UML 模型驱动的划分测试用例生成方法研究*

王瑞雪[1,2]，张 涛[2]

（1. 中国科学院大学，北京 100049；2. 中国科学院空间科学与应用总体部，北京 100094）

摘 要：测试用例的自动生成是软件测试研究的主要方向之一。针对现有的 UML 模型驱动测试方法在测试数据生成方面存在低效、无目的性、冗余等问题，提出了基于 UML 活动图模型驱动的划分测试用例自动生成方法。该方法将测试场景归为五种类型，并为每种类型规划了测试数据取值范围和选择方法。构建了基于该方法的原型工具软件，提供被测软件原模型和数据约束即可生成测试用例。实验结果表明，该方法能够在不降低测试覆盖率的情况下，能生成数量少、针对性强的测试用例集。

关键词：软件测试；测试用例生成；UML 活动图；划分测试

中图分类号：TP311　　**文献标志码**：A　　**文章编号**：1001-3695(2012)09-3334-04

doi:10.3969/j.issn.1001-3695.2012.09.035

Using partition method to generate test cases from UML diagrams

WANG Rui-xue[1,2], ZHANG Tao[2]

(1. *University of Chinese Academy of Sciences*, *Beijing* 100049, *China*; 2. *General Establishment of Space Science & Application*, *Chinese Academy of Sciences*, *Beijing* 100094, *China*)

Abstract: Generating test cases automatically is one of the main directions of the software testing research. There are a few problems such as inefficient data, no purpose, redundant, in the existing UML model-driven test methods. This paper proposed a new method to generate test case based on UML model-driven, which used partition theory to classify test scenarios into five types, and designed different ranges and ways to choose test data. And also designed a tool which realized this method. And demonstrated a tool to realize this method, which could generate test case automatically by supplied UML diagrams and data constraints of the software under test. Experimental result shows that the test cases get by this method are smaller number and targeted.

Key words: software testing; test cases generation; UML activity diagrams; partition testing

0 引言

据统计，在软件测试的全部开销中，约 40% 的花费消耗在设计测试用例阶段。因此，如何科学有效地设计测试用例自动生成算法，依据软件规格或程序结构自动构造测试用例已成为软件测试的研究重点[1]，其中，基于 UML 模型驱动测试用例生成的方法为研究方向之一。

基于 UML 模型，无论是活动图模型、顺序图模型还是状态图模型，生成的测试用例主要包括测试场景和测试数据。测试场景的生成方面得到了深入的研究，如针对 UML 活动图模型提出了灰盒算法[2,3]、反蚁群算法[4,5]、自适应算法[6]等路径生成算法；规定了测试覆盖率标准用于约束路径生成算法来达到相应覆盖率；还特别研究了活动图中并发模块的分类方法、解析方法和状态空间爆炸问题[7]。测试数据的生成则未得到同样的重视，以往的研究中大多根据测试场景的参数约束，使用统一的等价类、边界值、决策表等测试方法选择测试数据。由于不同的测试场景其实现的系统功能和解决的问题不同，采用统一的方法选择测试数据会导致一些场景的测试数据不够充分，而另一些场景的测试数据缺少针对性，存在大量重复的无效和冗余数据。为此，如何判断测试场景的执行目的，并有针对性地选择测试数据成为本文的研究重点。

划分测试[8]作为一种系统的软件测试方法，近年来在学术界及实际应用领域引起了广泛关注。划分测试的一个主要研究方面就是划分策略问题。划分策略由子域划分方案和测试用例选择方案两方面组成。子域划分方案主要解决以何种方式将系统输入域划分成多少个子域的问题，而测试选择方案解决在每个子域中如何合理选取测试用例进行测试以达到最佳测试效果的问题。本文采用划分测试的思想，从数据的角度定义测试场景的类型和测试数据的选择方法，在此基础上设计了基于 UML 活动图模型驱动的划分测试用例生成方法，并构建了相应原型工具软件。

