An executable modeling and analyzing approach to C4ISR architecture

HE Hongyue, ZHU Weixing*, LI Ruiyang, and DENG Qiaoyu
Institute of Command and Control Engineering, Army Engineering University of PLA, Nanjing 210007, China

Abstract: To analyze the behavioral model of the command, control, communication, computer, intelligence, surveillance, reconnaissance (C4ISR) architecture, we propose an executable modeling and analyzing approach to it. First, the meta concept model of the C4ISR architecture is introduced. According to the meta concept model, we construct the executable meta models of the C4ISR architecture by extending the meta models of fUML. Then, we define the concrete syntax and executable activity algebra (EAA) semantics for executable models. The semantics functions are introduced to translating the syntax description of executable models into the item of EAA. To support the execution of models, we propose the executable rules which are the structural operational semantics of EAA. Finally, an area air defense of the C4ISR system is used to illustrate the feasibility of the approach.

Keywords: command, control, communication, computer, intelligence, surveillance, reconnaissance (C4ISR) architecture, meta model, executable model, algebraic semantics.

DOI: 10.21629/JSEE.2020.01.12

1. Introduction

Command, control, communication, computer, intelligence, surveillance, reconnaissance (C4ISR) system is a military complex system. C4ISR architecture is a blueprint of system that bridges the gap between requirements and implements in the design of the C4ISR system. The quality of the C4ISR architecture can influence the quality of the C4ISR system. Thus, in the early development of the architecture, using modeling and simulation for analysis is particularly important [1].

Most architects design the C4ISR architecture according to Department of Defense Architecture Framework (DoDAF) and use UML to model the C4ISR architecture [2]. However, most behavioral models of the C4ISR architecture are not executable. To analyze the C4ISR architecture, these models need to be translated into some executable models. Petri Net is mainly used to describe these models [3–5]. Ni et al. [6] designed an approach which can map the productions of DoDAF into hierarchy coloured Petri Net (HCPN) to evaluate the behavior of the C4ISR architecture. These translations need the participation of an expert who masters Petri Net. ExtendSim is another method used to simulate models of the C4ISR architecture. Xiong et al. [7] used SysML to model the C4ISR architecture and map these models into the ExtendSim model. Ge et al. [8] translated the UML models which describe the production of the C4ISR architecture into the ExtendSim model. The structure of the ExtendSim model is complex, so it is difficult to analyze the C4ISR architecture.

Executable architecture [9] is another kind of method to analyze C4ISR architecture. The C4ISR Key Laboratory of National Defense Technology develops some theories and methods of executable architecture. Wang et al. [10] developed the C4ISR executable architecture based on service-oriented architecture (SOA). Ge et al. [11,12] made some research on the modeling and analysis of executable architecture using UML and ExtendSim. Milena Guessi et al. [13] investigated the main features that should be supported by architecture description language (ADL) for representing architects, and suggested some directions for improving the way the current ADL addresses architecture language. Judith et al. [14] provided an approach to represent an structural operational semantics (SOS) architecture based on the executable SysML model. These models of executable architecture must be translated into some executable modeling language, such as Object Petri Net or the ExtendSim model. Thus there is an argue about whether the executable architecture is needed. Hu et al. [15] proposed a generic modeling approach to executable architecture of system of systems (SoS) based on parallel discrete event system specification (P-DEVS). Cui et al. [16] proposed a domain specific language, which is interpreted through a combination of symbolic evaluation and generates executable evaluation functions, to support closed-form high-level architecture modeling.

Manuscript received May 14, 2018.
*Corresponding author.
Touraille et al. [17] integrated single platform tools for modeling, simulation, analysis and collaboration, and then developed SimStudio which is a modeling and simulation environment based on the discrete event systems specification (DEVS) formalism. Based on DEVS, Kara et al. [18] also proposed a simulation modeling architecture (SiMA). SiMA supports hierarchical and modular composition of reusable models. Combining DEVS and model-driven development (MDD), Cetinkaya et al. [19] developed an MDD framework for modeling and simulation (MDD4MS). MDD4MS helps the modeler to construct models faster and better. It provides the mechanism of model transformations and (semi)automatic code generation.

