19th ICCRTS

Executable Architecture Modeling and Simulation Based on fUML

Topic 5: Modeling and Simulation
Authors: WANG Zhixue, HE Hongyue, WANG Qinglong
POC: WANG Zhixue
PLA University of Science and Technology
Nanjing, China
E-mail: wzxcx@163.com
Authors' affiliations:
PLA University of Science and Technology

Abstract:

The DoD Architecture Framework is now popularly used for describing overall requirements and architectural design of the system-of-systems (SoS). However, it is very hard to validate and verify the architecture products, as most of them are modeled with informal constructs. The paper proposes an approach of executable architecture modeling and simulation by introducing formal UML specification. Firstly, definitions of executable architecture are provided, upon which both structural and behavioral meta-models of SoS architecture are built by extending fUML meta models. Then, a simulation language is defined based on Process Algebras, and the semantics of emergent behavior of SoS is discussed. The executable models of a SoS architecture are therefore constructed through: (1) modeling the structures and behaviors of SoS, (2) translating the models into process terms and (3) specifying simulation rules upon the process terms. Since Process Algebras based executable tools are available over relevant research institutions, it is not difficult to build a simulation execution environment. Finally, a case study is used to illustrate the feasibility of the approach.

Keyword: executable architecture; meta model; executable model; algebraic semantics

1. Introduction

Architecture describes the components of system, connectors between these components, rules which are used to guide the design and evolvement of system. It is a blueprint of system and bridges the gap between requirements and implements in the design of system-of-systems (SoS). The quality of architecture can influence the schedule of SoS designment and the quality of SoS. So in the early development of architecture, using modeling and simulation to analyze and demonstrate is particularly important [1].

The DoD Architecture Framework is now popularly used for describing overall requirements and architectural design of SoS[2]. However, it is very hard to validate and verify the architecture products, as most of them are modeled with informal constructs. These models cannot be executed before being translated into an formal language, such as Petri Net. And, the

translation might be a pain of system engineers due to their weakness in formal languages. They would hope that the architectual models they built be automatically translated into executable ones.

UML is accepted as an Architectural Description Language by architects, and it has become a standard notation to document the architecture of system [3]. To describe the dynamic behaviors of system, UML2.0 enriches the behavioral semantics by adding the detailed action semantics [4]. But these UML models are not executable. Object Management Group proposes the fUML to enable UML models execution [5]. Accordingly, we propose an approach of executable architecture modeling and simulation by introducing fUML specification. Firstly, we construct the meta-models for structural and behavioral models of SoS architecture by extending fUML. To enable execution, we introduce the concrete syntax and algebra semantics for the SoS architecture models, and the execution processes can be interpreted as run of algebra derivation. Finally, we use the process trace to exhibit the emergent behavior of SoS.

The rest of the paper is organized as follows. Section 2 describes the related research. Section 3 introduces the executable architecture. Section 4 describes the formalization of behavioral model. Section 5 proposes the behavioral analysis for SoS architecture. Section 6 uses a case study to demonstrate the availability of the theory.

2. Related Work

Architecture verification plays an important role in design of SoS. The current verification techniques based on model execution can be classified into two categories: executable architecture modeling and model transformation. The former requires that the architecture be modeled as executable models or executable rules be defined. The latter implies translation of the architecture models into executable ones, such as Petri Net, ExtendSim model, DEVS.

The MITRE Company proposes a method for executable architecture modeling. The business process models, communication models and campaign simulation environment are connected by the Runtime Infrastructure of High Level Architecture[6]. Jiang et al. [7] define the concepts and executable rules for executable architecture and hence make some products of DoD architecture executable. Wang et al. [8] develop an executable architecture based on SOA. But these executable models pay much attention to the details of systems and therefore need great efforts. So, there is an argue about whether the executable architecture is needed.

As most behavioral models of architecture are not executable, they need to be translated into some executable ones. Petri Net is a popular executable modeling language for SoS behaviors. Wang et al.[9] use SysML sequence diagram to model the behaviors and translate the models into Colored Petri Nets (CPN). Staines T.S [10] incorporate fundamental CPN concepts into UML activity models such that the behavioral models are therefore formalized with executable sementics. Jiang et al. [11] use Object Petri Nets to formalize the IDEF3 models which describe system processes. Ge et al[12]. translate the UML models of architecture products into ExtendSim model. Touraille et al.[13] integrate single platform tools for modeling, simulation, analysis and collaboration, and then develop SimStudio which is a modeling and simulation environment based on the Discrete Event Systems Specification (DEVS) formalism. Based on

DEVS, Kara et al.[14] also propose a Simulation Modeling Architecture (SiMA). SiMA supports hierarchical and modular composition of reusable models. Combining DEVS and MDD, Cetinkaya et al.[15] develop a MDD framework for modeling and simulation (MDD4MS).COMPASS (Comprehensive Modeling for Advanced Systems of Systems) project develops a formal language (COMPASS Modeling Language, CML) for modeling and analyzing SoS[16]. CML is based on VDM, CSP and Circus. The process algebraic combinators are used to describe the behavior of SoS. The analysis techniques and prototype tools for UML are still in the development process.

