Approach to Modeling Components in Software Architecture

Yong Yu[1,2]
1 School of Software, Yunnan University, Kunming, China
2 Key Laboratory in Software Engineering of Yunnan Province, Kunming, China
Email: yuy1219@163.com

Tong Li[1,2], Qing Liu[1,2] and Fei Dai[1]
1 School of Software, Yunnan University, Kunming, China
2 Key Laboratory in Software Engineering of Yunnan Province, Kunming, China
Email: {tli, liuqing}@ynu.edu.cn, flydai.cn@gmail.com

***Abstract*—Software components are increasingly central to efficient, cost-effective software development. Components are the special status of the software system, so the formal description of the components is very important. First, the concept and characteristics of components are given. Second, the definition of OR-transition Colored Petri Net is given. Third, in according to the properties of software components, a formal definition of component is presented. And based on OR-transition Colored Petri Net, an approach is put forward to modeling the software components formally. Finally, an example is given.**

***Index Terms*—component, Petri net, modeling, software architecture**

I. INTRODUCTION

Component-based software engineering (CBSE) is a branch of software engineering which emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system.

In the Dictionary of Object Technology [1] a component is described very generally as a "reusable entity". Nierstrasz [2] and Sametinger [3] also provide general definitions of components that include mixins, macros, functions, templates, modules, etc. as valid examples of software components.

A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.

Software components are increasingly central to efficient, cost-effective software development.

Large complex software systems are composed of many software components. Building software systems from reusable software components has long been a goal of software engineers. While other engineering disciplines successfully apply the reusable component approach to build physical systems, it has proven more difficult to apply in software engineering. A primary reason for this difficulty is that distinct software components tend to be more tightly coupled with each other than most physical components [4].

A component is simply a data capsule. Thus information hiding becomes the core construction principle underlying components. A component can be implemented in (almost) any language, not only in any module-oriented and object-oriented languages but even in conventional languages [5,6].

For component, it is unavoidable to interact with the environment. The component can be obtained information from the environment, and provides relaxed services to the environment. Component interface represents the interaction with the outside world, each interface show interaction between components and connector [5,6].

Components are the special status of the software system, so the formal description of the components is very important.

The remainder of this article is organized as follows: Section II resumes the related concepts and characteristics of our component. Section III presents concept of Or-transition colored Petri net. Section IV presents an approach to modeling software components. Section V gives an example.

II. THE CONCEPT AND CHARACTERISTICS OF COMPONENTS

An individual component is a software package, a web service, or a module that encapsulates a set of related functions (or data).All system processes are placed into separate components so that all of the data and functions inside each component are semantically related. Because of this principle, it is often said that components are modular and cohesive.

With regard to system-wide co-ordination, components communicate with each other via interfaces. When a component offers services to the rest of the system, it adopts a provided interface which specifies the services that other components can utilize, and how they can do so. This interface can be seen as a signature of the component - the client does not need to know about the

doi:10.4304/jsw.6.11.2196-2200

inner workings of the component (implementation) in order to make use of it. This principle results in components referred to as encapsulated.

However when a component needs to use another component in order to function, it adopts a used interface which specifies the services that it needs.

Another important attribute of components is that they are substitutable, so that a component can replace another, if the successor component meets the requirements of the initial component. Consequently, components can be replaced with either an updated version or an alternative without breaking the system in which the component operates.

So, a software component is simply a data capsule. Thus information hiding becomes the core construction principle underlying components. A component can be implemented in (almost) any language, not only in any module-oriented and object-oriented languages but even in conventional languages [8].

III. EXTENDED PETRI-NET

As a graphical tool, Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems.

Definition 1 [11] A triple N=(P, T; F) is called a Petri-net iff

1) P and T are disjoint sets;

$F\subseteq(P\times T)\cup(T\times P)$ is a binary relation, the flow relation of N.

Definition 2 [11] Let N=(P, T; F) be a Petri-net.

For $x\in N$,

1) $\bullet x=\{y \mid yFx\}$ is called the preset of x;

2) $x\bullet=\{y \mid xFy\}$ is called the postset of x.

