A Unified Framework for the Semantic Integration of XML Databases

Doan Dai Duong and Le Thi Thu Thuy
The University of New Brunswick
{Duong_Dai.Doan, Thuy_Thi_Thu.Le}@unb.ca

***Abstract*—This paper proposes an XML Declarative Description (XDD) - based integration framework for XML databases. All data components and processing components of the framework, such as databases, ontologies, queries and schema integration components can be formulated by XDD. Since the boundary between the system's components is removed, the interoperation capacity between them is enhanced, thus reducing the overhead of the system's communication. Moreover, an important achievement of this framework is that it can integrate *n* schemas at a time and simultaneously decompose a query into *n* subqueries.**

***Index Terms* — Database Integration, XML Declarative Description, Schema Integration, Query Decomposition.**

I. INTRODUCTION

THE advent of the XML language has laid the foundation for the Semantic Web revolution and has opened many chances for web metadata applications described in terms of XML Schema. Since distributed XML databases [14] of these applications usually belong to a specific domain (e.g., databases of banking systems), their integration is a demanding task. An integrated XML database system creates a global schema for all participating XML databases while query processing helps its users to get required and integrated data through the global schema, thus reducing computer usage of clients who want to find information.

Most of the previously developed integration systems, such as TSIMMIS [6], HERMES [10], and FLORID [1] deal with conventional data models. In TSIMMIS [6], an internal data model, namely OEM, is proposed, which is used to represent data objects from heterogeneous data sources. In HERMES [10], all components of the system are composed by a Prolog-like language. Several toolkits are used to access data sources and interact with components of the system. The FLORID system [1] uses F-Logic, a combination of an object-oriented database and a deductive rule language, to model schemas, data and queries.

Having inherited strong characteristics from these novel models, recent integration systems, such as LoPix [15], HERA [9] strengthen them by incorporating XML, well-suited to presenting semi-structured data. LoPix [15] uses XPathLog, an XPath-based language to integrate overlapping XML trees (by fusing and linking synonym nodes in XML trees), from which an XTreeGraph data model (possibly be cyclic) is created. The XTreeGraph plays the role of a global schema, where users can pose queries to get answers. Here, XPathlog is a modeling language. It models XML trees, user queries and supports inference mechanism. The HERA system [9] is designed as an integration architecture based on semantic integration and on demand of information retrieval. The system uses RDF as an underlying model, which contains a hierarchical structure of concepts, relations as well as facts and axioms. For more literature survey, see [3, 4].

From these surveyed integration systems, one can easily see that they share a main theme. Usually, a data model is used to model components of an integration framework. This enhances the communication between the system's components. If an integration system uses a variety of tools and (rule) languages, its internal structure is heterogeneous within itself requiring information to be reconciled before passing from some components to other ones.

In this paper, we show how to use XML Declarative Description (XDD) [13] to model all components of our XML database integration framework. The purpose of using XDD is two-fold. First, XDD can express both data components and processing components of an integration framework, such as facts (databases), rules (including constraints), ontologies (i.e., taxonomies and rules), and a mediator in a unified manner. All components of the framework can therefore interact with each other harmoniously to reduce the overheads of the components' communication. Second, since most integration systems [1, 7, 9, 10, 11, 15] can only integrate two schemas at a time, the complexity of these systems is extremely high, namely approximately $(n-1)!$ times, to modify and reconstruct mappings and to resolve conflicts (*n* is the number of participating schemas). Our system can both integrate *n* schemas and decompose a query into *n* subqueries at a time (*one-shot* strategy [3]). Details are referred to sections III.B.*1)* and III.B.*2)*.

Section II provides a brief introduction about XDD. Section III proposes an XDD-based integration system where its data components and processing components are modeled by XDD uniformly. Section IV describes a system prototype and gives

initial experimental results.

II. XML Declarative Description (XDD)

In order to represent a set of similar XML documents, as well as to create inferential mechanisms on them, XDD – a combination between conventional XML elements and variables (*name, string, pair of attribute, XML expression* and *intermediate expression* or *$N, $S, $P, $E,* and *$I* variables, respectively) - is used with high expressive power. Table 1 shows variables in the XDD language.

