基于关系树模型实现XML数据转换

杨文军　李涓子　王克宏
(清华大学计算机科学与技术系　北京100084)

摘　要　目前在同一行业内使用着多种XML模式语言,因此XML数据转换已成为数据交换的研究热点。当前一些转换模型不能清晰而有力地描述转换语义,为此我们提出了一种关系树模型,它能有效地把XML层次化的特点和成熟的关系理论结合在一起,能处理XML元数据而不是XML实例,并支持DTD和W3C XML模式语言。基于这一模型的转换语义不仅能进行简单元素的映射,而且也能提供复杂关系代数运算的能力,例如整合。这种转换语义中还提供算术运算,极限运算,统计运算和关系演算等多种运算,增加了该模型的转换能力。

关键词　XML模式,转换模型,转换语义

Transforming XML Data Based on Relational-Tree Model

YANG Wen-Jun　LI Juan-Zi　WANG Ke-Hong
(The Dept. of Computer Science & Technology, Tsinghua University, Beijing 100084)

Abstract　Because different kinds of XML schema languages are used in the same field, transformation between XML data which conforms to different schemas is one of the key research points for data exchange. Current transform models are unable to describe the transforming semantics clearly and powerfully, so we give a new one in this paper called Relation-Tree Model, which combines effectively the XML hierarchical character and mature relational theory together. The model deals with XML metadata rather than XML instance and it supports both DTD and W3C XML Schema language. The transform semantics based on this model can not only map among simple elements but also provide ability of complex relational algebraic operation, for instance, integration, which is naturally different to traditional one-to-one mapping. Several kinds of operations including arithmetic operation, limitation operation, statistics operation and relational calculus are also offered in this transform semantics.

Keywords　XML schema, Transform model, Transform semantics

1　引言

目前众多的数据交换标准为各个领域内提供了统一的数据交换平台,但是在同行业内的数据交换和表示仍然存在多个互不兼容的数据格式。例如在电子商务中ebXML (Electronic Business XML) 和CXML (Commercial XML) 均被广泛采用,而小的应用系统为了加快开发和方便维护,通常定义符合自身需求的数据交换格式。以图1中产品采购系统为例,该产品采购系统同时从网络中获取若干个产品供应商以XML格式提供的产品清单,这些产品清单分别符合不同的产品供应商定义的数据模式。在产品采购系统内,XML数据转换引擎组件首先把这些产品清单数据转换为系统可识别的XML格式,并把生成的XML数据作为输入,提供给后续处理流程(入库,查询,浏览等)。如何实现从异构XML数据(符合不同的XML模式)的转换是各种集成系统迫切需要解决的问题。

XML数据转换引擎面临两个方面的问题:一方面是如何根据用户的转换意图准确地描述XML数据的转换语义。DTD或者XML Schema均可以灵活定义XML元素间的层次结构和出现频率,因此引擎需要提供一套完整且清晰的转换描述语言来表示转换语义;另一方面是如何能够快速更新转换规则以满足不断变更的转换需求。从图1可以看出,XML数据转换引擎往往需要把符合多种不同模式的XML数据转换为符合同一种模式的XML数据,而且模式会随着应用需求不断添加或更新,所以需要规则能够快速生成。而作为数据转换引擎的核心的XML转换模型是解决这些问题的关键。

目前存在多种XML转换模型,包括树模型,OEM图模型,关系模型和自动机模型等。其中使用较为广泛的是树模型[3]和关系模型[1],而XSLT[5]语言是基于树模型的使用较为广泛的转换语言。基于树模型表示的转换语义的表达方式不同于用户的转换意图,因此开发周期长,不能满足目前转换引擎快速机动的需求;关系模型是把XML数据看成是一个关系数据源,XML数据经过数据库的导入导出实现转换。由于关系模式定义的是二维的数据结构,XML数据存入数据库后将丢失XML元素之间的层次关系,从而不能实现无损的数据转换。为此,本文提出了一种称之为关系树模型的XML转换模型,其主要思想是:根据XML模式文档中定义的元素和属性的出现频率,对XML模式文档中定义的元素和属性进行聚类划分,得到若干个仍然保持层次关系且内部节点具有相同出现频率的节点集。基于该模型的转换语义分为三个层次表示,其中处于中间层的节点集映射将包含多个处于底层的节点映射。每个节点集映射在转换过程中将生成一个关系实例,这些关系实例在经过处于上层的关系运算后生成目标XML数据。转换语义通过关系树模型把各种关系运算融入到转换过程中,转换语义更为清晰。

