程序切片技术在软件测试中的应用*

孙继荣[1,2], 李志蜀[1], 王 莉[1], 殷 锋[1,3], 金 虎[1]
(1. 四川大学 计算机学院, 四川 成都 610065; 2. 四川师范大学 软件重点实验室, 四川 成都 610068; 3. 西南民族大学 计算机科学与技术学院, 四川 成都 610041)

摘 要: 基于程序切片的软件测试是一种以程序或程序和需求相结合为基础的测试,它根据程序的不同切片来缩小软件的测试范围、提高软件测试的效率、辅助测试数据的自动生成等。同时由于程序切片不仅考虑了数据依赖和控制依赖,还考虑了程序存在的其他各种依赖关系,使得测试的准确性得到提高。详细阐述了目前存在的各种切片技术及其应用领域,重点探讨了目前切片技术在测试领域中的具体应用。

关键词: 程序切片; 软件测试; 数据依赖; 控制依赖; 测试数据自动生成
中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2007)05-0210-04

Overview of Software Testing Based on Program Slice

SUN Ji-rong[1,2], LI Zhi-shu[1], WANG Li[1], YIN Feng[1,3], JIN Hu[1]
(1. *School of Computer, Sichuan University, Chengdu Sichuan* 610065, *China*; 2. *Key Laboratory of Software, Sichuan Normal University, Chengdu Sichuan* 610068, *China*; 3. *College of Computer Science & Technology, Southwest University for Nationalities, Chengdu Sichuan* 610041, *China*)

Abstract: Author discussed different slicing methods and there applications to software engineering, elaborately to software testing. Software testing based on slicing can reduce the range of software under testing, improve the testing efficiency, and give a convenient hand to the process of automatic test data generation, etc. At the same time, this technique can enhance the testing accuracy and sufficiency.

Key words: program slice; software testing; data dependency; control dependency; automatic test data generation

程序切片是一种程序分析和理解技术。它通过把程序减少到只包含与某个特定计算相关的那些语句来分析程序。其概念最早是 1979年由 Mark Weiser[1]提出来的。他观察到程序员在调试过程中脑海中就有关于程序的某种抽象,人们在调试一个程序时总是从错误语句 s开始,并沿着依赖关系跟踪到它影响的程序部分。程序切片的发展基本成熟,在理论和应用方面的研究均取得了可喜的进展,特别是在程序的调试、测试、分解和集成、软件维护、代码理解以及逆向工程等领域具有广泛的应用。本文将侧重介绍程序切片技术在软件测试领域的应用。

1 程序切片简介

Weiser认为一个切片与人们在调试一个程序时所做的智力抽象相对应。他定义的程序 P的切片 S是一个可执行的程序,对某个兴趣点 s处的变量 v而言 ($<s, v>$称为切片准则), S由程序 P中可能影响 s处变量 v的值的所有语句构成。这个可执行部分相对于程序 P在功能上是等效的[2]。他提出的切片算法是基于数据流分析的,通过遍历控制流图分析数据依赖和控制依赖关系,从源程序中移去零条或多条语句从而得到过程内的切片。

定义 1 设 s是程序流图 CFG中的任一节点,定义:

(1)定义集 $Def(s) = \{x / x$是语句 s中值被改变了的变量$\}$;

(2)引用集 $Ref(s) = \{x / x$是语句 s中引用的变量$\}$。

定义 2 数据依赖:如果节点 n、m 满足以下两个条件, 则称 n数据依赖于 m:

(1)如果存在一个变量 v满足 $v \in def(m) \cap ref(n)$;

(2) G上存在一条由节点 m到 n的路径 p,对于路径上的其他节点 $m \in p - \{m, n\}$, $v \notin def(m)$。

定义 3 控制依赖: 如果节点 n、m 满足如下条件, 则称 n控制依赖于 m:

(1) G上存在一条由节点 m到节点 n的路径 p, 对于路径上的其他节点 $m \in p - \{m, n\}$, n是 m 的后必经点;

