第 37 卷 第 15 期
Vol.37 No.15

计 算 机 工 程
Computer Engineering

2011 年 8 月
August 2011

• 软件技术与数据库 •　　文章编号：1000—3428(2011)15—0030—04　　文献标识码：A　　中图分类号：TP311

# 基于决策树的回归测试子集选取

朱 彬

(上海第二工业大学计算机与信息学院，上海 201209)

**摘 要：**软件版本的频繁变更及测试资源的限制要求软件回归测试采用新的测试用例集合的生成和约简技术。为此，介绍基于决策树的回归测试子集的选取方法，将测试用例和测试需求作为一种知识表示系统，对测试知识表示系统进行约简，将约简后的系统构造成一棵决策树，由决策树获得被约简的回归测试子集。理论分析证明该方法复杂度较低。

**关键词：**决策树；回归测试；测试子集选取；测试需求；测试知识

# Selection of Regression Test Subset Based on Decision Tree

ZHU Bin

(School of Computer & Information, Shanghai Second Polytechnic University, Shanghai 201209, China)

**【Abstract】**Increasing number of software versions and the limitation of test resources challenge the traditional methods of regression test. This paper presents a new approach to select test subset for regression test by building the decision tree of test knowledge system. Test cases and test requirements are denoted as a test knowledge system in this method. The test knowledge system is reduced by eliminating redundant attributes. It builds the decision tree from the reduced test knowledge system. A test subset can be obtained from the decision tree. To implement the method by computer, it also gives the algorithm for reducing test knowledge system and the algorithm for building the decision tree.

**【Key words】**decision tree; regression test; test subset selection; test requirement; test knowledge

**DOI:** 10.3969/j.issn.1000-3428.2011.15.008

## 1 概述

快速原型法和敏捷开发方法已在软件开发中被广泛应用，造成软件版本的频繁变更及软件测试的困难。由于无法保证新引入的模块或变更的模块不给原有系统带来错误，因此需要对整个软件系统进行再次测试。现有的回归测试策略包括全部重测和选择性测试[1]，但这 2 种测试策略均有不足，前者会消耗大量的测试资源，后者则会造成测试不完全[2]，最终导致软件质量的低下和“软件可信任”的问题。

为了保证软件可靠性，同时有效地降低软件测试成本，回归测试中测试用例集合的生成及其约简技术已成为软件工程领域的研究热点之一[3-7]。然而测试用例的最小化过程是一个 NP 难问题[3]，因此，传统的测试用例集约简技术采用了启发式方法，如启发式 H 方法[3]、启发式 GRE 方法[4]、混合式方法[5]和贪婪选择 G 方法[6]。不同于传统启发式的方法，本文提出了构造决策树的回归测试子集选取方法。该方法借鉴传统知识表示的方式，将测试用例和测试需求看成一个测试知识表示系统，再应用知识表示系统中的绝对属性约简法[8]对测试知识表示系统进行约简，将约简后的系统构造成一棵决策树，选取决策树中非空的叶子结点构造回归测试子集。

## 2 测试用例

本节以程序 A 的修改为例，描述回归测试中测试用例集合的冗余问题。

**程序 A**

```
a:    read(x, y, z);
b:    while (x > 0 && y>0 && z>0) //原程序
b′:   while (x⩾0 && y⩾0 || z⩾1) //修改后程序
c:        if(z⩾3)
d:        x=x−z−y;
          else
e:        y=y+z−x;
f:        endif
g:        z=z−1;
h:    end while
```

在程序 A 中，对 b 行进行了修改，得到 b′行。若原程序的测试用例集为 $U=\{t_1=(x=0, y=0, z=0), t_2=(x=1, y=1, z=1), t_3=(x=1, y=1, z=3)\}$。针对 b′行的修改，补充设计了测试用例 $t_4=(x=0, y=0, z=2)$，$t_5=(x=-1, y=0, z=3)$，$t_6=(x=0, y=-1, z=3)$，$t_7=(x=0, y=0, z=1)$和 $t_8=(x=-1, y=-1, z=1)$。若将补充设计的测试用例加入原测试用例集 U 中，则集合 U 中包含了大量的冗余，导致测试成本的急剧增加。为约简测试集合 U 中冗余的测试用例，本文随后给出了测试知识的表示及约简方法。

