceleo [43], implicitly guarantee a unique output for each particular input.

#### 5.4 Simulation results and analysis

Once the code is generated and verified correctly, it needs to be integrated into a powerful simulation platform to validate the effectiveness and superiority of the proposed MDE techniques. The WESS (Weapon Effectiveness Simulation System), an engagement-level effectiveness simulation platform that has been applied successfully in

many institutes and industries, is chosen. This platform contains a tool suite including scenario editing, data preparation, Python-based script decision modeling, experiment design, 2D/3D displayer, and evaluation analysis, etc. In general, WESS has two workflows. One is to design and development simulation models from the professional simulation modeler's perspective, and the other is to apply WESS to develop a simulation application for certain requirements. More details about how to use WESS can be seen in [44, 45].

Consider the GFCCS simulation, we set the total logical running time to 1500 s for a simulation and run 150 rounds of Monte Carlo simulations. Figure 4 displays the 2D view. A benefit of the simulation display is that it assists us in a visual way to grasp the whole real-time situation of the battlefield, releasing us from thousands of lines of codes or texts. From the main window of this figure, we can see the overall situation of an engagement between two sides. The red side is a fighter which is equipped with radars and missiles, and the blue side contains an airfield, a warship, and also a fighter. At the simulation time 1160 s, the warship finds the enemy fighter and prepares to launch a surface to air missile. Below the main window we can see two charts, each of which contains couples of curves over time. Once an object is selected in the main window, the left one shows the corresponding elevation and velocity curves, while the right one shows curves for the yaw, roll, and pitch angles.

Table 4 presents the simulation data of a missile prototyped SM2New\_1 in terms of its key states such as Start, Boost, Guide, and Terminal. The surface to air defense missile, SM2New\_1, is prepared at 1172 s, launched and goes into boost phase by 3 s, then switches to guide phase at 1183 s, finally to terminal phase at 1201 s. Its velocity initially rises sharply then reaches a peak at the end of

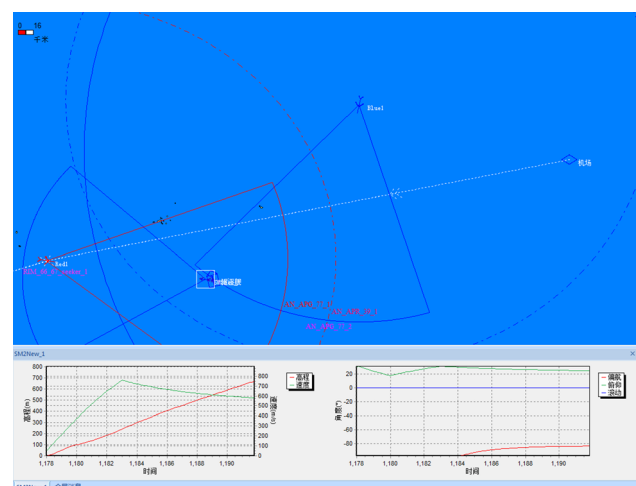


Fig. 4 The 2D display of GFCCS simulation

**Table 4** Simulation data of the SM2New\_1 missile

State	Start	Boost	Guide	Terminal
Time (s)	1172	1175	1183	1201
Velocity (m/s)	6.00	46.44	686.59	563.22
Elevation (m)	0.00	0.00	288.20	922.35

boost phase, afterwards, descends slowly. Unlike the velocity line, the elevation always rises smoothly, except approaching the target.

## 6 Conclusions

In general, a formal transformation requires that the models are specified in well-defined modeling languages and the transformation rules are defined with a well-defined transformation language. To ensure model continuity, the target model should contain as much as possible from the source model and the modeling relations should be preserved during the transformations. Consider the semantic gap between the different stages of a simulation modeling process, this paper defines a formal model transformation based on MDA and uses the GFCCS implementation as a motivating example through the overall transformation process. This process consists of GFCCS to P-DEVS and to JAVA transformations, accompanying with the initial DSML design based on EMF and the final source code generation. Before the transformation process, some criteria, i.e. correctness, completeness, uniqueness, and determinism, are listed to evaluate the model continuity during the transformation. The transformation results showed that model continuity is obtained when the transformations are conducted successfully according to the formal definition. Currently, there are little guidelines to instruct the explicit use of conceptual model in later stages of a simulation model development lifecycle [46–48]. A benefit of this study is that the formal definition of model transformation can be viewed as a referenced experience to guide other practices that have the needs of formal analysis. However, as a drawback some effort for further experiments and evaluations are required.