(2) n不是 m的后必经点。

程序切片不仅与兴趣点定义和使用的变量有关,还与影响该变量的值的语句和谓词以及受该变量的值影响的语句和谓词有关。程序切片过程中,采用合适的结构来表示语句间的依赖关系,寻找与兴趣点具有直接或间接数据依赖和控制依赖的

收稿日期: 2006-04-12; **修返日期**: 2006-05-20 **基金项目**: 四川省重点科技资助项目 (05GG021-003-2);四川师范大学软件重点实验室资助项目;西南民族大学青年重点资助项目 (05NQZ001)

作者简介:孙继荣 (1973-),女,讲师,博士,主要研究方向为实时软件工程、软件测试与软件可靠性、网络与信息系统 (sunjr@scrtvu.net);李志蜀 (1946-),男,教授,博导,主要研究方向为软件测试、网络与信息系统;王莉 (1970-),女,讲师,博士,主要研究方向为软件可靠性、网络与信息系统;殷锋 (1972-),男,副教授,博士,主要研究方向为网格、中间件; 金虎 (1973-),男,讲师,博士,主要研究方向为软件测试与软件可靠性、网络与信息系统.

节点是生成切片的重点。切片算法基本过程为:首先寻找语句 s的变量 v所直接数据依赖或控制依赖的节点;然后寻找这些新节点所直接数据依赖或控制依赖的节点;一直重复下去,直到没有新节点加进来为止;最后将这些节点按源程序的语句顺序排列,即为程序 P的关于语句 s的切片 S。

2 程序切片分类

程序切片技术的发展经历了从静态到动态、从前向到后向、从单一过程到多个过程、从过程型程序到面向对象程序、从非分布式程序到分布式程序等几个方面。

2.1 静态切片与动态切片

Weiser最初提出的程序切片概念就属于静态切片(Static Slicing)范畴。静态切片是在编译时间,即程序尚未运行时进行切片;该技术对程序的输入不作任何假设,所作的分析完全以程序的静态信息为依据。因此静态切片包含了所有与兴趣点处变量相关的语句,考虑了程序中所有可能的执行路径。其缺点是:①容易包含不相关节点,具有很大的冗余性。主要是由于程序的执行路径无法静态判断,尤其是数组和指针变量无法静态确定。②使用该技术的工作量较大。因为要分析程序所有可能的执行轨迹,静态切片技术一般用于程序理解与软件维护方面。

实际应用中,人们往往更关注某一具体输入下,程序实际执行时影响兴趣点处某一变量值的那些语句。

Korel等人[3]提出了动态切片(Dynamic Slicing)的概念。切片准则是一个三元组 (s, v, x),s、v的定义不变,x是一个输入序列,在该输入下与源程序计算出的该变量的值是相同的。动态程序切片计算过程使用用户的实际输入 x产生的精确数据流信息进行分析,通常情况下比静态切片要小得多;动态切片的另一个优点是在程序运行时间进行切片,数组中的每个元素和指针变量的值得到确定,因此动态切片要比静态切片精确得多。动态切片的缺点是需要保留程序的执行历史记录。采用这一技术,每一次的计算工作量较小,但每一次的计算都不尽相同,因此动态切片技术多用于程序调试、测试方面。动态切片还可以用在理解大型程序方面。图 1给出了静态切片与动态切片的例子。

2.2 前向切片与后向切片

如果切片 S由程序 P中可能影响 s处变量 v的值的所有语句组成,这是一种前向切片(Forward Slicing)。与此相反,后向切片(Backward Slicing)S是程序 P中兴趣点 s处变量 v的值影响到的语句和谓词组成的集合。图 2给出了前向和后向切片的例子。

