◎产品、研发、测试◎

基于MDA的软件测试用例生成

蒋忠炜,张云华,谢轩昂
JIANG Zhong-wei,ZHANG Yun-hua,XIE Xuan-ang

浙江理工大学 信电学院,杭州 310033
College of Informatics & Electronics,Zhejiang Sci-Tech University,Hangzhou 310033,China
E-mail:zhongweijiang@msn.com

JIANG Zhong-wei,ZHANG Yun-hua,XIE Xuan-ang.MDA-based software test case generation.Computer Engineering and Applications,2007,43(17):97-99.

Abstract: This paper proposes an approach of applying the model driven code in MDA into automatic generation of model driven software test case.Based on UML/OCL model,we develop a framework of automatic generation on unit test case by adopting fault-based theory,mutant analysis technology and constraint handle rule,which improves the automatic level of software test and the efficiency of software development.

Key words: MDA;OCL;fault-based test;test case generation

摘 要:将MDA中模型驱动的软件代码自动化生成思想应用于模型驱动的软件测试用例自动化生成。从UML/OCL模型出发,采用缺陷测试理论、变异分析技术,结合约束处理规则,开发一个可以自动生成单元测试用例的框架,提高软件测试的自动化程度,从整体上提高软件的开发效率。

关键词:MDA;OCL;缺陷测试;测试用例生成
文章编号:1002-8331(2007)17-0097-03 **文献标识码**:A **中图分类号**:TP311

1 引言

MDA是OMG提出的软件体系结构方法学,是一种基于模型的软件开发方式。在MDA中,模型被定义为系统的结构、功能或行为的形式化规约。MDA以模型为中心,目标是要实现从UML企业模型到最终代码的自动生成[8]。

UML在MDA中起到极其重要的作用,是MDA的关键实现技术。UML作为一种通用的可视化建模语言,在学术界和工业界得到了广泛的应用,但UML仅用非形式化和半形式化的方法描述其语义,因此UML图形化的规格说明并不是精确和无歧义的,并且UML模型也不能精确地描述模型中各个对象之间的约束关系。正因如此,OMG推出了对象约束语言(OCL)[9],OCL设计的原始目的是描述系统中各个对象之间的约束关系,消除UML规格说明中的模糊性。使用UML/OCL建立的模型是相对完整和精确的,支持规格说明正确性的形式化验证,使基于精确模型的模拟和确认成为可能,为软件的自动化生成奠定了坚实的基础。

本课题借鉴MDA中软件自动化生成的思想和现有的代码自动生成的方法,将模型驱动的软件开发应用于模型驱动的软件测试,开发一个从UML/OCL模型自动生成单元测试用例的软件自动测试框架。

2 UML/OCL建模应用实例

下面这个例子介绍了UML/OCL在实际建模中的应用。在处理器调度系统中,有多个处理器随时准备被调度,还有另外的一些处理器在等待一些外部动作发生以进入到准备状态,同时系统中最多只能有一个处理器处于活动状态。该系统的静态模型如图1所示。

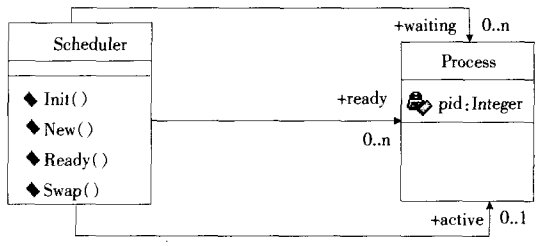

图1 处理器调度系统的静态模型

该系统还存在如下约束:在任何时候,等待处理器集合和准备处理器集合之间是不相交的;活动处理器既不在等待处理器集合,也不在准备处理器集合中;只有当没有准备处理器时,才可能会发生没有活动处理器。以OCL描述如下:

```
context Scheduler
inv:(ready->intersection(waiting))->isEmpty()and
    not ready->union(waiting)->includes(active)and
(active->isEmpty()implies ready->isEmpty())
context Scheduler::Init()
post:(ready->union(waiting))->isEmpty()and active->isEmpty()
context Scheduler::New(p: Process)
pre:p <> active and not((ready->union(waiting))->includes(p))
post:waiting=waiting@pre->including(p)and
    ready=ready@pre and active=active@pre
context Scheduler::Ready(p: Process)
```

```
pre:waiting->includes(p)
post:waiting=waiting@pre->excluding(p)and
    if active->isEmpty()then
        ready=ready@pre and active=p
    else
        ready=ready@pre->including(p)and active=active@pre
    endif
context Scheduler::Swap(p: Process)
pre:not active->isEmpty()
post:if ready@pre->isEmpty()then
        (active->isEmpty()and ready->isEmpty())
    else
        (ready@pre->excludes(active)and
        ready=ready@pre->excluding(active))
endif
and waiting=waiting@pre->including(active@pre)
```

