, , 1–18 ()

A Data Model for XML Databases

VILAS WUWONGSE vw@cs.ait.ac.th
Computer Science & Information Management Program,
Asian Institute of Technology, Pathumtani 12120, Thailand

KIYOSHI AKAMA akama@cims.hokudai.ac.jp
Center for Information and Multimedia Studies
Hokkaido University, Sapporo 060, Japan

CHUTIPORN ANUTARIYA ca@cs.ait.ac.th
Computer Science & Information Management Program,
Asian Institute of Technology, Pathumtani 12120, Thailand

EKAWIT NANTAJEEWARAWAT ekawit@siit.tu.ac.th
Information Technology Program, Sirindhorn International Institute of Technology
P.O. Box 22, Thammasat Rangsit Post Office, Pathumthani 12120, Thailand

Abstract. In the proposed data model for XML databases, an XML element is directly represented as a *ground* (*variable-free*) *XML expression*—a generalization of an XML element by incorporation of variables for representation of implicit information and enhancement of its expressive power—while a collection of XML documents as a set of ground expressions, each describing an XML element in the documents. Axioms and relationships among elements in the collection as well as structural and integrity constraints are formalized as *XML clauses.* An *XML database*, consisting of: (i) a document collection (or an *extensional database*), (ii) a set of axioms and relationships (or an *intensional database*), (iii) a set of integrity constraints, is therefore modeled as an *XML declarative description* comprising a set of ground XML expressions and XML clauses. Its semantics is a set of ground XML expressions, which are explicitly described by the extensional database or implicitly derived from the database, based on the defined intensional database, and satisfy all the specified set of constraints. Thus, selective and complex queries, formulated as sets of XML clauses, about information satisfying specific criteria and possibly implicit in the database, become expressible and computable. The model thereby serves as an effective and well-founded XML database management framework with succinct representational and operational uniformity, reasoning capabilities as well as complex and deductive query supports.

Keywords: data model, XML document, XML expression, XML specialization system, and XML declarative description

1. Introduction

XML—a W3C recommendation, recently emerged as a standard for data representation and interchange among various Web applications—provides simple means for a more meaningful and understandable representation of Web contents. An XML document need only be *well-formed*, i.e., its tags be properly nested, but need not conform to a particular *DTD* or *Schema*. Hence, it is viewed to be a variation of *semi-structured data* [11]—data which may be varied and are not restricted to any particular schema. Management of semi-structured data by highly-structured modeling techniques, such as relational and object-oriented models, not only leads to a very complicated logical schema, but also demands much effort and frequent schema modifications, and thus obstructs the use of such approaches in modeling XML data. Consequently, development of an appropriate and efficient data model for XML documents has become an active research area with major current models based on *directed, edge-labeled graphs* [1, 5, 8, 11], *functional programming* [10], *hedge automaton* [16] and *Description Logic* [9]. However, these models alone do not have sufficient mechanisms to represent and manipulate all important characteristics and functionalities of XML data such as provision of supports for DTD/Schema validation, integrity constraints and query processing as well as well-defined semantics and an efficient reasoning mechanism. Their extension and integration of additional formalisms, which may complicate the models and make their understanding more difficult, are required to overcome this limitation. For example, by application of *first-order logic theory*, the graph model in [8] has incorporated a facility for expression of and reasoning with *path and type constraints*.

This paper develops *XML Declarative Description* (*XDD*) theory [4, 19] as a *data model for XML databases* with an aim to provide in its single formalism a simple yet expressive mechanism to succinctly and uniformly represent both explicit and implicit information, rules, relationships, structural and integrity constraints. A *description* in XDD is a set of ordinary XML elements, extended XML elements with variables, called *XML expressions*, and their constraints and relationships represented in terms of *XML clauses*. Its meaning does not yield only all the explicit information, represented by ordinary XML elements, but also includes all the implicit information, described by the XML expressions with variables and the XML clauses in the description. Moreover, the data model allows DTDs, Schemas and queries to be represented in terms of XDD descriptions, and also provides mechanisms for the verification of document validity and evaluation of queries [4, 3]. The developed model therefore presents a unified approach for manipulating, constraining, querying as well as reasoning about XML data.

Since an informal introduction to XDD theory has been given by [20], Sect. 2 defines formally the notion of XML expressions, regarded as the underlying data structure of the theory, and Sect. 3 formulates the theory. Sect. 4 presents an approach to its employment in XML database modeling, Sect. 5 reviews current, related works and compares them with the proposed one, Sect. 6 concludes and discusses further work.

2. XML Elements and XML Expressions

Ordinary XML elements are ground or variable-free. In order to express inherent implicit information and enhance its expressive power, the definition of an XML element will be formally extended by incorporation of variables, and then called an *XML expression*.

Let $\sum_X$ be an *XML expression alphabet* comprising the symbols in the seven sets defined in Table 1.

Definition 1. An **XML expression** on $\sum_X$ takes one of the forms:

Table 1. The alphabet $\sum_X$.

Sets	Set Elements	Beginning with	Specialization into
C	*Characters*	-	-
N	*Names*	-	-
V_N	*Name-variables (N-variables)*	$N:	Names in N
V_S	*String-variables (S-variables)*	$S:	Strings in C^*
V_P	*Attribute-value-pair-variables (P-variables)*	$P:	Sequences of attribute-value pairs
V_E	*XML-expression-variables (E-variables)*	$E:	Sequences of XML expressions
V_I	*Intermediate-expression-variables (I-variables)*	$I:	Parts of XML expressions

1. *evar*,
2. $<t\ \ a_1 = v_1\ \ldots\ a_m = v_m\ \ pvar_1\ \ldots\ pvar_k/>$,
3. $<t\ \ a_1 = v_1\ \ldots\ a_m = v_m\ \ pvar_1\ \ldots\ pvar_k> v_{m+1} </t>$,
4. $<t\ \ a_1 = v_1\ \ldots\ a_m = v_m\ \ pvar_1\ \ldots\ pvar_k> e_1\ \ldots\ e_n </t>$,
5. $<ivar> e_1\ \ldots\ e_n </ivar>$,

where – $evar \in V_E$, – $k, m, n \geq 0$, – $t, a_i \in (N \cup V_N)$,
– $pvar \in V_P$, – $v_i \in (C^* \cup V_S)$, – $ivar \in V_I$,
– e_i are XML expressions on $\sum_X$.

