An XML Schema Integration and Query Mechanism System

Sanjay Madria[1], Kalpdrum Passi[2], Sourav Bhowmick[3]

[1] Dept. of Computer Science, University of Missouri-Rolla, Rolla, MO 65401, USA
madrias@umr.edu
[2] Dept. of Math. & Computer Science, Laurentian University, Sudbury ON P3E2C6, Canada
kpassi@cs.laurentian.ca
[3] School of Computer Engineering, Nanyang Technological University, Singapore
assourav@ntu.edu.sg

Abstract. The availability of large amounts of heterogeneous distributed web data necessitates the integration of XML data from multiple XML sources for many reasons. For example, currently, there are many e-commerce companies, which offer similar products but use different XML schemas with possibly different ontologies. When any two such companies merge, or make an effort to service customers in cooperation, there is a need for an integrated schema and query mechanism for the interoperability of applications. In applications like comparison-shopping, there is a need for an illusionary centralized homogeneous information system. In this paper, we propose XML Schema integration and querying methodology. We define an object-oriented data model called XSDM (XML Schema Data Model) and present a graphical representation of XML Schema for the purpose of schema integration. We use a three-layered architecture for XML Schema integration. The three layers included are namely *pre-integration*, *comparison* and *integration*. The three layers can conceptually be regarded as three phases of the integration process. During *pre-integration*, the schemas present in XML Schema notation are read and converted into the XSDM notation. During the *comparison* phase of integration, correspondences as well as conflicts between elements are identified. During the *integration* phase, conflict resolution, restructuring and merging of the initial schemas takes place to obtain the global schema. We define integration policies for integrating element definitions as well as their datatypes and attributes. An integrated global schema forms the basis for querying a set of local XML documents. We discuss various strategies for rewriting the global query over the global schema into the sub-queries over local schemas. Their respective local schemas validate the sub-queries over the local XML documents. This requires the identification and use of mapping rules and relationships between the local schemas.

Keywords: XML, Schema, Integration, query

1. Introduction

The integration of heterogeneous data sources has become a central problem of modern computing [8]. Data integration involves data from a variety of applications, repositories and legacy systems. With the advent of improvements to Internet technologies, there has been a greater demand on the integration of data from diverse sources, especially in e-business where companies need to connect their online systems with those of their suppliers. Besides E-commerce, integration of data from different sources is required when two or more organizations merge. In recent times web sites are linked to various sources of data which necessitates integration of data sources.

The availability of large amounts of XML data necessitates the integration from multiple XML sources for many reasons. Each organization or application creates its own document structure according to specific requirements. These documents/data may need to be integrated or restructured in order to efficiently share the data with other applications. Interoperability between applications can be achieved through an integration system, which automates the access to heterogeneous schema sources and provide a uniform access to the underlying schemas.

In e-commerce applications, XML documents can be used to publish any data ranging from product catalogs and airline schedules to stock reports and bank statements. XML forms can be used to place orders, make reservations, and schedule shipments. XML eliminates the need for custom interfaces with every customer and supplier applications, allowing buyers to compare products across many vendors and catalog formats, and sellers to publish their catalog information to reach many potential buyers.

XML enables online businesses to build on one another's published content and services to create innovative virtual companies, markets and trading communities. With a global view of the Internet-wide shopping directories, a query system can locate all merchants carrying a specific product or service, and then query each local schema in parallel to locate the best deals. The query system can use the integrated schema and can sort the offers according to criteria set by the buyers–the cheapest flight, the roomiest aircraft, or some weighted combination. Other examples where integrated view is useful are given below.

- When companies merge or endeavor to service customers jointly, their local schemas need to be merged to provide an integrated view of the data present with the companies and to enable conflict-free information sharing and retrieval.
- Applications like comparison-shopping that have to retrieve the data from different heterogeneous data sources and compare the prices and specifications of various items have a need for the integrated view of all the sources and a query mechanism.
- Any application that needs to interact with data from two or more XML sources need to have an integrated view of the schemas of their local schema and a mechanism to retrieve the data from different sources.

There are two popular styles for integrating heterogeneous sources, data integration and schema integration. During data integration the physical data from the heterogeneous sources is combined. However, during schema integration the data is not touched but rather the schemas of the sources are combined. In either case, the goal is to provide a uniform interface to a multitude of data sources. To mask the heterogeneity, a mediator presents a unified context to users. One of the key advantages of integration is that it frees the user from having to locate and interact with every source, which is related to their query.

For seamless information access, the mediation systems have to cope with different data representations and search capabilities [9]. A mediator presents a unified context for uniform information access, and consequently must translate original user queries from the unified context to a target source for native execution [7]. This translation problem has become more critical now that the Internet and Intranets have made available a wide variety of disparate sources, such as multimedia databases, web sources, legacy systems, and information retrieval (IR) systems. Integrating a number of heterogeneous sources [29] is difficult in part because each source has its own set of vocabulary and semantics, which can be used when formulating queries. Hence, the query processor needs to be able to efficiently collect related data from multiple sources, minimize the access to redundant sources, and respond flexibly when some sources are unavailable. To ensure semantic interoperability, information must be appropriately mapped from its source context to its target context where it will be used [8]. For this reason, mapping rules and algorithms must be created to ensure a query is rewritten properly.

The currently available integration systems for semistructured data [1, 15, 8, 16, 17, 31] use the approach where they integrate the data by using mediated schemas to reformulate queries on the disparate data sources for the purpose of integration. For every instance of data integration, the user's query must be reformulated onto the local sources and executed to present the required results to the user in a unified perspective. Mappings need to be defined to rewrite queries on the disparate sources of data to deal with semantic differences between the various sources.

In our approach, the schemas of the local XML documents need to be integrated to obtain a Global integrated schema. Schema integration process includes the resolution of different conflicts such as naming conflicts, datatype conflicts, structural conflicts and key conflicts. The Global schema i.e., the integrated schema obtained by integrating the local schemas validate all the documents validated by the local schemas. The documents that are created after the integration process are validated by the global schema but may not be validated by any of the local schemas. In our XML integration approach, the semantic differences between different sources are taken care of through the schema integration process along with ontology. The user has a unified view of the data in the form of the global schema on which he/she can query different sources of XML documents. A major advantage of creating a global schema is that integration of local schemas occurs only once or when the local schemas are modified instead of integration of data taking place at every instance of the query. This allows for more efficiency. Any changes in the local sources in turn require changes in the reformulation of the queries, whereas in the schema integration strategy used in our system, once changes in local schemas are reflected in the global schema, they are visible to the user. Our integration system uses XML Schema [32] language for integration of XML data.

In this paper, in addition, we present a query mechanism where the global query issued by the user on the global integrated scheme is converted into the local queries to be executed on the local schemas. The results returned are then integrated before presenting to the user. An integrated schema forms the basis for a valid query language over a particular set of XML documents. Knowing the global data structure of all documents helps validate potential queries of the data set. A user query formed on the global schema must be rewritten on the local schemas that validate the local XML documents and the results presented in a unified form to the user. The integration of data is simple in the form of conjunction of resultant data from local XML documents. The query rewriting process requires a repository of mapping rules on the local schemas. The mapping rules can be generated during integration process. In this paper, we present the mapping rules and strategies for rewriting queries on the local schemas for different cases of global schema. In our integration system, we have adopted XQuery [4] language for writing queries on the XML instance documents where global integrated schema is given to the user as an input.

1.1 Running Example

The takeover of Canadian Airlines by Air Canada in 2000 has demonstrated just how messy data integration can be. In addition to the widely publicized problem of passenger and flight data integration, Air Canada and Canadian Airlines each used their own method for tracking the inventory and maintenance schedules of ground support equipment (GSE), including baggage handling devices such as tow trucks. As of April 2001, Air Canada's GSE [18] tracking continued to employ separate Access and DOS based database systems, due in part to the cost of integrating the systems into one platform. If both had been written in applications that could create XML documents, it would have been much easier and cost-effective to integrate the documents XML Schemas, and in turn, the actual GSE data. More time could be spent on preventative maintenance leading to better baggage handling and less lost luggage, and less time spent making sure that data is correctly entered for *this* vs.

that system. As well, such an integrated view of their complete GSE inventory would help Air Canada to place equipment from the same vendor or having similar fuel requirements at the same airport terminal. This would lower service, fuel and equipment loss costs.