But a few of the above approaches take into consideration the emergence behavior of SoS. Our approach translates the behavoral model into the process items of Process Algebras, and use the process trace to exhibit the emergence behavior of SoS.

3. Executable Architecture

The executable architecture is made up of three parts: the executable model, the execution mechanism and the execution process. The executable model, the fundation for the executable architecture, comprises both static and action models. The mechansim describes the execution principle of dynamic models. The execution process describes the process of model execution.

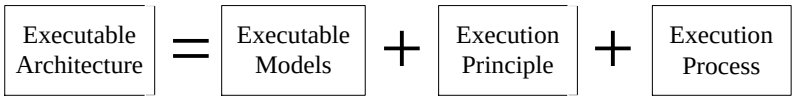

Fig. 1. The Executable Architecture

3.1 Meta Model of SoS Structure

The meta model of SoS structure extends the Class Model of. The system engineers use structure models to describe the structure of SoS, environment object and windows of simulator.

Structure of SoS

System is a core concept of the structure models. These systems and their relationships make up of the structure of SoS. To simulate the behaviors of SoS, the relationships can be abstracted as data flow. Systems are connected with each other through data flow. The output data of one system is the input data of another. A couple of output and input data makes up of a data flow. Systems use ports as interface to send and receive data. The data is information produced by activities. The activities are performed by systems which access data through their port.

Environment Object

There are two kinds of environment object. One is Data, the other is Event. Data can either be produced in SoS simulation or be given (input) by engineers. They can be input from or output to the windows of simulator. Event can take place inside or outside of SoS. It is used to trigger activity and change the execution process of activity. Time is a special event which is automatically triggered at a time. It is defined by the simulator of SoS. The simulator can monitor the changes of environment object according to the time. Time can also trigger activity and change the execution process.

Windows of Simulator

Windows of Simulator are the interfaces with which engineers can observe the changes of the environment and control the execution process. The simulator privides different windows for data, event and time. During the simulation, engineers play the role of Organizations. They use these windows to interact with SoS. They send inputs through the windows to simulator to control the execution process and receive the output from the windows to observe the SoS behaviors and their effects.

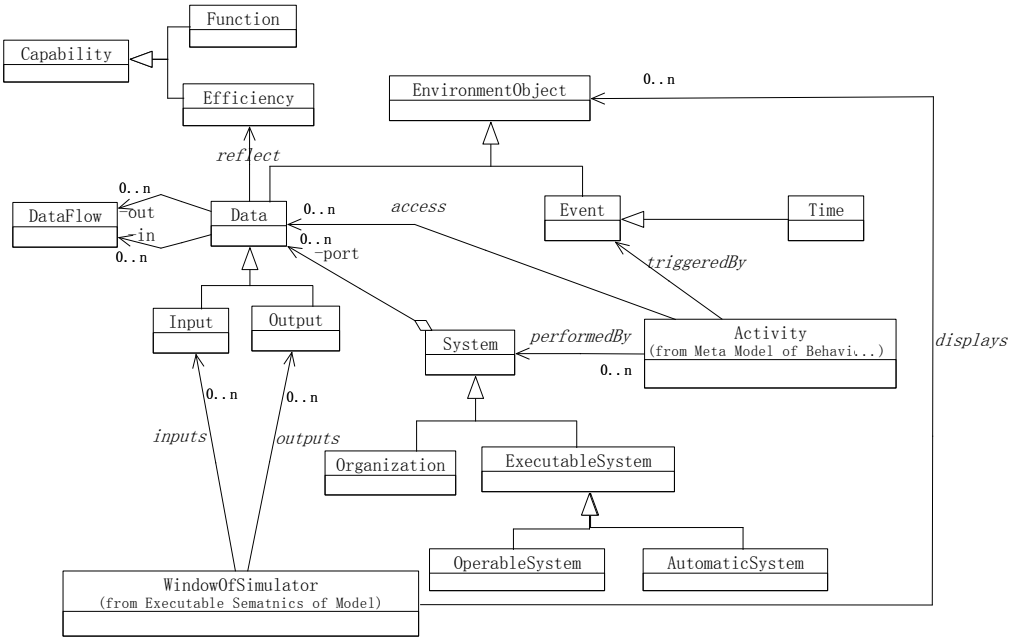

Fig. 2. The meta model of SoS structure

3.2 Meta Model of SoS Behaviors

We choose activity diagram to describe the behavior of SoS, because the simulation of SoS behaviors is processed as sequences of activities. The meta model of behaviors extends the meta model of Activity Diagram of fUML and is shown in Fig. 3.

The SoS behaviors are modeled with notations, such as Activity, NoActionActivity, Transition, GuardCondition, Action, Swimlane, Start and End. Action is the fundamental unit. An activity encapsulates a number of actions and is the execution body of the containing actions. NoActionActivity is a kind of control activity containing no action. It can be classified into Decision, Fork and Join. The notations are slightly different from fUML in followings.

