基于 XML 的软件构件查询匹配算法研究*

徐如志[1+], 钱乐秋[1], 程建平[2], 王渊峰[1], 朱三元[3]

[1](复旦大学 计算机与信息技术系,上海 200433)

[2](中创软件工程股份有限公司 软件研究院,山东 济南 250014)

[3](上海计算机软件技术开发中心,上海 200233)

Research on Matching Algorithm for XML-Based Software Component Query

XU Ru-Zhi[1+], QIAN Le-Qiu[1], CHENG Jian-Ping[2], WANG Yuan-Feng[1], ZHU San-Yuan[3]

[1](Department of Computing and Information Technology, Fudan University, Shanghai 200433, China)

[2](Software Research Institute, CVIC Software Engineering Co., Ltd., Ji'nan 250014, China)

[3](Shanghai Development Center of Computer Software Technology, Shanghai 200233, China)

+ Corresponding author: Phn: 86-21-65643785, Fax: 86-21-65642826, E-mail: rzxu@fudan.edu.cn

http://www.fudan.edu.cn

Received 2002-11-08; Accepted 2003-01-20

Xu RZ, Qian LQ, Cheng JP, Wang YF, Zhu SY. Research on matching algorithm for XML-based software component query. *Journal of Software*, 2003,14(7):1195~1202.

http://www.jos.org.cn/1000-9825/14/1195.htm

Abstract: Based on the research of unordered tree-inclusion matching, a matching algorithm for XML-based component query is proposed. This algorithm can greatly improve the recall and provide support for Boolean query while maintaining a high level precision. Moreover, by adding some constraints on the basis of features of software component and using dynamic programming, the computation of matching cost is resolved in polynomial time, so that a high efficiency for the component query is guaranteed. Furthermore, the feasibility and efficiency of the new matching algorithm in practical application to software component query are confirmed by the results of a series of experiments on a prototype system RCRS.

Key words: software component; XML; component query; tree matching

摘 要: 在研究无序树包含匹配的基础上,提出一种新的基于 XML 的软件构件查询匹配算法.该算法可以在保持较高构件查准率的前提下,显著地提高构件的查全率,并提供对布尔查询的支持.此外,通过合理地设定约束条件以及利用动态规划的方法,将计算查询匹配代价的算法时间复杂度限定为多项式级,确保构件查询具有足够的查询效率.最后,通过在构件库原型系统 RCRS 上进行的一系列实验,进一步证明了新的查询匹配算法在软件构件查询实际应用中的可行性和有效性.

关键词: 软件构件;XML;构件查询;树匹配

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA1100241 (国家高技术研究发展计划(863))

第一作者简介: 徐如志(1966－),男,山东泰安人,博士生,高级工程师,主要研究领域为软件复用,构件库系统,软件项目管理.

© 中国科学院软件研究所 http://www.jos.org.cn

中图法分类号: TP311 文献标识码: A

基于复用的软件开发可以有效地提高软件开发的质量和效率[1].随着对软件复用的研究与深入,软件构件库作为软件复用的一项重要基础设施,已得到产业界与学术界越来越多的重视.其中,软件构件的查询技术已成为一个主要的研究热点,并以查准率、查全率和查询效率作为评价其效能的主要指标[2].

针对构件的查询,国内外的研究人员做了许多有益的研究工作.目前构件查询主要还是使用基于关键字匹配的传统的图书馆及信息科学编目信息查询技术[3].但是,基于关键字匹配的查询无法体现所查询的关键字之间的相互关系,使得构件的查准率不高.伴随构件库面向网络的应用,XML 已经作为一种新的构件描述的标记语言[4].但是,基于 XML 的查询语言(XQL)只能查询到与查询条件精确匹配的结果[5],因而构件的查全率很低.Richter 提出一种基于结构的 XML 文档查询方法[6].尽管该方法具有较高的查全率,且算法的时间复杂度为多项式级,但其所应用的是有序树的匹配思想.由于同一层次上的构件描述信息没有先后顺序上的限制,因此这种基于有序树匹配的查询方法不能应用于构件查询.Torsten 在文献[7]中提出了一种新的 XML 查询模型和算法,虽然应用了无序树近似匹配的思想,但算法的时间复杂度是指数级的,无法保证较高的构件查询效率.

本文在研究软件构件查询具体特点的基础上,借鉴无序树匹配的有关理论,提出一种能够兼具较高的构件查全率、查准率和查询效率,同时又能够对布尔查询提供支持的软件构件查询匹配算法.为此,本文给出一种比树包含匹配更为宽松的树包涵匹配模型,在保证较高的构件查准率前提下,显著地提升构件的查全率,并提供对布尔查询的支持.此外,通过合理地设定约束条件以及利用动态规划的方法,将计算查询匹配代价的算法时间复杂度限定在多项式级,确保构件查询具有足够的查询效率.

