基于刻面树的可重构MES组件库中组件检索①

涂继跃[1]　周德俭[1,2]　刘电霆[1]

(1.桂林理工大学 信息科学与工程学院 广西 桂林 541004;

2.广西工学院 机械工程系 广西 柳州 545006)

摘　要： 提出了一种基于刻面树组件检索规约结合设计模式的组件检索算法。该算法首先在刻面分类的基础上抽取组件的本质特征创建了刻面树。然后，根据用户选择的刻面和术语制定了符合刻面树检索组件的规约,根据该规约结合迭代和组合二设计模式给出了适合从中小离散制造业可重构 MES 组件库中检索组件的算法。最后，对该算法进行了编码实现与实验测试。实验结果数据证明，同等条件下采用此算法检索时间更少，速度更快。运用该算法可解决从具有可重构性的 MES 组件库中快速检索出用户所需组件这一关键问题。

关键词： 可重构；MES；刻面树；组件库；组件检索

Component Retrieval from the Reconfigurable MES Component Libraries Based on Faceted Tree

TU Ji-Yue[1], ZHOU De-Jian[1,2], LIU Dian-Ting[1]

(1.College of Information Science and Engineering, Guilin University of Technology, Guilin 541004, China;

2.Department of Mechanical Engineering, Guangxi University of Technology, Liuzhou 545006, China)

Abstract: This paper proposes a component retrieval algorithm based on facted tree retrieval rules with design pattern. The algorithm first creates faceted tree through extracting the nature of components based on facet classification pattern. Then, according to users' chosen facets and terms, it formulates the rules to define faceted tree retrieval components. Based on the combination of the rule with iterator and that of these two design patterns, it gives a component retrieval algorithm for component libraries of reconfigurable MES. Finally, the algorithm is realized with coding by Java and tested with instances. The testing results show that under the same conditions, this algorithm uses less retrieval time. With this algorithm, needed components can be rapidly retrieved from reconfigurable MES component libraries.

Keywords: reconfigurable; MES; faceted tree; component libraries; component retrieval

1　引言

制造执行系统(MES)是介于上层企业资源计划和工业底层控制系统之间面向车间层的管理信息系统[1]。目前 MES 系统主要应用于流程工业和大型离散制造企业，在中小型离散制造企业应用中比较缺乏和不成熟，其中 MSE 可配置、可复用和可重构性能不强是原因之一。而中小型离散制造企业普遍存在制造实力弱、资金不足、管理制度不完善、基础落后、技术不强等不足，因此，提出能够实现快速可配置、可复用和可重构的 MES 系统是该类企业的迫切需求，而基于组件的复用技术是实现 MES 可重构的方法之一，从组件库中快速检索出用户所需组件是解决中小型离散制造企业 MES 重构的首要关键技术。

刻面分类检索法是 Prieto-Diaz 和 Freeman 在

① 基金项目:广西研究生教育创新项目(2008105960812M03);广西制造系统于先进制造技术重点实验室开放基金(桂科能 07109008_024_k)
收稿时间:2009-06-10

1987 年提出的，这种方式通过反映组件本质特性的视角(刻面)对组件进行精确的分类[1]，NATO 和 REBOOT 之后也提出了基于刻面的组件分类标准与模型，国内的北大青鸟、鲁大营和李颖等都提出过相应的组件检索算法[2-4]。

本研究基于创建的刻面树和已有的中小离散制造企业可重构 MES 组件库，在保证检索组件查全率和查准率高条件下，以更好的时间性能解决了该类企业从可重构 MES 组件库检索出所需组件这一关键技术。

2 刻面分类与刻面树

刻面分类是将描述组件的关键词(即术语)置于一定的环境中，根据组件的本质特征(刻面)进行精确分类。每个刻面含有一组基本的术语(关键词)，称为术语空间，它允许术语之间有同义词关系，术语只在给定的刻面中取值，术语空间可以演变[1]。

考虑检索算法效率，描述组件的刻面不宜过多或过少，本研究根据前人刻面分类的经验[5]结合该类企业的特点[6]为提高检索组件的时间性能而提出的 MES 组件本质特征即刻面有：层次、功能、表示法、开发工具、使用环境和应用领域，见下表 1

