# Composing Software Evolution Process Component*

Fei Dai and Tong Li

School of Information Science and Engineering, Yunnan University, Kunming 650091, China
flydai.cn@gmail.com, tli@ynu.edu.cn

**Abstract.** Composing software evolution process components into a complete software evolution process can effectively improve quality and efficiency of the software evolution process. However, existing researches do not propose a systematic method for composing software evolution process components. We propose a software evolution process component model (EPCM) which is based on 3C model and the concept of a software evolution process component (EPC). Based on EPCs, we propose three types of software evolution process component composition operations, namely, sequence composition, selection composition and concurrence composition.

**Keywords:** Petri Net, Component Model, Software Evolution Process, Component Composition, Process Reuse.

## 1 Introduction

As more and more successful software systems become legacy systems, software evolution becomes more and more important. On the one hand, software evolution has become an important characteristic in the software life cycle. On the other hand, software process plays an important role to increase efficiency and quality of software evolution. The term software evolution process denotes a set of interrelated software processes under which the corresponding software is evolving. A software evolution process provides a framework for managing activities that can very easily get out of control in software evolution, so we use software evolution processes to improve the effectiveness and efficiency of software evolution. Li [1] defined a formal evolution process meta-model (EPMM) based on extended Petri Net which is added with object-oriented technology and Hoare Logic to construct software evolution process models with four-level architecture. However, as more and more software evolution process modes are constructed by process designers based on EPMM [1], how we can reuse these existing software evolution models poses a challenging and exciting question for us.

* The project is supported by National Natural Science Foundation of China under Grant No. 60463002 and Education Science Foundation of Yunnan Province, China under Grant No. 04Z290D.

The term software evolution process reuse can be described as “usage of one process description in the creation of another process description” [10]. Osterweil presented a widely accepted view that software processes are software too [2]. According to Osterweil’s idea, a software evolution process can be made up of many serial or parallel software evolution process components. Thus we apply component technology to software evolution processes and propose the concept of a software evolution process component (EPC). An EPC is actually an internally high cohesive and consistent software evolution process that can be reused with other EPCs to assemble a more powerful EPC. In order to describe an EPC, a software evolution process component model (EPCM) based on 3C model is proposed. Comparing to traditional software reuse, four essential steps for software evolution process reuse based on EPCs are needed. Firstly, we need to describe an EPC. Secondly, we need to search EPCs from software evolution process component library (EPCL) according to process requirements. Thirdly, we need a mechanism to tailor EPCs. Fourthly, we need a mechanism to compose EPCs into a software evolution process. In this paper, composing software evolution process component is focused on.

This paper is organized as follows. In the next section, a software evolution process models is proposed. In section 3, we propose three types of EPC composition operations, namely sequence composition, selection composition and concurrence composition. Finally, we conclude in section 4 with a brief summary and discussion of the future work.

## 2 Evolution Process Component Model

EPCM is the foundation of EPCL and is the key factor in realizing software evolution process reuse. Recently, the component models can be classified into three different categories according to their usage: (1) Model for component description/classification, such as REBOOT model [5]; (2) Model for component specification/composition, such as 3C model [4] and JBCOM [6]; (3) Model for component implementation, such as COM/DCOM [7][8], CORBA/OM [3], and Enterprise JavaBeans [9].

Obviously, it is difficult for us to build a comprehensive model to meet all software evolution processes defined by other process description languages. Thus we propose a component model to only meet the need of evolution process description language (EPDL) [1]. 3C model is a prescriptive component model that was proposed by Will Tracz on the “Reuse in Practice Workshop” in 1989. In 3C model, a component consists of three parts: concept, content, and context [4]. The concept is the abstract description of what a component does. The content describes how a component implements the concept. The context depicts the dependencies between the component and its environment. Based on 3C model, we propose EPCM which is shown in Figure 1.

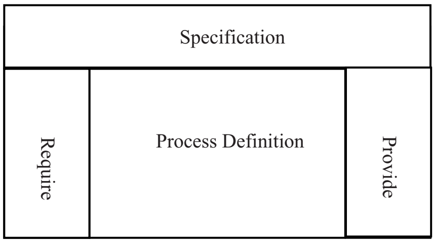

**Fig. 1.** Evolution Process Component Model

EPCM is a formal evolution process component model. The definition of EPCM is as follows:

Definition 1 EPCM is a 4-tuple epcm= (Req, Pro, Spec, PD)