1 方法设计

1.1 测试场景类型定义

测试场景类型的定义依据系统输入域在不同划分标准下所得的数据空间之间的关系。系统需求说明中规定的参数定义域和依赖关系是一种划分标准，它将输入域划分为三部分，即合理数据子域(valid domain，VD)、非法数据子域(invalid domain，IVD)和冲突子域(conflict domain，CD)，如图 1 所示。参数的定义域集合将输入域分为系统能够处理的数据空间和不

收稿日期：2012-03-01；**修回日期**：2012-04-12　　**基金项目**：中国科学院国防科技创新基金资助项目(CXJJ-11-Q74)

作者简介：王瑞雪(1986-)，女，黑龙江绥化人，硕士，主要研究方向为高可靠软件(snow.ruixue@163.com)；张涛(1972-)，男，研究员，博导，博士，主要研究方向为高可靠软件.

能处理的数据空间 IVD;在能够处理的空间内,依据参数依赖关系分为系统允许出现的数据组合 VD 和系统不允许或无意义的数据组合空间 CD。

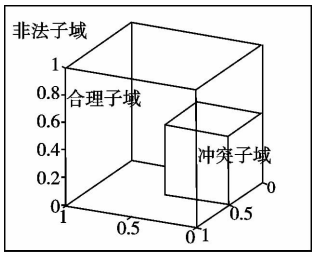

图1　系统输入子域划分方案

测试场景在执行过程中对参数的约束,如分支判定条件、循环判定条件、异常处理条件等,将形成该测试场景最终能够处理的输入参数空间——测试场景子域(test scene domain, TSD)。以解析 UML 活动图模型获得的测试场景作为划分标准,将输入域划分为 N 个测试场景子域(假设测试场景数目为 N)。场景类型定义如下:

a)空集型。若某个测试场景的 TSD 满足 TSD = ∅,则该场景定义为空集型;

b)越界型。若某个测试场景的 TSD 满足 TSD∩IVD≠∅,则该场景定义为越界型;

c)冲突型。若某个测试场景的 TSD 满足 TSD∩CD≠∅,则该场景定义为冲突型;

d)常规型。若某个测试场景的 TSD 满足:(TSD⊆VD)∧(TSD≠∅),则该场景定义为常规型;

e)虚拟型。若存在这样的数据空间($\mathrm{TSD}_v = D - \bigcup_{i=1}^{N} \mathrm{TSD}_i$)∧($\mathrm{TS}D_v \neq \varnothing$),则向测试场景集合中添加新的测试场景,且该场景子域为 TSD_v。

以上的定义中有一种情况需要说明,若有某个测试场景满足关系(TSD∩IVD≠∅)∧(TSD∩CD≠∅),则定义该测试场景类型为越界型。

1.2　测试数据选择方案

由以上的定义可知,根据测试场景的类型可以判断测试场景的执行目的,进而选择合适的测试数据加以检测。具体如下:

a)空集型。测试场景中对参数的约束存在矛盾导致的数据。当解析到空集型场景时,表示模型中存在相互矛盾的参数约束,为此需要终止模型的解析,并根据该场景检查系统模型和需求说明。

b)越界型。处理的数据全部或部分变量的取值不在参数定义域内,该类场景的执行目的是确保软件能够识别和处理异常的输入数据,保证软件的正常运行,为此需要在测试子域和非法子域的交集部分产生异常输入数据作为测试数据。

c)冲突型。处理那些系统限制的参数组合,该类场景的执行目的是保证系统在运行的过程中不会进入被禁止的状态,为此,只需要在测试子域与冲突子域相交的数据域内,随机地产生测试数据即可验证该场景的处理过程。

d)常规型。处理那些处于合理子域中的数据,该类场景的执行目的是实现软件需求中的功能,完成计算算法或控制流程。因为各种软件实现的功能不同,使用的算法也不同,所以需要为这种类型的场景生成比较全面的测试数据,包括单边界值、多边界值、带有时间约束的测试值和基于决策表的组合值等。

e)虚拟型。该类型没有对应的真实测试场景,而是捕捉系统输入域的漏洞,或许是由于系统需求的缺陷,又或许是由于系统模型的错误造成的,为此,只需要在虚拟型测试子域内随机地产生测试数据来检测系统软件对这些漏洞的处理方式。