1 树及映射的基本概念

树是图的一种特例,用 T 表示,$T=(V,E,root(T))$,有时简记为 $T=(V,E)$.其中 V 表示一个有限的节点集;$root(T)$ 表示树的根节点;E 表示边集,是 V 上的一个二元关系,满足反自反、反对称、可传递性.如果$(u,v)\in E$,则称 u 节点是 v 节点的父节点,记为 $u=parent(v)$.如果两个节点 v_1,v_2,有$(v_1,v_2)\in E^+$(其中 E^+是 E 的传递闭包),则称 v_1 是 v_2 的祖先节点,记为 $v_1=ancestor(v_2)$或 $v_2=descendant(v_1)$.如果将树的每一个节点对应于一个标签,这样的树称为标签树,用 $label(v)$表示节点 v 上的标签.如果树中兄弟节点的左右顺序是有意义的,则这样的树称为有序树,否则称为无序树.对一棵无序标签树,允许插入节点、删除节点和改变节点标签这 3 种编辑操作[8].两棵树之间的匹配可以映射为两棵树的节点之间的对应关系.在此给出关于映射及映射代价的定义.

定义 1(映射). 设 $T_1=(V,E,root(T_1)),T_2=(W,F,root(T_2))$为两棵无序标签树,一个从 T_1 到 T_2 的映射 M 定义为: $M\subseteq V\times W$,并且对所有 $(v_1,w_1),(v_2,w_2)\in M$ 满足以下的条件:

(1) $v_1=v_2 \Leftrightarrow w_1=w_2$,表示两棵树中参与映射的节点是一一对应的.

(2) $v_1=ancestor(v_2) \Leftrightarrow w_1=ancestor(w_2)$,表示映射保持节点对之间的祖先后代关系.

映射 M 的定义域 $Domain(M)$定义为 $Domain(M)=\{v\in V \mid \exists w\in W:(v,w)\in M\}\subseteq V$.

映射 M 的值域 $Range(M)$定义为 $Range(M)=\{w\in W \mid \exists v\in V:(v,w)\in M\}\subseteq W$.

定义 2(映射代价). $T_1=(V,E,root(T_1)),T_2=(W,F,root(T_2))$为两棵无序标签树,$M$ 为一个从 T_1 到 T_2 的映射,则定义 M 的映射代价 $\gamma(M)$:

$$\gamma(M)=\sum_{(v,w)\in M}\gamma\big(label(v)\to label(w)\big)+\sum_{v\in V-Domian(M)}\gamma\big(label(v)\to\lambda\big)+\sum_{w\in W-Range(M)}\gamma\big(\lambda\to label(w)\big).$$

从映射代价的定义可以看出,一个从 T_1 到 T_2 的映射 M 的映射代价计算可以分为 3 个部分:第 1 部分表示对参与映射的节点进行标签改变操作的代价($label(v)\to label(w)$);第 2 部分表示对出现在 T_1 中的,并且未参与映射的节点进行节点删除的代价($label(v)\to\lambda$);第 3 部分表示对出现在 T_2 中的,并且未参与映射的节点进行节点插入的代价($\lambda\to label(w)$).

© 中国科学院软件研究所 http://www.jos.org.cn

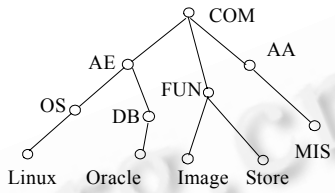
2 基于 XML 的构件描述文档及查询表示

2.1 基于XML的构件描述文档的树型表示

一个构件的 XML 描述文档可以映射为一棵无序标签树.目前构件大多采用刻面的分类描述模式[9,10],如图 1(a)所示为一个基于 XML 的构件刻面描述文档,如图 1(b)所示为图 1(a)的树型表示.这里图 1(b)中的部分节点标签以图 1(a)中对应描述的字母缩写来表示,如 AE 代表 Application Environment,FUN 代表 Function.

```
⟨Component⟩
  ⟨Application Enviroment⟩
    ⟨DataBase⟩ oracle ⟨/DataBase⟩
    ⟨Operation System⟩ linux ⟨/Operation System⟩
  ⟨/Application Enviroment⟩
  ⟨Function⟩ image store ⟨/Function⟩
  ⟨Application Area⟩ MIS ⟨/Application Area⟩
⟨/Component⟩
```