1. Req and Pro are called the interfaces of EPC. Req denotes the required functions of EPC; Pro denotes the provided functions of EPC. They correspond to the concept in 3C model.
2. PD (Process Definition) is called the body of EPC, which is defined by EPDL [1]. It corresponds to the content in 3C model.
3. Specification is called the specification of EPC, which is used to describe the EPC briefly; it corresponds to the context in 3C model.
   According to EPCM, the definition of EPC is as follows:
   Definition 2 A EPC is a 7-tuple epc = (C, A; F, $M_0$, $a_e$, $a_x$ , S,)

1. (C, A; F) is a net without isolated elements, $A \cup C \neq \Phi$ ;
2. C is a finite set of conditions; $\forall c \in C$ is called a condition;
3. A is a finite set of activities; $\forall a \in A$ is called an activity;
4. p=(C, A; F, $M_0$), called the body of epc, is a software evolution process with $M_0 = \Phi$ ;
5. $a_e$, $a_x \in$ A are called the entrance and the exit of EPC respectively, if $\exists$ step sequence $G_1G_2 \ldots G_{n-1}$ ($G_1$, $G_2$, …, $G_{n-1} \subseteq A$) and $\exists$ cases $M_1$, $M_2$, …, $M_n \subseteq C$, such that $[a_e > M_1$, $M_1[G_1 > M_2$, …, $M_{n-1}[G_{n-1} > M_n$, $M_n[a_x >$ and $(M_n - a_x) = \Phi$ ;
6. S, called the mini specification, is a set of strings which is used to describe the epc briefly;

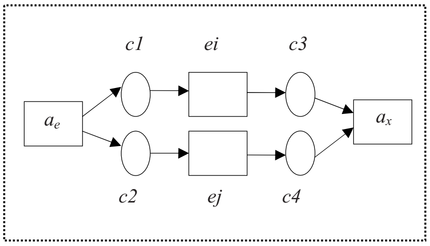

**Fig. 2.** An Evolution Process Component

From the definition above, we see that p corresponds to the PD in EPCM; $a_e$.I corresponds to the Req in EPCM; $a_x$.O corresponds to the Pro in EPCM; S corresponds to the Spec in EPCM. Here $a_e$.I denotes the input data structure of $a_e$, $a_x$.O denotes the output data structure of $a_x$ [1]. Graphically, we represent activity as rectangle and condition as circle respectively. An EPC is shown in Figure 2.

The definition of EPC shown in Figure 2 is as follows:

epc = (C, A; F, $M_0$, $a_e$, $a_x$ , S );

C = (c1, c2, c3, c4);

A = (ei, ej);

F=( ($a_e$,c1), ($a_e$,c2), (c1,ei), (c2,ej), (ei,c3), (ej,c4), (c3, $a_x$), (c4, $a_x$));

$M_0$ = $\Phi$ ;

$a_e$= $a_e$;

$a_x$ = $a_x$;

S=(…..);

The description of EPC shown in Figure 2 is defined by EPDL [1] as follows:

PROCESS An Evolution Process Component

Begin

ENTRANCE { $a_e$ }

EXIT { $a_x$ }

MINI SPECIFICATION

S={....};

CONDITION SET

C:={c1, c2, c3, c4};

ACTIVITY SET

A:={ $a_e$, ei, ej, $a_x$ };

ARC SET

F:={( $a_e$,c1), ($a_e$,c2), (c1,ei), (c2,ej), (ei,c3), (ej,c4), (c3, $a_x$), (c4, $a_x$)};

MARKING { $\Phi$ } ;

END;

## 3 A Systematic Method for EPC Composition

EPC composition is defined as composing EPCs into a complete software evolution process. After process designers find out the required EPCs from EPCL, EPC composition is the next step. In this paper, we define three types of EPC composition operations, namely sequence composition, selection composition, and concurrence composition. In the following, we will compose R and S into T using these three composition operations. R and C are EPCs as shown in Figure 3.

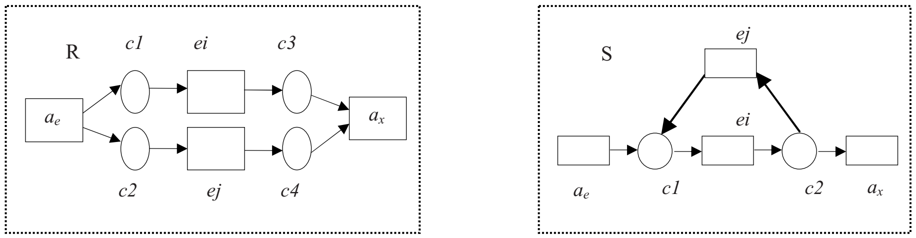

**Fig. 3.** Evolution Process Component R and C

### 3.1 Sequence Composition

Sequence composition is defined as composing R and S into T and T's execution sequence is that after R terminates, S then executes. During sequence composition, process designers should avoid interface conflict. The term interface conflict means that the interfaces between EPCs are mismatched. The following conditions are used for interface checking. By interface checking, process designers can check whether all the interfaces among EPCs are matched. If EPCs satisfy the following conditions, they are considered to be interface conformance. If EPCs are interface conformance, they can be composed into a more powerful EPC. The following algorithm 1 is used for sequence composition and Figure 4 shows the process of sequence composition.