2　关系树模型

2.1　XML模式文档建模

关系树模型针对XML模式描述转换语义,所以本节首先对XML模式建模。目前Document Object Model (DOM) 是比较成熟的层次化数据模型,主要描述文档本身的结构和内容;而本模型关注模式文档自身所定义的元素和属性的内容,且同时支持DTD(Document Type Definition)和XL Schema两种XML模式语言。我们称这种新的XML模式模型为Schema Logic Tree (模式逻辑树,SLT),是一种层次化数据模型。除根节点外,SLT还包括两类节点。一类是由模式

文档中元素或属性的定义而生成的元素节点(为方便转换,该模型把属性的定义看成元素定义来处理);另外一类是表示子元素关系的逻辑节点,包括序列节点和选择节点。序列节点表示子节点之间先后顺序出现;选择节点表示子节点之间只有一个节点能够出现。每一种节点都有两个属性,即"min_occurs"和"max_occurs",来表示它们的出现频率。它们的取值取决于DTD文档中的('?','*','+')等符号或者W3C XML Schema文档中的同名属性。图2给出了SLT实例,由图2左侧的DTD文档生成。

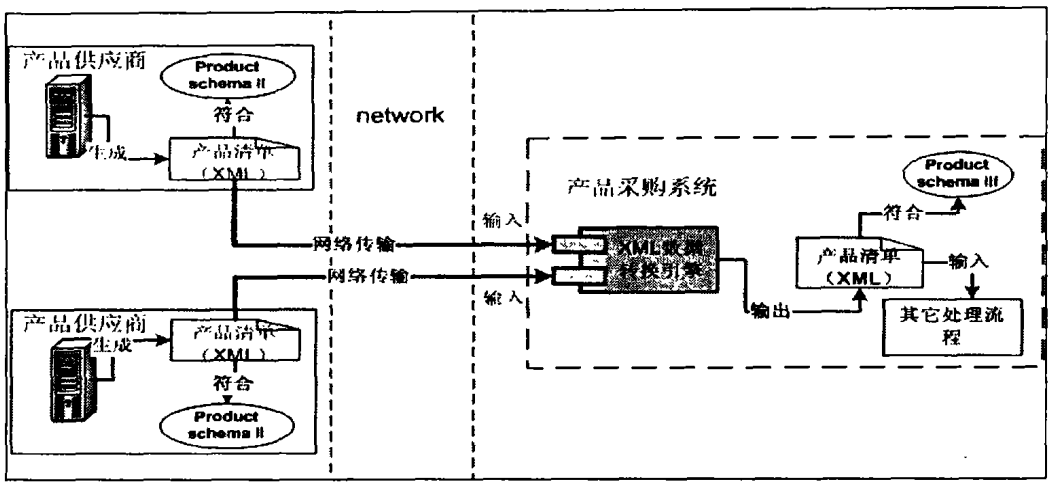

图1 产品采购系统中的XML数据转换引擎

根据节点内容的定义,节点可分为包含文本内容的数据节点和不包含文本内容的结构节点。由属性定义生成的元素节点和元素内容为"PCDATA"的元素节点均为数据节点,而其它节点均为结构节点。在图2中,*name*、*work_id*、*position_A*、*position_B* 和 *wage* 是数据节点。在SLT树上,从节点 N_i 到节点 N_j 的路径表示从节点 N_i 路由到节点 N_j 所遍历的节点的集合(包括节点 N_i 和节点 N_j),而节点 N_i 不要求必须是节点 N_j 的祖先节点。节点路径的准确定义如下:

定义1 $path(N_i,N_j)$ 表示从节点 N_i 到节点 N_j 的路径,定义为一个以 N_i 为起始节点 N_j 为终止节点的节点序列,在该序列上节点不能重复出现且任意两个相邻节点为父子或子父关系。

在图2的例子中,$path(name, work\_id)$ 为 $\{name, sequence\_A, employees, employee, sequence\_B, work\_id\}$。

2.2 关系树模型概念

SLT把模式文档表示成节点树,是建立关系树模型的基础。根据XML模式语言的特点,有些元素在XML文档中在同一个上下文内具有相同的出现频率。例如,在DTD文档片断(!*ELEMENT A (B,C,D)*)中,B、C和D元素在XML文档中同步出现。关系树模型就是根据元素的同步特点在SLT