## 3 测试知识

知识的表示来源于粗糙集理论[9]，该理论是用于刻画不完整和不确定性的一种数学工具，能对各种不精确、不一致、不完整信息进行有效的分析和处理，发现隐含的知识，揭示潜在的规律。然而，知识库中的冗余知识既浪费了存储空间，又干扰了人们基于知识库的信息决策，因此，需要在维持知识库的分类和决策能力不变的情况下，删除不必要的冗余知识。传统知识的约简方法包括构造区分矩阵的方法[10]、简化分明矩阵法[11]、绝对属性约简方法[8]和变论域知识约简方法[12]等。本节采用知识表示的方法对测试知识进行建模，再采用绝对属性约简方法化简测试知识系统。

### 3.1 测试知识的表示

与传统知识的表示不同，测试知识包含 2 个部分：测试

**基金项目：**上海市自然科学基金资助项目(09ZR1412100)

**作者简介：**朱 彬(1976－)，男，讲师、博士研究生，主研方向：软件形式化方法，软件测试

**收稿日期：**2011-03-07　**E-mail：**tozhubin@163.com

用例集合和测试需求集合。

**定义 1**(测试知识表示系统) 一个测试知识表示系统 TS=(U, A)，其中，U 是一个非空有限的测试用例集合；A 是一个有限的测试需求集合，并满足以下 2 个条件：

(1)每一个测试需求 a∈A 都是一个完全函数，即 a:U→Va，其中，Va⇨{0,1}是测试需求 a 的取值集合。对于任意的 x∈U，a(x)=0 表示某条测试用例并不满足测试需求 a，a(x)=1 表示某条测试用例满足测试需求 a。

(2)对于测试需求 A 的每个子集 B⊆A，都有一个与之相关的二元关系 IND(B)，称之为不可区分关系，其定义如下：

IND(B)={(x,y)|(x,y)∈U×U 且对于每一个 a∈B 都有 a(x)= a(y)}

**定义 2**(属性) 每个测试需求 a∈A 都称为测试知识表示系统 TS 上的一个属性。

属性 a 也可认为是关系 IND(a)的名称，或称之为等价关系 IND(a)所表示测试知识的名称。

对于程序 A，测试用例集合 U={$t_i$|1⩽i⩽8}，测试需求集合 A={a,b′,c,d,e,f,g,h}，其中，对于任意 a∈A，a 都是测试知识表示系统 TS 的一个属性。依据定义 1 和定义 2，对程序 A 的测试知识进行建模。

由定义 1 和定义 2 可知，二元关系 IND(A)可得到属性 A 的等价关系，如表 1 中 IND(c)为{$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$}和{$t_8$}，表明属性 c 列中 Vc=1 的用例集合为{$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$}，Vc=0 的用例集合为{$t_8$}。

**表 1　程序 A 的测试知识表示系统**

| U | a | b′ | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|---|
| $t_1$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| $t_2$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| $t_3$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $t_4$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| $t_5$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $t_6$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $t_7$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| $t_8$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

表 1 中的 8 个等价关系 IND(a)、IND(b′)、IND(c)、IND(d)、IND(e)、IND(f)、IND(g)和 IND(h)的等价类族集如下：

U/IND(a)={{$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$, $t_8$}}

U/IND(b')={{$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$, $t_8$}}

U/IND(c)={{$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$}, {$t_8$}}

U/IND(d)={{$t_3$, $t_5$, $t_6$}, {$t_1$, $t_2$, $t_4$, $t_7$, $t_8$}}

U/IND(e)={{$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$}, {$t_8$}}

U/IND(f)= {{$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$}, {$t_8$}}

U/IND(g)={{$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$}, {$t_8$}}

U/IND(h)={{$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$, $t_8$}}

### 3.2　测试知识约简

**定义 3**(可识别集) 设集合 $S_{1(a)}$，a∈A，$S_{1(a)}$∈ U/IND(a)，1(a)表示属性 a 列上所有为 1 的值，则称 $S_{1(a)}$为属性 a 在 U 上的可识别集。

依据定义 3，表 1 中 U 上的可识别集合为：$S_{1(a)}$={$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$, $t_8$}，$S_{1(b')}$={$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$, $t_8$}，$S_{1(c)}$={$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$}，$S_{1(d)}$={$t_3$, $t_5$, $t_6$}，$S_{1(e)}$={$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$}，$S_{1(f)}$= {$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$}，$S_{1(g)}$={$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$}，$S_{1(h)}$={$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$, $t_8$}。

