Algebraic Semantics-based Verification for EPDL at Task Level

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

Wei Wang, Xuan Zhang, Ye Qian, Rui Zhu
School of Software,
Yunnan University
Key Laboratory in Software Engineering of Yunnan Province,
Kunming, 650091, China

Abstract— **The algebraic semantics has been applied to describe and verify semantics for a long period of time. In this paper, the algebraic semantics of EPDL of task level is studied. The paper is divided into two parts. In the first part, the initial algebraic semantics of EPDL (AS-EPDL) of task level is given. As a consequence of the practical purpose, we give the expansion type, which can help the modularization of programming. However, there are two key problems we should concern about when the expansion type is built - hierarchy-consistency and** sufficient-completeness**. Therefore, we give the verification of these two properties in the second part of the paper.**

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

I. INTRODUCTION

With the dramatically increase of legacy systems in the last few years, software evolution is becoming increasingly important. Software systems change trigger by the changes in techniques and requirements. Software processes is a set of interrelated software processes which can provide a framework to manage activities in software development. Software evolution process is the interdisciplinary of software process and software evolution which becomes a key area in software engineering.

A software evolution process is a set of interrelated software processes under which the corresponding software is evolving[1]. A software evolution process description language that is used to describe software evolution processes[1]. 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-the global level, the process level, the activity level, the task level. However, EPMM is hard to enact directly because EPMM is an abstract description of software evolution process model. Therefore, Li[1] designed a detailed description of a software evolution process-software evolution process description language (EPDL).

As Osterweil has pointed out Human being must employ some powerful process abstractions owing to the complexity of software process entities[2]. 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].

EPDL is an object-oriented computer language that expends the descriptive power of EPMM. As the EPDL is designed based on EPMM, the syntax of EPDL also map into four levels- the global model level, the software process level, the activity level and the task level. The structure of EPDL is exactly the same as EPMM. To describe a software process correctly EPDL needs to capture the aspects of software evolution processes. Li[1] defined the syntax of EPDL in formally with Extended Backus-Naur Form without giving a formal semantics of it. In order to make the semantics of EPDL more explicit, we give the formal semantics of EPDL of task level based on algebraic semantics in this paper. In this paper, we use the abstract data type to define the task level of EPDL. The task level describes the function and messages of a task. A task is a method (or operation) of an activity[1].

Furthermore, we give an expansion type of the original semantics we defined which can help the modularization and reuse of the algebraic semantics. As the expansion type may cause new inconformity problems, we give the verification of hierarchy-consistency and sufficient-completeness afterwards.

II. BACKGROUND INFORMATION

The definitions of algebraic semantics using in this paper are show in the following. Because of the limit of pages, some formal definitions are omitted. In this paper, we use the abstract data type to describe EPDL. The abstract data type contains sorts, operation and axiom which its formal definition listed below.

Definition 1 Keynote[3] is 2-tuple Σ=(S, O) iff

1) S=$\{s_i \mid i\in I\}$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=$\{o_j \mid j\in J\}$is a finte set. J is a finite subscript set. Each o_j is called a operation.

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

1) A=$\{a_i \mid i\in I\}$ is a bearing set. Each a_i could be mapped into s_i and s_i also could be mapped into a_i;

2) $F=\{f_j \mid j\in J\}$ is an operation set. Each function f_j could be mapped into o_j and o_j also could be mapped into f_j.

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

1) Σ is a keynote;

2) E is an equation.

III. THE ALGEBRAIC SEMANTICS AT TASK LEVEL

The algebraic semantics of the task level are defined in the following based on the abstract data type.

```
Type TASK = {
    Sort        Message, Task, TargetTask, PreC, PostC, P(x), bool
    Operation   RECEIVE: Message →Task
                TEXE: PreC×Message →PostC
                SEND: Task×TargetTask →bool
                NOT: bool →bool
                AND: bool×bool →bool
                OR: bool×bool →bool
                IMPLY: P(x)×P(x) →bool
                IFF: P(x)×P(x) →bool
                ALL: x×P(x) →bool
                EXIST: x×P(x) →bool

    Declare     m: message; t: task; td: the target task;
Exp: the predicate formula; prec: refers to precondition, a
precondition is a first-order predicate formula which defines
the state before a task is executed; postc: refers to
postcondition, a postcondition is a first-order predicate
formula which defines the state after a task is executed; P(x):
refers to a first-order predicate formula.
    Axiom       RECEIVE (m) = m
                SEND(t1,t2) = True
                TEXE(prec, RECEIVE(m)) = postc or
                TEXE(prec, m) = postc or
                TEXE(SEND(t1,t2)) = True
    NOT(exp) = if exp = True then False
               else True fi
    AND(exp,NOT(exp)) = False
    AND(exp1,exp1) = exp1
    AND(exp1,exp2) = if exp1 = True and exp2 = True
                        then True
                     else False fi
    OR (exp,NOT(exp)) = True
    OR(exp1,exp1) = exp1
    OR(exp1,exp2) = if exp1 = False and exp2 = False
                       then False
                    else True fi
    IMPLY(exp1,exp2) = if exp1⇒exp2 then True
                       else False fi
    IFF(exp1,exp2) = if exp1 ⇔exp2 then True
                     else False fi
    ALL(x,P(x)) = if (∀x)(P(x)) then True
                  else False fi
    EXIST(x,P(x)) = if(∃x)(P(x)) then True
                    else False fi
}
```

