第 34 卷 第 3 期 计 算 机 工 程 2008 年 2 月
Vol.34 No.3 Computer Engineering February 2008
·博士论文· 文章编号：1000—3428(2008)03—0032—03 文献标识码：A 中图分类号：TP311

元素路径模型：高效的 XML Schema 提取方法

张海威，袁晓洁，杨 娜，王 鑫
(南开大学计算机科学与技术系，天津 300071)

摘 要：提出基于元素路径模型(EPM)的 XML Schema 提取方法，旨在提高 Hegewald 等提出的 XStruct 系统的运行效率。基于 EPM 的方法使用 SAX 解析 XML 文档，提取 XML 元素路径模型并根据规则进行合并，得到 XML 元素序列表达式进而生成 XML Schema。实验结果表明，基于元素路径模型方法的时间空间代价均优于 XStruct 系统。

关键词：元素路径模型；元素序列表达式；合并操作

Element Path Model: Efficient Method for XML Schema Extraction

ZHANG Hai-wei, YUAN Xiao-jie, YANG Na, WANG Xin
(Department of Computer Science and Technology, Nankai University, Tianjin 300071)

【Abstract】 This paper presents a method based on Element Path Models(EPM) for extracting XML Schema to enhance the efficiency of XStruct supposed by Hegewald, etc. XML Schema is generated by element sequence expression which is merged from element path models while parsing XML documents with SAX. Experiment shows that the method based on EPM uses less time and space than XStruct.

【Key words】 Element Path Model(EPM); element sequence expression; merging operation

XML(eXtensible Markup Language)是重要的数据交换与数据表示的标准。XML 模式语言能够使用户根据实际需要指定 XML 文档结构和内容并验证文档数据的有效性，从而提高 XML 文档中数据的质量。DTD 和 XML Schema 都是广泛使用的 XML 模式语言。DTD 定义了 XML 文档中的元素及其属性，以及元素的层次结构、出现顺序等 XML 文档内容和结构的基本信息，但却存在着如表达能力有限、约束定义能力不足、无法支持数据类型、不够结构化等缺点。XML Schema 是专门针对 DTD 的缺点而设计的，完全使用 XML 作为描述手段，具有很强的描述能力、扩展能力和处理维护能力。

1 相关工作

如何对无模式的XML文档进行合理的组织、高效的分析和充分的利用是XML技术研究者关注的重要问题。XML模式提取技术正是为了解决这个问题而成为XML技术领域的研究热点。许多研究者对自动提取XML模式的工具进行研究并取得了一定的成果。XTRACT[1], DDbE[2], DTD-Miner[3]都是自动提取XML文档DTD的工具；XStruct[4]出现在XML Schema成为标准之后，用来自动提取XML Schema。

文献[4]介绍了 XStruct 系统，XStruct 使用元素内容模型(Element Content Model, ECM)来表示 XML 文档中的元素并用正则表达式的形式表示元素内容模型。XStruct 系统首先递归地提取 XML 文档全部元素的内容模型，然后利用文献[1]中提出的正则表达式合并规则，从最底层开始逐层向上合并元素的 ECM 表达式，最终得到根元素的 ECM 表达式并转换为 XML Schema。XStruct 基本实现了 W3C 的 XML Schema 标准。

XStruct 系统的缺点是运行的时间和空间代价很大。本文提出基于元素路径模型的 XML Schema 提取方法，目的是减少提取 XML Schema 过程中的时间和空间代价。

2 XML Schema 提取方法

提取 XML Schema 的首要问题是从 XML 文档中提取与 XML Schema 相关的元素序列，再将元素序列转换为 XML Schema。

2.1 元素路径模型

定义 1 XML 文档树

一篇XML文档D^E (E表示XML文档元素的集合)可以表示为一棵有序标签树 $T=(V,root,ED,\Sigma,type,lab,val,\leqslant)$，其中：

V 是 XML 节点的集合；

$root\in V$ 是树的根节点；

二元关系 $ED\in V^2$ 是边的集合，如果 u 是 v 的父节点或 v 是 u 的子节点，则 $(u,v)\in ED$；

有穷字母表 Σ 是由文档D^E的元素和属性标签组成的集合；

函数 $type:V\to\{elem,attr,text\}$ 确定节点类型，$type(v)=elem$ (若 v 为元素)，$type(v)=attr$ (若 v 为属性)，$type(v)=text$ (若 v 为文本)；