Throughout the rest of the paper we shall use the *example* of the integration of GSE data of Air Canada and Canadian Airlines. Figure 1 shows sample XML documents pertaining to GSE for Air Canada and Canadian Airlines and Figure 2 shows their corresponding XML Schemas. The idea is to create a global schema to achieve interoperability among the existing XML documents mapping the local schemas to a single global view.

```
<?xml version="1.0" ?>
<gs_equipment xmlns="http://www.GSE1example.org"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.GSE1example.org GSE1.xsd">
    <machine type="baggage_handler">
        <supplier>Air to Ground</supplier>
        <serial_number>FRD6754</serial_number>
        <service_agreement>
            <expiry_date>01-01-2006</expiry_date>
        </service_agreement>
        <service_hours>345</service_hours>
    </machine>
    <location>
        <airport>Vancouver</airport>
        <terminal>6A</terminal>
     </location>
</gs_equipment>
```

```
<?xml version="1.0" ?>
<gs_equipment xmlns="http://www.GSE2.example.org"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.GSE2example.org GSE2.xsd">
  <placement>
        <airport>Winnipeg</airport>
        <terminal>main</terminal>
  </placement>
  <machine type="tow_truck">
      <serial_number>123456145</serial_number>
      <vendor>Quick as a Jet GSE</vendor>
      <service_agreement>QJ-TT-123456145-September 2003
                                                   </service_agreement>
      <service_hours>1090.75</service_hours>
   </machine>
</gs_equipment>
```

Figure 1. Sample XML documents, GSE1 of Air Canada and GSE2 of Canadian Airlines

2. Related Work

The problem of schema and integration of heterogeneous and federated databases has been addressed widely. Several approaches to schema integration exist as described in [5, 2, 7, 13, 15, 17, 31, 26, 27, 29]. A global schema in the general sense can be viewed as a regular schema, the rules of which encompass the rules of a common data model. A global schema eliminates data model differences, and is created by integrating local schemas. The creation of a global schema also helps to eliminate duplication, avoid problems of multiple updates and thus minimize inconsistencies.

The relational or functional model often ignores the possibility of name conflicts and contradictory specifications [14]. The semantic model is equipped to deal with more conflicts than the relational model. The object-oriented model allows one to define not only data elements, but also operations/methods associated with each type of data. The various methodology for schema integration use different data models, such as relational, functional, semantic, object-based and more recently XML based towards a solution to the problem of integrating heterogeneous data sources.

Generally, view integration is defined as the process, taking place in database design, which produces a global conceptual description of a proposed database [27]. Database integration is used to create a federated database management system. First, a canonical model is defined to overcome data model heterogeneity, then schema integration is performed to solve schematic heterogeneity. Both involve the "activities of integrating schemas of existing or proposed databases into a global unified schema which satisfies the constraints imposed by all component schemas" [5].

```
<?xml version="1.0"?>
<schema
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
targetNamespace="http://www.GSE1example.org"
elementFormDefault="qualified"
xmlns:GSE1="http://wwwGSE1example.org>
<element name ="gs_equipment">
  <complexType>
    <sequence>
      <element ref="GSE1:machine" minOccurs="1"
                                  maxOccurs="1"/>
       <element ref="GSE1:location"
                  minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>
</element>
<element name ="machine”>
  <complexType>
   <sequence>
     <element name="supplier" type=
                  "xsd:string" minOccurs="1"
                              maxOccurs="1" />
     <element name="serial_number"
              type="xsd:string" minOccurs="1"
                              maxOccurs="1" />
     <element ref=”GSE1:service_agreement"
                minOccurs="1" maxOccurs="1" />
     <element name="service_hours"
             type ="xsd:integer" minOccurs="0"
                              maxOccurs="1" />
     <xsd:attribute name="type"
                              use="required">
      <xsd:simpleType>
       <xsd:restriction base="xsd:string">
          <xsd:enumeration value
                         ="baggage_handler"/>
          <xsd:enumeration value
                         ="boarding_stairs"/>
       </xsd:restriction>
      </xsd:simpleType>
     </xsd:attribute>
   <sequence>
  </complexType>
</element>
<element name ="service_agreement”>
  <complexType>
    <sequence>
       <element name="expiry_date"
                type="xsd:date" minOccurs="1"
                              maxOccurs="1" />
    </sequence>
  </complexType>
</element>
<element name ="location">
  <complexType>
    <sequence>
       <element name="airport"
               type="xsd:string" minOccurs="1"
                              maxOccurs="1" />
       <element name="terminal"
               type="xsd:string" minOccurs="1"
                              maxOccurs="1" />
   </sequence>
  </complexType>
 </element>
</schema>
```

```
<?xml version="1.0"?>
<schema
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
   targetNamespace="http://www.GSE2example.org"
   elementFormDefault="qualified"
   xmlns:GSE2="http://wwwGSE2example.org>
<element name ="gs_equipment”>
  <complexType>
    <sequence>
      <element name="GSE2:placement”
               minOccurs="1"  maxOccurs="1" />
      <element ref="GSE2:machine"
                 minOccurs="0" maxOccurs="1"/>
    </sequence>
  </complexType>
</element>
<element name="placement">
  <complexType>
    <sequence>
      <element name="GSE1:airport"
                minOccurs="1" maxOccurs="1" />
      <element name="GSE1:terminal"
                minOccurs="1" maxOccurs="1" />
    </sequence>
  </complexType>
</element>
<element name ="machine">
  <complexType>
    <all>
       <element name=”vendor”
               type=”xsd:string” minOccurs=”0”
                               maxOccurs=”1”/>
       <element name="service_hours"
              type="xsd:decimal" minOccurs="0"
                               maxOccurs="1" />
       <element name="serial_number"
                type="xsd:positiveInteger"
                minOccurs="0" maxOccurs="1" />
       <element name="service_agreement"
            type ="xsd:string"  minOccurs="0"
                              maxOccurs="1" />
    </all>
    <xsd:attribute name="type" use="optional">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value
                          ="baggage_handler"/>
            <xsd:enumeration
                         value="tow_truck"/>
          </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
  </complexType>
</element>
</schema>
```

Figure 2. XML Schema documents for the sample XML documents

Most schema integration approaches decompose integration into a multi-layered architecture like the one followed in this paper constituting *pre-integration*, *comparison* and *integration* [5, 21]. There have been some systems [1, 3, 25, 33] that integrate data from multiple sources. Most of these systems provide a set of mediated/global schema(s). Some systems like *Garlic* [34, 30] use wrappers to describe the data from different sources in its repositories and provide a mechanism for a middleware engine to retrieve the data. The *Garlic* system also builds global schema from the individual repositories. The *comparison* and *restructuring* phase of integration is handled in some systems through human interaction using a graphical user interface as in Clio [22, 23, 16] and in others semi-automatically through machine learning techniques such as in *Tukwila* data integration system [31]. The *Tukwila* integration system reformulates the user query into a query over the data sources, which are mainly, XML documents corresponding to DTD schemas and relational data. Tukwila uses an x-scan operator that can query streaming XML data. Tukwila x-scan matches regular path expression patterns from the query, returning results in pipelined fashion as the data streams across the network. XML integration in YAT system [7] is considered as a query mechanism on the local documents and algebra has been defined to query local documents for integration. Source capability based query re-writing is done to achieve optimization. Nimble Technology's [13] integration system is also based on XML-like data model at the system's core.

Most of the above integration systems integrate the documents using query mechanism or document restructuring. The data sources are queried according to a required given mediated schema and the results obtained are integrated. A mediated schema is created to represent a particular application domain and data sources are mapped as views over the mediated schema. The data is integrated using SQL type queries on the instance documents, which are broken into multiple fragments based on target sources. The main difference between a mediated schema and a global schema is that in a mediated schema data integration is done through queries and for every instance of data integration the query must be reformulated into multiple queries on the local sources, whereas in a global schema the schemas are integrated once, not for every instance of data integration. In case of a global schema a user simply posts a query on the global schema. The translation of this query onto to the local sources is transparent to the user. In both approaches, queries have to be translated to be evaluated against local data sources. The main difference is that the query translation is simpler in our case because there is no restructuring and there is a very direct relationship between the global schema and the local schema, whereas in a mediated approach the difference between the mediated representation/schema and the local sources is very important.

