高技术通讯 2010 年 第 20 卷 第 7 期:727 ~ 732

doi:10.3772/j.issn.1002-0470.2010.07.012

基于用户需求的构件行为测试①

李良明②* 王志坚** 刘 磊*

(* 南京航空航天大学信息科学与技术学院 南京 210016)

(** 河海大学计算机及信息工程学院 南京 210098)

摘 要 针对基于构件的软件开发过程中构件的使用者难以验证构件的动态行为这一问题,提出了一种基于用户需求的构件行为测试方法。用接口自动机为构件的行为建模,研究如何根据模型和用户需求对构件进行测试的问题。首先通过对行为模型的分析,确定出构件中不同接口之间的关系,进而用一组相关的接口序列来表示构件的动态行为;然后再根据用户在使用时的具体要求,确定出实际要运行的测试序列。文中介绍的方法能够从整体上检验构件的行为并且可以根据构件模型和使用要求自动生成测试序列,便于用户对构件的验证和测试。

关键词 构件测试,构件行为,测试序列,基于模型的测试

0 引 言

在基于构件的软件开发(component based software development, CBSD)过程中,由于构件本身固有的性质,如异质性、源代码的不可得性,使得构件测试面临许多新的问题[1]。软件开发过程中使用的构件通常是黑盒构件,一般只提供与接口相关的信息,使用者缺少对构件的详细了解,由此带来了对构件的"信任"问题[2]。为解决此问题,一方面可要求开发者附加一定的信息,用于描述构件行为,便于对构件的使用和测试;另一方面,使用者必须在应用环境中对构件进行验证和测试,以确定其行为是否满足要求。例如,为了分析构件在实际使用过程中遇到的问题,文献[3]中介绍了一种通过构件状态空间的可视化和使用协议注释的方式来提高模型检验效率的方法。在文献[4]中,作者给出了一系列算法,对确定构件中是否具有用户想要的行为进行了研究。而为了消除构件中用户不需要功能的影响,文献[5]介绍了一种通过构造极大包含环境从构件中提取满足场景规约的所有行为的方法。上述这些方法都侧重于对构件行为的描述和分析,但没有说明如何系统地对构件的行为进行测试。

构件的行为协议常用有限状态机(finite state machine, FSM)和统一建模语言(unified modeling language, UML)状态图来表示,对 FSM 测试的研究常常基于识别序列(distinguishing sequence)和唯一输入/输出序列(unique input/output sequences),研究的内容主要是如何生成这些序列,再将其应用于状态识别、状态确认和错误检查等一致性检验方面[6-8];基于 UML 模型的测试技术则一般是以 UML 的状态图、协作图和顺序图为基础[9-11],再应用传统的数据流和控制流分析技术或者与 FSM 的测试相结合,生成测试用例。基于模型的测试主要研究模型是否正确地实现了预期的功能,其重点是对模型中每一步活动进行检验,未考虑如何对完成特定功能的多个活动的组合进行测试。

本文以构件的接口自动机模型为基础[12],通过对不同接口关系的分析,提出了一种构件功能行为序列的生成方法,再结合用户在使用时的具体要求,能够生成所需要运行的全部测试序列。通过对构件行为的测试,可以为构件的组装验证和系统测试等工作奠定基础。

1 基本概念及符号表示

开发者提供的构件一般是黑盒构件,由于使用者一般不了解构件的实现细节,很难对构件进行彻

① 863 计划(2007AA01Z178)和 973 计划(2002CB312002)资助项目。

② 男,1977 年生,博士生,讲师;研究方向:软件测试与构件技术;联系人,E-mail: lilm@nuaa.edu.cn

(收稿日期:2009-03-10)

底的测试,因此对构件所提供的功能和表现出的整体行为进行测试是一种较直观和实用的方法。本节中借助一些相关的概念,给出构件功能行为的一种表示方法。其中接口自动机的定义主要来自文献[12]。

定义 1(接口自动机) 接口自动机 P 是一个六元组 $P = < V_P, V_P^{\text{init}}, A_P^I, A_P^O, A_P^H, T_P >$,其中:

- V_P 是状态的有穷集合。
- $V_P^{\text{init}} \subseteq V_P$ 是初始状态集,且 V_P^{init} 中至少包含一个元素。
- A_P^I, A_P^o, A_P^H 分别为输入活动、输出活动和内部活动集合,且两两互不相交。记所有活动的集合为 A_P,$A_P = A_P^I \cup A_P^o \cup A_P^H$。
- $T_P \subseteq V_P \times A_P \times V_P$ 是迁移的集合,将 $(v, a, v') \in T_P$ 记为 $v \xrightarrow{a} v'$。

