Generating Executable Capability Models for Requirements Validation

Weizhong Zhang
Institute of Command Automation, PLA University of Science & Technology, Nanjing, China.
Email: willson_zwz@163.com

Zhixue Wang[1], Wen Zhao[1, 2], Yingying Yang[1], Xin Xin[3]
1. Institute of Command Automation, PLA University of Science & Technology, Nanjing, China.
2. Army Reserve Duty No.47 Infantry Division of Liaoning, Siping, China
3. Xi'an Communication Institute of PLA, Xi'an, China.
Email: {wzxcx, chip_ai , flyto2016, zwjz1029}@163.com

***Abstract*--Executable modeling allows the models to be executed and treated as prototypes to determine the behaviors of a system. In this paper, we propose an approach for formalizing requirement models and generating executable models from them. Application activity diagrams (AADs), which are used to represent dynamic behaviors of systems in capability requirements models, are firstly formalized and saved as XML documents. Then, on the basis of these models, a mapping algorithm of translating AADs into instances of executable models for simulation is proposed. A case study is finally given to demonstrate the applicability of the method.**

***Index Terms*--UML, Capability requirement, Executable model, Simulation**

I. INTRODUCTION

Architectural analysis of Information systems (IS), such as Command, Control, Communication, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) system requirements is a hard and challenging work. Most architecture development methods, such as UK Ministry of Defence Architecture Framework (MoDAF) [1] and US Department of Defense Architecture Framework (DoDAF) [2], recommend Unified Modeling Language (UML) to model the capability concepts and C4ISR requirements. DoDAF provides a set of UML meta-models to define its Meta-Model Data Groups [2]. In our early research, Capability Requirements Modeling Language (CRML) was proposed and applied to C4ISR capability requirement analysis [3]. CRML extends UML meta-model [4] [5][3], with the various views such as class diagrams and activity diagrams etc.

However, some problems might arise if UML is directly applied to the capability analysis. It is a semi-formal and weakly constrained modeling language. As a result, it's hard to build rigorous and formal system models. The normal UML models are not executable because they describe the dynamic behavior of models with natural language, and hence cannot be directly used to verify and validate the system behaviors in stage of analysis and design.

The popularly applied methods include Petri Net [6] [7], executable UML (xUML) [8] [9] etc. But when the Petri Net applied, the requirements models built with UML have to be transformed into those of Petri Net, and some part of model information might be lost during the transformation due to the different model semantics. xUML is designed to describe the software behavior from the view of program execution and cannot directly applied to describe the behavior of the models of system requirements and architecture unless the modeling semantics are refined and the action semantics are redefined.

In this paper, we suggest a method of generating an executable model for requirements validation by providing action semantics of domain models, focusing on the domain of C4ISR system architectures. Firstly, a capability meta-concept model is provided extending the UML constructs. Then, an algorithm mapping Application activity diagrams (AADs) to simulation instances is proposed. And, a case study is given to show the final executable codes and demonstrate the applicability of the method.

The rest of the paper is organized as follows. Section II describes the research background, including architectural modeling and simulating methods. Section III describes the language of capability requirement modeling and the extending method of AADs. Section IV gives an explicit and formalized definition of AADs. Section V discusses in detail the algorithm of generating executable capability requirement models. Section VI demonstrates the applicability of the method by analyzing a Missile Intercept example.

II. RELATED WORK

An enterprise is one or more organizations sharing a

Weizhong Zhang PhD. Tel: +86−13851564082.
E-mail: willson_zwz@163.com

definite mission, goals and objectives to offer an output such as a product or a service [10]. There are many different approaches to enterprise system engineering and integration, which leads to various enterprise architecture frameworks, such as Generalized Enterprise Reference Architecture and Methodology (GERAM)[11], Zachman Framework [12], DoDAF [2], The Open Group Architectural Framework (TOGAF) [13], Treasury Enterprise Architecture Framework (TEAF) [14], etc.

Among them, the DoDAF provides an architectural modeling methodology for defense information system, nowadays referred to C4ISR system [15][16]. Such mission-critical systems are growing in complexity as more computing devices are networked together to help automate tasks previously done by human operators [17], which brings tremendous difficulties in system engineering and integration. The system engineers have to pay great efforts on simulation to validate the built architecture models.