TABLE 1. VARIABLES IN THE XDD LANGUAGE

Variable	Instantiate to
$N	Element or attribute names
$S	Strings
$P	Sequences of zero or more attribute value pairs
$E	Sequences of zero or more XML expressions
$I	Part of XML expressions

In XDD, an ordinary XML element without variables is called a *ground XML expression*. One containing variables is called a *non-ground XML expression,* used to model a set of similar ground XML expressions or ordinary XML documents. Besides XML expressions, an important concept of XDD language is an *XDD clause* or an *XDD rule*. An XDD clause has the following form:

$$H \leftarrow B_1, \ldots, B_m, C_1, \ldots, C_n$$

where H is the head and the set $\{B_1, \ldots, B_m, C_1, \ldots, C_n\}$ is the body. H, B_i $(i=1..m)$ are XML expressions while the C_j $(j=1..n)$ denote constraints or restrictions on the XML expressions. When an XDD clause does not include a body, it is referred to as a *fact* or an *XML unit clause* $(H\leftarrow.)$ or (H).

III. An XDD-Based Integration System

A framework for the XML database integration system is proposed (Fig. 1). It consists of two main components: *data components* and *processing components*, discussed in the following sections.

A. Modeling of Data

1) XML Databases

A database includes two parts: *intension* and *extension*. The intension defines internal structures of a database like its schemas, logical specifications, relationships, indexes and constraints. The extension contains actual data values called *occurrences* or *instances*. While the extension changes with time, the intension is supposed to be time invariant [3]. In our framework, the extension and intension are modeled as *ground XML expressions* and *non-ground XML expressions* (or *unit* and *non-unit clauses*), respectively. Keeping in mind that since the XDD clause's format is: $H \leftarrow B_1, \ldots, B_m, C_1, \ldots, C_n$, where $C_1, \ldots, C_n$ are constraints, data integrity can be formally expressed. With these characteristics, data from XML databases can be extracted and processed directly by using XDD rule-based components of the system. Details of these expressive processes are discussed in more depth in [5].

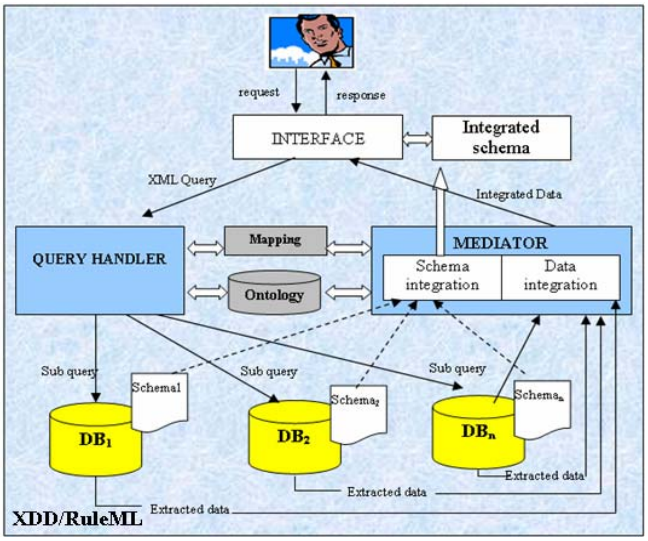


Fig.1. System's architecture

2) XML Query

In XDD, a query is modeled in three parts: a *constructor*, *patterns* and *filters* corresponding to the three parts (H, B_i, C_j) of an XDD rule ($H \leftarrow B_1 \ldots, B_m, C_1, \ldots, C_n$). When executed, the patterns will match with facts in participating XML databases. If these facts satisfy the filters (*constraints* of the query), they will be extracted as answers of the query in terms of the constructor. The following example (Fig. 2) shows an XML query modeled by XDD.

```
<Answer>                                  %  List name and
     <name>$S:name</name>                 %  nationali
     <nationality>$S:nation               %  ty of all
     </nationality>                       %  students
</Answer>                                 %  whose gpa
←                                         %  are greater
   <Student>                              %  than 3.5.
      <name>$S:name</name>
      <nationality>$S:nation
      </nationality>
      <GPA>$S:gpa</GPA>
      $E:properties
   </Student>
   [$S:gpa>3.5]
```

Fig. 2. Query modeled by XDD language

In this example, the body of the rule contains two parts, the *pattern* and the *filter* corresponding to

`<Student>...</Student>` and `[$S:gpa >3.5]`, respectively. The pattern will match with contents of XML databases (i.e. any `Student` expression contains three sub-elements, namely `name`, `nationality` and `gpa`). Because of the generality of the rule (the `Student` expression may contain extra information), the `$E:properties` variable is used whose values can be zero or more XML expressions. The filter describes the selection condition (value of the `gpa` element must be greater than 3.5). When executed, actual data are bound to `$S:name` and `$S:nation` variables and returned to users in terms of the structure of the constructor (the head of the rule).

