QUERY FORMULATION AND EVALUATION FOR XML DATABASES

Kiyoshi Akama
Center for Information and Multimedia Studies,
Hokkaido Unversity, Sapporo 060, Japan
akama@cims.hokudai.ac.jp

Chutiporn Anutariya
Department of Telematics, Norwegian University of Science and Technology,
N-7491 Trondheim, Norway
Chutiporn.Anutariya@item.ntnu.no

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

Ekawit Nantajeewarawat
Information Technology Program, Sirindhorn International Institute of Technology,
Thammasat University, Pathumtani 12120, Thailand
ekawit@siit.tu.ac.th

Abstract The proposed approach to XML query formulation and evaluation developed by means of *XML Declarative Description* (*XDD*) theory formalizes a query as an *XDD description* comprising one or more *XML clauses* the syntax of which can be subdivided into the three specifications: pattern of XML elements to be selected, the query's selection criteria and the structure of the resulting elements. It supports formulation of essential functionality requirements for an XML query language such as selection and extraction, combination, transformation, closure and nested queries. Evaluation of a query on a specified XML database is carried out by employment of *Equivalent Transformation* paradigm. Moreover, since XDD theory provides a simple mechanism for representation of knowledge and relationships among elements in a

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35616-7_23
W. Cellary et al. (eds.), *Internet Technologies, Applications and Societal Impact*

database, queries about derivable information, contained implicitly in the database are expressible and computable.

Keywords: XML databases, query formulation and evaluation, XML Declarative Descriptions, Equivalent Transformation.

1. Introduction

The advent of XML [9] as a standard for representation and interchange of data on the Web has provided simple means for the publication and distribution of both *human-readable* and *machine-understandable* data. In order to achieve effective manipulation on a large collection of XML documents, there arises an essential need for an XML standard query language, which allows users as well as agents to query, retrieve, transform and construct XML data that precisely meet their requirements. Several certain attempts have been made at the development of specifications [10, 17], algebra [7, 13, 14] as well as syntax [1, 8, 11, 18] for XML query languages. However, so far there exists no consensus on such a standard. Moreover, query languages such as *XQuery* [8, 12] *XML-QL* [11], *XQL* [18] and *Lorel* [1] merely aim at construction of user-friendly languages rather than formalization of effective computational mechanisms to process a query.

Based on the functionality requirements for an XML query language, identified by [10, 17], a declarative description approach is presented for both formulation and evaluation of queries on XML databases by employment of *XML Declarative Description* (*XDD*) theory [4, 6, 20, 21]. In the developed approach, an XML database is formalized as an *XDD description* which comprises a set of XML elements, representing a collection of XML elements/documents in the database, and a set of *XML clauses*, describing relationships, axioms as well as some other derivable information. A query is also modeled as an XDD description, which not only facilitates the pattern-matching or selective retrieval of XML data but also supports reasoning capability. Every query is executed on some specified XML database and will return as its result a set of XML elements, derived from the database and satisfying the query conditions. Evaluation of a query is defined by means of *Equivalent Transformation* (*ET*) [3]—a new, flexible, efficient computational model which is based on successive equivalent transformations of the query into the answer.

Sect. 2 summarizes the proposed data model for XML databases, Sects. 3 and 4 present approaches to formulation and evaluation of queries, respectively, Sect. 5 reviews current, related work, and Sect. 6 concludes.

2. A Data Model for XML Databases

2.1. XDD: An Informal Review

XML Declarative Description (*XDD*) [4, 6, 20, 21]—a data model for XML databases, developed by employment of *Declarative Description* theory [2]—extends ordinary, well-formed XML elements by incorporation of variables for an enhancement of expressive power and representation of implicit information into so-called *XML expressions*. Ordinary XML elements—XML expressions without variable—are called *ground XML expressions*. Every component of an XML expression can contain variables, e.g., its expression or a sequence of sub-expressions (*E-variables*), tag names or attribute names (*N-variables*), strings or literal contents (*S-variables*), pairs of attributes and values (*P-variables*) and some partial structures (*I-variables*). Every variable is prefixed by '$\$T$:', where T denotes its type; for example, $S:value and $E:expression are S- and E-variables, which can be specialized into a string and a sequence of XML expressions, respectively.