UML and SysML are accepted as ADL by architects [20]. Oquendo et al. [21] presented the executable viewpoint based on SysML profile for modeling the architecture of software-intensive systems. Most models of the C4ISR architecture can be modeled by UML. Object Management Group proposed the foundational subset for executable UML models (fUML) to enable UML models execution [22,23]. Based on the above research, we propose an executable modeling and analyzing approach to C4ISR architecture using fUML. First, we construct the C4ISR architecture meta concept model according to the DoDAF meta-model (DM2) [2]. Then we extend the meta models of fUML to define the executable meta model of the C4ISR architecture. To describe the execution semantics of C4ISR architecture executable models, we introduce the concrete syntax and algebra semantics. Finally, we define the executing rules for model execution and use a case study to demonstrate the availability of the theory.

2. Executable meta model of C4ISR architecture

Most architects design the C4ISR architecture according to DoDAF. To make these data consistently used by different architects, DoDAF2.0 proposed the DM2 which is a meta model of the Department of Defense (DoD) architecture framework. DM2 makes architects easily understand these data of the C4ISR architecture, and these data can be exchanged consistently between architects. These data must be described in some models by an ADL, but the DM2 is not an ADL.

According to the requirements of analyzing the C4ISR architecture, we extract some key classes and relations from the DM2 and use them to construct the meta concept model (MCM) of the C4ISR architecture, as shown in Fig. 1. According to these classes of the MCM, the relevant executable meta models of the C4ISR architecture are constructed. These relations of the MCM can guide the simulation process of C4ISR architecture executable models.

- Goal
  - Committed_to
  - Realize
  - Require
  - Have
  - Perform
  - Produce
  - Transition
  - Produce
  - Triggered by
- Capability
  - Featured_by
  - Require
  - Have
- System
  - Contact_with
- Activity
  - Node
  - Information

Fig. 1 MCM of C4ISR architecture

UML is widely used as an ADL, and fUML makes UML behavioral models executable. Therefore, fUML can be used to model the executable models of the C4ISR architecture. Because the model of the C4ISR architecture is a special domain model, fUML is not well applicable. Thus, we extend the meta models of fUML to construct the executable meta model of the C4ISR architecture. We choose goal, capability, system, activity and information of the MCM to extend the class meta model and activity meta model of fUML. Then, we construct executable goal meta model, executable capability meta model, executable system meta model, executable activity meta model and executable information meta model. We use these meta models to construct relevant executable models of the C4ISR architecture. Due to the limitation of paper space, we only introduce the executable capability meta model and exe-
Executable activity meta model, as shown in Fig. 2, is extended from the class meta model of fUML. The property of capability can be divided into some measures. A father capability can be decomposed into some son capabilities. A capability clings to some performers and needs some activities to realize it.

Executable activity meta model, as shown in Fig. 3, is extended from the action meta model of fUML. Activities can be classified into different groups according to the performers. The transition connects the activities which have an executing sequence relationship. Some activities have input or output which can be used to transfer data object.

Fig. 2 Executable meta model of capability

Fig. 3 Executable meta model of activity
### 3. Formalization of executable model

Because the activity model describes the behavior of the C4ISR system, the executable analysis of the C4ISR architecture is realized by the execution of the activity model. Other executable models supply relevant information during the execution process of the activity model. Thus, we mainly introduce the formalization of the activity model in this section. To support the execution of the activity model, we define the concrete syntax and algebra semantics for the activity model.

#### 3.1 Concrete syntax

In the activity model, system is the actor of activity. Activities belong to different swimlanes according to their actors. The ControlNode and transition control the execution sequence of activities. The OutPut and InPut supply the information during the execution of activities. Based on the above introduction, extended Backus-Naur form (EBNF) is used to describe the concrete syntax of the activity model as follows.

<activity model> ::= activity model <modelid> <actor list> <control list> end activity model
<actor list> ::= <actor> <actor list> | <>
<actor> ::= actor <actor name> <activity list> end actor
<activity list> ::= <activity> <activity list> | <>
<activity> ::= <input> | <output> | <action>
<input> ::= in <messageid> from <address>
<output> ::= out <messageid> to <address>
<address> ::= <actor name>
<action> ::= action <action name> <precondition> <guardcondition> <postcondition> end action
<precondition> ::= pre <condition>
<guardcondition> ::= guard <condition>
<postcondition> ::= post <condition>
<control list> ::= <control> <control list> | <>
<control> ::= fork <source list> <target list> end fork
<join> ::= join <source list> <target list> end join
<decision> ::= decision <source list> <condition>
<target> <target> end decision
<source list> ::= <source> <source list>
<source list> ::= <target> <source list>
<source> ::= source <actor name> <action name>
end source
<target> ::= target <actor name> <action name> end target

The black identifiers are terminal symbol. Other identifiers are nonterminal symbol, such as modelid, messageid, actor name, action name and condition. The map between the activity model and a concrete syntax is straightforward, so we omit its introduction.