**定义 4**(不可识别集) 设集合 $S_{0(a)}$，a∈A，$S_{0(a)}$∈U/IND(a)，0(a)表示属性 a 列上所有为 0 的值，则称 $S_{0(a)}$为属性 a 在 U 上的可识别集。

依据定义 4，表 1 中 U 上的不可识别集合为：$S_{0(a)}$=∅，$S_{0(b')}$ =∅，$S_{0(c)}$={$t_8$}，$S_{0(d)}$={$t_1$, $t_2$, $t_4$, $t_7$, $t_8$}，$S_{0(e)}$={$t_8$}，$S_{0(f)}$={$t_8$}，$S_{0(g)}$={$t_8$}，$S_{0(h)}$=∅。

**定义 5**(识别等效性) 对于测试知识表示系统 TS=(U,A)上的任意 2 个属性 a,b∈A，若 $S_{1(a)}$=$S_{1(b)}$且 $S_{0(a)}$=$S_{0(b)}$，则称属性 a 和 b 在 U 上具有识别等效性。

在表 1 中，具有识别等效性的属性集有{a, b′, h}和{c, e, f, g}。依据识别等效性可以对表 1 进行属性约简。

约简规则：在知识表示系统 TS 中，删除所有具有识别等效性的冗余属性。

依据约简规则，表 1 的测试知识表示系统被约简为表 2。

**表 2　约简后的测试知识表示系统**

| U | a | c | d |
|---|---|---|---|
| $t_1$ | 1 | 1 | 0 |
| $t_2$ | 1 | 1 | 0 |
| $t_3$ | 1 | 1 | 1 |
| $t_4$ | 1 | 1 | 0 |
| $t_5$ | 1 | 1 | 1 |
| $t_6$ | 1 | 1 | 1 |
| $t_7$ | 1 | 1 | 0 |
| $t_8$ | 1 | 0 | 0 |

由于采用属性约简方法，因此在约简后的测试知识表示系统 TS=(U,A)中，测试用例集合 U 不变。又由于约简的是具有识别等效性的冗余属性，因此测试需求集合 A 的减小不会影响测试用例对测试需求的覆盖能力。