Our approach is to create an integrated global schema from the local schemas and use the integrated schema for the applications to work. In our approach, a user can query global schema and results are retrieved from the local documents. The fragments are translated into the appropriate query language for the target sources. The automated integration of XML schemas is beneficial to both the traditional form of view integration and database integration. An integrated schema forms the basis of valid query language over a set of XML documents. Knowing the global data structure of all the documents helps validate potential queries over the data sets. A global schema eliminates data model differences and it helps eliminate duplication, avoid problems of multiple updates and thus minimize inconsistencies. These issues are very important in XML domain, as elements are repeated and can be updated at multiple places, which can cause inconsistencies in the documents. For this reason, we present a dynamic mechanism, which can interface the different XML data and can present an integrated representation of the XML sources, rather than physically integration of data. The integration or to add a new schema has to be done in an incremental fashion. Systems such as Tukwila and Clio are definitely faster as they use data mining and machine learning techniques to speed up the system, to decide the possible approximate matches and whereas we rely on human experts to focus on efficiency and accuracy of the system rather than on speed. We believe that accuracy in data integration is much more desired than faster speed. Having said that, such techniques can be integrated with our system.

In [19], the goal is to rewrite a query in order to reduce the number of database relation literals in the rewritten query. Yang and Larson [20] consider the problem of finding rewritings for select-project-join queries and views. In their analysis, they consider what amounts to one-to-one mappings from the view to query, and do not search the entire space of rewritings. Much effort has been put into query rewriting in the terms of views. Several authors have considered the problem of implementing a query processor that uses the results of materialized views [19]. Views are often used to describe source contents. Furthermore, the different and limited query capabilities of the sources are often described by "views" where the constants are parameterized [28]. The problem of answering queries using views is to find efficient methods of answering a query using a set of previously materialized views over the database, rather than accessing the database relations [26]. In this case, a query over a database schema or data sources must be rewritten in terms of a given set of views, which have been defined over the same schema. In this paper, we address the semantic mappings of constraints, or in other words the translation of vocabulary, through the use of query rewriting. The queries on the source XML documents using the global schema are rewritten onto the local schemas through mappings that relate the local schemas to the global schema.

3. System Architecture and Schema Integration Model

The system architecture is depicted in Figure 4 and is similar to many integration systems [1, 15, 16] with the key distinction that our system is based on XML Schema and XQuery language. During *phase 1* of the integration process the local XML Schemas are integrated by the integration engine as in Figure 3 and a global schema made available to the user or application. The integration of local XML Schemas is done through *Integration Builder* tool. A user can query different XML documents that are validated by their respective local XML Schemas by having an integrated view in the global schema. The integration of the schemas is done only once unlike the other systems [1, 3, 25, 33] where each time a query is posted to the system it has to be reformulated into multiple queries on the local database systems and the data integrated. In our system, the user has access to the global schema based on which the XML documents are queried without any knowledge to the user of distributed instance documents across platforms. The global schema can be rebuilt incrementally once a local schema has changed or another schema added to the database. This technique is in conformance with "build-once run-any-number-of-times". This separates the integration of the schemas from the querying and simplifies the design of integration system, leads to a more efficient maintenance of the system.

The queries are posed in XQuery and are submitted to the query engine. When a query based on the global schema is posed to the query engine it is parsed and rewritten into multiple queries based on the local XML Schemas that validate the instance documents. This constitutes *phase 2* of the integration process. The translation into the local queries is carried out through a mapping file that is generated during phase 1 of integration process as shown in Figure 4. In phase 3 of the integration process, the mapped queries onto the local schemas are executed on the instance documents and the results of the queries integrated and displayed to the user in the form of an integrated output document. The Query Builder provides the interface to post an XQuery based on the global schema to the query engine and the output is displayed as an XML file. The output can be formatted to display in a browser through the application of XSLT. We also provide some offline tools for system management. The *ontology checker* is provided to check and modify the ontologies of the local schemas. The *data administrator* tool can be used to view the mapping file and to make the changes if necessary.

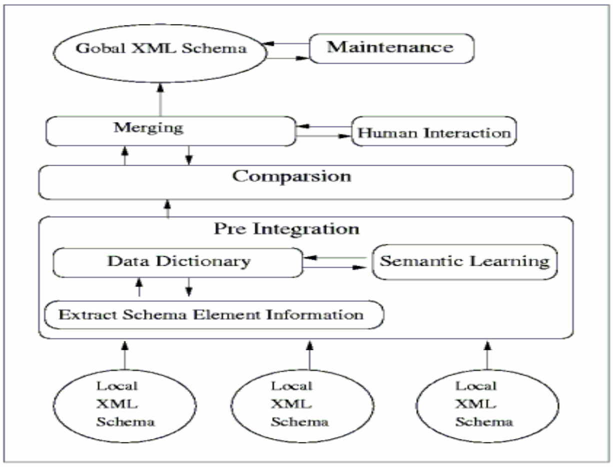


Figure 3. XML Schema Integration Engine

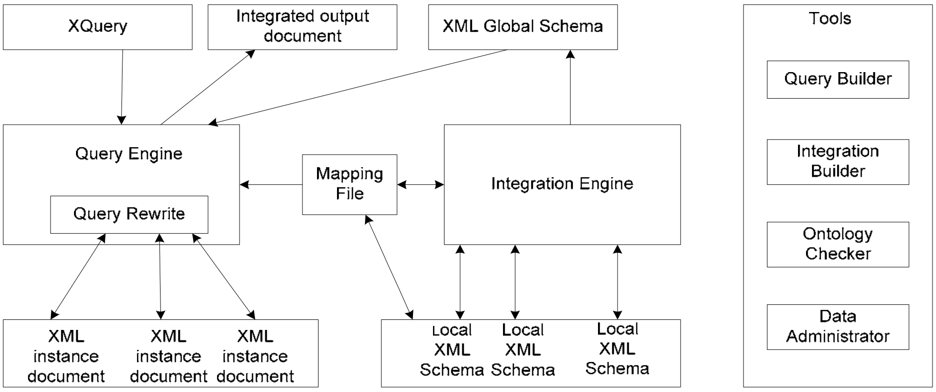


Figure 4. XML System Integration Architecture

3.1 Requirements of Integrated Global Schema

Any global integrated schema must meet the following three criteria: *completeness*, *minimality*, and *understandability* [5]. The definitions of *completeness and correctness*, *minimality*, and *understandability* as given in [5] are given below.

Completeness and correctness: The integrated schema must contain all concepts present in any component schema correctly. The integrated schema must be a representation of the union of the application domains associated with the schemas.

Minimality: If the same concept is represented in more than one component schema, it must be represented only once in the integrated schema.

Understandability: The integrated schema should be easy to understand for the designer and the end user. This implies that among the several possible representations of results of integration allowed by a data model, the one that is (qualitatively) the most understandable should be chosen.

In order to meet the first, *completeness*, all the elements in the initial schemas should be in the merged schema. The merged schema can be used to validate any of the XML instance documents that were previously validated by one of the initial schema specifications. The global schema represents as much information as the initial schemas. In other words, the resultant merged schema must retain the information capacity of the initial schemas [26]. As well, a complete schema exhibits conflict free-ness [17]. The merge rules that we will define in the integration process for element definitions should be such that the element definitions in the initial schema are still valid. If the global schema is incomplete, some of the XML documents may not be validated and this indirectly will result in the loss of data due to inaccessibility.

To satisfy the second criterion, *minimality*, each unique element should be defined only once in the schema. Redundancy should be eliminated wherever possible through the identification of equivalent elements and attributes, and the subsequent use of substitution groups. Datatypes for terminal elements should be expanded through the use of constraint facet redefinition [8], or unions of incompatible datatypes, only to the point necessary to satisfy boundary conditions (taking the minimum scale that satisfies both the datatypes). Optionality of elements (i.e. minOccurs and maxOccurs values) is expanded to meet boundary restrictions only.