#### 3.2 Algebra semantics

In the execution of the activity model, objects are the main participators. The execution of the activity model is realized by the interactions between objects or the actions of objects. The object is an instance of swimlane in the activity model. It can execute one or more actions. Action is executed by the object. An action may be executed by different objects, which results in different effects, so the action must cling to one object. The behavior of the C4ISR system comprises these actions of objects. In the activity model, the behavior of the C4ISR system can be abstracted as a process, and every process contains one or more couples of actions and objects. Based on the above analysis, we define executable activity algebras (EAA) referencing performance evaluation process algebras (PEPA). EAA can be used to describe the process of the activity model.

**Definition 1** $O$ is the set of objects which describe the swimlane of the activity model, $o \in O$. $A$ is the set of actions of the activity model, $a \in A$. $P$ denotes the process. The syntax of EAA is defined using the Backus-Naur-Form (BNF) notation:

$$P ::= 0 | (o, a). P | P_1 + P_2 | P_1 + . P_2 | [x = y]. P | P_1 :: P_2 | P_1 || P_2$$

where $0$ (Empty) is the empty process; $(o, a). P$ (Prefix) means when $o$ has executed $a$, $P$ is active. $P_1 + P_2$ (Sequence) means when $P_1$ has been executed, $P_2$ is active. $P_1 + . P_2$ (Choice) means if $c$ is true, $P_1$ will be active; otherwise, $P_2$ will be active. $[x = y]. P$ (Condition) mean if $x = y$ is true, $P$ will be active; otherwise, $P$ is waiting. $P_1 || P_2$ (Fork) means $P_1$ and $P_2$ will be executed synchronously. $P_1 || P_2$ (Join) means $P_1$ and $P_2$ are executed synchronously. When they are all ended, $P$ will be active.

EAA supplies an algebras semantics domain for the activity model. The algebras semantics of the activity model is an instance of EAA. We need to translate the syntax description of the activity model into the item of EAA. First, we introduce two action definitions. If $<\text{string}>$ is the non-terminal symbol in the syntax description of the activity model, $H(<\text{string}>)$ expresses the identifier which represents $<\text{string}>$.

**Definition 2** The action definition for input message is

$$in(o, m), o \in H(<\text{actor name}>) ,$$

$$m \in H(<\text{messageid}>) .$$

**Definition 3** The action definition for output message is

$$out(o, m), o \in H(<\text{actor name}>) ,$$
Then, we introduce some semantics function definitions for the main terminal syntax of the concrete syntax of the activity model.

**Definition 4** The semantics function definition for the symbol `<input>` is

\[ S(<\text{input}>)=in(H(<\text{actor name}>)), \]
\[ H(<\text{messageid}>)). \]

**Definition 5** The semantics function definition for the symbol `<output>` is

\[ S(<\text{output}>)=out(H(<\text{actor name}>)), \]
\[ H(<\text{messageid}>)). \]

**Definition 6** The semantics function definition for the symbol `<action>` is

\[ S(<\text{action}>)=(H(<\text{actor name}>)), \]
\[ H(<\text{action name}>)). \]

**Definition 7** The semantics function definition for the symbol `<fork>` is

\[ S(<\text{fork}>)=S(<\text{source}>);S(<\text{target list}>)= \]
\[ S(<\text{source}>);(S(<\text{target list}>)||S(<\text{target list}>))). \]

**Definition 8** The semantics function definition for the symbol `<join>` is

\[ S(<\text{join}>)=S(<\text{source list}>);S(<\text{target}>)= \]
\[ S(<\text{source list}>)||S(<\text{target list}>)). \]

**Definition 9** The semantics function definition for the symbol `<decision>` is

\[ S(<\text{decision}>)=S(<\text{source}>);S(<\text{target list}>)> \]
\[ +H(<\text{condition}>))S(<\text{target list}>)). \]

The definitions of \( S(<\text{source}>) \) and \( S(<\text{target}>) \) are as follows:

**Definition 10** The semantics function definition for the symbol `<source>` is

\[ S(<\text{source}>)=(H(<\text{actor name}>)), \]
\[ H(<\text{action name}>)). \]

**Definition 11** The semantics function definition for the symbol `<target>` is

\[ S(<\text{target}>)=(H(<\text{actor name}>)), \]
\[ H(<\text{action name}>)). \]

Using these semantics functions, we can translate the concrete syntax of the activity model into the item of EAA.