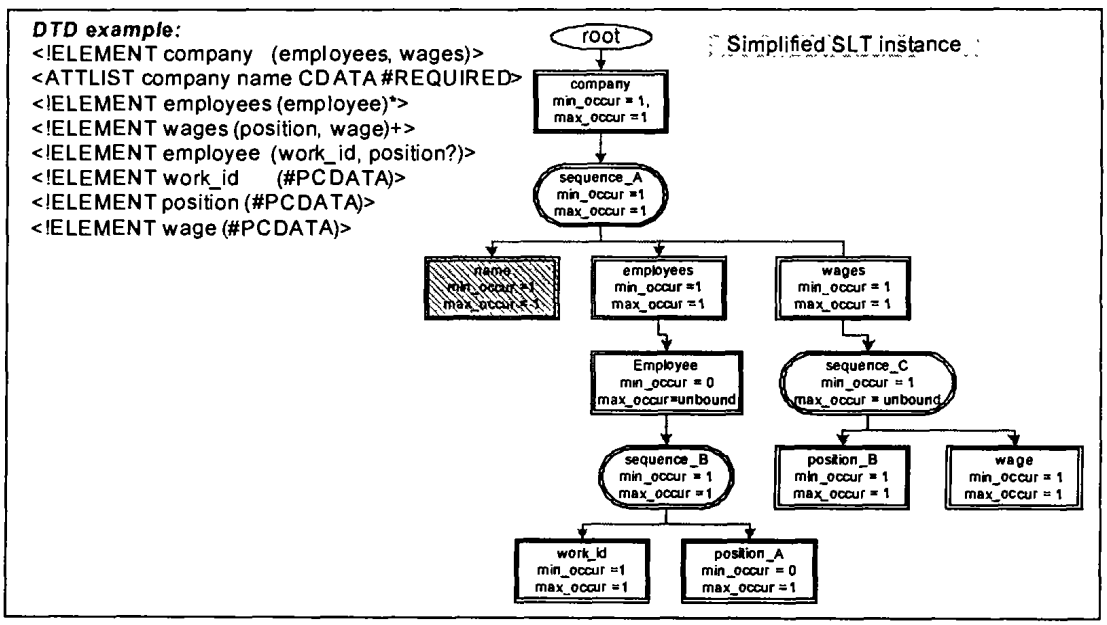

图2 Schema Logic Tree (SLT) 例子

模型上对节点进行聚类划分,并重新构造由聚类后得到的节点集作为节点构造新的树结构。

2.2.1 节点聚类划分 节点的出现频率是根据节点本身的"min_occurs"和"max_occurs"属性以及节点所处的位置来决定的,所以,我们需要首先定义节点维度来量化节点的出现频率。

定义2 $degree(N)$ 表示节点 N 的维度,值为 SLT 中从根节点到节点 N 的路径上"*max_occurs*"属性值大于1的节点的数目。

根据节点维度的定义,SLT实例中每个节点均有一个维

度值。例如，图2的 *employees* 维度为0，而 *employee* 的维度为1。

节点维度相同还不能完全表示它们具有相同的出现频率，因为它们可能处于不同的上下文中。以图2为例，*work_id* 和 *wage* 具有相同的维度，但在相应的 XML 文档中并不同步出现，所以我们需要更严格的约束来划分节点。

定义3 相关节点集(用 *R-Set* 表示) 定义为在某个 *SLT* 实例中满足下列条件的节点集合：该集合中的任意两个节点的路径上(包括这两个节点)的节点均有相同的维度。最大相关节点集(用 *M-R-Set* 表示)表示在这个 SLT 实例中所有满足该条件的节点均包含在内的相关节点集。

以图2中 SLT 为例，通过手工方式我们可以把所有节点划分为三个最大相关节点集：

M-R-Set A = {*root*, *company*, *sequence_A*, *name*, *employees*, *wages*;

M-R-Set B = { *employee*, *sequence_B*, *work_id*, *position_A* };

M-R-Set C = { *sequence_C*, *position_B*, *wage*}

根据相关节点集的定义，任意一个 SLT 实例均可被划分为若干个最大相关节点集，且如果把 SLT 实例看成所有树节点的集合，那么这些最大相关节点集均为 SLT 的子集。为了给出一个有效的节点聚类算法，我们首先给出相关节点集的两个重要定理(证明略去)。