3 测试用例生成理论

本节从测试用例的形式定义,介绍了基于缺陷测试[2]的用例生成算法,以及如何将该算法建模为人工智能中的约束满足问题(CSP)[3]的求解。

3.1 测试用例的形式定义

在测试驱动开发的理念中,测试用例就是软件设计的精化,软件的设计可以抽象为测试用例集[4]。

定义 1 精化(Refinement)

$D1\subseteq D2=df\ \forall v,w,\cdots\in A\cdot D2\Rightarrow D1$ 也可以表示为$[D_2\Rightarrow D_1]$

定义 2 设计的精化

$[(P_1\vdash Q_1)\Rightarrow(P2\vdash Q2)]$ iff $[P2\Rightarrow P1]$ and $[P2\wedge Q1]\Rightarrow Q2$

定义 3 测试用例

假设变量列表 v 和 v',输入时 v 的值为 i,输出时 v'的值为 o,测试用例的定义 $t_d(i,o)=df\ v=i\vdash v'=o$。

考虑到非确定型的程序,输出是一个有限集合 O,修改测试用例定义为:

$t'_d(i,O)=df\ v=i\vdash v'\in O$。

定义 4 缺陷设计

设 D 是期望的设计,而 D'是非期望的设计,如果 $D\not\subset D'$则称 D'为缺陷设计。

定义 5 缺陷充分的测试用例(Fault-adequate Test Case)

t 是测试用例,D 为设计,D'是 D 的缺陷设计,如果 $t\subseteq D\wedge\neg(t\subset D')$,则测试用例 t 可以区分 D 和 D',即测试用例 t 为缺陷充分的测试用例。从变异分析的角度来说,即测试用例 t 可以杀死缺陷设计 D'。

3.2 缺陷测试和变异分析的基本概念

缺陷测试的主要目的是通过产生特定测试用例来发现软件中特定的缺陷[5]。虽然要发现软件中所有的缺陷是不可能的,缺陷测试可以发现一些软件开发人员经常会犯的常规错误。

变异分析[6]是一种经典的缺陷测试技术,可以划分为功能性测试。变异分析的方法是对设计(规格说明或程序)执行各种变异算子操作来引入特定的缺陷,产生各种变异体,特定的变异体被用于特定的测试覆盖。

最初的变异分析是在程序变异的基础上开展的,随着形式化方法及软件测试用例自动生成技术的发展,变异分析应用的重点逐渐转移到形式化规格。变异分析可以简单描述为(以程序的变异分析为例):

对原始的程序 p,执行各种变异算子操作,得到 p 的变异体的集合 P',在测试用例集 T 上,$p'\in P'$,$\exists t\in T\cdot(\neg\ Pass\ (p',t))$,则 p'被"杀死",如果 $\forall t\in T\cdot(\neg\ Pass(p',t))$,则称 p'"存活"。在以下两种情况下,会使 p'"存活":

(1)测试用例数据不够,如果有大量的 p'"存活",那么很明显,测试用例太少,可以通过增加测试用例来解决。

(2)p'和 p 是等价的,发生这种情况是小概率事件,在这种情况下变异分析失效。

3.3 测试用例的生成算法

从 UML/OCL 模型生成测试用例集可以建模为一个人工智能中的约束满足问题(CSP),通过求解该 CSP,寻找该 CSP 的最大解,来生成测试用例集。

根据缺陷充分的测试用例的定义,可以使用如下的算法[4]来生成测试用例:给定一个设计 $D(Pre\vdash Post)$和它的缺陷设计 $D'(Pre'\vdash Post')$作为输入,则一个输入-输出的测试用例可以按照如下的步骤生成:

第一步:

(1)找到满足 $Pre\wedge Post'\wedge\neg\ Post$ 的一个解(i_c,o_c)。

(2)如果找到解,则测试用例 $T=t(i,O)$可以通过搜寻 $Pre\wedge Post\wedge(v=i_c)$的最大解$(i,O)$求得。

第二步:如果第一步没有成功,则搜寻满足$\neg\ Pre'\wedge Pre\wedge\neg\ Post$ 的最大解(i,O),测试用例为 $T=t(i,O)$。

