基于 UML 活动图的软件可靠性预测方法

苏 岳 李 蜜 汪文心 张德平
(南京航空航天大学计算机科学与技术学院 南京 210016)

摘 要 以构件化的软件开发方法为背景,提出了一种能自动将 UML 活动图转换为模型 Markov 链的可靠性预测方法。该方法基于构件化的软件体系结构,从 UML 的活动图和顺序图出发,通过构造一个名为"控制结构转移图"的中间模型,将标注了可靠性信息的 UML 模型转换为 Markov 链,并通过递归方法自动生成测试路径,然后依据每条路径的可靠性信息来估计整个软件系统的可靠性。转换结果可以直接被现有分析方法用来进行软件可靠性预测工作,从而使分析变得高效和模型化。

关键词 UML 模型,Markov 链,软件可靠性,模型转换,路径测试
中图法分类号 TP301 **文献标识码** A

Software Reliability Prediction Approach Based on UML Activity Diagram

SU Yue LI Mi WANG Wen-xin ZHANG De-ping
(College of Computer Science & Engineering, Nanjing University of Aeronautics & Astronautics, Nanjing 210016, China)

Abstract In the context of component-based software development, this paper proposed an approach to automatically transform UML activity diagrams of software architecture to Markov chain for the quantitative evaluation of reliability. Based on the component-based software architecture, it utilizes four types of UML diagrams: use case, sequence, activity and component diagrams, extending them and annotating them with reliability related attributes. Then, the diagrams are transformed into a Markov chain based on analysis model by constructing an intermediate model called Component Transition Graph(CTG). Result of this transformation can be directly used in the existing analysis methods to predict software reliability, which facilitates the analysis task of software designer.

Keywords UML model, Markov chain, Software reliability, Model transformation, Path test

随着计算机技术的日益成熟,软件系统已经被广泛地应用到社会中的各个领域,在人们的工作和生活中发挥着举足轻重的作用。软件可靠性是评价软件质量的一个重要指标,它通常由"在特定环境、特定时间下系统无故障运行"的概率[1]来度量。在传统研究工作中,软件可靠性分析和度量通常在软件开发实现完成后进行,其主要依据软件可靠性模型(如可靠性增长模型)来进行分析和预测。虽然这种方法得到的评估结果较为精确,但若是在此阶段才发现软件系统不满足可靠性需求,修改的代价将十分巨大。因此,能在软件开发早期就提前对软件系统的可靠性进行分析、指导和优化,对于构造满足用户可靠性需求的高质量软件具有重要意义[2]。

软件体系结构包含了对构成软件系统的构件的描述,构件间的交互、组合的模式的描述以及每种模式的约束条件。这些都为整个软件系统的可靠性评估提供了有效和必需的信息。一方面,它包含了整个系统的结构信息,而系统结构是软件可靠性预测的一个重要评估因素;另一方面,它还展示了系统在运行时刻活动状况的动态信息,也是用来预测软件可靠性的重要依据[3]。

统一建模语言 UML 作为一种常用的系统建模工具,能够从不同角度对系统进行描述。如用用况图(use case diagram)来描述系统的需求,用类图(class diagram)来描述系统的静态结构,用构件图(component diagram)来描述软件系统实现的物理结构,用活动图(activity diagram)来描述系统的动态行为等。这些工具为早期的系统可靠性设计和分析提供了清晰的框架和必需的信息,且具有描述复杂的控制流程和并行活动的能力,能够对不同的复杂场景进行建模。此外,路径模型[4]是一种最常用、最有效的软件可靠性分析模型,它将执行路径定义为一次软件运行时的典型执行序列,使用运行场景(Scenario)生成测试路径并依据各路径的可靠性和出现概率得到软件可靠性。

然而,现有的一些基于 UML 模型进行的软件可靠性估计和预测的研究成果[3-16]需要根据系统特征建立相应的数学模型[10,11],要求软件设计人员必须具有相关的数学背景才能完成建立可靠性分析模型的工作,这增加了可靠性分析的难度。为此,本文提出了一种基于构件软件体系结构的软件可靠性分析方法。该方法充分利用 UML 活动图对复杂场景的建模能力,能够自动识别控制流程中的基本结构,采用组合结点[2]的概念简化软件体系结构以自动生成测试路径,再根据

本文受南京航空航天大学计算机科学与技术学院科技创新资助项目资助。
苏 岳(1994-),男,主要研究方向为软件测试与可靠性分析,E-mail:447532207@qq.com;李 蜜(1993-),男,主要研究方向为软件测试与可靠性分析;张德平(1973-),男,博士,主要研究方向为软件测试与软件可靠性建模,E-mail:depingzhang@163.com。

各路径中构件标注的可靠性信息来计算路径的可靠性，并对系统可靠性进行进一步的分析和预测。

1 基本概念

用况图描述系统所具有的功能，每个用况都与一个系统功能相对应，且都有其发生的概率。每个用况对应的功能都可用一个活动图来进行描述[13]。

构件图描述软件系统中各种构件及它们之间的依赖关系，主要用来表示组成系统的所有构件以及各个构件自身的可靠性。这些可靠性信息可能来源于构件本身的描述文档、专家知识、功能相似的构件等；对于自己开发的构件，也可以通过现有方法[13]进行计算。UML 活动图主要包括初始结点、终止结点、结点集合和控制边集等基本元素。为了更好地描述这些元素，对 UML 活动图进行如下形式化定义[14]：