The architecture models and simulation models are usually built in different ways, because the architecture modeling aims at capturing higher level of system requirements and the modeling paradigms usually lack of rigorous and executable semantics. One of the popular solutions is to find semantic mappings between the two different modeling paradigms and transform the architecture models into the simulation models. As for building DoDAF executable architecture models, the research community focuses on mapping between UML and Discrete Event System Specification (DEVS) modeling.

Wagenhals et al. [18] provide a description of an architecting process based on the object-oriented UML. They describe a mapping between the UML implementations and an executable model based on Colored Petri-nets, focusing on the UML Sequence Diagram (OV6c), the UML Collaboration Diagram (OV5b) and the Class Diagram (OV5a-with extensions). Ziegler and Mittal [19] describe the translation of DoDAF compliant architectures into DEVS simulations by providing a set of DoDAF foundational Views and related UML diagrams for construction of DEVS-based simulations. Mittal et al [20] describe a means for semantically strengthening the critical OV-6a Rules Model, through application of Domain Meaning, Units of Measure (UOM), and formatting to domain specific rules, thereby removing ambiguity and aiding in translation of static to dynamic architectures. Risco-Martin et al. [21] describe the essential mappings between UML and DEVS modeling, focusing on the UML Structure and Behavior models that contribute to the development of a DEVS-based system model. Those UML models are the Component Diagram, the State Machine, the Sequence Diagram, and the Timing Diagram. Andrade et al [22] present the methodology for mapping System Modeling Language (SysML) activity diagram to time Petri-net.

III. Capability Requirement Modeling

A. Capability Meta-concept Model

The Capability meta-concept model, as shown in Fig.1, defines the capability ontology of C4ISR systems [3], which contains basic concepts that describe C4ISR system architecture. Those concepts include OperationalNode, OperationalEntity, Activity, Mission, Capability, Information, etc.

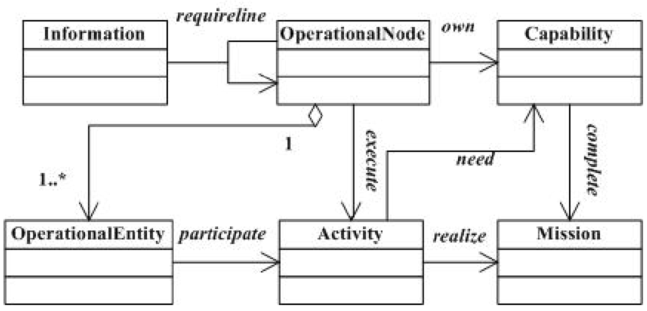

Fig.1 Capability meta-concept model

From the architectural viewpoint, the concept OperationalNode defines a logic node which possesses capabilities for completing a mission. The communication requirements between nodes are specified by the needlines, and exchanging Information associated with the needlines describes the contents of communication. Capability describes the resources, such as information systems, weapon platforms and materials, which are owned by an operational node and which are required for a mission. OperationalEntity describes the participants of activities. The operational entities are the main component of an operational node which executes activities. Activity describes a number of operations to realize a mission. Mission defines a business goal and task details of an enterprise.

B. Application Activity Diagrams

The behavior of a C4ISR system is specified in an Application Activity Diagram model and it is the main concern of the paper. Such model is built with a UML paradigm which extends UML 2.1 constructs of Class Diagram and Activity Diagram with the meta-concepts of capability ontology. It describes which operational nodes participate in the activities to realize the mission goals, what activities are executed in the course and their time sequences, what capabilities are required, and so on.

With UML profile idea, we define new stereotypes of UML2.1 constructs according to the meta-concept model. Fig.2 reflects the UML extension mechanism, where the Class at Meta Object Facility (MOF) level is elaborated as the meta-concepts such as <<OperationalNode>> etc.

