浙江理工大学学报(自然科学版),第35卷,第1期,2016年1月
Journal of Zhejiang Sci-Tech University (Natural Sciences)
Vol. 35, No. 1, Jan. 2016

DOI:10.3969/j.issn.1673-3851.2016.01.014

基于架构的服务组合可靠性预测

黄鸿云[a],张　娟[b],丁佐华[a]
(浙江理工大学,a.信息学院;b.理学院,杭州 310018)

摘　要: 软件可靠性是衡量软件质量的重要指标,而软件架构可以帮助预测软件的可靠性。为了研究基于服务组件架构的服务组合的可靠性计算,提出一种新的无需假设组件的失效是独立的可靠性计算方法。根据服务组件架构的端口执行情况构建Markov模型,并结合端口的失效数据来计算服务组合的可靠性。通过在线购物系统的可靠性计算论证方法的有效性。

关键词: 服务组合;面向服务组件的架构;端口;可靠性;失效数据
中图分类号: TP311.5　　**文献标志码:** A　　**文章编号:** 1673-3851 (2016) 01-0084-09　　**引用页码:** 010602

0　引　言

在过去三十多年里,人们对软件可靠性的计算进行了较多的研究[1-9],其中一种计算模型是基于架构的模型[5-9]。它不仅有助于基于组件的软件开发,同时还提供了一种可进行早期软件质量预测的方法。因此,可以利用软件架构来预测基于组件的软件的质量。例如,Chen等[5]提出了一种分析系统可靠性的模型,该模型适合具有不同架构层次的系统。此时,系统的可靠性分析依赖于组件的可靠性和不同层次间的连接器。此外,通过利用不同组件间的变迁概率来考虑操作剖面。但是他们要求假设组件和连接器的可靠性和变迁概率是相互独立的。单个组件的可靠性依赖于操作剖面和组件在系统的位置。系统可靠性的计算等价于组件执行路径的可靠性计算。执行路径的可靠性问题可以转化为一个基于组件的问题[6]。路径可靠性等于该路径上执行的组件的可靠性乘积。此时,可靠性的计算需假设组件间的失效是相互独立的以及接口的可靠性为1。Yacoub等[7]提出了一类关于基于组件的软件的可靠性模型和可靠性分析技术。为了分析系统的可靠性,他们利用组件间的交互场景构造称为组件依赖图(component-dependency graph,CDG)的概率模型。但是,CDG仅描述系统行为的序列模型,因此场景描述中的并发性质丢失了。

本文提出了一种新的计算面向服务组件架构(service component architecture,SCA)的服务组合的可靠性的方法。SCA的端口执行构成一个Markov链,端口映射为状态,则状态的可靠性即为端口的可靠性。于是,给定端口的失效数据即可在设计阶段预测系统的可靠性。另外,当系统在运行时,还可以获得实际失效数据。因此,所提方法能够从设计阶段到实施阶段提高系统的可靠性。一旦完成了组件的设计,同时也获得了可靠性计算公式;在实施之后,通过测试和失效密度分析,可以确定公式中个参数的具体值。该方法无需假设组件间的失效是相互独立的。

1　研究动机

Cheung[10]提出的基于组件的可靠性模型同时考虑了组件的可靠性和操作剖面,该模型使用描述

收稿日期:2015-02-27
基金项目:国家自然科学基金项目(61210004,61170015)
作者简介:黄鸿云(1977-),女,江苏如皋人,硕士研究生,主要从事软件工程方面的研究。
通信作者:丁佐华,E-mail:zouhuading@hotmail.com

系统行为的状态图。状态代表单个组件的执行,从一个状态到另一个状态的转移概率从系统的操作剖面获得。系统的可靠性依赖于状态的执行序列和单个状态的可靠性。

假设一个系统由 3 个组件构成,组件间的通信如图 1 所示。A、B 为组件 1 的 2 个端口,C_1、C_2、S 是组件 2 的 3 个端口,D、E 是组件 3 的 2 个端口。组件 1 通过端口 B 将控制传递给组件 2,组件 2 通过端口 C_2 将控制传递给组件 3,而组件 3 通过端口 E 将控制传递给组件 1。

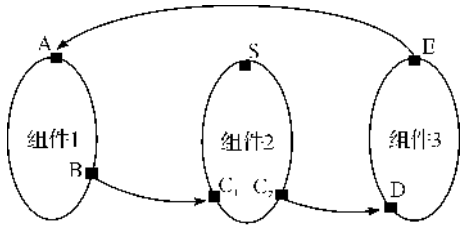

图 1　具有 3 个组件的系统

在文献[11-12]提出的方法中,假设不同组件间的控制切换具有 Markov 性质,则组件架构可以用离散时间 Markov 链(discrete time markov chain, DTMC)进行建模,其中状态表示活动组件,有向弧表示控制的切换。图 2(a)给出了上述系统的具有 3 个状态的 Markov 链。该方法将组件作为原子单元而忽略了端口间的交互。如果考虑端口的交互作用,则无需假设组件的失效是相互独立的,得到的 Markov 模型如图 2(b)所示,它有 6 个状态。