一个活动图是一个四元组 $G=\langle A,E,in,F\rangle$：

(1)in 为初始结点，从该点出发总存在一条路径能到达其它所有结点；

(2)F 表示所有终止结点集；

(3)$A=\langle AN,ON,CN\rangle$为结点集合，其中 AN 表示活动结点集，ON 为对象结点集，$CN=DN\cup MN\cup FN\cup JN$ 为控制结点，满足 DN 为分支结点(decision node)，MN 为合并结点(merge node)，FN 为分岔结点(fork node)，JN 为汇合结点(join node)；

(4)E 为控制边集，满足 $E=\{(x,y)\mid x,y\in A\}$。

从 UML 活动图的定义可知，活动图中的控制结点包括分支、合并、分岔和汇合。分支结点在软件系统流程中很常见，它一般用于表示对象类所具有的条件行为。分支结点包括一个入迁移和两个或者两个以上带条件的出迁移，且出迁移的条件是互斥的。合并结点包括两个或者两个以上的入迁移和一个出迁移。合并结点代表的控制流不需要同步发生，当单个控制流达到合并结点后，可以继续往下进行。在活动图中分支结点与合并结点用空心菱形表示。

UML 活动图中引入分岔结点与汇合结点，实现并发控制流的建模，分岔结点和汇合结点都使用加粗的水平线段表示。对象在运行时可能会存在两个或者多个并发运行的控制流，从宏观角度而言，各并发线程的控制流之间并无时间先后的制约。分岔结点用来描述并发线程，将动作流分为两个或者多个并发运行的控制流，每个分岔结点可以有一个入迁移和两个或者多个出迁移，每个并发输出独立执行且互不干扰。汇合结点代表两个或者多个并发控制流同步发生，当所有的控制流都达到汇合结点后，控制才能继续往下进行，每个汇合结点可以有两个或者多个入迁移和一个出迁移。汇合结点用于同步这些并发分支，以达到共同完成一项事务的目的。为了便于可靠性分析，将一个基本结构用一个虚拟的结点表示，称为组合结点。

下面引入 UML 活动图中常见的基本控制结构：

(1) 顺序结构

顺序结构是指 UML 活动图中活动结点的执行依时间先后而发生，是一个有序的活动结点序列，其中结点可以是基本结点也可以是组合结点，可用链式结构描述。活动图中的顺序结构对应于构件软件的顺序结构，表示各个构件依次执行。

(2)选择结构

选择结构表明程序将从几条候选路径中选出一条来执行，而具体选择哪一条，取决于运行时刻的程序上下文信息。显然，这种软件系统运行时刻的行为对整个系统运行的成功与否有着明显的影响。我们一般无法预测到这种行为，但可以通过统计每个分支被执行的概率进行概率标注，而这些概率足以用来对系统的可靠性进行评估。

(3)循环结构

活动图中一个循环结构是一个包含可以重复执行的活动的集合，它通过一个分支结点来建模，循环结构具有两个离开这个分支结点的分支(出迁移)。在这两个分支中，一个分支通过与区域外面的结点相连退出循环，另一个分支通过与区域内的结点相连继续循环。如果分支结点是该区域的首个结点(即区域的入口结点)，则称这种循环结构为前测回路(pre-test loop)。分支结点也可作为区域的最后结点出现(即区域的出口结点)，这种循环称为后测回路(post-test loop)。

基于构件的软件中，循环结构常被用来表示某个构件将被重复执行。在这里，也有一些概率信息，这些信息表示控制流离开和回到循环体的概率。

(4)分岔结构

在 UML 活动图中，分岔结点常用来对两条或多条并行执行路径建模。这些并行执行路径可能需要被合并到单个控制流中。基于这个目的，连接结点如汇合结点和合并结点常常会根据其设计需求来选择采用哪种连接结点。汇合结点常用来描述需要等待完成所有并行执行的路径后才继续执行的进程。我们把这种结构称为同步分岔结构。有时，一个进程并不需要等待所有的路径都执行完成，在这种情况下常采用合并结点来控制。当一条路径执行到合并结点，并不需要等待所有其它的路径也执行到合并结点，而是可以继续执行下面的活动，称这种结构为合并分岔结构。分岔结构也可分为匹配分岔结构、不匹配分岔结构和 N 中取 K 分岔结构。

2 可靠性分析方法

2.1 方法概述

在可靠性分析过程中需要针对不同用户确定每个功能的使用情况，通常使用剖面来描述用户使用系统各个功能的情况，可以通过加入概率标注信息后的用况图来获取这些信息。对于软件的执行环境，本文采用连接件的方式来描述，在各个相邻执行构件之间增加一个连接件，并标注相应的可靠性信息。当程序从一个构件运行到另一个构件时，就看作是发生了一次从相应的活动结点到另一个活动结点的转移。每条转移都被赋予了一个转移概率，表明转移发生的可能性大小。有了这些信息后，利用图论相关的数学方法或工具生成相应的测试路径，再对单个构件的可靠性信息估计路径可靠性，就能对系统的可靠性进行分析。