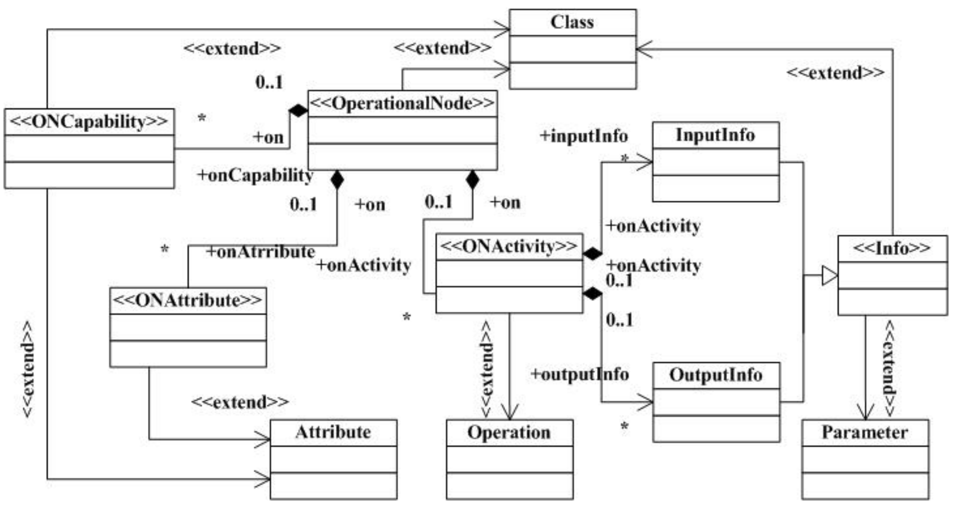

Fig.2 OperationalNode structure based on UML extension

Tab.1 reflects the UML extension, where the Class is extended by the meta-concept OperationalNode and denotes an operational node. ONAttribute defines the property of an operational node, and extended from MetaClass *Class::Attribute*. ONCapabilty, extending the meta-concept Capability, defines the capability property of an operational node and extends *Class::Attribute* of UML MOF.

TABLE.1

METACLASS AND SEMANTIC OF OPERATIONALNODE

Stereotype	MetaClass	Semantics
OperationalNode	Class	Define an operational node
ONAttribute	Class:: Attribute	Define an attribute of an operational node
ONActivity	Class:: Operation	Define an activity executed by an operational node as its an operation
ONCapabilty	Class:: Attribute	Define a capability owned by an operational node as its an attribute
Info	Class:: Parameter	Define an information received or sent by an operational node as its input or output parameter

IV. FORMALIZATION OF MODELS

To describe dynamic behaviors of well-built models in detail, we propose a method of model formalization, which will profit models simulating.

In an AAD, there are usually five types of basic elements: *Swimlane*, *Activity*, *Capability*, *Info* and *Flow*. Therefore, an AAD can be defined as follows:

Definition 1: *An application activity diagram model is described by AADM ::= <CapabilityList, InfoList, ActivityList, FlowList, SwimlaneList >:*

Definition 2: *A CapabilityList is defined as follows:*

- *CapabilityList ::={ <Capability >}*
- *Capability ::= <Attribute> {<CapValue>}*
- *Attribute ::= <ID, Name>*
- *CapValue ::= <Name, Type, Value>*
- *Type ::= Int | Float | String | Bool*

CapabilityList in an AAD model is an element of the set – *Capability*, which contains some *Attributes* and *Capvalues*. The *Attribute* has its unique *ID and a Name*, while the *Capvalue* is represented as a three-tuple *<Name, Type, Value>*.

Definition 3: *A InfoList is defined as follows:*

- *InfoList::={ < Info >}*
- *Info ::= <Attribute, ReceiveNode, InfoValue>*
- *Attribute ::= <ID, Name>*
- *ReceiveNode ::= <ONID>|NULL*
- *InfoValue ::= {< Name, Type, Value>}*
- *Type ::= Int | Float | String | Bool*

InfoList in an AAD model is an element of the set – *Info*, which contains some *Attributes*, *ReceiveNodes and Infovalues. ReceiveNode* denotes which OperationalNode receives the *Info*. *<ONID>* indicates the right OperationalNode, and *NULL* indicates the *Info* is received by an end node.

Definition 4: *A ActivityList is defined as follows:*

- *ActivityList::={ < Activity >}*
- *Activity ::= <Attribute, Input, ActBody>*
- *Attribute ::= <ID, Name, Capability>*
- *ActBody ::= {<Event>}*
- *Event ::= <Condition, Action, Output>*

ActivityList in an AAD model is an element of the set –*Activity*, which contains some *Attributes*, *Inputs and ActBodys. Capability* is derived from *CapabilityList,* and here is an attribute belongs to an *Activity*. *ActBody* is a series of *Events*, and produces *Outputs* after taking some *Actions*.