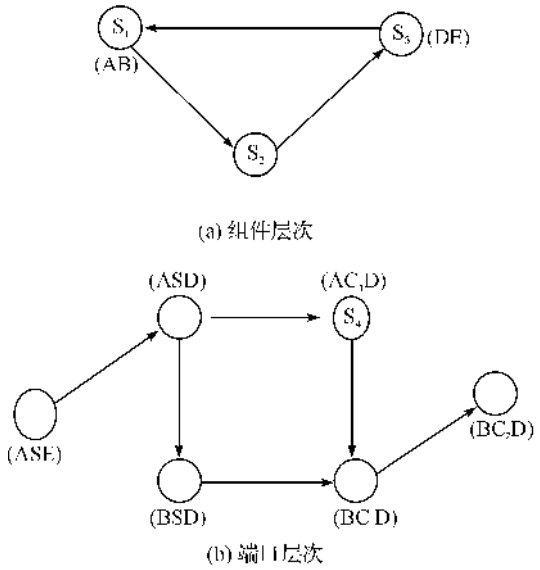

图 2　组件架构到 Markov 链的映射

2　基于端口的组件模型

SCA 模型[13]的提出使得基于服务的系统的实施变得更为容易,它提供一种定义和构建服务组件的方法,是诸如 BPLE 等服务组合语言的补充,使得开发者能够更加方便、有效地开发服务。但是 SCA 组装模型[13]缺乏形式化定义。因此,在以前工作中[14],形式化定义了 SCA 模型。

定义 1[14]:端口 p 是一个三元组(M,t,c),M 表示 p 中的有限方法集,t 表示端口类型(provided 或 required),c 表示通信类型(同步或异步)。

任一个方法具有如下形式:$op(\overline{Tx};\overline{Ty})$,其中 op 表示方法名,$\overline{Tx}$ 和$\overline{Ty}$ 分别表示输入参数和输出参数列表,T 表示参数类型,x、y 表示参数名字。$\bullet p$ 表示同步通信端口,$\blacklozenge p$ 表示异步通信端口。

定义 2[14]:组件 Com 是一个四元组(P_p, P_r, G, W),其中 P_p 是有限 provided 端口集;P_r 是有限 required 端口集;G 是有限子组件集合;$W \subseteq TP \times \bigcup_{C\in G}(C.P_p \cup C.P_r)$ 表示非自反的端口关系,其中 $TP = P_p \cup P_r \cup (\bigcup_{C\in G}(C.P_p \cup C.P)_r)$,$C.P_p$ 和 $C.P_r$ 分别表示子组件 C 的 provided 端口集合和 required 端口集合。

因为端口用来处理事件,因此可用端口活动来描述组件的动态行为。组件的动态行为表达式可定义为:

$BE ::=$

$p\bullet_m$ $p_1 \bullet_m p_2$	(Synchronous sending massage)
p°_m $p_1{}^\circ_m p_2$	(Synchronous receiving message)
$p\blacklozenge_m$ $p_1\blacklozenge_m p_2$	(Asynchronous sending message)
$p\lozenge_m$ $p_1\lozenge_m p_2$	(Asynchronous receiving message)
$stop$	(stop)
BBE	(Basic behavior expression)
$BE;BE$	(Sequence)
$BE \lhd b \rhd BE$	(Condition)
$b * BE$	(Loop)
$BE \sqcap BE$	(Non-determinism)
$BE \parallel BE$	(Parallel)
$BE[p_1/p_2]$	(Renaming)
$BE[p_1 \rightarrow p_2]$	(Wiring)

此表达式包含如下结构:序列化、选择、循环、并发和不确定。$p \bullet_m$ 表示端口 p 同步发送消息 m,$p_1 \bullet_m p_2$ 表示端口 p_1 同步发送消息 m 给端口 p_2。

$$BE[p_1 \to p_2] = \begin{cases} p_1 \bullet_m p_2 \\ p_2 \circ_m p_1 \\ p_1 \blacklozenge_m p_2 \\ p_2 \diamond_m p_1 \\ BE_1[p_1 \to p_2]; BE_2[p_1 \to p_2] \\ BE_1[p_1 \to p_2] \triangleright b \triangleleft BE_2[p_1 \to p_2] \\ b * BE_1[p_1 \to p_2] \\ BE_1[p_1 \to p_2] \sqcap BE_2[p_1 \to p_2] \\ BE_1[p_1 \to p_2] \Box BE_2[p_1 \to p_2] \\ BE \\ BBE \end{cases}$$

其中 $\odot \in \{\bullet_m, \circ_m, \blacklozenge_m, \diamond_m\}$。

动作轨迹 tr 定义为一系列的动作 $a_1, a_2, \cdots, a_n$。一个行为表达式 BE 可以通过操作语义规则变迁为另一个行为表达式 BE',记为 $BE \xrightarrow{tr} BE'$。变迁序列为动作轨迹 tr,即 $BE \xrightarrow{a_1} BE_1 \to \cdots \to BE_{n-1} \xrightarrow{a_n} BE'$。

两个组件 $Com_i = (P_p^i, P_r^i, G_i, W_i)(i \in \{1,2\})$ 称为可组合的,如果他们没有相同的 provided 端口,即 $P_p^1 \cap P_p^2 = \phi$。自动关联 required 端口集定义为:

$M_p(Com_1, Com_2) = \{p \mid p \in P_r^1 \wedge (p.M, \text{provided}, p.c) \in P_p^2\} \cup \{p \mid p \in P_r^2 \wedge (p.M, \text{provided}, p.c) \in P_p^1\}$

自动绑定集合定义为:

$M_w(Com_1, Com_2) = \{(p_1, p_2) \mid p_1 \in M_p(Com_1, Com_2) \wedge p_2 = (p_1.M, \text{provided}, p_1.c)\}$。

绑定设置操作定义为:

$$BE[W] = \begin{cases} BE, & W = \phi \\ BE[p_1 \to p_2][W \backslash \{(p_1, p_2)\}], & (p_1, p_2) \in W \end{cases}$$