定理1 在同一个 SLT 实例中，任意两个最大相关节点集没有交集。

定理2 对于两个具有相同维度的节点，如果其中一个节点是另外一个节点的祖先，那么这两个节点属于同一个最大相关节点集。

定理1和2给出了一个实现 SLT 聚类划分的算法。我们中序遍历 SLT 树，并根据节点的维度值判断当前节点与其子节点是否同属一个最大相关节点集。算法1给出详细过程。

算法1 SLT 节点聚类算法

```
Main procedure:
1. construct an M-R-Set object and assign it to s (M-R-Set
   variable); construct a vector object and assign it to v (vector
   variable v); assign zero to the degree of root (document node)
   and insert root into s;
2. insert s into v as the first generated M-R-Set instance;
3. invoke the following recursive procedure Partition with root, s as
   input and v as output
Partition procedure:
1. Input: SLT_node N, M-R-Set instance s
2. Output: vector v (its elements are M-R-Set instances)
3. If N != null then for every child node (represented by C) in the
   node N
   a) if C.max_occuring > 1 then
      i. degree(C)=degree(N)+1; construct M-R-Set instance and
         assign it to s'; insert C into s'; insert s' into v;
      ii. Invoke the procedure Partition recursively with C, s' as input
         and v as output
   b) else
      i. degree(C)=degree(N); insert C into s;
      ii. Invoke the procedure Partition recursively with C, s as input
         and v as output;
```

2.2.2 *构造关系树模型* 根据算法1，我们把 SLT 实例划分为若干个最大相关节点集。为了保持 XML 的层次化特性，我们依据不同节点集中节点之间的层次关系来确定这些节点集的层次关系，以此构造新的以节点集为树节点的树结构。为此，我们首先给出相关节点集的层次关系的定义：

定义3 当相关节点集 *A* 中的某个节点 *a* 是相关节点集 *B* 中的某个节点 *b* 的祖先，那么相关节点集为相关节点集 *B* 的祖先。

根据相关节点集的定义以及定理1，我们可以把由节点作为树节点构成的 SLT 树形结构重新构造成由最大相关节点集作为树节点构成的新的树形结构，称之为关系树模型(Relational-Tree Model)。图3给出了根据图2的 SLT 树构造而成的关系树实例。关系树模型与 SLT 模型相比，它们都是对 XML 模式语言的建模，但是关系树模型在保留节点的层次关系的同时，把节点按照出现频率进行区域划分(即节点集的划分)，同一区域内的节点保持相同的出现频率，降低了树的复杂度，为转换时的区域映射提供了可行性。

3 转换语义

转换语义就是在源模式和目标模式中建立转换规则的语言，用于表示用户的转换意图。基于关系树模型的转换语义划分为三层底层是节点映射，即从源节点(源 SLT 模型中的节点)映射到目标节点(目标 SLT 模型中的节点)，其目的是定义如何从源节点获取数据并进行指定的运算操作，最后赋值给目标节点。处于中间的节点集映射是把源最大相关节点集

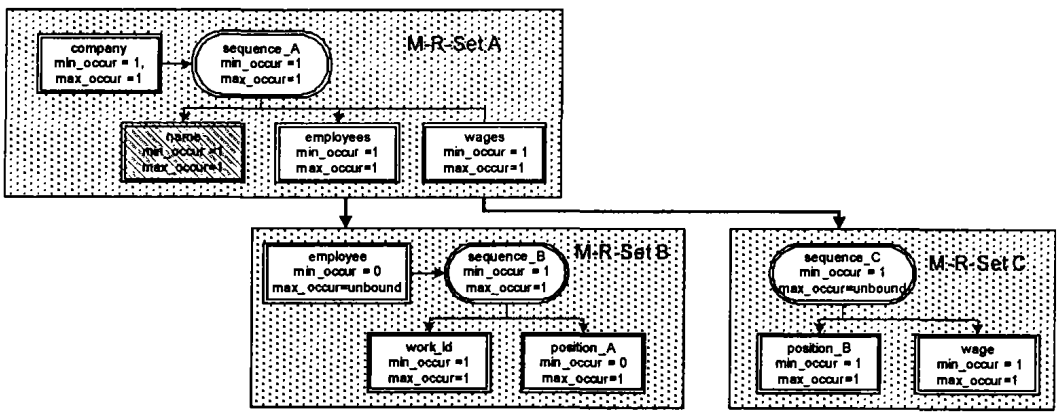

图3 由图2的 SLT 实例生成的关系树模型实例

(简称源节点集)映射到目标最大相关节点集(简称目标节点集)，其目的是定义一个关系模式且在转换过程中指导转换引擎生成一个符合该关系模式的关系实例。每个节点集映射包含若干个节点映射，表示节点取值的规则。每个目标节点集可以被映射多次，从而在转换过程中我们可以生成多个关系实例。处于最高层的关系运算设定这些关系实例的运算，其运算结果用于构造最终生成的 XML 文档。