(a) XML document
(a) XML 文档

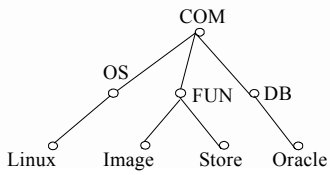


(b) Component tree
(b) 构件树

Fig.1 XML document and tree interpretation of a component
图 1 构件的 XML 描述文档及其树型表示

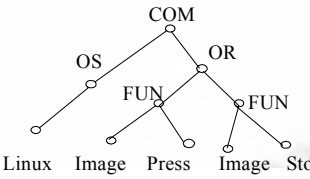
2.2 基于XML的构件查询表示

一个基于 XML 的连接查询,是指在 XML 查询构造中仅使用逻辑连接符号“and”而没有使用“or”的查询.它可以映射为一棵无序标签树.如图 2(a)所示,每个名称项,如“COM”,“OS”被分别映射为查询树的一个节点,每个文本项,如“linux”(text( )=“linux”的简写)被映射为一个叶子节点,包含运算关系“[ ]”映射为树中节点之间的父子关系,逻辑连接符“and”映射为树中节点的兄弟关系.



COM[OS[“linux”]andDB[“oracle”]andFUN[“imagie store”]]

(a) Mapping of a conjunctive query to a tree
(a) 一个连接查询到一棵树的映射



COM[OS[“linux”]]andFUN[[“image press”]or[“image store”]]]

(b) Mapping of a Boolean query to a tree
(b) 一个布尔查询到一棵树的映射

Fig.2 Mapping of a component query to a tree
图 2 一个构件查询到一棵树的映射

一个基于 XML 的布尔查询,是指在 XML 查询构造中既使用逻辑连接符号“and”,又使用逻辑连接符号“or”的查询.它可以等价地分解为由多个连接查询组成的集合.如果一个布尔查询中有 k 个“or”逻辑连接符,则可等价地分解为 2^k 个连接查询的集合.相应地,一个布尔查询可以映射为 2^k 棵连接查询树.如果仅用对连接查询树的匹配算法,则理论上需要进行 2^k 次查询匹配计算,然后再对所有查询计算结果进行对比和优选.显然,这种方法的查询计算复杂且容易出错.

为了解决这类问题,本文把逻辑连接符“or”本身映射为查询树中标签为“OR”的特殊节点.以如图 2(b)所示的“COM[OS[“linux”]andFUN[[“image press”]or[“image store”]]]”查询为例,首先将“or”两侧的“image press”和“image store”分别与其前面共同的名称项“FUN”映射为一棵子树,“or”映射为这两棵子树的父节点“OR”,并使其成为前面更高一级的名称项“COM”的子节点.采取这样的处理方式,树中增加了一类特殊类型的+节点“OR”,但将一个布尔查询映射为一棵查询树表示,从而将构件的布尔查询转化为一棵构件查询树到一棵构件描述树的匹配计算,不再需要由多棵连接查询树分别与同一棵构件描述树进行匹配计算了.

3 构件查询的匹配模型及匹配代价

根据上面的介绍,构件查询转化为查询树与构件库中构件描述树之间的匹配问题.查询结果为与构件查询

© 中国科学院软件研究所 http://www.jos.org.cn

树具有最小匹配代价的构件描述树.显然,两棵树之间的匹配代价越小,两棵树的匹配越好.

3.1 构件查询的匹配模型

为了对比分析和阐述本文给出的构件查询的匹配模型,先引入树的包含及树的嵌入的定义,在此基础上给出树的包涵的定义,然后结合构件查询的特点,分析其作为基于 XML 的构件查询匹配模型的意义.

定义 3(树的包含/树的嵌入)[11]. 设 $Q=(V,E,root(Q)),D=(W,F,root(D))$为两棵无序标签树,如果存在一个 $V\to W$ 的映射 f,使得满足以下的条件:

(1) $u=v \Leftrightarrow f(u)=f(v)$, $u,v\in Domain(f)=V$,

(2) $label(v)=label(f(v))$,

(3) $u=parent(v) \Leftrightarrow f(u)=ancestor(f(v))$,

则称存在一个 Q 到 D 的包含,Q 为被包含树,D 为包含树.

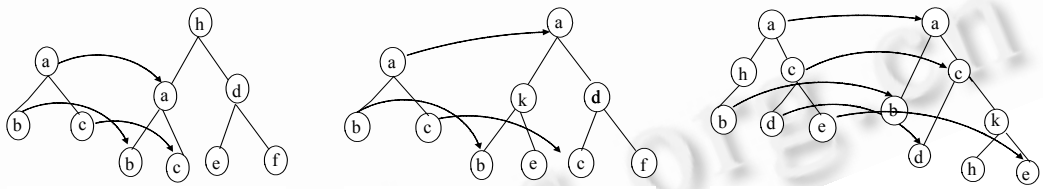
如果上述条件(3)改为 $u=parent(v) \Leftrightarrow f(u)=parent(f(v))$,则称存在一个 Q 到 D 的嵌入.