An OR-transition colored Petri net can be defined as follows [7]:

Definition 3 An OR-transition Petri net system is a 4-tuple $\sum=<P, T, F, M_0>$, where:

1) $P\cup T\neq\Phi$ and $P\cap T=\Phi$;

2) $F\subseteq(P\times T)\cup(T\times P)$;

3) $M_0\subseteq P$ is the initial mark of the OR-transition Petri-net system;

4) A transition $t\in T$ is enabled in a marking M iff

$\exists p\in {}^{\cdot}t$, M (p)=1 and $\forall p'\in t^{\cdot}$, M(p′)=0.

It is said that the transition t is enabled under the mark M and the place p.

Let a transition $t\in T$ fires under a mark M and a place p, the mark M is transformed into the mark M'; we often say that the mark M' is reachable from the mark M in a step. M′ is the successor mark of M under t and p. It is written as M(p)[t>M'. where: $\forall p'\in P$:

$$M'(p')=\begin{cases} M(p')-1, & p'=p; \\ M(p')+1, & p'\neq p \text{ and } p'\in t^{\cdot}; \\ M(p'), & \text{else.} \end{cases}$$

Definition 4 In an OR-transition Petri net system $\sum=<P, T, F, M_0>$, the corresponding underlying net $N=<P, T, F>$ is called as OR-transition Petri net.

Definition 5 In an OR-transition Petri net ORPN=<P, T, F>, let $x, y\in T\cup P$, $\exists b_1, b_2, b_3, \ldots, b_k\in T\cup P$, such that $<x, b_1>, <b_1, b_2>, <b_2, b_3>, \ldots, <b_k, y>\in F$, then we say that y is structure-reachable from x, which is denoted as $xF^{*}y$.

Definition 6 1) S is a limited and non-empty type set, also known as the color set;

2) The multi-set m is a function of non-empty color set S: $m\in(S\to N)$.

For the non-empty set S, $m=\sum_{s\in S} m(s)s$ is the multi-set of S, $m(s)\geq 0$ is called the coefficient of s.

3) Let S_{MS} be the set of all multi-sets of based on S, and m, $m_1, m_2\in S_{MS}$, $n\in N$ then:

(1) $m_1+m_2=\sum_{s\in S}(m_1(s)+m_2(s))s$;

(2) $n\times m=\sum_{s\in S}(n\times m(s))s$;

(3) $m_1\neq m_2\equiv\exists s\in S: m_1(s)\neq m_2(s)$;

(4) $m_1\leq m_2\equiv\forall s\in S: m_1(s)\leq m_2(s)$;

(5) $m_1\geq m_2\equiv\forall s\in S: m_1(s)\geq m_2(s)$;

(6) If $m_1\leq m_2$, $m_2-m_1=\sum_{s\in S}(m_2(s)-m_1(s))s$.

Definition 7 An Or-transition colored Petri net (ORCPN) is a 7-tuple $N=<P, T, F, S, A_P, A_T, A_F>$, where:

1) <P, T, F> is an OR-transition Petri-net, which is called as the underlying net of ORCPN;

2) S is a non-empty color set, which is called as color set of ORCPN;

3) A_P: $P\to S_S$, A_P is a function of P, where S_S is the power set of S.

4) A_T: $T\to S_{MS}$, A_T is a guard function of T, where S_{MS} is the set of all multi-sets of based on S and it meets the following condition: $\forall t\in T$, $A_T(t)\in(\bigcup_{p\in\bullet t} A_P(p))_{MS}$.

5) A_F:$F\to S_{MS}$, A_F is the arc expression function, where S_{MS} is the set of all multi-sets of based on S and meet the following condition:

$\forall f\in F$, $A_F(f)\in(A_P(P(f)))_{MS}$, where P(f) describes the corresponding place p of arc f.

Definition 8 An OR-transition colored Petri net (ORCPN) system is an 8-tuple ORCPN system $\sum=<P, T, F, S, A_P, A_T, A_F, M_0>$, where:

1) $N=<P, T, F, S, A_P, A_T, A_F>$ is an OR-transition colored Petri net, which is called as the underlying net of ORCPN system;

2) M_0 is the initial marking of ORCPN system $\sum=<P, T, F, S, A_P, A_T, A_F, M_0>$, and meets the following condition:

$\forall p\in P$: $M_0(p)\in(A_P(p))_{MS}$.