若 $| V_P^{\text{init}} | = 1$,且 $\forall v \xrightarrow{a} u, v \xrightarrow{a} u'$,则有 $u = u'$,称 P 是确定性的;若 $\forall v, u \in V_P, \exists\ v \xrightarrow{a1} v1 \cdots vi \xrightarrow{ai} u$,其中 $a1, \cdots, ai \in A_P, v1, \cdots, vi \in V_P$,则称 P 为强连通的。本文假定研究所涉及的接口自动机都是确定性和强连通的。

定义 2(执行片段) 接口自动机 P 的一个执行片段是其状态与活动交替排列的有穷序列 $v_i a_i v_{i+1} a_{i+1} \cdots a_{n-1} v_n$。

定义 3(运行) 接口自动机 P 中的任意执行片段 $\eta = v_i a_i v_{i+1} a_{i+1} \cdots a_{j-1} v_j (i < j)$,其中 $v_i \in V_P^{\text{init}}$,$v_k \notin V_P^{\text{init}}, k = i+1, \cdots, j-1$,如果 $v_i = v_j$,则称 η 为 P 中的一次运行。

定义 4(简单环) 对于 P 中的任意执行片段 $\eta = v_i a_i v_{i+1} a_{i+1} \cdots a_{j-1} v_j (i < j)$,如果 η 满足:(1) $v_i = v_j$ 且 $v_s \notin V_P^{\text{init}} (i \leqslant s \leqslant j)$;(2) $v_s \neq v_t (s \neq t, i \leqslant s, t < j)$,则称 η 为简单环,用 $SLoop$ 表示。即简单环中不包括初始状态且除了头状态与尾状态相同外,不能再有相同的状态出现。

定义 5(简单运行) 接口自动机 P 中的运行 $\eta = v_i a_i v_{i+1} a_{i+1} \cdots a_{j-1} v_j$,对于 η 中的任意两个执行片段 η_1, η_2,若 η_1, η_2 均为 $SLoop$,有 $\eta_1 \neq \eta_2$,则称 η 为简单运行,用 $SimOpr(\eta)$ 表示。由定义可知,在一次简单运行中不能出现相同的简单环。

定义 6(功能行为) 对于构件 P 的一个简单运行序列 $\eta = v_0 \cdots (\eta_i) \cdots (\eta_j) \cdots v_n$,其中 $v_0 = v_n \in V_P^{\text{init}}$,$\eta_i, \cdots, \eta_j$ 均为 $SLoop$,则由 $\eta = v_0 \cdots (\eta_i) * \cdots (\eta_j) * \cdots v_n$ 构成的所有运行序列称为 P 在 η 上的功能行为,用 $FBhv_P(\eta)$ 表示,$*$ 表示循环执行一次至多次。

若在一个简单运行序列中存在某几个小循环组成一个大循环的情况,则应将大小循环分开考虑,区别对待。由所有简单运行序列上的功能行为构成的集合,称为构件 P 的功能行为,用 $FBhv_P$ 表示,构件的一个功能行为序列表示了构件的一个特定功能。

定义 7(用户需求说明) 用户需求 $Req_{\text{user}} = (Func, Res)$,其中 $Func = (func(1), \cdots, func(m))$,$Res = (res(1), \cdots, res(n))$。用户需求说明分为两部分,一是要求构件所完成的功能,二是在实际使用时的一些限制条件。

图 1 是构件 Comp 的接口自动机模型,符号"?""!"和";"分别表示该活动为输入、输出和内部活动。例如 $(0, msg?, 1, send!, 2)$ 是一个执行片段,而 $(0, msg?, 1, send!, 2, ack?, 5, ok!, 0)$ 是一个简单运行,同时也是构件 Comp 的一个功能行为。

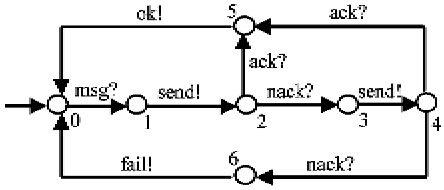

图 1 构件 Comp 的接口自动机模型

2 构件行为的测试过程

本文提出的基于用户需求的构件行为测试过程主要分为两部分,一是根据模型生成构件的功能行为序列,二是确定用户需求场景与构件行为序列之间的对应关系,再根据使用的具体条件来生成所需的测试序列。