□

The order of the attribute-value pairs $a_1 = v_1\ \ldots\ a_m = v_m$ and the order of the P-variables $pvar_1\ \ldots\ pvar_k$ are immaterial, while the order of the expressions $e_1\ \ldots\ e_n$ is important. XML expressions with and without variable will be referred to as *non-ground XML expressions* and *ground XML expressions* (or XML elements), respectively. An expression of the second, the third or the fourth form is referred to as a *t-expression*, while that of the fifth form as an *ivar-expression*. A ground t-expression will also be called a t-element. An I-variable is employed to represent an XML expression when its structure, nesting pattern as well as list of attribute-value pairs are not fully known. For example, a given *ivar*-expression $<ivar>\ e_1\ \ldots\ e_n\ </ivar>$, where e_i are XML expressions, represents XML expressions which contain the sub-expression sequence $e_1 \ldots e_n$ to an arbitrary depth.

As an example of ground XML expressions, consider the element a of Fig. 1. Obviously, mappings between ordinary XML elements and ground XML expressions are straightforward. The expressions a' and a'' of Fig. 1 represent examples of non-ground XML expressions on $\sum_X$, which employ various types of variables for the representation of groups or classes of XML elements with some common structures,

```
<AirTrip from="Bangkok" to="London">     % The element a describes an AirTrip from Bang-
   <Path>                                % kok to London.
      <City>Bangkok</City>
      <City>Singapore</City>
      <City>London</City>
   </Path>
   <Price>650</Price>
</AirTrip>
```

a. A ground XML expression *a*.

```
<AirTrip from=$S:from to="London">   % a' represents a class of elements which encode
   $E:details                        % information about AirTrips originating from
</AirTrip>                           % anywhere and terminating in London.
```

b. A non-ground XML expression *a′*.

```
                                     % a'' represents another class of XML elements
                                     % which contain the sub-element <City>Singa-
<$I:element>                         % pore</City> to an arbitrary depth. Moreover,
   <City>Singapore</City>            % the exact structure, tag names, list of attribute-
</$I:element>                        % value pairs and the nesting pattern of the ele-
                                     % ments in the class are unknown and represented
                                     % by an I-variable $I:element.
```

c. A non-ground XML expression *a″*.

Figure 1. XML expression examples.

attributes or sub-elements. It will be seen in Example 1 that both a' and a'' can be specialized into the element a.

3. XDD: A Data Model for XML Databases

An *XML specialization system* and *XML declarative descriptions*, which serve as a data model for XML documents, will be formulated. First, the concept of an *XML specialization generation system* employed to define the XML specialization system in Definition 3 will be presented.

Definition 2. Let $\Delta_X = \langle \mathcal{A}_X, \mathcal{G}_X, \mathcal{C}_X, \nu_X \rangle$ be an **XML specialization generation system** on $\sum_X$, where

- $\mathcal{A}_X$ is the set of all XML expressions on $\sum_X$,
- $\mathcal{G}_X$ is the subset of $\mathcal{A}_X$ comprising all ground XML expressions on $\sum_X$,
- $\mathcal{C}_X$ is the set of all *basic specializations*, which is the union of the sets:
  - Variable Renaming: $(V_N \times V_N) \cup (V_S \times V_S) \cup (V_P \times V_P) \cup (V_E \times V_E) \cup (V_I \times V_I)$,

Table 2. The basic specialization operator ν_X.

Basic Specialization $c \in \mathcal{C}_X$	**Given** $a \in \mathcal{A}_X, \nu_x(c)(a)$ **is Obtained from** a **by**
1. Renaming $c = (v, u) \in (V_N \times V_N) \cup (V_S \times V_S) \cup (V_P \times V_P) \cup (V_E \times V_E) \cup (V_I \times V_I)$	Replacement of all occurrences of v in a by u.
2. Expansion	
• *P-variable* $c = (v_P, (u_N, u_S, v'_P)) \in (V_P \times (V_N \times V_S \times V_P))$	Simultaneous replacement of all occurrences of v_P in a by the sequence of the pair $u_N = u_S$ and the P-variable v'_P.
• *E-variable* $c = (v, (v_1, v_2)) \in V_E \times (V_E \times V_E)$	Simultaneous replacement of each occurrence of v in a by the sequence $v_1 v_2$.
3. Removal	
• *P-, E-variable* $c = (v, \varepsilon) \in (V_P \cup V_E) \times \{\varepsilon\}$	Removal of each occurrence of v in a.
• *I-variable* $c = (v, \varepsilon) \in V_I \times \{\varepsilon\}$, where ε denotes the null symbol	Removal of each occurrence of $<v>$ and of $</v>$ in a.
4. Instantiation	
• *N-variable* $c = (v, n) \in V_N \times N$	Simultaneous replacement of each occurrence of v in a by n
• *S-, E-variable* $c = (v, u) \in (V_S \times C^*) \cup (V_E \times \mathcal{A}_X)$	Simultaneous replacement of each occurrence of v in a by u.
• *I-variable* $c = (v_I, (u_N, u_P, u_E, w_E, v'_I)) \in V_I \times (V_N \times V_P \times V_E \times V_E \times V_I)$	Simultaneous replacement of each occurrence of the v_I-expression $<v_I>\ e_1 \ldots e_n\ </v_I>$ in a by the v_N-expression $<u_N\ u_P>\ u_E\ <v'_I> e_1 \ldots e_n </v'_I>\ w_E\ </u_N>$.