(1) Action is the only executable unit. It can be either SysDefAction or UserDefAction. SysDefAction is the executable units pre-defined by simulator, and UserDefAction is the user-defined executable units programmed by engineers.

(2) Swimlane is the bridge between Activity and Data. It represents constituent systems that perform activities and provide the data ports through which activities access and modify the data as Input or Output.

The semantics of other notations is the same as fUML, except following constraints that guarantee the model robustness.

(1) There should be a sole pair of Start and End in the behavioral model.

(2) No any notation object except Activity contains an event or action.
(3) Assume the transition that flows to an activity is In-Transition, and the one that flows from an activity is Out-Transition. Each activity should have only one In-Transition and one Out-Transition.
(4) Each swimlane should represent one constituent system. There should not be two different swimlanes which represent a same constituent system.

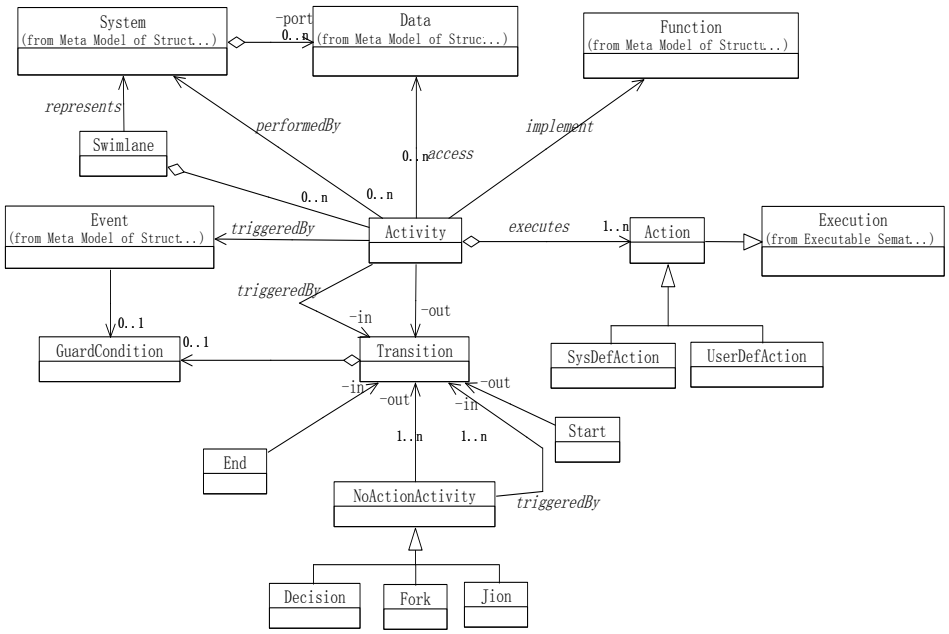

Fig. 3. The behavioral meta model of SoS

3.3 Executable Semantics of Models

To simulate the SoS, the mechanism and executable semantics must be defined. The structure models of SoS define the structure of systems and the types of data which are classified as Input or Output. The behavioral models of SoS define activities to simulate the behaviors of SoS. During the simulation, the simulator interprets the behavioral models and transform them into an executable language which will be discussed in Section 4.

The simulator is a multithreading system. It controls the execution of behavioral models. Once a SoS simulation is started, the simulator runs the SimulationMain and load the behavioral model. SimulationMain is the main thread of simulation. The simulator, while interpreting the behavioral model, creates a number of SimulationProcs to execute activities. It receives the instructions and data from its observation windows, and sends them to the SimulationProcs. If an activity has been executed, the SimulationProc is killed.

The executable semantics of models is described in Fig. 4. Each activity, including NoActionActivity, is executed in a SimulationProc. The Start creates a SimulationProc to execute the first activity. The End creates a SimulationProc to send an event to terminate the simulation. The Decision creates a Deciding SimulationProc to set the decision result which affects the conditions of its Out-Transitions and creates a SimulationProc for the subsequent activity whose In-Transition condition is satisfied. Fork creates a SimulationProc for every

subsequent activity whose In-Transition connects with it. Join creates a Waiting SimulationProc which waits for all SimulationProcs of the Fork activities being terminated and then creates a SimulationProc for the activity whose In-Transition connects with Join.