3) Ontology and Mapping

- ***Ontology***

Ontology is a significant component of an integration system. It is represented in terms of a hierarchy of concepts, which is extremely useful for supplying semantic information to combine local schemas into an integrated schema and for yielding mappings to describe correspondences between the local schemas and the integrated one. Many types of ontologies are suggested, such as RDF[1] or OWL[2], which describe many types of relationships among classes and properties. However, their expressive knowledge representation will be increased if they can be combined with a rule language to fully express inferential mechanisms, which are often found in human thinking. In our framework, we consider an ontology represented in terms of XML as a set of *ground XML expressions*; thus, no further transformation is required since a set of *ground XML expressions* is a well-formed XML document. For example, to express statements:

A *fullname* is a union of a *firstname* and a *lastname*. A name is a *fullname*.

and a rule:

If class *A* is equivalent to class *B*, then the content of class *B* is also the content of class *A*.

These are then marked up by OWL modeled by XDD as three clauses c_1, c_2 and c_3 as follows:

c_1
```
<owl:Class rdf:ID="name">
 <rdfs:equivalentClass
     rdf:resource="#fullname"/>
</owl:Class>←.
```

c_2
```
<owl:Class rdf:ID="fullname">
 <rdfs:unionOf>
    <owl:Thing rdf:about="Fname"/>
    <owl:Thing rdf:about="Lname"/>
 </rdfs:unionOf>
</owl:Class>←.
```

c_3
```
 <owl:Class rdf:ID="$S:ClassA">
  <rdfs:unionOf>
     $E:expr1
  </rdfs:unionOf>
 </owl:Class>
 ←
  <owl:Class rdf:ID="$S:ClassA">
   <rdfs:equivalentClass
       rdf:resource="$S:ClassB"/>
  </owl:Class>

  <owl:Class rdf:ID="$S:ClassB">
   <rdfs:unionOf>
       $E:expr1
   </rdfs:unionOf>
 </owl:Class>
```

c_4
```
<owl:Class rdf:ID="name">
 <rdfs:unionOf>
    <owl:Thing rdf:about="Fname"/>
    <owl:Thing rdf:about="Lname"/>
 </rdfs:unionOf>
</owl:Class>←.
```

Fig. 3. Ontology modeled by the XDD language.

In this example, the unit clauses c_1 and c_2 represent explicit information or data instances of the ontology, while the non-unit clause c_3 represents implicit information. The clause c_4 is produced from *specialized operators* – binding variables to values – (`$S:ClassA`, `"name"`), (`$S:ClassB`, `"fullname"`) and (`$S:expr1` to the content of `$S:ClassB`) applied to the unit clauses c_1 and c_2. Of course, the clause c_4 can be derived from the two clauses c_1 and c_2 if being reasoned by an OWL engine. However, by adding an extra rule c_3, we have shown that we can use XDD to model and process an OWL ontology efficiently without using any OWL engine.

```
<Mapping>
   <global>
      <student>
          <country>$S:country</country>
      </student>
   </global>
   <local>
      <SATstudent source="A">
           <country>$S:country</country>
      </SATstudent>
      <SOMstudent source="B">
           <nation>$S:country</nation>
      </SOMstudent>
   </local>
</Mapping>
```

Fig. 4. Example of a mapping

- ***Mapping***

While the ontology supplies information for the integration processes, a set of mappings modeled by XDD supports decomposing XML queries and converting data. Each mapping is modeled by a *non-ground XML expression*. It describes a correspondence between an object in the

[1] http://www.w3.org/TR/REC-rdf-syntax/
[2] http://www.w3.org/TR/owl-xmlsyntax/

integrated schema and its corresponding objects in the local schemas by using two XML sub-expressions [8]. Fig. 4 shows an example of a mapping, specifying that the `country` element in the integrated schema has two corresponding elements `country` and `nation` in source `A` and source `B`, respectively.

In our system, mappings are combined with special rules to produce information for both query decomposition and data conversion presented in detail in the two subsections *2)* and *3)* of sections B.

B. Modeling of Processing Components

In the previous section, we present the data components of the system including the databases, queries, schemas, mappings and ontology modeled by the XDD language. This section will show how we build processing components such as a schema integration, a query decomposition and a data conversion based on the XDD language.