When we design the algebraic semantics, the keynote $\Sigma = <T, \Omega t>$ is always defined first. It contains seven sorts, every sort is declared in the Declare of the algebraic semantics description of the task level. “RECEIVE”, “TEXE”, “SEND”, “NOT”, “AND”, “OR”, “IMPLY”, “IFF”, “ALL” and “EXIST” are used to define the rules in the Operation elements which are used in the axioms.

The Operation “RECEIVE: Message →Task” illustrates the operation name is RECEIVE and indicates that this task received the message that was sent by other tasks. Thereby, in the axioms, the axiom “RECEIVE(m) = m” is valid according to the operation element.

The Operation “SEND: Task×TargetTask →bool” illustrates the operation name is Send and indicates that the task send message to other tasks. Hence, the axiom “SEND(m) = True” means this task send message to others and they received according to the operation element.

The Operation “TEXE: PreC×Message →PostC” illustrates the operation name is TEXE and indicates that the execution of tasks. Precondition defines the state before task t is executed and postcondition defines the state after task t is executed. And $A(F) = (\{Q_1\}, \{Q_2\})$ is called a 2-assertion, which defines the function of task t, as shown in Figure 3.3. So, in the axioms, the axiom “TEXE(prec) = postc” denotes the execution of tasks. “TEXE(prec, RECEIVE(m)) = postc” denotes that when the task is executing, it may receive messages from other tasks otherwise it cannot keep executing. And the function of some tasks is to send messages, consequently “TEXE(SEND(t_1,t_2)) = True” denotes the success of sending message.

In the task level, it uses first-order predicate formula. “NOT” denotes “¬”; “AND” denotes “∧”; “OR” denotes “∨”; “IMPLY” denotes “⇒”; “IFF” denotes “⇔”; “ALL” denotes “∀” and “EXISTS” denotes “∃”.

So, in the axioms, the axiom “NOT(exp) = if exp = True then False else True” denotes that if the expression is true NOT(exp) is false and vice versa.

The axiom “AND(exp_1,exp_2) = if exp_1 = True and exp_2 = True then True else False” denotes that if the expression$_1$ and the expression$_2$ are all true and(exp) is true and vice versa.

The axiom “OR(exp_1,exp_2) = if exp_1 = False and exp_2 = False then False else True” denotes that if the expression$_1$ and the expression$_2$ are false OR(exp_1,exp_2) is false and vice versa.

The axiom “IMPLY(exp_1,exp_2) = if $exp_1 \Rightarrow exp_2$ then True else False” denotes that if the expression$_1$ implies the expression$_2$ IMPLY(exp_1,exp_2) is true and vice versa.

The axiom “IFF(exp_1,exp_2) = if $exp_1 \Leftrightarrow exp_2$ then True else False” denotes that if the truth value of the expression$_1$ is equivalent to the truth value of the expression$_2$ IFF(exp_1,exp_2) is true and vice versa.

∀x denotes all x in the field. (∀x)(P(x)) denotes all x have the characteristic P in the field. The axiom “ALL(x, P(x)) = if (∀x)(P(x)) then True else False” denotes that if all x in the field have the characteristic P ALL(x, P(x)) is true and vice versa.

∃x denotes that the x is exist in the field. (∃x)(P(x)) denotes that x has the characteristic P in the field. The axiom

"EXIST(x, P(x)) = if(∃x)(P(x)) then True else False" denotes that if there is a x in the field which has the characteristic P EXIST(x, P(x)) is true and vice versa.