- P- and E-variable Expansion: $(V_P \times (V_N \times V_S \times V_P)) \cup (V_E \times (V_E \times V_E))$,
- P-, E- and I-variable Removal: $(V_P \times V_E \times V_I) \cup \{\varepsilon\}$, where ε denotes the null symbol,
- Variable Instantiation: $(V_N \times N) \cup (V_S \times C^*) \cup (V_E \times \mathcal{A}_X) \cup (V_I \times (V_N \times V_P \times V_E \times V_E \times V_I))$,

• $\nu_X : \mathcal{C}_X \rightarrow$ *partial_map*$(\mathcal{A}_X)$ is the *basic specialization operator*, which determines, for each basic specialization c in $\mathcal{C}_X$, the change of an XML expression in $\mathcal{A}_X$ caused by c and is defined in Table 2.

□

Fig. 2 shows that successive applications of the given basic specializations c_1, c_2, c_3 and c_4 to the expression a' of Fig. 1, by the operator ν_X, yield the element a.

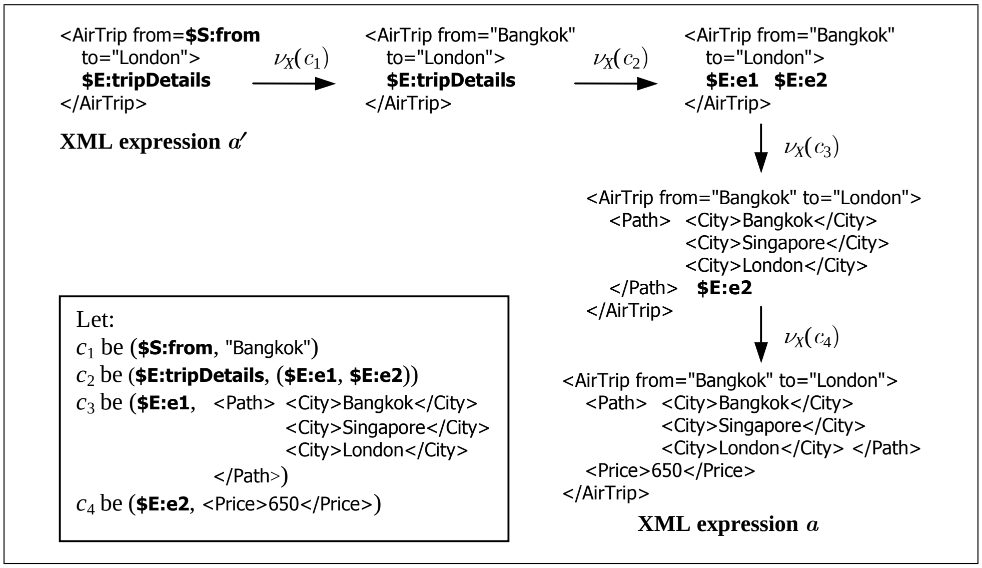

Figure 2. Successive application of $\nu_X(c_1), \ldots, \nu_X(c_4)$ to the expression a' of Fig. 1, yielding the element a of Fig. 1.

Definition 3. Based on Δ_X, let $\Gamma_X = \langle \mathcal{A}_X, \mathcal{G}_X, \mathcal{S}_X, \mu_X \rangle$ be an **XML specialization system** on $\sum_X$, where

- $\mathcal{S}_X = \mathcal{C}_X^*$ is the set of all sequences of zero or more basic specializations in $\mathcal{C}_X$ and their elements are called *specializations*,

- $\mu_X : \mathcal{S}_X \rightarrow partial\_map(\mathcal{A}_X)$ is the *specialization operator*, which determines, for each specialization s in $\mathcal{S}_X$, the change of an XML expression in $\mathcal{A}_X$ caused by s and is defined in terms of the basic specialization operator ν_X such that: For each $a \in \mathcal{A}_X$,

  – $\mu_X(\lambda)(a) = a$, where λ denotes the null sequence,

  – $\mu_X(c \cdot s)(a) = \mu_X(s)(\nu_X(c)(a))$, where $c \in \mathcal{C}_X$ and $s \in \mathcal{S}_X$.

□

Intuitively, a specialization s in $\mathcal{S}_X$ is a sequence of zero or more basic specializations in $\mathcal{C}_X$ and the specialization operator μ_X is defined in terms of the basic specialization operator ν_X such that for each $a \in \mathcal{S}_X$ and $s = (c_1 \ldots c_n) \in \mathcal{S}_X$, $\mu_X(s)(a)$ is obtained by successive applications of $\nu_X(c_1), \ldots, \nu_X(c_n)$ to a.

Example: With reference to Fig. 1 and 2, let $\theta = (c_1 c_2 c_3 c_4) \in \mathcal{S}_X$. By the operator μ_X, it follows that θ can specialize the expression a' into a, i.e., $a = \mu_X(\theta)(a')$

or by shorthand notation $a = a'\theta$. Similarly, the \$I:element-expression a'' can be specialized into the element a by some specialization in $\mathcal{S}_X$. Due to page limitation, such a specialization will not be given. □

After the definition of the XML specialization system Γ_X, which reflects the data structure and the specialization characteristics of XML expressions, definitions of *XML constraints*, *XML clauses*, *XDD descriptions* and the *declarative semantics* of an XDD description can now be given.

Definition 4. Let $\mathcal{K}_X$ be a set of *constraint predicates*. An **XML constraint** on Γ_X—useful for the definition of restrictions on XML expressions in $\mathcal{A}_X$—is a formula

$$q(a_1, \ldots, a_n), \tag{1}$$

where q is a constraint predicate in $\mathcal{K}_X$ and a_i an XML expression in $\mathcal{A}_X$. The truth and falsity of a *ground constraint* $q(g_1, \ldots, g_n)$, where $g_i \in \mathcal{G}_X$, is predetermined. Denote the set of all true ground constraints by $Tcon$. A specialization θ is applicable to a constraint $q(a_1, \ldots, a_n)$ if θ is applicable to $a_1, \ldots, a_n$. The result of $q(a_1, \ldots, a_n)\theta$ is the constraint $q(a_1\theta, \ldots, a_n\theta)$. □

Example: In order to define a constraint which restricts a less-than condition on numeric data, assume that LT is a constraint predicate in $\mathcal{K}_X$ and LT(a_1, a_2) is an XML constraint on Γ_X which will be true, iff a_1 and a_2 are XML elements of the forms <Num>n_1</Num> and <Num>n_2</Num>, respectively, where n_1 and n_2 are numbers and $n_1 < n_2$. For example, a given constraint LT(<Num>1</Num>, <Num>5 </Num>) is obviously a true ground constraint in $Tcon$, and a constraint LT(<Num>\$S:x</Num>, <Num>10</Num>) will be true, iff \$S:x is instantiated into a number less than 10.