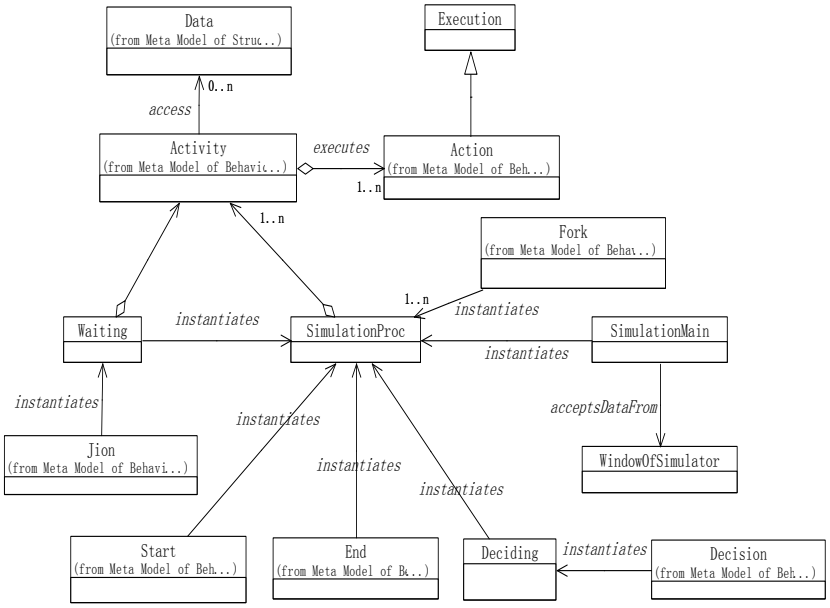

Fig. 4. The executable semantics of SoSEM

4. Formalization of Behavioral Model

Formalization is fundamental for the models to be executable. This section will focus on building the concrete syntax, algebra semantics and executable rules for the executable behavioral model (EBM) of SoS.

4.1 Concrete Syntax

Since activities are performed by systems, each of them should belong to an operator which is visually presented as a swimlane. Actions are encapsulated in the activities. The NoActionActivity and Transition control the execution sequence of activities. The Input and Output supply the information for execution of activities. The concrete syntax of EBM is thereafter defined with Backus-Naur Form (BNF), and is shown in table 1 where the terminators are highlighted in bold font.

Table 1 The concrete syntax of EBM

<activity model> ::= **activitymodel** <model name>;<operator list>;<control list> **end activitymodel**
<operator list> ::= <operator><operator list>|<>
<operator > ::= **operator** <operator name><activity list> **end operator**
<activity list> ::= <activity><activity list>|<>
<activity> ::= {<input>}<action>{<output>}
<input> ::= **in** <messageId> **from** <operator name>
<output> ::= **out** <messageId> **to** <operator name>

<action> ::= **action** <action name>{<precondition>}{<guard condition>}{<post condition>}
end action
<precondition> ::= **pre** <condition>
<guard condition> ::= **guard**<condition>
<post condition>::= **post**<condition>
<condition> ::= <expr>
<control list> ::= <control><control list>|<>
<control> ::= <fork>|<join>|<decision>|<transition>
<transition> ::= **transition** <source>{<condition>}<target> **end transition**
<fork>::= **fork** <source><target list> **end fork**
<join> ::= **join**<source list><target> **end join**
<decision> ::= **decision** <source><condition><target><target> **end decision**
<source list> ::= <source><source list>|<>
<target list> ::= <target><target list>|<>
<source> ::= **source** <operator name><action name> **end source**
<target> ::= **target** <operator name><action name> **end target**

4.2 Algebra Semantics

The execution of the EBM can be abstracted as running processes. Every activity or its containing action may take a process. An activity, when being performed by a system on the Input or Output data objects, would initiate one process for each containing action which is considered to be operated by the system. Therefore, an atomic process can be abstracted as a pair of operator and action. To formalize the EBM, we define an Algebras Process based language, the executable activity algebras (EAA), to explain the process semantics.

Definition 1. Assume *O* is the set of operators (or systems) and $o \in O$; *A* is the set of actions and $a \in A$; *P* denotes the process. The syntax of EAA can be thereafter defined with BNF as follows:

$$P ::= 0 \mid (o,a).P \mid P_1; P_2 \mid P_1 +_c P_2 \mid [< \text{expr} >]P \mid P_1 \parallel P_2 \mid P_1 \parallel_P P_2$$, where:

- 0(Empty): 0 is an empty process;
- $(o,a).P$ (Prefix): *P* becomes active when *a* has been executed by *o*;
- $P_1; P_2$ (Sequence): P_2 becomes active when P_1 has been executed;
- $P_1 +_c P_2$ (Choice): if *c* is true then P_1 becomes active; otherwise, P_2 becomes active;
- $[< \text{expr}>]P$ (Condition): if <expr> is true then *P* becomes active; otherwise, *P* is suspended;
- $P_1 \parallel P_2$ (Fork): P_1 and P_2 are executed concurrently;

- $P_1 \|_P P_2$ (Join): P becomes active when the two concurrent processes P_1 and P_2 have all finished their execution.

The EAA provides a domain of process algebras semantics for the EBM, or the process algebras semantics of EBM is an instance of EAA. Accordingly, we can translate the models of EBM into terms of EAA by building mappings between the syntax of EBM and EAA.

Followings introduce functions for mappings of the main productions of the concrete syntax. Assume the simulator interprets EBM by bottom-up compilation, and if the <string> is a non-terminator of the concrete syntax of EBM, then the function H(<string>) gives the set of all possible terminators from which the non-terminator <string> can be deduced.