◇*节点集映射层* 源节点集、目标节点集、上下文节点和节点映射数组构成一个节点集映射。上下文节点是源 SLT 实例的一个节点，它定义被映射的源节点集的上下文(在转换过程中，不同的上下文节点可以得到不同数量的源节点集)。而包含在节点集映射内的节点映射，是把源 SLT 实例中的文本

节点映射到目标节点集内的文本节点。单个节点映射的目的是为被映射的目标节点赋值，目标节点集内的每个文本节点只能被映射一次(因为只需要赋值一次)。而由这个节点集映射得到的关系模式，就是由这些在节点映射中被映射的目标文本节点组成的。关系模式的属性对应被映射的目标文本节点，属性名来自于节点名，属性的域是该节点的取值范围。同时节点集映射还可以加入过滤条件和排序条件来过滤和排序源节点集。

在转换过程中，根据节点集映射的上下文节点和过滤条件，我们可以从源 XML 文档得到多个源节点集实例(XML 子树)，转换引擎依次遍历源节点集实例并生成关系实例。对于每个被遍历的源节点集实例，转换引擎将以此作为上下文来解析该节点集映射中的节点映射来为目标文本节点赋值，从而为关系实例生成一行数据。因此，关系实例中的行数等于所选取的源节点集的数目。

◇*节点映射层* 节点映射是转换语义最基本的单元，且必须包含在某个节点集映射中。它的基本组成为源节点、目标文本节点、上下文节点和运算类型。这里上下文节点同样是源 SLT 中的一个源节点，用于选取源节点时提供上下文。节点映射的上下文可继承于所在的节点集映射的上下文。通过上下文节点，我们可以在转换过程中从 XML 文档中选取满足条件的源节点实例。在节点映射中需要设定的源节点的数目取决于运算类型。一一映射是最简单的运算类型，只需要设定一个源节点，在转换过程中目标文本节点的取值直接来自于该源节点。在节点映射中，运算类型可以根据用户的需要任意扩展。在我们实现的转换引擎中支持多种运算类型，包括字符串运算、算术运算、统计运算等。

◇*关系运算层* 在转换过程中，转换引擎为每个节点集映射生成一个关系实例，这个关系实例的模式由被映射的目标文本节点组成。同一个目标节点集可以设定多个节点集映射，从而在转换中可以生成多个关系实例。关系运算层对这些关系实例设定关系运算操作，根据运算结果(仍然是一个关系实例)生成最终的 XML 文档。

关系运算本身也是对关系模式的运算[4]，而由于关系运算的结果用于构造最终的 XML 文档，因此在设定节点集映射以及关系运算的时候要满足以下一致性条件：经过运算后的关系模式必须由该目标节点集内的所有文本节点构成。因为一旦有一个文本节点不在其中，那么该文本节点就不能在转换过程中正确赋值，也就无法生成最终的 XML 文档。以下几种关系运算操作在实际应用中使用较为广泛：

联合运算(Union)：如果最终生成的关系实例的事由多个由节点集映射生成的子关系实例横向合并得到，比如我们要把各个厂家提供的产品清单按照采购商的文档模式合成在一个 XML 文档中，那么我们可以采用联合运算来实现(用符号"U"表示)。

笛卡尔积运算(Cartesian-product)：当我们需要把两个子关系实例的数据进行纵向合并，比如某个厂家在一个 XML 文档中分别描述所有产品名称的列表和产品出厂日期的列表，通过笛卡尔积运算，可以把产品和出厂日期一一绑定在一起，生成一个新的文档。笛卡尔积运算用符合"x"表示，需要有两个节点集参与运算，生成的关系模式是原关系模式的合并。