### 4. Executable rules

Currently, SOS is widely applied in the process algebra. SOS is a set of derivation rules to describe the behavioral evolution of process algebra. A process term \( P \) can evolve, subject to the change of the active label \((o, a)\), into a different term which is called the successor of \( P \) and defined as \( \text{Succ}(P) = \{P'|\exists(o, a) : P \xrightarrow{(o,a)} P'\} \)

If the process term \( P \) evolves into another term \( P' \), there must be a sequence of active labels \((o, a) = (o_1, a_1)(o_2, a_2)\cdots(o_n, a_n) (n > 0) \) which results in \( P \xrightarrow{(o_1,a_1)} P_1 \xrightarrow{(o_2,a_2)} \cdots \xrightarrow{(o_n,a_n)} P' \), and \((o, a)\) is the trace of the term \( P \) evolved into the term \( P' \). The trace is used to describe the execution of the activity model. To support the evolution of the process item in EAA, we define the derivation rules for any kind of syntax of EAA. In these rules, \( P \) denotes any process, \( P' \) is the successor of \( P \); \( P_1 \) and \( P_2 \) denotes some process.

#### 4.1 Prefix rule

- **Action-1**
  
  ![Diagram](Image)

  Rule Action-1 denotes that when the process \((o, a)\) is active, if \( o \) has executed \( a \), process 0 will be active. Rule Action-2 denotes that when the process \((o, a)\) is active, if \( o \) has executed \( a \), process \( P \) will be active. \( \square \) denotes that precondition of the rules is always true. The Prefix rule is used when the Prefix process item is evolved.

#### 4.2 Sequence rule

- **Sequence**
  
  ![Diagram](Image)

  Rule Sequence denotes that if process \( P_1 \) activates process \( P'_1 \) when \( o \) has executed \( a \), process \( P_1 \); \( P \) activates process \( P'_1 \); \( P \) when \( o \) has executed \( a \). The Sequence rule is used when the Sequence process item is evolved.

#### 4.3 Choice rule

- **Choice-1**
  
  ![Diagram](Image)

  Rule Choice-1 denotes that if condition \( c \) is true, process \( P_1 \); \( c \) \( P \) activates process \( P_1 \). Rule Choice-2 denotes that if condition \( c \) is true, process \( P_1 \); \( c \) \( P \) activates process \( P_2 \).
The Choice rule is used when the Choice process item is evolved.

### 4.4 Condition rule

\[ \text{Condition: } \quad \Box \quad [x = y] \rightarrow P \]

Rule Condition denotes that if condition \([x = y]\) is true, process \(P\) activates process \(P\). \(\Box\) denotes that the precondition of the rules is always true. The Condition rule is used when the Condition process item is evolved.

### 4.5 Fork rule

- **Rule Fork-1**
  \[ 0 || P_2 \rightarrow P_2 \]

- **Rule Fork-2**
  \[ P_1 || 0 \rightarrow P_1 \]

- **Rule Fork-3**
  \[ P_1 || P_2 \rightarrow (a,a) P'_1 \]

- **Rule Fork-4**
  \[ P_2 \rightarrow (a,a) P'_2 \]

Rule Fork-1 denotes that process \(0||P_2\) activates process \(P_2\). Rule Fork-2 denotes that process \(P_1||0\) activates process \(P_1\). Rule Fork-3 denotes that if process \(P_1\) activates process \(P'_1\) when \(o\) has executed \(a\), and process \(P_1||P_2\) activates process \(P'_1||P_2\) when \(o\) has executed \(a\). Rule Fork-4 denotes that if process \(P_2\) activates process \(P'_2\) when \(o\) has executed \(a\), process \(P_1||P_2\) activates process \(P_1||P'_2\) when \(o\) has executed \(a\). The Fork rule is used when the Fork process item is evolved.