Definition 5: *A FlowList is defined as follows:*

- *FlowList::={ <Flow>}*
- *Flow::= DataFlow | CtrlFlow*
- *DataFlow ::= <ID, SAct, DAct, Info, Condition>*
- *CtrlFlow ::= < Join> | < Fork> | <Decision>*

FlowList in an AAD model is an element of the set –*Flow*, which is denoted as a transition line. The *Flow* can be divided into two types of *DataFlow* and *CtrlFlow*. The former describes the flowing *Infos* from source

Activity *SAct* to destination Activity *DAct* and the guard *Condition.* The later decides the way of the execution of several actions by controlling the execution sequence of actions. A *Join* can be described as a tuple, j-n = < *Activity*, n, JOIN>. If all *Activity* nodes complete, then n is invoked. A *Fork* can be described as a tuple, f-n = (n, *Activity*). If n completes, then every *Activity* node is invoked. A *Decision* can be described as a tuple, d-n = (n, (c1,n1)… (cn, nn)). If n completes, and the *Decision* condition ci is true, then *Activity* node ni is invoked. When *Decision* node is invoked, only one *Decision* condition is true among c1… cn.

Definition 6: *A SwimlaneList is defined as follows:*

- *SwimlaneList::={ < Swimlane >}*
- *Swimlane ::= <Attribute, Element>*
- *Attribute ::= <ID, Name, Position >*
- *Element ::= STARTNODE | ENDNODE | <Activity> | <CtrlFlow>*
- *Element ::= <ID, Name, Swimlane, Position, Type, ActivityID | NULL, IsNested | NULL>*
- *Type ::= ACT | NOTACT*

SwimlaneList in an AAD model is an element of the set –*Swimlane*, which contains *Attributes and* several *Elements.* The elements in a *Swimlane* include start node, end node, some activities and control flows. The *ActivityID* and *IsNested* are only meaning when the type of *Element* is *ACT*. The *Swimlane* in *Element* indicates which swimlane the element belongs to.

We have defined all the model elements in an AAD. Now, we can save the AADs with XML documents format, which holds well reusability, expansibility and readability and prepares for simulating. The formalization models are shown as following format:

```
<ActivityDiagram>
  <CapabilityList>
  <Capability ID="" Name=" ">
     <CapValue Name=" " Type=" " Value="" />
       …
  < /CapabilityList>
  < InfoList>
  <Info ID=”” Name=”” ReceiveNode=””>
     < InfoValue Name=" " Type=" " Value="" />
          …
  < /InfoList>
  <ActivityList>
  <Activity ID=”” Name=”” Capability=”” Input ID=””>
  <Event Condition=”” Act=”” Output=””>
          …
  < /ActivityList>
  <FlowList>
  <Flow ID=”” SAct=”” DAct=”” Info=”” Condition=””/>
  </FlowList>
  <SwimlaneList>
  <Swimlane ID=”” Name=””>
     <Element ID=”” Type=”” Position=”” Nested=”” />
       …
  </SwimlaneList>
</ActivityDiagram>
```

V. Translate AADs Into Instances For Simulation

Having made the UML extension and defining the domain modeling semantics, we can translate AAD models into simulation instances for requirements validation. An algorithm is provided as follows

Step 1: Each *Swimlane* in AAD is translated into a simulation instance of OperationalNode.

Step 2: Each *Activity* within the *Swimlane* is translated into the instance of an ONActivity.

Step 3: For each *Activity*, the object inflow is regarded as its input and the object outflow is regarded as its output. The input and output, which are instantiated by the Class stereotyped with Information, are treated as parameters and return values for the operation invocation. The nodes *Decision*, *Fork* and *Join* are respectively processed through following steps:

Step 3.1: The output of an *Activity* before the *Decision* is considered as the outflow of the *Decision* node.

Step 3.2: Identify concurrent processes for the activities following the *Fork*, which can be realized with reload mechanism.

Step 3.3: All inflow of the *Join* is considered as input parameters of the *Activity* following the *Join*.

Step 4: Every *Capability* is translated into an ONCapability of the OperationalNode that owns it.

