软件可靠性评估中 UML 应用①

尹鲁燕，吴开贵，张　烨
(重庆大学 计算机学院，重庆 400044)

摘　要：在开放环境下，软件规模日趋扩大，结构更加多元化，传统的基于状态的软件可靠性评估方法，状态空间膨胀增加了计算复杂度，而且不能对多种典型的系统结构进行很好的描述。为此，对传统的方法进行了改进，用 UML 的用例图分解系统，序列图描述子系统，并都作为软件可靠性分析的输入，通过自底向上的方法评估软件的可靠性，符合当前大规模复杂结构的软件系统可靠性评估。
关键词：开放环境；软件可靠性；状态；UML

Software Reliability Evaluation of UML-Based Application

YIN Lu-Yan, WU Kai-Gui, ZHANG Ye
(Dept. of Computer Science, Chongqing University, Chongqing 400044, China)

Abstract: As the scale of the software is increasing in the open environment, the structure and architecture of the software is becoming more and more complex and diversified. Using the traditional state-based software reliability evaluation method, the complexity of reliability analysis will be highly increased due to the state space explosion. Besides, traditional reliability evaluation method couldn't describe system with typical complex structures very well. So, this paper presents an improved state-based reliability evaluation method. It utilizes the user case diagram to divide system and utilizes the sequence diagram to describe the architecture of the subsystems. Both diagrams are used as the input for the software reliability analysis method. This method utilizes a bottom-up way approach to evaluate the software reliability, which adapts to the evaluation of large scale systems with complex structures.
Key words: open enviroment; software reliability; state; UML

1　引言

在过去的三十年中，对软件可靠性进行了大量的研究，出现了很多种分析方法[1]，早期的可靠性分析采用黑盒方法，利用软件测试或运行时期统计的数据来来进行分析和度量软件可靠性，精确度较高，但是如果在此阶段才发现软件未满足可靠性要求，修改的代价巨大，阻碍了这些方法的应用发展。因此，一个有效的设计开发过程应该在早期，如体系结构设计阶段，就包含对系统可靠性的分析和预测，以指导软件开发过程，确保最终构造出来的系统满足可靠性要求[2]。

基于体系结构的分析方法[3,4]大致分为两类，基于路径的方法和基于状态的方法。基于路径的方法的一个显著的缺点是精确度不高，只是对系统可靠性的大概估算，而且不适用于具有无限路径的系统，存在一定的局限性。相对来说，基于状态在适用性和精确度上都更加优越。基于状态的方法，可适用无限路径的软件系统，并且在开放型环境中，构件之间是松耦合的，构件之间具有相对较高的独立性，构件间转移的符合 Markov 性，因此，基于状态的方法更加适合在开放环境下使用。但是传统的基于状态的可靠性评估方法还存在以下不足：

1）状态空间的膨胀。在开放环境下，构成软件的构件可能成百上千，采用传统的分析方法，会造成状态空间的急剧膨胀，增加计算的复杂度甚至不可计算。

① 基金项目:国家自然科学基金(90818028)
收稿时间:2011-08-24;收到修改稿时间:2011-09-20

© 中国科学院软件研究所 http://www.c-s-a.org.cn

2）不适用程序结构复杂的软件。在开放环境下，软件的体系结构更加多元化，传统的分析方法在适用范围和易用程度上都有一定的局限性，大部分只适用于经典的结构，无法处理并行、调用等复杂结构的系统[4]。

为了解决上述问题，本文提出一种改进的在软件开发周期早期进行可靠性计算的基于 UML 用例图和顺序图的软件可靠性评估方法（UML-based Software Reliability Evaluation，USRE）。一方面利用用例图对软件系统进行分解，划分为若干子系统并分别计算可靠性，最后再综合的方式以缩小状态空间，降低计算的复杂度。另一方面利用顺序图对各种程序结构进行了分析，如并行、循环、调用、冗余等，进一步增强本方法的使用范围。