树的嵌入匹配,如图 3(a)所示,要求 f 的定义域等于查询树的节点集合(见定义 3 中的条件(1)),表明它不容许出现多余的构件查询信息,即不容许用户提出的查询节点信息没有在构件描述树中描述的情况,而且树嵌入要求保持查询节点对的父子关系($u=parent(v) \Leftrightarrow f(u)=parent(f(v))$).因而基于树的嵌入匹配的构件查询结果仅为与查询条件精确匹配的构件,这使得构件查询的查全率很低.

树的包含匹配,如图 3(b)所示,放宽了节点信息之间的对应关系(见定义 3 中的条件(3)),只要求保持节点的祖先后代关系,从而有效地提高了构件查询的适用性,也使构件的查全率有一定程度的提高,但由于其仍然要求 f 的定义域等于查询树的节点集合(见定义 3 中的条件(1)),仍然不允许查询信息出现多余的情况,而且传统的无序树包含问题(tree inclusion)的算法已经被证明是 NP-Hard[12],使得基于树的包含匹配的构件查询的查全率和查询效率仍然较低.

此外,树的嵌入和树的包含匹配都不提供对布尔查询的支持.因为布尔查询树中含有在构件的描述树中找不到与其对应匹配的节点“OR”,显然这与定义 3 中的条件(1)相背离.

下面在树包含的基础上,放宽约束条件,给出适用于构件查询的树包涵匹配的定义.



(a) Tree embedding (a) 树嵌入 (b) Tree inclusion (b) 树包含 (c) Tree containment (c) 树包涵

Fig.3 Three kinds of tree matching instances

图 3 3 种树匹配示意图

定义 4(树的包涵). 设 $Q=(V,E,root(Q)),D=(W,F,root(D))$为两棵无序标签树,$V'\subseteq V$,$W'\subseteq W$,如果存在一个 $V'\to W'$ 的映射 f,使得对所有 $u,v\in V'$ 满足以下的条件:

(1) $u=v \Rightarrow f(u)=f(v)$, $u,v\in Domain(f)$,

(2) $label(u)\approx label(f(u))$,

(3) $u=ancestor(v) \Rightarrow f(u)=ancestor(f(v))$,

则称 f 是一个 Q 到 D 的树包涵匹配,简称树的包涵或包涵匹配.

与树嵌入、树包含的定义相比,树包涵的定义放宽了有关约束条件.下面结合构件查询的特点来分析树包涵匹配作为构件查询的匹配模型的适用性,如图 3(c)所示.

第 1,树包涵匹配允许 f 的定义域为查询树节点集合的子集(定义 4 中的条件(1)),突破了树嵌入和树包含对 f 的定义域的限制,即允许出现多余的查询信息,这从根本上为提升构件的查全率创造了条件.

第 2,树包涵只要求保持节点的祖先后代关系(定义 4 中的条件(3)),所以它容许参与匹配的节点信息出现错

© 中国科学院软件研究所 http://www.jos.org.cn

层的情况,以进一步提高构件的查全率.但由于其同时要求匹配的节点间仍然保持一定的辈份关系,即保持一定语义结构层次上的匹配,使得其有别于单纯的关键字查找,确保构件查准率维持在较高的水平.

上述两点改进,为查询构造提供了极大的包容性.用户可以不必详细了解构件库的分类描述方案,对多个分类描述方案不同的构件库进行查询.同时,上述改进也为支持布尔查询创造了条件.

第 3,上述关系的保持只要求是单向"$\Rightarrow$"(定义 4 中的条件(1)、条件(3)),可以在不破坏构件查询含义的前提下为降低算法的时间复杂度奠定基础.

第 4,树包涵匹配只要求匹配的节点对的标签具有近似关系即可(定义 4 的条件(2)),这一改进有利于增强构件查询的模糊匹配能力,从而进一步提高构件查询的查全率.

第 4.3 节的算法分析以及第 5 节的实验数据表明,正是树包涵在树嵌入、树包含定义基础上的上述改进,使得基于树包涵匹配模式的查询既能提供对布尔查询的支持,又兼具较高的构件查准率和查全率,而且将计算树包涵匹配代价算法的时间复杂度限定在多项式级,从而保证了足够的构件查询效率.