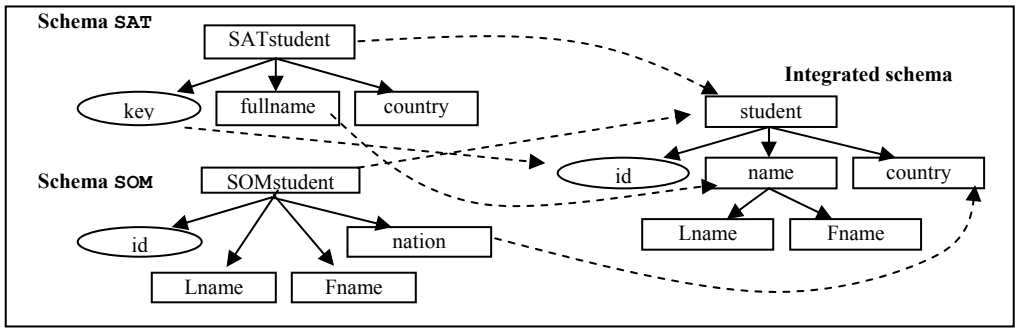


Fig. 5. Schemas of two XML databases `SAT` and `SOM` and the integrated schema

1) Schema Integration Component

Schema integration is a crucial component of the integration system. It harmonizes conflicts between schemas of participating databases. The result of this process is an integrated schema for all local sources and a set of mappings describing correspondences between objects in the local schema and the integrated one.

In our system, conflict resolution between various schemas is done at one time (the *one-shot* strategy). This strongly affects the way we solve conflicts. In order to do this, the local schemas are labeled by name (to be distinguished later) and are joined into a document to be harmonized simultaneously. Each local schema is considered as a big *ground XML expression* represented by an XML expression variable (i.e., `$E_variable`). An XET (XML Equivalent Transformation) engine [2] (built for the XDD language) can thus process all these documents simultaneously as variables. By following the *one-shot* strategy, the complexity of the system is greatly reduced.

When integrating XML schemas, we encounter many types of conflicts such as *Name, Structure, Constraint* and *Data type*. Each of them is then sub-divided into many types of conflicts (e.g. *synonym*, *homonym*, *missing items* and *aggregation conflicts*). For example, Fig. 5 shows the two schemas of XML databases `SAT` and `SOM` and the integrated schema, which is the result of the integration processes. Rectangle nodes represent elements, and oval nodes attributes. In this example, the two elements `SATstudent` and `SOMstudent` in the schemas `SAT` and `SOM` represent the same element `student`; thus, they should be relabeled `student` in the integrated schema. Fig. 6 shows an XDD rule for solving the synonym conflict. It specifies that if `$S:name1` and `$S:name2` in schemas `$S:A` and `$S:B` are synonyms, they are then replaced by `$S:common_name` (suggested by a term dictionary) in the integrated schema and the content of `$S:common_name` is the union of all children elements. When we apply this rule to the example in Fig. 5, `$S:name1`, `$S:name2` and `$S:common_name` correspond to `SATstudent`, `SOMstudent` and `student`, respectively. We therefore obtain `student` in the integrated schema.

Another complex type of conflict is the aggregation conflict, happening when elements or attributes in one schema are the result of aggregation of some elements or attributes in another schema. For instance, the element `fullname` in the schema `SAT` is the aggregation of the elements `Lname` and `Fname` in the schema `SOM`. The resolution of this problem is based on information from the ontology and the definition of a new data type `name` from the existing elements. Details of conflicts and their resolution are discussed in depth in [5].

```
<xsd:element name=$S:common_name source="integrated">
    $E:exp
</xsd:element>
  ← <xsd:element name=$S:name1 source=$S:A>
        $E:exp1
    </xsd:element>
    <xsd:element name=$S:name2 source=$S:B>
        $E:exp2
    </xsd:element>
    [synonyms($S:common_name,$S:name1, $S:name2),
    $E:exp=Union($E:exp1, $E:exp2)]
```

Fig. 6. XDD rule for solving the synonym conflict

2) Query Decomposition Component

In our approach, a global XML query modeled by XDD can be simultaneously decomposed into *n* sub-queries conforming to specific structures of local sources [8]. For instance, in Fig. 7, a user's query Q_I based on the integrated schema is decomposed into two sub-queries Q_{SAT} and Q_{SOM} conforming to the schemas SAT and SOM, respectively.