2.3 对象切片

自 Weiser提出程序切片概念后,人们提出了许多用于过程型程序的切片方法,但这些方法并不适用于 OO程序。因为OO编程语言提出了一些新的概念与特性,如类、对象、动态绑定、封装、继承、消息传递以及多态。所以面向对象程序切片不仅要考察语句和数据之间的依赖关系,还要考察各个类之间的关系。为了获得更准确、更有效的程序切片,D Liang和 M. J. Harrod提出了对象切片(Object Slicing)技术这一概念;他们的工作是通过扩展系统依赖图实现的。目前对它的研究更多是侧重于静态切片这一部分,而且基本都是基于依赖图的。李必信等人提出了一种逐步求精的基于 OO程序的分层切片方法。

```
main()
{
  int x, y, z;
  scanf("%d",&x);
  if (x<0)
     y=-1;
  else if (x = = 0)
        y = =1;
     else y = =1
  printf("y=%d",y);
  z=2y
}
```

```
main()
{
  int x, y, z;
  scanf("%d",&x);
  if (x<0)
     y=-1;
  else if (x = = 0)
        y = =1;
     else y = =1
  printf("y=%d",y);
  z=2y
}
```

图 1　语句 printf("y=%d",y)的静态切片和当输入 x=0 的动态切片

```
main()
{
  int x, y, z;
  scanf("%d",&x);
  if (x<0)
     y=-1;
  else if (x = = 0)
        y = =1;
     else y = =1
  printf("y=%d",y);
  z=2y
}
```

```
main()
{
  int x, y, z;
  scanf("%d",&x);
  if (x<0)
     y=-1;
  else if (x= = 0)
        y= =1;
     else y = =1
  printf("y=%d",y);
  z=2y
}
```

图 2　语句 printf("y=%d",y)的后向切片(左)和前向切片(右)

2.4 其他切片技术

(1)准静态切片(Quasi Static Slicing)。准静态切片的产生是对于一些特殊程序 P,某些输入值可以确定,而另外一些输入不停变化;在此情况下对程序 P进行分析时,需要混合使用静态切片和动态切片方法。在计算切片时,固定一部分输入值,使得程序 P中的某些特定子路径得以执行,可以删除一些分支。这样得到的切片比纯粹的静态切片要精简得多。部分静态切片用于程序理解和转换。

(2)同步动态切片(Simultaneous Dynamic Slicing)。Hall[4]扩展了动态切片,他将一个测试集而不是单个测试用例用于程序动态切片中。一个测试集可以看成是对某个需求的完全测试用例集。Hall提出的同步动态切片并不是简单地对每个测试用例的动态切片进行的并集,而是采用了迭代算法,从初始切片开始在迭代过程中逐步增长为大型的动态切片。同步动态切片适用于定位程序 P的某个需求有关的代码部分。

(3)分解切片技术(Decomposition Slicing)。它是一种以把程序分解为不同模块为目的的切片技术。分解切片是由一组关注某一变量的程序切片构成的集合,可以捕获程序中对某一变量的所有计算。分解切片不依赖于语句在程序中的位置,构成分解切片的程序切片按照一定的规则排列成网格(Lattice),通过使用这种网格来实现对程序的分解。分解切片技术适用于回归测试方面。

(4)条件切片(Conditioned Slicing)。Confora等人提出的条件切片技术通过增加一个条件扩展了传统的静态切片准则[5],这个条件对应着程序的某个或某些初始状态。在进行切片算法时,只有满足该切片条件的那些输入才会被分析。条件切片技术主要用于程序理解和软件重用方面。

(5)无定型切片技术(Amorphous Slicing)。该技术施加更广泛的切片准则,在简化源程序的过程中充分利用传统切片技术保留源程序语义映射的简化功能。无定型切片的特点使其更适合程序理解领域,而传统的切片技术则更多地应用在调试领域。

3 程序切片技术在软件测试中的应用

基于程序切片的软件测试是一种以程序或程序和需求相结合为基础的测试,它根据程序的不同切片来缩小软件的测试范围,并提高软件测试的效率。同时由于程序切片考虑程序存

在的各种依赖关系(不仅仅是数据依赖和控制依赖),使得测试的准确性得到提高。因为任何一个程序都可以和一组程序切片的并集等价,所以测试每个切片实际就是测试了整个程序,从而保证了测试的充分性。软件测试中,测试数据生成是至关重要的一步。引入切片技术的目的是,计算程序在某一具体输入下兴趣点处变量的当前值,以辅助测试数据的生成,提高测试数据的生成效率。