其它的关系运算符，包括连接运算、半连接运算等均可以方便加入到关系运算层。

4 实验示例

我们以图2给出的 DTD 文档为源模式文档。该文档分别给出职员信息和工资信息，职员的工资数需要根据职员信息中的工作职位(position_A)到工资信息中去匹配相应的工作职位(position_B)才能得到。position 元素的作用等同于关系数据的外键。那么在目标 XML 模式文档中，我们希望职员的信息包括他们的工资金额，用 DTD 语言定义如下：

```
〈!ELEMENT employees(employee)*〉
〈!ELEMENT employee(work_id,position,wage)〉
〈!ELEMENT work_id # PCDATA〉
〈!ELEMENT position#PCDATA〉
〈!ELEMENT wage#PCDATA〉
```

根据聚类划分算法，该目标节点集可以划分为两个最大相关节点集：

M-R-Set A' ={*document'*,*employees'* }

M-R-Set B' = { *employee'*, *sequence_A'*, *work_id'*, *position'*,*wage'*}

而源模式文档已被划分为2.2.1节给出的三个最大相关节点集。设定转换规则如下：

[节点集映射：*M-R-Set B*〈------〉*M-R-Set B'*

包含的节点映射：*work_id* 〈------〉*work_id'*，*position_A* 〈------〉*position'*]

[节点集映射：*M-R-Set C*〈------〉*M-R-Set B'*

包含的节点映射：*position_B* 〈------〉*position'*，*wage* 〈------〉*wage'*]

[关系运算符为自然连接，参与自然连接的目标文本节点为 *position'*。]

5 相关工作比较

目前公开发布的 XML 数据转换引擎可以分为两类，其中一类转换引擎采用自定义脚本来描述转换语义，同时提供对这些脚本的解析器。由于自定义脚本不能很好地界定转换范围，而且自定义的脚本使用范围小，不利于调试和推广；而另外一类是采用 XSLT 语言作为转换脚本语言。除了基于本模型实现的转换引擎外，比较突出的转换引擎还包括 IBM 公司的 XSLerator[6]和 TiBCO 公司的 XML Transform Tool[7]等。它们均以树模型作为转换模型，按照 XSLT 语言的语法来设计转换规则。尽管这些转换引擎为用户提供了非常友好的界面来设定转换规则，但是要求用户掌握一定的 XSLT 语法；而且一旦转换语义比较复杂，需要进行一些数据整合的操作(比如前面例子中提到的自然连接操作)，用户很难通过界面上的拖拽以及一些参数的设定在树模型上完成转换规则的设定。而关系树模型已经把数据整合的功能已经通过关系运算融入到转换语义中，用户无须了解 XSLT 即可设定转换规则。

结论 在这篇文章里，我们提出了一种称之为关系树模型的 XML 数据转换模型。该模型有效地把 XML 层次化的特点和成熟的关系理论结合在一起，把 XML 模式语言建模成由多个节点集组成的关系树，以此定义数据转换语义，并把关系运算引入到转换规则中，增加该模型的转换能力。我们将进一步优化 XML-Transformer 引擎的可视化界面，方便用户设定转换规则。

(下转第137页)

$(ID_C, ID_M, S')_{K_{CM}}$

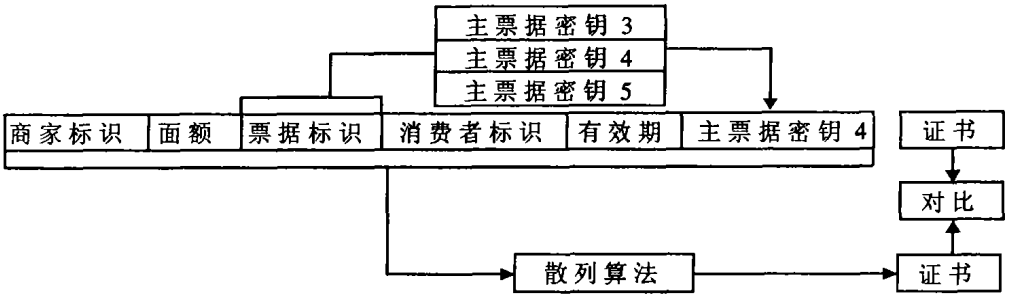

图4 票据合法性验证示意图

2.5 存款协议

当商家票据被消费者在有效期花费完毕后，则商家通过执行如下存款协议与经纪人进行资金转账：

(1)商家发送一个存款请求给经纪人，包括消费者标识 ID_C、商家的标识 ID_M、原始票据的标识 ID_S、面额 V_S 和票据有效期 E_S 等信息，其格式如下：