2.1 简单运行序列分析

本节对构件的简单运行序列进行分析,以确定其在进行构件测试时的有效性。我们假定构件的模型要满足两个条件:所有的状态都是可达的;所有的迁移都可以激活。通过对简单运行序列的分析,得到如下结论。

定理 1 $SimOpr_P$ 可以实现对构件行为模型的状态覆盖。

证明:因为接口自动机是强连通的,对于 P 来说,$\forall v, v \in V_P$,至少存在一条从 V_P^{init} 到达 v 的简单路径 η_i,也存在至少一条从 v 到 V_P^{init} 的简单路径 η_j。而 η_i 与 η_j 的组合构成了 P 的一次运行 η,且 η 中

同一回路最多出现一次。因为如果 η 中同一回路出现两次,则说明 η_i 与 η_j 至少有一个存在回路,而这与 η_i 和 η_j 都是简单路径相矛盾,所以此种情况不存在。因此存在 $\eta \in SimOpr_P$,使得状态 v 被覆盖,所以 $SimOpr_P$ 可以覆盖构件动态行为模型中的所有状态。

定理 2 $SimOpr_P$ 可以实现对构件行为模型的迁移覆盖。

证明:与定理 1 的证明类似,将迁移抽象看作是构件行为模型中的一个整体,可得到与定理 1 同样的结论。

定理 3 $SimOpr_P$ 可以实现对构件行为模型中不同条件组合的覆盖。

证明:接口自动机中的条件通过迁移来实现,不同条件的组合即相关联迁移的组合。我们假定迁移 t_1 和 t_2 是要组合的不同条件,多个条件组合的情况与此类似,可通过两两组合完成。

因为 t_1 和 t_2 要组合产生某种结果,所以 t_1 到 t_2 之间至少存在一条可达的简单路径 η_i,否则 t_1 与 t_2 不相关。然后可将 (t_1, η_i, t_2) 看作是一个大的迁移,进而可将其当作一个整体对待。由定理 1 和定理 2 的结果可知,存在 $\eta \in SimOpr_P$ 使得 (t_1, η_i, t_2) 被覆盖,所以 $SimOpr_P$ 可以实现对动态行为模型中不同条件组合的覆盖。

因为本文研究的接口自动机都是强连通的,所以任意的活动之间都存在一条可达路径;接口自动机又是确定性的,故不考虑活动之间并行的情况。通过上述定理和 $FBhv_P$ 的定义可知,若构件 P 的功能可在一次运行中完成,则构件的所有功能都可由 $FBhv_P$ 来描述。简单运行序列是生成构件功能行为的基础,通过上述分析过程定可以看出其在进行构件功能测试时的有效性。

2.2 构件功能行为序列的生成

生成功能行为序列的主要思想是从构件的初始状态开始,参照图的深度优先遍历算法,查找出所有的简单运行序列;再将简单运行序列中不同的循环部分进行标识,从而得到构件所有的功能行为,具体的描述如算法 1 所示。

算法中用的符号主要有 3 个:

• "*tail*()":取尾运算,取目标元素的最后一个状态;

• "+":连接操作,将运算符前后两部分相连接,要求运算符之前部分的尾状态是其后一部分的头状态;

• "-":分解运算,从运算符前的内容中去掉运算符之后的相关部分,去掉某一状态时,也将到达该状态的活动去掉。

算法 1 生成构件的功能行为序列

//P 只有一个初始状态,num 为所生成的简单运行序列个数

Begin:

Step1://生成所有简单运行序列

$v_0 = V_P^{init}$; $\eta = v_0$; $num = 0$;

$v_i = v_0 \rightarrow next$; //即 $v_0 \xrightarrow{a} v_i$

$\eta = \eta + v_i$

while (($v_i \neq V_P^{init}$) and (v_i is not visited))

{//查找第一个简单运行 $\eta = v_0 a_i v_i \ldots v_j a_{j+1} v_0$

$v_i = v_i \rightarrow next$; $\eta = \eta + v_i$;

}

if (found η)

{ $num = 1$; $SimOpr_P[num] = \eta$; }

else

return ("can't find any SimOpr");

$\eta = \eta - tail(\eta)$;

while (($tail(\eta) \neq V_P^{init}$) or ($\exists t, Source\_State(t) == V_P^{init}$ and t is not visited))