表 1　部分主要刻面及术语表

父刻面	子刻面	术语
层次	软件工程	需求分析、概要设计、详细设计、编码实现、测试、实施等
	系统层次	视图层、视图层、服务层、业务层、DB层、数据层等
功能	简合功能	注册、查询、浏览、查询、信息反馈等
	后台功能	系统管理、用户管理、资源管理、数据管理、任务管理、权限管理、安全管理等
表示法	代码表示	源代码、二进制代码等
	文档表示	需求分析文档、概要设计文档、详细设计文档、测试文档等
	页面表示	Jsp页面、html页面、velocity模板页面等
	图表表示	UML图、数据流图、流程图、甘特图、表格等
	语言表示	伪代码语言、程序设计语言、自然语言等
开发工具	开发环境工具	应用服务器、web服务器、数据库服务器、图片服务器等
	开发语言工具	Java、C#、VC++、ts、Ruby、Dephi、vb等
	数据存储工具	xml文件、txt文件、doc文件、属性文件等
使用环境	软件环境	jdk5.0、myeclipse5.1、visio2007、office等
	硬件环境	CPU、内存、主板、硬盘、显卡等
应用领域	企业类型领域	中国离散制造企业、小型离散制造企业
	系统性质领域	可复用、可重构、可集成等

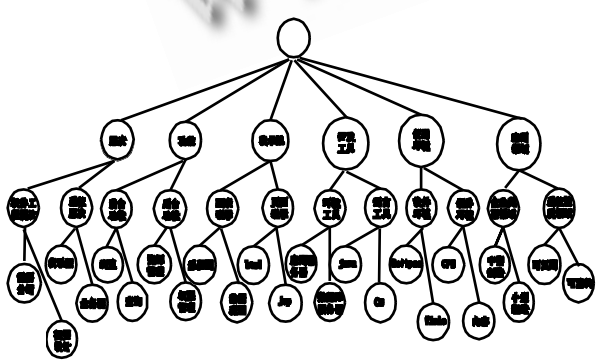

图 1　刻面树

由刻面分类概念和上面提出的刻面结合相应的映射方法[2]创建了本研究所需刻面树，见图 1(只列出了部分刻面和术语)：

3 基于刻面树的组件检索规约

规约 1：若用户查询组件时只选择其中一个叶子节点(术语)，则查询结果为该叶子节点术语所描述的组件集合。

规约 2：若用户查询组件时选择同一刻面下两个或以上的叶子节点(术语)，则查询结果为这些叶子节点术语并集所描述的组件集合。

规约 3：若用户查询组件时选择不同兄弟刻面下两个或以上的叶子节点(术语)，则查询结果为各个刻面下叶子节点术语并集再取交集所描述的组件集合。

规约 4：若用户查询组件时选择其中一个刻面(组件特征)，则查询结果为该刻面所有术语正交所描述的组件集合。

规约 5：若用户查询组件时选择两个或以上无父子关系的刻面(组件特征)，则查询结果为各个刻面下术语正交后再取交集后所描述的组件集合。

规约 6：若用户查询组件时选择的是树根(所有组件特征)，则查询结果为其下所有刻面下术语正交后再取交集后所描述的组件集合，即查询所有的组件。

为方便阐述这里的规约，在这把图 1 表示的刻面树简化为下图 2。

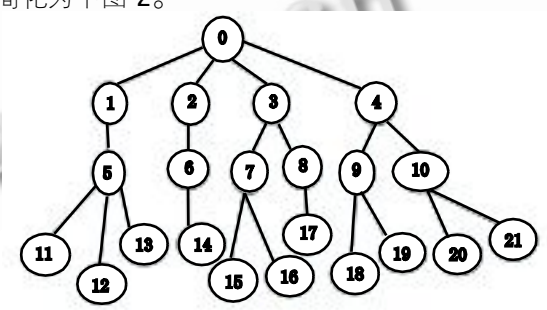

图 2　简化的刻面树

为描述方便，图 2 的所有节点都是按层次编号的，现在具体阐述上述六条规约：

设每个术语 T 所描述的组件集合为 C_T，则根据规约 1 可得到编号为 14 的术语所描述的组件集合为 C_{14}；根据规约 2 可得到编号为 11 和 13 术语所描述的组件集合为 $C_{11\wedge13}=C_{11}\cup C_{13}$；根据规约 3 可得到编号为 11、13 和 14 术语所描述的组件集合为 $C_{11\wedge13\wedge14}=(C_{11}\cup C_{13})\cap C_{14}$；根据规约 4 可得到编号