于是,自动组合可定义为:

定义 3[14]:给定两个可组合的组件 $Com_i = (P_p^i, P_r^i, G_i, W_i)$ 和它们的动态行为表达式 $BE_i(i = 1,2)$,则它们的自动组合 $Com = (P_p, P_r, G, W)$(记为 $Com_1 \oplus Com_2$)定义为:

a)$P_p = (P_P^1 \cup P_P^2) \backslash \{(p.M, provided, p.c) \mid p \in M_p\}$;

b)$P_r = (P_r^1 \cup P_r^2) \backslash M_p$;

c)$G = G_1 \cup G_2$;

$BE[p_1 \to p_2]$ 表示不同组件的端口的绑定操作,具体定义为:

$$BE = p \bullet_m \wedge p = p_1$$
$$BE = p \circ_m \wedge p = p_2$$
$$BE = p \blacklozenge_m \wedge p = p_1$$
$$BE = p \diamond_m \wedge p = p_2$$
$$BE = BE_1; BE_2$$
$$BE = BE_1 \triangleright b \triangleleft BE_2$$
$$BE = b * BE_1$$
$$BE = BE_1 \sqcap BE_2$$
$$BE = BE_1 \| BE_2$$
$$BE = p_2 \Box p_1$$
$$BE = BBE$$

d)$W = M_w \cup \{(p,p) \mid p \in P_p \cup P_r\}$;

e)$BE = (BE_1 \Box BE_2)[M_w]$。

其中 M_p 和 M_w 分别是 $M_p(Com_1, Com_2)$,$M_w(Com_1, Com_2)$ 的缩写。

显然,根据定义,两个组件的组合还是一个组件。

3 从 SCA 到 Markov 可靠性模型

3.1 Markov 模型

假定有 n 个状态,s_1 是进入状态,s_n 是离开状态。状态图是一个有向图,其中每个节点 s_i 代表一个状态,有向边 (s_i, s_j) 表示从状态 s_i 到 s_j 的变迁。令 R_i 表示状态 s_i 的可靠性,P_{ij} 表示从状态 s_i 到 s_j 的可达概率,则根据状态图可得到转移矩阵 M 和各元素的值,即

$$M = \begin{bmatrix} 0 & R_1P_{12} & R_1P_{13} & \cdots & R_1P_{1i} & \cdots & R_1P_{1n} \\ 0 & 0 & R_2P_{23} & \cdots & R_2P_{2i} & \cdots & R_2P_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & RP_{i2} & RP_{i3} & \cdots & 0 & \cdots & R_iP_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & R_{n-1}P_{n-1,2} & R_{n-1}P_{n-1,3} & \cdots & R_{n-1}P_{n-1,i} & \cdots & R_{n-1}P_{n-1,n} \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}_{n \times n}$$

则整个系统的可靠性计算公式为:

$$R = T(1,n) \times R_n,$$

$$T(1,n) = (-1)^{n+1} \frac{|E|}{|I\text{-}M|},$$

其中:I 是单位矩阵,E 是矩阵 I-M 中除去第 n 行和第 1 列后的矩阵,$|E|$ 表示矩阵 E 的行列式,$|I\text{-}M|$ 表示矩阵 I-M 的行列式。

Cheung[10] 提出的模型只有一个初始状态和一

个终止状态。对于具有多个初始状态和多个终止状态的大型系统,可以通过在状态图中引入一个超初始状态和超终止状态将多初始状态多终止状态问题转化为单初始状态单终止状态问题。

定义 4:有限状态机是一个 5 元组(Σ,S,s_0,δ,F),其中:

a)Σ 是有限输入字符(有限非空符号集);

b)S 表示有限非空状态集;

c)s_0 表示系统的初始状态,$s_0 \in S$;

d)δ 表示状态变迁函数,$\delta:S \to S$;

e)F 表示终止状态集合,是S的一个字迹(可以为空)。

Markov 链是这样一个有限状态机:每个变迁都附加了一个概率,且到下一个状态的转移概率仅依赖于当前状态。对于离散时间 Markov 链,变迁仅发生在离散时间间隔或离散事件,变迁概率服从离散分布。因此,离散时间 Markov 链包含:

a)n 个状态组成的有限状态集合,$S=\{s_1,\cdots,s_n\}$;

b)$n\times n$ 的随机矩阵 $T=(P_{ij})_{n\times n}$,$P_{ij}\geqslant 0$ 且 $\sum_{j=1}^{n}P_{ij}=1$,其中 P_{ij} 表示当且仅当系统的当前状态为 s_i,系统的下一个状态为 s_j 的转移概率;

c)向量 $\pi^0=(\pi_1^0,\cdots,\pi_n^0)$,其中 π_i^0 表示系统初始处于状态 s_i 的概率。

3.2　从 SCA 建立 Markov 链的算法

根据端口的语义,可以通过计算 SCA 模型中所有的可达状态来建立有限状态机。下面给出了针对 3 个组件的组合的可达状态计算算法。令 Com_1、Com_2、Com_3 为 3 个组件,它们对应的动态行为表达分别为 BE_1、BE_2 和 BE_3。

算法①:建立服务组合的可达状态

输入:Com_1、Com_2、Com_3

输出:可达状态 RS

Begin

1　$M_w=\{p'\to p''\};\varepsilon=\phi;i=1;j=1;k=1$

2　$BE_1=x_1\odot_2\odot\cdots\odot x_{n_1}\odot;BE_2=y_1\odot y_2\odot\cdots\odot y_{n_2}\odot;BE_3=z_1\odot z_2\odot\cdots\odot z_{n_3}\odot$