The conditions are as follows:

-R.$a_e$ = S.$a_x$ ;
-R.$a_e$ is a part of S.$a_x$

```
Algorithm 1 Fun SequenceComposing(R, S)
//Supposing that R.ax and S.ae are interface conformance
Begin
 // New(c) denotes the added conditions.
T.C = R.C + S.C + New(C);
T.A = R.A + S.A;
// New(F) denotes the added flow relations.
```

```
T.F = R.F + S.F + New(F);
T.M_0 = R.M_0 + S.M_0;
T.a_e = R.a_e;
T.a_x = S.a_x;
T.S =R.S ∪ S.S;
return T;
End;
```

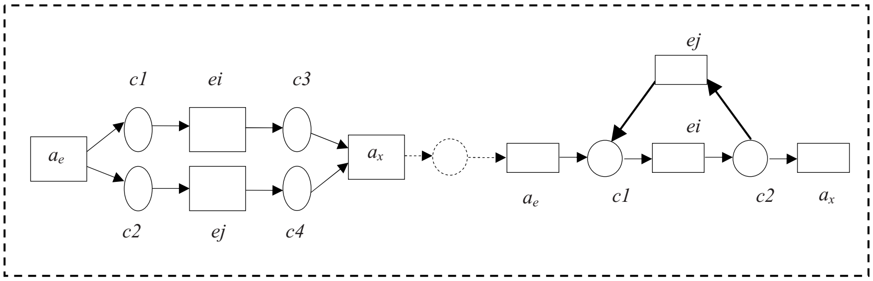

**Fig. 4.** Sequence Composition

## 3.2 Selection Composition

Selection composition is defined as composing R and S into T and T's execution sequence is that only R or S can execute according to process requirements. The following algorithm 2 is used for selection composition and Figure 5shows the process of selection composition.

```
Algorithm 2 Fun SelectionComposing(R, S)
Begin
T.C = R.C + S.C + c5 + c6;
T.A = R.A + S.A + a1 + a2;
// New(F) denotes the added flow relations.
T.F = R.F + S.F + New(F);
T.M_0 = R.M_0 + S.M_0;
T.a_e = a1;
T.a_x = a2;
```

```
a1.I = R.$a_e$.I + S.$a_e$.I;
a1.O= R.$a_e$.O + S.$a_e$.O;
a1.L= R.$a_e$.L + S.$a_e$.L;
a2.I= R.$a_x$.I + S.$a_x$.I;
a2.O= R.$a_x$.O + S.$a_x$.O;
a2.L= R.$a_x$.L + S.$a_x$.L;
T.S =R.S $\cup$ S.S;
return T ;
End;
```

Algorithm 2 will introduce new activities and new conditions when running selection composition. It is necessary to notice that the newly added activities are virtual activities and the newly added conditions are virtual conditions. These activities have no actual operations in T except for passing a token from a condition to another condition. These conditions have no other meanings in T except for connections.

