第35卷 第13期 计 算 机 工 程 2009年7月
Vol.35 No.13 Computer Engineering July 2009
·软件技术与数据库· 文章编号：1000—3428(2009)13—0058—03 文献标识码：A 中图分类号：TP311

基于XSLT模板展开的对等模式映射合成

赵智超[1]，赵 政[2]
(1. 中国电子科学研究院，北京 100041；2. 天津大学计算机科学与技术学院，天津 300072)

摘 要：针对对等数据管理系统中绕过离开节点时多次重写的耗时问题，提出一种基于XSLT模板展开的对等模式映射合成方法。处理模板匹配风格的XSLT子集表示的2个映射，通过按序对模板内容再次匹配展开，形成等价于多个映射的合成映射。当映射路径上的相邻节点一起离开时，代理节点使用合成映射直接绕过。仿真结果表明，合成方法能够正确生成等效映射，绕过时间缩短，网络拓扑更加强健。
关键词：映射合成；对等计算；数据共享

Peer-to-Peer Schema Mapping Composition Based on XSLT Template Unfolding

ZHAO Zhi-chao[1], ZHAO Zheng[2]
(1. China Academy of Electronics and Information Technology, Beijing 100041;
2. School of Computer Science and Technology, Tinajin University, Tianjin 300072)

【Abstract】Aiming at the time-consuming problem of multiple reformulations when bypassing left peers in peer data management system, a peer schema mapping composition method is proposed based on XSLT template unfolding. By matching and unfolding template contents consecutively, the method processes two mappings expressed in an XSLT subset with the style of matching templates, and forms a composed mapping equivalent to multiple mappings. When neighboring peers on mapping path depart together, agent peer no longer reformulates multiple times, but uses composed peer to bypass directly. Simulation results show that the composition method can produce equivalent mappings correctly. By passing time shortens significantly and network topology becomes robust.
【Key words】mapping composition; Peer-to-Peer computing; data sharing

1 概述

基于对等架构的设计消除了系统中的关键访问点，使系统更强健。但是对等节点的自治性使其能动态加入和离开，当其离开时映射路径断裂，查询不能继续转发，造成数据共享范围大幅缩减。文献[1]给出解决网络拓扑动态性问题的方法，采用模式映射沿映射路径提前备份的机制。但该方法还有一定缺点，当节点连续离开时，需受阻查询所在节点代理进行多次重写，非常耗时，且这些重写之间存在多个过渡性查询，但这些查询可执行的节点都已离开，因此，除了作为中间形态，它们的存在没有任何其他意义。

如事先能构造出一个从受阻查询所在节点的模式到路径前方仍在系统中节点模式的映射，并保证这一映射在查询重写效果上等价于需跨过的一串映射的连接使用，则当网络中出现节点连续离开时，直接使用该等价映射就可绕过断裂处，从而避免多次重写要耗费的大量时间，且不会产生无执行可能性的查询。

获得上述等价映射的方法是映射合成。已知数据源模式σ_1与σ_2之间的映射m_{12}和σ_2与σ_3之间的映射m_{23}，m_{12}与m_{23}的映射合成产生一个σ_1与σ_3之间的映射m_{13}，等价于原映射的连接使用。等价的意思是，对于查询语言Q中的每一个σ_1上的查询Q，满足数据源模式σ_1，σ_2和σ_3的任何数据库实例，用m_{13}重写Q与用m_{12}与m_{23}2次重写Q，2种方法得到的查询分别在同一个σ_3实例数据库上执行，得到的查询回答相同。

映射合成可起到快速绕过、节约时间的作用，实际上合成产生的直接映射添加到原系统网络拓扑中后，会使拓扑获得更强健的结构。如图1所示，上边从A到F的映射路径，无论其中哪一个节点离开，路径都会断裂，部分数据源无法继续共享。下面是映射合成完成后添加了直接映射的新拓扑，其对节点离开的适应能力明显加强。任何节点离开造成的后果只是它本身的数据无法继续共享，而对其他节点数据的共享没有产生影响。这对节点自治、网络高度动态化的对等数据管理系统来说非常需要。