3.2 构件查询的匹配代价

在给出构件查询树 Q 到构件描述树 D 的包涵匹配代价的定义之前,先给出映射谱的定义.映射谱的概念主要用来描述参与映射的节点以及它们之间的结构层次信息,为下面定义树的包涵匹配代价奠定了基础.

定义 5(映射谱). 设 $Q=(V,E,root(Q)),D=(W,F,root(D))$为两棵无序标签树,$f$ 为一个从 Q 到 D 的映射,则 f 的映射谱 $spectrum(f)$定义为

$$spectrum(f)=Range(f)\cup\left\{w\left|\begin{array}{l}w\notin Range(f),\exists w_1,w_2\in Range(f)\\ \text{such that } w_1=ancestor(w)\wedge w_2=descendant(w)\end{array}\right.\right\}.$$

定义 6(树的包涵匹配代价). 设 $Q=(V,E,root(Q)),D=(W,F,root(D))$为两棵无序标签树,则树 Q 到树 D 的树包涵匹配代价 $TCostM(Q,D)$为 $TCostM(Q,D)=\min\{\gamma(f)\mid f:Q\Rightarrow D\}$,其中

$$\gamma(f)=\sum_{v\in domain(f)}\gamma\big(label(v)\to label(f(v))\big)+\sum_{v\in V-domain(f)}\gamma\big(label(v)\to\lambda\big)+\sum_{w\in spectrum(f)-Range(f)}\gamma\big(\lambda\to label(w)\big).$$

如果 f 是树 Q 到树 D 的树包涵匹配,并且有 $\gamma(f)=TCostM(Q,D)$,则称 f 是一个树 Q 到树 D 的具有最小匹配代价的树包涵匹配.该定义也可以推广应用于后面用到的树与森林、森林与森林之间的包涵匹配.

与定义 2 的映射代价相比,树的包涵匹配代价中第 3 项 w 的取值范围从 w–$Range(f)$改为 $Spectrum(f)$–$Range(f)$,这种修改将计入插入代价的节点仅限于与本次匹配有关的节点(映射谱中的),而避免将构件描述树中没有被映射到的节点的插入代价计算在内,使构件匹配代价的计算更为精确合理.

4 树的包涵匹配代价算法及分析

4.1 构件查询树到构件描述树的最小匹配代价*TCostM*

构件的连接查询和布尔查询都可以被映射为一棵构件查询树.与连接查询树相比,构件的布尔查询树中含有一类特殊类型的节点(OR),连接查询树可以看作是没有特殊节点 OR 的布尔查询树.下面用枚举法分析从一棵布尔查询树到一棵构件描述树的最小匹配代价 $TCostM$ 的计算.

假设 $T_1=(V_1,E_1,root(T_1)),T_2=(V_2,E_2,root(T_2))$为两棵无序标签树,$root(T_1)=i,root(T_2)=j$.$T[i]$表示树 T 中以 i 为根节点的子树,$F[i]$表示树 $T[i]$中删除节点 i 后得到的森林,$C(i)$表示节点 i 的儿子节点集合,$D(i)$表示节点 i 的子孙节点集合,$type(i)$表示节点 i 的标签类型(OR 或非 OR 两种类型).

需要说明的是,布尔查询树中标签为 OR 的节点只是逻辑上的树节点,虽然在形式上与其他节点相同,但具有与其他节点不同的特性,由于它本身仅代表查询的连接符号,因此对其删除的代价为 0.

计算从树 T_1 到树 T_2 的包涵匹配代价 $TCostM(T_1[i],T_2[j])$,首先要判断查询树的根节点 i 的标签类型.显然,当 $type(i)$=OR 时,构件树中无对应的节点,这时必然有 $f(i)=\lambda$.当 $type(i)\neq$OR 时,存在 $f(i)\neq\lambda$ 和 $f(i)=\lambda$ 两种情况,而当 $f(i)\neq\lambda$ 时,又存在 $f(i)=j$ 和 $f(i)\neq j$ 两种情况.

© 中国科学院软件研究所 http://www.jos.org.cn

综合上面的分析,计算布尔查询的 $TCostM(T_1[i],T_2[j])$共分为 4 种情况:(1) $type(i)\neq OR, f(i)\neq\lambda, f(i)\neq j$;(2) $type(i)\neq OR, f(i)\neq\lambda, f(i)=j$;(3) $type(i)\neq OR, f(i)=\lambda$;(4) $type(i)=OR, f(i)=\lambda$.

下面对上述 4 种情况依次进行分析.设 f 所对应的编辑序列为 S.

(1) $type(i)\neq OR, f(i)\neq\lambda, f(i)\neq j$.