Finally, to comply with the third criterion, *understandability*, in the case of XML Schema integration, the global schema must be formulated in a referenced style, rather than an inline style (nested definitions) [32], for ease of reading and assessment.

3.2 XML Schema Integration Methodology

There are four approaches listed in [5] for integrating schemas. The integration process could be *binary* where two schemas at a time are integrated or *n-ary* where more than two schemas are integrated at the same time as shown in figure 5. Binary processes have two styles. The *ladder* style integrates two schemas at a time, always integrating one initial schema with the integrated schema to date. The *balanced binary* style integrates initial schemas in pairs and continues to do so with resultant mid-point integrated schemas until the global schema integrated schema is produced. *N-ary* integration processes also have two styles. The *one shot n-ary* style integrates all the initial schemas at once. The *iterative n-ary* style integrates groups of arbitrary size of initial schemas at one time, continuing in a ladder fasion until the global integrated schema is produced.

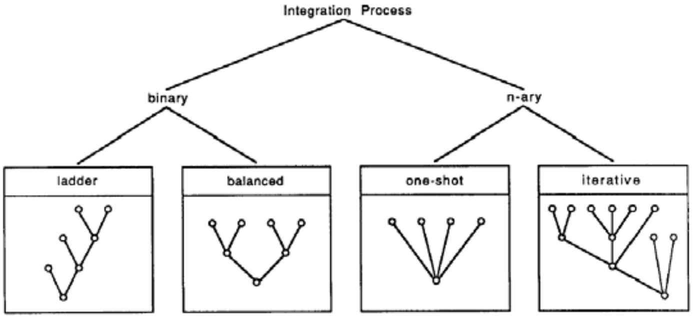


Figure 5. Types of integration-processing strategies

The XML Schema integration process given in this paper uses a *one shot n-ary* strategy. The *one shot n-ary* style integrates all the initial schemas at once. For ease of understandability, only two schemas are merged together in the examples throughout this paper. However, the actual integration of the individual element definitions themselves occurs in more of a ladder style where comparisons and subsequent integration occurs between two element definitions at a time with the merged element to date then being compared to and merged with the other definitions for the given element in the remaining schemas.

Schema integration should be both *extensible* and *scalable* [23]. It should be easy to add or remove sources of data (i.e. schemas), to manage large numbers of schemas, and to adjust the resulting global schema. With the XML Schema integration approach described here, multiple schemas can be integrated at one time. While the resulting schema is easy to adjust, this must be done with caution, as the element definitions described within it are used to validate existing XML documents. An initial merging can be automated, but human interaction is a must for further adjustments such as in synonym conflict resolution for terminal elements.

Our approach to XML Schema integration involves an initial *pre-integration* step. Here element, attribute and datatype definitions are extracted through parsing of the actual schema document. Then, for each element, *comparison* and *merging* occurs. In the final step, the merged schemas are transformed into a human readable global XML Schema document. During *pre-integration*, initially the schema of the XML documents present in the XML Schema notation should be read and must be converted into the XSDM (see section 3.3) notation and an analysis of the schemas occurs [5]. Priority of integration must be determined if the process is not to be *one shot*. Preferences may be given to the retaining of entire or certain portions of schemas as whole parts of the global schema.

For the XML Schema integration process described herein, one schema is not given priority over another. Though elements are integrated one at a time, the schemas are merged in a way so that the children are first merged and then the parent nodes. Ultimately, a terminal is integrated before a non-terminal. The complete set of terminals will be merged before the complete set of non-terminals is merged, but a given non-terminal may be merged before a non-related (non-child) terminal.

During the *comparison* stage of integration, correspondences as well as conflicts between elements should be identified and resolved. This can be done by using either the semantic learning [12] or by using user interaction. The graphical representation must be shown to the user in a GUI environment and the user selects the corresponding attributes and elements from both the schema. The fundamental activity in the *comparison* phase of integration is *conflict resolution* [5]. Conflict identification and resolution is central to successful integration. *Naming conflicts*, *datatype conflicts* & *scale differences*, and *structural conflicts* can occur during XML Schema Integration.

In the Integration phase, integration of the two schemas actually occurs using the conflicts that are resolved in the comparison phase. After integrating the two schemas, the result is obtained in the XSDM notation that is followed through out the integration process. The global schema thus obtained in the XSDM notation is transformed in to the XML Schema notation, which is required.

3.3 XML Schema Data Model (XSDM)

The XML Schema structures are both syntactic and partially semantic in nature. Syntactically, they contain the structure and type of the data they are used to describe. Semantically, their structures, constraints, and namespaces allow partial inference of the meaning of the data that they describe. XML Schemas follow strict semantic and syntactic guidelines, thus making them easier to interpret, and as a result, to integrate.

The XML Schema integration process described herein adapts the essentially flat text based semantic schema document into an object oriented model of nodes, datatypes, namespaces and operations that allow easy, automated, computer based comparisons of elements, conflict resolution and, ultimately, a merged global schema which can be transformed back into a flat text based XML Schema document.

We define three structures for use in the XML Schema integration process – *Node Object*, *Child Object*, *Datatype Object* and *Attribute Object*.

- A *Node Object* represents an element, which may be either non-terminal or terminal in nature. Each Node has the following set of structures:
    - Name – a string which represents the name of the element
    - Namespace – identifies the schema which contains the definition of the element
    - Ancestral Namespace – identifies schemas with which a merged node was identifed
    - Max Occurrence – the maximum times the named element can appear
    - Min Occurrence – the minimum times the named element can appear
    - Attribute – a node may have attribute(s)
    - Datatype – terminal elements are defined in terms of a datatype structure
    - Substitution Group Name(s) – alternate names for the same element
    - ChildList – a list of Child structures which define a node
- *Datatype Object* represents datatypes of the terminal nodes. Attributes may have an associated datatype. Each *datatype Object* has the following structures:
    - Name – a string by which the datatype may be referred to in a schema, e.g., decimal, money
    - Variety – datatypes can be atomic (one instance of one kind), union (instances of two or more distinct datatype kinds), or a list of atomic or union datatypes (two or more instances of related datatype kinds)
    - Kind – there are 43 simple and derived datatypes represented by the XML Schema Datatypes [6]
    - Constraining Facets – the following 15 values are used to constrain a datatype: Length, MinLength, MaxLength, Enumeration, Whitespace, Pattern, MaxInclusive, MinInclusive, MaxExclusive, MinExclusive, Precision, Scale, Duration, Period and Encoding
- An *Attribute Object* represents attributes that may be associated with a non-terminal or a terminal element. Each Attribute has the following structures:
    - Name – a string by which the attribute may be referred to in the schema
    - Namespace – identifies the schema which contains the definition of the element
    - Form, - attributes may have a *qualified*, *unqualified* or *not stated* form
    - Use – attributes may be prohibited, fixed, required, optional or default
    - Datatype – attributes have associated datatypes
    - Value – a default value may be given to an attribute

3.4 Graphical Representation of XML Schemas

XML Schemas can be represented graphically as a modified directed graph where each element should appear only once and elements are modeled according to their relationship with their children if they are non-terminal nodes. In a completely integrated schema, the children of any two elements with the same name are identical as the merging is done bottom-up. The conflicts have been resolved for all the children and they have been merged. The following rules are defined as a mean to model the graphical representation of XML Schema structures.

- The name of a given *Node* or *Child* as defined in XSDM is contained within a rectangle.
- A unique element no is assigned to all the elements present in the schema and the element number is shown in the left bottom corner of the top rectangle.