In addition, for some a_1, a_2, $a_3 \in \mathcal{A}_X$, let Add$(a_1,\ a_2,\ a_3)$ be a constraint which will be true, iff a_1, a_2, a_3 are of the forms: <Num>n_1</Num>, <Addendum> n_2 </Addendum> and <Result> n_3 </Result>, respectively, where n_3 is the result of adding n_2 to n_1, i.e., $n_3 = n_1 + n_2$. □

Definition 5. An **XML clause** on Γ_X is a formula of the form:

$$H \leftarrow B_1, \ldots, B_n, \tag{2}$$

where $n \geq 0$, H is an XML expression in $\mathcal{A}_X$ and B_i an XML expression in $\mathcal{A}_X$ or a constraint on Γ_X. H is called the *head* and $\{B_1, \ldots, B_n\}$ the *body* of the clause. An **XML declarative description** or simply an **XDD description** on Γ_X is a (possibly infinite) set of XML clauses on Γ_X. □

Let C be an XML clause $(H \leftarrow B_1, \ldots, B_n)$. If $n = 0$, C is called an *XML unit clause*, if $n \geq 0$, an *XML non-unit clause*. When clear from the context, an XML unit clause $(H \leftarrow)$ will be simply represented by H. The head of C is denoted by $head(C)$, and the set of all XML expressions and constraints in the body of C by

$object(C)$ and $con(C)$, respectively. Let $body(C) = object(C) \cup con(C)$. A clause C' is an instance of C iff there is a specialization $\theta \in \mathcal{S}_X$ such that θ is applicable to $H, B_1, \ldots, B_n$ and $C' = C\theta = (H\theta \leftarrow B_1\theta, \ldots, B_n\theta)$. An XML clause C is a *ground XML clause*, iff C comprises only ground XML expressions and ground constraints.

Definition 6. Let P be an XDD description on Γ_X. Associated with P is the mapping T_P on $2^{\mathcal{G}_X}$:

$$T_P(X) = \{head(C\theta) \mid C \in P,\ \theta \in \mathcal{S}_X,\ C\theta \text{ is a ground clause}, \\ object(C\theta) \subset X,\ con(C\theta) \subset Tcon\} \tag{3}$$

Based on T_P, the declarative semantics of P—$\mathcal{M}(P)$—is defined by

$$\mathcal{M}(P) = \bigcup_{n=1}^{\infty} T_P^n(\emptyset) \tag{4}$$

where $T_P^1(\emptyset) = T_P(\emptyset)$ and $T_P^n(\emptyset) = T_P(T_P^{n-1}(\emptyset))$ for each $n > 1$. □

The next section presents an XDD approach to modeling XML databases and their queries, and at the same time demonstrates examples of XDD descriptions and their computation mechanisms.

4. Formalizing XML Databases and their Queries

4.1. XML Database Modeling

In XDD data model, an ordinary XML element is represented directly by a ground XML expression in $\mathcal{G}_X$, while a class of XML elements with similar components and structures, is represented by an XML expression with variables in $\mathcal{A}_X$, as illustrated in Fig. 1.

An XML document, comprising a sequence of n XML elements, is formalized as an XDD description, consisting of n ground XML unit clauses, each of which describes its corresponding XML element in the document.

An extensional XML database, comprising m documents $D_1, \ldots, D_m$, is then represented as an XDD description $XDB_E = P_1 \cup \ldots \cup P_m$, where for $1 \leq i \leq m$, P_i is an XDD description, representing the document D_i and comprising only unit clauses.

An intensional XML database is formalized as an XDD description XDB_I comprising a set of XML non-unit clauses, defining axioms, relationships or knowledge deducible from the database.

A set of structural and integrity constraints on an XML database is modeled as an XDD description XDB_C comprising a set of XML non-unit clauses, each of which describes a particular constraint on the database.

C_{e1}:
```
<Flight number="TG916" airline="TG">
    <Origin>Bangkok</Origin>
    <Destination>London</Destination>
    <Price>750</Price>
</Flight>
```

C_{e2}:
```
<Flight number="SQ61" airline="SQ">
    <Origin>Bangkok</Origin>
    <Destination>Singapore</Destination>
    <Price>150</Price>
</Flight>
```

C_{e3}:
```
<Flight number="SQ320" airline="SQ">
    <Origin>Singapore</Origin>
    <Destination>London</Destination>
    <Price>500</Price>
</Flight>
```

Figure 3. XML extensional database $XDB_E = \{C_{e1}, C_{e2}, C_{e3}\}$.

An XML database, consisting of the three parts: an extensional database XDB_E, a set of integrity constraints XDB_C and an intensional database XDB_I, is modeled as an XDD description $XDB = XDB_E \cup XDB_I \cup XDB_C$.

The semantics of XDB—$\mathcal{M}(XDB)$—yields all directly represented XML elements in the database, i.e., those expressed by unit clauses, together with all derived ones, possibly constrained. Thus, in addition to simple queries, only based on text/pattern matching, selective, complex queries regarding derived information may be posed. Moreover, by incorporation of the notion of *set-aggregation* [4, 3], the proposed approach readily allows formulation and evaluation of group-by and aggregate queries. In addition, the approach provides simple means to restrict XML data to those which satisfy a given DTD or Schema. They are materialized by direct translation of a DTD/Schema into a corresponding XDD description for the checking of the validity of an XML document with respect to the DTD/Schema. The theoretical details of such formalization are given in [4, 3].