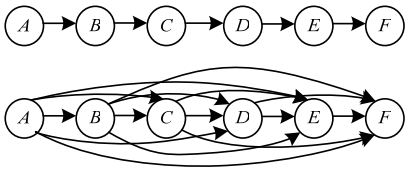

图1 映射合成产生的直接映射对系统映射拓扑的影响

2 相关工作

映射合成也在模式进化问题中用到，以合并多步进化。已有的研究主要针对关系模型。文献[2]表明用GLAV表示的两映射合成可能无法用有限数量的一阶约束表示。文献[3]引入一种二阶映射语言，它在合成下封闭，但标准数据库工具

作者简介：赵智超(1980－)，男，博士，主研方向：分布式数据库；赵 政，教授、博士生导师
收稿日期：2009-01-13 **E-mail：**zhaozhichaomail@gmail.com

不支持二阶语言。文献[4]对 XML 数据模型上的映射合成进行研究，针对的映射语言为 XQuery 的子集。

作为 XML 查询语言，XQuery 有很多优点，但对 XML 的映射表达来说，XML 变形语言 XSLT 有天然的优势。XQuery 输出文档的完整结构需直接在语言中给出，这显然是针对查询输出文档不会太复杂的应用设计的。在映射的应用场景下，两边都是结构复杂的模式，在 XML 中更可能出现无限嵌套的情况。另外，XML 映射常会有满足相对语境对应的需求，这时 XQuery 无法满足要求。而 XSLT 既然能指导各种结构 XML 间的变形，就能记录它们间的对应信息[5]。

3 模板展开

这里给出的映射合成方法只适用于 XSLT 的一个子集，因 XSLT 中的一些指令在对等数据管理系统的模式映射表达场景下无用，或与模板匹配思想不一致。即使这样，它完全提供映射所需指出对应的能力。其余部分是为习惯流程控制表达方式的使用者提供方便，没有增加新功能。

这一映射合成方法采用模板连续展开。设 m_{12} 和 m_{23} 都用 XSLT 表达，每个映射的样式表中都有一组模板规则。m_{12} 中一个模板规则匹配 σ_1 中的节点，产生 σ_2 中一个片断。m_{23} 中一个模板规则匹配 σ_2 中的节点，产生 σ_3 中一个片断。而所求的 m_{13} 中一个模板规则匹配 σ_1 中的节点，产生 σ_3 中一个片断。因此，产生 m_{13} 中一个模板规则的方法是，将 m_{12} 中一个模板规则中要产生的 σ_2 片断中的节点，继续用 m_{23} 中的匹配模板规则展开，直到片断中只有 σ_3 的节点，这样就得到一个 m_{13} 中的模板规则。

映射合成过程是，首先检查 m_{12} 中的模板规则，如其输出片断中无任何模式 σ_2 中的节点，则该规则也就不需再用 m_{23} 中的模板规则展开，只需将其直接作为 m_{13} 中的一个模板规则。如 m_{12} 中的模板规则：

```
<xsl:template match="student">
<xsl:apply-templates select="name" />
</xsl:template>
```

但有一个特例，如果模板规则的式样匹配 σ_1 的根节点，那么目标片断中即使无任何显式 σ_2 中的节点，该规则也须展开。因为 σ_1 的根节点匹配产生 σ_2 根节点，而 σ_2 根节点在目标片断中不可见。如 m_{12} 中的模板规则：

```
<xsl:template match="/">
<xsl:apply-templates select="people" />
</xsl:template>
```

虽然其中不显式包含 σ_2 中任何节点，但隐含了 σ_2 根节点，因此，匹配 m_{23} 中的模板规则：

```
<xsl:template match="/">
<result>
<xsl:apply-templates select="people" />
</result>
</xsl:template>
```

用该规则进行展开的结果是 m_{13} 中的模板规则：

```
<xsl:template match="/">
<result>
<xsl:apply-templates select="people" />
</result>
</xsl:template>
```

