Proceedings of the 2012 2nd International Conference on Computer and Information Application (ICCIA 2012)

The Algebraic Semantics of EPDL at Activity Level and Verification

Jinzhuo Liu

School of Software,
Yunnan University
Kunming, 650091, China
E-mail: jinzhuo.liu@hotmail.com

Lixia Wang

School of Economics,
Yunnan University
Kunming, 650091, China
E-mail: lxwang@ynu.edu.cn

Tong Li, Qian Yu, Na Zhao,
Feilu Hang

School of Software,
Yunnan University
Key Laboratory in Software
Engineering of Yunnan Province,
Kunming, 650091, China

Abstract— **In recent years, algebraic semantics and its verification are increasingly important in software engineering. In this paper, firstly, the algebraic semantics of software evolution process description language (AS-EPDL) at activity level is explored. The algebraic semantics of activity level in EPDL provide an accurate framework for defining the semantics. In addition, the hierarchy-consistency and sufficient-completeness properties of the AS-EPDL at activity level are verified.**

Keywords- Software Evolution Processes; Activity; EPDL; algebraic semantics.

I. INTRODUCTION

Software evolution process is the interdisciplinary of software process and software evolution which are two key areas in software engineering.

A software evolution process is a set of interrelated software processes under which the corresponding software is evolving [1]. The traditional software process cannot well-supported a software evolution framework. Therefore, Li[1] defined a formal evolution process meta-model(EPMM) which is based on the extended Petri Net mixed with object-oriented technology and Hoare Logic to construct software evolution process models with four-level architecture. But EPMM as a software evolution process model is only a static and abstract description of a software evolution process. Therefore, Li[1] designed a more detailed description of a software evolution process-software evolution process description language (EPDL) which expend the description power of EPMM. EPDL is an object-oriented computer language. The syntax of EPDL is formally defined with Extended Backus-Naur Form. However the most important semantics of EPDL are described informally. This may lead to ambiguity and affect the proper use of language. In this paper, we effectively describe the semantics of activities level of EPDL based on algebraic semantics. And also we verified the hierarchy-consistency and sufficient-completeness properties. Its use, however, is contributed to the normalization and correctness of the definition of semantics.

II. BACKGROUND INFORMATION

A. Introduction to EPDL

As Osterweil[2] has showed clearly that Human being must employ some powerful process abstractions due to the complexity of software process entities. Because software processes are complex entities, researchers have created a number of language that make it possible to represent in a precise and comprehensive way a number of software process features and facets[1]. These languages must be tolerant and allow for incomplete, informal, and partial specification [4].

A software evolution process description language is a computer language that is used to describe software evolution processes [1]. As the EPMM only defined in an abstract way, it cannot be executed. So, a software evolution description language should be defined as a program to show the detail of a software evolution process. Due to the requirement of software evolution, Li[1] designed an object-oriented software evolution process description language (EPDL).

As EPDL expend EPMM, EPDL syntax components have four levels: the task, the activity, the software process, the global model. The structure of EPMM and EPDL are the same. EPDL can embody the evolutionary characteristic and give worthy information of every abstraction level.

In this paper, we define the activity level of EPDL based on abstract data type. Afterwards, we verify two properties - hierarchy-consistency and sufficient-completeness based on algebraic semantics.

The activity level describes the inner structure of an activity. An activity is a set of tasks. The task level describes the function and messages of a task. A task is a method (or operation) of an activity[1]. It describes a class in an object-oriented system. A software process can be regarded as an object-oriented system[1].

B. Preliminary

The following is the definition of algebraic semantics using in this paper. For the sake of conciseness, some of the formal definitions are omitted. ∑ Algebra is directly used to describe EPDL. The abstract data type is made up of sorts, operation and axiom. In order to clearly understanding the

© the authors

description, the following algebraic definitions need to introduce beforehand.

Definition 2.1[3] keynote is 2-tuple Σ=(S, O) iff

1) S={s_i | i∈I}s a finite set. I is a finite subscript set. Each s_i is called a sort. $s_i = s_j$ or $s_i \neq s_j$ or $s_i \cap s_j = \varnothing$ or $s_i \cap s_j \neq \varnothing$;

2) O={o_j | j∈J}is a finte set. J is a finite subscript set. Each o_j is called an operation.

Definition 2[3] suppose 2-tuple Σ=(S, O) is a keynote, 2-tuple(A, F) is called a Σ algebra[3]. iff

1) A={a_i | i∈I} is a bearing set. Each a_i could be mapped into s_i and s_i also could be mapped into a_i;

2) F={f_j | j∈J} is a operation set. Each function f_j could be mapped into o_j and o_j also could be mapped into f_j.

Definition 3[3] abstract data type[3] is 2-tuple D = <Σ, E> iff

1) Σ is a keynote;

2) E is an equation.

III. THE ALGEBRAIC SEMANTICS AT ACTIVITY LEVEL

A. Syntax

The following is the syntax of EPDL at the activity level based on Extended Backus-Naur Form (EBNF) which was defined by Li[1].