An *XDD description* is a set of *XML clauses* of the form:

$$H \leftarrow B_1, \ldots, B_n,$$

where $n \geq 0$, H is an XML expression, and B_i is an XML expression, a *constraint* or a *set-aggregation*. The XML expression H is called the *head*, the set $\{B_1, \ldots, B_n\}$ the *body* of the clause. When the body is empty, such a clause is referred to as an *XML unit clause*. When clear from the context, a unit clause ($H \leftarrow$) will be represented simply by H; hence an XML element or document can be mapped directly onto a *ground XML unit clause*.

A *constraint*—useful for defining a restriction on XML expressions, their components or their sets—is a formula of the form $q(a_1, \ldots, a_n)$, where $n > 0$, q is a *constraint predicate*, and a_i is an XML expression or a set of XML expressions. The truth or falsity of a *ground constraint* $q(g_1, ..., g_n)$, where g_i is a ground XML expression or a set of ground XML expressions, is predetermined. Denote the set of all true ground constraints by $Tcon$.

A *set-aggregation*—a concept employed in the description of complex queries/ operations on sets of XML expressions—has the form:

<xdd:SetAggregation>
  <xdd:Set> S </xdd:Set>
  <xdd:Constructor> E_C </xdd:Constructor>
  <xdd:Pattern> E_P </xdd:Pattern>
</xdd:SetAggregation>,

where S denotes a set of XML elements, E_C and E_P are XML expressions which specify that for each XML element, matching the pattern represented by E_P, an XML element represented by E_C will be constructed and included in the set S.

Intuitively, given an XDD description P, its meaning—denoted by $\mathcal{M}(P)$—is the set of all XML elements which are directly described by and are derivable from the unit and non-unit clauses in P, respectively.

2.2. XML Database Modeling

In the proposed data model for XML databases [4, 6, 20], ordinary XML elements can be mapped directly onto ground XML expressions, while a collection of XML documents, consisting of sequences of XML elements, is represented as a set of ground expressions, each describing an element in the documents. Integrity and structural constraints imposed on elements in the documents as well as rules, axioms and conditional relationships among them are formalized as a corresponding set of XML clauses. Thus, an XML database is then represented as an XDD description which is the union of a set of ground XML expressions, representing a collection of XML elements/documents in the database, and a set of XML clauses, describing their constraints and relationships. The database's semantics is the set of XML elements, which are explicitly represented in the document collection or implicitly derived from the specified set of relationships and satisfy all the defined constraints.

Example 1 Let XDB be an XDD description which represents a database of a human resource office and comprises the elements $C_{e1} - C_{e4}$ and the clauses $C_{r1} - C_{r2}$ of Fig. 1. The former describes a collection of XML elements in the database and the latter models a supervisor-subordinate relationship among Employee objects of the database. The representation of a relationship in terms of XML clauses offers a more compact form and provides an easier way of information modification than explicit enumeration of such information. □

3. Query Formulation

A query about information in an XML database is formalized as an XDD description, comprising one or more XML clauses. For each query clause, its head describes the structure of the resulting XML elements, the set of XML expressions in its body represents some particular XML documents or XML elements to be selected, the set of XML constraints describes selection condition, and the set of set-aggregations constructs sets or groups of related XML elements to be used for computation of

C_{e1}: <Employee eid="e01"/>
<Name>Sirichai</Name>
<WorksFor dept="d01"/>
<Salary>2500</Salary>
</Employee>

C_{e2}:<Employee eid="e02"/>
<Name>Orathai</Name>
<DirectSupervisor eid="e01"/>
<WorksFor dept="d01"/>
<Salary>2000</Salary>
</Employee>

C_{e3}: <Employee eid="e03"/>
<Name>Tawee</Name>
<WorksFor dept="d02"/>
<Salary>2200</Salary>
</Employee>

C_{e4}: <Employee eid="e04"/>
<Name>Manop</Name>
<DirectSupervisor eid="e02"/>
<WorksFor dept="d01"/>
<Salary>1800</Salary>
</Employee>

(a) Modeling database contents – a collection of XML elements.

```
C_r1: <EmpRelation level="1"/>
        <Supervisor eid=$S:X>
          $S:Xname </Supervisor>
        <Subordinate eid=$S:Y>
          $S:Yname </Subordinate>
      </EmpRelation>
    ←   <Employee eid=$S:Y/>
          <Name>$S:Yname</Name>
          <DirectSupervisor eid=$S:X/>
          $E:Y_info
        </Employee>,
        <Employee eid=$S:X/>
          <Name>$S:Xname</Name>
          $E:X_info
        </Employee>.
```