在这种情况下,参考图 4(a),编辑序列 S 可以看作是一个将 $T_1[i]$转换到 $F_2[j]$的具有最小匹配代价的包涵匹配所对应的编辑序列.这时有 $TCostM(T_1[i],T_2[j])=TCostM(T_1[i],F_2[j])$.

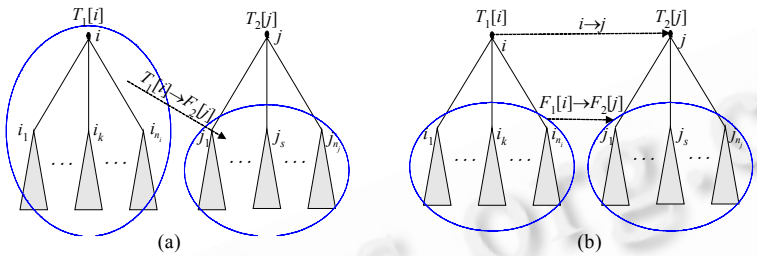


Fig. 4 Matching instances of (1) and (2)

图 4 (1)和(2)的匹配示例

(2) $type(i)\neq OR, f(i)\neq\lambda, f(i)=j$.

在这种情况下,如图 4(b)所示,编辑序列 S 可以分为 S_1 和 S_2 两部分:$S_1=\{(i\to\lambda)\}$,$S_2=S-S_1$.其中编辑序列 S_2 可以看成是一个将 $F_1[i]$转换到 $F_2[j]$的具有最小广义匹配代价*的包涵匹配所对应的编辑序列.

于是可以推出 $TCostM(T_1[i],T_2[j])=\lambda(i\to j)+\hat{T}CostM(F_1[i],F_2[j])$.

(3) $type(i)\neq OR, f(i)=\lambda$.

在这种情况下,参考图 5(a),编辑序列 S 可以分为 S_1 和 S_2 两部分:$S_1=\{(i\to\lambda)\}$,$S_2=S-S_1$.那么编辑序列 S_2 可以看作是一个将 $F_1[i]$转换到 $T_2[j]$的具有最小匹配代价的包涵匹配所对应的编辑序列.

于是可以推出 $TCostM(T_1[i],T_2[j])=\gamma(i\to\lambda)+TCostM(F_1[i],T_2[j])$.

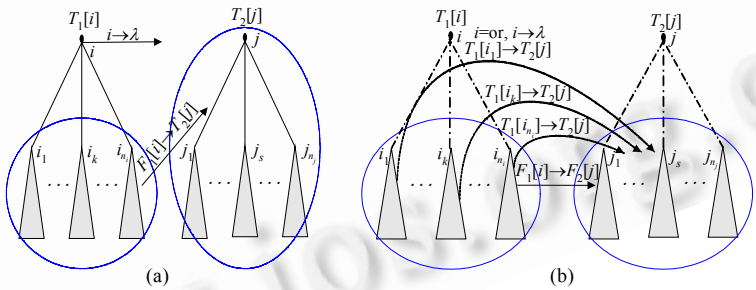


Fig.5 Matching instances of (3) and (4)

图 5 (3)和(4)的匹配示例

(4) $type(i)=OR, f(i)=\lambda$.

类似于(3),在这种情况下,编辑序列 S 仍可以分为 S_1 和 S_2 两部分:$S_1=\{(OR\to\lambda)\}$,$S_2=S-S_1$.参考图 5(b),编辑序列 S_2 可以看作是节点 OR 下其中的某一棵子树转换到 $T_2[j]$的具有最小匹配代价的包涵匹配所对应的编辑序列.由于各个子树到 $T_2[j]$的最小包涵匹配代价不尽相同,按照最小包涵匹配代价越小,则匹配越好越相近的原则,显然代价最小的 OR 节点下的那棵子树到构件描述树的包涵匹配最好.由于 OR 节点的删除代价为 0,于是有 $TCostM(T_1[i],T_2[j])=\min(TCostM(T_1[i_k],T_2[j]))$,其中 $i_k\in C(i)$.

综合上述分析可以得出:

---

* $\hat{T}CostM$ 的定义及计算方法与 $TCostM$ 类似,前者只是多计入了 $label(V)\to\lambda$的代价计算.由于篇幅关系,此处不再赘述.感兴趣者,请与作者联系.

© 中国科学院软件研究所 http://www.jos.org.cn

$$TCostM(T_1[i],T_2[j])=\min\begin{cases}TCostM(F_1[i],T_2[j])+\gamma(i\to\lambda)\\ \hat{T}CostM(F_1[i],F_2[j])+\gamma(i\to j)\\ TCostM(T_1[i],F_2[j])\\ \min(TCostM(T_1[i_k],T_2[j]))\end{cases}. \quad (1)$$

4.2 计算*TCostM*的相关定理

下面给出动态规划方式计算 $TCostM(T_1,T_2)$的相关定理.其中(1)~(5)为树(或森林)与空树之间的包涵匹配代价计算,(6)~(8)分别为森林与树、森林与森林、树与森林之间包涵匹配代价的计算,分别对应于式(1)计算 $TCostM(T_1[i],T_2[j])$中的 3 种情况.对这些定理的证明可参见文献[9].