$V_e=\{v\mid v\in V\wedge type(v)=elem\}$ 表示元素节点集合，$V_a=\{v\mid v\in V\wedge type(v)=attr\}$ 表示属性节点集合，$V_t=\{v\mid v\in V\wedge type(v)=text\}$ 表示文本节点集合；

函数 $lab:V_e\cup V_a\to\Sigma$ 返回元素或属性节点的标签，记作 $lab(v)=l$，$v\in V_e\vee V_a$，且 $l\in\Sigma$；

$\Sigma_e=\{l\mid l\in\Sigma\wedge lab(v)=l\wedge v\in V_e\}$ 表示元素节点标签的集合，$\Sigma_a=\{l\mid l\in\Sigma\wedge lab(v)=l\wedge v\in V_a\}$ 表示属性节点标签的

基金项目：天津市科技发展计划基金资助项目(06YFGZGX05700)
作者简介：张海威(1980－)，男，博士研究生，主研方向：XML 数据管理与数据挖掘技术；袁晓洁，教授、博士、博士生导师；杨 娜、王 鑫，博士研究生
收稿日期：2007-02-20 **E-mail：**zhhaiwei@mail.nankai.edu.cn

集合；

函数 $val:V_a \cup V_t \to Str$ 返回属性或文本节点的值， str 是XML 文档中所有合法字符串的集合；

二元关系 $\leq \subset V^2$ 定义XML文档顺序，在文档D^E中如果节点 u 出现在 v 之前或 $u=v$ ，则 $(u,v)\in \leq$ 或记作 $u \leq v$ ；

T 称为 XML 文档树。

根据定义 1 将文档树简化定义为

$$T=(V_e,\Sigma_e,lab_e,root)$$

其中，函数 $lab_e:V_e \to \Sigma_e$ ，返回元素节点的标签。后文使用的XML 文档树节点全部是元素节点。

文档树T中元素节点的标签与XML文档D^E的元素可以建立一一对应的关系，即T中 $\forall v \in V_e$ ，在D^E中有且仅有一个元素 $e \in E$ ，满足 $Name(e)=lab_e(v)$ ，其中，函数 $Name(e)$ 表示元素 e 的名称。

设XML文档D^E，$E=\{e_0,e_1,\cdots,e_m\}$，表示D^E的XML文档树 $T=(V_e,\Sigma_e,lab_e,root)$ ，其中， $V_e=\{v_0,v_1,\cdots,v_n\}$ ，$\Sigma_e=\{l_0,l_1,\cdots,l_m\}$， $root=v_0$ ，且 $Name(e_j)=l_j(0\leq j\leq m)$ ，节点 $v_i \in V_e(0\leq i\leq n)$ 。有如下定义：

定义 2 元素路径模型(Element Path Model, EPM)

v_i 的元素路径模型形式化定义为

$$EPM_{v_i} := e_0^{<\min,\max>} e_{i_1}^{<\min,\max>} \cdots e_i^{<\min,\max>}$$

其中，$e_{i_k} \in E,(k=1,2,\cdots,n)$ 是 v_i 除根节点 v_0 外的祖先节点在 E 中对应的元素； e_0 和 e_i 分别为 v_0 和 v_i 在文档 E 中对应的元素；min 和 max 分别表示 E 中元素 e_i 随着其父元素 e_{pi} 出现的最少次数和最多次数。

定义 3 路径表达式与元素序列表达式

用逻辑与关系运算符“ . ”连接 v_i 的元素路径模型中全部元素，得到 v_i 的路径表达式：

$$P_{v_i} := e_0^{<\min,\max>} \cdot e_{i_1}^{<\min,\max>} \cdots\cdots e_i^{<\min,\max>}$$