{//按深度优先遍历的思想,查找其他简单运行

$v = tail(\eta)$;

if (($\exists \eta_i, \eta_i = va_i v_{i+1} \ldots v_j a_{j+1} V_P^{init}$) and ($\forall \eta_1, \eta_2 \in \eta_i, \eta_1, \eta_2 \in SLoop$ 有 $\eta_1 \neq \eta_2$))

{$\eta = \eta + \eta_i$;

if ($\eta \notin SimOpr_P$)

{ $num = num + 1$; $SimOpr_P[num] = \eta$; }

}

else $\eta = \eta - tail(\eta)$;

}

Step2://生成 $FBhv_P$

for (each $SimOpr_P[i]$ in $SimOpr_P$)

{

标记每一个 $SimOpr_P[i]$ 中的 $SLoop$;

将不同 $SLoop$ 组合所构成的运行序列确定出来;

}

End.

功能行为是一个构件在其运行期间所表现出来的整体行为,该算法所得到的结果,对构件来说它描述了每一个功能行为所涉及到的接口操作之间的关

系;对构件使用者来说,它给出了构件所能完成的功能集合,可以用来验证构件的可用性。

在接口自动机的图形表示中,活动与状态的数目是有限的,且简单运行序列中要求简单环不能重复出现,所以上述算法一定会结束。设接口自动机中变迁的数目为 n,算法 1 中 Step1 内第一个 while 循环的时间复杂度为 $O(n)$;第二个 while 循环是算法中的主要循环部分,其中确定从 V_P^{init} 发出的所有活动是否都已查找过的复杂度为 $O(n)$,而在最坏情况下,一个简单运行序列的长度为 n,所以确定是否存在与其构成新的简单运行序列的时间复杂度为 $O(n^2)$,故第二个 while 循环的时间复杂度为 $O(n^3)$;Step2 中的时间复杂度为 $O(n^2)$。综上所述,算法 1 的时间复杂度为 $O(n^3)$。

2.3 基于用户需求的构件行为测试过程

用户在需求说明中给出了期望构件实现的功能,构件则通过不同的功能行为表现出可以完成的操作。可以通过比较二者的相互关系,决定构件是否可以使用。若用户要求的功能大于构件所能实现的功能,则单个构件不能满足要求,该构件需要与其他构件组合来实现用户所需的功能;若构件除了用户要求的功能外还有多余的功能,则可参考文献[5]中介绍的方法,将用户不需要的功能去掉。

当生成构件的功能行为序列后,首先要判断用户的需求包含在哪些序列中,将可能的对应关系全部找出来;然后将需求中的限制条件应用于相应的功能行为上,确定构件使用时的具体条件;最后分析得到的功能行为序列,根据其涉及到的接口操作生成测试用例。我们对此过程进行了分析,给出相应的算法描述如下。

算法 2 基于用户需求场景的构件测试过程

//构件 P 的功能行为序列 $FBhv_P$,用户需求场景 $Req_{user} = (Func, Res), Func = (func(1), \cdots, func(m)), Res = (res(1), \cdots, res(n))$。

Begin:

Step1: For each $func(i)$ in $Func$

if 可以找到 $FBhv_P(k) \in FBhv_P$

$func(i) \subseteq FBhv_P(k)$ //用户需求包含在构件功能行为中

then $func(i) \rightarrow FBhv_P(k)$; //确定二者的对应关系

else 返回相关信息,结束。

Step2: For each $res(i)$ in Res

确定 $res(i) \rightarrow func(j) \rightarrow FBhv_P(k)$ 的关系;//找到每一个限制条件有关的功能行为

得到 $FBhv_P \leftarrow Res$ 集合;// $FBhv_P \leftarrow Res$ 表示将限制条件应用于所有相关的功能行为

Step3: For each $FBhv_P(i) \leftarrow res(j)$ in $FBhv_P \leftarrow Res$

根据其涉及的接口操作,选择输入数据生成测试用例,得到 P 的测试用例集 T_P;

Step4: For each t_P in T_P

运行 t_P,通过测试的结果,确定该构件是否可以满足用户的需求。

End.