Step 5: The Start node sends initialization information which triggers the other activities execution.

The Tab.2 shows the mappings between ADD modeling constructs and generated simulation instances. Applying a programming language, the simulation program can be generated from those instances.

TABLE.2

MAPPINGS BETWEEN ADD CONSTRUCTS AND SIMULATION INSTANCES

ADD constructs	Simulation instances
Swimlane	OperationalNode
Activity	OperationalNode.ONActivity
Capability	OperationalNode.ONCapability
Transition	OperationalNode.Activity.Input or OperationalNode.Activity.Output

After translating AADs into instances, we are prone to use proper OO programming languages to realize and execute them. For example, C# language is used to construct the Class templets of OperationalNode, Capability and Info, as show in Fig.3.

```
Class OperationNode
{
    Capability capability;
    ......                    //other capabilities
    Info Activity (Info info)
   {
       ......                //content of activity}
   ......                    //other activities
}
Class Capability
{
    ValueType value;
    ......                   //other values of capability
}
Class Info
{
     ValueType value;
     ......                  //other values of information
}
```

Fig.3 Class templets of OperationalNode, Capability and Info with C#

VI. Implementation Of Model Simulation

We take an example to illustrate implementation of the simulation instances. Fig.4 shows an AAD model of the activity of missile interception for the air defense system.

The swimlanes of the Activity Diagram represent operational nodes. For example, XXX MInterceptON is an instance which is defined by the class MissileIntercepNode and owns a capability of CapabilityOfMissileIntercept which has valued properties EffectiveRange (type: int, value: 100, unit: Km), FirstHitRatio (type: Float, value: 0.85), etc.

Firstly, according to the translation algorithm, the AAD can be translated into simulation instances, for example, the instances XXX MinterceptON and ONCapability are shown in Fig.5. They look like executable programs, but are only pseudocode templates, which can not be executed. They have to be further translated into ones in a programming technology, for example, the Code Document Object Model (CodeDOM) [23]. CodeDOM is a kind of mechanism that contained in .NET Framework. It is used to generating modularization code and compiling dynamically.

In above example, the instances of OperationalNodes, ONCapabilities, ONActivities and Infos are added to the object structure in CodeDOM, including namespaces, properties, and methods and so on, which are stored in CodeCompileUnit. CodeDOM offers a compiling mechanism which can compile code objects dynamically, as Fig.6. Therefore, the graphic models are ultimately transformed into executable models.

VII. Conclusion

The paper presents an approach a capability meta-concept models, for providing an executable capability modeling language to realize simulating of requirement models. The approach extends UML paradigm by adding the domain modeling constructs of DoDAF, defining the structure and semantics of operational nodes and related concepts, and building mappings between the surface grammars of AADs and the execution instances. CodeDOM in .NET Framework is used to support dynamically compiling and generating the executable capability requirement models. And, a case study is provided to show the availability of our method. The future research will be on modeling evaluation indicators of mission effectiveness for the capability requirements, generating executable models for the indicators and thereby realizing integration of simulation and evaluation for the C4ISR system.

Acknowledgement

The research is supported by the National High-Tech Research and Development Program of China under the project number 2007AA01Z126 and the National Defense Advanced Research Program of China under the project number 51306010202.

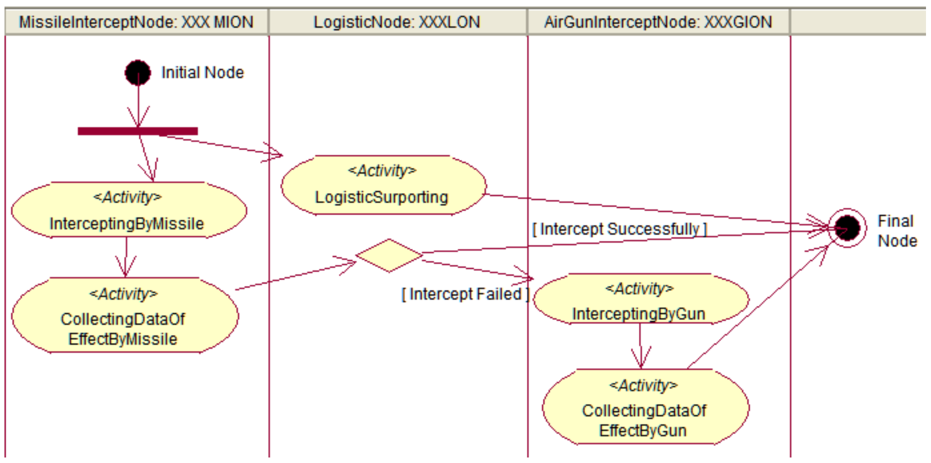
MissileInterceptNode: XXX MION
LogisticNode: XXXLON
AirGunInterceptNode: XXXGION
Initial Node
<Activity> InterceptingByMissile
<Activity> LogisticSurporting
<Activity> CollectingDataOf EffectByMissile
[Intercept Successfully]
[Intercept Failed]
<Activity> InterceptingByGun
<Activity> CollectingDataOf EffectByGun
Final Node