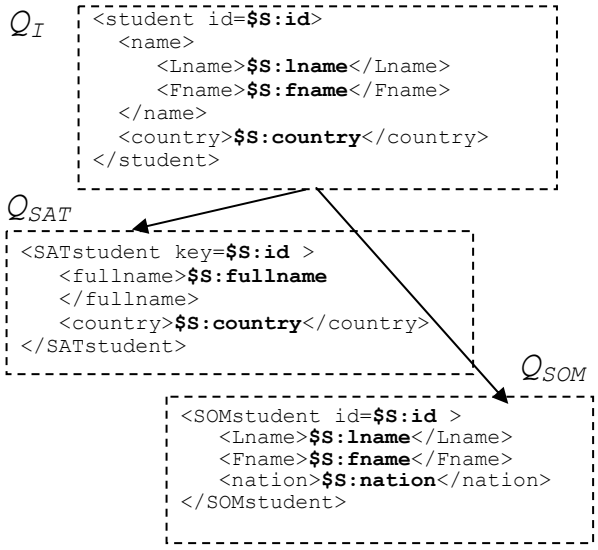


Fig. 7. Example of query decomposition

To achieve this power, the query decomposition component consists of a set of decomposition rules. The body of a decomposition rule contains two *XML expressions* (Fig. 8).

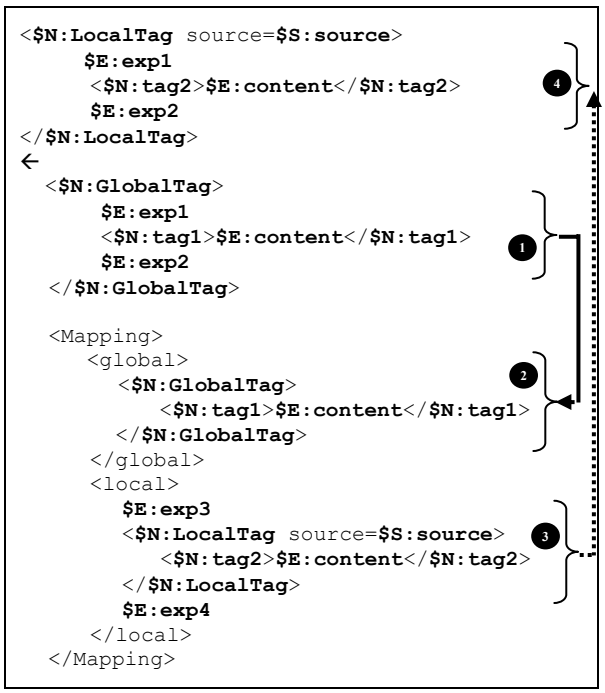


Fig. 8. Rule for query decomposition

The first one is a user query conforming to the structure of the integrated schema, while the second one is a general form of a mapping which shows corresponding objects (tags, attributes) between local sources and the global one. When the body of the rule ① matches with the global part of mappings ②, sub-queries ④ corresponding to local sources ③ are returned. The rule (Fig. 8.) can be interpreted as follows: If a user query contains a `$N:tag1` element whose corresponding element in a local schema `$S:source` via the mapping is `$N:tag2`. It then replaces `$N:tag2` with `$N:tag1` in the user query, yielding a decomposed query.

In the above query decomposition rule, values of the variables `$N:tag1`, `$N:tag2`, `$E:exp1`, `$E:exp2`, `$E:exp3` and `$E:exp4` are changed automatically, depending on values of `$S:source` in the mappings. By using this rule recursively, sub-queries for local sources are automatically produced.

3) Data Conversion Component

Users interact with the integrated schema and therefore like to get results in terms of the integrated schema format. Since data directly extracted from local sources still follow structures of the local schemas, they must be converted to the integrated schema format. In order to do this, the mappings are used again. This work is similar to *query decomposition* by using XDD rules but in the opposite direction. Using the combination of mappings and XDD rules, all extracted data are converted to the integrated schema format directly and simultaneously. It is worth emphasizing that the mappings constructed in the system are very useful. They are constructed at one time when integrating schemas but can be applied successfully to both the query decomposition and the data conversion by combining with the processing rules.

IV. System Prototype and Experimental Results

A. XML Equivalent Transformation (XET) – An XDD-Based Engine

From the XDD theory, a logic programming language is created named XET (XML Equivalent Transformation) [2]. XET runs on the ground of ETI[3] and Java. It transforms *XDD clauses* into *S-expressions* [13] so that these *S-expressions* can be executed in an ETI engine. In our system, all components are carefully pre-designed in XDD before being transformed into XET.