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] Nevertheless, this idea have contradicts with the modularization of program design in reality. It may easy make mistakes during writing a large abstract data type. For sake of correctness and modularization, we can divide the abstract data type into modules. In addition, the reuse of data type module is also important such as bool, int etc. They are usually written separately but used combinatorial. In consideration of the above principles, we rewrite the expansion type of the AS-EPDL at task level. We extract the bool type separately in the following to improve the modularization and correctness.

Type TASK = {BOOL+
Sort Message, Task, TargetTask, PreC, PostC, P(x)
Operation RECEIVE: Message →Task
TEXE: PreC×Message →PostC
SEND: Task×TargetTask →bool
NOT: bool →bool
AND: bool×bool →bool
OR: bool×bool →bool
IMPLY: P(x)×P(x) →bool
IFF: P(x)×P(x) →bool
ALL: x×P(x) →bool
EXIST: x×P(x) →bool
Declare m: message; t: task; td: the target task; Exp: the predicate formula; prec: refers to precondition, a precondition is a first-order predicate formula which defines the state before a task is executed; postc: refers to postcondition, a postcondition is a first-order predicate formula which defines the state after a task is executed; P(x): refers to a first-order predicate formula.
Axiom RECEIVE(m) = m
$SEND(t_1,t_2)$ = True
TEXE(prec, RECEIVE(m)) = postc or
TEXE(prec, m) = postc or
$TEXE(SEND(t_1,t_2))$ = True
NOT(exp) = **if** exp = True then False
else True **fi**
AND(exp, NOT(exp)) = False
$AND(exp_1,exp_1) = exp_1$
$AND(exp_1,exp_2)$ = **if** exp_1 = True and exp_2 = True then True
else False **fi**
OR (exp, NOT(exp)) = True
$OR(exp_1,exp_1) = exp_1$
$OR(exp_1,exp_2)$ = **if** exp_1 = False and exp_2 = False then False
else True **fi**
$IMPLY(exp_1,exp_2)$ = **if** $exp_1 \Rightarrow exp_2$ then True
else False **fi**
$IFF(exp_1,exp_2)$ = **if** $exp_1 \Leftrightarrow exp_2$ then True
else False **fi**
ALL(x, P(x)) = **if** (∀x)(P(x)) then True
else False **fi**
EXIST(x, P(x)) = **if**(∃x)(P(x)) then True
else False **fi**
}

After the expansion type of original semantics is given, two problems are coming up:

1) Whether the two basic type in the original semantics which are not equal become equally in the expansion type;

2) During the expansion, it may add new basic types which are not original exist.

According to the concern above, in the algebraic semantics there are two properties which can check these two problems.

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 D_2;

2) t=t' is provable in D_2.

Proposition 1 TASK is hierarchy-consistency.

PROOF. As we only extract the basic type Bool, to proof the hierarchy-consistency in the axiom of TASK, the equation which the consequence sort is BOOL should deduce True≠False. As AND(exp, NOT(exp))= False is the only equation can deduce False, therefore, we need to prove for all the AND(exp, exp)=TRUE, exp≠NOT(exp). According to the axioms, there aren't any axiom can deduct to exp=NOT(exp). Hence, TASK is hierarchy-consistency.

Proposition 2 TASK is sufficient-completeness.

PROOF. For the sufficient-completeness, it involves the following axioms: NOT(exp) , AND(exp1, exp2), OR(exp1, exp2), IMPLY(exp1, exp2), IFF(exp1, exp2), ALL(x, P(x)), EXIST(x, P(x)). We should prove for all exp, P(x) the deductive result of these equations is True or False. For these axioms, they only have two results True or False. Consequently, we can deduct the result is True or False in certainly steps. Therefore, the TASK is sufficient-completeness.

V. CONCLUSIONS

It is important to describe the semantics of EPDL with formal semantics. In this paper, we not only define the algebraic semantics of EPDL at task level, but also expand the expansion type of it, which can improve the normalization, modularization. Furthermore, we verify the expansion type of its hierarchy-consistency and sufficient-completeness properties.

However, there still remains much work. Firstly, there're four levels of EPDL. In this paper, our works mainly focus on the task level which can not reflect the entire formal semantics of EPDL. Secondly, after the whole 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; Science Foundation of Yunnan Province Education Department No. 2011Y121; Science Foundation of Yunnan Province under Grant No. 2012FB118 and No. 2012FB119; Science Foundation of Yunnan Province Education Department No. 2011Y120.

REFERENCES

[1] T. Li, "An Approach to Modelling Software Evolution Processes" 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.