定理 1. $T_1=(V_1,E_1,root(T_1)),T_2=(V_2,E_2,root(T_2))$为两棵无序标签树,$\theta$ 表示空树,$root(T_1)=i$, $root(T_2)=j$,则

(1) $TCostM(\theta,\theta)=0$,

(2) $TCostM(\theta,T_2[j])=0$,

(3) $TCostM(\theta,F_2[j])=0$,

(4) $TCostM(F_1[i],\theta)=\sum_{i_k\in C(i)}TCostM(T_1[i_k],\theta)$,

(5) $TCostM(T_1[i],\theta)=TCostM(F_1[i],\theta)+\gamma(i\to\lambda)$,

(6) $TCostM(F_1[i],T_2[j])=\sum_{i_k\in C(i)}TCostM(T_1[i_k],T_2[j])$,

(7) $TCostM(F_1[i],F_2[j])=\sum_{i_k\in C(i)}TCostM(T_1(i_k),F_2[j])$,

(8) $TCostM(T_1[i],F_2[j])=\min\begin{cases}\min\limits_{j_s\in C(j)}(TCostM(T_1[i],T_2[j_s]))\\ TCostM(F_1[i],F_2[j])+\gamma(i\to\lambda)\end{cases}$.

4.3 算法的时间复杂度分析

假定 $TCostM(Q,D)$为计算构件查询树 Q 到构件描述树 D 的包涵匹配代价,$Q_F[i]$表示从树 $Q[i]$中除去根节点 i 以后所形成的森林,$D_F[j]$表示从树 $D[j]$中除去根节点 j 以后形成的森林.

由第 4.1 节可知,计算 $TCostM(Q,D)$的时间复杂度由计算 $TCostM(Q_F[i],D[j])$,$TCostM(Q[i],D[j])$,$TCostM(Q_F[i],D_F[j])$,$TCostM(Q[i],D_F[j])$的时间复杂度决定.根据第 4.1 节和第 4.2 节可知,计算 $TCostM(Q[i],D[j])$的时间复杂度为 $O(4)$,计算 $TCostM(Q_F[i],D[j])$和计算 $TCostM(Q_F[i],D_F[j])$的时间复杂度为 $O(degree(Q))$,计算 $TCostM(Q[i],D_F[j])$的时间复杂度为 $O(degree(D))$,由此可知,计算 $TCostM(Q[i],D[j])$算法总的时间复杂度为

$$(\sum_{i=1}^{|Q|}\sum_{j=1}^{|D|}O(4+2\mathrm{deg}ree(Q)+\mathrm{deg}ree(D))=\sum_{i=1}^{|Q|}\sum_{j=1}^{|D|}O(\mathrm{deg}ree(D)+\mathrm{deg}ree(Q))=O(|Q|\times|D|\times(\mathrm{deg}ree(D)+\mathrm{deg}ree(Q))).$$

5 实验结果及分析

为了论证本文所提出的构件查询的匹配模式及其算法在实践应用中的可行性和有效性,我们在所设计的一个构件库查询原型系统 RCRS(VC++6.0, SQL Sever2000)上进行了查询实验.实验分 A,B,C 三个组进行,每组 10 人.所有查询人员对 RCRS 及构件库都有一些基本的常识,但不详细了解构件库中构件的具体分类描述方案.每个查询人员从构件库中查询 30 个构件,构件库中的构件都是从 MFC,STL 和 VCL 中选取出来的,共 267 个.

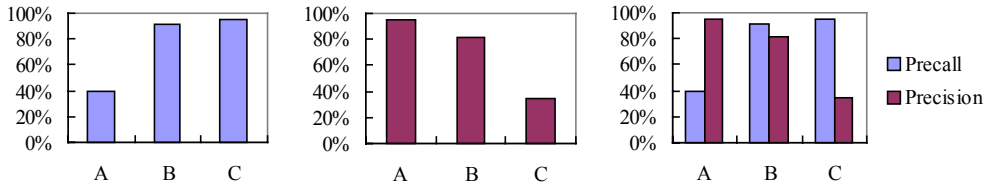
A 组使用 XQL 查询,查询结果为与查询条件完全匹配的构件;B 组使用基于树包涵匹配的查询,查询结果既包括那些与查询条件完全匹配的构件,也包括与查询条件部分匹配、近似匹配的构件;C 组使用关键字的查询,查询结果为与查询的关键字匹配的构件.显然,这种查询完全忽略了查询节点信息之间的结构关系.