本文组织如下：第二部分是相关的工作的介绍。第三部分是本文的方法概述。第四部分给出了一个实例进一步展示本文提出的方法。第五部分是对本文的总结和下一步工作的展望。

2 相关工作

针对开放环境下的大规模软件系统，有人提出用基于场景的方法来评估软件可靠性，例如 Yacoub[7]方法，将系统划先分为若干个执行场景分别计算，然后计算整个系统的可靠性。其中，场景是由特定输入激发的一组构件间的交互，表示完成一个功能模块。此方法有效的分解了大规模系统，化繁为简，降低了计算复杂度，但是缺点是没有给出如何划分场景的方法，而且由于是基于路径的方法，局限了此方法的适用度。在本文中借鉴了这种方法中大而化小的思想，并提出用 UML 用例图作为分解软件系统的工具模型。

统一建模语言 UML 是现有的最普遍使用的软件建模工具之一，可以从各个角度对软件进行描述，如用例图描述参与者与系统之间的交互情况，类图表示系统的静态结构，活动图、序列图等动态表示系统的行为，可以描述包括并行在内的各种复杂的结构，勾勒出软件的体系结构。可以作为在早期进行可靠性分析的重要输入。

目前有很多工作研究将非功能性属性加入到原有的基础 UML 模型中，作为可靠性分析的输入[2,8,9]，这些方法提供了如何将现有的可靠性建模工具和可靠性分析方法结合，但是对包含了并行、循环、调用等复杂结构的系统束手无策。因此本文并对其进行了扩展，提出了一种基于 UML 的可对包含多种程序结构的软件系统进行可靠性评估方法（UML-based Software Reliability Evaluation，USRE）。

3 基于UML的软件可靠性评估方法概述

3.1 系统分解

UML 中提供的模型图中，通常每个用例对应于系统中的一个功能模块。本文中，用用例图来反映系统的使用情况。根据用例图将系统划分为不同的用例模块即不同的子系统 U_1，U_2，U_3……U_n。假设一个系统由两类参与者，两个用例组成，根据 profile 将系统使用的可靠性信息标注在用例图中，如图 1 所示，将每个参与者使用系统的概率，每个参与者使用子系统的概率都标注在用例图中。

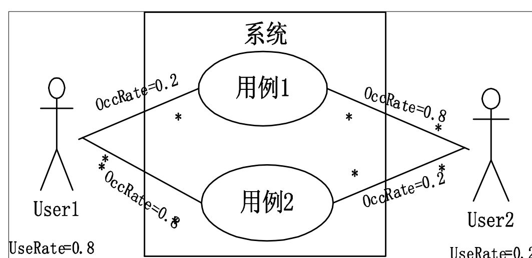

图 1 用例图

整体系统的可靠性表示为

$$R(A)=\sum_{1}^{n}(R_{U_i}*P_{U_i}) \tag{1}$$

其中，R_{U_i} 表示用例 U_i 的可靠性，R_{U_i} 根据基于状态的可靠性进一步进行计算。P_{U_i} 是用例 U_i 的发生概率，P_{U_i} 根据图中标注的可靠性信息由公式（2）可得:

$$P_{U_i}=\sum_{i=0}^{n}(P_{a_i}*P_{a_i,j}) \tag{2}$$

P_{a_i} 为第 i 类参与者使用系统的概率，$P_{a_i,j}$ 为第 i 类参与者使用用例 j 的概率。

每个用例是对完成某项功能的一组动作序列的总描述，可以抽象为一个子系统，但是用例不必说明怎样实现此子系统，对子系统的描述可以利用 UML 的另一模型顺序图。本文利用 UML 顺序图作为子系统的动态建模工具。相对于 UML 的其他模型，顺序图以时间为轴线描述了用例中构件交互的时序和方式，对构件的转移关系做出了详细的描述，任何用例都可

© 中国科学院软件研究所 http://www.c-s-a.org.cn

以使用顺序图进一步阐明和实现。下一部分，介绍如何用 UML 顺序图对复杂的子系统进行描述，并转换为局部的状态转移图。

Webster[10]指出状态是指在特定条件和活动中的描述系统的一系列事件或属性。据此，本文给出状态模型中状态和状态间的转移的定义。

定义：

1）状态是指在特定条件下描述系统的事件集。其中事件是激发系统中的一个构件获得控制。也就是说在一个状态下可能是一个或多个构件同时被激发。状态用 S 表示，S_i 表示系统处于第 i 种状态。