Fig.4 An AAD of Missile Intercept

```
OperationalNode XXX MinterceptON
{
    ONCapability CapabilityOfMissileIntercept;
    Info InterceptingByMissile (Info distancefromtarget)
    {       //content of activity
       if (distancefromtarget<= CapabilityOfMissileIntercept.EffectiveRange & distancefromtarget>5)
          send the command CommandOfInterception ="Intercept the target by missile";
       else
          generate a random number x between 0 and 1;
          possibilityofhitbymissile = x;
    }
    Info CollectingDataOfEffectByMissile (Info possibilityofhitbymissile)
    {       //content of activity
       If (possibilityofhitbymissile/x > 0.8)
          distancefromtarget = -1;     //Intercept successfully
       else
          send the command CommandOfInterception ="Intercept the target by Gun";
    }
}
ONCapability CapabilityOfMissileIntercept
{
    Int EffectiveRange = 100;
    Float FirstHitRatio =0.85;
}
```

Fig.5 Instances of OperationalNode and CapabilityOfMissileIntercept

```
public object DoCompiler(CompileUnit unit, string className)
{
    object operationNodeInstance;                                //ON Instance
    CompilerParameters cp = new CompilerParameters();            //parameters for compile
    cp.GenerateExecutable = true;                                //generating executable file
    cp.GenerateInMemory = true;                                  //generating in memory
    CompilerResults _cr =CodeDomProvider.CreateProvider("CSharp").CompileAssemblyFromDom(cp, unit);
    Assembly _assembly = _cr.CompiledAssembly;                  //return program set
    operationNodeInstance=_assembly.CreateInstance(className); //ON Instance names
    Type type = operationNodeInstance.GetType();                          //ON Instance types
    Object o = type.GetProperty("XXX").GetValue();        //ON Instance property values
    MethodInfo[] mi = type.GetMethods();                          //ON Instance methods
}
```

Fig.6 Compiling executable codes

REFERENCES

[1] UK Ministry Of Defence. MOD architecture framework overview v1.0. http://www.modaf.org.uk/, 2005.

[2] US Department of Defense. DoD architecture framework version 2.0 (Volume I-II-III). http://www.us.army.mi/suite/page/454707, MOD Partner, 2008.