Definition 2. The semantic functions for mapping of source or target are defined as:

$$S_{ac}(< source >) = (H(< operator\ name >), H(< action\ name >))$$
$$S_{ac}(< target >) = (H(< operator\ name >), H(< action\ name >))$$

Definition 3. The semantic functions for mapping of target list are defined as:

$$S(< target\ list >) = S_{ac}(< target_1 >) \| S_{ac}(< target_2 >),$$
$$\forall < target_1 >, < target_2 > \in < target\ list >$$

Definition 4. The semantic functions for mapping of fork are defined as:

$$S_{fo}(< fork >) = S_{ac}(< source >); S(< target\ list >),$$
$$< source >, < target\ list > \in < fork >$$

Definition 5. The semantic function for mapping of join is defined as:

$$S_{jo}(< join >) = S_{ac}(< source_1 >) \|_{S_{ac}(<target>)} S_{ac}(< source_2 >),$$
$$\forall < source_1 >, < source_2 > \in < source\ list >, < source\ list >, < target > \in < join >$$

Definition 6. The semantics function for mapping of decision is defined as:

$$S_{de}(< decision >) = S_{ac}(< source >); S_{ac}(< target_1 >) +_{H(<condition>)} S_{ac}(< target_2 >)$$

Definition 7. The semantics function for mapping of transition is defined as:

$$S_{tr1}(< transition >) = S_{ac}(< source >); [H(< condition >)] S_{ac}(< target >), < condition > \neq \phi$$
$$\text{or } S_{tr2}(< transition >) = S_{ac}(< source >); S_{ac}(< target >)$$

Definition 8. The semantic function for mapping of operators is defined as:

$$S_{op}(< operator >) = \{(H(< operator\ name >), H(< action\ name >))$$
$$| \forall < action > \in < activity\ list >, < activity\ list > \in < operator >\}$$

Basing on above functions, we provide the algorithm of transforming the model of EBM into the process terms of EAA.

Input: the whole deduction stack of EBM model

Output: the set of process terms of EAA P

Procedure

Step 1: create an empty set P to express the algebra model of EBM;

Step 2: for every <operator> of <operator list>, invoke the semantic function $S_{op}()$ to generate the atomic processes, (H(<operator name>， H(<action name>)), and add them to the set P.

Step 3: for every <transition> in the <control list>, invoke $S_{tr1}()$, if <condition> =null, or $S_{tr2}()$ to generate complex process terms and add them to the set P.

Step 4: for every <fork> in the <control list>, invoke $S_{fo}()$ to generate complex process terms and add them to the set P.

Step 5: for every <join> in the <control list>, invoke $S_{jo}()$ to generate complex process terms and add them to the set P.

Step 6: for every <decision> in the <control list>, invoke $S_{de}()$ to generate complex process terms and add them to the set P.

Step 7: output the set of process terms P.

4.3 Executable Rules

The simulation of SoS behaviors is driven by a set of derivation rules applied for evolution of process algebra. To support the evolution, we define the rule set as follows.

Prefix Rule

$$Action-1: \quad \frac{\square}{(o,a) \xrightarrow{(o,a)} 0}$$

$$Action-2: \quad \frac{\square}{(o,a).P \xrightarrow{(o,a)} P}$$

Rule *Action-1* shows that process *0* will become active when the process (*o*,*a*) is active and *a* has been executed by *o*. Rule *Action-2* shows that process *P* will become active when the process (*o*,*a*).*P* is active and *a* has been executed by *o*. $\square$ denotes that precondition of the rules is always true.

Sequence Rule

$$Sequence: \quad \frac{P_1 \xrightarrow{(o,a)} P_1'}{P_1; P_2 \xrightarrow{(o,a)} P_1'; P_2}$$

Rule *Sequence* shows that if the process P_1' can be activated by the process P_1 when *a* has been executed by *o*, the process $P_1; P_2$ will activate the process $P_1'; P_2$ when *o* has executed *a*.

Choice Rule

Choice-1:	$\dfrac{\square}{P_1 +_c P_2 \xrightarrow{c} P_1}$
Choice-2:	$\dfrac{\square}{P_1 +_c P_2 \xrightarrow{\neg c} P_2}$
Choice-3:	$\dfrac{P_1 \xrightarrow{(o,a)} P_1'}{P_1 +_c P_2 \xrightarrow{(o,a)} P_1'}$
Choice-4:	$\dfrac{P_2 \xrightarrow{(o,a)} P_2'}{P_1 +_c P_2 \xrightarrow{(o,a)} P_2'}$

Rule *Choice-1* shows that if condition *c* is true, $P_1 +_c P_2$ activates P_1 . Rule *Choice-2* shows that if condition *c* is false, $P_1 +_c P_2$ activates P_2 . Rule *Choice-3* shows that if P_1 activates P_1' when *a* has been executed by *o*, $P_1 +_c P$ activates P_1' when *a* has been executed by *o*. Rule *Choice-4* shows that if P_2 activates P_2' when *a* has been executed by *o*, $P_1 +_c P_2$ activates P_2' when *a* has been executed by *o*.