< **Activity** >::= **ACTIVITY** < Activity Name > [**FROM** [< Software Process Name >.] < Activity Name >] [**IMPORTS** < Variable Declaration List > ;] [**EXPORTS** < Variable Declaration List > ;] [**LOCALS** < Variable Declaration List > ;] **BEGIN** < Activity Body > **END**;

< **Variable Declaration List** > : : = < Variable Declaration > | < Variable Declaration > ; < Variable Declaration List >

< Variable Declaration List > is a set of variable declarations.

< **Variable Declaration** > : := <Variable List>:<Variable Type>

< Variable Declaration List > declares variables used by an activity.

< **Task List** > : : = < Task > | < Task > < Task List >

< Task List > is a set of definitions in which the tasks of the activity are defined.

< **Activity Body** > : : = < Task List > | < Software Process Name >

B. Semantics

The activity is an abstract data type which defines the data structures and the tasks. According to the syntax of the activity level, the following model was defined which is based on abstract data type.

Type ACTIVITY = {

Sort Task, Process, Message, Activity, Sub Activity, Activity Object, bool

Operation RECEIVE: Message →Activity

AEXE:Activity×Message →ActivityObject

SET: Task×Task×...×Task →Activity

REFINE: Activity →Process

INHERIT: Activity×Sub Activity →bool

Declare t: task; sp: software process which the activity is refined as; m: message; a: this activity; ao: activity object;

Axiom RECEIVE(m) = A

AEXE(m, Receive(m)) = ao or

AEXE(m, Set(t_1,t_2,…,t_n)) = ao

REFINE(A) = sp

INHERIT($\varnothing$, $Suba_1$) = False

INHERIT(a_1,$Suba_1$) = **if** $Suba_1$ inherit a1 then True **else** False **fi**

}

At activity level, the keynote Σ = <A, Ωa> was defined at first. It contains six sorts, every sort is declared in the Declare of the algebraic semantics description. In the Operation elements, “RECEIVE”, “AEXE”, “SET”, “REFINE” and “INHERIT” are used to define the rules which are in the axioms.

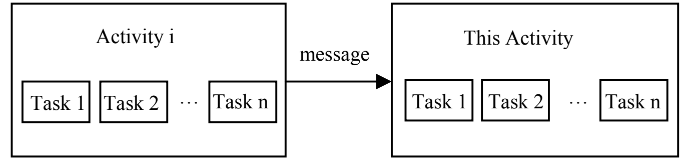

Fig. 1 Send message between activities.

In EPDL, activities are made up of a set of tasks. Therefore, the execution of activities is the execution of a series of tasks. In this paper, SET(t1,t2,⋯,tn) denotes that an activity is a set of tasks. Tasks interact with each other by sending message. The task may receive message from other tasks during the execution, as shown in Figure 4. Therefore, the Operation “RECEIVE: m →a” illustrates the operation name is Receive and indicates that this task received the message, as shown in Fig. 1.

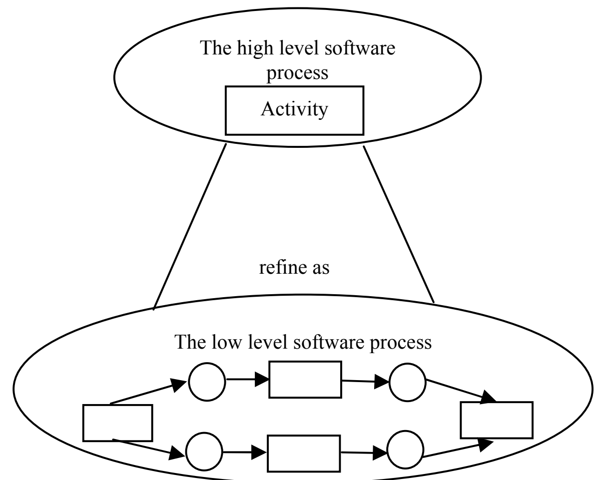

Fig. 2 The refinement of activity

© the authors

The Operation "AEXE: a×m →ao" illustrates the operation name is AEXE and denotes the execution of activity. So, in the axioms, the axiom "AEXE(m, RECEIVE(m))=ao or AEXE(m, SET(t_1,t_2,…,t_n))=ao" means that when an activity is executed, an activity object is created. And its task Main (one of the tasks in the activity) is executed firstly on receiving the message Execution. As is mentioned above, SET(t_1,t_2,…,t_n) denotes an activity is a set of tasks, thus, the execution of activity can also be regarded as AEXE(m, SET(t_1,t_2,…,t_n))=ao.

A new software process is created on a lower level if an activity is defined a software process, as shown in Fig. 2. The software process on the lower level is refined by the higher level of software process. The axiom "REFINE(A)=sp" denotes the process of refinement.

IV. HIERARCHY-CONSISTENCY AND SUFFICIENT-COMPLETENESS

So far, a type is always described as an entirety when the algebraic semantics described based on abstract data type.[3]

However, it contradicts with the modularization of program design in practical. To write a large abstract data type is easy to make mistakes. If the abstract data type can be divided into modules, it would be likely programming and also easy to keep the correctness. In the meantime, the reuse of data type module is another important principle, especially some common use basic types (bool, int etc.) are better to be written separately and will be combinatorial used. Considering the above principles, the modularization and combination of algebraic semantics should be paid more attention.