综合以上的分析,根据测试场景的类型及其测试目的,测试用例的选择方案如表 1 所示。

表 1　测试数据选择方案

场景类型	数据取值空间	测试用例选择方法
空集型	空集	检查被测软件模型和需求
越界型	TSD∧IVD	随机生成法,边界值法
冲突型	TSD∧CD	随机生成法
常规型	TSD	边界值法、因果图法、组合测试法等
虚拟型	TSD_v	随机生成法

1.3　UML 模型驱动测试框架

UML 模型驱动测试框架如图 2 所示,共分为以下四个部分:模型建立、模型解析、数据域划分和测试数据生成。

a)根据软件需求说明构造被测软件的 UML 活动图模型,提取需求中的参数定义域;b)解析 UML 活动图模型,生成被测软件的测试场景集合;c)分别根据测试场景数据约束和参数定义域划分被测软件输入域;d)判断每个测试场景的类型,计算测试数据取值空间,根据相应的方案生成测试数据。

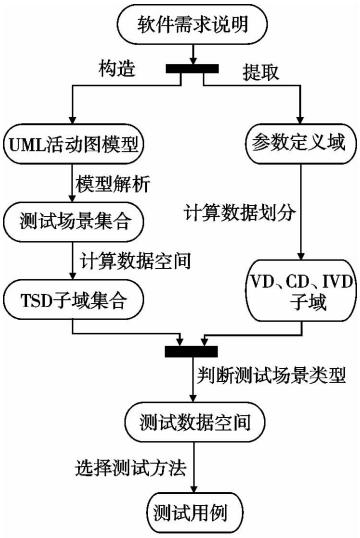

图2　UML活动图模型驱动测试框架

2　方法实现

2.1　UML 活动图模型建立

UML 活动视图因其能够表示系统的业务流程、工作流和交互场景而适用于描述系统中各组成部分之间的相互关系,因此选择 UML 活动视图建立软件测试模型。本文中建模工具选用 Enterprise Architect[9] 软件,该软件能够将用户设计的 UML 模型转换为 XML 格式文档,便于数据的存储和解析。

值得注意的是,测试不同于开发,它更多地关注系统功能的正确性、系统的可靠性、稳定性、抗压性和容错能力等,而较少关注算法的详细实现过程。因此,相比于开发模型,测试模型应具有更多的异常处理模块和精简的功能实现模块。

2.2　UML 活动图模型解析

活动图模型解析分为两个部分:a)从 XML 文档中提取模型信息;b)根据这些信息设计算法生成测试场景。

2.2.1 XML 文档的解析

XML 文档的解析主要有两种方法:SAX 和 DOM。DOM 解析方法把 XML 文档转换为一个包含其内容的树,并可以对树进行遍历。该方法的优点是可以很容易地添加和修改树中的元素;缺点是因其需要解析整个文档,导致对计算机的性能和内存要求比较高。SAX 解析方法视 XML 文档为流媒体,分析能够立即开始,而不是等待所有的数据被处理,并且只是在读取数据时检查数据,不需要将数据存储在内存中。该方法的优点是解析速度快,对计算机的性能要求低;缺点是添加和修改树中的元素不容易。