Condition Rule

$Condition-1:$	$\dfrac{\square}{[x=y]P \xrightarrow{[x=y]} P}$
$Condition-2:$	$\dfrac{\square}{[x=y]P \xrightarrow{[x\neq y]} 0}$

Rule *Condition-1* shows that if the condition *[x=y]* is true, *[x=y]P* activates *P*. Rule *Condition-2* shows that if the condition *[x=y]* is false, *[x=y]P* activates *0*. $\square$ denotes that precondition of the rules is always true.

Fork Rule

$Fork-1:$	$\dfrac{\square}{0 \,\|\|\, P_2 \xrightarrow{\square} P_2}$
$Fork-2:$	$\dfrac{\square}{P_1 \,\|\| 0 \xrightarrow{\square} P_1}$
$Fork-3:$	$\dfrac{P_1 \xrightarrow{(o,a)} P_1'}{P_1 \,\|\| P_2 \xrightarrow{(o,a)} P_1' \,\|\| P_2}$
$Fork-4:$	$\dfrac{P_2 \xrightarrow{(o,a)} P_2'}{P_1 \,\|\| P_2 \xrightarrow{(o,a)} P_1 \,\|\| P_2'}$

Rule *Fork-1* shows that $0 \,||\, P_2$ activates P_2. Rule *Fork-2* shows that $P_1 \,||\, 0$ activates P_1. Rule *Fork-3* shows that if P_1 activates P_1' when *o* has executed *a*, $P_1 \,||\, P_2$ activates $P_1' \,||\, P_2$ when *o* has executed *a*. Rule *Fork-4* shows that if P_2 activates P_2' when *o* has executed *a*, $P_1 \,||\, P_2$ activates $P_1 \,||\, P_2'$ when *o* has executed *a*.

Join Rule

Join-1:	$\dfrac{\square}{0 \Vert_P P_2 \xrightarrow{\square} P_2; P}$
Join-2:	$\dfrac{\square}{P_1 \Vert_P 0 \xrightarrow{\square} P_1; P}$
Join-3:	$\dfrac{P_1 \xrightarrow{(o,a)} P_1'}{P_1 \Vert_P P_2 \xrightarrow{(o,a)} P_1' \Vert_P P_2}$
Join-4:	$\dfrac{P_2 \xrightarrow{(o,a)} P_2'}{P_1 \Vert_P P_2 \xrightarrow{(o,a)} P_1 \Vert_P P_2'}$

Rule *Join-1* shows that $0 \Vert_P P_2$ activates $P_2; P$. Rule *Join-2* shows that $P_1 \Vert_P 0$ activates $P_1; P$.Rule *Join-3* shows that if P_1 activates P_1' when *o* has executed *a*, $P_1 \Vert_P P_2$ activates $P_1' \Vert_P P_2$ when *o* has executed *a*. Rule *Join-4* shows that if P_2 activates P_2' when *o* has executed *a*, $P_1 \Vert_P P_2$ activates $P_1 \Vert_P P_2'$ when *o* has executed *a*.

Evolution between process terms can be described with Trace. A process term *p* can evolve, subjected to change of active label (*o*,*a*), into a different term which is called successor of *p* and defined as *Succ*(*p*) = { $p' \mid \exists (o,a) : p \xrightarrow{(o,a)} p'$ }. If a term *p* evolve into another term p', there must be a sequence of active labels $\overline{(o,a)} = (o_1,a_1)(o_2,a_2)\cdots(o_n,a_n)(n>0)$ which result in $p \xrightarrow{(o_1,a_1)} p_1 \xrightarrow{(o_2,a_2)} \cdots \xrightarrow{(o_n,a_n)} p'$, and $\overline{(o,a)}$ is the trace of term *p* evolving into term p'. The trace is used to describe the execution sequence of executable units of the behavior model.

6. A Case Study

In this section, we give a simple example of threat air defense system (*TADS*) to demonstrate the availability of our theory. The executable model of *TADS* is built using the meta models in section 3. Due to limited pages, we provide only a fragment of the behavioral model and its algebra semantics.

There are four operators of *TADS*: *Radar Battalion* (*RB*), *Threat Command Center*(*TCC*), *Missile Battalion*(*MB*) and *Anti-air Gun Battalion*(*AGB*). The missile interception is simulated in following way. Once the *RB* has found an air target, the information will be sent to the *TCC* in time. The TCC will then make a threat assessment on it and decide whether they have to intercept it and who shall undertake the task. When *MB* or *AGB* have accepted the interception order and the target information, they will act on the target, and at the same time collect the data of interception effect and send them back to *TCC*. The *TCC* will make an assessment again to decide whether it is a successful interception or they have to arrange another interception. The model is illustrated in Fig. 6. The formal descriptions of the executable model, according to

concrete syntax of EBM discussed in Section 4.1, are shown in Table 2.