Example: This example illustrates modeling of an XML database as an XDD description $XDB = XDB_E \cup XDB_I \cup XDB_C$, where

- $XDB_E = \{C_{e1}, C_{e2}, C_{e3}\}$ (Fig. 3) represents an extensional database containing flight information. For the sake of simplicity, the database stores, for a given flight, its flight number, airline code, origin, destination and price.

- $XDB_I = \{C_{i1}, C_{i2}\}$ (Fig. 4) formalizes an intensional database, which assembles air-trip information from the basic flight information. This is an example of complex, recursive relationships which can be simply expressed in the proposed approach. Moreover, additional conditions and rules can be included to avoid cyclic and to determine more appropriate routes. For example, one can

  – Specify a minimum waiting time between two connecting flights as well as find routes which are shortest in time or cheapest in price, after information of departure and arrival data/time of each flight has been provided in the database.

  – Retrieve all possible air trips which use the same airline for the duration of the trips.

```
Ci1: <AirTrip from=$S:X to=$S:Y>
        <Path>
            <City takingFlight=$S:flight>$S:X</City>
            <City>$S:Y</City>
        </Path>
        <Price>$S:price</Price>
        <TotalFlights>1</TotalFlights>
     </AirTrip>
        ← <Flight number=$S:flight $P:attr>
              <Origin>$S:X</Origin>
              <Destination>$S:Y</Destination>
              <Price>$S:price</Price>
          </Flight>.
```

% C_{i1} defines that if there is a
% direct Flight from $S:X to $S:Y,
% then there is also an AirTrip
% without an intermediate stop
% from $S:X to $S:Y. Its route is
% $S:X - $S:Y via $S:flight, its
% Price is equivalent to that of the
% Flight, and its TotalFlights (the
% total number of flights taken) is
% one.

```
Ci2: <AirTrip from=$S:X to=$S:Z>
        <Path>
            <City takingFlight=$S:flight>$S:X</City>
            $E:path
        </Path>
        <Price>$S:totalPrice</Price>
        <TotalFlights>$S:totalNum</TotalFlights>
     </AirTrip>
        ← <Flight number=$S:flight $P:attr>
              <Origin>$S:X</Origin>
              <Destination>$S:Y</Destination>
              <Price>$S:fprice</Price>
          </Flight>,
          <AirTrip from=$S:Y to=$S:Z>
              <Path>$E:path</Path>
              <Price>$S:tprice</Price>
              <TotalFlights>$S:num</TotalFlights>
          </AirTrip>,
          Add(<Num>$S:fprice</Num>,
              <Addendum>$S:tprice</Addendum>,
              <Result>$S:totalPrice</Result>),
          Add(<Num>$S:num</Num>,
              <Addendum>1</Addendum>,
              <Result>$S:totalNum</Result>).
```

% C_{i2} specifies that an AirTrip from
% $S:X to $S:Z can be arranged
% based on a direct Flight from
% $S:X to $S:Y and an intermedi-
% ate AirTrip from $S:Y to $S:Z.
% The trip's route is $S:X –
% $E:path, i.e., starting from $S:X
% and then following the path of
% the intermediate trip, repre-
% sented by $E:path. Its Price is
% the addition of that of the direct
% Flight, represented by $S:fprice,
% and that of the intermediate trip,
% represented by $S:tprice. Its
% TotalFlights is that of the inter-
% mediate trip, represented by
% $S:num, plus one.
%

Figure 4. XML intensional database $XDB_I = \{C_{i1}, C_{i2}\}$.

- $XDB_C = \{C_{c1}, C_{c2}\}$ (Fig. 5) models a set of integrity constraints, which restrict the price and the origin/destination of each flight to certain conditions. If the database violates a particular constraint, a ConstraintViolation-element, describing the type of constraint and the details of elements in the database causing the violation, will be derived. On the other hand, if the database satisfies all the defined constraints, the meaning of the database will not include a ConstraintViolation-element. In addition to the given constraints, other kinds of integrity constraints can similarly be defined; for instance, to restrict that

  - The contents of Price-elements must be integer—a kind of *type constraints.*
  - The values of the number-attributes belonging to Flight-elements must be unique—a *unique key constraint.*

```
Cc1: <ConstraintViolation type="PriceConstraint">
        <Flight number=$S:flight $P:attr>
            <Price>$S:price</Price>
        </Flight>.
     </ConstraintViolation>
        ← <Flight number=$S:flight $P:attr>
              $E:FlightDetails
              <Price>$S:price</Price>
          </Flight>,
          GT(<Num>$S:price</Num>,<Num>1500</Num>).

% Cc1 indicates that if there is
% a Flight with price higher
% than 1500 in the database,
% a ConstraintViolation-
% element reporting such a
% violation will be derived.

Cc2: <ConstraintViolation type="OriginDestinationConstraint">
        <Flight number=$S:flight $P:attr>
            <Origin>$S:origin</Origin>
            <Destination>$S:origin</Destination>
        </Flight>
     </ConstraintViolation>
        ← <Flight number=$S:flight $P:attr>
              <Origin>$S:origin</Origin>
              <Destination>$S:origin</Destination>
              $E:FlightDetails
          </Flight>.

% Cc2 indicates that the data-
% base violates an OriginDes-
% tinationConstraint, if there
% exists a flight the origin
% and destination of which
% are the same.
```

Figure 5. A set of integrity constraints $XDB_C = \{C_{c1}, C_{c2}\}$.

– Elements of the database must conform to a particular DTD or Schema—a *DTD/Schema constraint*.

The database's meaning, $\mathcal{M}(XDB)$, is a set of XML elements including those elements in the extensional database, i.e., the elements C_{e1}, C_{e2}, C_{e3}, together with those which are deducible from the database and satisfy the constraints. □

Although XML merely provides data encoding syntax and lacks a mechanism to define global, fixed semantics for a particular element, associated with each document, DTD and Schema are the user's intended meaning which denotes certain objects in some application domain. Based on a particular XML-based application markup language (or simply an XML application), conforming XML documents yield a common understanding of data objects in that domain, hence both syntactic and semantic interoperability among applications are enabled. As a result, much effort has been devoted to definitions of standard XML applications for each domain of interest, such as *XMI*, *SMIL*, *WML* and *MathML*. Besides direct representation of documents in these languages, XDD readily provides well-defined facilities for the enhancement of their expressive power by specifications of relationships and constraints as well as for the determination of their semantics.