从标注有可靠性相关信息的用况图、活动图和构件图出发，将其转换为可靠性分析模型的测试路径，关键的一步是根据活动图构造出相应的测试场景，然后再利用活动与构件之间的对应关系转化为构件路径。为使测试场景生成过程更加简捷、高效，本文提出了一种称为 ITM 的中间模型，用来作为 UML 活动图到测试场景生成的桥梁。基于 UML 活动图的可靠性分析的基本框架如图 1 所示。

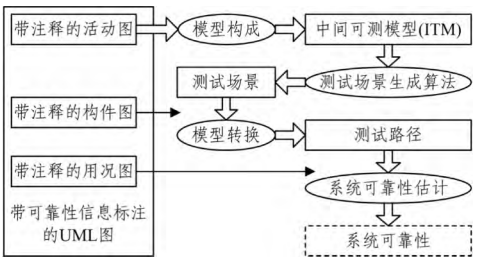

图 1　可靠性分析的基本框架

2.2　可靠性信息标注

(1)用况图的可靠性信息标注

用况图(Use Case Diagram,UCD)确定系统中所包含的参与者(Actor)、用况(Use Case)和两者之间或其自身的关系,用况图是基于系统要实现功能的一个可视化描述。一个用况对应系统的一个功能,用况图中的可靠性标注主要用来表示每个功能被使用的概率。

本文采用的标注方法[2,9]为:假设 q_i 表示第 i 个用户使用系统的比率,p_{ix} 表示第 $i(i=1,2,\cdots,m)$ 类用户在使用系统的前提下,使用某个功能的概率。则每个用况(或功能) x 发生的概率可通过用户使用系统的比率以及用户在使用系统的前提下使用该用况的概率计算得到,即:

$$P(x)=\sum_{i=1}^{m} q_i p_{ix} \tag{1}$$

其中,m 表示不同用户类型数。注意到所有类型用户的使用比率之和为 1,每一类用户使用各个用况的概率之和为 1,并且所有用况发生的概率之和也为 1。

(2)构件图的可靠性信息标注

可靠性信息标注主要是对构件图的构件之间的每一个连接子进行可靠性标注,这些可靠性信息一般假定是已知的,信息来源于构件本身的描述文档、专家知识、功能相似的构件等;对于自己开发的构件,也可以通过现有方法[3]进行计算,一些传统的可靠性模型如 JM 模型、NHPP 类模型等也可以用来估计构件的可靠性。为不失一般性,这里可以将连接子的可靠性合并到它的前驱结点(构件)之中或直接令连接子的可靠性为 1.0。

(3)活动图的可靠性标注

活动图中的分支结点表明活动执行过程中不同的路径,它通常与用况执行时发生的不同场景对应。因此,在分支结点的每条出边标上一个概率,表明在当前活动发生的条件下该分支发生的概率,该概率与对应场景发生的概率相同。3 种常见的迁移概率确定方法为:无信息方法(uninformed approach),主要对分支结点的每一条出边分配相同的概率;有信息方法(informed approach),根据功能相似的系统或由先前版本抽样得到的用户活动序列来计算分支结点的每条出边的概率;有意图方法(intended approach),允许假定用户得到相应概率信息,允许更改用户关注重点即某些出迁移,而有针对性地给出不同的概率。

为获得迁移概率可采用以下方法:

1) 基于假设:假定在一个状态下的全部 n 个激励事件具有相同的发生概率 $1/n$,迁移上监视条件为真或假的概率均为 1/2;

2) 基于历史数据:通过对旧版本软件或相同应用的类似软件的使用情况进行记录和分析,获得软件的历史使用数据,用该数据推算迁移概率;

3) 基于评估:通过经验丰富的专家或用户经验建议对软件未来的使用状态进行分析、预测和评估,从而推断软件的使用情况,得到迁移概率。

2.3　UML 活动图的简化方法

在 UML 活动图中每一个活动对应一个构件,活动之间的转移概率对应构件之间的转移概率,活动图的一个场景对应于一个构件序列(或测试路径),本文中称之为"进程"[12]。然而,对于一个复杂系统而言,其活动图较为复杂,特别是对于一些含有并发嵌套结构的情形,正确生成、理解活动图的每一个场景会变得十分困难。为了使复杂的活动图更加清晰简单,便于测试场景的自动生成和理解,需要对 UML 活动图进一步简化。考虑活动图特性,本文采用重复迭代的方法通过对 UML 活动图中的基本结构进行自动识别,采用相对应的组合结点替代,直到 UML 活动图成一个链式的顺序结构。下面分别介绍几种常见基本结构的识别方法和化简规则,实现对活动图的化简。

(1)循环结构的识别与合并

假定 x 是一个分支结点,结点 $y(y\neq x)$ 是 x 的前驱结点(predecessor node)。如果存在一个从结点 x 发出的分支 b,经过一系列的结点后合并到结点 x,则称结点 x 是一个循环结构的分支结点。

如果前驱结点 y 在分支 b 上,则该循环为后测回路,否则为前测回路。如果循环结构为前测回路,分支结点 x 既为循环区域的入口结点也为出口结点。另一方面,如果循环结构为后测回路,在分支 b 上分支结点 x 后的直接结点为入口结点,x 为出口结点。一旦一个循环结构的入口结点和出口结点被识别,则包含出口结点和出口结点在内的结点和边形成的区域就为控制结构图中的一个循环结构,从而可以用一个组合结点来替代。