3　$(BE_1\parallel BE_2\parallel BE_3)[M_w]=BE_1[M_w]\parallel BE_2[M_w]\parallel BE_3[M_w]$

$=x_1\odot x_2\odot\cdots\odot x_{n_1}\odot[M_w]\parallel y_1\odot y_2\odot\cdots\odot y_{n_2}\odot[M_w]\parallel z_1\odot z_2\odot\cdots\odot z_{n_3}\odot[M_w]$

$=x_1\odot[M_w]x_2\odot[M_w]\cdots x_{n_1}\odot[M_w]\parallel y_1\odot[M_w]y_2\odot[M_w]\cdots y_{n_2}\odot[M_w]\parallel z_1\odot[M_w]z_2\odot[M_w]\cdots z_{n_3}\odot[M_w]$

4　*while*$(i<=n_1\ \&\&\ j<=n_2\ \&\&\ k<=n_3)$

5　　if$(x_i\bullet_m=p'\&\&y_j\circ_m=p'')$

6　　$RS((BE_1\parallel BE_2\parallel BE_3)[M_w])=RS(x_{i+1};\cdots[M_w]\parallel y_{i+1};\cdots[M_w]\parallel z_k;z_{k+1};\cdots[M_w])$

7　　$RS=RS\to(x_i,y_j,z_k)\to\{(x_{i+1},y_j,z_k)\parallel(x_i,y_{j+1},z_k)\}$

8　　$i{+}{+};\ j{+}{+}$

9　　end if

10　　if$(x_i\bullet_m\neq p'\&\&y_j\circ_m=p'')$

11　　$RS((BE_1\parallel BE_2\parallel BE_3)[M_w])=RS((x_i\bullet_m[p'\to p'']);(y_j\circ_m;y_{j+1}\cdots[M_w]\parallel x_{i+1};\cdots[M_w])\parallel z_k;z_{k+1}\cdots[M_w])$;

12　　$RS=RS\to\{(x_i,y_j,z_{k+1})\to\}$;

13　　$i{+}{+};\ j{+}{+}$;

14　　end if

15　　if$(x_i\bullet_m\neq p'\&\&y_j\circ_m\neq p'')$

16　　$RS((BE_1\parallel BE_2\parallel BE_3)[M_w])$

$=\chi_1(BE)RS((y_j\circ_m;(y_{j+1}\cdots x_i;\cdots))[M_w]\parallel z_k;\cdots[M_w])+\chi_2(BE')RS((x_i\bullet_m;(x_{i+1}\cdots\parallel y_j;\cdots))[M_w]\parallel z_k;\cdots[M_w])$;

$BE=(y_j\circ_m;(y_{j+1}\cdots\parallel x_i;\cdots))\parallel z_k;\cdots,BE'=(x_i\bullet_m;(x_{i+1}\cdots\parallel y_j;\cdots))\parallel z_k;\cdots$;

17　　$i{+}{+};\ j{+}{+}$;

18　　end if

19　　if$(x_i\blacklozenge_m=p'\&\&y_j\diamond_m=p'')$

① 注:本文只列出 BE1 和 BE2 间存在绑定的 6 种情况;事实上,绑定还可以在 BE1 和 BE3 之间,也可以在 BE2 和 BE3 间,故共有 18 种情况。

20 $RS((BE_1 \parallel BE_2 \parallel BE_3)[M_w]) = RS(x_i \blacklozenge_m [p' \rightarrow p'']; x_{i+1}[p' \rightarrow p'']; \cdots \parallel y_j{}_m[p' \rightarrow p'']; y_{j+1}[p' \rightarrow p'']; \cdots \parallel z_k;$
$z_{k+1} \cdots [M_w] \parallel \langle x_i \blacklozenge_m y_j, \varepsilon \rangle \xrightarrow{x_i \blacklozenge_m y_j} \langle stop, \varepsilon \widehat{x_i \blacklozenge_m y_j} \rangle = RS(x_i \blacklozenge_m y_j) r(x_{i+1}; \cdots [M_w] \parallel z_k; z_{k+1} \cdots [M_w]); // \langle y_j \diamond_m x_i,$
$\varepsilon \rangle \xrightarrow{y_j \diamond_m x_i} [\text{stop}, \varepsilon / x_i \blacklozenge_m y_j]$

21 $RS = RS \rightarrow (x_i, y_j, z_k) \rightarrow (x_{i+1}, y_j, z_k) \rightarrow$

22 $i{+}{+}; \ j{+}{+};$

23 end if

24 if($x_i \blacklozenge_m \neq p' \&\& y_j \diamond_m = p''$)

25 $RS((BE_1 \parallel BE_2 \parallel BE_3)[M_w]) =$

$RS((x_i \blacklozenge_m [p' \rightarrow p'']); (y_j \diamond_m; y_{j+1} \cdots [M_w] \parallel x_{i+1}; \cdots [M_w]) \parallel z_k; z_{k+1} \cdots [M_w]) \parallel$
$\langle x_i \blacklozenge_m z_k, \varepsilon \rangle \xrightarrow{x_i \blacklozenge_m z_k} \langle \text{stop}, \varepsilon \widehat{x_i \blacklozenge_m z_k} \rangle$

26 $RS = RS \rightarrow (x_i, y_j, z_{k+1}) \rightarrow;$

27 $i{+}{+}; \ j{+}{+};$

28 end if

29 if($x_i \blacklozenge_m \neq p' \&\& y_j \diamond_m \neq p''$)