2）转移是指从一种状态成功执行完后进入到另一状态。用 P_{ij} 表示从状态 S_i 成功转移到状态 S_j 的概率。

当系统中存在多个输入节点或多个输出节点的时候，引进一个超级开始节点连接到输入节点或所有的输出节点连接到一个超级终止节点，转移概率都为 1。

3.2 复杂子系统局部结构的描述

在这一部分我们描述如何利用顺序图模型化几种典型结构：分支、并行、调用、循环、冗余的软件系统，并将其转换为基于状态的软件可靠性计算模型。

我们首先对顺序图进行扩展，将在 profile 中获取的可靠性信息如分支概率、转移概率等标注在顺序图中。在 UML 的序列图中返回消息是一个可选择部分。在本文中，我们只表明主动消息；而且，我们只考虑没有嵌套结构的情况。

3.2.1 顺序结构和分支结构

顺序结构表示构件依次执行，分支结构表示程序将会从几个候选构件中选择一个执行，每个分支的执行概率取决于运行时的上下文信息。在顺序图中，构件间一般是是按照时间为序，依次执行，而用关键字 alt 标注并用框线框起的部分是分支结构。图 3-2(a)包含了分支结构和顺序结构。$c1,c2,\cdots\cdots c5$ 五个构件，当构件 $c1$ 运行完毕后，会以一定的概率选择分支 $c2$ 或分支 $c3$ 、$c4$ 执行，分支内 $c3$ 、$c4$ 是顺序执行，包含顺序结构和分支结构的顺序图向状态图转变时，构件与状态是一一对应的，状态间的转换概率为顺序图中各个转移发生的概率。转换后的状态图如图 3.2(b)所示，$S_1,S_2,___S_5$ 是 Markov 链中的状态，构件 $c1$ 与状态 S_i 一一对应。

在分支结构中，每个状态的可靠性为：

$$R(S_i)=R(i) \tag{3}$$

其中，$R(i)$ 为状态 S_i 对应的构件 c_i 的可靠性。

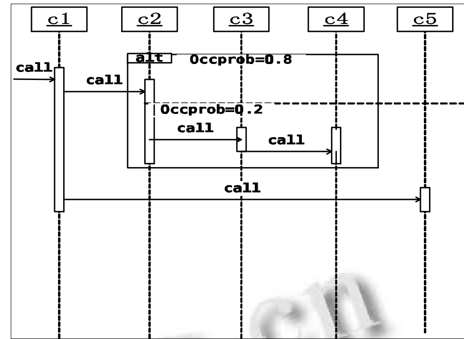

图 2(a) 分支结构的序列图描述

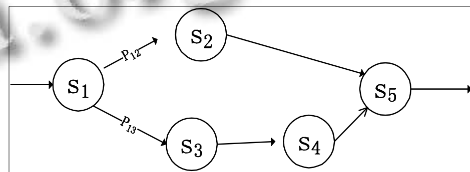

图 2(b) 图 2(a)对应的状态图

3.2.2 循环结构

在顺序图中，循环可能由一个构件或多个构件共同组成，将循环视为一个一个整体，用关键字 Loop 标记并用方框框起，循环以一定的概率再返回本身执行，当循环体一直执行直到满足某一终止条件时，跳出循环体向下执行。根据状态的定义，将顺序图中的循环体向状态图转变时，循环体可整体抽象成一个状态。在图 3（a）中，循环体 Loop 由构件 $c3$ 、$c4$ 、$c5$ 组成，循环体内构件顺序执行，Loop（$c3$ 、$c4$ 、$c5$）顺序执行一次的可靠性为 $R(l)=R(c3)R(c4)R(c5)$,控制流返回循环体 Loop 的概率为 $P(l)$，当满足某个终止条件时，跳出循环向下执行。顺序图中的循环体向状态图转变，如图 3（b）所示，Loop 抽象成状态 S_l 。