随着软件规模越来越大,对应的活动图模型的数据量也十分庞大,而且考虑到在测试过程中不需要对模型作修改,所以本文选用 SAX 方法从 XML 文档中提取模型信息。

2.2.2 测试场景的生成

UML 活动图不仅能够表示顺序执行过程,而且能够表示并发执行过程。为此,模型解析需要处理两种类型的模块,即并发模块和顺序模块。并发模块作为整个模型的一部分,可以将其看做是仅有一个入迁移(fork 节点入迁移)和出迁移(join 节点出迁移)的黑盒子活动节点,从而 UML 活动图被分为两个层次,整体为一个类似的有向图,包括普通活动节点和黑盒子活动节点。黑盒子节点内为并发模块,包括入口节点、出口节点和并发流程。

1)并发模块的处理方式 本文中采用“随机生成过滤法”[10]来计算并发模块的执行路径。对于并发模块中的每个节点(包括 fork 和 join 节点),定义了两个变量:TW 和 W。其中 W 是该节点的权重,TW 是该节点的计算权值。具体算法如下:

a)采用随机数生成算法赋予每个节点的 W 变量一个随机值。

b)计算并发模块中每个节点的 TW 值,$TW(A) = W(A) + \max\{TW(A_{out})\}$。其中,$A$ 表示并发模块中的节点,$\max\{TW(A_{out})\}$ 表示 A 节点的所有出迁移目标节点中 TW 的最大值。

c)将并发模块中所有节点按 TW 值从大到小排列,即可得到一个并发模块的执行路径。

d)如果用户有约束条件(如节点 A 要在 B 之前执行),则根据条件检查执行路径是否符合要求,并进行过滤。

e)对上述过程循环多次,就可以得到满足条件的、指定数目的并发模块执行路径。

公式 $TW(A) = W(A) + \max\{TW(A_{out})\}$ 保证了不同层次上的并发节点执行顺序的先后;随机的 W 值保证了同一层次上的并发节点执行顺序的随机性(或并发性)。每次循环只生成一条执行路径,避免了并发状态空间爆炸问题,并且算法中可以设计过滤函数生成具有部分指定顺序的执行路径,有利于获得特殊情况下的测试场景。

2)整体的处理方式 本文中整体的处理方式主要考虑分支、循环和覆盖率问题。以下提出的算法能够有效地覆盖所有测试场景并控制循环的执行次数。其算法如下:

a)使用堆栈作为测试场景的中间结构,所有的测试场景存储在堆栈集中,从每个栈底到栈顶读出数据作为一个测试场景。

b)从初始状态开始按照 DSP 算法遍历活动图,获得当前活动,其入迁移数目为 IN_COUNT,出迁移数目为 OUT_COUNT。若 IN_COUNT = 0,则创建新的堆栈将其放入栈顶,并复制 OUT_COUNT - 1 个该堆栈在堆栈集中;若 IN_COUNT > 0,则在堆栈集中寻找栈顶为其前置节点的堆栈 S,将其放入 S 栈顶,并复制 OUT_COUNT - 1 个 S 在堆栈集中。

c)循环执行次数 N 放置在节点信息当中,如果遇到循环,活动节点访问次数达到 $N + 1$ 次,则进入下一个可触发的迁移。

d)黑盒子活动节点展开为“随机生成过滤法”计算的并发执行路径。

2.3 数据域划分

输入数据域的划分主要有两种策略,即根据参数定义域划分和根据测试场景约束划分。其中,每个参数的数据约束用 Condition 类实例表示,定义如下:

```
enum Type{INT,DOUBLE,ENUM};
public class Condition {
  private String name;
  private Type type;
  private String domain;
  private int precision;
  ……
  public void parseDomain(){};
}
```

其中:name 表示参数的名称;type 表示参数的类型(整型、浮点型、枚举型);domain 表示参数的取值范围,如[16,100);precision 表示参数的精度,如 1 表示精确到小数点后一位;parseDomain()是解析 domain 字段数学含义的方法。TSD、IVD、CD、VD 子域为装载 Condition 类实例的哈希表,定义为 HashMap〈key,value〉domain,其中 key 为 condition.name,value 为 Condition 实例。