30 $RS((BE_1 \parallel BE_2 \parallel BE_3)[M_w] = \chi_1(BE) RS((y_j \diamond_m; (y_{j+1} \cdots \parallel x_i; \cdots))[M_w] \parallel z_k; \cdots [M_w]) +$

$\chi_2(BE') RS((x_i \blacklozenge_m; (x_{i+1} \cdots \parallel y_j; \cdots))[M_w] \parallel z_k; \cdots [M_w]);$
$BE = (y_j \diamond_m; (y_{j+1} \cdots \parallel x_i; \cdots)) \parallel z_k; \cdots, BE' = (x_i \blacklozenge_m; (x_{i+1} \cdots \parallel y_j; \cdots)) \parallel z_k; \cdots$

31 $RS = RS \rightarrow (x_i, y_j, z_k) \rightarrow (x_i, y_j, z_{k+1}) \rightarrow;$

32 $i{+}{+}; \ j{+}{+};$

33 end if

34 end while

End

4 Markov 可靠性模型的配置

根据上述算法，可以计算可达状态并建立有限状态机；但是，状态间的转移概率和各状态的可靠性还需要在后面通过测试进行配置。

4.1 配置状态间的转移概率

在软件开发的后期，当测试或者领域数据可以使用时，组件轨迹可以通过分析工具获得，而测试覆盖工具用来获得一系列执行轨迹和变迁弧的频数。需要说明的是，每个变迁的概率必须包含轨迹的重复次数。

4.2 配置状态的可靠性

两个服务的交互的特点包含如下 3 个属性：伙伴链接、端口类型以及操作。服务组合的后期绑定可以通过将服务的结束点分配给伙伴链接而获得。如果将服务看作是组件，则可以定义包含这 3 个属性的端口，如此对服务组合的测试实际上就是对端口交互的测试。于是，服务组合的可靠性依赖于端口的交互行为，或者端口可靠性。状态的可靠性可以通过下述方法进行计算。假设某个状态关联了端口集 (p_1, p_2, p_3)。在测试阶段可以获得端口 p_1，p_2，p_3 的失效，所有这些失效合在一起就是该状态的失效。这也是和传统的状态的可靠性计算方法的不同之处。注意，本文考虑的是一个状态对应一个来自不同组件的端口的组合；如果一个状态有好几个不同的端口组合，则可用隐式 Markov 链计算系统的可靠性。

获取失效数据的方法之一是监测每个端口中变量的所有值，并判断这些值是否在预期的范围内。另一种方法是利用程序不变量，而程序不变量可以用 Daikon 工具获得，于是每个端口中“不好”的不变量就可以当作失效数据[15]。

5 实例分析

本文通过一个在线购物系统的例子来介绍所提出模型的使用。虽然该系统是一个简化的版本，但是

它还是包含了如下结构:序列化、选择、循环和并行。

在线购物系统包含 5 个组件,分别是:INIT(发出订单请求)、ORDER(处理订单)、INVOICE(创建发货清单)、PAYMENT(信用卡支付)、DELIVERY(运输货物)。首先,客户发送订单请求;然后系统检查订单并创建发货清单。如果通过信用卡支付成功,系统发送运输数据给运输服务提供者。为方便起见,令 C_{11} = INIT(包含 3 个端口 $p^{11}_{request}$, p^{11}_{check}, p^{11}_{order}), C_{12} = ORDER(包含 4 个端口 p^{12}_{check}, p^{12}_{reject}, p^{12}_{order}, $p^{12}_{confirm}$), C_{21} = INVOICE(包含 4 个端口 p^{21}_{order}, $p^{21}_{question}$, p^{21}_{update}, $p^{21}_{invoice}$), C_{22} = PAYMENT(包含 5 个端口 $p^{22}_{invoice}$, $p^{22}_{orderStatus}$, $p^{22}_{deliverReq}$, $p^{22}_{cardInfo}$, $p^{22}_{paycheck}$), C_3 = DELIVERY(包含 4 个端口 $p^{3}_{deliverReq}$, $p^{3}_{deliver}$, $p^{3}_{getStatus}$, $p^{3}_{terminate}$)。在线购物系统的 SCA 如图 3 所示。

在实现时,首先创建组件,定义接口,然后将这些组件绑定到一起并用 BPEL 编辑器来实现。本文利用 IBM WebSphere Integration Developer 来开发 BPEL 的自主服务绑定。如此,在服务选择之后,服务绑定可以自动完成。通过将 SCA 模型映射为完整的 J2EE 应用包来部署应用,Java 应用可由 Tomcat 和 SQL 数据库来实现。

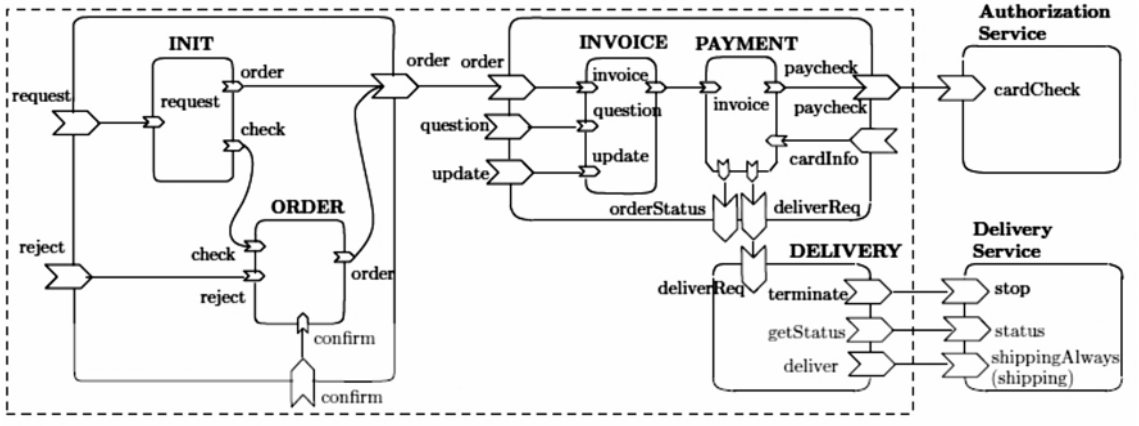

图 3　在线购物系统的 SCA

原子组件 C_{11} 的端口定义为:

$p^{11}_{request} = (\{request(\overline{Req},\overline{Resp})\}, provided, synchronous)$,

$p^{11}_{check} = (\{check(\overline{checkReq},\overline{checkResp})\}, required, synchronous)$,

$p^{11}_{order} = (\{order(\overline{order})\}, required, asynchronous)$

其动态行为表达为 $BE^{11} = p^{11}_{request}\circ$; $(p^{11}_{check} \bullet \lhd b_1 \rhd p^{11}_{order}\blacklozenge)$,其中 b_1 = "检查订单"。

原子组件 C_{12} 的端口定义为:

$p^{12}_{check} = (\{check(\overline{checkReq},\overline{checkResp})\}, provided, synchronous)$

$p^{12}_{reject} = (\{reject(\overline{orderData},\overline{rejectResp})\}, provided, synchronous)$

$p^{12}_{order} = (\{order(\overline{order})\}, required, asynchronous)$

$p^{12}_{confirm} = (\{confirm(\overline{orderData},\overline{confirmResp})\}, provided, synchronous)$

其动态行为表达为 $BE^{12} = p^{12}_{check}\circ$; $(p^{12}_{reject}\circ \lhd b_2 \rhd p^{12}_{confirm}\circ)$; $p^{12}_{order}\blacklozenge$,其中 b_2 = "取消订单"。同理可得到其他组件的端口信息和动态行为表达。例如,原子组件 C_{21} 的动态行为表达为 $E^{21} = p^{21}_{order}\diamond$; $(p^{21}_{question}\circ \lhd b_3 \rhd p^{21}_{update}\circ)$; $p^{21}_{invoice}\bullet$,其中 b_3 = "需要更多问题";原子组件 C_{22} 的动态行为表达为 $EB^{22} = p^{22}_{invoice}\circ$; $b_4 * (p^{22}_{cardInfo}\diamond; p^{22}_{paycheck}\bullet); p^{22}_{orderStatus}\blacklozenge$,其中 b_4 = "支付失败";根据用户输入,假设 b_4 最多可循环三次,存在第 i 次循环的概率为 $p_{b_4}(i), i = 1,2,3$;原子组件 C_3 的动态行为表达为 $EB^3 = p^{3}_{deliverReq}\circ; p^{3}_{deliver}\blacklozenge; p^{3}_{getStatus}\bullet; p^{3}_{terminate}\blacklozenge$。

令 C_1 为组件 C_{11} 和 C_{12} 的自动组合,则 $C_1 = C_{11} \oplus C_{12} = (P^1_p, P^1_r, G_1, W_1)$,其中:

a) $P^1_p = P^{11}_p \cup P^{12}_p \setminus \in \{(p.M, provided, p.c) \mid p \in M_p\} = \{p^{11}_{request}, p^{12}_{reject}\}$,

b) $P^1_r = P^{11}_r \cup P^{12}_r \setminus M_p(C^{11}, C^{12}) = \{p^{12}_{confirm}, p^{11}_{order}, p^{12}_{order}, p^{11}_{check}\} \setminus \{p^{11}_{check}\} = \{p^{12}_{confirm}, p^{11}_{order}, p^{12}_{order}\}$,

c) $G_1 = \{\ \}$,

d) $W_1 = M_w \cup \{(p,p) \mid p \in P^1_p \cup P^1_r\} = \{(p^{11}_{check}, p^{12}_{check})\} \cup \{(p,p) \mid p \in P^1_p \cup P^1_r\}$

$= \{(p^{11}_{check}, p^{12}_{check})\} \cup \{(p^{11}_{request}, p^{11}_{request}), (p^{11}_{order}, p^{11}_{order}), (p^{12}_{reject}, p^{11}_{order}), (p^{12}_{confirm}, p^{12}_{confirm}), (p^{12}_{order}, p^{12}_{order})\}$,

e) $BE^1 = (BE^{11} \parallel BE^{12})[\{(p^{12}_{check} \rightarrow p^{11}_{check})\}]$。

令 C_2 为组件 C_{21} 和 C_{22} 的自动组合，即 $C_2 = C_{21} \oplus C_{22}$。同理上述可得 C_2 的 P_p^2, P_r^2, G_2 和 W_2，以及行为表达式 $BE^2 = (BE^{21} \parallel BE^{22})[\{(p_{invoice}^{22} \rightarrow p_{invoice}^{21})\}]$。

此外，假设运输服务提供者 C_{ds} 提供的服务为：$BE^{ds} = p_{shippingAlways} \diamond \triangleleft b \triangleright p_{shipping} \circ ; p_{status} \circ ; p_{stop} \diamond$；信用卡检查服务提供者 C_{cs} 提供的服务为：$BE^{cs} = p_{cardcheck} \circ$。