3.1 基于切片技术的测试数据自动生成

将程序切片技术应用到软件测试数据生成中,只要重点关注程序中与指定路径上兴趣点相关的那部分语句,即可有效提高基于路径的测试数据生成效率。

基于程序切片的软件测试数据生成系统主要由预处理器、程序切片生成器、测试数据生成器三部分组成,如图3所示。

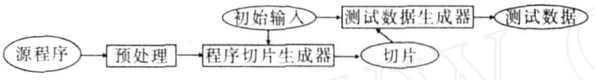

图3 基于切片的测试数据自动生成器框架

(1)预处理器。它对源程序进行预处理,包括标记语句、分析程序数据流信息、记录变量的定义—引用属性、确定语句间的支配关系等。

(2)程序切片生成器。它结合初始输入(可手工或随机生成),根据程序指定执行路径,计算源程序各兴趣点(主要是谓词节点)处的程序切片。

(3)测试数据生成器。它比较程序实际执行路径与指定路径的差异,从程序切片生成器产生的切片集合中选出该路径上兴趣点处的切片 s;在测试数据生成器中按照一定规则调整程序输入,并在新的程序输入上执行切片 s,直到该兴趣点满足路径要求。

该过程的处理算法主要有分支函数最小化、松弛迭代法、遗传算法等。

Neelam Gupta等人于1998年提出了迭代松弛法自动生成测试数据[6]。该方法采用程序切片思想,从定义域中任意选择一组输入 x,通过静态/动态数据流分析确定指定路径上的各分支谓词函数对于输入变量的依赖关系(即使在输入 x 下该路径不经过),构造谓词片和动态切片,建立这些谓词函数关于当前输入的线性算术表示;然后建立输入变量的增量线性约束系统,进而建立输入变量的增量线性方程系统,求解方程后得到关于指定路径的测试数据。

Gupta提出的谓词片是从路径上的输入语句和赋值语句静态计算得到的。它既不是传统意义上的静态切片,也不是动态切片,而是面向路径的静态切片。指定路径 P 上的谓词节点 s 的谓词片 $Slice(s, P)$ 是路径 P 上的 s 谓词语句之前的 s 所数据依赖的(包括直接和间接数据依赖的输入变量),仅由输入语句和赋值语句组成的程序片。通过引入谓词片,无须考虑其他分支谓词的计算结果即可计算路径 P 上的每个分支谓词的结果。根据输入 x,通过执行谓词片 $Slice(s, P)$ 的语句,并从动态数据依赖图上提取动态切片,确定谓词节点的输入变量依赖集,Gupta等人采用均差作为谓词函数 F 关于输入变量导数的近似,得到谓词函数 F 关于输入 x 的线性算术表示。

同时引入谓词残量,表示根据输入 x,谓词片执行之后计算相应谓词函数 F 所得之值。该谓词残量用来指示为使分支谓词得到满足,谓词函数允许改变的范围。对路径 P 上的每个谓词函数,迭代松弛法利用上面求得的线性算术表示和相应的谓词残量共同生成一个关于程序输入的增量线性约束。

指定路径 P 上的各分支谓词函数关于当前输入 x 的增量线性约束表示一切构成了线性约束方程组,利用迭代松弛法求解。求解后得到各输入变量的增量,从而获得一组新输入。如果路径 P 上的谓词函数都是输入变量的线性函数,并且所有输入变量均无整数限制时,该方法迭代一次后,或者找到 P 的解,或者保证 P 不可行。当谓词函数中含有输入变量的非线性函数时,该方法可能需要迭代多次。该方法允许变量的数据类型为整数类型、实数类型。