- In cases where the element is a child of another element, the minimum and maximum numbers of times it may appear are indicated. The number in the right hand bottom corner of the rectangle indicates the minimum times that the element may occur. The number in the right hand top corner indicates the maximum number of times that the element may occur. The infinity symbol in the right hand top corner indicates that the maxOccurs is unbounded.
- If present, the namespace prefix [32] associated with an element may be recorded in the top left-hand corner of the rectangle.
- The symbol in the bottom most rectangle indicates the structure [32] of the element. The symbol 'S' represents a *sequence*; a 'C' represents a *choice*; an 'A' represents an *all*. A capital "E" represents a terminal element that is defined as *empty*. A capital "T" represents a terminal element that contains data (i.e. not empty). A capital "N" represents an *any*. The presence of "-m" beside the symbol indicates that the element may have mixed content.
- The element that appears at the top of the tree is referred to as the *root element*. Each schema, whether it is a DTD or XML Schema, may have only one root element [32].
- Lines connect a given element to its child(ren). In cases where an element is defined in terms of a *sequence*, the sequence is denoted in the adjacent flower braces present after 'S'.
- The attributes for any element if present are shown in the same rectangle that shows the element name.

The figure 6a shows the graphical representation whereas Figure 6b shows the implementation view of the schemas.

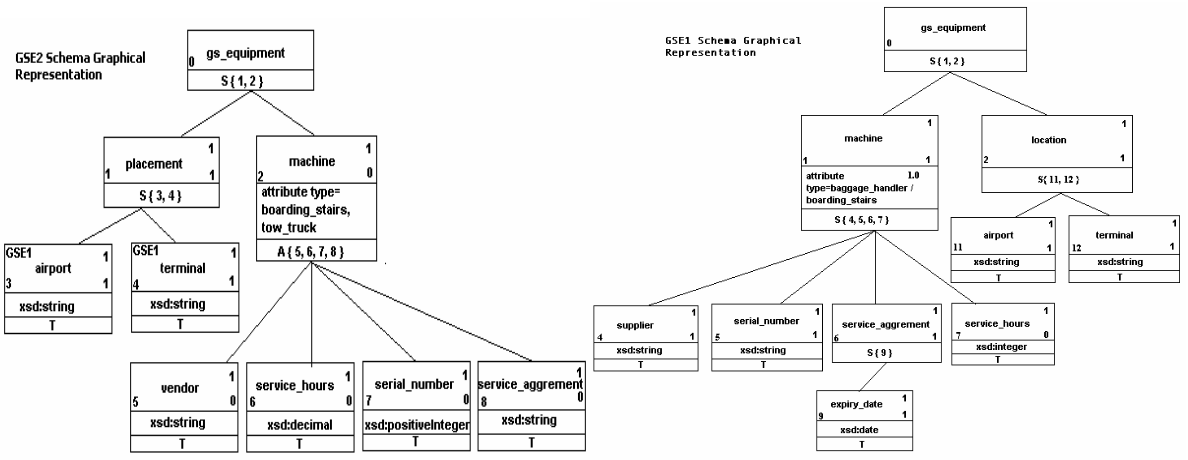


Figure 6a. Graphical representation of the sample XML Schemas

4. Issues in Integration of Schemas

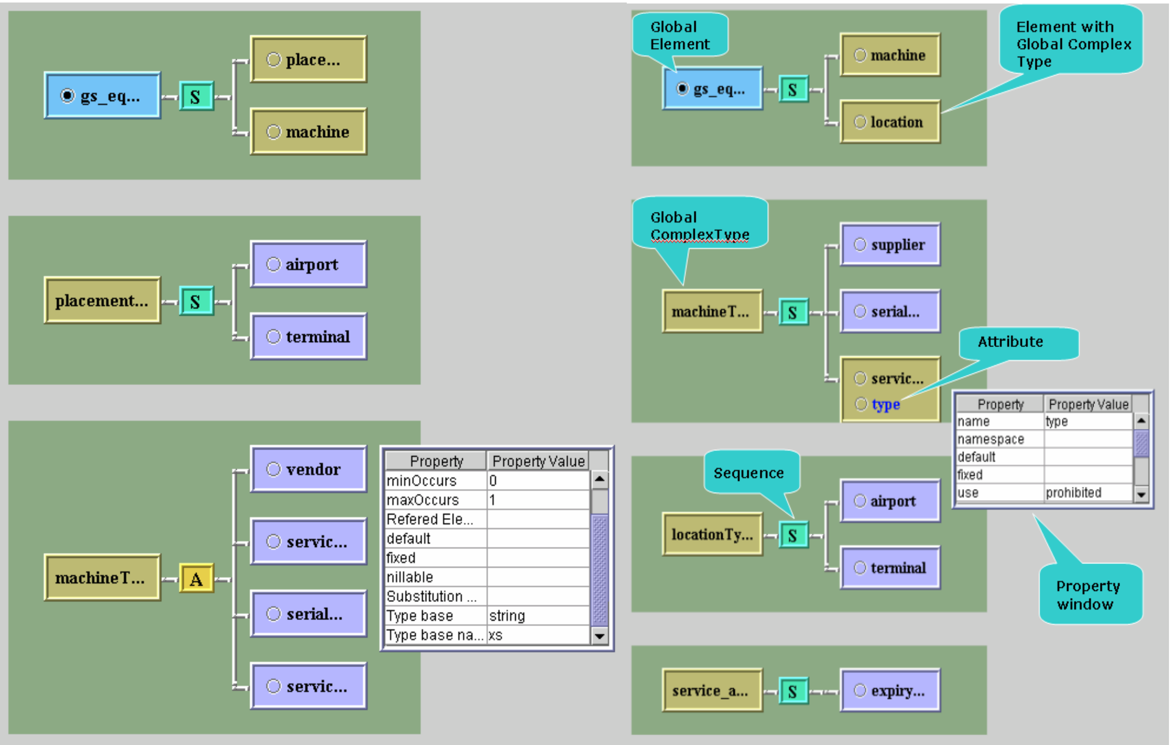
In this section, we discuss research issues in integrating schemas.

Pre-integration: The tasks in the pre-integration phase include parsing the given XML Schema documents and converting the schema into a tree-like structure using the XML Schema Data Model (XSDM). The parser "walks" the tree of document nodes in an XML Schema document and provides with a set of elements, attributes and constraints that are used in building the tree structure. These are deployed by the application in the process of schema integration.

Comparison: In the comparison process the user is provided with the Graphical User Interface (GUI) that enables the user to identify the correspondences between the data entities present in the schemas. The schemas are represented in the graphical representation for easy understandability of the schemas during the integration process for the user.

Integration: During schema integration, initial schemas are superimposed onto each other to result in a merged global schema. The elements and attributes that are obtained after resolving the conflicts present between the corresponding data entities are used in the global schema. The merged schema should be complete and minimal. The global schema is rearranged to ensure the highest level of minimality and understandability.

Representing and storing the integrated schema: The integrated schema obtained after the integration process is represented in the form of a tree, which consists of integrated elements belonging the global schema. The schema is represented in the graphical representation and should be displayed for the user so that it helps users during the integration phase. Finally, the obtained tree structure present in XSDM notation is converted into XML schema notation and should be stored in a schema document file.



Property	Property Value
minOccurs	0
maxOccurs	1
Refered Ele...	
default	
fixed	
nillable	
Substitution ...	
Type base	string
Type base na...	xs

Property	Property Value
name	type
namespace	
default	
fixed	
use	prohibited

Figure 6b. XSDM Implementation View of Air Canada and Canadian Airlines Schemas

4.1 Conflict resolution during integration of elements/attributes

During the *comparison* stage of integration, correspondences as well as conflicts between elements are identified. There are four *semantic relationships* defined in [5]. The schematic representations can be viewed as *identical*, *equivalent*, *compatible* or *incompatible*. We identify six types of semantic relationships, which apply to XML Schema elements – *identical*, *equal*, *equivalent*, *subset*, *unique*, and *incompatible*. Let S be the set of all nodes across schema(s). Let x, y be nodes in S. We first define the structure of a node x.

Definition 1: Structure of a node x is defined as a 4-tuple <nodetype, attribute, datatype, childList>, where nodetype, attribute, datatype and childList are structures of a node as defined in the XSDM model in section

4.2.1. Further, x.structure = y.structure iff (x.nodetype = y.nodetype) $\wedge$ (x.attribute = y.attribute) $\wedge$ (x.datatype = y.datatype) $\wedge$ (x.childList = y.childList).

Next we give definitions of the six semantic relationships.