Next, an approach to formulation and evaluation of queries will be outlined. [4, 3] discuss details of such an issue and show that the proposed approach can express and compute all essential XML query operations, e.g., selection, joining, transformation, aggregation, regular path expression and recursion.

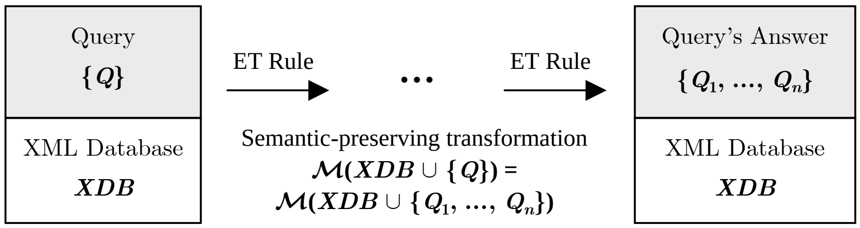

Figure 6. Query evaluation mechanism by Equivalent Transformation.

4.2. Query Formulation and Evalaution

A *query* is formalized as an XML non-unit clause, called a query clause. For a *query clause* Q, $head(Q)$ describes the structure of the resulting XML expressions and $body(Q)$ specifies the pattern of the XML elements to be selected as well as the query's filtering conditions.

Each query will be executed on some specified XML database and will return as its answer a set of XML elements, explicit in or derivable from the database and satisfying all of its conditions. Intuitively, given a database XDB and a query Q of the form $(H \leftarrow B_1, \ldots, B_n)$, an XML element $g \in \mathcal{G}_X$ is an answer to the query Q with respect to the database XDB, iff there exists $\theta \in \mathcal{S}_X$ such that $g = H\theta$ (i.e., H can be specialized into g by θ) and the meaning of $XDB \cup \{Q\}$ includes g. Thus, the answer to the query Q is the set $\{H\theta \mid H\theta \in \mathcal{M}(XDB \cup \{Q\}), \theta \in \mathcal{S}_X\}$.

Consider next an algorithm for the computation of an answer to a query (Fig. 6): Given a database XDB and a query Q, Q is evaluated by means of the *Equivalent Transformation* (*ET*) paradigm [2]—a new, flexible, efficient computational framework—by successive transformation of the description $XDB \cup \{Q\}$ into a simpler but equivalent description, from which the answers can be obtained readily and directly. In brief, $XDB \cup \{Q\}$ will be successively transformed until it becomes the description $XDB \cup \{Q_1, \ldots, Q_n\}$, $n \geq 0$, where the Q_i are ground XML unit clauses; the set $\{Q_1, \ldots, Q_n\}$ is the answer to the query Q.

The correctness of the answers obtained by application of such a mechanism relies solely on the equivalence of all declarative descriptions in the transformation process. In this paper, only unfolding transformation—the only ET rule used in SLD refutation, an inference mechanism employed by most Prolog implementations—will be used. Other kinds of semantics-preserving transformations specific for XML data structure, such as manipulation of sub-elements of an XML element, scan also be applied, especially to improve computational efficiency. Research on a generation of efficient ET rules for XDD is in progress.

Example: A query, which finds a trip from Bangkok to London with the price less than 700 and the number of taken flights less than three, can be formulated as the clause Q of Fig. 7. The AirTrip-expression in its body encodes the information of AirTrip from Bangkok to London. Its route is represented by the E-variable

```
Q: <MyTrip from="Bangkok" to="London">
       $E:path
       <Price>$S:price</Price>
       <TotalFlights>$S:totalFlights</TotalFlights>
   </MyTrip>
         ← <AirTrip from="Bangkok" to="London">
               $E:path
               <Price>$S:price</Price>
               <TotalFlights>$S:totalFlights</TotalFlights>
           </AirTrip>,
           LT(<Num>$S:price</Num>, <Num>700</Num>),
           LT(<Num>$S:totalFlights</Num>, <Num>3</Num>).

Q': <AirTrip from="Bangkok" to="London">
       <Path>
          <City takingFlight="SQ61">Bangkok</City>
          <City takingFlight="SQ320">Singapore</City>
          <City>London</City>
       </Path>
       <Price>650</Price>
       <NumFlightsTaken>2</NumFlightsTaken>
   </AirTrip>   ←   .
```

Figure 7. A query Q and its answer, Q', obtained by means of the unfolding rule.

$E:path, its price by the S-variable $S:price and its total number of flights taken by the S-variable $S:totalFlights. The two LT (LessThan) constraints restrict that the trip's price must be less than 700 and the number of flights taken less than three, respectively. The head of Q defines that the query returns MyTrip-elements encoding information of the trips which satisfy all the specified criteria.

By means of the unfolding rule, the description $XDB \cup \{Q\}$ can be successively transformed into the description $XDB \cup \{Q'\}$, where Q' is given by Fig. 7. Since only the unfolding rule, which always preserves the equivalence of descriptions, is used in each transformation step, it follows that $\mathcal{M}(XDB \cup \{Q\}) = \mathcal{M}(XDB \cup \{Q'\})$ and the obtained answer, i.e., Q', is guaranteed to be correct.

This example shows that although information about air trips is not explicitly specified in the database, it can be uncovered through the clauses C_{i1} and C_{i2}.

$\square$

5. Related Work

Three important approaches to the modeling of semi-structured/SGML data prior to 1995, i.e., *traditional information retrieval*, *relational model* and *object-oriented approaches*, have been reviewed in [17]. A review of more recent work and their comparisons to the proposed one follow.

5.1. Graphs

In *graph* models [1, 5, 8, 11], a collection of XML documents is represented by a *directed, edge-labeled graph*. Although the graph model provides an effective and straightforward approach to representing XML data, it encounters difficulties with restricting XML data to a given DTD or a Schema [5]. The model requires substantial extension to resolve this difficulty. For example, by application of first-order logic theory, the approach [8] has incorporated an ability to express *path and type constraints* for the specification of the structure of XML data; the integration of these *two different formalisms* also yields an ability to reason about path constraints. However, the complex notions of *interpretation* and *implication* in first-order logic tend to complicate the syntax and semantics of path constraints and complicate their understanding.