文献[7]提出了一种基于谓词切片的字符串测试数据自动生成方法。对指定路径 P 上给定的字符串谓词以相应的动态切片标准为准则,将该路径上与给定谓词有关的语句以及谓词抽取出来,形成关于输入变量的谓词切片;对任意输入,通过执行该谓词切片,获取谓词中变量的当前值,进而对谓词变量中的每一字符,构造其线性分支函数,进行分支函数最小化,动态生成给定字符串谓词边界的 ON-OFF 测试点。

文献[8]提出了一种基于前向分析的动态切片测试数据生成方法。首先对程序细化分块,以块之间的支配关系来表示块与块之间的嵌套包含关系。该方法以 Korel 提出的程序前向动态切片算法为基础,并结合块与块之间的支配关系计算各节点的动态切片;然后调整输入,使各分支节点的执行条件得到满足。

V. M. Vedula等人将程序切片思想扩展到硬件设计语言 Verilog,并对硬件进行分层自动测试[9]。一般而言,用 Verilog 描述的数字系统都是由多个模块分层组成;每个模块是一个不中断的程序,由若干过程组成,这些过程之间通过一些共享信号进行通信。Vedula等人首先将每个待测模块 M_1 视为独立的部分,而将硬件设计的其他部分均抽象为一个环境模块 $\overline{M}_1$,构造每个待测模块兴趣点处的约束切片:M_1 的约束切片是一个可执行的 $\overline{M}_1$ 的子程序,采取后向切片算法构造 M_1 的输入信号的约束切片,前向切片算法构造输出信号的约束切片。然后根据约束切片产生每个待测模块的检错测试集。

3.2 程序切片技术在错误定位中的应用

软件测试中,错误定位是一个非常复杂而费时的过程,W. Eric Wong等人利用测试用例驱动的可执行动态切片和块之间的数据依赖关系,高效准确地在小范围的代码内对程序进行错误定位[10]。对于某个需求,有一组测试集,某些测试用例执行的结果正确,某些错误。基于每个测试用例,进行动态切片。Wong认为,如果该代码出现在成功切片中的次数越频繁,越不可能是错误代码。在此先验下,采取两阶段逐步扩大可疑代码段的方法,即精练法和扩展法。

(1)如果某个测试用例的实际结果与预期结果不符,生成该输入下的动态切片 F;同时选取三个测试相同需求(功能)的结果正确的测试用例,生成相应的动态切片 T_1、T_2、T_3。

(2)精练法:依次测试 $F-(T_1 \cup T_2 \cup T_3)$、$F-(T_1 \cup T_2)$ 和 $F-T_1$ 是否含有 Bug,如果有,则表示已经成功地在极小的代码段内定位到错误,算法终止。

(3)逐步扩展法:以 αΔβ表示,如果 α中某语句定义了某变量而 β中某语句使用了该变量,或者反之,都称为 β直接数据依赖于 α,则将 α中的这些语句添加进 β中。令 $\chi^1 = F - T_1$,依次测试 $\chi^k = [F - (F - T_1)]\Delta\chi^{k-1}$ $(k = 1, 2, \cdots)$是否含有 Bug。如果发现 Bug或者没有新代码增加进来,则算法终止。

该算法在最坏情况下需要测试 Bug的范围就是运行失败的动态切片。

3.3 程序切片技术在回归测试中的应用

软件测试从本质上来说是发现程序错误的过程,即一旦找到错误就进行修改。另外,软件维护过程中,由于功能的增加或改变,也需要对源程序进行修改。对程序的任何修改都可能引入新的错误,因此就需要进行回归测试。按照传统的测试方法,必须对整个程序进行重新测试。如果对软件仅作小的改动,而要对它重新测试则既费时费钱,效率又极低。如果将切片技术应用回归测试中,解决的方法就会经济方便得多。