## 4　决策树

为了从约简后的测试知识表示系统 TS 中获得 U 的一个有效子集，本文给出构造决策树的测试子集选取方法。决策树的构造思想如下：

(1)将集合 U 中所有的元素作为根结点，构造一棵决策树 T。

(2)依次从集合 A 选取一个属性 a，将 $S_{1(a)}$作为树的左结点，$S_{0(a)}$作为树的右结点。

(3)若左结点(或右结点)为空或者仅包含一个元素，则它们为叶子结点。

(4)若左结点(或右结点)包含 2 个以上的元素，则跳转到步骤(2)。

(5)当集合 A 中所有元素均已取到，则决策树构造完成。

依据上述思想，本文构造了表 2 的决策树 T，见图 1。

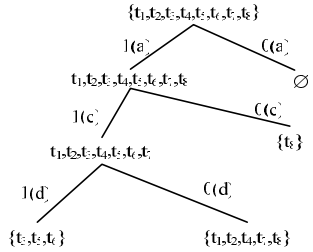


**图 1　知识表示系统 TS 的决策树 T**

依据决策树的叶子结点可以方便地选取集合 U 的一个子集。具体选取思想如下：

(1)访问决策树 T 中的一个叶子结点。

(2)若结点为空，则访问下一个叶子结点。

(3)若结点不空，则：1)若是单元素结点，则取这个元素；2)若是多个元素的结点，则取任意一个元素。

(4)当所有叶子结点都被访问到时，将所有取出的元素构成新的集合 U′，该集合就是测试用例集合 U 的子集。

依据测试子集的构造思想，由图 1 中的决策树得到一个集合 U′={$t_8$, $t_3$, $t_1$}。

测试知识表示与传统知识表示的联系和区别如下：

(1)传统知识表示和测试知识表示均采用了二元关系表

的形式，不同之处是测试知识是由测试用例集合和测试需求集合构成的。

(2)在传统知识表示中，关系表的组成元素可取任意数字，在测试知识表示中，二元关系表的组成元素仅为 0 和 1。

基于决策树的测试子集选取方法与传统的测试子集选取方法的区别如下：

(1)在传统的测试子集选取方法中，测试需求为程序控制流图中的边，而在基于决策树的选取方法中，测试需求为程序对应的行，因此，本文方法不需要构造程序的流程图。

(2)传统的测试子集选取方法仅能获得一个测试子集，而基于决策树的测试子集选取方法能够获得所有子集，如可从图 1 的决策树中获得另外的回归测试子集{$t_8$, $t_6$, $t_4$}和{$t_8$, $t_5$, $t_2$}等。

(3)传统的测试子集选取方法是从测试用例对测试需求覆盖的角度进行测试用例的冗余约简，约简对象是测试用例，而基于决策树的测试子集选取方法是对属性(测试需求)进行约简，再利用决策树获得测试用例子集。

## 5 测试知识约简算法及决策树构造算法

为了在计算机中实现基于决策树的回归测试子集选取方法，本节设计了测试知识表示的存储结构、测试知识约简的算法及决策树的构造算法。

(1)存储结构

**定义 6**(测试覆盖信息) 在测试知识表示系统 TS=(U,A)中，测试用例 t∈U 对应 TS 中某一行的信息被称为 t 对 A 的测试覆盖信息。

测试知识表示系统 TS 包含测试用例集合和测试需求集合，因此，需要分别设计测试用例和测试需求对应的存储结构。本文采用字符串作为测试覆盖信息的存储结构，采用哈希表关联测试需求、测试用例及测试覆盖信息。如表 1 中，测试用例 $t_1$ 的测试覆盖信息可以定义为字符串“11101111”，而哈希表的 key 域为测试用例 $t_1$，哈希表的 value 域为字符串“11101111”，该哈希表的存储结构如图 2 所示。为获得测试需求的可识别集，本文定义了另一个哈希表，其中，可识别集采用字符串进行存储，而哈希表的 key 域为测试需求，哈希表的 value 域为代表可识别集的字符串，该哈希表的存储结构如图 3 所示。

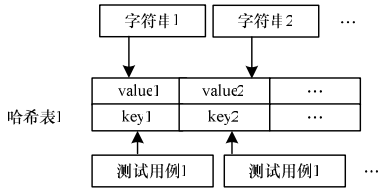


**图 2 测试用例与测试覆盖信息的哈希表存储结构**

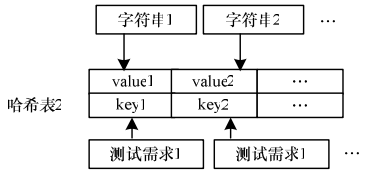


**图 3 需求与可识别集的哈希表存储结构**

哈希表存储结构的优点是可以利用哈希表的 put()和 get()函数直接进行存取操作。字符串存储结构的优点是可以利用字符串的各种函数进行比较操作。

(2)初始化操作

定义 2 个哈希表 hash1 和 hash2，其中，hash1 存储测试用例与测试覆盖信息；hash2 存储测试需求和可识别集。hash1 的内容可以直接给出，hash2 的 value 域内容可由 hash1 获得，具体的实现伪代码如下：

```
for ( int i=0; i<hash1.length; i++){
      str1=hash1.get(k1(i));//k1(i)为测试用例 i
      for ( int j=0; j<str1.length; j++){
            str2=str1.charAt(j);
            if (str2==”1”){
                str2=hash2.put(k2(j))+k1(i);
hash2.put(k2(j), st2);
                }
}}
```