### 4.6 Join rule

- **Rule Join-1**
  \[ 0 || P_2 \rightarrow P_2; P \]

- **Rule Join-2**
  \[ P_1 || P_2 \rightarrow P_1; P \]

- **Rule Join-3**
  \[ P_1 \rightarrow (a,a) \rightarrow P'_1 \]

- **Rule Join-4**
  \[ P_2 \rightarrow (a,a) \rightarrow P'_2 \]

Rule Join-1 denotes that process \(0||P_2\) activates process \(P_2; P\). Rule Join-2 denotes that process \(P_1||P_2\) activates process \(P_1; P\). Rule Join-3 denotes that if process \(P_1\) activates process \(P'_1\) when \(o\) has executed \(a\), and process \(P_1||P_2\) activates process \(P'_1||P_2\) when \(o\) has executed \(a\). Rule Join-4 denotes that if process \(P_2\) activates process \(P'_2\) when \(o\) has executed \(a\), and process \(P_1||P_2\) activates process \(P_1||P'_2\) when \(o\) has executed \(a\). The Join rule is used when the Join process item is evolved.

During the evolution process of EAA, a set of labels are needed. There are four kinds of labels in the activity model. The set of input message labels is described as \(A_i = \{S(<\text{input}>)\}\); the set of output message label is described as \(A_o = \{S(<\text{output}>)\}\); the set of action label is described as \(A_a = \{S(<\text{action}>)\}\); the set of condition label is described as \(A_c = \{H(<\text{condition}>)\}\). The set of labels of the activity model is described as \(A = A_i \cup A_o \cup A_a \cup A_c\). During the execution process of the activity model, the set of used labels is the subset of \(A\). We can choose different sets of labels to execute the activity model according to different analysis problems.

### 5. Case study

In this section, we use the threat air defense system (TADS) which is a kind of C4ISR system to demonstrate the availability of this theory. The structure and activity models of TADS are built using the meta models. Because of the limited space, we only describe the activity model and algebra semantics of the activity model.

In TADS, there are constituent systems: radar battalion (RB), threat command center (TCC), missile battalion (MB) and anti-air gun battalion (AGB). When an unknown light plane (UFP) flies in the area of TADS, it triggers the behaviors of constituent systems of TADS. The process of UFP interception can be described as follows. Once the UFP has flown in the area, the RB detects the target. The information of UFP will be sent in time to the TCC which will make a threat assessment on it and decide who shall undertake the task of interception. After the undertaker (MB or AGB) has accepted the interception order and has received the UFP information, it will aim at and shoot on the UFP, and the UFP modifies its state to indicate the degree of destroy. Then RB collects the interception effect and sends it to TCC. TCC assesses the interception. If the interception fails, TCC sends an order to MB or AGB to arrange another interception. The activity model is illustrated in Fig. 4.

According to the syntax of EAA and the mapping rules between activity model notations and process terms, the activity model of TADS can be expressed using the following process terms:

\[ P_1 = (\text{UFP, } \text{FlyIn}).P_2 \]
\[ P_2 = P_3||P_4 \]
\[ P_3 = (\text{UFP, } \text{ModifyState}) \]
\[ P_4 = (\text{RB, } \text{DetectTarget}).P_5 \]
\[ P_5 = (\text{TCC, } \text{AssessThreat}).P_6 \]
Fig. 4 The activity model of TADS

\[ P_6 = P_7 + \text{OrderMissile} \]
\[ P_7 = (\text{MB}, \text{Intercept}) \]
\[ P_8 = (\text{AGB}, \text{Intercept}) \]
\[ P_9 = P_3 | P_{10} \]
\[ P_{10} = (\text{RB}, \text{CollectDataOfEffect}) \]
\[ P_{11} = (\text{TCC}, \text{Assess}) \]
\[ P_{12} = P_6 + \text{Fail} 0 \]

0 is the empty process, and 0 \(\rightarrow\) \(P_1\) is always true. The label transition set of this activity model is

\[ T = (0, P, \text{Lab}, \rightarrow): \]
\[ P = \{ P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, P_{12} \}. \]
\[ \text{Lab} = \{ (\text{UFP}, \text{FlyIn}), (\text{UFP}, \text{ModifyState}), (\text{RB}, \text{DetectTarget}), (\text{TCC}, \text{AssessThreat}), (\text{MB}, \text{Intercept}), (\text{RB}, \text{CollectDataOfEffect}), (\text{AGB}, \text{Intercept}), (\text{TCC}, \text{Assess}), \text{OrderMissile}, \text{OrderAnti-airGun}, \text{Fail}, \text{Success} \}. \]
\[ \rightarrow = \{ (P_1, (\text{UFP}, \text{FlyIn}), P_2), (P_4, (\text{RB}, \text{DetectTarget}), P_5), (P_5, (\text{TCC}, \text{AssessThreat}), P_6), (P_6, \text{OrderMissile}, P_7), (P_8, \text{OrderAnti-airGun}, P_8), (P_{10}, (\text{RB}, \text{CollectDataOfEffect}), P_{11}), (P_{11}, (\text{TCC}, \text{Assess}), P_{12}), (P_{12}, \text{Fail}, P_6), (P_{12}, \text{Success}, 0) \}. \]