(2)选择结构的识别与合并

一个选择结构可以用来对两个或两个以上互不相容的具有可选择性的路径进行建模,它由一个具有两个或两个以上出边的分支结点来描述。各个路径的执行依赖于条件表达式的值,对于条件表达式的每一个值,从分支结点引发一条边。基于分支结点出边的个数,选择结构可以分为两维或多维选择。一个选择结构可能是匹配选择结构,也可能是不匹配或 N 中取 K 类型的选择结构。因此,对于不同类型的选择结构,应根据其特性进行识别。

情形 1　匹配选择结构

假定 x 为分支结点,如果存在一个合并结点 y 满足所有从结点 x 发出的分支都合并到结点 y,则称 x 为一个匹配选择结构的分支结点。在这种情形下,如果分支结点 x 为入口结点,合并结点 y 为出口结点,则所有包含入口结点和出口结点在内的入口结点和出口结点之间的结点和边形成了一个匹配选择结构,可以用一个组合结点来替代。

情形 2　不匹配选择结构

不匹配选择结构中,在控制流分支的最后没有聚集结点。然而,通过这种区域表示的控制流在知道这些分支的终止结点的情形下,可以约简为一个组合结点。因此可以利用这种信息来识别不匹配选择结构。假定 x 是一个决策结点,如果

由 x 发出的所有分支通过一系列的结点后终止于控制流的终止结点，则可判断从结点 x 开始的控制流结构是一个不匹配选择结构。在这种情形下，决策结点 x 是不匹配选择结构的入口结点，每个分支的终止结点是控制结构的出口结点。

每条分支终止时，其后继不再有活动发生。这样就可以对这类控制结构用一个组合结点代替。

情形 3　N 中取 K 选择结构

N 中取 K 选择结构中，K 条分支被汇集，剩下 $N-K$ 条分支不匹配，满足 $K<N$。N 中取 K 选择结构中，对于 K 条汇集分支，其出口结点是合并结点，剩下的 $N-K$ 为不匹配选择分支，终止结点为出口结点。因此可以分别采用情形 1 和情形 2 的方法对其识别，然后用一个组合结点替代。

(3)分岔结构的识别与组合

基于构件的软件系统，分岔结构的可靠性很难判断，有的情况下要求所有的构件都要成功完成，而有的情况下只要求其中的几个构件成功完成。因此，在基于构件的软件系统进行可靠性评估之前，需要对这些分岔结构进行识别。

情形 1　匹配分岔结构

假定 x 为一个分岔结点，如果从 x 开始的所有分支通过一系列的结点后汇集到一个汇合结点或合并结点 y，则称这类控制结构为匹配分岔结构。若分岔结点 x 为入口结点，结点 y 为出口结点，则匹配分岔结构为包含结点 x、结点 y 和结点 x 与结点 y 之间所有结点和边组成的区域。

情形 2　不匹配分岔结构

从一个分岔结点 x 发出的不同执行路径可能并不会汇集到一个汇合结点或合并结点，相反，每一条路径都分别结束于控制流的一个终止结点，我们称这种情形为不匹配分岔结点。假定 x 是一个分岔结点，如果从 x 发出的所有分支通过一系列的结点后结束于控制流的终止结点，则结点 x 为入口结点，而所有的控制流终止结点为出口结点；对于具有多个终止结点的情形，只保留一个终止结点而去掉其它的终止结点，然后对控制流重新定向到这个终止结点。从入口结点到终止结点的所有结点和边组成的集合构成了一个不匹配分岔结构，可以用一个组合结点来代替。

情形 3　N 中取 K 匹配分岔结构

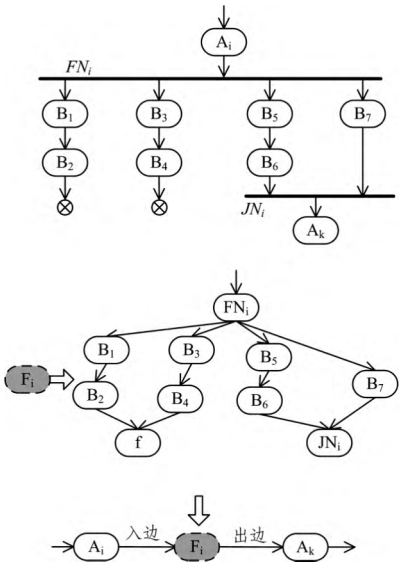

图 2　N 中取 K 合并分岔结构的识别与合并

在分岔结构中有时并不完全匹配，从分岔结点发出的路径中有一部分结束于终止结点，剩余部分路径则聚集于合并结点或汇合结点，我们称这类分岔结构为 N 中取 K 匹配分岔结构。在这种情形下，K 条路径使用合并(或汇合)结点聚集于单个控制流，而剩余的 $N-K$ 条路径为不匹配路径。因此可以采用情形 1 和情形 2 的方法分别进行识别，最后由一个组合结点替代，如图 2 所示。

对于给定的 UML 活动图，通过对活动图中的各个基本结构进行识别和替代，最后生成了只有顺序结构的中间可测模型。

2.4　基于中间可测模型的场景生成

由生成的只具有顺序结构的中间可测模型出发，逐步对组合结点进行展开，各个组合结点中根据其基本结构方式，生成相应的含有组合结点或不含有组合结点的路径。然后对每条含有组合结点的路径进行展开，直到所有生成的路径中都不含有组合结点为止。其中不同的组合结点其内部路径生成也不相同，详细算法分析如下：