% If X is a direct supervisor of an employee Y, one can conclude that X is a first-leveled supervisor of Y.
% C_{r1} models such a relationship. Its body comprises two XML expressions which retrieve the eid and the name of Y and select the eid and the name of Y's DirectSupervisor (i.e., Employee X). Its head then constructs an EmpRelation-element which explicitly represents such a supervisor-subordinate relationship between X and Y.

```
C_r2: <EmpRelation level=$S:m/>
        <Supervisor eid=$S:X>
          $S:Xname </Supervisor>
        <Subordinate eid=$S:Z>
          $S:Zname </Subordinate>
      </EmpRelation>
    ←   <EmpRelation level=$S:n/>
          <Supervisor eid=$S:X>
            $S:Xname </Supervisor>
          <Subordinate eid=$S:Y>
            $S:Yname </Subordinate>
        </EmpRelation>,
        <Employee eid=$S:Z/>
          <Name>$S:Zname</Name>
          <DirectSupervisor eid=$S:Y/>
          $E:Z_info
        </Employee>,
        Add(<Num>$S:n</Num>,
            <Addendum>1</Addendum>,
            <Result>$S:m</Result>).
```

% If X is the n^{th}-level supervisor of Y, and Y is a DirectSupervisor of Z, one can imply that X is the $(n+1)^{th}$-level supervisor of Z.
% The first XML expression in C_{r2}'s body represents an EmpRelation-element in the database which encodes that X is the n^{th}-level supevisor of Y. The second expression in the body describes an Employee-element in the collection specifying that Y is a DirectSupervisor of Z. The constraint Add then defines that m is the result of increasing n by 1. The head of C_{r2} then derives an EmpRelation-element which describes that X is an m^{th}-level (i.e., $(n+1)^{th}$-level) supervisor of Z.

(b) Modeling a relationship among Employee-elements in the database.

Figure 1. An XDD description XDB: modeling of an XML database (Example 1).

summary information. This syntax intuitively separates a query clause into three parts: (i) a *constructor* described by the head, (ii) a *pattern* described by the set of XML expressions in the body, and (iii) a *filter*, described by the set of XML constraints and set-aggregations in the body.

Each query will be executed on some specified collection of XML documents, modeled as an XDD description, and will return as its answer a set of XML elements explicit in or derivable from the document collection and satisfying all the conditions of the query. Besides simply returning a sequence of qualified XML elements, new ones with a desirable structure may be obtained from those selected elements. This objective can be achieved by an appropriate specification of the head expression.

The following subsections explain the formulation of the essential functionalities which a particular query language should provide [10, 13, 14, 17].

3.1. Selection and Extraction

Selection of XML elements, which satisfy a Boolean composition of a query's selective constraints on the elements' components, such as tag names and child elements, can be expressed by one or more XML clauses. If all the conditions are related by 'AND', a single clause will suffice. However, if the operation is formed by n 'OR' conditions, n separate clauses will be required. The body of a clause comprises a single XML expression, representing XML documents or elements to be selected, together with XML constraints, describing selection conditions.

Extraction is an operation which returns a set of partial XML documents with only required components. It can be expressed by an XML clause, the body of which consists of a single XML expression, describing the pattern of the elements to be extracted, and the head of which specifies the desired fragments.

Example 2 Clause Q_1 of Fig. 2 formulates a query which returns only Names and eids of all Employees who work for the department "d01" and earn more than 2000. □

3.2. Combination / Joining

Combination or *joining* of related information from different parts of XML elements/documents is expressed by a single XML clause with n XML expressions in the body, representing the n parts to be combined, together with zero or more XML constraints, specifying the join con-

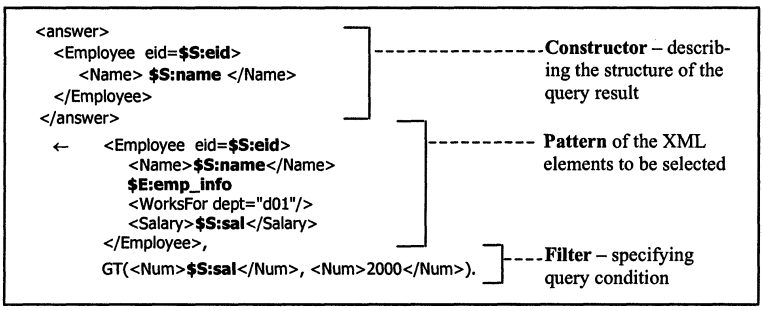

Figure 2. Q_1: a selection and extraction query (Example 2).

ditions. The head of the clause then specifies the combining structure. Note that an *equi-join* can be simply expressed by using the same variable to represent the equality condition on a desired component because all occurrences of the same variable in an XML clause must be specialized into identical component.