## References

- Bryant, B. R., Gray, J., Mernik, M., Clarke, P. J., France, R. B., & Karsal, G. (2011). Challenges and directions in formalizing the semantics of modeling languages. *Computer Science and Information Systems*, 8(2), 225–253. <https://doi.org/10.2298/CSIS110114012B>.
- Abdulah, M. S. (2006). *A UML profile for conceptual modeling of knowledge-based systems*. Dissertation, University of York.
- Langer, P., Wieland, K., Wimmer, M., & Cabot, J. (2011). From UML profiles to EMF profiles and beyond. In *Proceedings of the 49th international conference on objects, models, components, patterns*, Zurich, Switzerland (pp 52–67). [https://doi.org/10.1007/978-3-642-21952-8\\_6](https://doi.org/10.1007/978-3-642-21952-8_6).
- Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., et al. (2001). The generic modeling environment. In *Proceedings of the IEEE international workshop on intelligent signal processing (WISP)*, Budapest, Hungary (pp. 1–6).
- Getir, S., Challenger, M., & Kardas, G. (2014). The formal semantics of a domain-specific modeling language for semantic web enabled multi-agent systems. *International Journal of Cooperative Information Systems*, 23(3), 1–53. <https://doi.org/10.1142/s0218843014500051>.
- Meyers, B. (2016). *A multi-paradigm modeling approach to design and evolution of domain-specific modeling languages*. Dissertation, University of Antwerpen.
- Balci, O. (2012). A life cycle for modeling and simulation. *Simulation*, 88(7), 870–883. <https://doi.org/10.1177/0037549712438469>.
- Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2), 25–31. <https://doi.org/10.1109/MC.2006.58>.
- Sarjoughian, H. S. (2006). Model composability. In *Proceedings of the 38th WSC*, Monterey, CA (pp. 149–158). <https://doi.org/10.1145/1218112.1218144>.
- Feng, T. H., Lee, E. A., & Schruben, L. W. (2010). Ptera: An event-oriented model of computation for heterogeneous systems. In *Proceedings of the 10th ACM international conference on embedded software*, Scottsdale, AZ (pp. 219–228). <https://doi.org/10.1145/1879021.1879050>.
- Lee, E. A., & Sangiovanni-Vincentelli, A. L. (1998). A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12), 1217–1229. <https://doi.org/10.1109/43.736561>.
- Lee, E. A., & Messerschmitt, D. G. (2009). Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1), 24–35. <https://doi.org/10.1109/TC.1987.5009446>.
- Hardebolle, C., & Boulanger, F. (2009). Exploring multi-paradigm modeling techniques. *Simulation*, 85(11–12), 688–708. <https://doi.org/10.1177/0037549709105240>.
- Mosterman, P. J., & Vangheluwe, H. (2004). Computer automated multi-paradigm modeling: An introduction. *Simulation*, 80(9), 433–450. <https://doi.org/10.1177/0037549704050532>.
- Çetinkaya, D. (2013). *Model driven development of simulation models: Defining and transforming conceptual models into simulation models by using metamodels and model transformations*. Dissertation, Middle East Technical University.
- Hu, X., & Zeigler, B. P. (2005). Model continuity in the design of dynamic distributed real-time systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 35(6), 867–878. <https://doi.org/10.1109/TSMCA.2005.851283>.
- Strembeck, M., & Zdun, U. (2010). An approach for the systematic development of domain-specific languages. *Software Practice & Experience*, 39(15), 1253–1292. <https://doi.org/10.1002/spe.936>.
- Ehrig, H., & Ermel, C. (2008). Semantical correctness and completeness of model transformations using graph and rule transformation. In *Proceedings of the 4th international conference on graph transformations*, Leicester, UK (pp. 194–210). [https://doi.org/10.1007/978-3-540-87405-8\\_14](https://doi.org/10.1007/978-3-540-87405-8_14).
- Jiang, D., Huo, L., & Song, H. (2018). Rethinking behaviors and activities of base stations in mobile cellular networks based on