令端口的关系集为：

$$W = W_1 \cup W_2 \cup \{(p_{order}^1, p_{order}^2)\} \cup \{(p_{deliverReq}^2, p_{deliverReq}^3)\} \cup \{(p_{deliver}^3, p_{shippingAlways})\} \cup \{(p_{getStaus}^3, p_{status})\} \cup \{(p_{terminate}^3, p_{stop})\}.$$

则在线购物系统的动态行为语义模型（记作 SCA_{online}）为：

$$SCA_{online} = C_1 \oplus C_2 \oplus C_3 \oplus C_{ds} \oplus C_{cs}.$$

其行为表达为：

$$BE = (BE^1 \parallel BE^2 \parallel BE^3 \parallel BE^{ds} \parallel BE^{cs})[M_w] = BE^1[M_w] \parallel BE^2[M_w] \parallel BE^3[M_w] \parallel BE^{ds}[M_w] \parallel BE^{cs}[M_w].$$

其次，根据测试可获得运行轨迹，于是有：

$$M_w = \{p_{check}^{12} \rightarrow p_{check}^{11}, p_{invoice}^{22} \rightarrow p_{invoice}^{21}, p_{order}^{21} \rightarrow p_{order}^{12}, p_{cardcheck} \rightarrow p_{paycheck}^{22}, p_{deliverReq}^3 \rightarrow p_{deliverReq}^{22}, p_{deliver}^3 \rightarrow p_{shippingAlways}, p_{getStatus}^3 \rightarrow_{status} p_{terminate}^3 \rightarrow p_{stop}, I = User - Inputs\};$$

$$BE^1 = BE^{11} \parallel BE^{12} = (p_{request}^{11} \circ ; (p_{order}^{11} \blacklozenge \triangleleft b_1 \triangleright p_{check}^{11} \bullet)) \parallel (p_{check}^{12} \circ ; (p_{reject}^{12} \circ \triangleleft b_2 \triangleright p_{confirm}^{12} \circ) ; p_{order}^{12} \blacklozenge);$$

$$BE^2 = BE^{21} \parallel BE^{22} = (p_{order}^{21} \diamond ; (p_{question}^{21} \circ \triangleleft b_3 \triangleright p_{update}^{21} \circ) ; p_{invoice}^{21} \bullet) \parallel (p_{invoice}^{22} \circ ; b_4 * (p_{cardInfo}^{22} ; \diamond p_{paycheck}^{22} \bullet) ; p_{orderStatus}^{22} \blacklozenge);$$

当选择好条件 b_1，b_2，b_3，b_4 后，可以计算可达系统的可达状态：

$$RS(BE)[M_w] = (p_{request}^{11}, p_{check}^{12}, p_{invoice}^{21}, p_{paycheck}^{22}, p_{deliverReq}^3)$$
$$\rightarrow (p_{order}^{11}, p_{check}^{12}, p_{invoice}^{21}, p_{paycheck}^{22}, p_{deliverReq}^3) \parallel (p_{check}^{11}, p_{check}^{12}, p_{invoice}^{21}, p_{paycheck}^{22}, p_{deliverReq}^3)$$
$$\rightarrow (p_{order}^{11}, p_{check}^{12}, p_{invoice}^{21}, p_{cardInfo}^{22}, p_{deliverReq}^3) \parallel (p_{check}^{11}, p_{order}^{12}, p_{invoice}^{21}, p_{paycheck}^{22}, p_{deliverReq}^3)$$
$$\rightarrow (p_{order}^{11}, p_{check}^{12}, p_{invoice}^{21}, p_{deliverReq}^{22}, p_{deliverReq}^3) \parallel (p_{check}^{11}, p_{order}^{12}, p_{invoice}^{21}, p_{cardInfo}^{22}, p_{deliverReq}^3)$$
$$\rightarrow (p_{order}^{11}, p_{check}^{12}, p_{invoice}^{21}, p_{deliverReq}^{22}, p_{terminate}^3) \parallel (p_{check}^{11}, p_{order}^{12}, p_{invoice}^{21}, p_{deliverReq}^{22}, p_{deliverReq}^3)$$
$$\rightarrow (p_{order}^{11}, p_{check}^{12}, p_{invoice}^{21}, p_{deliverReq}^{22}, p_{getStatus}^3) \parallel (p_{check}^{11}, p_{order}^{12}, p_{invoice}^{21}, p_{deliverReq}^{22}, p_{terminate}^3)$$
$$\rightarrow (p_{order}^{11}, p_{check}^{12}, p_{invoice}^{21}, p_{deliverReq}^{22}, p_{deliver}^3) \parallel (p_{check}^{11}, p_{order}^{12}, p_{invoice}^{21}, p_{deliverReq}^{22}, p_{getStatus}^3)$$
$$\rightarrow (p_{order}^{11}, p_{check}^{12}, p_{invoice}^{21}, p_{deliverReq}^{22}, p_{deliver}^3) \parallel (p_{check}^{11}, p_{order}^{12}, p_{invoice}^{21}, p_{deliverReq}^{22}, p_{deliver}^3)$$
$$\rightarrow (p_{requset}^{11}, p_{check}^{12}, p_{invoice}^{21}, p_{paycheck}^{22}, p_{deliverReq}^3).$$