B 是所有可能的路径的集合。每条可能路径是 t_i，每个可能路径 t_i 的节点是 N_{ij}。T 是生成的初始路径集。只要 T_i 路径上有至少一个符合的节点，则判定 T_i 符合既定结构生成路径，即路径 T_i 归入测试生成路径集合 T，并从 B 集合中删除。这样类似于降低了对路径 T_i 的要求，使得原来的路径 T_i 有了更大的机会成为符合生成规则的路径。若路径 T_i 的某节点不是符合既定结构的节点，那么用已经判断得到的内部符合生成路径上的 m 个复合节点依次替换 T_i 路径上不符合既定结构的该节点。将替换后得到的所有更换过该节点的新路径重新加入到可能的路径集合 B 中。不断重复上述操作，外层操作为每条可能路径，内层操作为每条可能路径上的每个节点。

1)选择结构的内部路径生成

一个选择结构包含条件语句(即 if-then-else 或 switch-case)，其不同取值使得不同活动发生。在选择结构中，假定从分支结点 x 发出 N 条分支，每一条分支可以看作一个顺序结构，每条分支出现的概率之和为 1。分支结构生成的内部路径数为分支结点发出的分支数。3 种不同类型的选择结构展开生成的内部路径不同之处在于：不匹配选择结构生成的所有路径最后的结点为终止结点，匹配选择结构生成的所有路径的最后结点为合并结点，而 N 中取 K 匹配选择生成的所有路径中，有 K 条路径的最后结点为合并结点，剩余 $N-K$ 条路径的最后结点为终止结点。

2)循环结构的内部路径生成

对于循环结构，前测回路或后测回路都可以看作为一类特殊的选择结构，即从循环结构的分支结点出发，至少有两条转移路径，一是回到循环结构的入口结点，一是退出循环进入下一个结点，因此，循环结构生成的内部路径至少有两条。实际上，可以根据循环次数 l，将循环结构扩展为图 3 所示的分支结构形式。

然后根据选择结构的内部路径生成方法生成循环次数为 l 的循环结构的内部路径，生成的路径数为 $l+1$。为了避免天文数字的循环场景，限制了循环的次数，本文统一采用二次循环的策略，以提高算法处理活动图中各种复杂结构的能力。

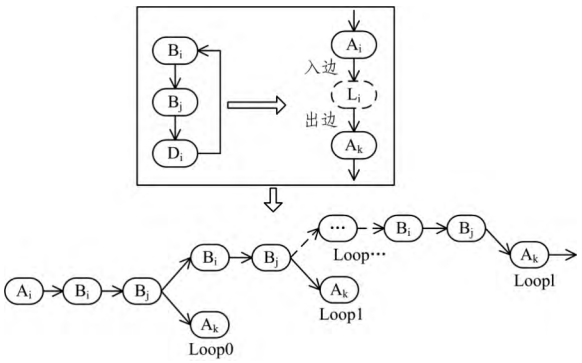

图 3　循环结构的展开

3)分岔结构的内部路径生成

UML 活动图中分岔结构包括同步分岔结构和合并分岔结构。利用分岔和汇合结点表示软件系统中存在并发的情况,也即系统中在同一时刻可能有多个构件在运行。活动图的同步分岔中分岔结点把一个单独的控制流分成两个或更多的并发控制流。一个分岔结点可以有 1 个进入转移和 2 个或更多的离去转移,每一个离去转移表示一个独立的控制流。在这个分岔之下,与每一个路径相关的活动将并行地继续。因此,一般来讲,活动图表示的并发结构中的各个分支之间都是相互独立、无交互存在的,只有当所有的并发分支都成功执行后,并发片段才算成功执行,所以,这种情况下同步分岔结构的内部路径可以看作是所有并发分支的顺序结构,因此,如果同步分岔结构中的活动可以按任意的顺序排列,对于包含多个并发活动的流程,可能出现活动组合数量爆炸的问题,假设有 n 个独立进程并发,每个进程包含的活动数目为 m_i,$i=1,2,\cdots,n$,则生成基本路径数为

$$(\sum_{i=1}^{n} m_i)\ !/(\prod_{i=1}^{n} m_i\ !) \tag{2}$$

依据上述原理进行排列组合生成的完全测试场景中有相当的部分是不合理的,因此常对并发活动加以约束来减少不合理的内部路径,如:测试人员定制约束条件(如活动 a 必须在活动 b 之前发生);对活动图的动作分类,按照活动的重要性选择排列组合的顺序;制定活动的权值,在生成测试场景时,权值大的活动优先选择;测试人员制定生成测试场景的数量。根据这些约束条件过滤掉不合理的内部路径,减少内部路径的数量。所以,这种情况下同步分岔结构的内部路径可以看作是所有并发分支的顺序结构,因此,并发片段的可靠性就可以看作是所有并发分支可靠性的乘积。从而根据测试路径估计基于构件的软件系统可靠性时,对于并发分岔结构生成的内部路径只需 1 条即可。

2.5　测试场景向系统"进程"转换

活动图主要用来对系统的一项功能进行描述,其中每个动作都由相应的构件来执行完成。因此,动作和动作间的一次转移就对应了相应的构件和构件间的一次转移。找出动作对应的构件,根据动作和动作间的转移关系,就可以得到一个构件和构件间的转移关系。动作间转移的概率就是当前活动发生的条件下,构件和构件间转移的概率。