为 3 术语所描述的组件集合为 $C_3=C_7\cup C_8=(C_{15}\cup C_{16})\cap C_{17}$；根据规约 5 可得到编号为 5 和 6 术语所描述的组件集合为 $C_{5\wedge6}=C_5\cap C_6=(C_{11}\cup C_{12}\cup C_{13})\cap C_{14}$；根据规约 6 可得到树根节点编号为 0 所描述的组件集合为 $C_0=C_1\cap C_2\cap C_3\cap C_4=(C_5\cup\Phi)\cap(C_6\cup\Phi)\cap(C_7\cup C_8)\cap(C_9\cup C_{10})=(C_{11}\cup C_{12}\cup C_{13})\cap(C_{14})\cap((C_{15}\cup C_{16})\cup C_{17})\cap((C_{18}\cup C_{19})\cup(C_{20}\cup C_{21}))=(C_{11}\cup C_{12}\cup C_{13})\cap C_{14}\cap(C_{15}\cup C_{16}\cup C_{17})\cap(C_{18}\cup C_{19}\cup C_{20}\cup C_{21})$。

4　基于刻面树的组件检索规约结合设计模式的算法及其实现

4.1 算法的伪代码描述

通过迭代器模式判断一棵刻面树下是否有节点或非叶子节点下是否还有节点；一棵刻面树的子树也是一棵刻面树，因此用组合模式解决此递归问题；在此基础上再结合先根遍历这棵刻面树可使搜索节点时间降至更低。最后将查询到的节点作为 SQL 语句的查询条件，与数据库表中组件描述相匹配，返回符合查询节点条件的组件列表，以组件在表中存储的组件相对位置为链接提供给检索用户使用。

算法的伪代码描述：

(1) 前置条件：组件查询页面按刻面树的层次列出所有的组件刻面和术语供用户多选。

(2) 算法输入：查询组件用户在查询页面上勾选组件刻面和术语(即除刻面树根节点外的所有节点 $N_1,N_2,\ldots\ldots,N_n$)作为输入数据。

(3) 算法处理流程：

C_T 为最后查询出的术语集合,C_X 为用户前所选的刻面和术语,size()为集合的大小

```
while(Cx.size()>0){
  if(Ni是刻面节点){//如:C1
//用迭代器模式判断该节点下是否还有节点
  if(Ni>0)//代表有节点{
//用组合模式向 CT 加入所有节点的并集,//如:C11∪C12∪C13
   }else{ //向 CT 加入空集
   }
  }else{ //否则就是术语节点
//用组合模式递归调用类似上面判断刻面节//点情况的代码}
//同理,其它情况也用这两种设计模式来理
}
```

(4) 算法输出：C_T 集合所有的节点元素(用作 SQL 语句查询条件参数)

4.2 算法实现的主要代码

```
public interface Node{//节点接口
    void addNode(Node node);//添加节点
    Iterator<Node> iterator();//迭代输出节点
}
public abstract class AbstractNode imple- ments Node{
    protected String name;
    protected AbstractNode(String name){
        this.name=name;}
    public String toString(){
        return name;}}
public class BranchNode extends Abstract- Node{//非叶子节点
    public BranchNode(String name){
     super(name);}
    private Collection<Node> childs=new ArrayList<Node>();
    public void addNode(Node node){
    childs.add(node);//组合模式}
    public Iterator<Node> iterator(){//迭代器模式
    return childs.iterator();}}
public class LeafNode extends Abstract- Node{//叶子节点
    public LeafNode(String name){
    super(name);}
   public Iterator<Node> iterator(){//迭代器模式
    return null;}}
public class DepthFirstIterator implements Iterator<Node>{//先根遍历刻面树
    private Stack<Iterator<Node>> stack= new Stack<Iterator<Node>>();//栈保存节点
    pulic DepthFirstIterator (Iterator<No- de>it){
       stack.push(it);
    }
public boolean hasNext(){//用迭代器模式判断是否
```

还有节点

```
        if(stack.isEmpty()){
            return false;}else{
            Iterator<Node> it=stack.peek();
         if(it.hasNext()){
              return true;}else{
            stack.pop();return hasNext();}}}
    public  Node next(){//输出节点
        if(hasNext()){
            Iterator<Node> it=stack.peek();
            Node next=it.next();
            if(next instanceof BranchNode){
               stack.push(next.iterator());
             }
             return next;
    }else{return null;}}}
```