根据可达状态，可获得对应的 Markob 链模型，如图 4 所示。其中 R_1 表示进入状态的可靠性，R_{15} 表示退出状态的可靠性，概率矩阵为：

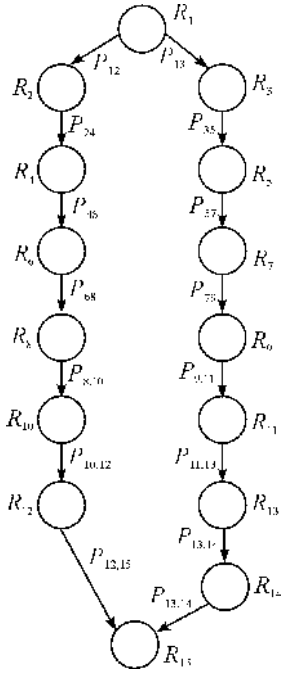

图 4 在线购物系统对应的 Markov 链可靠性模型

$$P = \begin{bmatrix} 0 & 0.6 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

因此，迁移矩阵为：

$$M=\begin{bmatrix}
0 & R_1P_{1,2} & R_1P_{1,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & R_2P_{2,4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & R_3P_{3,5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & R_4P_{4,6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & R_5P_{5,7} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & R_6P_{6,8} & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_7P_{7,9} & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_8P_{8,10} & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_9P_{9,11} & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_{10}P_{10,12} & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_{11}P_{11,13} & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_{12}P_{12,15}\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_{13}P_{13,14} & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_{14}P_{14,15}\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

其中:$R_1=0.9812$,$R_2=0.9943$,$R_3=0.9932$,$R_4=0.9961$,$R_5=0.9903$,$R_6=0.9973$,$R_7=0.9906$,$R_8=0.9983$,$R_9=0.9953$,$R_{10}=0.9975$,$R_{11}=0.9946$,$R_{12}=0.9952$,$R_{13}=0.9934$,$R_{14}=0.9878$,$R_{15}=0.9978$。

因此,根据可靠性计算公式有:$T(1,15)=0.9484$,$R=T(1,15)*0.9978=0.9436$。

6 结　语

本文提出了一种新的计算面向服务系统的可靠性方法。利用面向服务组件的架构,构建了Markov可靠性模型,它可在测试阶段重新配置。该方法的优点在于无需假设组件的失效是独立的。其次,考虑失效数据的获得,所提方法和已有方法是一个均衡过程。对于已有的基于组件的可靠性模型,因为失效数据来自组件,所以容易测试,但是无法反应内部的错误,因为对于一个输入,只有一个输出;对于本文提出的模型,失效数据来自端口,故能反应系统内部错误信息,但是需要增加额外的测试工作。

参考文献:

[1] CHEN M, LYU M R, WONG W E. Effect of code coverage on software reliability measurement[J]. IEEE Transactions on Reliability, 2001, 50(2): 165-170.

[2] HAMLET D. Are we testing for true reliability[J]. Software, IEEE, 1992, 9(4): 21-27.

[3] KRISHNAMURTHY S, MATHUR A P. On the estimation of reliability of a software system using reliabilities of its components[C]//Software Reliability Engineering, 1997. Proceedings, The Eighth International Symposium on. IEEE, 1997: 146-155.

[4] LITTLEWOOD B. Software reliability modeled for modular program structure[J]. IEEE Transactions on Reliability, 1979, 28(3): 241-246.

[5] MA J, CHEN H. A reliability evaluation framework on composite web service[C]//Service-Oriented System Engineering, 2008. SOSE′08. IEEE International Symposium on. IEEE, 2008: 123-128.

[6] DOLBEC J, SHEPARD T. A component based software reliability model[C]//Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research. IBM Press, 1995:19-29.

[7] YACOUB S, CUKIC B, AMMAR H H. A scenario-based reliability analysis approach for component-based software[J]. IEEE Transactions on Reliability, 2004, 53(4): 465-480.

[8] GOKHALE S S, WONG W E, Trivedi K S, et al. An analytical approach to architecture-based software reliability prediction[C]//Computer Performance and Dependability Symposium, 1998. IPDS′98. Proceedings. IEEE International. IEEE, 1998: 13-22.

[9] WANG W, PAN D, CHEN M. Architecture-based software reliability modeling[J]. The Journal of System and Software, 2006, 79(1): 132-146.

[10] CHEUNG R C. A User-Oriented Software Reliability Model[J]. IEEE Transitions on Software Engineering, 1980, 6(2):118-125.

[11] GOSEVA P K, TRIVEDI K S. Architecture-based approach to reliability assessment of software systems[J]. Performance Evaluation, 2001, 45(2): 179-204.

[12] GOSEVA P K, MATHUR A P, Trivedi K S. Comparison of architecture-based software reliability models[C]//Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on. IEEE, 2001: 22-31.

[13] CHAPMAN M, EDWARDS M, BEISIEGEL M, et al. Service component architecture assembly mode specification version1. 1 [R]. Technical report, OASIS, 2011.

[14] DING Z H, CHEN Z, LIU J. A rigorous model of service component architecture[J]. Electronic Notes in Theoretical Computer Science, 2008, 207: 33-48.

Architecture-based Prediciton of Service Combination Reliability

HUANG Hongyun[a], *ZHANG Juan*[b], *DING Zuohua*[a]

(a. School of Information Science and Technology; b. School of Science, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: Software reliability is an important index to measure the quality of software, while software architecture can help us to predict the software reliability. In order to study reliability of service combination based on Service Component Architecture (SCA), a new method is proposed, which needs not assume that component failures are independent. Markov model is built based on the execution of ports of SCA. Then, reliability of service combination is calculated in the combination of failure data of ports. The effectiveness of the method is demonstrated through the reliability calculation of an online shopping system.

Key words: service combination; architecture oriented to service component; port; reliability; failure data

(责任编辑: 陈和榜)