假定测试用例的总数为 $n$，测试需求的总数为 $m$，则上述伪代码的时间复杂度为 $O(n\times m)$。

(3)约简算法

测试知识的约简实际上是对 hash2 中内容的约简。由于 hash2 的 value 域为字符串，因此可以利用字符串匹配方法约简冗余的可识别集，具体的实现伪代码如下：

```
str=“”;
for ( int i=0; i<hash2.length; i++ ){
      str1=hash2.get(k2(i))+ “*”;
      if (str.indexOf(str1)==-1){
          str=str1+k2(i)+ “#”;
}}
str=str.substring(0, str.length-1);//删除最后的“#”
str3[]=str.split(“#”);
for ( int i=0; i<str3.length; i++){
      value=str3[i].split(“*”)[0];
      key=str3[i].split(“*”)[1];
      hash3.put(key, value);//hash3 为临时表
}
hash2=hash3;
```

上述算法的时间复杂度为 $O(m)$。

(4)决策树生成算法

决策树的构造需要用到可识别集和不可识别集，其中，可识别集由 hash2 的 value 域获得，不可识别集可以采用类似于 hash2 的构造算法实现。在构造决策树时，若测试需求 a 的可识别集是树 T 的左孩子，不可识别集是树 T 的右孩子，则具体的实现伪代码如下：

```
//构造不可识别集的哈希表 hash3
for ( int i=0; i<hash1.length; i++){
    str1=hash1.get(k1(i));//k1(i)为测试用例 i
    for ( int j=0; j<str1.length ; j++){
      str2=str1.charAt(j);
      if (str2==”0” && hash2.get(k1(j))!=null){
          str2=hash3.put(k1(j))+k1(i);
hash3.put(k1(j), st2);
          }}}
//构造决策树 T
struct TreeNode{
    String data;
    struct TreeNode *left;
    struct TreeNode *right;
}
TreeNode tree, node;
*root=tree;//root 为树的根指针
str=””;
for (int i=0; i<hash1.length; i++){
```

```
    str=str+k1(i);//k1 为 hash1 的 key 值
}
tree.data=str;
List lt.add(tree);//链表
//采用层次构造决策树
for (int i=0; i<hash2.length; i++){
    tree=lt.getNext();//取链表中下一个元素
value1=hash2.get(k2(i));
    value2=hash3.get(k2(i));
    node.data=value1;
    tree.left=node;
    if (value1=null || value1.length=1)
        lt.add(node);
    node.data=value2;
    tree.right=node;
if (value2=null || value2.length=1)
        lt.add(node);
}
return root;//返回决策树的根结点指针
```

在上述算法中，若哈希表 hash2(或 hash3)的 key 域中没有测试需求 a，则函数 hash2.get(a)=null(或 hash3.get(a)=null)，即左子树(或右子树)为空。value1.length=1 表明该结点中仅包含一个元素。决策树构造算法的最大时间复杂度为 $O(n\times m)$。依据决策树构造算法返回的 root 指针可以遍历这棵决策树，相关的内容可参考二叉树的遍历算法，本文不再累述。

## 6 结束语

在回归测试中，测试用例集合大小的增加会导致软件测试成本的急剧增长。为约简测试用例集合中的冗余测试用例，本文提出基于决策树的测试子集选取方法。与传统的启发式选取测试用例子集方法不同，本文方法采用知识表示的方式对测试知识进行建模和约简，将约简后的测试知识构造成一棵决策树，由决策树获得相关的测试用例子集。本文的回归测试用例集合约简方法便于测试用例子集的计算机实现。

## 参考文献


[1] Kim Jung-Min, Adam P, Gregg R. An Empirical Study of Regression Test Application Frequency[J]. Software Testing, Verification & Reliability, 2005, 15(4): 257-279.

[2] 王晓华, 张 涛, 尚景亮, 等. 多维标度法选择回归测试子集[J]. 计算机科学, 2010, 37(11): 131-134, 140.