需要注意的是,Step1 中的结果可能存在用户需求中的某个场景可以由多个构件行为序列实现的情况,此时需要将涉及该运行场景的行为序列全部找出来,以确保用户需求在所有可能的构件行为中都得到满足。

3 实例分析

本节通过一个简化的 ATM (automatic teller machine)构件的例子对所提出的方法进行了分析说明,其接口自动机模型如图 2 所示。

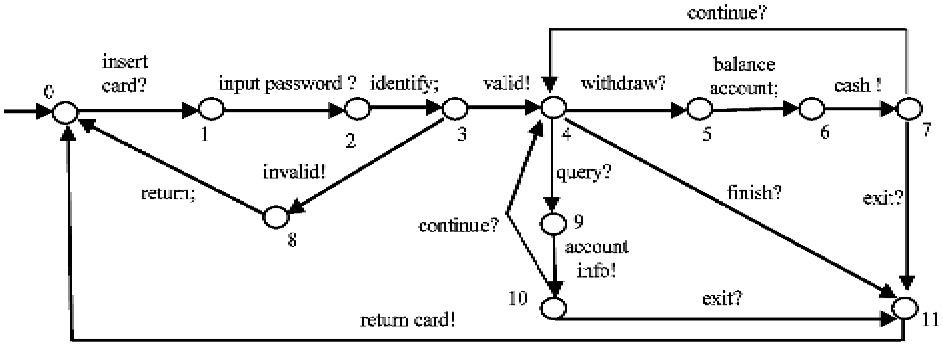

图 2 构件 ATM 的接口自动机模型

假设该构件实现的功能有3个:用户身份验证—Identify;取款—Withdraw;查询—Query。用户需求说明 $Req_{user} = (Func, Res)$。

- $Func = (Identify*, Withdraw*, Query*, (Withdraw \cdot Query)*)$, $Withdraw \cdot Query$ 表示先取款后查询;
- $Res = (|Identify*| \leqslant 3, |Withdraw*| \leqslant 2, |(Withdraw \cdot Query)*| \leqslant 2)$, 用 $|X*|$ 表示其中 X 执行的最大次数。

通过分析可知,用户要求该构件最多可以验证身份3次,其中成功与失败各3次;可以取款最多2次;取款与查询的组合最多2次,包括先取款1至2次,再查询1至2次。

首先我们按照算法1生成该构件的所有功能行为,为描述方便我们将构件的行为序列用其所经过的状态序列表示。结果如下所示:

$Bhv_P(1) = (0,1,2,3,8,0)$,

$Bhv_P(2) = (0,1,2,3,4,11,0)$,

$Bhv_P(3) = (0,1,2,3,4,5,6,7,11,0)$,

$Bhv_P(4) = (0,1,2,3,(4,5,6,7,4)*,11,0)$,

$Bhv_P(5) = (0,1,2,3,(4,5,6,7,4)*,9,10,11,0)$,

$Bhv_P(6) = (0,1,2,3,(4,5,6,7,4)*,(4,9,10,4)*,11,0)$,(其中第二个循环的头状态4与前一个循环的尾状态是同一个,为了便于表示,用这种分开的形式给出。)

$Bhv_P(7) = (0,1,2,3,4,9,10,11,0)$,

$Bhv_P(8) = (0,1,2,3,(4,9,10,4)*,11,0)$,

$Bhv_P(9) = (0,1,2,3,(4,9,10,4)*,5,6,7,11,0)$,

$Bhv_P(10) = (0,1,2,3,(4,9,10,4)*,(4,5,6,7,4)*,11,0)$,

$Bhv_P(11) = (0,1,2,3,(4,5,6,7,4,9,10,4)*,11,0)$,

$Bhv_P(12) = (0,1,2,3,(4,9,10,4,5,6,7,4)*,11,0)$。

再根据算法2,结合用户的需求,得到实际使用时需要运行的测试序列,如下所示:

$Identify = Bhv_P(1) | Bhv_P(2), |Identify*| \leqslant 3$,因为 $Bhv_P(1)$ 和 $Bhv_P(2)$ 都表示验证用户身份的操作,所以这两个功能行为都需要测试,用"|"表示二者需要分别测试。故其测试序列为:$(0,1,2,3,8,0)^3$,$(0,1,2,3,4,11,0)^3$,我们用上标表示该序列执行的最大次数;

$Withdraw = Bhv_P(3) | Bhv_P(4), |Withdraw*| \leqslant 2$,故其测试序列为:$(0,1,2,3,4,5,6,7,11,0)^2$,$(0,1,2,3,(4,5,6,7,4)^2,11,0)$;

$Query = Bhv_P(7) | Bhv_P(8)$,因次数未作限制,故其测试序列为:$(0,1,2,3,4,9,10,11,0)*$,$(0,1,2,3,(4,9,10,4)*,11,0)$,此种情况,可根据测试时的其他限制条件确定执行次数。