[3] Zhixue W, Qingchao D, Bin C, et.al. Towards C4ISR system capability requirements analysis and validation based on UML model. Systems Engineering and Electronics, 2009, 31(9), pp: 2167-2171.
[4] OMG. OMG Unified Modeling LanguageTM Superstructure. Version2.1, formal/2007-02-03. http://www.omg.org/spec/UML/2.1/Superstructure.
[5] OMG. OMG Unified Modeling LanguageTM Infrastructure. Version2.1, formal/2007-02-04. http://www.omg.org/spec/UML/2.1/Infrastructure.
[6] Liqiong C, Guisheng F, Yunxiang L. An Approach to Formally Modeling and Verifying Distributed Real-time Embedded Software. Journal of Software, 2010, 5(9), pp: 990-997.
[7] Vincent C, Matthieu R. Interoperability Constraints and Requirements Formal Modelling and Checking Framework. International Federation for Information Processing, 2010, pp: 219-226.
[8] Chris R, Paul F, John W, et.al. Model Driven Architecture with Executable UML. Cambridge: Cambridge University Press, 2004.
[9] Helle H, Jeroen K, Bas L, et al. Towards model checking executable UML specifications in mCRL2. Innovations in Systems and Software Engineering, 2010, 6(1), pp: 83-90.
[10] Neaga E, Harding J. An enterprise modeling and integration framework based on knowledge discovery and data mining. International Journal of Production Research, 2005, 43(6), pp: 1089-1108.
[11] Ehsan M, Ali S, Mohammad R. Using Axiomatic Design in the Process of Enterprise Architecting. Third International Conference on Convergence and Hybrid Information Technology, 2008.
[12] Noran O. An analysis of the Zachman framework for enterprise architecture from the GERAM perspective. Annual Reviews in Control, 2003, 27(2), pp: 163-183.
[13] Open Group. TOGAF: The Open Group Architecture Framework, Version 9, Document Number: G091. The Open Group, United Kingdom, 2009.
[14] Department of the Treasury CIO Council. Treasury Enterprise Architecture Framework. V.l. Department of the Treasury, Washington D.C., USA, 2000.
[15] Schorling S, Rine D. A methodology for designing toolkits for specification level verification of interval-contained information systems requirements. Information and Software Technology, 2002, 44(2), pp: 77-90.
[16] Noran O. A systematic evaluation of the C4ISR AF using ISO15704 Annex A (GERAM). Computers in Industry, 2005, 56(5), pp: 407-427.
[17] Lardieri P, Balasubramanian J, Schmidt D, et.al. A multi-layered resource management framework for dynamic resource management in enterprise DRE systems. The Journal of Systems and Software, 2007, 80 (7), pp: 984-996.
[18] Wagenhals L, Haider S, Levis A. Synthesizing Executable Models of Object Oriented Architectures. Proceedings of Workshop on Formal Methods Applied to Defence Systems, Adelaide, Australia, 2002, pp: 85-93.
[19] Zeigler B, Mittal S. Enhancing DoDAF with a DEVS-based System Lifecycle Development Process. IEEE International Conference on Systems, Man and Cybernetics, Hawaii, October, 2005.
[20] Mittal S, Mitra A, Gupta A, et.al. Strengthening OV-6a Semantics with Rule-Based Meta-models in DEVS/DoDAF based Life-cycle Architectures Development. IEEE Information Reuse and Integration, Special Section on DoDAF, Hawaii, 2006, pp: 80-85.
[21] Risco-Martin J, de la Cruz J, Mittal S, et.al. Eudevs: Executable UML with DEVS Theory of Modeling and Simulation. Simulation, 2009, 85(7), pp: 419-450.
[22] Andrade E, Maciel P, Callou G, et.al. A methodology for mapping SysML activity diagram to Time Petri net for requirement validation of embedded real-time systems with energy constraints. 3rd International Conference on Digital Society, 2009, pp: 266-271.
[23] MSDN Library. http://msdn.microsoft.com/zh-cn/library/f7ykdhsy.aspx, 2010.

Weizhong Zhang was born in 1983. He is a PhD student of Institute of Command and Automation, PLA University of Science and Technology. His research interests include requirements engineering and model simulation. He received bachelor degree in command automation engineering and master degree in system engineering, both from PLA University of Science and Technology.

Zhixue Wang was born in 1961. He is a professor of Institute of Command Automation, PLA University of Science and Technology. He received bachelor degree in computer engineering from Hefei Polytechnic University and master degree in computer science from National University of Defense Science and Technology, P.R. China. His research interests include software engineering, requirements engineering, theory and technology of command automation, currently focusing on domain-specific modeling and formal verification. He used to be a visiting researcher in Faculty of Information Technology, University of Brighton, England.

Wen Zhao was born in 1983. He is a MS student of Institute of Command Automation, PLA University of Science and Technology. His research interests include system engineering and formal specification and simulation. He received bachelor degree in computer science from PLA University of Science and Technology.

Yingying Yang was born in 1980. She is a PhD student of Institute of Command Automation, PLA University of Science and Technology. Her research interests include requirement engineering and formal specification. She received bachelor degree in command automation engineering and master degree in system engineering from PLA University of Science and Technology.

Xin Xin was born in 1983. He is a assistant of Xi'an Communication Institute of PLA. His research interests include computer application and wireless communication. He received bachelor degree and master degree in computer science from PLA University of Science and Technology.