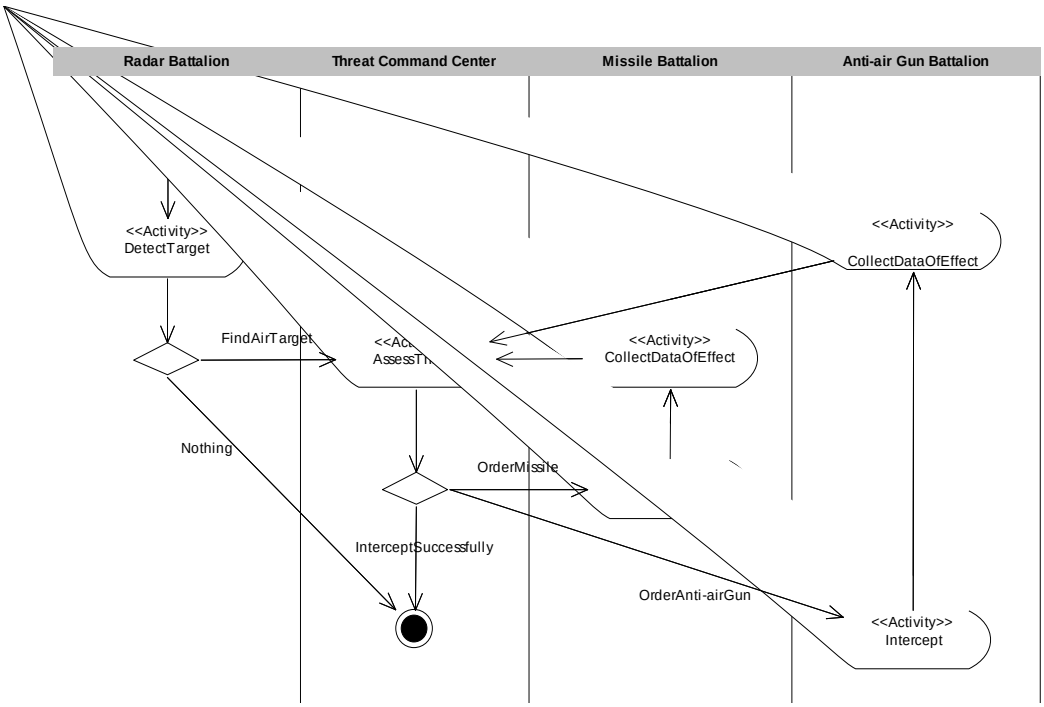

Fig. 6. The behavioral model of TADS

Table 2 The formal descriptions of the TADS behavioral model

activitymodel MissileIntercept;
operator RB
action DetectTarget **end action**
end operator
operator TCC
action AssessThreat **end action**
end operator
operator MB
action Intercept **end action**
action CollectDataOfEffect **end action**
end operator
operator AGB
action Intercept **end action**
action CollectDataOfEffect **end action**
end operator
decision
source RB DetectTarget **end source**
FindAirTarget
target TCC AssessThreat **end target**
target Over **end target**
end decision
decision
source TCC AssessThreat **end source**
OrderMissile
target MB Intercept **end target**
target Over **end target**
end decision
decision
source TCC AssessThreat **end source**
OrderAnti-airGun
target AGB Intercept **end target**
target Over **end target**
end decision
transition
source MB CollectDataOfEffect **end source**
target TCC AssessThreat **end target**
end transition
transition
source AGB CollectDataOfEffect **end source**
target TCC AssessThreat **end target**

```
    end transition
    transition
        source MB Intercept end source
        target MB CollectDataOfEffect end target
    end transition
    transition
        source AGB Intercept end source
        target AGB CollectDataOfEffect end target
    end transition
end activitymodel
```

Applying the transformation algorithm in Section 4.2, the EAA process terms of *TADS* activity model can be automatically generated and is listed as follows.

$P_1 = (RB, DetectTarget).P_2$

$P_2 = P_3 +_{FindAriTarget} 0$

$P_3 = (TCC, AssessThreat).P_4$

$P_4 = P_5 +_{OrderMissile} P_6$

$P_6 = P_7 +_{OrderAnti\text{-}airGun} 0$

$P_5 = (MB, Intercept).P_8$

$P_8 = (MB, CollectDataOfEffect).P_3$

$P_7 = (AGB, Intercept).P_9$

$P_9 = (AGB, CollectDataOfEffect).P_3$

The set of label is A= {(*RB, DetectTarget*), *FindAirTarget*, *Nothing*, (*TCC, AssessThreat*), *OrderMissile*, (*MB, Intercept*), (*MB, CollectDataOfEffect*), *InterceptSuccessfully*, *OrderAnti-airGun*, (*AGB, Intercept*), (*AGB, CollectDataOfEffect*)}.

To verify the activity model, let us check two label sets.