5.2. Functional Programming

[10] has developed a *functional programming approach* to modeling XML documents and formalizing operations on them by introducing, as its underlying data structure, a user-defined typed feature term, called a *node*. Based on this model, an algebra for XML queries, expressed in terms of *list comprehensions* in the functional programming paradigm, has also been constructed. Using list comprehensions, one can express various kinds of XML query operations such as navigation, nesting, grouping and joins. However, this approach has considerable limitations as it cannot model a DTD or a Schema, hence a mechanism to verify document conformance is not readily devised.

5.3. Hedge Automaton

By means of *hedge automaton* theory [15], [16] has constructed an approach to the formalization of XML documents and their DTDs. A *hedge* is an ordered sequence of trees or, in XML terminology, a sequence of XML elements. A document is therefore represented by a hedge and a set of documents conforming to a DTD by a *regular hedge language* (*RHL*), expressible by a *regular hedge expression* (*RHE*) or a *regular hedge grammar* (*RHG*). The approach employs a *hedge automaton* to determine whether a document conforms to a given RHG (representing some particular DTD) or not. However, the formalism merely provides means for representation and restriction of element structuring, while lacking an ability to deal with attributes associated with an element such as ID and IDREF(S) attributes, used for restricting uniqueness and referential constraints, respectively. Moreover, other kinds of integrity constraints cannot be imposed.

5.4. Datalog

Since *Datalog* [14, 18] and some of its extensions, e.g., LDL and RelationLog, provide inexpressive flat structures and cannot directly support the complex, hierarchical structure common in XML syntax, they divulge a significant problem in modeling and representation of XML data. An XML element must be translated and expressed in Datalog in terms of its permitted representations only, e.g., as a set of atomic formulae with simple-structure terms. Identical XML elements may have several corresponding representations depending on the employed translational scheme and the definition of a rigid relational schema. Moreover, the difficulties, which are encountered during application of the relational approach to modeling XML data, remain inherent in deductive database approaches. In addition, it is difficult to express a query whenever the document schema, element tag name or the nesting level, at which the required element occurs, are unknown. Hence, such an approach trades the structural information and expressive power underlying XML documents for an application of an existing theory.

Because Datalog is founded on first-order logic theory, formulation as well as evaluation of queries, which involve certain higher-order features such as a variable ranging over relation (predicate) names, are not yet supported.

5.5. Description Logic

By means of *Description Logic* (*DL*), [9] has developed a formalism for representation of and reasoning about XML DTDs, which exploits the DL's reasoning capability to verify conformance of a document to a DTD, and to check for two given DTDs D_1 and D_2, whether D_1 is *included* in, *equivalent* to or *disjoint* from D_2. It models a document as a tree and a DTD as a set of DL-assertions. The verification of conformance of a document to a DTD is performed by *model checking*, while the determination of inclusion, equivalence and disjointness between two DTDs is processed by *concept subsumption checking*.

A query is formulated as a DTD and its answer is the set of documents conforming to such a DTD. Hence, computation of a query is defined by means of DTD conformance checking. Although the approach claims that its mechanism provides efficient query evaluation, its expressiveness is insufficient, because many essential query operations, such as extraction, selection, document transformation and joining of data from different documents, cannot be represented in terms of DTDs. Therefore, the DL approach does not readily provide complete means for modeling and management of XML data.

5.6. Comparisons

Compared with other models, XDD provides a more direct and succinct insight into the computation of and reasoning with XML databases. It naturally fuses XML syntax and its semantics, in order to provide effective means for modeling of XML data, their interrelationships and constraints. It has sufficient expressive power to represent complex forms of information and to infer information implicit in XML databases.

XDD is not a logic programming language, although its clauses and descriptions have similar forms to Datalog clauses and Datalog programs, respectively. In contrast to Datalog, XDD has been formally defined without such complicated notions as interpretations and models. Moreover, it has a higher-order syntax, because it allows complex, nesting structured objects, i.e., XML expressions, to be directly represented and manipulated without decomposition into sets of flat structured objects.

As a summary, Table 3 deliberately compares these important approaches to XML data management.

6. Conclusions

The XDD approach to modeling and manipulation of XML databases is highly general and expressive, because it is the first unified theory, which not only provides a straightforward and uniform representation of XML documents, XML databases and their relationships, but also facilitates simple means of modeling DTDs/Schemas as well as other kinds of constraints. It allows users to precisely formulate queries which describe their informational needs and obtain query results, which may be implicit in the database, and hence result in a substantial improvement in the precision of retrievals. Moreover, it readily supplies sufficient mechanisms for the devise of important semantic query optimization techniques which exploit knowledge on the DTDs/Schemas. For example, given a database, its DTD and a query, an empty query result can be immediately returned without searching the entire database, if the specified query pattern does not conform to the DTD. Discovery of other kinds of optimization rules and development of an indexing technique for XDD data model are part of ongoing research.

Development of a prototype *XDD system*—a Web-based XML engine, founded on XDD and the ET computational paradigm, available at http://kr.cs.ait.ac.th/xdd—and its preliminary tests on several XML applications reveal the approach's viability and potential in real applications.

As elaborated in [20], XDD can succinctly represent and reason about metadata, Web resources, ontologies and axioms expressed in terms of *RDF* [13], *RDF Schema* [7] or *DAML+OIL* [12]. Moreover, it can enhance these languages' expressiveness by additionally allowing representation of implicit complex information items, rules, conditional relationships, integrity constraints and arbitrary axioms, and thus can provide a solid foundation for modeling the *Semantic Web* [6]—a vision introduced for the next-generation Web—which will evolve today's Web from being merely a vast unstructured data repository into a rich and meaningful knowledge base and allow not only human-human, but also machine-machine communication. Thus, with this well-established mechanism for expression of machine-comprehensible information, communication and interoperation among intelligent Web services as well as automated software agents will also become possible.