B. System Prototype

A system prototype named XSIS [12] - standing for XML Schema Integration System - is implemented in a local network under the Window environment. It consists of two main independent components, namely a *mediator* and a *query*

handler. Each component, in turn, contains many independent subcomponents, in which the output of one component is the input of another.

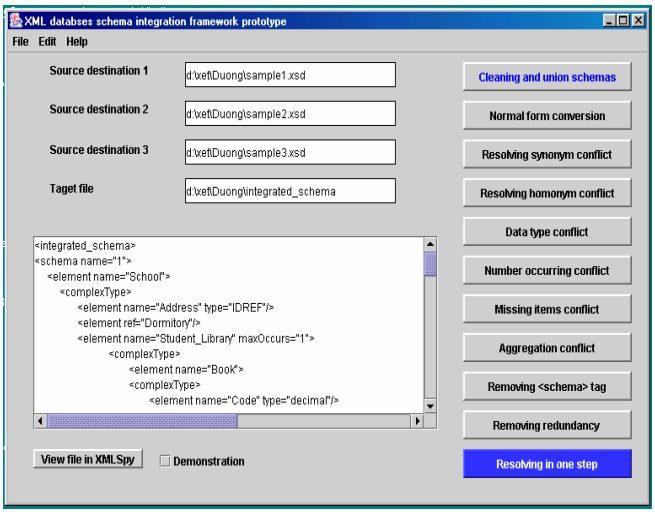


Fig. 9. Mediator of the system

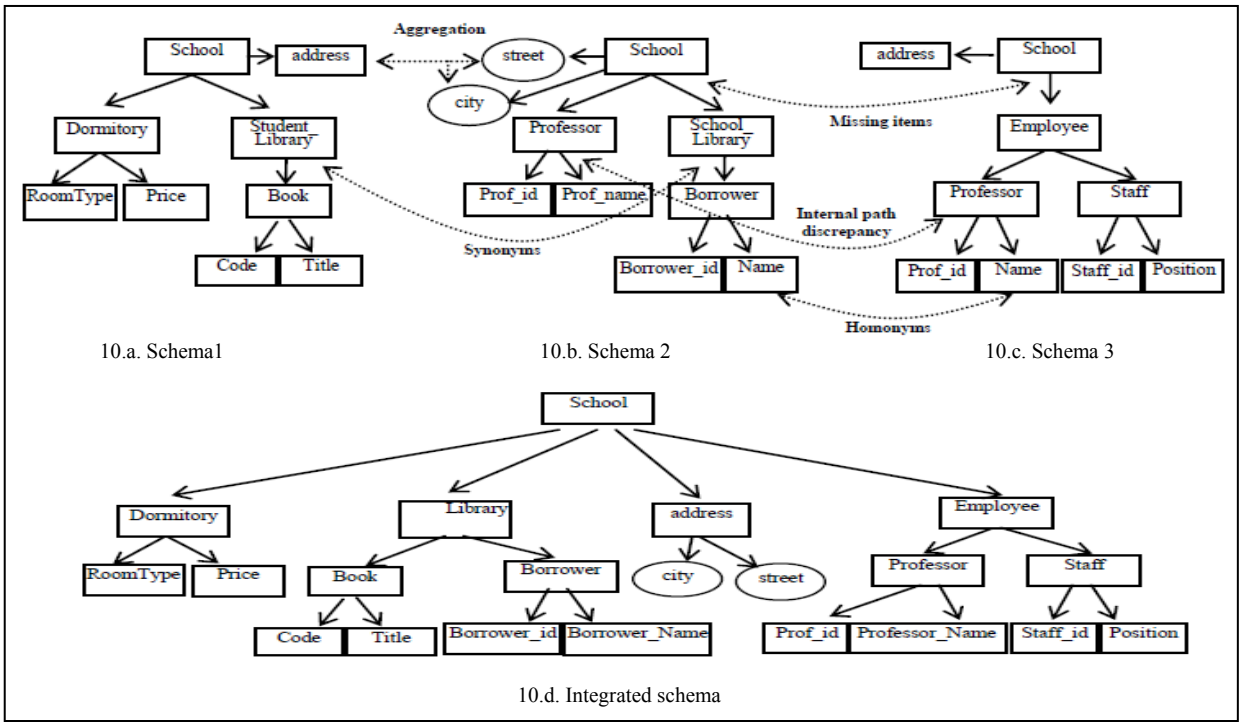
The *mediator* (Fig. 9.) corresponding to the schema integration component in our framework, includes twelve subcomponents, which are mostly used to solve conflicts. Besides cleaning and union schemas as well as normal form conversion [5], six types of conflicts are solved including synonym, homonym, data type, number occurring, missing item, and aggregation conflicts to produce an integrated schema. This integrated schema is then refined by removing redundant (duplicated) information.