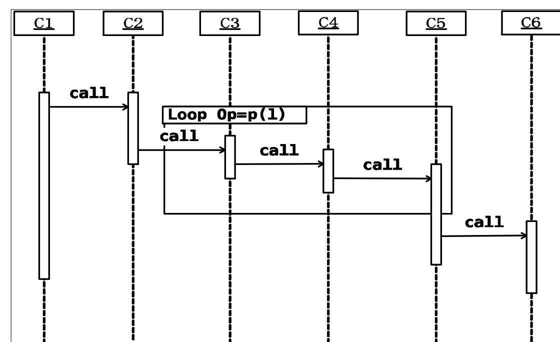

图 3(a) 循环结构的序列图描述

© 中国科学院软件研究所 http://www.c-s-a.org.cn

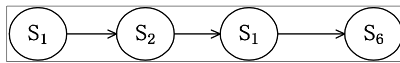

图 3(b)　图 3(a)对应的状态图

循环状态 S_l 的可靠性为：

$$R(S_l)=\frac{1}{1-R(l)P(l)}R(l)[1-P(l)] \tag{4}$$

3.2.3 并行结构

在并行执行环境中，多个构件同时运行，当并行结构中的每一个构件都成功运行后，整个并行结构才算成功。将一个并行结构视为一个并行体，用关键字 par 标记并框起，如图 4（a）中方框中的构件 $c2$ 和 $c3$，组成了一个并行体，箭头指向方框内的调用消息是并行的，同样箭头指向方框外的调用消息也是并行的。跟循环一样，当将顺序图中的并行结构向状态图转变时，将序列图中的并行体抽象成一个状态，如图 4(b)所示，par 抽象成状态 S_p。

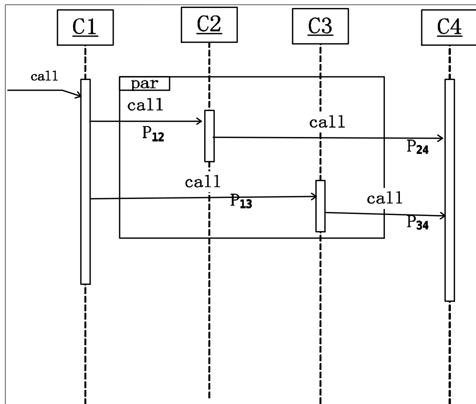

图 4（a）　并行结构的序列图描述

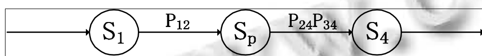

图 4（b）　图 4（a）对应的状态图

在图 4（b）中，状态 S_p 是构件 $c2$、$c3$ 组成的并行结构，$c1$ 到 $c2$、$c3$ 的转移概率相等,$P_{12}=P_{13}$，由状态 S_1 到 S_2 的转移概率也为 P_{12}。$c2$、$c3$ 到 $c4$ 的转移概率也是相等的，$P_{24}=P_{34}$，当 $c2$、$c3$ 都成功执行后，才会转移到 $c4$，所以从 S_p 转移到 S_3 的概率为 $P_{24}P_{34}$。并行体状态 S_p 的可靠性为：

$$R(S_p)=\prod_{C_n\ in\ S_i} R_n \tag{5}$$

3.2.4 冗余结构

冗余结构是指一个主构件有几个备用构件，当主构件出现故障时，其他的备用构件会提供服务。冗余体用关键字 backups 标注并框起。如图 5（a）中方框中的构件 $c2$ 和 $c3$，构件 $c3$ 作为构件 $c2$ 的冗余，$c2$、$c3$ 构成 backups。将顺序图中的冗余结构向状态图转变时，冗余体抽象成是一个状态，如图 5（b）所示，pal 抽象成状态 S_b。