根据每个活动结点中标注的信息,将活动图中的各个结点转换为对应的构件,控制结点(如分支节点等)一般不会直接与某个构件对应,也可以把它们看成其前驱结点对应的构件的行为,所以转换时不再单独考虑控制结点。活动图中的初始结点和终止结点分别转换为构件转移图中的起始结点和终止结点。原有的活动结点间的转移就映射为相应的构件和构件间的转移,概率保持不变,这样就能得到一个与活动图对应的局部构件转移图。

2.6　系统可靠性估计

Step 1　找出 UML 活动图中所有的最小区域,用其等价的组合结点代替,然后继续查找含有组合结点的 UML 活动图中的最小区域,直到没有最小区域存在为止;

Step 2　通过上面的简化后活动图生成基本测试路径 t_b,基本测试路径 t_b 中可能包含 0 个或多个带嵌套或不带嵌套的组合结点,然后对每个组合结点进行扩展,用组合结点的内部路径替代相应的组合结点,直到所生成的路径集中不再含有组合结点为止,形成初始路径集;

Step 3　对初始路径集进行约简,排除无效测试路径;

Step 4　根据每条有效路径中包含的组件及其转移概率信息,计算每个测试场景可靠性,根据各个测试场景可靠性估计出系统可靠性。

1)计算每个测试场景的可靠性。根据每个测试场景(一条组件序列)中各组件的可靠性信息,计算每个场景的可靠性。假设场景序列 S_i 为 $C_0,C_1,\cdots,C_n$,若组件 C_i 的可靠性为 $R(c_i)$,$i=1,2,\cdots,n$,则该场景的可靠性计算如下:

$$R_{S_i}=\prod_{i=1}^{n} R(c_i) \tag{3}$$

2)计算系统可靠性,各场景的可靠性及其发生概率 P_i 的乘积和就是系统的可靠性,即:

$$R_S=\sum_{i=1}^{m} P_i R_{S_i} \tag{4}$$

其中构件可靠性 $R(c_i)$ 应根据其类型采用不同方法来确定。一般地,构件可分为自主开发的构件和第三方构件。对于自主开发的构件,由于了解其内部实现,可靠性估计可采用传统方法进行,因而对于可靠性的分析主要集中在第三方构件上。目前,对于独立构件的性能估计主要采用第三方认证的方法解决。在此,假设独立构件的可靠性可通过第三方认证之后,以公式或相关函数的形式计算得知。

3　实例分析

为了更好地说明本文所提方法的有效性和实用性,这里以两个实际的应用程序作为具体的应用实例进行分析,实例 1 是一个医疗会诊系统[16],实例 2 是一个邮件派发系统。

3.1　实例 1

实例 1 为一个医疗会诊系统。患者要有一个私有帐户,必须提供用户名和密码才可登录,登录后患者可以选择查看病历和挂号。患者挂号后,医生在了解患者病情之后,给出治疗建议。

结点转移概率性标注信息如表 1 所列。

表 1　结点转移概率性标注

转出结点	转入结点	转移概率	转出结点	转入结点	转移概率	转出结点	转入结点	转移概率
A	B	1.0000	B	A	0.2567	D	E	0.3417
C	D	1.0000	B	C	0.3754	D	F	0.4571
H	G	1.0000	B	D	0.3679	D	E、F	0.2012

其活动图如图 4 所示。

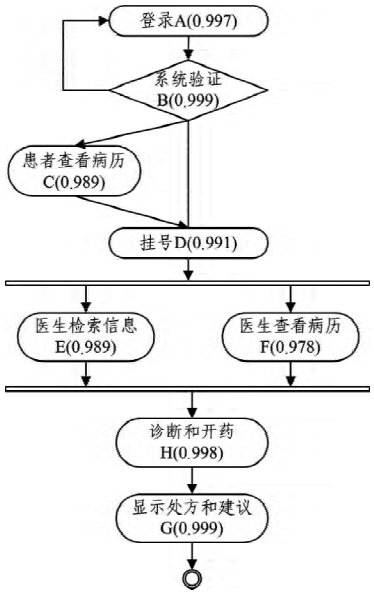

图 4　系统活动图

根据中间可测模型生成测试场景和测试路径，本例共生成 12 条测试路径，每条路径的概率标注信息如表 2 所列。

表 2　每条路径的发生概率

路径	发生概率
A-B-A-B-D-E-H-G	0.03229
A-B-A-B-D-F-H-G	0.04317
A-B-A-B-D-(E,F)-H-G	0.01900
A-B-A-B-C-D-E-H-G	0.03293
A-B-A-B-C-D-F-H-G	0.04317
A-B-A-B-C-D-(E,F)-H-G	0.01937
A-B-D-E-H-G	0.12571
A-B-D-F-H-G	0.16816
A-B-D-(E,F)-H-G	0.07402
A-B-C-D-E-H-G	0.12827
A-B-C-D-F-H-G	0.17160
A-B-C-D-(E,F)-H-G	0.07553

根据计算出的每条路径的可靠性，由式(4)可得该医疗系统的可靠性为 0.9553。

3.2　实例 2

实例 2 为一个邮件派发系统，邮件由邮局发出，经过某种路线到达地方总站，各地方总站对邮件进行汇总，当邮件到达目的地后，对每个邮件进行单独派发。系统可以抽象为 42 个节点，其活动图如图 5 所示。