Example 3 Clause Q_2 of Fig. 3-a represents a query which lists Names and eids of all Employees who are subordinates of both Sirichai and Orathai. Its formulation in XQuery demands a definition of a recursive function which expresses the supervisor-subordinate relationship; thus, it is defined in Fig. 3-b and called EmpRelation. Based on that function, Fig. 3-c gives an XQuery representation of the example. One may notice that XDD is simpler and provides a more declarative and comprehensible approach. □

3.3. Operations on Basic Datatypes, Literal, Names and Schemas

XDD defines various types of variables ranging over different kinds of XML expressions' components, e.g., strings (or literal data), names, sub-expressions, and provides also a concept of XML constraints. Thus, a particular operation on a datatype or an expression component can be modeled by appropriate definitions of an XML constraint. For example, in order to define a numeric comparison operator 'less than', let LT be a constraint predicate; an XML constraint LT(<Num>x</Num>, <Num>y</Num>) is a true constraint, iff x and y are numbers and $x < y$. The constraints Add and GT contained in the clauses C_{r2} of Fig. 1 and Q_1 of Fig. 2, respectively, demonstrate how constraints can be used.

```
<Sirichai_Orathai_Subordinate eid=$S:eid>      % The first expression in Q2's body represents
   $S:name                                     % an EmpRelation-element which encodes in-
</Sirichai_Orathai_Subordinate>                % formation about a supervisor-subordinate
   ←  <EmpRelation level=$S:m>                 % relationship among employees in the data-
         <Supervisor eid=$S:sirichai>          % base, where the supervisor is Sirichai and
            Sirichai</Supervisor>              % the subordinate is any employee. The sec-
         <Subordinate eid=$S:eid>              % ond expression represents an EmpRelation-
            $S:name</Subordinate>              % element, where the supervisor is Orathai
      </EmpRelation>,                          % and the subordinate is the one who is also
      <EmpRelation level=$S:n>                 % the subordinate of Sirichai. Note: the S-
         <Supervisor eid=$S:orathai>           % variables $S:eid and $S:name, representing
            Orathai</Supervisor>               % the eid and name of such a subordinate, re-
         <Subordinate eid=$S:eid>              % spectively, are used to equi-join the two ex-
            $S:name</Subordinate>              % pressions together.
      </EmpRelation>.
```

(a) XML clause Q_2.

```
define function EmpRelation(element $super) returns element* {   % For a given XML ele-
   for $emp in /Employee                                         % ment represented by
   where $emp/DirectSupervisor/@eid = $super/@eid                % the variable $super,
   return (                                                      % the function Emp-
    <EmpRelation level="1">                                      % Relation returns a
      <Supervisor eid= {$super/@eid} >                           % sequence of EmpRela-
         {$super/Name/text()} </Supervisor>                      % tion-elements which
      <Subordinate eid= {$emp/@eid}>                             % encode information
         {$emp/Name/text()} </Subordinate>                       % about all subordinates
    </EmpRelation>                                               % of $super, by invok-
   union                                                         % ing itself recursively.
      for $e in EmpRelation($emp)
      return
      <EmpRelation level= {number($e/@level)+1} >
         <Supervisor eid={ $super/@eid} >
            {$super/Name/text()} </Supervisor>
         <Subordinate eid= {$e/@eid} >
            {$e/Name/text()} </Subordinate>
      </EmpRelation>
   )
}
```

(b) XQuery representation of the supervisor-subordinate relationship, corresponding to the XML clauses C_{r1} and C_{r2} of Fig. 1.

```
let $sirichai := /Employee[Name = "Sirichai"],      % In the query, the variables
      $orathai := /Employee[Name = "Orathai"]       % $sirichai and $orathai repre-
for   $s_empRelation IN EmpRelation($sirichai),     % sent the Employee-elements,
      $o_empRelation IN EmpRelation($orathai)       % whose names are Sirichai and
where $o_empRelation/Subordinate/@eid =             % Orathai, respectively, and the
         $s_empRelation/Subordinate/@eid            % variables $s_empRelation and
      and $o_empRelation/Subordinate/text() =       % $o_empRelation represent
         $s_empRelation/Subordinate/text()          % EmpRelation-elements, which
return                                              % encode information about
   <Sirichai_Orathai_Subordinate                    % Sirichai's and Orathai's sub-
      eid= {$s_empRelation/Subordinate/@eid}>       % ordinates, respectively.
      {$s_empRelation/Subordinate/text()}
   </Sirichai_Orathai_Subordinate>
```

(c) XQuery representation of the example query.