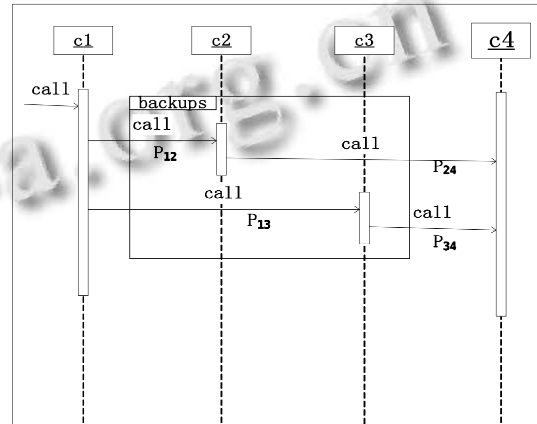

图 5（a）　冗余结构的序列图描述

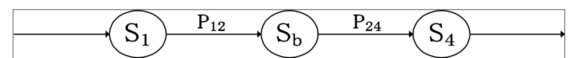

图 5（b）　图 5（a）对应的状态图

在图 5（b）中，S_b 是构件 $c2$、$c3$ 组成的冗余结构，和并行结构相似，$c1$ 到 $c2$、$c3$ 的转移概率相等，$P_{12}=P_{13}$，构件 $c1$ 只能往构件 $c2$ 或 $c3$ 转移，因此状态 S_1 到 S_b 的转移概率为 P_{12}。$c2$ 是主构件，$c1$ 成功执行后转到 $c2$，当 $c2$ 成功执行后，转移到 $c4$，但是如果 $c2$ 出现故障，$c3$ 代替 $c2$ 执行，然后转到 $c4$，$c2$、$c3$ 到 $c4$ 的转移概率是相等的，$P_{24}=P_{34}$，状态 S_b 转移到 S_3 的概率为 P_{24},。冗余体对应的状态的可靠性为：

$$R(S_b)=(1-\prod_{C_k\ in\ S_i}(1-R_n)) \tag{6}$$

3.2.5 调用结构

调用结构也是在各种程序开发方法中经常用到的一种经典结构，像分布式环境中构件间的相互交互、面向对象中的方法等。我们可以把调用结构看成是顺序执行的，按照顺序结构的方法来计算。

3.3 规约状态图

在子系统的运行中，相同的事件集可能发生多次，这时系统多次处于同一状态下，在将顺序图转换为状态图的过程中，在状态图中会出现相同的状态结点的

© 中国科学院软件研究所 http://www.c-s-a.org.cn

情况，这在基于状态的可靠性评估方法中是不允许的。因此，要对顺序图初步转换的状态图进行规约处理，即合并相同的边和结点。使得规约后的状态图中，每个状态节点唯一。在进行规范的前，初始状态图中分支结构中所有分支中的边要乘以分支发生的概率，表示在此分支发生的情况下，状态间的转移概率。

如前面的描述，我们将当下参与循环体、冗余体、并行体的构件认为是一个超级状态节点，如不在这些超级状态中，构件还是转换为普通状态。

算法：规约初级状态图

设转换前的状态图用一个二元组表示<S,E>。$S=\{s_1,s_2,s_3......s_n\}$，表示状态结点的集合。$E=\{e\mid\forall e\in V, e=(s_i,s_j,p_{ij})\}$，表示所有有向边的集合，$p_{ij}$表示状态间的转移概率。设转换后的图为<S’,E’>，定义相同。