$(ID_C, ID_M, ID_S, V_S, E_S)_{K_{MB}}$

(2)经纪人通过遍历数据库来验证存款请求的合法性，若通过，则经纪人将与票据价值相当的资金转移到商家的账户上。

3 系统性能分析

3.1 可转移性

在本文提出的微支付方案中，票据是可以转移的。消费者可将商家票据转让给其他人，这可以通过商家更换票据中的消费者标识，并为其他消费者颁发一个新的等额合法票据来实现。

3.2 可分性

消费者在经纪人处购买的大额票据，可以与同一商家进行多次交易，并更新相应票据的面值。也就是说，商家在对票据扣除掉与商品或服务价格相当的数额后，为消费者提供一个具有新面值的票据和证书，从而实现了数字货币的可分性。

3.3 安全性

为了提高协议的安全性和机密性，防止票据被入侵者截获或交易信息被窃听者获取，本方案在交易双方之间建立了一个共享的对称密钥，并使用安全高效的密码算法来加密交易信息，为交易双方建立一个安全的通信通道，从而使得攻击者无法获取相关的敏感信息，也无法伪造合法的票据。同时，由于证书是单向散列函数运算的结果，它能够有效地防止对票据域进行任何形式的修改，票据域内容的任何变化将导致重新计算得到的证书与原证书不同。此外，只有知道主票据密钥的商家和可信的经纪人才能产生合法的票据，票据标识的唯一性又能防止消费者进行重复花费。

3.4 效率

在本方案中，由于商家票据可根据需要由经纪人实时产生，因而经纪人不需要保存大量的票据代码，存储开销大大减小；商家也不需要因生成票据而进行大量的计算，从而大大节省了计算开销；通过网络传输和验证生产许可证比传输大量的票据要经济得多，即减少了通信开销。同时，由于本方案完全没有采用公钥密码算法，因而协议的执行效率大大提高。

结论 本文设计了一个基于票据的新型高效微支付方案，它采用票据许可生产的模式，由商家授权经纪人生成商家票据，并由商家在线验证票据的合法性。该方案是一个离线的预付方案，它使用会话密钥对交易信息进行了加密，方案的安全性较高，且能够有效防止消费者的重复花费。同时，该方案支持数字货币的可分性和可转移性，适用于短期内消费者与同一商家的重复交易。与其它微支付方案相比，由于本方案完全没有使用公开密钥算法，且协议执行中所需保存的信息比较简单，因而系统的计算开销和存储开销大大减少。

参 考 文 献

1 Furche A, Wrightson G. SubScrip—An Efficient Protocol for Pay-Per-View Payment on the Internet. In: Proc. 5th Int. Comf. on Computer Communications and Networks (ICCCN'96), Rockville, MD, Oct. 1996. 16～19

2 Burstein J. An Implementation of MicroMint. M. Sc thesis. Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1998

3 Buttyán L. Removing the Financial Incentive to Cheat in Micropayment Schemes. IEEE Electronics Letters, 2000, 36(2): 132～133

4 Adachi N, Aoki S, Komano Y, et al. The Security Problems of Rivest and Shamir's PayWord Scheme. In: Proc. of the IEEE Int. Conf. on E-Commerce (CEC'03), 2003. 126～129

5 Yen S, Lee C, Ho L. PayFair: a prepaid Internet micropayment scheme promising customer fairness. In: IEEE Proc of Comput. Digit. Tech. 2001, 148(6): 207～213

(上接第117页)

参 考 文 献

1 Fernandez M, Tan W-C, Suciu D. SilkRoute: trading between relations and XML. Computer Networks, 2000, 33: 723～745

2 孟小峰. Web 信息集成技术研究. 计算机应用与软件, 2003, 20(11): 32～36

3 Deutsch A, Fernandez M, Florescu D, Levy A, Suciu D. A query language for XML. Computer Networks, 1999, 31: 1155～1169

4 Siberschatz A, Korth H F, Sudarshan S. Database System Concepts (Fourth Edition). McGraw-Hill companies, 2002

5 XSL transformation (XSLT) specification. http://www.w3.org/TR/XSLT/

6 XSLerator, IBM. http://www.alphaworks.ibm.com/tech/xslerator/

7 XML Transform, TIBCO. http://www.tibco.com/