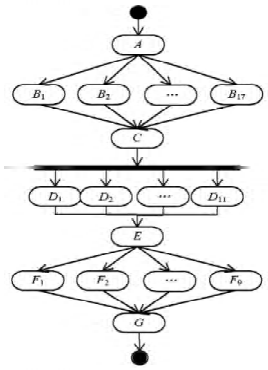

图 5　系统活动图

为了说明本方法的实用性，这里采用本文提出的方法和随机生成的方法[17]估计该系统的可靠性。随机生成方法主要是根据各个功能使用频率，随机仿真邮件的派发，分别以每次随机生成 100、1000、10000 条测试路径 3 种情形，每种情形分别仿真 50 次，计算出每次实验的可靠性估计值，得到的平均可靠性分别为 0.856414、0.846438、0.84403。采用本文提出的方法估计出该系统的平均可靠性为 0.8438，两种方法拟合结果如图 6 所示。

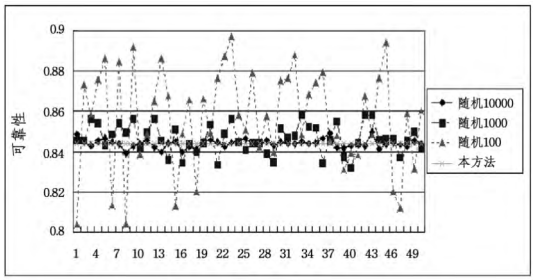

图 6　可靠性数据散点图

由图 6 可知，采用随机生成方法，估计出的可靠性波动性较大，其方差基本保持在 10^{-3} 数量级上，适合一般软件的可靠性估计，且随着生成路径数增加，其估计值越来越趋近平稳，随着随机生成的路径越多，其仿真开销越大。而本文提出的方法则很稳定，其方差基本保持在 10^{-9} 数据级以下，适合安全关键高可靠软件的可靠性评估。

结束语　软件可靠性预测是保证软件质量的重要方法，本文以构件化的软件开发方法为背景，提出了一种能自动将 UML 活动图转换为模型 Markov 链的可靠性预测方法。该方法基于构件化的软件体系结构，从 UML 的活动图和顺序图出发，通过构造一个名为控制结构转移图的中间模型，将标注了可靠性信息的 UML 模型转换为 Markov 链，并通过递归方法自动生成测试路径，然后依据每条路径的可靠性信息来估计整个软件系统的可靠性。实验表明，不论是简单的还是复杂的例子，通过 UML 活动图转换的方式可以直接被现有分析方法使用来进行软件可靠性评估，从而使分析变得高效和模型化，依据本文提出的测试可靠性评估方法所得到的系统可靠性评估数据，在精度结果上较其他方法有所提高，能显著提高软件估计的精度和评估效率。

参 考 文 献

[1] Chao-Jung H, Huang Chin-yu. An Adaptive Reliability Analysis Using Path Testing for Complex Component-Based Software Systems[J]. IEEE Transactions on Reliability, 2011, 60(1): 158-170

[2] 柳毅，麻志毅，何啸，等. 一种从 UML 模型到可靠性分析模型的转换方法[J]. 软件学报，2010(2)：287-304

[3] 陆文，徐锋，吕建. 一种开放环境下的软件可靠性评估方法[J]. 软件学报，2010，33(3)：452-462

[4] 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

[5] Kundu D, Sarma M, Samanta D. A novel approach to system testing and reliability assessment using use case model[C]// Proceedings of the 1st India Software Engineering Conference. Hyderabad, India: ACM, 2008: 147-148

(下转第 560 页)

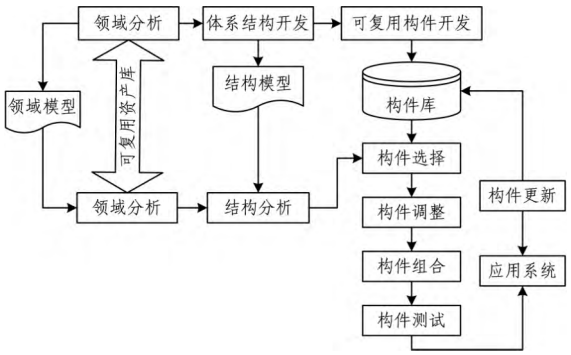

图 3　敏捷开发过程

以常见的天气预报 APPS 为例，在规划 APPS 开发风险之初，拆分开发所需资源，并进行分类，发现此类 APPS 所需界面元素、整体架构、细分功能可基本复用于此平台，在确定接口的基础上，组建小型开发团队，分工进行界面设计、构件集成、功能实现等环节。此种开发形式可实现快速的 APPS 开发，在此基础上，进入程序测试环节，与此同时，开发团队将更新后的构件置于可复用平台，并完善文档，此时的构件可应用于下一次的 APPS 开发。

结束语　为提高移动应用程序开发的效率与质量，减少大量重复的构件设计过程，本文基于移动应用开发平台，设计了一种跨平台的快速 APPS 开发模式，采用可复用技术，同时结合敏捷开发模式，帮助软件开发人员快速、高效、高品质地完成程序开发，与此同时，利用具体的开发流程验证此平台界面生成器的使用效果，并验证平台中大量构件功能的正确性。