合并路径表达式 $P_0,P_1,\cdots,P_n$ 得到的元素逻辑表达式称为元素序列表达式。

例 1 设图 1 中 XML 文档树 $T=(V_e,\Sigma_e,lab_e,root)$ ，其中：

$V_e=\{v_0,v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8\}$ ；

$\Sigma_e=\{"C","P","B","S","D","L"\}$；

$lab_e=$ { $v_0 \to "C"$,(v_1 , v_2) $\to "P"$,(v_3 , v_6) $\to "B"$, (v_5 , v_8) $\to "D"$, $v_4 \to "S"$, $v_7 \to "L"$}；

$root=v_0$ 。

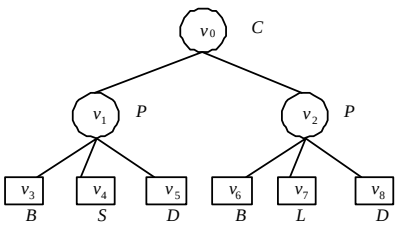

图 1　XML 文档树示例

节点 v_4 的元素路径模型：

$$EPM_{v_4}=C^{<1,1>}P^{<1,1>}S^{<0,1>}$$

当元素 S 的父元素 P 第 1 次出现时(节点 v_1)，S 随之出现了 1 次，当元素 S 的父元素 P 第 2 次出现时(节点 v_2)，S 没有随之出现(即出现 0 次)，因此，元素 S 随着其父元素 P 出现的最少次数为 0，最多次数为 1，即 S 在元素路径模型 EPM_{v_4} 中 min 值为 0，max 值为 1。

节点 v_4 的路径表达式：

$$P_{v_4}=C^{<1,1>}\cdot P^{<1,1>}\cdot S^{<0,1>}$$

根据下文介绍的算法合并节点 v_3 和 v_4 的路径表达式得到元素序列表达式：

$$ESE_{(v_3,v_4)}=C^{<1,1>}\cdot P^{<1,1>}\cdot B^{<1,1>}\cdot S^{<0,1>}$$

2.2　元素序列表达式的合并操作

XML 文档树全部叶节点的路径表达式集合，包含文档中的全部元素，需要合并路径表达式来简化集合，减少集合中的相同元素。合并路径表达式等同于合并元素序列表达式。合并操作分为 2 类：直接连接和因式分解。2 类合并操作既要考虑元素或元素序列表达式之间的逻辑关系变化，还要考虑元素 min 和 max 值的变化。

定义 4 直接连接

设 2 个元素序列表达式 ESE_1 和 ESE_2 ，其公共前缀为 ESE_f ，将 ESE_1 和 ESE_2 直接连接得到的元素序列表达式为 ESE_1ESE_2 ，其中， ESE_2 中属于 ESE_f 的元素是冗余元素，全部标记为“ 待修剪 ”。删除冗余元素，得到元素序列表达式为

$$ESE_1(ESE_2-ESE_f)$$

直接连接的运算符为逻辑与运算符。执行直接连接操作，各元素的 min 和 max 值不变。例 1 中的元素序列表达式 $ESE_{(v_3,v_4)}$ 是将节点 v_3 和 v_4 的路径表达式直接连接得到的。

定义 5 因式分解

设 2 个元素序列表达式 ESE_1 和 ESE_2 ，其公共前缀为 ESE_f ，公共后缀为 ESE_s ，将 ESE_1 和 ESE_2 因式分解得到的元素序列表达式为

$$ESE_f[(ESE_1-ESE_f-ESE_s)|(ESE_2-ESE_f-ESE_s)]ESE_s$$

因式分解的运算符为逻辑或操作符。执行因式分解操作后，含有逻辑或运算符的因式中元素的 min 值为 0，max 值不变；其余元素的 min 值为 2 个元素序列表达式中该元素最小的 min 值，max 值为 2 个元素序列表达式中该元素最大的 max 值。例 1 中，将节点 v_3 v_4 v_5 的路径表达式合并得到元素序列表达式：

$$ESE_{(v_3,v_4,v_5)}=C^{<1,1>}\cdot P^{<1,1>}\cdot B^{<1,1>}\cdot S^{<0,1>}\cdot D^{<1,1>}$$