Figure 3. A query which joins several elements and constructs new ones (Example 3)

Moreover, a document's schema—a DTD or an XML Schema—can also be modeled by corresponding XML clauses, thus, yielding verification of the conformance of a given XML element with respect to such a schema [4, 6]. Hence, the developed approach readily supports all necessary operations on basic datatypes, literal data, names and schemas. In addition, user-defined operations or functions can also be defined in terms of XML clauses. Fig. 3-b of Example 3 shows that EmpRelation, a complex, recursive function, can be simply formalized as the clauses $C_{r1} - C_{r2}$.

With a concern that XML has not defined a distinction between encoding textual data and other types, XDD provides an ability of implicit data casting among different types based on certain coercion rules. For instance, an element type (tag name) can be immediately used in a string operation, but only a string, which contains only valid characters and follows XML naming rules, can be regarded as an element type or an attribute name.

3.4. Aggregation

Grouping of related XML elements, sharing certain specified criteria, and computation of its summary information is formalized as an XML clause. Its body contains a set-aggregation and XML constraints, describing the grouping criterion and aggregation functions, respectively, and its head specifies the structure of the resulting elements.

Example 4 The clause Q_3 of Fig. 4 selects Name of each Employee in the database having at least two Subordinates and lists also Names of such Subordinates. □

Example 5 The clause Q_4 of Fig. 5 counts the number of Employees working for the department "d01" and calculates the average salary of the Employees. □

3.5. Sorting

Sorting of query results is supported by employment of an XML constraint:

sort(L_1, L_2,<OrderBy path=p direction=d />),

where L_1 specifies a list of XML expressions to be sorted, L_2 the result of sorting L_1 according to the specification given by the path p and direction d of the OrderBy-element, which indicates the order-item and the sorting direction (ascending or descending), respectively.

```
<Employee>                                          %  The Employee-expression
   <Name>$S:super_name</Name>                       %  in the body of Q3 repre-
   <SubordinateList>$E:list</SubordinateList>       %  sents each employee ob-
</Employee>                                         %  ject in the database, and
  ←   <Employee eid=$S:supervisor>                  %  the set-aggregation defines
         </Name>$S:super_name</Name>                %  that the variable $E:list is
         $E:emp_info                                %  the set of all subordinates
      </Employee>,                                  %  of such an employee. The
      <xdd:SetAggregation>                          %  constraints Count and GT
         <xdd:Set>$E:list</xdd:Set>                 %  restrict the set's cardinal-
         <xdd:Constructor>                          %  ity to a number greater
            <Subordinate level=$S:level>            %  than one. The head of Q3
               $S:sub_name</Subordinate>            %  describes that the query
         </xdd:Constructor>                         %  returns Employee-elem-
         <xdd:Pattern>                              %  ents which list the names
            <EmpRelation level=$S:level>            %  as well as the subordinates
               <Supervisor eid=$S:supervisor>       %  of selected employee.
                  $S:super_name</Supervisor>
               <Subordinate eid=$S:sub>
                  $S:sub_name</Subordinate>
            </EmpRelation>
         </xdd:Pattern>
      </xdd:SetAggregation>
      Count($E:list, <Result>$S:count</Result>).
      GT(<Num>$S:count</Num>, <Num>1</Num>).
```

Figure 4. Q_3: a query which groups and lists a set of related XML elements (Example 4).

For example, let the query given in Example 4 be modified so that, in the output, a list of Subordinate-elements, nested in a SubordinateList-element, are sorted alphabetically by their textual contents; this modified query is formulated by insertion of the following constraint into Q_3's body:

```
sort($E:list, $E:sortedlist,
   <OrderBy path="Subordinate/text()" direction="ascending" />),
```

and by replacement of the variable $E:list in Q_3's head by the variable $E:sortedlist.