根据参数定义域的划分算法如下(x 为 n 维数据向量,数据约束来自系统需求说明):

a)将所有变量数据约束整理为参数定义域空间 D,则

$$IVD = \{x \mid x \notin D\}$$

b)由参数依赖关系生成返回布尔值的依赖判定函数 $fun(x)$,则

$$CD = \{x \mid x \in D \cap fun(x) = \text{false}\},$$
$$VD = \{x \mid x \in D \cap fun(x) = \text{true}\}。$$

根据测试场景的划分算法如下(数据约束来自测试场景中活动或迁移上的约束条件):

a)从测试场景集合中取出一个场景,为其生成一个空的 HashMap〈key,value〉domain;

b)从测试场景初始活动开始,当获取一个参数约束条件时,为其生成相应的 Condition 类实例 c:

(a)若 domain 中没有与 c 同名的参数,则将 c 放入 domain 之中;

(b)若 domain 中有与 c 同名的参数 d,则计算 c 与 d 的 domain 字段的交集,如果计算结果为空集,那么定义该场景为空集型,划分停止;如果结果不为空,将结果写入 d 的 domain 字段。

(c)检查 domain 中是否包含了所有系统参数,如果有缺少的参数,则提示用户采用以下两种方法处理:

①使用需求中的参数定义域作为该参数在这个测试场景中的约束;

②停止数据划分过程,检查 UML 活动图模型中该测试场景的参数约束条件。

2.4 测试数据生成

测试数据的生成主要是判定测试场景的类型,再根据表 1

中给出的方案计算数据取值区间和选择测试数据。涉及的算法主要为集合的相交运算(虚拟型中并集也可转换为交集计算)。

相交判断:如果存在某个参数 x,其在 A 子域的 domain 字段与在 B 子域的 domain 字段交集不为空,那么 A、B 两个子域相交。交集计算:取子域 A,B 中所有同名的参数对,计算每对的交集,并将结果写入子域 C,则 C 为 A、B 子域的交集。

3　实验验证

本文构造了基于上述模型驱动测试方法的原型工具软件,该软件只需提供被测软件原模型和测试数据约束即可生成测试用例。其中,UML 活动图以 XML 文档的格式导入,测试数据约束分为区间类型约束、枚举类型约束和组合约束,分别对应软件输入域中的连续参数、离散参数和参数之间的依赖关系,生成的测试用例存储在数据库中,分为测试场景和测试数据两部分。软件操作界面如图 3 所示。

图3　工具软件操作界面

选取的被测软件为保险金计算软件[11],该软件中根据投保人年龄(age),投保人驾照点数(pot)和违规次数(num)来计算保险金。连续参数有 age、pot,离散参数有 num,num 和 pot 之间存在依赖关系(图 3)。该软件的 UML 活动图如图 4 所示。

测试用例如图 5 所示,上半部显示了该软件的所有 11 条测试场景和相应的 TSD 子域(图 5 中 conditions 列),下半部显示了第 3 个测试场景对应的测试数据,由测试数据类型得出该场景类型为冲突型,包括了用于检验冲突处理的 F 类数据。