Step0　初始化<V’,E’>，$S'=\varnothing, E'=\varnothing$；

Step1　将分支路径中的各条转移边乘以分支发生的概率。

Step2　对 E 中的每一条有向边：$e_{jm}(s_j,s_m,p_{jm})$

1 检查 S’中是否存在同名的节点，若没有，将s_j、s_m添加到 S’中。

2 如果s_j与s_m具有相同的名称，即有向边连接的是相同的节点，那么合并两个状态，从该结点到达其他结点的转移和其他结点到达该结点的转移保持不变；

否则　如果$e_{jm}\notin E'$，那么将$e_{jm}(s_j,s_m,p_{jm})$加入到E'中，否则$p'_{jm}+=p_{jm}$

Step3　对每一个$s_i\in S'$

定义$p_{sum}=0$；

定义E_i为所有从状态 si 出发的有向边的集合，$E_i=\{e\mid e=(s_i,s_j,p_{ij})\}$对$E_i$中的每个元素 e

$p_{sum}+=p_{ij}$；

$p_{ij}=p_{ij}/p_{sum}$，使得状态s_i的出边概率为 1；

4　实例分析

在大规模的软件系统中，通常涉及到多个甚至数十个子系统。根据本文提出的 USRE 的步骤：

第一步利用 UML 用例图将系统分解为若干子系统，并计算出每个子系统的使用概率。

第二步对每个子系统用顺序图进行建模描述，并将其转换为对应的状态图。

第三步对每个子系统的初始状态图进行规约，形成规范化的状态图，并根据 Chuang 方法计算每个子系统的可靠性。

第四步根据公式(1)综合计算整个系统的可靠性。

为了进一步说明本方法的使用，本文只结合一个只有两个子系统的软件实例来说明 USRE 方法的评估过程，实例软件的 UML 用例图如图 1 所示：由 2 类参与者，2 个用例组成，每个用例是一个子系统。首先根据用例图中标注的有关可靠性信息，计算每个用例的发生概率。参与者a_1、a_2使用系统的概率为：$P_{a_1}=0.8$、$P_{a_2}=0.2$，参与者a_1、a_2使用每个用例的概率为：$P_{a_2,1}=0.2$、$P_{a_1,2}=0.2$，$P_{a_2,1}=0.2$、$P_{a_2,2}=0.8$，根据公式（2）可得$P_{U_1}=0.18$，$P_{U_2}=0.82$，即子系统 1 发生的概率为 0.18，子系统 2 发生的概率为 0.82。然后们用 UML 的顺序图描述各个子系统。以子系统 1 为例，用 UML 顺序图进行描述，如图 6 所示：

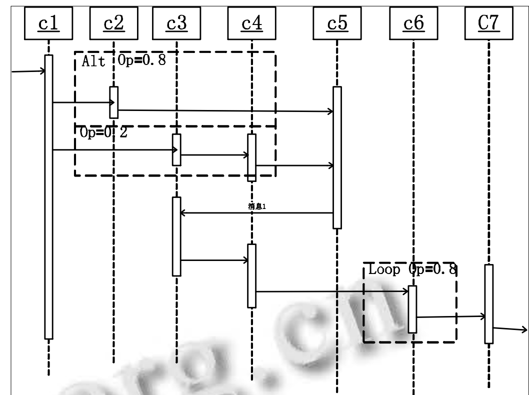

图 6　子系统 1 的执行顺序图

软件从构件$c1$开始执行，按照随机概率调用构件$c2$或构件$c3$、$c4$，然后顺序执行$c5$,然后再顺序执行$c3$、$c4$和循环体 Loop，最后执行$c7$后结束。根据 3.1 小节中描述的方法，并加入了开始节点和终止节点，将图 6 转换为初始状态图，如图 7 所示：

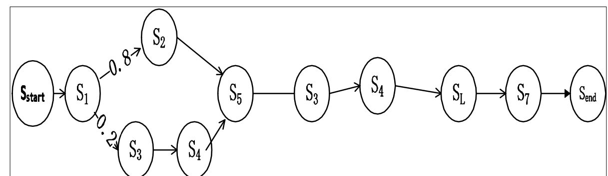

图 7　图 6 对应的初始状态图

将图 7 按照 3.2 中的算法进行规范化，得到图 8：

© 中国科学院软件研究所　http://www.c-s-a.org.cn

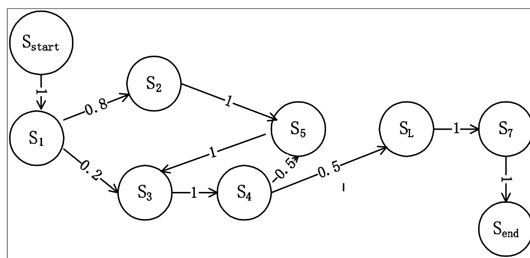

图 8 规范化后的状态图

各个构件的可靠性如表 1 所示:

表 1 各构件的可靠性

C_i	C_1	C_2	C_3	C_4	C_5	C_6	C_7
R_i	0.99	0.98	0.95	0.99	0.90	0.90	0.99

根据公式（3）、公式（4）计算各个状态的可靠性如表 2 所示:

表 2 场景 1 中个状态的可靠性

S_i	S_{strat}	S_{end}	S_1	S_2	S_3	S_4	S_5	S_l	S_7
R_i	1	1	0.99	0.98	0.95	0.99	0.90	0.64	0.99

按照 chuang 方法，根据图 8，我们可得状态转移矩阵：

$$M=\begin{array}{c}\\ S_{start}\\ S_1\\ S_2\\ S_3\\ S_4\\ S_5\\ S_l\\ S_7\\ S_{end}\end{array}\begin{array}{c}\begin{array}{ccccccccc} S_{start} & S_1 & S_2 & S_3 & S_4 & S_5 & S_l & S_7 & S_{end}\end{array}\\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0.792 & 0.198 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0.98 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0.99 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0.495 & 0 & 0.495 & 0 & 0\\ 0 & 0 & 0 & 0.90 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.640 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.99\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}\end{array}$$

n=9

$|I-M|$=0.5050

$|E|$=0.2169

$T(S_{start},S_{end})=\ (-1)^{n,1}\dfrac{|E|}{|I-M|}$=0.4295

子系统 1 的可靠性：$R_{u1}=T(S_{start},S_{end})\times R_{end}$=0.4295

同样可得子系统 2 的可靠性为 R_{u2}=0.7535

因为 $P_{U_1}=0.18$， $P_{U_2}=0.82$，根据公式（3）可得整个系统的可靠性为

$$R(A)=\sum_{1}^{n}(R_{U_i}*P_{U_i})=0.6947$$

5 总结与展望

本文提出一种新的基于 UML 的软件可靠性分析方法。描述了如何用 UML 分解大规模软件并分析各种控制结构，得到初步的构件状态图进而规范化，最终形成了一种符合当前开放环境、大规模、高复杂度的软件系统的软件可靠性评估方法。在适用范围上都对传统的软件可靠性评估方法进行了改进。

在今后的工作中，还需要通过实际的软件项目进一步验证该方法的有效性和可操作性；同时考虑更加复杂的嵌套结构等控制结构以扩展此方法适应各种结构的软件系统，而且计算构件的重要性，以便对系统进行优化和改进，也是下一步深入研究的方向。

参考文献

1 Chen MH. Investigating coverage-reliability relationship and sensitivity of reliability estimates to errors in the operational profile. Computer Science and Informatics Journal, 1995, 25(3):165–170.

2 柳毅,麻志毅,何啸,邵维忠.一种从 UML 模型到可靠性分析模型的转换方法.软件学报,2010,21(2):287–304.

3 Goseva K. Architecture-based approach to reliability assessment of software systems. Preprint submit to Elsevier Science, 27 February 2001.

4 Gokhale SS. Architecture-based software reliability analysis: overview and limitations. IEEE Trans. on Dependable And Secure Computing, 2007,4(1).

5 黄宁,陈未知.基于架构风格的软件可靠性评估.计算机系统应用,2009,18(5):198–201.

6 陆文,徐峰.一种开放环境下的软件可靠性评估方法.计算机学报,2010,33(3):452–462.

7 Sherif Y. A scenario-based reliability analysis approach for componet-based software. IEEE Trans. on Reliability, 2004, 53(4).

8 Bernardi S, Merseguer J, Petriu DC. Adding dependability analysis capabilities to the MARTE profile. In: Czarnecki K, Ober I, Bruel JM, Uhl A, Volter M, eds. Proc. of the 11th Int'l Conf. on Model Driven Engineering Languages and Systems. Berlin: Springer-Verlag, 2008. 736–750.

9 Corellessa V, Pompei A. Towards a UML profile for QoS: A contribution in the reliability domain. ACM SIGSOFT Software Engineering Notes, 2004,29(1):197–206.

10 Agnes M. Webster's new world college dictionary. 4th ed. Macmillan, USA, 2000.1399.

© 中国科学院软件研究所 http://www.c-s-a.org.cn