3.6. Universal and Existential Quantifiers

Although XDD theory does not explicitly treat universal and existential quantifiers, each XML clause can typically be interpreted as all of its variables are universally quantified. The theory allows the employment of XML constraints, *negations* and *Skolem* functions as well as those techniques developed in logic programming theory for formulation of a query with a similar meaning of universal and existential quantifiers [4].

```
<Department id="d01">
   <EmployeeNum>$S:count</EmployeeNum>
   <AvgSalary>$S:avg</AvgSalary>
</Department>
   ← <xdd:SetAggregation>
        <xdd:Set>$E:list</xdd:Set>
        <xdd:Constructor>
           <Salary eid=$S:eid>$S:sal</Salary>
        </xdd:Constructor>
        <xdd:Pattern>
           <Employee eid=$S:eid>
              $E:emp_info
              <WorksFor dept="d01"/>
              <Salary currency="USD">
                 $S:sal</Salary>
           </Employee>
        </xdd:Pattern>
     </xdd:SetAggregation>
     Count($E:list, <Result>$S:count</Result>),
     Avg($E:list, <Result>$S:avg</Result>).
```

```
% The set-aggregation contained in
% the body of the clause defines
% that the variable $E:list repre-
% sents the set of XML elements
% which encode the salary of each
% employee in the database who
% works for the department "d01".
% The constraints Count and Avg
% then specifies that the S-
% variables $S:count and $S:avg
% represent the number of ele-
% ments in the set and the average
% salary, respectively.
```

Figure 5. Q_4: a query with aggregation functions: Count and Avg (Example 5).

3.7. Closure and Nested Queries

Since both input to and output from a query are sets of XML elements, queries formulated under the proposed approach are closed. A composite or nested query is expressed as $n + m$ XML clauses, the subquery of which is formulated as n clauses and the super-query as m clauses. For each subquery clause, its head specifies the structure of the intermediate results to be used as input to the super-query, and for each super-query clause, the set of XML expressions contained in its body describes the desired patterns of XML elements to be matched with the results obtained by evaluation of the subquery.

4. Query Evaluation

Given an XML database represented by an XDD description XDB, the answer to a query formulated as an XDD description Q is the set of XML elements, explicit in or derivable from the database XDB and satisfying all of its conditions. By employment of *Equivalent Transformation* (ET) paradigm [3], such a query Q is evaluated by transforming the description ($XDB \cup Q$) successively into a simpler, but equivalent description, from which the answer can be obtained easily and directly. More precisely, such a description ($XDB \cup Q$) will be successively transformed until it becomes the description ($XDB \cup Q'$), where Q' holds

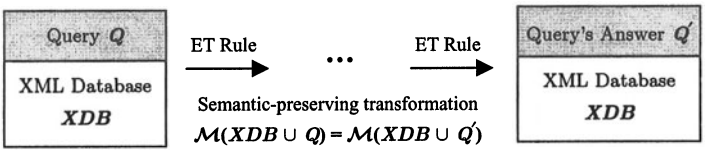

Figure 6. Proposed query evaluation mechanism by Equivalent Transformation.

only ground XML unit clauses and $\mathcal{M}(XDB \cup Q') = \mathcal{M}(XDB \cup Q)$. The XML elements, directly described by the XML unit clauses in Q', readily yield the answer to the query Q. Figure 6 depicts the proposed computational mechanism for query evaluation.

In order to guarantee the computation's correctness, only *ET rules* or *semantics-preserving transformations* can be applied at every transformation step. ET is a new, flexible computational model, which is considered to be more efficient than the inference in the logic paradigm and the function evaluation in the functional programming paradigm. The unfolding transformation presents an example of ET rules. Other kinds of ET rules specific for XML data structure, XML constraints and set-aggregations can also be devised and applied, especially to improve the computational efficiency.

Papers [4, 19] develop ET rules for transformation of set-aggregations. Speaking intuitively, given an XML database XDB and a query clause with a set-aggregation

<xdd:SetAggregation>
<xdd:Set> S </xdd:Set>
<xdd:Constructor> E_C </xdd:Constructor>
<xdd:Pattern> E_P </xdd:Pattern>
</xdd:SetAggregation>,

in its body, the set S is obtained by equivalent transformation of the description $XDB \cup \{E_C \leftarrow E_P\}$ into $XDB \cup \{E_1, \ldots, E_n\}$, where E_i is an XML element. Thus, S becomes the set $\{E_1, \ldots, E_n\}$. That is, for each element in the database XDB which can be matched with the pattern E_P, an element represented by E_C will be constructed and included in the set S.