将节点 v_6 v_7 v_8 的路径表达式合并得到元素序列表达式

$$ESE_{(v_6,v_7,v_8)}=C^{<1,1>}\cdot P^{<1,1>}\cdot B^{<1,1>}\cdot L^{<0,1>}\cdot D^{<1,1>}$$

将 $E_{(v_3,v_4,v_5)}$ 与 $E_{(v_6,v_7,v_8)}$ 因式分解得到元素序列表达式

$$ESE=C^{<1,1>}\cdot P^{<1,1>}\cdot B^{<1,1>}\cdot(S^{<0,1>}|L^{<0,1>})\cdot D^{<1,1>}$$

执行合并操作得到的元素序列表达式都可能会含有冗余元素。有些冗余元素在后续的合并操作中会起到非常重要的作用，将这样的冗余元素标记为“ 待修剪 ”状态，待全部元素路径表达式合并完毕后，执行修剪操作，删除冗余元素。

2.3　元素序列表达式的合并算法

元素序列表达式的合并算法中需要 2 个重要的概念：存储区和子序列表达式

定义 6 存储区

存储区是元素序列表达式的存储单元，一个存储区的空间能够且只能够存储一个元素序列表达式。

定义 7 子序列表达式

子序列表达式是元素序列表达式中以某个根元素开始，到下一个根元素结束的元素序列。如果某个根元素后面没有

其他根元素，则该根元素与其后所有元素构成一个子序列。如元素序列 *rabcrabd*(*r* 是根元素)的子序列分别为 *rabc* 和 *rabd*。

元素序列表达式的合并算法依据如下规则(用文档树的节点表示元素)：存储区的初始状态为 NULL。

规则 1 如果存储区 SU 状态为 NULL，当前路径表达式直接置入 SU，min 和 max 值都设置为 1。

对于 XML 文档树中 2 个相邻的叶节点 v_i 和 v_j，其父节点分别为 v_{pi} 和 v_{pj}，在 v_i 的路径表达式已完成合并操作的前提下，有如下合并规则：

规则 2 如果 v_{pj} 的路径 $P_{v_{pj}}$ 在存储区 SU 中出现过，而且 v_{pi} 与 v_{pj} 是同一个节点，那么，将 P_{vj} 与 SU 中的元素序列表达式进行直接连接并删除冗余元素

规则 3 如果 v_{pj} 的路径 $P_{v_{pj}}$ 在存储区 SU 中没有出现过，在 *SU* 的元素序列表达式和 P_{vj} 中分别寻找 v_i 和 v_j 相同的祖先节点，记作 v_a。在元素序列表达式中截取 v_a 所属的子序列表达式 S_{v_a}，将 $P_{v_{pj}}$ 与 S_{v_a} 直接连接得到的元素序列表达式与 SU 中的元素序列表达式进行因式分解。

规则 4 如果 v_{pj} 的路径 $P_{v_{pj}}$ 在存储区 SU 中已经出现过，但是 v_{pi} 与 v_{pj} 是不同节点，将 P_{vj} 作为一个新的元素序列表达式置入临时存储区 TSU，在 TSU 中递归地根据 4 条规则合并后续的叶节点路径表达式直到节点 v_j 的全部兄弟节点所属路径表达式合并完毕为止。最后，将 TSU 与 SU 中的元素序列表达式进行因式分解。

根据合并规则得出的元素序列表达式合并算法如下：

设：存储区中的元素序列表达式为 ESE_{SU}，

临时存储区的元素序列表达式为ESE_{TSU}

参数：当前存储区 SU 和元素序列表达式 ESE

```
GenerateSequence(SU, ESE){
 if(存储区 SU=NULL){
    ESE_SU=当前ESE;
    ESE_SU中每个元素的min和max值设置为 1;}
 if(P(v_j) ⊂ ESE_SU){
    if(v_pj=v_pi){
       Join(ESE_SU, P_vj);//直接连接操作
       PurneElements(ESE_SU);//删除冗余元素}
    else{if(P_vj的全部兄弟节点所属路径表达式合并完 //毕){
       从临时存储区返回ESE_TSU;//因式分解操作
       ESE_SU=Factoring(ESE_SU ,ESE_TSU);}
     else{创建临时存储区 NewTSU;
//在临时存储区递归合并 ESE
 ESE_TSU=GenerateSequence(NewTSU,P_vj);}}}
 else{在P_vj和ESE_SU中寻找v_a;//P_vj ⊄ ESE_SU
//寻找v_a所属的子序列表达式
  SubSUE=GetSubSequenceExp(v_a);
  创建临时存储区 NewTSU;
  ESE_TSU=Join(SubMSUE,P_vj);
  从临时存储区返回ESE_TSU ;
  ESE_SU=Factoring(ESE_SU,ESE_TSU); //因式分解}
 return ESE_SU;}
```