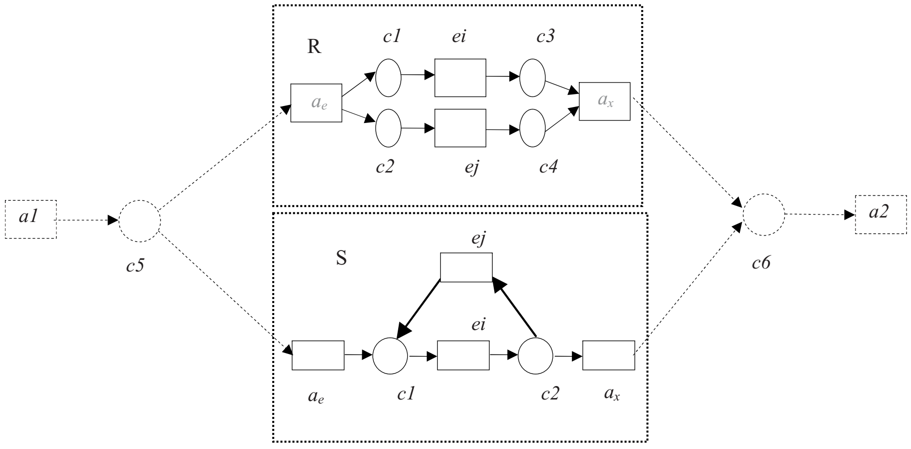

**Fig. 5.** Selection Composition

## 3.3 Concurrence Composition

Concurrence composition is defined as composing R and S into T and T's execution is that R and S can execute concurrently. The following algorithm 3 is used for concurrence composition and Figure 6 shows the process of concurrence composition.

```
Algorithm 3 Fun ConcurrenceComposing(R, S)
Begin
T.C = R.C + S.C + c5 + c6 + c7 + c8;
T.A = R.A + S.A + a1 + a2;
// New(F) denotes the added flow relations.
T.F = R.F + S.F + New(F);
T.$M_0$ = R.$M_0$+ S.$M_0$;
T.$a_e$ = a1;
T.$a_x$ = a2;
a1.I = R.$a_e$.I + S.$a_e$.I;
a1.O= R.$a_e$.O + S.$a_e$.O;
a1.L= R.$a_e$.L + S.$a_e$.L;
a2.I= R.$a_x$.I + S.$a_x$.I;
a2.O= R.$a_x$.O + S.$a_x$.O;
a2.L= R.$a_x$.L + S.$a_x$.L;
T.S =R.S $\cup$ S.S;
return T;
End;
```

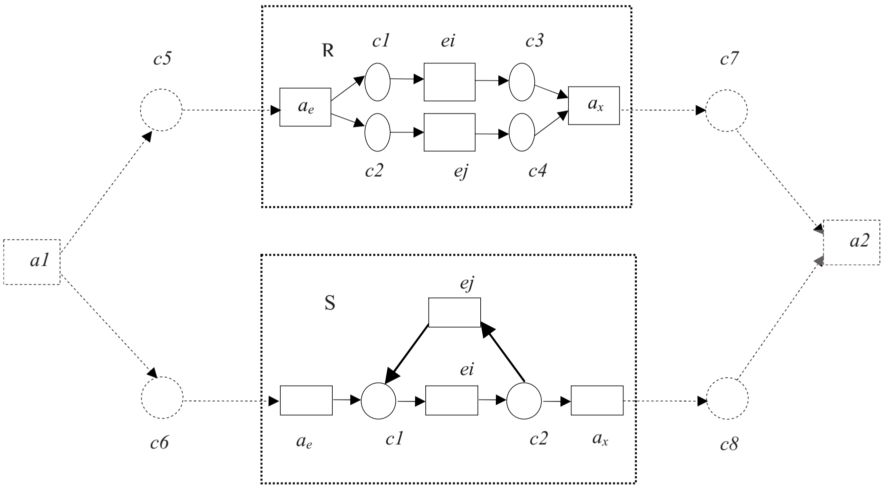

**Fig. 6.** Concurrence Composition

## 4 Conclusions

In this paper, the main idea is to compose EPCs into a complete software evolution process. In order to achieve the goal, we firstly propose an evolution process component model and define an evolution process component. Based on evolution process components, we propose three types of software evolution process component operations, namely, sequence composition, selection composition and concurrence composition.

However, there still remains much work. Firstly, a process operating system which is used to mange software evolution processes will be investigated. Secondly, after composing or tailoring EPCs, software evolution process's simulation will be researched.

## References

1. Li, T.: Modeling formal software evolution process [Ph.D Thesis]. DeMontfort University (2007)
2. Osterweil, L.J.: Software Processes are Software too. In: Proceedings of the 9th International Conference on Software Engineering, pp. 2–13. ACM Press, New York (1987)
3. Object Management Group home page [online].Available WWW URL, http://www.omg.org
4. Implementation Working Group Summary, Reuse in Practice Workshop. Pittsburgh, Pensylvania (July 1989)
5. Weighted Term Spaces for Related Search. In: CIKM 1992. proceedings of the 1st International Conference on Information and Knowledge Management, pp.5–8 (November 1992)
6. JadeBird Project Group, JadeBird Component Model, Technical report, Department of Computer Science and Technology, Peking University (1997)
7. Microsoft Corporation. The Component Object Model Specification, Version 0.9, (October 24, 1995) Available WWW URL: http://www.microsoft.com/oledev/
8. Microsoft Corporation. Distributed Component Object Model Protocol COM/1.0, draft (November 1996) Available WWW. URL: http://www.microsoft.com/oledev/
9. Sun Microsystems, Inc., Enterprise JavaBeans Specifications Version 1.1, Available WWW. URL: http://java.sun.com/products/ejb/docs.html
10. Hollenbach, C., Frakes, W.: Software Process Reuse in an Industrial Setting, Fourth International Conference on Software Reuse, Orlando, FL, IEEE Computer Society Press, Los Alamitos, CA, pp. 22-30 (1996)