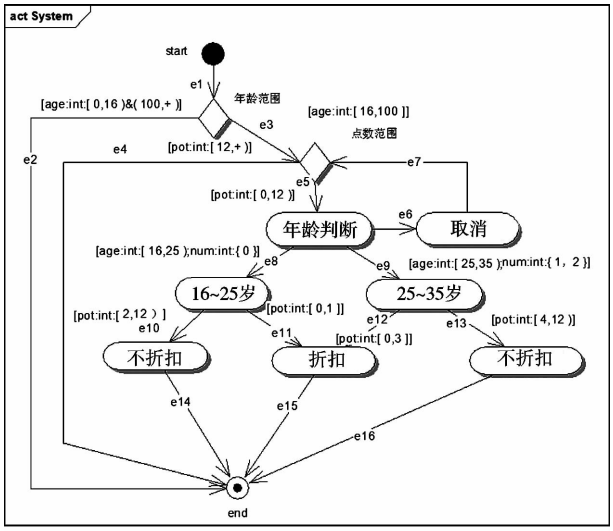

图4　保险金计算软件UML活动图模型

图5　数据库中测试用例结果

表 2 为原保险金计算软件的测试用例数量统计,原软件中只有两个连续变量 age、pot,且无参数依赖关系约束,其中采用弱边界方法生成的测试用例有 21 个,强边界方法有 273 个,等价类方法 25 个,决策表 10 个,若使用全部三种方法,则测试用例数目最少为 56 个,最多为 308 个。而本次实验在更多变量约束和更复杂处理情况下生成的测试用例只需 43 个,并且覆盖了软件当中的所有执行路径。

表 2　原保险计算软件测试用例统计

弱边界值	强边界值	等价类	决策表
21	273	25	10

实验结果表明,该工具软件能够检测出 UML 活动图的所有测试场景,并准确地判断场景的类型,根据不同的场景类型生成规模小、质量高、具有针对性的测试用例。

4　结束语

测试用例的设计是软件测试的核心和关键,测试用例的自动生成能够减少软件测试时间、提高软件测试质量。本文提出了基于 UML 活动图模型驱动的划分测试用例自动生成方法。该方法中采用“随机生成过滤法”和改进的深度优先搜索算法生成测试场景;从划分测试的角度出发,根据不同的标准划分了软件输入域,并在此基础上定义了测试场景的五种类型,为不同类型场景设计了相应的测试数据生成方法,最后得到由测试场景和测试数据组成的测试用例。最后以基于该方法实现的工具软件进行了实验验证,结果表明,该方法能够有效地解析模型、划分软件输入域,并选择适合的方法生成规模小、覆盖率高、针对性强的测试用例集合。

参考文献:

[1] 聂鹏,耿技,秦志光. 软件测试用例自动生成算法综述[J]. 计算机应用研究,2012,29(2):401-405,413.

[2] 袁洁松,王林章,李宣东,等. UMLTGF:一个基于灰盒方法从 UML 活动图生成测试用例的工具[J]. 计算机研究与发展,2006,43(1):46-53.

[3] 张楣,刘超,孙昌爱. 基于 UML 活动图模型的测试用例生成技术研究[J]. 北京航空航天大学学报,2001,27(4):433-437.

[4] LI Huai-zhong,LAM C P. Using Anti-Ant-like agents to generate test threads from the UML diagrams[C]//Proc of the 17th International Conference on Testing of Communicating Systems. Montreal:Springer-Verlag,2005:69-80.

[5] 陈明师,刘晓洁,李涛. 基于多态蚁群算法的测试用例自动生成[J]. 计算机应用研究,2009,26(6):2347-2348.

[6] XU Dong, LI H, LAM C P. Using adaptive agents to automatically generate test scenarios from the UML activity diagrams[C]//Proc of the 12th Asia-Pacific Software Engineering Conference. Washington DC:IEEE Computer Society,2005:385-382.

[7] 高红梅,旭东,刘宗田. 基于 UML 活动图的测试研究进展[J]. 计算机科学,2008,35(2):263-267,281.

[8] 张德平,查日军. 划分测试用例选择的风险决策方法[J]. 计算机应用研究,2010,27(12):4536-4540.

[9] 赖信仁. UML 与 Enterprise Architect 7.5 团队开发实用手册[M]. 北京:电子工业出版社,2010.

[10] 粘新育. 基于 UML 活动图模型的测试用例生成方法研究[D]. 济南:山东大学,2007.

[11] JORGENSEN P C. 软件测试[M].2 版.韩柯,杜旭涛,译. 北京:机械工业出版社,2007.