Example 6 Based on ET framework, this example demonstrates evaluation of the queries Q_2 of Example 3 and Q_3 of Example 4 with respect to the sample XML database XDB of Example 1. However, due to page limitation, details of each transformation step for solving the queries will be omitted.

By means of unfolding, the description $XDB \cup \{Q_2\}$ can be transformed equivalently and successively into $XDB \cup \{Q_2'\}$, where Q_2' is

```
Q'2 :  <Sirichai_Orathai_Subordinate eid="e04">
          Manop
       </Sirichai_Orathai_Subordinate>.
```

Hence, the obtained element Q_2' is the answer to the query Q_2.

Next, consider the processing of the query Q_3. Using unfolding and the ET rules for set-aggregation, one obtains that the variable $E:list represents the set

```
{ <Subordinate level="1"> Orathai </Subordinate>,
  <Subordinate level="2"> Manop </Subordinate> },
```

and the constraints

```
Count($E:list, <Result>$S:count</Result>), and
GT(<Num>$S:count</Num>, <Num>1</Num>)
```

are true constraints, iff $S:count is specialized into the number 2. Thus, the description $XDB \cup \{Q_3\}$ can be transformed successively into $XDB \cup \{Q_3'\}$, where Q_3' is

```
Q'4 :  <Employee>
          <Name> Sirichai </Name>
          <SubordinateList>
             <Subordinate level="1"> Orathai </Subordinate>
             <Subordinate level="2"> Manop </Subordinate>
          </SubordinateList>
       </Employee>.
```

Since only unfolding and ET rules for set-aggregation are applied, in each transformation step, the correctness of the obtained answers to the queries Q_2 and Q_4 are assured. □

5. Related Work

It should be emphasized that the developed approach is not proposed as a query language for XML. Its primary goal is to provide a theoretical foundation for the development of an XML query processor. Certain query languages, such as *XQuery* [8, 12], *XML-QL* [11] or *XQL* [18], can be employed as an interface language, which enables users to formulate queries in their preferred syntax. Submitted queries, expressed in such a syntactic-sugar language, will be translated into the proposed query formulation and then evaluated accordingly. Thus, the proposed approach

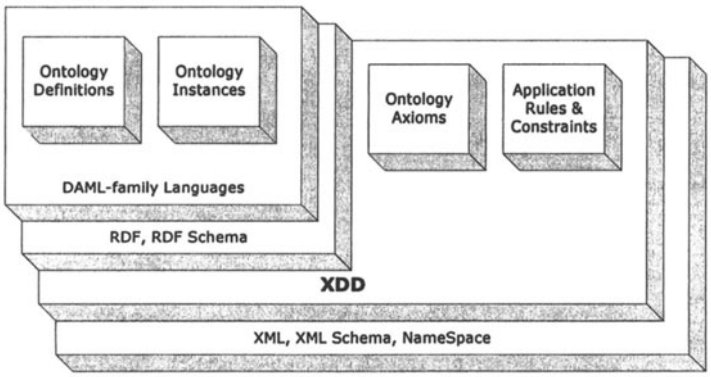

Figure 7. XDD: a foundation for RDF(S) and DAML-family markup langugages.

to query formulation and evaluation can be utilized as a foundation for the processing of XML queries, expressed in other languages.

Moreover, Fig. 7 shows that XDD readily provides a direct support for manipulation of *RDF* data—an emerging, prominent standard for representation of metadata, which employs XML as its encoding and interchange syntax. RDF data—including RDF statements, *RDF Schemas* (*RDFS*) as well as extensions to RDF(S), such as, *DAML-family markup languages* [15]—can be mapped directly onto ground XML expressions. In addition, XDD can enhance these languages' expressive power by allowing expression of arbitrary constraints, rules and axioms—an important feature missing in such languages—in terms of XML clauses. Databases of RDF and DAML data as well as their queries can then be precisely formulated and manipulated [21].

6. Conclusions

Besides the abilities to express and process various kinds of queries, the XDD approach to query formulation and evaluation also has the following properties and advantages: (i) declarativeness, (ii) expressiveness, (iii) independence of schema availability, and (iv) provision of a direct support for manipulation of RDF data,

In order to effectively manage a large collection of XML documents and to improve the efficiency of the query processing, development of semantic query optimization techniques—based on knowledge of the database's constraints, DTDs and Schemas—is envisaged.

On the basis of XDD and ET paradigm, *XDD System*—a prototype Web-based XML processor available at http://kr.cs.ait.ac.th/xdd—has been implemented. Preliminary tests on several XML and RDF applications, such as software configuration management [16] and agent-based systems [5], reveal the framework's viability and potential in real applications.