图 6 为 A,B,C 三个组在相同查询条件下的构件查全率、查准率的实验统计结果.此外,在实验记录中,只有 B 组可以使用布尔查询,而且 B 组的所有查询响应时间均低于 380ms(最大查询节点数为 16 个).

实验总结如下:

- A 组具有最高的查准率,但查全率却最低.C 组具有较高的查全率,但查准率相对较低.这表明基于精确匹配的 XQL 查询和基于关键字匹配的查询都不能很好地适用于软件构件的查询.
- B 组可以使用连接查询,也可以使用布尔查询,具有与 A 组相近的查准率和与 B 组相近的查全率.平均查准率达到 81%,平均查全率为 92%.这表明基于树包涵匹配的查询方法能够很好地适用于构件的查询.
- B 组查询的响应时间为毫秒级,表明本文设计的查询算法能够满足构件查询效率的要求.

© 中国科学院软件研究所 http://www.jos.org.cn



(a) Recall (b) Precision (c) Matching between recall and precision

(a) 查全率 (b) 查准率 (c) 查全率和查准率的匹配

Fig.6 Experimental results of recall and precision based on three kinds of matching models

图 6 基于 3 种不同匹配模型的查全率和查准率的实验结果

6 结 语

本文提出一种全新的基于树包涵匹配模型的构件查询算法.该方法提供对布尔查询的支持,并具备很强的松弛匹配能力,可以在保证较高构件查准率的前提下兼具较高的构件查全率.另外,通过利用动态规划和合理设定约束条件,算法的实现克服了求解树匹配代价 NP 难的计算复杂性问题,使得构件查询的时间复杂度为多项式级,从而保证构件查询具有足够的查询效率.

通过在原型系统上进行的构件查询实验,表明该构件查询匹配算法具有较高的构件查全率和查准率以及足够的构件查询效率,并提供对布尔查询的支持.这证明了本文所设计的匹配模型及算法在实践应用中是可行的、有效的.

本文的研究成果可广泛适用于跨构件库和面向网络构件库的基于 XML 描述的软件构件查询问题.

References:

[1] Ivar J. Software reuse: Architecture, process and organization for business success. Reading: Addison-Wesley Publishing Company, 1997. 4~15.

[2] Mili H, Mili A. Reuse based software engineering. New York: John Wiley & Sons Inc., 2002. 444~459.

[3] Frakes WB, Pole TP. An empirical study of representation methods for reusable software components. IEEE Transactions on Software Engineering, 1994,l20(8):617~630.

[4] Gibb F, McCartan C, O'Donnell R, Sweeney N, Leon R. The integration of information retrieval techniques within a software reuse environment. Journal of Information Science, 2000,26(4):520~539.

[5] Torshen S. ApproXQL: Design and implementation of an approximate pattern matching language for XML. Technical Report, B 01-02, Freie University at Berlin, 2001.

[6] Thorsten R. A new measure of the distance between ordered trees and its applications. Research Report, 85166, Department of Computer Science, University of Bonn, 1997.

[7] Torshen S, Naumann F. Approximate tree embedding for querying XML data. In: Proceedings of ACM SIGIR Workshop on XML and Information Retrieval. Athens, 2000.

[8] Zhang KZ. On the editing distance between unordered labeled trees. Information Processing Letters, 1992,42(3):133~139.

[9] Wang YF. Research on retrieving reusable components classified in faceted scheme [Ph.D. Thesis]. Shanghai: Fudan University, 2002 (in Chinese with English abstract).

[10] Chang JC, Li KQ, Guo LF, Mei H, Yang FQ. Representing and retrieving reusable software components in JB (Jadebird) system. Electronic Journal, 2000,28(8):20~24 (in Chinese with English abstract).

[11] Kilpelainen P. Tree matching problems with applications to structured text databases. Technical Report, A-1992-6, Department of Computer Science, University of Helsinki, 1992.

[12] Kilpelainen P. Ordered and unordered tree inclusion. SIAM Journal on Computing, 1995,24(2):340~356.

附中文参考文献:

[9] 王渊峰.基于刻面描述的构件检索算法研究[博士学位论文].上海:复旦大学,2002.

[10] 常继传,李克勤,郭立峰,梅宏,杨芙清.青鸟系统中可复用软件构件的表示与查询.电子学报,2000,28(8):20~24.

© 中国科学院软件研究所 http://www.jos.org.cn