The *query handler* processes a user query based on the integrated schema and returns integrated data to user. It consists of five components, namely query decomposition, query executor, data conversion, data cleaner, and data integration, which are used for decomposing and executing queries, converting, cleaning and integrating extracted data respectively. Underlying each subcomponent is a set of XET rules written for a specific task. Java is used for linking these independent components. Although the XET language lacks some features such as file operations, its extensive built-in functions along with its intrinsic simplicity and flexibility proved to be an overwhelming advantage for the rapid prototype. A graphical user interface (GUI) is also implemented using Java libraries to provide a visual human interaction with the system.

C. *Experiment Results*

In order to verify the effectiveness of our approach, test cases (Fig. 10.) have been designed containing many types of conflicts discussed in section III.B. 1) as follows:

- Synonym conflict: *Student_Library* (Fig. 10.a) vs. *School_Library* (Fig. 10.b).
- Homonym conflict: *name* (Fig. 10.b) vs. *name* (Fig. 10.c).
- Datatype conflict: *IDREF* (Fig. 10.a) vs. *IDREFS* (Fig. 10.c) of *address*.
- Number occurring conflict: *maxOccurrs="1"* (Fig. 10.a) vs. *maxOccurrs="2"* (Fig. 10.b) of *School_Library* (see XML source codes in [12]).



10.a. Schema1 10.b. Schema 2 10.c. Schema 3

10.d. Integrated schema

Fig. 10. XML schema integration usecase

[3] http://assam.cims.hokudai.ac.jp/eti

- Missing item conflicts: the subelements of *Library* (converted from *Student_Library* - Fig. 10.a) vs. the subelements of *Library* (converted from *School_Library* - Fig. 10.b), similarly to the element *School*.
- Aggregation conflict: *Address* (Fig. 10.a, Fig. 10.c) vs. *City, Street* (Fig. 10.b).

Moreover, unlike other approaches in which designed test cases normally follow a specific structure to be easily processed by a specific rule language, these test cases (schemas) are composed as schemas in real commercial applications are. These XML schemas can be arbitrarily declared to be nesting (Fig. 10.b), modulating (Fig. 10.c) or mixing (Fig. 10.a) which makes integration tasks more complicated. This problem has been discussed previously in [5]. Readers are referred to XML source codes in [12] for further details.

From the incoming schemas, we successfully produce an integrated schema (Fig. 10.d). For example, using a term dictionary and the rule for solving the synonym conflict (Fig. 6.), *Student_Library* (Fig. 10.a) and *School_Library* (Fig. 10.b) are replaced by *Library*. Homonym conflict, *name* (Fig. 10.b) vs. *name* (Fig. 10.c), is solved by tracking back to ancestor nodes (*Borrower* and *Professor*, respectively). The term *name* is therefore replaced by *Borrower_Name* and *Professor_Name.* Since *IDREF* is a special case of *IDREFS*, it is replaced by *IDREFS*, similarly to *maxOccurrs*. Missing item conflicts are resolved by combining all subelements and attributes of the same element from incoming schemas. All these conflicts are solved by the mediator consisting of XET rules. Note that as stated in the introduction of this paper, by using generalized rules, our integration framework can integrate not only three schemas but also *n* schemas at a time.

In addition to producing an integrated schema, we process a user query efficiently by using the query handler. A user poses a query conforming to the format of the integrated schema (e.g., *Professor_Name*). This query is decomposed into subqueries by the query decomposition component (similarly to Fig.7. and Fig. 8) and then routed to appropriate local data sources (e.g., *Prof_name* and *name* to the sources in Fig. 10.b. and 10.c, respectively) where they are executed (using the query executor component). Extracted data are then converted by the data conversion component and returned to the user in a user-friendly format conforming to the structure of the integrated schema (e.g., *Professor_Name*) [8]. These underlying processes are totally transparent to the user.