方法一:程序有若干输出,如果某个输出变量出现错误,使用分解切片技术,那么关于该变量的所有计算组成一个模块,维护人员可以直观地判断一个模块中哪些语句和变量可以被安全地修改,即这种修改不会扩散到其他模块中;哪些语句不能被随意修改。K B. Gallagher等人还给出了一些用于禁止那些会影响到其他模块的修改原则。这样,维护者可以在一个模块中测试对语句或变量所作的修改,而不必担心这种修改会影响到其他部分。

方法二:在回归测试中,为了找到这种修改可能影响到的语句,首先找到新旧版本的不同之处,以及两者的切片与依赖图不同之处(无须考虑那些具有相同切片的节点,这些节点在新旧版本中的作用不变),标记为影响节点;然后计算其后向切片和前向切片,并取两者的并集进行测试。

这样极大地缩小了测试的范围,不必对源程序进行分析。

3.4 面向对象切片技术在测试中的应用

李必信等人从程序切片着手,根据测试人员的需要对源程序进行分层切片,把对整个程序的测试按照一定的规则转换为只对程序切片进行测试[11]。

他们对 Java源程序按数据级、方法级(过程级)、对象类级、类层次结构等进行分层:首先对源程序进行分层,抽象出分层切片模型,并确定层次之间的各种依赖关系和流关系;然后在不同层次上进行数据流和控制流以及依赖关系分析建立各种流图和依赖关系图,从顶层出发利用切片算法(两阶段图形可达性算法)分别计算各个层次不同粒度的切片,切片算法在层次之间移动时采用逐步求精的算法。利用分层切片算法结合各种切片准则计算各个层次的切片,以便确定引起错误的根源、错误的影响范围并进行回归测试。

3.5 利用切片进行测试覆盖分析

白盒测试中一个重要的问题是测试进行到什么程度时结束。这就需要有一定的覆盖准则。常用的覆盖测试的方法有语句覆盖、分支覆盖、条件覆盖、判定—条件覆盖以及路径覆盖。一种可能的覆盖标准是语句覆盖百分之百,分支覆盖百分之八十五。

陈振强等人运用切片技术在保持语义的情况下对源程序进行精简,即在任何输入下,两者的输出始终保持一致。将一个大系统分解为若干子系统,再进一步通过部分切片技术过滤掉不感兴趣的程序部分,对源程序的测试覆盖则由各子系统的覆盖情况合成[12]。该算法的基本思想是,在对子系统进行测试覆盖分析时,陈振强等人提出一种增长模式,层次从低到高为语句覆盖、分支覆盖和路径覆盖;分别赋予不同的级别,高层的覆盖情况由低层的覆盖情况和级差合成。同时考虑一定的数据流覆盖程度,对重要的变量赋予高的权值,进行重点测试。该方法得到的覆盖程度一般比源程序的要高。

3.6 利用切片进行测试用例集约简

测试用例集约简技术就是从大量的输入数据中精心挑选出少数有代表性的测试数据,使得采用这些测试数据即能达到最佳的测试效果,高效地把隐藏的故障揭露出来。目前,关于测试用例表生成的理论研究的论文还不是很多。

如果通过切片就可以确定数据间的依赖关系,得到各输出变量所依赖的输入变量集,那么在进行组合测试时也就能够缩小组合范围,从而极大地缩减测试用例的数量。同时将一个大的程序划分成若干个小的子系统,可以更方便、更精确地设计测试用例。

通过切片分析,如果一个测试用例当且仅当能够改善覆盖程度,则可以加进测试集中,否则可以认为是多余的。如果同时引入切片技术删除多余的无用代码,还可以查找出更多的冗余测试数据。

3.7 基于切片的变量完整性测试

由于软件测试用例的输出部分很难确定,通过测试变量自身的定义域和变量间的一致性约束关系,只需要确定输出值的范围而不用知道其确切的值,就可以提高错误检测的效率。同时,检测的范围不局限于程序最后的输出结果,而是散布在程序中的各个有意义的变量,如关键变量、全局变量、局部变量和数据结构变量以及函数的返回值或传出参数等的定值点等。文献[13]引入变量依赖关系图来进行程序切片,并通过插桩来检测这些重要的变量定值点的定义域和变量间的一致性约束关系,正如调试过程中设置断点观察的那些变量,使得检测错误更加精准。