算法就是将设计精化为测试用例,(i_c,o_c) 表示 $D\subset D'$的反例。下面是算法的证明:$t(i,O)\subseteq D$

测试用例的正确性:

当(i,O)是 $Pre\wedge Post\wedge(v=i_c)$的解,

$$t(i,O)\subseteq D=[(v=i)\Rightarrow Pre]\wedge[(v=i)\wedge Post\Rightarrow v'\in O]=$$
$$\text{true}\wedge\text{true}=\text{true}$$

另一种情形类似可证明。

测试用例的缺陷覆盖性:$t(i,O)\not\subset D$

当(i,O)是 $Pre\wedge Post'\wedge\neg\ Post$ 的解时

$$t(i,O)\not\subset D=\neg\ [(v=i)\Rightarrow Pre']\vee\neg\ [((v=i)\wedge Post')\Rightarrow v'\in O]=\exists v\cdot((v=i))\wedge\neg\ Pre')\vee\exists v,v'\cdot((v=i))\wedge Post'\wedge(v'\notin O))=\exists v\cdot((v=i)\wedge Pre')\vee\text{true}=\text{true}$$

当(i,O)是$\neg\ Pre'\wedge Pre\wedge Post$ 的解时

$$t(i,O)\not\subset D=\neg\ [(v=i)\Rightarrow Pre']\vee\neg\ [((v=i)\wedge Post')\Rightarrow(v'\in O)]=\exists v\cdot((v=i))\wedge\neg\ Pre')\vee\exists v,v'\cdot((v=i))\wedge\exists v,v'\cdot((v=i))\wedge Post'\wedge(v'\notin O))=\text{true}\vee\exists v,v'\cdot((v=i))\wedge\exists v\cdot v'((v=i))\wedge Post'\wedge(v'\notin O))=\text{true}$$

4 将 OCL 表达式转化为约束目标集

测试单元可以是一个函数或类的一个方法,生成该测试单元的测试用例所需要的 OCL 表达式为系统的不变式、前置表达式和后置表达式,由于执行操作前和执行操作后都要满足系统的不变式,因此一个测试单元的约束集为 $Inv@pre\wedge Pre\wedge Post\wedge Inv$。本文采用 DNF 分析将 OCL 表达式转化为约束集。DNF 分析[7]是针对 OCL 表达式中 and、or、xor、implies 属性操作和 If 表达式进行处理。

定义分离函数 P:$OCL\ Expression\rightarrow Partitions$。操作符×定义为:$Partitions\times Partitions\rightarrow Partitions$,$P_1\times P_2=\{c_1\wedge c_2|c_1\in P_1,c_2\in P_2\}$,$P_1\times P_2\times\cdots\times P_n=\cup\{P_1,\cdots,P_n\}$。DNF 的分析规则为

$$P(A \vee B)=\cup\begin{Bmatrix}P(A \wedge \neg B)\\P(A \wedge B)\\P(\neg A \wedge B)\end{Bmatrix} \quad (1)$$

$$P(A \Rightarrow B)=P(\neg A \vee B) \quad (2)$$

$$\text{if } A \text{ then } B \text{ else } C=\cup\begin{Bmatrix}P(A \wedge B)\\P(\neg A \wedge C)\end{Bmatrix} \quad (3)$$

$$P(A \oplus B)=\cup\begin{Bmatrix}P(A \wedge \neg B)\\P(\neg A \wedge B)\end{Bmatrix} \quad (4)$$

$$P(A \wedge B)=P(A) \times P(B) \quad (5)$$

将 OCL 表达式转化为约束集分成两个阶段，首先是遍历 OCL 抽象语法树,将 OCL 表达式转化为一棵正则二叉树,然后再将这棵正则二叉树转化为约束集。

(1)将一个 OCL 抽象语法树转化为一棵正则二叉树,该树的叶子为约束式,非叶子节点为合取或析取算子(在这里析取算子为"∪",而不是逻辑或)。根据 OCL 的文法规则,使用访问者模式来遍历抽象语法树，在遍历的同时应用前 4 条规则,生成一棵正则二叉树。该树结点如图 2 所示。

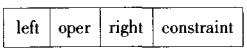

图 2 正则二叉树的结点示意图

如果为叶子节点,则只有 constraint 字段包含数据,其余各项均为 null,否则 constraint 为 null、oper 为合取或析取算子。

(2)获取表达式的正则二叉树后,再应用规则(5)将此转化为约束目标集,算法实现如下:

使用如下的数据结构来表示该约束目标库(Constraint Store)。约束目标库用向量来表示，向量的每个元素为约束目标,约束目标使用一个链表来表示,链表中的每个元素表示一个约束,简图如图 3 所示:

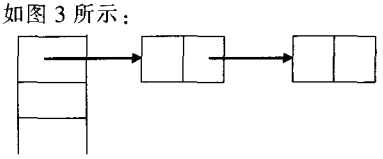

图 3 约束目标库的数据结构示意图

该算法用伪代码表示如下:

```
ConstraintStore P(Expression exp,ConstraintStore C){
if(exp is a constraint){
    add the constraint to each of constraint goal in C
    return C
}
if(exp is a conjuction){
    C1=P(exp.left,C)
    C2=P(exp.right,C1)
    return C2
}
if(exp is a disjuction){
    C1=P(exp.left,C)
    C2=P(exp.right,C)
    return(C1 unify C2)
  }
}
```

5 系统的整体架构

测试用例的生成过程可以分为三步：首先是将 UML/OCL 模型进行文法分析,获取抽象语法树,由于直接分析图形化的 UML 模型比较繁琐,使用 XMI[10]/XSLT 技术,先将其转化为符合 BNF 规范的模型;然后应用变异分析、DNF 分析技术生成约束集,将测试用例的自动生成算法转化为 CSP 模型;最后应用约束推理规则,求解 CSP,生成测试数据,将生成的测试数据和 xUnit 测试框架结合,生成单元测试用例集。整个系统的框架如图 4 所示:

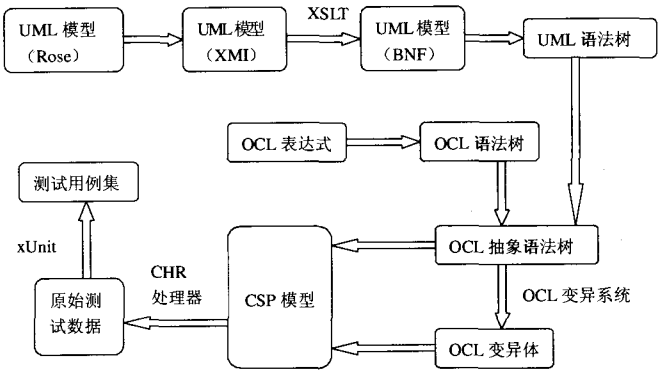

图 4 系统实现简图

6 结束语

当前很多关于测试用例自动生成技术的研究是基于传统的形式化语言或 UML 模型。传统形式化语言的复杂性导致其在工业界并没有被广泛采用，而 UML 模型规格说明的歧义性也降低了测试用例生成算法的精确性。本文介绍基于 UML/OCL 模型来自动生成测试用例,因此既有数学严谨性,又容易被实际的软件开发过程所采用。本文另一个技术上的特色是将缺陷测试分析技术同 CSP 结合求解。课题的意义在于将模型驱动架构中基于模型的软件代码自动化生成思想应用于基于模型的软件测试用例自动生成。课题开发的框架,可以使单元测试的自动化程度大大提高，从整体上提高软件开发的效率。

(收稿日期:2006 年 9 月)

参考文献:

[1] OMG UML 2.0 OCL Specification[EB/OL].[2005-03-28].http://www.omg.org/docs/ptc/03-10-14.pdf.

[2] Morell L J.A theory of fault-based testing[J].IEEE Transactions on Software Engineering,1990,16(8):844-857.

[3] Russell S,Norvig P.Artificial intelligence:a modern approach[M]. 2nd.[S.l.]:Prentice Hall,2003.

[4] Aichernig B K,Pari Salas P A.Test case generation by OCL mutation and constraint solving[C]//QSIC 2005,Fifth International Conference on Quality Software EDEN,Australia,September 19-20,2005.

[5] Kuhn D R.Fault classes and error detection capability of specification based testing[J].ACM Transactions on Software Engineering and Methodology,1999,8(4).

[6] Black P E,Okun V,Yesha Y.Mutation operators for specifications [C]//ASE'00,Proceedings of the The Fifteenth IEEE International Conference on Automated Software Engineering(ASE'00),Washington,DC,USA,2000.IEEE Computer Society,2000:81-90.

[7] Meudec C.Automatic test cases generation from formal specifications[D].Faculty of Science,The Queen's University of Belfast,1998.

[8] Frankel D S.应用 MDA[M].北京:人民邮电出版社,2003.

[9] Grose T J.精通 XMI[M].北京:电子工业出版社,2004.