Definition 2: x.***identical***(y) iff (x.name = y.name) $\wedge$ (x.namespace = y.namespace). Elements that have the same name and namespace are *identical*. Each namespace is unique and each element name within a given namespace is unique. Therefore, two elements with the same name and namespace must be the same element.

Definition 3: *x.**equal**(y)* iff (x.name = y.name) $\wedge$ (x.structure = y.structure) $\wedge$ (x.namespace $\neq$ y.namespace). Elements that have the same name and structure but different namespaces are *equal*.

Definition 4: x.***equivalent***(y) iff (x.name $\neq$ y.name) $\wedge$ (x.structure = y.structure) $\wedge$ (x.namespace $\neq$ y.namespace). Elements that have different names and namespaces but the same structure are *equivalent*.

Definition 5: x.***subset***(y) iff (x.name = y.name) $\wedge$ (x.namespace $\neq$ y.namespace) $\wedge$ (x.childList $\subseteq$ y.childList) $\wedge$ (y.nodetype = 'all' $\vee$ y.nodetype = 'choice'). Elements with the same name, different namespaces, and the condition that the children of one element exist as a direct child list of the second element that is defined in terms of an *all* or *choice* satisfy the *subset* semantic relationship.

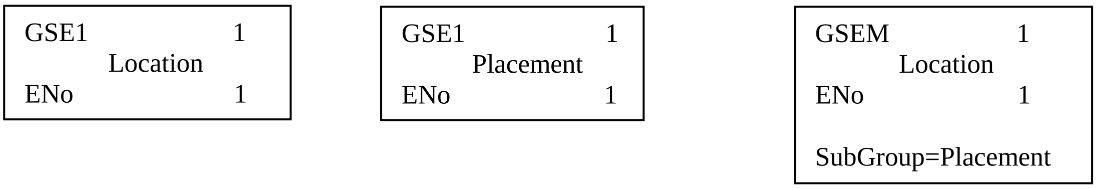
Definition 6: x.***incompatible***(y) iff (x.name = y.name) $\wedge$ (x.namespace $\neq$ y.namespace) $\wedge$ (x.structure $\neq$ y.structure) $\wedge$ $\neg$x.subset(y). Elements with the same name, different namespaces and structure that do not satisfy the subset semantic relationship are seen as *incompatible*.

Definition 7: ***unique***(x) iff $\forall$ y $\in$ (S-x), (x.name $\neq$ y.name) $\wedge$ (x.structure $\neq$ y.structure) $\wedge$ (x.namespace $\neq$ y.namespace). Elements that have different names and structure across all the local schemas in different namespaces are considered to be *unique*.

During the integration process, the process should follow certain integration rules and strategies. The fundamental activity in the comparison phase of integration is the conflict resolution. Conflict resolution and identification is central to successful integration. Naming conflicts, data type conflicts & scale difference and structural conflicts can occur during the XML schema integration.

Naming conflicts: Naming conflicts are of two types – synonyms and homonyms.

a) **Synonym Naming Conflict**: Synonym XML Schema elements have different names but the same definitions. Synonym naming conflict corresponds to the *equivalent* semantic relationship. For the non-terminal elements that are *equivalent*, synonym naming conflict can be resolved in the global schema through the use of a substitution group. This is the case in our example schemas (figure 2) with GSE1:*location* and GSE2:*placement* in the local schema set. Both are defined in terms of GSE1:*airport* and GSE1:*terminal*. In the global schema, *placement* is now defined as a substitution group for location. Terminal elements are defined in terms of their datatypes. Terminal elements can also be *equivalent*. For example *vendor* and *supplier* in our local schemas (figure 2) have the same datatypes - string, both help to define the *machine* element, and both are unique across the set of local schemas. It is quite possible that *vendor* and *supplier* are semantically *equivalent*. However, without semantic learning or through human interaction, the synonym conflict cannot be resolved for terminal elements that may be semantically *equivalent*. For an automatic integration in this case, both the terminal elements (*vendor* and *supplier* in our example) must be included in the global schema (figure 11).



Element in Schema 1 Element in Schema 2 Element in Global Schema

b) **Homonym Naming Conflict**: Homonym XML Schema elements have the same name but different definitions. Homonym naming conflict corresponds to *subset* and *incompatible* semantic relationships. *Incompatibility* can occur in non-terminal elements due to various combinations of *sequence*, *choice*, *all*, *any* or *empty* characterizations of the elements. Homonym naming conflicts are overcome for the non-terminal elements in the global schema through the *choice* and *all* mechanisms inherent to XML Schema structures [32]. Figures 9 and 10 show two examples of resolving homonym conflict for non-terminal elements that are *incompatible*. For the terminal elements, homonym conflict becomes datatype conflict and is discussed below.

Datatype conflicts & Scale Differences: Two terminal elements or attributes may have the same name but have different datatypes. The conflict may be of a *scale difference* or because of *disjoint datatypes* among the terminal elements. To resolve such conflicts, datatypes are expanded through the use of constraint facet redefinition (i.e. adjustment of scale), or through union of disjoint datatypes only so far as necessary to satisfy boundary conditions. In the case of the GSE schemas (figure 2), in the schema GSE1, the element *service_hours* is defined as having an *integer* datatype. In the schema GSE2, it is defined as having a *decimal* datatype. Since *decimal* and *integer* are not disjoint sets, this conflict is resolved through the adjustment of scale. In this case we can assign *decimal* as the datatype for the global element *service_hours*. For the attribute type, its list of enumeration values is expanded in the global schema to include *baggage_handler*, *boarding_stairs*, and *tow_truck* (figure 11). *Datatype* conflict occurs for the element *serial_number* as it has disjoint datatypes in the two schemas. In GSE1, it is defined as a *string,* and in GSE2 it is defined as a *positiveInteger*. The conflict is resolved by defining a new datatype, which is a union of the datatypes used to define *serial_number* in the global schema (figure 11).

Structural Conflicts: Structural conflicts are of two types – type conflicts and key conflicts.

a) **Type Conflicts**: A given element in one schema may be modeled as a terminal element, while in the second schema it might be modeled as a non-terminal element. We suggest two possible solutions for resolving the structural conflict and argue for our preference of one solution over the other. According to XML Schema recommendation [32], one cannot define an element as both a non-terminal and a terminal. As well, each element may be defined only once in a given namespace. In the first solution we add the element from both the local schemas to the global schema, so that the element that is modeled as a terminal element has a reference to its original namespace, thereby distinguishing it from the element that is modeled as a non-terminal element. In the second solution, we define that global element should be a non-terminal with mixed content, its datatype being the datatype of terminal local element and the children of the non-terminal local element as optional. In the first solution where both the terminal and non-terminal elements are added, a redundancy is created. Moreover, the name of the terminal element must be changed and a substitution group added to the terminal element in order to make it unique and different from the non-terminal element.

The second solution is more robust as no redundancy is created and the element is easily validated by the local schemas. Therefore we prefer the second solution, which is illustrated in figure 7.

b) **Key Conflicts**: A terminal element may be defined as a key element in a given schema. By definition of a key its contents must be not null and unique across the document in which it occurs. Three key conflicts can arise. In the first case, two different elements are defined as keys. This can be resolved in the integrated schema by making the new key as a group of the two original key elements. In the second case, two elements which have the same name may be defined such that in one schema, the element is defined as a key, and in the second schema, it is not defined a key. Without having knowledge of the entire data set, it is impossible to know that in each document, which relies on the key-less element, its contents are indeed not null and unique. Hence, in the integrated schema, the element in question cannot be defined as a key. In the third case, two elements with the same name are defined as keys in each of their local schemas. They may be equivalent locally, i.e. unique locally but not globally. The solution is to have a mapping from local keys to a global key, by attaching a prefix of document name or namespace to make them globally unique. The determination of the global key cannot be inferred directly from a set of schemas. Instead the actual data set (i.e. the documents) would have to be examined to find and/or add a terminal element where the content is both unique and not null. This will require human interaction.