4 结束语

目前对程序切片技术的研究主要集中在静态切片方面,而在程序调试、测试领域大多采用动态切片技术。如何快速、有效、准确地计算动态切片,如何将切片技术更好地应用在软件测试方面是本文关注的重点。测试用例的选择和自动生成技术是软件测试的一个重要研究领域,无论对白盒测试还是黑盒测试都起着非常关键的作用。将程序切片技术引入今后的研究,采用切片技术将程序和需求相结合,用以辅助生成最优的测试用例集;改进 Gupta算法,使得对输入的非线性表示的谓词函数也能自动生成测试数据;减少错误定位和回归测试的工作量。

参考文献:

[1] WEISER M. Program slicing: formal, psychological and practical investigations of an automatic program abstraction method[D]. Michigan: University of Michigan, 1979.

(下转第 217页)

Handle协议定义了 Handle的构成规则。Handle通常情况下由命名授权 (Naming Authority, NA)和本地名字 (Local Name)构成。NA实际上是一个名称空间,保证了不同组织的 Handle的唯一性。一个组织有了 NA以后,就可以在这个 NA下注册 Handle。DMOI的生成采用了与 Handle相同的规则。DMOI也是由两个部分组成,即 NA和本地名字。NA可以具有层次结构,笔者在 DMOI系统中规定所有分馆的 NA都在 dmcu命名授权下,如南京大学数字博物馆的 NA是 dmcu. nju,复旦大学数字博物馆的 NA是 dmcu. fudan等;本地名字则采用原系统中的藏品 ID,这样就得到了每个藏品唯一的标志符。当然此平台还没有在 GHR注册笔者使用的 NA,DMOI系统目前只是本共享平台内相互协作的一个系统,与外界并没有关联。

笔者基于 Handle System构建了 DMOI服务解决藏品的命名、解析定位问题,同时基于 OAI-PMH实现了各个馆之间的用户权限控制和资源的版权保护机制。以及其他的诸如面向藏品的自动回答系统、藏品 3D数据获取动态展示、虚拟现实等机制。笔者实现了中国大学数字博物馆共享平台 (http://dmcu.nju.edu/cn)。

数字博物馆为很多学科的研究人员提供了丰富且可靠的科研资料,共享平台的建设更为互操作问题。

OAI协议本身已经有不少的实现。本系统在实现中采用了开源的 OAICat[10]包。由于 OAICat仅仅支持从一张数据表中获得不同的元数据格式,而不支持从不同的数据表中获得不同的元数据格式。笔者在该软件包的基础上进行扩展,实现了从多张数据表中获得不同元数据格式的功能,并将其应用到本平台中。

中国大学数字博物馆共享平台中存在着两个不同的角色,即分馆和地区中心。各个分馆发布的数据遵循核心元数据格式 (指定为 Dublin Core)和扩展元数据格式。扩展元数据格式满足该藏品所属类别 (人文艺术、生命科学、地球科学、科学技术)的领域元数据标准。分馆和地区中心都是支持了 OAI-PMH接口的数据提供者和服务提供者。其中分馆服务基于本馆数据提供,而地区中心服务基于的数据来自所有分管数据提供者提供的数据。

基于这样的共享平台框架,笔者满足了大学数字博物馆共享平台基本服务建设的要求并达到预期的目标,实现了相关资源的共享,提供了相应的服务。

5　结束语

基于笔者提出的这样一个构建数字资源共享平台框架可以较容易地构建数字资源共享平台的应用,既满足了各个系统的共享与互操作,又建立了一种地址解析系统,将服务门户和数字资源提供者无缝连接起来。当然同时还有一些问题有待完善。比如:①建立元数据提供者的目录,实现数据提供者基地址注册和更新维护的机制,即由于数据提供者的数量较多,并且基地址有变化的情况下,手工维护就不能保证元数据的正常更新,需要建立一种数据提供者基地址注册和更新维护的机制;②低成本的系统转换支持,即作为元数据的拥有者,如果为元数据的发布而单独建立数据发布系统,将增加其系统运行的成本,对数据维护和更新是不利的,需要为内容拥有者提供内容管理与发布为一体的元数据发布模块,使其很容易嵌入到内容发布系统中去,以便降低系统转换的成本。