- big data analysis. *IEEE Transactions on Network Science and Engineering*, 1(1), 1–12.
20. Jiang, D., Wang, Y., Lv, Z., et al. (2019). Big data analysis-based network behavior insight of cellular networks for industry 4.0 applications. *IEEE Transactions on Industrial Informatics*. <https://doi.org/10.1109/TII.2019.2930226>.
  21. Mens, T., & Gorp, P. V. (2005). A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152, 125–142. <https://doi.org/10.1016/j.entcs.2005.10.021>.
  22. Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA explained: The model driven architecture<sup>TM</sup>: Practice and promise*. Boston: Addison-Wesley.
  23. Zeigler, B. P., Praehofer, H., & Kim, T. G. (2000). *Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems* (2nd ed.). San Diego, CA: Academic Press.
  24. Wang, F., Jiang, D., Wen, H., et al. (2019). Adaboost-based security level classification of mobile intelligent terminals. *The Journal of Supercomputing*, 75(11), 1–19. <https://doi.org/10.1007/s11227-019-02954-y>.
  25. Jiang, D., Huo, L., & Li, Y. (2018). Fine-granularity inference and estimations to network traffic for SDN. *PLoS ONE*, 13(5), 1–23. <https://doi.org/10.1371/journal.pone.0194302>.
  26. Szabo, C., & Teo, Y. M. (2007). On syntactic composability and model reuse. In *Proceedings of 1st Asia international conference on modeling and simulation* (pp. 230–237). <https://doi.org/10.1109/AMS.2007.74>.
  27. Estanol, M., Sancho, M. R., & Teniente, E. (2018). Ensuring the semantic correctness of a BAUML artifact-centric BPM. *Information and Software Technology*, 93, 147–162. <https://doi.org/10.1016/j.infsof.2017.09.003>.
  28. Jiang, D., Li, W., & Lv, H. (2017). An energy-efficient cooperative multicast routing in multi-hop wireless networks for smart medical applications. *Neurocomputing*, 220(2017), 160–169. <https://doi.org/10.1016/j.neucom.2016.07.056>.
  29. Jiang, D., Zhang, P., Lv, Z., & Song, H. (2016). Energy-efficient multi-constraint routing algorithm with load balancing for smart city applications. *IEEE Internet of Things Journal*, 3(6), 1437–1447. <https://doi.org/10.1109/JIOT.2016.2613111>.
  30. Atkinson, C., & Kuhne, T. (2003). Model-driven development: A metamodeling foundation. *IEEE Software*, 20(5), 36–41. <https://doi.org/10.1109/ms.2003.1231149>.
  31. Nordstrom, G., Sztipanovits, J., Karsai, G., & Ledeczi, A. (1999). Metamodeling—Rapid design and evolution of domain-specific modeling environments. In *Proceedings of IEEE conference on engineering of computer-based systems* (pp. 68–74). <https://doi.org/10.1109/ECBS.1999.755863>.
  32. Zhu, Z., Lei, Y. L., Zhu, Y. F., & Sarjoughian, H. S. (2017). Cognitive behaviors modeling using UML profile. *IEEE Access*, 5, 21694–21708. <https://doi.org/10.1109/ACCESS.2017.2760060>.
  33. De, L. J., Guerra, E., & Cuadrado, J. S. (2015). Model-driven engineering with domain-specific meta-modeling languages. *Software and System Model*, 14(1), 429–459. <https://doi.org/10.1007/s10270-013-0367-z>.
  34. Seo, K. M., Choi, C., Kim, T. G., & Kim, J. H. (2014). DEVS-based combat modeling for engagement-level simulation. *Simulation*, 90(7), 759–781. <https://doi.org/10.1177/0037549714532960>.
  35. Zhu, Z., Lei, Y. L., Sarjoughian, H. S., Li, X. B., & Zhu, Y. F. (2018). UML-based combat effectiveness simulation system modeling within MDE. *Journal of Systems Engineering and Electronics*, 29(6), 1180–1196. <https://doi.org/10.21629/JSEE.2018.06.07>.
  36. Zhu, Z., Lei, Y. L., Alshareef, A., Sarjoughian, H. S., & Zhu, Y. F. (2018). Domain specific metamodelling for deep semantic composability. *IEEE Access*, 6, 18276–18289. <https://doi.org/10.1109/access.2018.2822661>.