During the *conformance* phase, the *semantic relationships* and *conflicts* identified in the comparison phase are resolved [5]. Initial schemas may be transformed in order to make them more suitable for integration. The XML Schemas in question are by definition *correct*, *complete*, and *minimal* because there XML documents exist that have been successfully validated by the schema(s) in question. A well-formed XML Schema, by definition, is *minimal* and *complete*. There is only one root element. All other elements present in the schema are related to the root element as either direct or indirect children. Each element has a unique name. Each element is defined only once. The schema should be viewed as a document that is fixed in nature. The schema is currently used to validate an existing non empty set of XML documents. Therefore, it is important that these initial schemas are not altered in any fashion.

Transformations that occur to align the initial schemas are accomplished using *restructuring* (renaming, substitution groups and subsetting) in the global schema only. The datatypes of terminal elements and attributes are expanded in the global schema to meet boundary restrictions through constraint facet redefinition and unions of incompatible datatypes. Optionality (minOccurs, maxOccurs) of child-elements is expanded to satisfy the largest minimum boundaries.

During schema *merging*, initial schemas are superimposed onto each other to result in the merged global schema. The merged schema is *complete* and *minimal*. The global schema is rearranged to ensure the highest level of minimality and understandability [5].

4.2 XML Schema Integration Rules and Strategies

We employ XSDM for the integration process. Essentially, two Schemas represented as trees are merged together. The merging of trees representing XML Schemas is achieved by merging the nodes. We define rules and strategies for merging of different types of nodes in the local XML Schemas into nodes representing global Schema. For the purpose of illustrating some of the rules, Schema 1 and Schema 2 from figure 2 are taken to show only specific parts of the nodes being merged.

4.2.1 Merging Nodes

All the corresponding element-element pairs, attribute-attribute pairs and the element-parent element of attribute pairs from the schemas should be integrated to form the global schema elements and attributes.

By definition, each schema has exactly one *root* element. All other elements in a given schema are direct or indirect children of the schema's root element [5]. If the root elements of the schemas to be merged have different names and namespaces, then the global root element consists of a *choice* of the initial two root elements. If the root elements have the same name but different namespaces, the roots are merged according to the applicable non-terminal merging rule. Should the roots have the same namespace and the same name, then the two roots are by definition *identical*, i.e. the schemas are identical and no further integration is required.

For all XSDM structures, when two structures of the same kind are merged, the new structure is assigned the namespace of the new global schema. Ancestral namespaces are recorded as a mean of identifying to which schema(s) the original structure belonged. Non-terminal elements may have mixed content, i.e. specified content combined with unspecified content. If one or both of the initial non-terminal elements have mixed content, then the resulting merged node must also allow mixed content to occur.

4.2.2 Integrating a Non-Terminal Element with a Terminal Element

If no element is defined in terms of an ANY or an EMPTY, and they have the same name but possibly different namespaces, the global element should be a non-terminal with mixed content, its datatype being the datatype of terminal local element and the children of the non-terminal local element as optional. The element *service_agreement* is defined as a non-terminal element in GSE1, and as a terminal element in GSE2. Figure 7 shows the merging process.

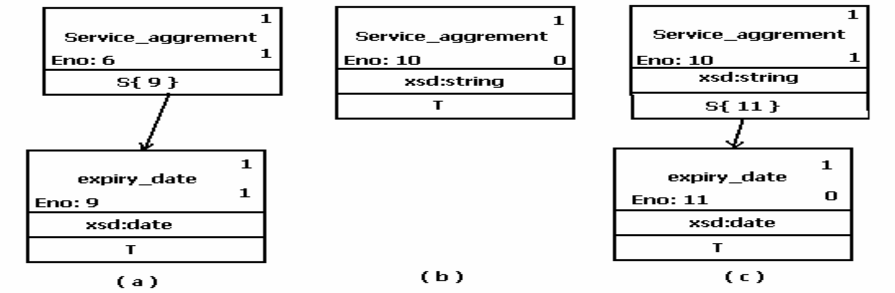


Figure 7. Structural conflict resolution: The element "service_agreement" is a non-terminal node in schema GSE1 as shown in (a) and is a terminal node in schema GSE2 as shown in (b). In the global schema the element "service_agreement" is a non-terminal with its children optional.

4.2.3 Integrating Terminal Elements

The merged element in the global schema inherits the name of the element in the local schema and obtains the global namespace. The integrated definition is determined according to the six cases given in Table 1. The attributes of the two initial elements are integrated and assigned to the global integrated element. In case (i) the element is defined in one schema and referred to in the second schema. The terminals are identical in this case. In case (ii) any attributes defined as part of the Empty element in one of the schemas, are included in the global Schema but are defined as *optional* to retain the validity of both the initial schemas. In case (iii) validation will occur but a strategy needs to be developed to retain the initial non-ANY terminal's definition. In cases (v) and (vi) We consider simple datatypes and the simple datatypes that are derived by restriction from other simple datatypes, as described in the W3C's paper on XML Schema Datatypes [6]. Complex datatypes can be built from these simple datatypes. Compatible datatypes are closely related; for example, a decimal and integer. Scale adjustment is used to merge the two terminal elements, as is shown for the element *service_hours* in the global schema GSEM (figure 11). Incompatible datatypes are not closely related; for example a Boolean and CDATA. The element *serial_number* in the global schema GSEM (figure 11) is constructed by taking the union of the datatypes string and positiveInteger. Note that the definition of compatible and incompatible datatypes is not related to the incompatible semantic relationship defined in section 4.1 between two nodes.

Table 1. Merge rules for terminal elements

Case	**Element in Schema A**	**Element in Schema B**	**Relationship**	**Merged Element in Global Schema**
(i)	Defined	Referred	Identical	Datatype of element in Schema A
(ii)	Non-empty	Empty	Homonym	Datatype of element in Schema A and attribute of the element in Schema B, if any, as an optional
(iii)	Non-ANY	ANY	Homonym	Element if of type ANY
(iv)	Unique	Unique	Unique	Elements with original datatypes with namespace of Global Schema
(v)	Non-ANY, non-empty	Non-ANY, non-empty	Homonym with compatible datatypes	Less constraining of the two datatypes with scale adjustment
(vi)	Non-ANY, non-empty	Non-ANY, non-empty	Homonym with incompatible datatypes	UNION of the two datatypes with their originally stated constraints

4.2.4 Merging Non-Terminal Elements

Knowing how to properly merge non-terminal elements is the key to being able to validate all existing instances of the schemas that are being integrated. The attributes of the initial non-terminals are integrated after integrating the non-terminal elements. The merged attributes become the attribute structure for the global non-terminal. The integrated definition for non-terminal elements is determined according to the sixteen cases given in Table 2.

In case (i) the non-terminal is defined in one schema and referred to in the second schema. The elements are identical in this case. In case (iii) The integrated element is assigned the name and definition of one of the initial elements. It is also assigned a substitution group [BA04] that indicates the name of the second element. Figure 8 shows an example of the merging of two *equivalent* elements. Non-terminal elements may have mixed content – specified content combined with unspecified content. If one of the initial non-terminal elements have mixed content, then the resulting merged node must also allow mixed content to occur as shown in case (v). Cases (v) to (xvi) define rules for integrating non-terminal elements with incompatible or subset relationship. Not all apply to integrating DTD non-terminal elements. Since an *all* is not defined in a DTD, the defined rules are not applicable in the integration of DTD non-terminal elements for the combinations having an *all* as one of the element nodes. In case (vi) the merged element will be a *choice* of the two sequences. Moh et al. [11] chose to use a Longest Common Sequence approach (LCS) to integrate two elements defined in terms of sequences of children. Such an integration strategy introduces a high level of possibility that XML documents formed after the integrated schema is described may contain structures that are invalid according to all the initial schemas. Our solution to integrating two non-terminal elements defined as sequences ensures that the integrated schema structures remain valid according to the initial schemas. The merged Schema from GSE1 and GSE2 for the nodes ‘gs_equipment’ represented as a *choice* between the two sequences is shown in figure 9. Case (xiv) is a special case of (viii). Element in one schema is defined as a sequence of elements that occur at most one time and the element in the second schema is defined as an a*ll*. The merged element will be an *all* of unique *all* and *sequence* elements from the two namespaces. The merged Schema from GSE1 and GSE2 for the nodes representing ‘machine’ as a *sequence* in GSE1 and as *all* in GSE2 is shown in figure 10. Case (xii) is a special case of (vii) and case (xiii) is a special case of (x).