L_1 = ((*RB, DetectTarget*), *FindAirTarget*, (*TCC, AssessThreat*), *OrderMissile*, (*MB, Intercept*), (*MB, CollectDataOfEffect*), *InterceptSuccessfully*).

L_2 = ((*RB, DetectTarget*), *FindAirTarget*, (*TCC, AssessThreat*), *OrderAnti-airGun*, (*AGB, Intercept*), (*AGB, CollectDataOfEffect*), *InterceptSuccessfully*).

For L_1, we can observe the evolution trace of (P_1, P_2, P_3, P_4, P_5, P_8, P_3,0). And for L_2, we can observe the evolution trace of (P_1, P_2, P_3, P_4, P_6, P_7, P_9, P_3,0). The two traces are all valid, for they all end by 0 which means the *End* of activity model. The above process traces show that the air target is intercepted by either *MB* or *AGB*.

7. Conclusions

The behavioral models of SoS architecture are usually built with UML activity diagram. Due to poor formalization of UML, these models are not executable, which brings inconveniency during evaluation of SoS architecture. The paper suggests a new approach of executable modeling and simulation. The main contributions are summarized as follows:

(1) The meta models for both structures and behaviors of SoS are presented by extending fUML meta models. With these meta models, the engineers may accurately describe SoS

architecture and build UML models.

(2) The concrete syntax and algebra semantics of executable models are discussed in details. And a number of semantic functions are introduced for mapping of the syntax descriptions of behavioral models into items of executable activity algebras.

(3) Executable rules are defined. With these rules, the process terms which simulate the behaviors of SoS can evolve as process traces, which make model analysis and validation easier.

However, our work is just in beginning. We are challenged by dealing with existing products of behavioral models, some of them being built in other paradigms such as UML sequence diagram and IDEF 3. The future research will be on extending the meta models to make our approach to be compatible with other modeling paradigms.

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[J]. IEEE Systems Journal, 2010,4(2):156-166.

[2] DoD Architecture Framework Working Group. DoD Architecture Framework 2.0[R]. The United States Department of Defense, 2009.

[3] Pandey R. K. . Architecture Description Languages(ADLs) vs UML: A Review[J]. ACM SIGSOFT software Engineering Notes,2010,35(3)

[4] Group Object Management. UML 2.0 Superstructure Specification[EB/OL]. http://www.omg.org/spec/UML/2.0/superstructure/ PDF/, 2005.

[5] Group Object Management. Semantics of a Foundational Subset for Executable UML Models (fUML) 1.0[EB/OL]. http://www.omg.org/spec/fUML/1.0/PDF/,2011.

[6] Thomas J. Pawlowski III, Paul C. Barr, Steven J. Ring, et al. Executable Architecture Methodology for Analysis, FY04 Final Report [R]. MITRE, September, 2004

[7] Jiang Jun. Research on Executable Architecture and the Executable Method of DoDAF[D]. Hunnan: Dissertation of National University of Defense Technology. 2008, 9.

[8] Wang Lei, LUO Xue-shan and LUO Ai-min. Research on C4ISR Executable Architecture Based on SOA [J]. Fire Control & Command Control, 2012,37(1):52-56.

[9] Wang Renzhong and Dagli C H. An executable system architecture approach to discrete events system modeling using SysML in conjunction with colored Petri net[C]. 2nd Annual IEEE International Systems Conference, 2008:1-8.

[10] Staines T S. Intuitive mapping of UML2 activity diagrams into fundamental modeling concepts Petri net diagrams and colored Petri nets[C]. 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2008,191-200.

[11] Jiang Jun, Bai Xiaoli, Luo Xueshan, et al. Study on the method of transforming IDEF3 process model to object Petri-net model[J]. Systems Engineering and Electronics ,2008,30(12):2434-2438

[12] Bingfeng Ge, Keith W.Hipel, Kewei Yang et.al. A Data-Centric Capability-Focused Approach for System-of-Systems Architecture Modeling and Analysis. Systems Engineering Vol. 16, No. 3, 2013,pp:363-377

[13] Luc Touraille, Manmadou K. Traore and David R.C. Hill. "A Model-Driven Software Environment for Modeling Simulation and Analysis of Complex Systems", Proc. of the *2011 Symposium on Theory of Modeling & Simulation : DEVS Integrative M&S Symposium*, 2011,pp: 229-237.

[14] Ahmet Kara, Fatih Deniz, Doruk Bozagac et al. "Simulation Modeling Architecture(SiMA) A DEVS based Modeling and Simulation Framework". Proc. of the *2009 Summer Computer Simulation Conference*, 2009,pp: 315-321.

[15] Deniz Cetinkaya, Alexander Verbraeck and Mamadou D.Seek. MDD4MS: "A Model Driven Development Framework for Modeling and Simulation", Proc. of *the 2011 Summer Computer Simulation Conference*, 2011, pp: 113-121.

[16] Jim Woodcock and Alvaro Miyazawa. CML Definition 2. Public Document. Deliverable Number: D23.3-1, COMPASS Project, University of York,March, 2013.