本文还介绍了基于可复用技术的 APPS 开发规范和标准，基于此，介绍了 APPS 的开发流程，可帮助开发人员快速且正确地掌握此平台的使用方法。

移动应用开发方向目前正处于不断完善和规范的阶段，大量不稳定且界面欠佳的软件被逐步淘汰，本文介绍的跨平台的快速 APPS 开发模式将帮助开发公司提高软件质量，提升市场的占有率。然而此种开发模式目前仍存在一定的局限性，只适用于可重复开发的 APPS 资源，且平台中构件的数量和功能的完备性有待提高，这些问题将在后期的项目开发中进一步完善。

参 考 文 献

[1] Guo J, Lu Q. A survey of software reuse repositories[C]// Proceeding of IEEE International Conference and Workshop on the Engineering of Computer Systems(ECBS2000). 2000:92-100

[2] Almeida, Alvaro, Garcia. Designing Domain-Specific Software Architecture(DSSA): Towards a New Approach [C] // The Working IEEE/IFIP Conference Software Architecture(WICSA'07). 2007:30

[3] Shirogane J, Iwata H, Fukaya K, et al. Support Method for Changing GUIs according to Roles of Widgets and Change Patterns[C]// IMECS. 2007:1098-1103

[4] Schobbens P Y, Heymans P, Trigaux J C. Feature diagrams: A survey and a formal semantics[C]// Glinz M, Lutz R, eds. Proceedings of 14th International Requirements Engineering Conference. 2006:139-148

[5] Benavides D, Segura S, Ruiz-Cortés A. Automated analysis of feature models 20 years later: A literature review [J]. Inform Syst, 2010, 35(6): 615-636

[6] 杨芙清. 软件复用及相关技术[J]. 计算机科学, 1999, 26(5): 1-4

[7] Czarnecki K, Helsen S, Eisenecker U. Formalizing cardinality-based feature models and their specialization [J]. Software Process Improve Pract, 2005, 10(1): 7

[8] 杨燕燕, 梅宏, 陈海文. 数据仓库技术和可复用构件库系统[J]. 计算机科学, 1999, 26(5): 56-60

[9] Clarke D, Helvensteijn M, Schaefer I. Abstract delta modeling [C] // Visser E, Järvi J. eds. Proceedings of 9th International Conference on Generative Programming and Component Engineering. 2010: 13-22

[10] 彭鑫, 赵文耘, 肖君. 基于本体的构件描述和检索[J]. 南京大学学报: 自然科学版, 2005, 41(Z1): 470-476

[11] 梅宏, 刘譞哲. 互联网时代的软件技术: 现状与趋势[J]. 科学通报, 2010, 55(13): 1214-1220

[12] Frakes W B, Kang K C. Software reuse research: Status and future[J]. IEEE T Softw Eng, 2006, 31(7): 529-536

[13] Apel S, Kastner C. An overview of feature-oriented software development [J]. J Object Tech, 2009, 8(5): 49-84

(上接第 536 页)

[6] Rodrigues G, Rosenblum D, Uchitel S. Using Scenarios to Predict the Reliability of Concurrent Component-Based Software Systems[M]. Fundamental Approaches to Software Engineering, Cerioli M, Springer Berlin/Heidelberg, 2005

[7] Mohanta S, Vinod G, Ghosh A K, et al. An approach for early prediction of software reliability [J]. SIGSOFT Softw. Eng. Notes., 2010, 35(6): 1-9

[8] Cortellessa V, Singh H, Cukic B. Early reliability assessment of UML based software models[C]// Proceedings of the 3rd International Workshop on Software and Performance. Rome, Italy: ACM, 2002: 302-309

[9] 颜炯, 王戟, 陈火旺. 基于 UML 的软件 Markov 链使用模型构造研究[J]. 软件学报, 2005, 16(8): 1386-1394

[10] Böhr F. Model Based Statistical Testing and Concurrent Streams of use[C] // 3rd Workshop on Model-based Testing in Practice6th European Conference on Modelling Foundations and Applications(ECMFA 2010). 2010: 41-50

[11] Luo Y, Ben K. Scenario-Based Early Reliability Model for Distributed Software[C]// 2010 Fifth International Conference on Frontier of Computer Science and Technology. 2010: 201-207

[12] Cheung L, Roshandel R, Medvidovic N, et al. Early prediction of software component reliability[C]// Proceedings of the 30th International Conference on Software Engineering. Leipzig, Germany: ACM, 2008: 111-120

[13] Nayak A, Samanta D. Synthesis of test scenarios using UML activity diagrams[J]. Software and Systems Modeling, 2011, 10(1): 63-89

[14] Musa J D. Operational profile in softw are-reliability engineering [J]. IEEE Trans Software, 1993, 10(2): 14-32

[15] Ouabdesselam F, Parissis I. Constructing operational profiles for synchronous critical software[C]// Proceedings of 6th International Symposium on Software Reliability Engineering. Los Alamitos, USA: IEEE Computer Society, 1995: 286-293

[16] Priya S S. Test Path Generation Using UML Sequence Diagram [J]. International Journal of Advanced Research in Computer Science and Software Engineering, 2013, 3(4): 123-134

[17] 张德平, 聂长海, 徐宝文. 软件可靠性评估的重要抽样方法研究[J]. 软件学报, 2009, 20(10): 2859-2866