When the activity model is executed, a serial of labels in the lab are given. We can use these labels to evolve these processes described in the form of the activity model, and then to analyze the behavior of TADS. For example, there are three label lists:

\[ L_1 = ((\text{UFP}, \text{FlyIn}), (\text{RB}, \text{DetectTarget}), (\text{TCC}, \text{AssessThreat}), \text{OrderMissile}, (\text{MB}, \text{Intercept}), (\text{UFP}, \text{ModifyState}), (\text{RB}, \text{CollectDataOfEffect}), (\text{TCC}, \text{Assess}), \text{Success}), (\text{UFP}, \text{ModifyState}), (\text{RB}, \text{CollectDataOfEffect}), (\text{TCC}, \text{Assess}), \text{Fail}). \]

Using \(L_1\), we can observe a process trace \((P_1, P_2, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, P_{12}, 0)\). The process
trace is ended by 0 which denotes the end of the activity model. During the activity model simulation process, we can observe that the UFP is intercepted by MB. However, using $L_2$, we observe a process trace $(P_1, P_2, P_3, P_4, P_6)$. The process trace is ended by $P_6$. During this simulation, we can observe that the UFP is not intercepted by AGB, so that TCC must make a new order. Using $L_3$, we observe a process trace $(P_1, P_2, P_4, P_5, P_6)$. During this simulation, we can observe that TCC has made a threat assessment on UFP, but TCC does not make an interception order.

Using different labels to simulate the activity model, we can observe and analyze the different behaviors of TADS.

6. Conclusions

Nowadays, the models of the C4ISR architecture are usually constructed by IDEF or UML, but these models are not executable. There are some inconveniences during the evaluation of the C4ISR architecture. In this paper, we propose an executable modeling and analyzing approach to the C4ISR architecture. The executable models of the C4ISR architecture are executed to analyze it. The contribution of this work is described as follows. First, the executable meta models of the C4ISR architecture are constructed by extending the meta model of fUML. Using these meta models, we can construct executable models of the C4ISR architecture. Second, the concrete syntax and algebra semantics of executable models are defined, and the semantics functions are introduced to translating the syntax description of executable models into the items of EAA. Third, to support the execution of models, the executable rules, the SOS of EAA, are proposed. Using these rules, the items which denote the behavior of executable models can be evolved to express the process of behavior.