V. Conclusion

In this paper, we use XDD to model all data components and processing components of an XML database integration framework. This offers many advantages. First, all components of the system modeled by a unified rule-based language can communicate and exchange data easily. Second, a special structure for XDD-based bidirectional mappings is designed. With this kind of construction, mappings are combined with special rules to produce information efficiently for both query decomposition and data conversion, avoiding data redundancy. Finally, our framework is one among a very few rule-based integration frameworks that can integrate *n* participating schemas and also decompose a query into *n* subqueries at a time. It is worth emphasizing that most other researchers integrate schemas in a pair-wise fashion even though they know that the time complexity of their systems is high, approximately *(n-1)!* times to resolve conflicts and reconstruct mappings. The reason is that before XDD there were not many well-suited languages supporting the one-shot strategy. Further more, even though XDD is designed to model XML documents, it is an art to flexibly make use of XDD and its variables, especially *$E:expression*, to create a special and robust framework which is capable of processing *n* XML documents at a time.

In order to verify the effectiveness of our framework, a prototype has been built and tested [12]. With the current implementation, we have demonstrated that we are able to use XDD rules to produce a minimal and complete integrated schema from distributed XML schemas. Based on this integrated schema and a set of bidirectional mappings, our framework can process a global query efficiently and return user-friendly results.

References

[1] B. Ludascher, R. Himmeroder, G. Lausen, W. May, and C. Schlepphorst. “Managing semistructured data with FLORID: a Deductive object-oriented perspective”. *Journal of Information Systems*, Vol. 23, No. 8 (1998) 1-25.

[2] C. Anutariya, V. Wuwongse, and V. Wattanapailin. “An Equivalent-Transformation-Based XML Rule Language”. *Proc. of the International Workshop on Rule Markup Languages for Business Rules in the Semantic Web*, Sardinia, Italy (2002).

[3] C. Batini, M. Lenzerini and S. B. A. Navathe. “Comparative Analysis of Methodologies for Database Schema Integration”. *ACM Computing Surveys.* Vol. 18, No. 4 (1986) 323-364.

[4] D. AnHai and A. Y. Halevy. “Semantic integration research in the Database Community”. *AI Magazine, special issue on Semantic integration* (2005).

[5] D. D. Duong and V. Wuwongse. “XML Database Schema Integration Using XDD”. *Proc. of Advances in Web-Age Information Management Conference*, China. Springer Verlag, Vol. 2762 (2003) 92-103.

[6] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajamaran, Y. Sagiv, J. Ullman, V. Vassalos and J. Widom. “The TSIMMIS Approach to Mediation: Data Models and Languages”. *Journal of Intelligent Information Systems*, Vol. 8 (1997) 117-132.

[7] K. R. Bouguettaya and M. Parazoglou. “On Building a Hyperdistributed Database”. *Journal of Information Systems*. Vol. 20, No.7 (1995) 557-577.

[8] L. T. T. Thuy and D. D. Duong. “Query Decomposition Using the XML Declarative Description Language”. *Proc. of International Conference of Computational Science and Its Applications*, Singapore. Springer Verlag, Vol. 3481 (2005) 1066-1075.

[9] R. Vdovjak, F. Frasincar, G. J. Houben and P. Barna. “Engineering semantic web information systems in HERA”. *Journal of Web Engineering*, Rinton Press, Vol. 2(1&2) (2003) 003-026.

[10] S. Adali and V. S. Subrahmanian. “Amalgamating Knowledge Bases, II - Distributed Mediators”. *International Journal of Intelligent and Cooperative Information Systems(IJICIS)*, Vol. 3(4) (1994) 349-383.

[11] S. Busse, R. D. Kutsche, and U. Leser. ”Strategies for the Conceptual Design of Federated Information Systems”. *Engineering Federated Information Systems, Proc. of the 3rd Workshop EFIS* (2000) 23-32.

[12] The XSIS system. Available online http://v37s3b4h7dn47s37hg1br4h7rs7n3du7s8nu.unbf.ca/~b89ct/XSIS/index.html

[13] V. Wuwongse, C. Anutariya, K. Akama and E. Nantajeewarawat. "XML Declarative Description (XDD): A Language for the Semantic Web". *IEEE Intelligent Systems*, Vol. 16, No. 3 (2001) 54-65.

[14] V. Wuwongse, K. Akama, C. Anutariya and E. Nantajeewarawat. "A Data Model for XML Databases". *Journal of Intelligent Information Systems*. Vol. 20, No. 1 (2003) 63-80.

[15] W. May. "Logic-based XML data integration: a semi-materializing approach". *Journal of Applied Logic*, Vol. 3, No. 1 (2005) 271–307.