参考文献:

[1] MARTHA L, BROGAN A. Survey of digital library aggregation services[M]. Washington, D. C.: Digital Library Federation, 2003.

[2] 马文峰. 数字资源整合研究[J]. 中国图书馆学报, 2002, **28**(4): 64-67.

[3] 肖珑. 元数据格式在数字图书馆中的应用[J]. 大学图书馆学报, 1999, **17**(4): 23-25.

[4] 张晓林. 数字对象的唯一标识符技术[J]. 现代图书情报技术, 2001(3): 8-11, 14.

[5] 林绮屏. 数字资源互操作协议 OAI与 OpenURL之比较研究[J]. 情报杂志, 2004, **23**(7): 12-13, 16.

[6] 毛军. 我国数字图书馆标准规范建设:数字资源唯一标识符的现状与发展[EB/OL]. (2004-05). http://cdls.nstl.gov.cn.

[7] 牛振东,朱先忠. 我国数字图书馆标准规范建设:OAI-PMH协议应用指南[EB/OL]. (2004-05). http://cdls.nstl.gov.cn.

[8] CNRI[EB/OL]. http://www.cnri.reston.va.us/.

[9] OAI-PMH[EB/OL]. http://www.openarchives.org/.

[10] OAICat[EB/OL]. http://www.oclc.org/research/software/oai/cat.htm.

(上接第 213页)

[2] WEISER M. Program slicing[J]. **IEEE Transactions on Software Engineering,** 1984, **210**(4): 352-357.

[3] KOREL B, LASKI J. Dynamic program slicing[J]. **Information Processing Letters,** 1988, **29**(3): 155-163.

[4] HALL R J. Automatic extraction of executable program subsets by simultaneous program slicing[J]. **Journal of Automated Software Engineering,** 1995, **2**(1): 33-53.

[5] CANFORA G, CIMITILE A, LUCIA A De. Conditioned program slicing[J]. **Information and Software Technology,** 1998, **40**(11/12): 595-607.

[6] GUPTA N, MATHUR A P, SOFFA M L. Automated test data generation using an iterative relaxation method: proc of the 6th ACM SIGSOFT International Symposium on the Foundations of Software Engineering(FSE-6 SIGSOFT '98)[C]. Orlando: [s.n.], 1998: 231-244.

[7] 赵瑞莲. 软件测试方法研究[D]. 北京:中国科学院, 2001: 83-88.

[8] 王雪莲,赵瑞莲,李立健. 一种用于测试数据生成的动态程序切片算法[J]. 计算机应用, 2005, **25**(6): 1445-1450.

[9] VIVEKANANDA M, VEDULA, JACOB A, *et al* Program slicing for hierarchical test generation: proceedings of the 20th IEEE VLSI Test Symposium (VTS '02)[C]. [S.l]: [s.n.], 2002.

[10] WONG W E, QI Yu. An execution slice and inter-block data dependency-based approach for fault localization: proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC '04)[C]. [S.l]: [s.n.], 2004.

[11] 李必信. 程序切片技术及其在面向对象软件定量和软件测试中的应用[D]. 南京:南京大学, 2000: 6-18.

[12] CHEN Zhenqiang, XU Baowen, YANG Hongji, *et al* Test coverage analysis based on program slicing: IRI '03[C]. Las Vegas: [s.n.], 2003: 559-565.

[13] 黄光燕,李晓维. 软件的变量完整性测试方法[J]. 计算机辅助设计与图形学学报, 2004, **16**(11): 1584-1589.