以一个简单的 XML 文档(以 XML 文档树的形式表示)为例，如图 2 所示。根据合并算法，图 2 中的叶节点路径表达式合并完毕得到的元素序列表达式为

$$rabE(F|K)G(qPN|dL)cHI(J|M)$$

标记 min 和 max 值不都为 1 的元素得到表达式：

$$ra^{<2,2>}bE(F^{<0,1>}|K^{<0,1>})G(q^{<0,1>}PN|d^{<0,1>}L)cHI(J^{<0,1>}|M^{<0,1>}) \quad (1)$$

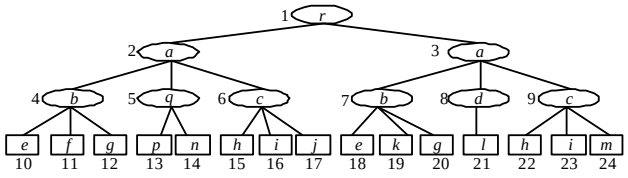

图 2　简单的 XML 文档树示例

根据文献[4]中的转换规则，可以将合并完毕得到的元素序列表达式，如式(1)转换为 XML Schema。XML Schema 中元素 *b* 的定义如图 3 所示。

```
<element name ="b">
    <ComplexType>
     <sequence>
          <element name ="e"/>
          <choice >
          <element name ="f" minOccurs="0" maxOccurs=="1" />
          <element name ="k" minOccurs="0" maxOccurs=="1"/>
          </choice>
          <element name ="g"/>
     </sequence>
     </ComplexType>
</element>
```

图 3　元素 *b* 的定义

3　实验分析

实验目的是比较 XStruct 系统基于 ECM 与本文基于 EPM 的 XML Schema 提取方法的时间和空间代价。实验环境为 P4 处理器 3.0 GHz，512 MB 主存，Windows XP 操作系统和 Visual Studio.NET 程序开发环境；数据集来自 http://monetdb.cwi.nl/xml/。实验结果如图 4、图 5 所示。

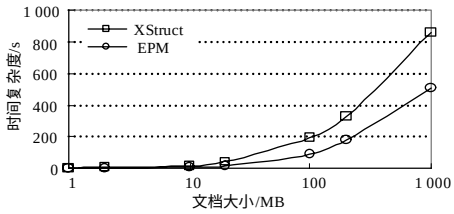

图 4　时间复杂度比较

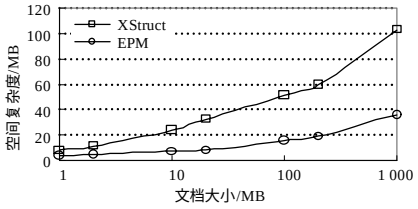

图 5　空间复杂度比较

实验结果表明，基于 EPM 的 XML Schema 提取方法所占用的存储空间和执行时间均优于 XStruct 方法，而且随着文档规模的增大，这种优势越来越明显。XStruct 为每一个元素节点建立 ECM，显然，存储所有元素的 ECM 需要大量的存储空间，而且，寻找父子元素关系需要多次遍历 XML 文档中的元素。基于 XML 文档元素路径模型的方法在使用 SAX 解析 XML 文档的同时，生成文档树叶节点的路径模型即路径表达式，将该表达式与存储区表达式(前一次合并操作的结果)根据规则进行合并。从合并算法中可知，只需一个主存储区和有限个临时存储区的空间就可以完成合并。另外，只需对 XML 文档元素进行一次遍历在时间代价方面也要优于 XStruct 系统。

(下转第 37 页)