展开时需对 apply-templates 指令 select 属性的不同情况分别处理。设 2 个匹配的模板规则为 *a* 和 *b*，*a* 需要用 *b* 展开。如 *a* 中有 apply-templates 指令，*b* 中无，说明即使前一个映射对当前节点产生后代，*b* 也不对其处理，因此，*a* 中 apply-templates 指令等于不存在。如果 *b* 中有 apply-templates 指令，*a* 中无，则将 *a* 的目标片断作为源树处理。如果 *a* 和 *b* 中都有 apply-templates 指令且都没有 select 属性，则展开后的 apply-templates 指令也没有 select 属性。如 *a* 中 apply-templates 指令有 select 属性，*b* 中 apply-templates 指令无，则展开后 apply-templates 指令的 select 属性与 *a* 中相同。如果 *a* 中 apply-templates 指令无 select 属性，*b* 中 apply-templates 指令有，则展开后 apply-templates 指令的 select 属性为 *b* 中 select 属性在模式 *A* 中的对应。如 *a* 和 *b* 的 apply-templates 指令都有 select 属性，则展开后 apply-templates 指令的 select 属性为 2 个条件的与。对上面 2 个模板规则的连续展开，它们中的 apply-templates 指令都有 select 属性，后一个模板规则的 select 属性为 people，需找出它在模式 *A* 中的对应。由模板

```
<xsl:template match="people">
<people>
<xsl:apply-templates />
</people>
</xsl:template>
```

可知，模式 σ_2 中 people 对应模式 σ_1 中的 people，而前一个模板规则中的 select 属性也为 people，这样 2 个 apply-templates 指令中的 select 属性条件重合，因此，展开后模板 apply-templates 指令中的 select 属性也为 people。

在展开模板规则

```
<xsl:template match="name" mode="std">
<student>
<xsl:apply-templates />
</student>
</xsl:template>
```

时，σ_2 中的 student 元素在 m_{23} 中找不到匹配的映射，但要注意这时可用在 m_{23} 中的内置模板规则展开，即对子节点继续应用模板规则，展开后的模板规则为

```
<xsl:template match="name" mode="std">
<xsl:apply-templates />
</xsl:template>
```

4 网络服务与协议设计

为支持映射合成，需修改文献[1]中的网络服务基础设施。节点保存自己与邻居的原有映射和合成后产生的直接映射，直接映射可看成节点与通过合成找到的新邻居间的映射，这样节点保存映射由层次结构变为平等关系，索引信息也可用一张表表示。索引表由 2 列组成：一列是邻居节点 IP 地址，另一列是与该邻居映射在本地节点中的保存位置。文献[1]中给出映射传递服务仍需保留，不同的是，这时返回的是索引表，而不是索引树。

节点的映射获取程序先按索引取回映射，将上游映射与本地映射合成，将合成后的映射放入本地映射库，在本地索引表中插入对应的一行信息，上游映射抛弃。由于网络拓扑会发生改变，映射获取程序需每隔一段时间重新获取上游映射的信息，查看是否有新映射建立或原有映射撤销。要快速查看上游映射的增删情况，需对上游索引表进行备份。

添加直接映射后，网络中两节点间的映射路径变得非常多，且大部分是等价路径，这时需记录查询中的查询号，只

对第 1 次出现的查询号进行本地数据库的回答和重写转发。

5 实验

在映射合成方法的实现中，用 libxml2 解析 2 个输入映射。对于 m_{12} 中的每个 template 节点，进行上述模板连续展开处理。因为输出中的 template 元素本身并没对 m_{12} 中的 template 元素做任何改变，只是其内容需连续展开，所以可直接复制输出 stylesheet 和 template 元素。如 template 内容中无 apply-templates 指令，可直接使用 libxslt 来处理其内容，将 m_{23} 解析为样式表指针，然后用 xsltApplyStylesheet 函数来对 template 的子节点进行匹配，将生成内容作为子节点，添加到合成后映射中对应的 template 下。按上述方法对 m_{12} 中的每个 template 元素处理完成后，就生成完整合成映射。

用多对符合上述条件的 XSLT 子集表示的映射进行合成，可得到正确结果。正确结果的检验方法与映射合成的定义相一致，即用合成映射对数据源变形，得到的结果应与用 2 个映射对数据源 2 次连续变形得到的结果一样。在实验中有些映射对会因无法合成而失败，原因是在模板规则 b 中的 apply-templates 指令有 select 属性时，需找出它在 σ_1 中的对应。如这一属性的式样非常复杂，就很难通过 m_{12} 找到对应式样。如算法复杂度用执行搜索的次数衡量，因为需多次搜索，所以 a 和 b 中的 apply-templates 指令都由 select 属性展开，过程最耗时。

对 1 个~8 个节点同时离开的情况，分别用映射合成方法和备份机制，对它们的离开节点绕过效率比较。实验对 20 组不同映射组成的路径进行 10 次不同查询，为方便不同映射情况的比较，耗时用相对比率计算，即相对每组映射直接访问的耗时。对各组映射路径耗时比率平均，得到近似一般情况的效率比较。