Definition 9 M: $P\to S_{MS}$ is the marking of ORCPN system $\sum=<P, T, F, S, A_P, A_T, A_F, M_0>$, where $\forall p\in P$: $M(p)\in(A_P(p))_{MS}$.

IV. APPROACH TO MODELING SOFTWARE COMPONENTS

A. The Related Definitions of Components

In software architecture, a component should include two parts: interface and implementation. Interface defines the functions and specifications provided by the component, and implementation includes a series of related operations [9].Therefore, in this paper, the definition of components is as follows:

Definition 10 A component is a 3-tuple C = <Interface, Imp, Spec>, where:

1) Interface is a set of component interfaces. Interface = IP ⋃ OP, IP represents the input interfaces of component, OP represents the output interfaces;

2) Imp is the implementation of component, and it includes a series of operations: t_1, t_2, ..., t_n; and each operation completes specific function;

3) Spec represents the internal specification of component, and it is mainly used to describe the relationships between the implementations and the interfaces.

Definition 11 Each interface in component is a 2-tuple: p = <ID, DataType>, where ID is the unique identifier of the interface p, DataType is the type of the information which can be accepted by the interface p.

In component, each input interface represents the certain set of some operations, a component can have some input interfaces, the outside environment can request services from one or more input interfaces of the component. The output interfaces of component describe the requests of the outside environment, when the component completes a function, it may need other components to provide some help.

In component, the operation is complete certain function, and it can be defined as:

Definition 12 An operation t is a 5-tuple t = <S, D, R, PR(X), PO(X,Y)>, where:

S is the syntax of the operation t, and Y = t(X). X is the input vectors of the operation t, and Y is the output vectors of the operation t; X = (x_1, x_2, ..., x_m), Y = (y_1, y_2, ..., y_n). D = $D_1 \times D_2 \times ... \times D_m$ is the domain of the input vectors, $x_i \in D_i$ ($1 \le i \le m$). R = $R_1 \times R_2 \times ... \times R_n$ is the range of the output vector, $y_j \in R_j$ ($1 \le j \le n$). D_i, R_j is a legal data type. PR (X) is called pre-assertion; PO (X, Y) is called post-assertion. Satisfy the PR(X) of the input vector X is called the legitimate input. For legal input X, to meet the PO (X, Y) as the legitimate output of the output vector Y [10].

From the definition, the implementation of the operation t needs certain conditions, when the conditions are met, the related operations are implemented.

B. Modeling the Components

The transitions in an OR-transition colored Petri net can be used to model the operations of the components defined above, therefore, it is straight-forward to map a software component into an OR-Transition colored Petri net. In components, the operations are modeled by transitions of the OR-Transition colored Petri net and the states of the software components are modeled by places of the OR-Transition colored Petri net. The arrows between places and transitions are used to specify causal relations in the software components. And based-on ORCPN, we can present component-net (CN for short) to model software component.

Definition 13 A component-net C-net (CN) is a 9-tuple, CN = <P, T, F, S, A_P, A_T, A_F, IP, OP>, it is extended from a colored Petri net, where:

1) ORCPN = <P, T, F, S, A_P, A_T, A_F> is a or-transition colored Petri net;

2) P is a finite set of places, and it presents the states of component;

3) T is a finite set of transitions, and it presents the operations of component;

4) $F \subseteq P \times T \bigcup T \times P$ is an arc set, it describes the constraint relations between states and operations in the component;

5) S is a non-empty finite set; it describes the data types of component C;

6) $A_T(t)$ presents input vectors, which can be accepted by operation t;

7) IP, OP($\subseteq$P) are called the input interfaces and output interfaces of the component, and $\forall ip \in$ IP, $ip^{\cdot}=\varnothing$; $\forall op \in$ OP, $op^{\cdot}=\varnothing$.

In a component net CN = <P, T, F, S, A_P, A_T, A_F, IP, OP>:

The places P\(IP⋃OP) present the internal states of a component;

Transitions T present the various operations of component;

Color-set S presents the data types of a component.