References

[1] Abiteboul, S., Quass, C., McHugh, J., Widom, J., Wiener, J: The Lorel Query Language for Semistructured Data. Int'l J. Digital Libraries, **1**(1) (1997) 68–88

[2] Akama, K.: Declarative Semantics of Logic Programs on Parameterized Representation Systems. Advances in Software Science and Technology, **5** (1993) 45–63

[3] Akama, K., Shimitsu, T., Miyamoto, E.: Solving Problems by Equivalent Transformation of Declarative Programs. J. Japanese Society of Artificial Intelligence (JSAI). **13**(6) (1998) 944–952 (in Japanese)

[4] Anutariya, C. XML Declarative Description. Doctoral Dissertation, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (2001)

[5] Anutariya, C., Wuwongse, V., Akama, K., Wattanapailin, V.: Semantic Web Modeling and Programming with XDD. Proc. Semantic Web Working Symposium (SWWS-01), CA (2001), 161–180.

[6] Anutariya, C., Wuwongse, V., Nantajeewarawat, E., Akama, K.: Towards a Foundation for XML Document Databases. Proc. 1st Int'l Conf. on Electronic Commerce and Web Technologies (EC-Web 2000), London, UK. LNCS, Springer Verlag, **1875** (2000) 324–333.

[7] Beech, C., Malhotra, A., Rys, M.: A Formal Data Model and Algebra for XML. W3C XML Query Working Group Note (1999)

[8] Boag, S., Chamberlin, D., Clark, J., Fernandez, M., Florescu, D., Robie, J., Siméon, J., Stefanescu, M.: XQuery 1.0: An XML Query Language, W3C Working Draft (2001) `http://www.w3.org/TR/xquery/`

[9] Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0. W3C Recommendation (1998) `http://www.w3.org/TR/REC-xml`

[10] Chamberline, D. Fankhauser, P., Marchiori, M., Robie, J.: XML Query Requirements. W3C Working Draft (2001) `http://www.w3.org/TR/xmlquery-req`

[11] Deutsch, A., Fernandez, M., Florescu, C., Levy, A., Suciu, C.: XML-QL: A Query Language for XML. Proc. Query Languages Workshop (QL'98), Boston, MA (1998) `http://www.w3.org/TR/NOTE-xml-ql-19980819.html`

[12] Fankhauser, P., Fernandez, M., Malhotra, A., Rys, M., Siméon, J., Wadler, P.: XQuery 1.0 Formal Semantics. W3C Working Draft (2001) `http://www.w3.org/TR/query-semantics`

[13] Fernandez, M. and Robie, J.: XML Query Data Model, Feb. 2001. W3C Working Draft (2000) `http://www.w3.org/TR/query-datamodel`

[14] Fernandez, M., Siméon, J., Suciu, C., Wadler, P.: A Data Model and Algebra for XML Query. Draft Manuscript (1999)
`http://www-db.research.bell-labs.com/~wadler/topics/xml.html`

[15] Hendler, J., McGuinness, D.L.: The DARPA Agent Markup Language. IEEE Intelligent Systems, **15**(6) (2000) 72–73

[16] Kitcharoensakkul, S., Wuwongse, V.: Unified Versioning using Resource Description Framework. Annals of Software Engineering, Kluwer Academic Publishers, **11** (2001) 259–297.

[17] Quass, C.: Ten Features Necessary for an XML Query Language. Proc. Query Languages Workshop (QL'98), Boston, MA (1998)

[18] Robie, J., Lapp, J., Schach, C.: XML Query Language (XQL). Proc. Query Languages Workshop (QL'98), Boston, MA (1998)

[19] Tsuji, T., Akama, K., Koike, H., Mabuchi, H: Representation and Computation of Set Expressions. Proc. IASTED Int'l Conf. Applied Informatics, Innsbruck, Austria (2001)

[20] Wuwongse, V., Akama, K., Anutariya, C., Nantajeewarawat, E.: A Data Model for XML Databases. Proc. 1st Int'l Conf. on Web Intelligence (WI-2001), Maebashi, Japan. LNAI, Springer Verlag **2198** (2001) 237–246.

[21] Wuwongse, V., Anutariya, C., Akama, K., Nantajeewarawat, E.: XML Declarative Description: A Language for the Semantic Web. IEEE Intelligent Systems, **16**(3) (2001) 54–65