At activity level, there is only one basic type (bool) we can reuse. So we write an expansion type of the semantics described above. We extract the bool type separately in the following in order to reuse.

Type ACTIVITY = { BOOL+

Sort Task, Process, Message, Activity, SubActivity, Activity Object

Operation RECEIVE: Message →Activity
AEXE:Activity×Message →ActivityObject
SET: Task×Task×...×Task →Activity
REFINE: Activity →Process
INHERIT: Activity×SubActivity →bool

Declare t: task; sp: software process which the activity is refined as; m: message; a: this activity; ao: activity object;

Axiom RECEIVE(m) = A
AEXE(m, RECEIVE(m)) = ao or
AEXE(m, SET(t_1,t_2,…,t_n)) = ao
REFINE(A) = sp
INHERIT(∅, $Suba_1$) = False
INHERIT(a_1,$Suba_1$) = **if** $Suba_1$ inherit a1 then True
else False **fi**
INHERIT(a_1, ∅) = True
}

There are two key problems that can reflect the semantics of basic type when the expansion type is built:

1) In the basic type, two inequality basic type may become equal in the expansion type.

2) The expansion type may add basic types which are not original exist.

Definition 4 hierarchy-consistency[3]Let D_2 denote the expansion type of D_1. The expansion type D_2 is called hierarchy-consistency relative to D_1, iff:

1) $s \in S_1$. Each s is the consequence sort of basic types t and t' in D_2;

2) t=t' is provable in D_2 iff t=t' is provable in D_1.

Definition 5 Sufficient-completeness[3]Let D_2 denote the expansion type of D_1. Suppose any $s \in S_1$ and s is the consequence sort of basic types t and t' in D_2, t=t'is provable in D_2 iff t=t' is provable in D_1. The expansion type D_2 is called sufficient-completeness relative to D_1, iff:

1) $s \in S_1$. Each s is the consequence sort of a basic type t in D2;

2) t=t' is provable in D_2.

Based on the definitions above, the verification of the expansion type of hierarchy-consistency and sufficient-completeness are proved in the following.

Proposition 1 ACTIVITY is hierarchy-consistency.

PROOF. To proof the hierarchy-consistency in the axiom of ACTIVITY, the equation which the consequence sort is BOOL should deduce True≠False. As INHERIT(∅, Suba1)= False is the only equation can deduce False, therefore, we need to prove for all the activity a INHERIT(a, Suba1)=True, then, a1≠∅. According to the equation, only the axiom INHERIT(a, ∅)= True can deduct to True. But there isn't any axiom can prove a1=∅, so ACTIVITY is hierarchy-consistency.

Proposition 2 ACTIVITY is sufficient-completeness

PROOF. For the sufficient-completeness, it only involves the sixth axiom. We should prove for all a1 and Suba1 the deductive result of INHERIT(a1, Suba1) is True or False. For this axiom, there only have two results True or False. Consequently, we can deduct the result is True or False in certainly steps. Therefore, the ACTIVITY is sufficient-completeness.

V. CONCLUSIONS

It is dramatically important to describe the algebraic semantic of software evolution process description language of activity level. After the description of the semantics of EPDL in formal method, the EPDL become more accurately. Moreover, based on the formal semantic this research also verify its hierarchy-consistency and sufficient-completeness, which is contributed to the modularization and correctness of the EPDL semantics of the activity level.

However, there still remains much work. Firstly, there're four levels of EPDL. In this paper, our works mainly focus on the activity level.

There're the other levels' semantics need to be described using the method of algebraic semantics. And better to be proof like this paper. And a full algebraic semantics of EPDL (AS-EPDL) and its verification will be given at the end.

© the authors

Secondly, after AS-EPDL is given, the characteristic of AS-EPDL such as soundness and completeness will be researched.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No. 61262024 and No. 61262025; the Key Subject Foundation of School of Software of Yunnan University and the Open Foundation of Key Laboratory in Software Engineering of Yunnan Province under Grant No. 2010KS01; the Postgraduates Science Foundation of Yunnan University under Grant No. ynuy201131; Scientific Research Fund of Yunnan Provincial Department of Education under Grant 2012C108; Science Foundation of Yunnan Province under Grant No. 2012FB119; Science Foundation of Yunnan Province Education Department No. 2011Y120; Yunnan Province Science Youth Experts Fund No.2012FD005.

REFERENCES

[1] T. Li, “An Approach to Modelling Software Evolution Processes[M],” Springer-Verlag, Berlin, 2008.

[2] L.J. Osterweil, “Understanding process and the quest for deeper questions in software engineering research,” ACM SIGSOFT Software Engineering Notes 8: 6-14.

[3] R.Q. Lu, “Formal semantics of computer language, ”Science press, Peking, 1992.

[4] A. Fuggetta, “Software process: a roadmap,” Proceedings of the conference on the future of software engineering, ACM Press, New York, pp 25-34.

© the authors