References

[1] ENDER T, LEURCK R F, WEAVER B, et al. System-of-systems analysis of ballistic missile defense architecture effectiveness through surrogate modeling and simulation. IEEE Systems Journal, 2010, 4(2): 156 – 166.
[2] DoD Architecture Framework Working Group. DoD architecture framework 2.0. Washington: The United States Department of Defense, 2009.
[3] LI G, HAN L M, ZHU Y L. Trusted modeling and analysis of complex software systems based on spatial-temporal Petri nets. Journal of System Simulation, 2016, 28(8): 1740 – 1747. (in Chinese)
[4] NIKOLAIDOU M, KAPOS G D, TSADIMAS A, et al. Simulating SysML models: overview and challenges. Proc. of the System of Systems Engineering Conference, 2015: 328 – 333.
[5] MEZIANI L, BOUABANA-TEBIBEL T, BOUZAR-BENLABIOD L. From Petri nets to UML model: a new transformation approach. Proc. of the IEEE International Conference on Information Reuse and Integration, 2018: 503 – 510.
[6] NI F, WANG M Z, ZHOU F, et al. Approach to executable architecture modeling from DODAF to HCPN. Systems Engineering and Electronics, 2010, 32(5): 959 – 965. (in Chinese)
[7] XIONG J, CHEN Y W, WANG D. Executable model of weapon architecture. Systems Engineering and Electronics, 2010, 32(5): 966 – 970. (in Chinese)
[8] GE B, HIPEL K W, YANG K, et al. A data-centric capability-focused approach for system-of-systems architecture modeling and analysis. Systems Engineering, 2013, 16(3): 363 – 377.
[9] OQUENDO F, LEITE J, BATISTA T. Specifying executable software architectures. OQUENDO F, LEITE J, BATISTA T. Ed. Software architecture in action. Basel, Switzerland: Springer International Publishing, 2016.
[10] WANG L, LUO X S, LUO A M. Research on C4ISR executable architecture based on SOA. Fire Control & Command Control, 2012, 37(1): 52 – 56.
[11] GE B F, REN C S, ZHAO Q S, et al. Executable architecture modeling and analysis for system-of-systems. Systems Engineering – Theory & Practice, 2011, 31(11): 2191 – 2201. (in Chinese)
[12] GE B, HIPEL K W, YANG K, et al. A novel executable modeling approach for system-of-systems architecture. IEEE Systems Journal, 2017, 8(1): 4 – 13.
[13] MILENA G, EVERTON C, LUCAS B R O. Characterizing architecture description languages for software-intensive systems-of-systems. Proc. of the IEEE/ACM 3rd International Workshop on Software Engineering for Systems-of-Systems, 2015: 12 – 18.
[14] DAHMANN J, MARKINA-KHUSID A, DOREN A, et al. SysML executable systems of system architecture definition: a working example. Proc. of the Annual IEEE International Systems Conference, 2017: 1 – 6.
[15] HU J P, HUANG L P. Modeling and simulation of executable architecture based on P-DEVS. Journal of System Simulation, 2016, 28(2): 283 – 291. (in Chinese)
[16] CUI W, DING Y, DANGWAL D, et al. Charm: a language for closed-form high-level architecture modeling. Proc. of the 45th ACM/IEEE Annual International Symposium on Computer Architecture, 2018: 152 – 165.
[17] TOURAILLE L. A model-driven software environment for modeling, simulation and analysis of complex systems. Proc. of the Symposium on Theory of Modeling & Simulation Devs Integrative M & S Symposium, 2011: 229 – 237.
[18] RHoads G, KAGAN A. Simulation modeling architecture (SiMA), a DEVS based modeling and simulation framework. Proc. of the Summer Computer Simulation Conference, 2009: 315 – 321.
[19] CETINKAYA D, VERBRAECK A, SECK M D. MDDMS: a model driven development framework for modeling and simulation. Proc. of the 45th Summer Computer Simulation Conference, 2010: 113 – 121.
[20] PandeY R K. Architectural description languages (ADLs) vs UML: a review. ACM SIGSOFT Software Engineering Notes, 2010, 35(3): 1 – 5.
[21] OQUENDO F, LEITE J, BATISTA T. Executing software architecture descriptions with SysADL. Proc. of the European Conference on Software Architecture, 2016: 187 – 194.
[22] Object Management Group. UML 2.0 superstructure specification. http://www.omg.org/spec/UML/2.0/superstructure/PDF/.
[23] Object Management Group. Semantics of a foundational subset for executable UML models (fUML) 1.0. http://www.omg.org/spec/fUML/1.0/PDF/.
Biographies

**HE Hongyue** was born in 1985. He received his M.S. degree in 2007 and Ph.D. degree in 2014 from PLA University of Science and Technology. Now, he is a lecturer of Army Engineering University of PLA. His research interest is system of systems engineering, focusing on specification.
E-mail: hehy2008@sina.com

**ZHU Weixing** was born in 1978. He received his M.S. degree in 2006 and Ph.D. degree in 2012 from PLA University of Science and Technology. Now, he is an associate professor of Army Engineering University of PLA. His research interest is requirements engineering, focusing on specification and management.
E-mail: zwxplau@126.com

**LI Ruiyang** was born in 1992. He is a Ph.D. student in Institute of Command and Control Engineering, Army Engineering University of PLA. After graduating in 2014, he was admitted to PLA University of Science and Technology, and his major was system analysis and integration. Then he continued to pursue a doctorate in software engineering. Now, his major research interests are requirement engineering and model and method of system of systems planning and management.
E-mail: 15601591998@163.com

**DENG Qiaoyu** was born in 1996. She is a postgraduate student in Institute of Command and Control Engineering, Army Engineering University of PLA. After graduating in 2017, she was admitted to PLA University of Science and Technology, with the major network engineering. Now, her major research interests are requirement engineering, command and control theory and practice.
E-mail: coco515256301@qq.com