5　算法实例测试

拿图 2 的数据为例：若用户选了 1、5、11、13、14 五个节点(注意：测试时都在同一环境下)。

5.1 采用本研究给出算法的实例测试

```
    private Node createTree(){
        Node root=new BranchNode("1");Node a=new BranchNode("5");
        Node b=new LeafNode("11");Node c=new LeafNode("13");
        Node d=new LeafNode("14"); root.addNode(a);a.addNode(b);
        a.addNode(c);root.addNode(d);return root;}
    public static void main(String []args){
        TreeTest tt=new TreeTest();StringBuffer sb=new StringBuffer("");
        long start=System.currentTimeMillis();
    for(Iterator<Node>it=new DepthFirstIterator(tt.createTree().iterator());it.hasNext()){sb.append(it.next()+"");} long end=System.currentTimeMillis();
        System.out.println("所用时间"+(end-start)+"ms");}
```

测试结果：(横向为每次测试所用时间及平均时间(ms:毫秒),纵向为测试数据节点个数)

表 2　采用本研究给出算法的测试结果

次数 / 个数	1	2	3	4	5	6	7	8	9	10	平均时间
5	0	16	0	15	0	16	16	0	0	0	6.3
50	0	16	0	0	15	16	16	0	16	16	9.5
500	15	31	32	16	31	31	31	31	15	16	24.9

5.2 采用基于刻面树而不结合设计模式算法的实例测试

```
    Long start=System.currentTimeMillis();List<Node> list=new ArrayList<Node>();StringBuffer sb=new StringBuffer("");
        Node root=new BranchNode("1");Node a=new BranchNode("5");
        Node b=new LeafNode("11");Node c=new LeafNode("13");
        Node d=new LeafNode("14"); list.add(root);list.add(a);
        list.add(d);list.add(b);list.add(c);for(Node n:list){sb.append(n+"");
        }long end= System.currentTimeMillis();System.out.println("所用时间"+(end-start)+"ms");
```

测试结果：(横向为每次测试所用时间及平均时间(ms:毫秒),纵向为测试数据节点个数)

表 3　基于刻面树未结合设计模式算法的测试结果

次数 / 个数	1	2	3	4	5	6	7	8	9	10	平均时间
5	16	31	15	16	16	0	16	15	0	15	14
50	45	15	16	16	15	0	15	0	16	32	17
500	78	31	31	31	32	47	47	47	32	94	39.1

5.3 结果分析

以上两种情况符合 CT 的输出结果都为:11、13、14,根据该研究提出的规约,组拼的 SQL 语句应该为：SELECT 组件位置路径 FROM 组件表 WHERE 条件 1 IN (11，13) AND 条件 2=14;通过此 SQL 查询语句可以检索出用户所需的组件集合。查询语句执行的快慢因机器和底层数据库不同而不同，但在同一条件下，通过上面测试结果对比分析可得出：本研究给出的算法比仅基于刻面树而不结合设计模式的算法执行时间将近减少了一半。若查询语句的执行环境(如机器配置和数据库情况)相同，则由于本研究给出的算法

(下转第 201 页)

(上接第 145 页)

查询到用于动态组拼 SQL 语句的条件参数的时间更少，因此其检索速度更快。

6 结语

本研究在中小型离散制造企业可重构 MES 组件库基础上，充分利用刻面分类法有查全率和查准率高的优势，结合企业实际情况抽取企业可重构 MES 组件特征创建了用作检索组件依据的刻面树。根据用户查询时实际选择的刻面和术语制定了基于刻面树检索组件的规约，并在该规约的基础上结合迭代器和组合两个设计模式提出了先根遍历刻面树节点用于检索组件的算法，并对该算法进行了编码实现与实例测试。通过测试结果的对比分析，本研究给出的算法在检索组件库组件上所用时间更少，检索速度更快。通过该组件检索方法可解决中小型离散制造企业 MES 重构的首要关键性问题即从具有可重构性的组件库中快速检索出用户所需组件问题，为实现该类企业根据其需求快速重构所需 MES 提供了良好的基础。

参考文献

1 邹博.基于刻面分类的软件构件检索的研究[硕士学位论文].2006.12－15.

2 梁粤,任洪敏,张敬周.基于刻面分类模式的构件库检索方法.电脑知识与技术, 2008, 16 (2):1315－1317.

3 鲁大营,曹宝香,王华.基于刻面构件描述与检索算法研究.计算机系统应用, 2008,17(2):55－58.

4 李颖,李闯.基于刻面描述和术语的构件检索算法.通化师范学院学报, 2008,29(12):22－24.

5 王渊峰,张涌,任洪敏,朱三元,钱乐秋.基于刻面描述的构件检索.软件学报, 2002,13(8):1546－1551.

6 丁谋. MES 系统在典型离散制造车间的选型与应用分析.航空制造技术, 2008,(22):74－75.