$Withdraw \cdot Query = Bhv_P(5) | Bhv_P(6) | Bhv_P(11), |(Withdraw \cdot Query)*| \leqslant 2$,其测试序列为:$(0,1,2,3,(4,5,6,7,4)^2,9,10,11,0)$,$(0,1,2,3,(4,5,6,7,4)^2,(4,9,10,4)^2,11,0)$,$(0,1,2,3,(4,5,6,7,4,9,10,4)^2,11,0)$。

上述过程可以只测试在用户需求中定义的构件行为,从而能够节省测试的时间和费用。若在实际使用的限制条件下构件表现出的行为满足用户要求,即认为构件可以使用。

4 结 论

由于构件使用者通常无法得到构件的源代码及详细的设计知识,在验证构件的功能与其需求是否真正一致时往往比较困难。本文提供了一种比较直观的通过将构件所提供的功能与用户具体的需求相比较来检验构件行为的方法。该方法在生成构件的功能行为序列时,充分考虑到不同条件组合时的情况,具有较强的实用性。另外,该方法具有一定的通用性,只要将文中介绍的算法根据具体模型进行适当的修改,不仅可以应用于接口自动机,也可应用于其他形式的模型,如FSM,Petri网等。

本文的研究过程中对构件的模型做了一定的限制,要求构件中的活动不包括并行操作。而在实际使用中可能存在不同活动之间的并行运行,因此下一步考虑研究此类情况下构件行为序列的生成过程。此外,还要研究如何将行为序列应用于构件组装测试的过程。

参考文献

[1] 毛澄映,卢炎生. 构件软件测试技术研究进展. 计算机研究与发展, 2006, 43(8): 1375-1382

[2] Beydeda S, Gruhn V. State of the art in testing components. In: Proceedings of the 3rd International Conference on Quality Software, Dallas, Texas, USA, 2003. 146-153

[3] Jezek P, Kofron J, Plasil F. Model checking of component

behavior specification: a Real life experience. In: Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005). *Electronic Notes in Theoretical Computer Science*, 2006,160:197-210

[4] 胡军, 于笑丰, 张岩等.基于场景规约的构件式系统设计分析与验证.计算机学报, 2006, 29(4):513-525

[5] 张岩, 胡军, 于笑丰等.场景驱动的构件行为抽取.软件学报,2007, 18(1):50-61

[6] Lee D, Yannakakis M. Testing finite-state machines: state identification and verification. *IEEE Transactions on Computer*, 1994, 43(3): 306-320

[7] Yalcin M C, Yenigun H. Using distinguishing and UIO sequences together in a checking sequence. In: Proceedings of the International Federation for Information Proceeding (IFIP) 2006, LNCS 3964, 2006. 259-273

[8] Yannakakis M, Lee D. Testing finite state machines. In: Proceedings of the 23rd annual ACM Symposium on Theory of Computing, New Orleans, Louisiana, United States, 1991. 476-485

[9] Samuel P, Mall R, Bothra A K. Automatic test case generation using unified modeling language (UML) state diagrams. *IET software*, 2008, 2(2):79-93

[10] Abdurazik A, Offutt J. Using UML collaboration diagrams for static checking and test generation. In: Proceedings of the 3rd International Conference on the Unified Modeling Language (UML'00), York, UK, 2000. 383-395

[11] Lund M S, Stølen K. Deriving tests from UML 2.0 sequence diagrams with neg and assert. In: Proceedings of the 2006 International Workshop on Automation of Software Test, Shanghai, China, 2006. 22-28

[12] de Alfaro L, Henzinger T A. Interface automata. In: Proceedings of the 9th Annual ACM Symposium On Foundations of Software Engineering (FSE 2001), Vienna, Austria, 2001. 109-120

Component behavior testing based on user requirements

Li Liangming*, Wang Zhijian**, Liu Lei*

(* College of Information Science & Technology, Nanjing University of Aeronautics & Astronautics, Nanjing 210016)

(** College of Computer & Information Engineering, Hohai University, Nanjing 210098)

Abstract

In view of the problem that in component based software development it is difficult to validate the component's dynamic behavior, the paper presents a novel approach for component behavior test. The component behavior is modeled using the interface automata to study the problem of how to test component behavior according to the model and user requirements. The relationships among component interfaces are first detected after analyzing the component model, and then the component behavior is expressed with a set of sequences of relevant interfaces. Subsequently, the testing sequences are derived according to the user requirements. The experimental results show that the approach proposed can verify the component behavior at the system level and generate testing sequences automatically, thus facilitating the component validation and test.

Key words: component testing, component behavior, testing sequence, model-based testing