  37. Selic, B. (2007). A systematic approach to domain-specific language design using UML. In *Proceedings of 10th IEEE international symposium on object and component-oriented real-time distributed computing* (pp. 2–9). <https://doi.org/10.1109/ISORC.2007.10>.
  38. Warmer, J., & Kleppe, A. (1999). *The object constraint language—Precise modeling with UML*. Boston: Addison-Wesley.
  39. Huo, L., Jiang, D., Zhu, X., et al. (2019). An SDN-based fine-grained measurement and modeling approach to vehicular communication network traffic. *International Journal of Communication Systems*, 8(1), 1–12. <https://doi.org/10.1002/dac.4092>.
  40. Huo, L., & Jiang, D. (2019). Stackelberg game-based energy-efficient resource allocation for 5G cellular networks. *Telecommunication System*, 23(4), 1–11. <https://doi.org/10.1007/s11235-019-00564-w>.
  41. Wang, F., Jiang, D., & Qi, S. (2019). An adaptive routing algorithm for integrated information networks. *China Communications*, 7(1), 196–207.
  42. Jouault, F., Allilaire, F., Bezivin, J., & Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72(1–2), 31–39. <https://doi.org/10.1016/j.scico.2007.08.002>.
  43. Benouda, H., Essbai, R., Azizi, M., & Moussaoui, M. (2013). Modeling and code generation of android applications using Aceleo. *International Journal of Software Engineering and Its Applications*, 10(3), 83–94. <https://doi.org/10.14257/ijseia.2016.10.3.08>.
  44. Lei, Y. L., Zhu, Z., Li, Q., Yang, F., & Zhu, Y. F. (2017). WESS: A generic combat effectiveness simulation system. In *Proceedings of the 17th Asia simulation conference* (pp. 272–283). [https://doi.org/10.1007/978-981-10-6502-6\\_24](https://doi.org/10.1007/978-981-10-6502-6_24).
  45. Zhu, Z., Lei, Y. L., Zhu, N., & Zhu, Y. F. (2014). Domain specific metamodelling for deep semantic composability. *NUDT Transactions*, 36(5), 186–190. <https://doi.org/10.11887/j.cn.201405031>.
  46. Huo, L., Jiang, D., & Lv, Z. (2018). Soft frequency reuse-based optimization algorithm for energy efficiency of multi-cell networks. *Computers & Electrical Engineering*, 66(2), 316–331. <https://doi.org/10.1016/j.compeleceng.2017.09.009>.
  47. Jiang, D., Huo, L., Lv, Z., Song, H., & Qin, W. (2018). A joint multi-criteria utility-based network selection approach for vehicle-to-infrastructure networking. *IEEE Transactions on Intelligent Transportation Systems*, 19(10), 3305–3319. <https://doi.org/10.1109/TITS.2017.2778939>.
  48. Jiang, D., Wang, W., Shi, L., & Song, H. (2018). A compressive sensing-based approach to end-to-end network traffic reconstruction. *IEEE Transactions on Network Science and Engineering*, 5(3), 1–12. <https://doi.org/10.1109/TNSE.2018.2877597>.



**Zhi Zhu** was born in 1989. He is a Ph.D. and an assistant professor at National University of Defense Technology. He was also a visiting Ph.D student at Arizona State University. His research interests are language-driven development and system engineering.



**Qun Li** was born in 1971. He is a Ph.D. and a professor at National University of Defense Technology. He was also a visiting scholar at Nanyang Technological University. His researches are software engineering, simulation-based design, and agent-based simulation.



**Yonglin Lei** was born in 1978. He is a Ph.D. and a professor at National University of Defense Technology. He was also a visiting scholar at Arizona State University. His research interests are complex system modeling & simulation, model-driven engineering, and simulation composability.



**Yifan Zhu** was born in 1963. He is Ph.D. and a professor at National University of Defense Technology. He was also a visiting scholar at Virginia Polytechnic Institute and State University. His research interests are equipment system evaluation and demonstration, and agent-based modeling and simulation.