在图 2 中，实线代表映射合成方法在绕过不同数目离开节点时的效率，虚线代表映射备份机制在绕过不同数目离开节点时的效率。由于在映射合成方法中，查询通过直接映射绕过离开节点，因此绕过节点数目对效率的影响不大。随着离开节点数增加，其效率略有上升是因为跨过多个节点的直接映射需多次合成，合成的次数越多，产生的直接映射越复杂，造成耗时有轻微增加。

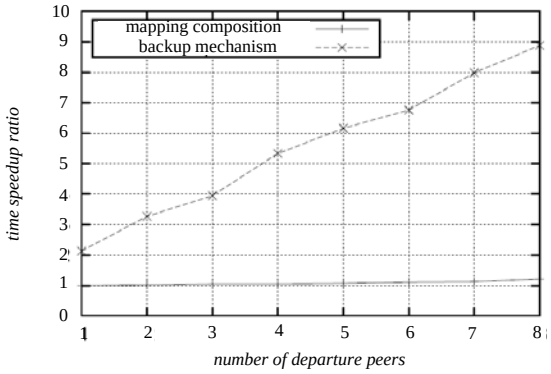

图 2 映射合成与备份机制绕过离开节点的效率比较

6 结束语

本文提出的映射合成方法能够正确地处理 XSLT 子集表达的模式映射，不仅可强健网络拓扑，而且使查询响应速度大大加快。但这一方法还只限于处理 XSLT 映射子集，对映射设计者构成限制。下一步需构造一种映射表达方式，能根据具体情况决定合成还是多步处理，即可消除该缺陷。

参考文献

[1] Zhao Zhichao, Zhao Zheng, Zhang Jie, et al. Dynamic Coordination Rules in Peer-to-Peer Database[C]//Proceedings of SPIE'06. Bellingham, Washington, USA: [s. n.], 2006.

[2] Madhavan J, Halevy A Y. Composing Mappings Among Data Sources[C]//Proceedings of the 29th International Conference on Very Large Data Bases. Berlin, Germany: [s. n.], 2003: 572-583.

[3] Fagin R, Kolaitis P G, Popa L, et al. Composing Schema Mappings: Second-order Dependencies to the Rescue[J]. ACM Transactions on Database Systems, 2005, 30(4): 994-1055.

[4] Tatarinov I, Halevy A. Efficient Query Reformulation in Peer Data Management Systems[C]//Proceedings of the ACM SIGMOD'04. New York, USA: ACM Press, 2004: 539-550.

[5] Zhao Zhichao, Zhao Zheng, Shi Qingwei. Dependence Topology Optimization in Dynamic Peer-to-Peer Database Network[C]// Proceedings of SPIE'07. Bellingham, Washington, USA: [s. n.], 2007.

编辑 索书志

（上接第 19 页）

参考文献

[1] 李仁杰. 基于监控的可信软件构造技术研究与实现[D]. 长沙: 国防科技大学, 2007.

[2] 徐 理. 基于 AOP 的应用软件监控技术研究[D]. 长沙: 国防科技大学, 2007.

[3] Object Management Group Inc.. Unified Modeling Language Specification, Version 2.1.2[Z/OL]. (2007-11-01). http://www.omg.org/.

[4] Object Management Group Inc.. OCL 2.0 Specification[Z/OL]. (2006-05-01). http://www.omg.org/.

[5] Database Systems Group. Use a UML Based Specification Environment[Z/OL]. (2007-05-16). http://www.db.informatik.uni-bremen.de/projects/USE/.

[6] Gogolla M, Bohling J, Richters M. Validating UML and OCL Models in USE by Automatic Snapshot Generation[J]. Journal on Software and System Modeling, 2005, 4(4): 386-398.

[7] Arlow J, Neustadt I. UML2 and the Unified Process: Practical Object-oriented Analysis and Design[M]. 2nd ed. [S. l.]: Person Education Inc., 2005.

[8] Richters M, Gogolla M. Aspect-oriented Monitoring of UML and OCL Constraints[C]//Proc. of the 4th AOSD Modeling with UML Workshop. San Francisco, CA, USA: [s. n.], 2003.

编辑 陈 晖