The dynamic characteristics of a software component can be described by the component system defined as follows:

Definition 14 A component system (CS for short) is a 10-tuple, CS=<P, T, F, S, A_P, A_T, A_F, IP, OP, M>, where:

1) CN = <P, T, F, S, A_P, A_T, A_F, IP, OP> is a component net, called the base net of the component system;

2) M: P→S_{MS} is a marks set of a component system and it meet the following relationship:

a) $\forall p \in P$: $M(p) \in (AP(p))_{MS}$;

b) M_0 is the initial mark of a component system.

V. AN EXAMPLE

A component CN = <P, T, F, S, A_P, A_T, A_F, IP, OP>, where:

P = {p_1, p_2, p_3, ip_1, ip_2, ip_3, op_1, op_2, op_3};

T = {t_1, t_2, t_3, t_4, t_5};

F ={<ip_1, t_5>, <ip_2, t_1>, <ip_3, t_2>, <p_1, t_1>, <p_1, t_2>, <p_2, t_3>, <p_3, t_4>, <t_5, op_1>, <t_1, p_2>, <t_2, p_3>, <t_3, p_1>, <t_3, op_2>, <t_4, p_1>, <t_4, op_3>};

S ={a_1, a_2, a_3, a_4, b_1, b_2, b_6, b_7, d_2, f_2, u_2};

A_P: {$A_P(ip_1)$ = {a_1, b_1}, $A_P(ip_2)$ = a_3, $A_P(ip_3)$ = b_6, $A_P(p_1)$ = f_2, $A_P(p_2)$ = d_2, $A_P(p_3)$ = u_2, $A_P(op_1)$ = {a_2, b_2}, $A_P(op_2)$ = a_4, $A_P(op_3)$ = b_1};

A_T: {$A_T(t_1) = a_3+f_2$, $A_T(t_2) = b_6+f_2$, $A_T(t_3) = d_2$, $A_T(t_4) = u_2$, $A_T(t_5) = a_1+b_1$};

A_F: {$A_F(<ip_1, t_5>) = a_1+b_1$; $A_F(<ip_2, t_1>) = a_3$; $A_F(<ip_3, t_2>) = b_6$; $A_F(<p_1, t_1>) = f_2$; $A_F(<p_1, t_2>) = f_2$; $A_F(<p_2, t_3>) = d_2$; $A_F(<p_3, t_4>) = u_2$; $A_F(<t_5, op_1>) = a_2+b_2$; $A_F(<t_1, p_2>) = d_2$; $A_F(<t_2, p_3>) = u_2$; $A_F(<t_3, p_1>) = f_2$; $A_F(<t_3, op_2>) = a_4$; $A_F(<t_4, p_1>) = f_2$; $A_F(<t_4, op_3>) = b_1$};

IP = {ip_1, ip_2, ip_3};

OP = {op_1, op_2, op_3 }.

An example of a component shown in Figure 1:

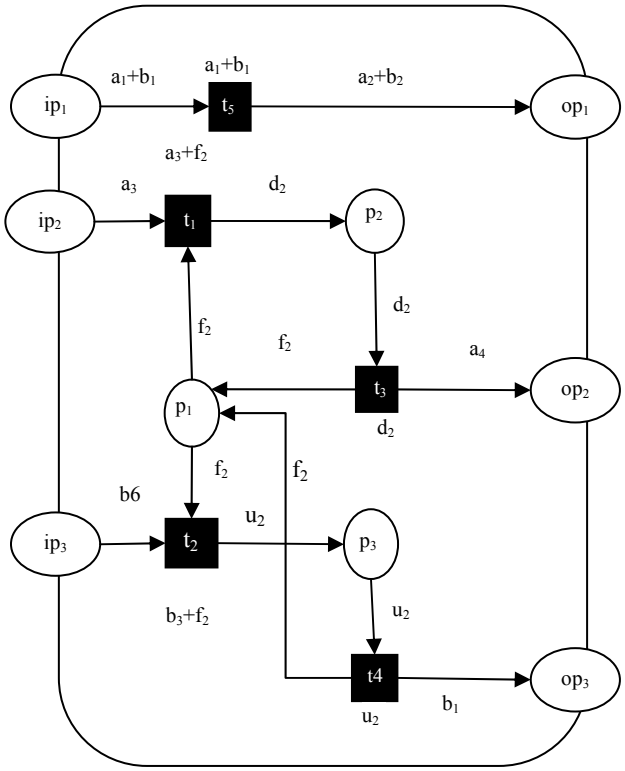

Figure 1. An example of a component

V. CONCLUSION

Component-based software engineering (CBSE) is an important branch of software engineering. In component-based software engineering, components are the special status of the software system. Large complex software systems are composed of many software components. Building software systems from reusable software components has long been a goal of software engineers.

In order to describe the software components effectively, an approach is presented to model components. First, the concept and characteristics of components are given. Second, the definition of OR-transition Colored Petri Net is given. In according to the properties of software components and OR-transition Colored Petri Net, The transitions in an OR-transition colored Petri net can be used to model the operations of the components defined above, therefore, it is straight-forward to map a software component into an OR-Transition colored Petri net. So based on OR-transition Colored Petri Net, an approach is put forward to modeling the software components formally.

ACKNOWLEDGMENT

This work has been supported by the National Science Foundation of China under Grant No. 60963007, by the Science Foundation of Yunnan Province, China under Grant No. 2007F008M, 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 promotion program for youth key teachers of Yunnan university No.21132014, by the Science Foundation of Yunnan Province Education Department No. 09J0037 and Yunnan University, China under Grant No. ynuy200920.

REFERENCES

[1] Firesmith D., Eykholt E, *Dictionary of Object Technology, SIGS Reference Library*, 1995.

[2] Nierstrasz O., Dami L, *Component-Oriented Software Technology*, Object-Oriented Software Composition, Prentice Hall, 1995.

[3] Sametinger J, *Software Engineering with Reusable Component*", Springer-Verlag, 1997.

[4] B. W. Weide and J. E. Hollingsworth, ``Scalability of reuse technology to larege systems requires local certifiability,'' in Proceedings of the Fifth Annual Workshop on Software Reuse , 1992.

[5] Talor R N, Medvidovic N, Anderson K M etal, "A component- and message-based architectural style for GUI software", IEEE Transactions on Software Engineering, 1996, 22(6):390-406.

[6] Shaw M, Garlan D, *Software architecture: Perspectives on an emerging discipline*. Prentice Hall，Inc.，Simon & Schuster Beijing Office, Tsinghua University Press，1996.

[7] Yong Yu, Tong Li, Qing Liu, Fei Dai, Na Zhao, "OR-Transition Colored Petri Net and its Application in Modeling Software System", Proceedings of 2009 International Workshop on Knowledge Discovery and Data Mining. January 2009, Moscow, Russia, 15-18

[8] Wang Zhi jian, Fei Yu kuai, Lou Yuan qing, *The technology and application of software component*, Beijing: Science Press, 2005.

[9] Clements P C, Weiderman N, *Report on the 2nd international workshop on development and evolution of software architectures for Product families*, Technique Report, CMU/SEI-98-SR-003, Carnegie Mellon University, 1998.

[10] Tong Li, *An Approach to Modelling Software Evolution Processes*, Springer-Verlag, Berlin, 2008.

[11] W. Reisig, *Petri Nets: An Introduction*, Springer-Verlag, Berlin, 1985.

Yong Yu, born in 1980, Ph. D. Currently lecturer in the School of Software, Yunnan University, Kunming, China. Received his Bachelor, Master and Doctor degrees from Yunnan University in 2002, 2006 and 2009 respectively. His main research interests include software engineering and information security.

Tong Li, born in 1963, Ph. D. Professor. Ph.D. supervisor in the School of Software, Yunnan University, Kunming, China. His main research interests include software engineering and software methodologies.

Qing Liu, born in 1963. Currently professor in the School of Software, Yunnan University, Kunming, China. His main research interests include software engineering and software methodologies.

Fei Dai, born in 1982, Ph.D. candidate in the School of Software, Yunnan University, Kunming, China. Received his Bachelor and Master degrees from Yunnan University in 2005 and 2008 respectively. His main research interests include software engineering and software process.