[3] Harrold M J, Gupta R, Soffa M L. A Methodology for Controlling the Size of a Test Suite[J]. ACM Transactions on Software Engineering and Methodology, 1993, 2(3): 270-285.

[4] Chen T Y, Lau M F. A New Heuristic for Test Suite Reduction[J]. Information and Software Technology, 1998, 40(5): 347-354.

[5] Jeffrey D, Gupta N. Improving Fault Detection Capability by Selectively Retaining Test Cases During Test Suite Reduction[J]. IEEE Transactions on Software Engineering, 2007, 33(2): 108-123.

[6] Black J, Melachrinoudis E, Kaeli D. Bi-criteria Models for All-uses Test Suite Reduction[C]//Proc. of International Conference on Software Engineering. Edinburgh, Scotland, UK: [s. n.], 2004: 106-115.

[7] 蔡素梅, 梅登华. 基于动态切片和 UML 图的回归测试用例生成[J]. 计算机工程, 2009, 35(8): 70-72.

[8] 刘 清, 刘少辉, 郑 非. Rough 逻辑及其在数据约简中的应用[J]. 软件学报, 2001, 12(3): 415-419.

[9] Pawlak Z. Rough Sets[J]. International Journal of Computer and Information Sciences, 1982, 11(5): 341-356.

[10] Skowron A, Rauszer C. The Discernibility Matrics and Function in Information System[M]//Slowinskied R. Intelligent Decision Support Handbook of Application and Advances of the Rough Sets Theory. Dordreecht, Netherlands: Kluwer Academic Publishers, 1992.

[11] Guan J W, Bell D A. Rough Computational Methods for Information Systems[J]. Artifical Intelligence, 1998, 105(1/2): 77-103.

[12] 王亚英, 张春慨, 邵惠鹤. 变论域知识约简算法[J]. 上海交通大学学报, 2002, 36(4): 566-569.


编辑 张正兴

(上接第 29 页)

上述虚拟执行结果表明实例 $I$ 能在工作流模型 $W'$ 产生相同的历史记录，符合公理 1 的迁移条件，可以迁移到工作流模型 $W'$ 正常运行，该虚拟过程忽略大多数任务类型活动对数据的操作，仅关注 $N_6$ 和 $N_7$ 活动对数据单元的操作。

## 7 结束语

本文给出一种虚拟执行算法，适用于模型结构变化和循环结构，结合回退、撤销、异常(检测)处理策略，对于不符合迁移准则的实例，修改历史记录后进行迁移[9]。与同类方法相比，算法考虑数据流、模型结构、循环结构等情况，具有简洁易于实现等特点；由于工作流模型相对复杂(业务逻辑建模本身比较复杂)，因此模型的正确性需要进一步验证。

## 参考文献


[1] Rinderle S, Reichert M, Dadam P. Correctness Criteria for Dynamic Changes in Workflow Systems[J]. Data & Knowledge Engineering, 2004, 50(1): 9-34.

[2] Smari W W, Donepudi S, Kim Seung-yun, et al. Efficient Handling of Changes in Dynamic Workflow Systems[Z]. 2003.

[3] 铁菊红, 李长河, 彭 辉. 一种基于虚拟执行规则的工作流实例迁移算法[J]. 计算机应用, 2006, 26(3): 688-691.

[4] 杨书新, 王 坚. 状态的工作流实例迁移算法[J]. 计算机集成制造系统, 2008, 14(2): 372-378.

[5] Workflow Management Coalition. The Workflow Reference Model(WFMC-TC00-1003 Issue 1.1)[Z]. 1995.

[6] Ceris C. Workflow Evolution[J]. Data and Knowledge Engineering, 1998, 24(3): 211-238.

[7] van Aalst W. Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change[D]. Eindhoven, UK: Eindhoven University of Technology, 2000.

[8] 周天明, 王敏毅, 姚绍文. 一种基于扩展任务结构的工作流实例迁移方法[J]. 软件学报, 2003, 14(4): 757-762.

[9] 刘 畅, 王晓琳, 曾广周, 等. 迁移工作流系统中的工作位置异常处理模型[J]. 计算机工程, 2010, 36(6): 97-99.


编辑 张正兴