Table 3. Comparison of approaches to XML data management.

Characteristics/ Functionalities	***Approaches***					
	Graph	**Hedge Automaton**	**Functional Programming**	**Datalog**	**Application of DL**	**XDD**
XML data representation	Directed, edge-labeled graphs	Hedges	Typed feature terms	Atomic formulas (relational tuples)	Trees	XML expressions
DTD representation	Unable to represent DTDs	Regular Hedge Grammar (RHG)	Unable to represent DTDs	Datalog programs	DL-assertions	XDD descriptions
DTD validation support	No	Yes, by using hedge automaton. However, it cannot validate attribute-list declarations.	No	Yes, by means of resolution priciple (logical inference)	Yes, by means of model checking in Description Logic	Yes, by means of equivalent transformation of XDD description
Integrity constraint support	Yes, but the support is only for path and type constraints	No	No	Yes, by means of built-in predicates	Yes, but limited to only constraints on document structures	Yes, by means of constraints in the XDD theory
Query processing support	Yes	Yes	Yes, by means of list comprehension evaluation	Yes, but rather difficult and involving numerous joins when dealing with complicated structured data	Yes, but not complete since a query is formalized as a DTD	Yes, by means of equivalent transformation of XDD description
Possession of reasoning capability	Yes, but limited to reasoning about path and type constraints	Yes, by means of the hedge automaton	Yes, by means of list comprehension evaluation	Yes	Yes, but limited to only reasoning about document structures	Yes
Regular path expression support	Yes	Yes	Yes	Indirect support by using variables and recursions in Datalog clauses	Yes	Indirect support by using variables and recursions in XML clauses
Succinct representation and operation	No	No	No	No	No	Yes, since no translation is needed

Acknowledgments

This work was supported in part by the Thailand Research Fund.

References

1. Abiteboul, S, Buneman, P. and Suciu, D., *Data on the Web: From Relations to Semistructured Data and XML*, CA: Morgan Kaufmann Publishers, 2000.
2. Akama, K., Shimitsu, T. and Miyamoto, E., "Solving Problems by Equivalent Transformation of Declarative Programs," *J. Japanese Society of Artificial Intelligence (JSAI)*, Vol. 13, No. 6, 1998, pp. 944–952 (in Japanese).
3. Anutariya, C., "XML Declarative Description," *Doctoral Dissertation*, Computer Science and Information Management Program, Asian Institute of Technology, Thailand, 2001.
4. Anutariya, C., Wuwongse, V., Nantajeewarawat, E. and Akama, K., "Towards a Foundation for XML Document Databases," In K. Bauknecht, S.K. Madria and G. Pernul (Eds.), *Electronic Commerce and Web Technologies*, LNCS 1875, Berlin: Springer-Verlag, October 2000, pp. 324–333.
5. Beech, D., Malhotra, A. and Rys, M., "A Formal Data Model and Algebra for XML," *W3C XML Query Working Group Note*, September 1999.
6. Berners-Lee, T., *Weaving the Web*, San Francisco, CA: Harper, 1999.
7. Brickley, D. and Guha, R.V., "Resource Description Framework (RDF) Schema Specification 1.0.," *W3C Candidate Recommendation*, March 2000, http://www.w3.org/TR/rdf-schema.
8. Buneman, P., Fan, W. and Weinstein, S., "Interaction between Path and Type Constraints," In *Proc. 18th ACM Symposium on Principles of Database Systems*, 1999.
9. Calvanese, D., De Giacomo G. and Lenzerini, M., "Representing and Reasoning on XML Documents: A Description Logic Approach," *J. Logic and Computation*, Vol. 9, No. 3, 1999, pp. 295–318.
10. Fernández, M., Siméon, J., Suciu, D. and Wadler, P., "A Data Model and Algebra for XML Query," Draft Manuscript, 1999.
11. Goldman, R., McHugh, J. and Widom, J., "From Semistructured Data to XML: Migrating the Lore Data Model and Query Language," In *Proc. 2nd Int. Workshop on the Web and Databases (WebDB '99)*, Philadelphia, Pennsylvania, 1999.
12. Hendler, J. and McGuinness, D.L., "The DARPA Agent Markup Language," *IEEE Intelligent Systems*, Vol. 15, No. 6, 2000, pp. 72–73.
13. Lassila, O. and Swick, R.R., "Resource Description Framework (RDF) Model and Syntax Specification," *W3C Recommendation*, February 1999, http://www.w3.org/TR/REC-rdf-syntax.
14. Liu, M., "Deductive Database Languages: Problems and Solutions," *ACM Computing Surveys*, Vol. 31, No. 1, 1999, pp. 27–62.
15. Murata, M., "Hedge Automata: A Formal Model for XML Schemata," *Technical Report*, Fuji Xerox Information Systems, 1995.
16. Murata, M., "Transformation of Documents and Schemas by Patterns and Contextual Conditions," In C.K. Nicholas and D. Wood (Eds.), *Principles of Document Processing*, LNCS 1293, Berlin: Springer-Verlag, 1997, pp. 153–159.
17. Sacks-Davis, R., Arnold-Moore, T. and Zobel, J., "Database Systems for Structured Documents," *IEICE Transactions on Information and System*, Vol. E78-D, No. 11, 1995, pp. 1335–1341.
18. Ullman, J.D., *Principles of Database and Knowledge-Base Systems*, MD: Computer Science Press, 1998.
19. Wuwongse, V., Akama, K., Anutariya, C. and Nantajeewarawat, E., "A Data Model for XML Databases," In N. Zhong, Y. Yao, J. Liu and S. Ohsuga (Eds.), *Web Intelligence: Research and Development*, LNAI 2198, Berlin: Springer-Verlag, 2001, pp. 237–246.
20. Wuwongse, V., Anutariya, C., Akama, K. and Nantajeewarawat, E., "XML Declarative Description (XDD): A Language for the Semantic Web," *IEEE Intelligent Systems*, Vol. 16, No. 3, 2001, pp. 54–65.