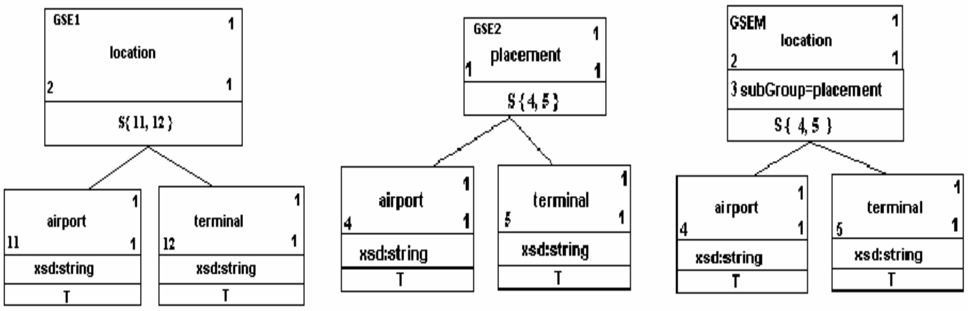


Figure 8. Synonym conflict for non-terminal elements that are equivalent. The non-terminal element “location” (Schema GSE1) and “placement” (Schema GSE2) are corresponding. The conflict is resolved through Substiution Group in the global schema GSEM, which keeps a record of equivalent names for the same element.

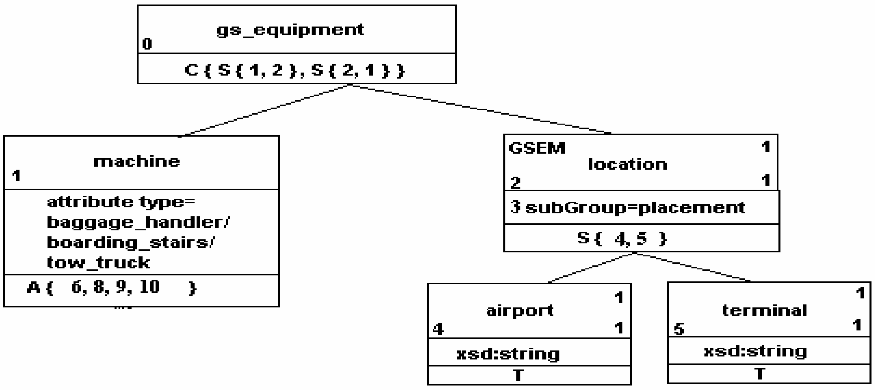


Figure 9. Conflict resolution for incompatible non-terminal elements. “gs_equipment” is a non-terminal element in both GSE1 and GSE2 and is a sequence of its child elements, but is incompatible in the two schemas as it has different definations in the two schemas. The conflict is resolved by merging the element “gs_equipment” from the two schemas as a Choice between the two original sequences.

Table 2. Merge rules for Non-terminal elements

Case	**Element from Schema A**	**Element from Schema B**	**Special case**	**Relationship**	**Merged element in Global schema**
(i)	Defined	Referred		Identical	Same definition as original
(ii)	Name: X Namespace: N	Name: X Namespace: M		Equal	Same definition as original
(iii)	Name: X Namespace: N	Name: Y Namespace: M		Equivalent (synonym)	Definition of X with substitution group = Y
(iv)	Name: X Namespace: N	Name: Y Namespace: M		Unique	X and Y added to global schema with global namespace
(v)	Name: X	Name: X	X has mixed	Incompatible	X will have mixed

	Namespace: N	Namespace: N	contents in one of the namespaces		contents
(vi)	Name: X Namespace: N Type: Sequence	Name: X Namespace: M Type: Sequence		Incompatible	Choice of the two sequences in namespaces N and M
(vii)	Name: X Namespace: N Type: Sequence	Name: X Namespace: M Type: Choice		Incompatible	Choice of sequence in namespace N and choice group in namespace M
(viii)	Name: X Namespace: N Type: Sequence	Name: X Namespace: M Type: All		Incompatible	Choice of sequence in namespace N and All in namespace M
(ix)	Name: X Namespace: N Type: Choice	Name: X Namespace: M Type: Choice		Incompatible	Choice of unique child elements of X from namespaces N and M
(x)	Name: X Namespace: N Type: Choice	Name: X Namespace: N Type: All		Incompatible	Choice of Choice group in namespace N and ALL in namespace M
(xi)	Name: X Namespace: N Type: All	Name: X Namespace: M Type: All		Incompatible	Choice of unique ALL elements of X from namespaces N and M
(xii)	Name: X Namespace: N Type: Sequence	Name: X Namespace: M Type: Choice	Sequence in namespace N is one of the choice group in namespace M	Subset	X from namespace M will acquire global namespace
(xiii)	Name: X Namespace: N Type: All	Name: X Namespace: M Type: Choice	All in namespace N is one of the choice group in namespace M	Subset	X from namespace M will acquire global namespace
(xiv)	Name: X Namespace: N Type: Sequence	Name: X Namespace: M Type: All	Elements in the sequence in namespace N appear 0 or 1 time	Incompatible	Unique elements from sequence in namespace N and All in namespace M
(xv)	Name: X Namespace: N Type: Non-ANY, Non-empty	Name: X Namespace: M Type: Empty		Incompatible	X from namespace N with minOccurs = 0 and maxOccurs = 1
(xvi)	Name: X Namespace: N Type: Non-empty/Empty	Name: X Namespace: M Type: ANY		Incompatible	X is defined as ANY and attributes integrated if present

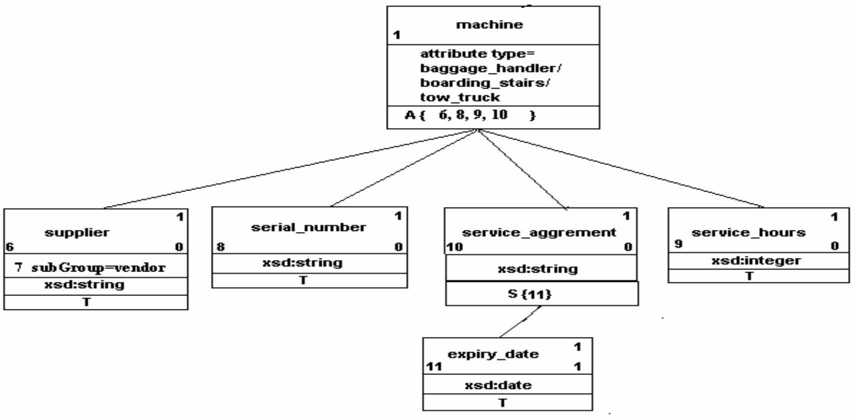


Figure 10. Incompatible conflict among non-terminal elements. Element "machine" is a non-terminal element in both GSE1 and GSE2, but it is incompatible in the two schemas as it has different definations. The conflict is resolved by merging the element "machine" from schema GSE1 and schema GSE2as an "all" between all of the child elements from the two original schemas.

4.2.5 Attribute Integration

Attributes may be associated with all classes of terminal and non-terminal elements. Attributes, like terminal elements, are data containing structures. Attributes are defined, in part, by their datatype. Thus, the integration strategy for attributes is very similar to that for terminal elements. The structures and datatypes of attributes are merged using the rules outlined for terminal elements.

Attributes have a different optionality structure than elements, referred to as their *use* [32]. An XML Schema attribute may be *optional* or *required*, with a *default* or *fixed* value associated with it. A given attribute occurs at most once for each instance of the element it is associated with. A survey of attributes suggests the following rules for attribute optionality integration:

1. If the attributes are *identical*, its *use* remains the same in the integrated schema.
2. If the attribute is present in only one of the initial two schemas (i.e. *unique*), then it must be an *optional* attribute in the integrated schema.
3. If the attribute is present in both schemas, and it is *optional* in one, then it must be *optional* in the integrated schema.
4. If the attribute has a *fixed* value in only one of the schemas, then this may be a *default* value in the integrated schema if no *default* value was present in the second initial schema.
5. If an attribute is present in both initial schemas, and has two different *fixed* values, it is *required* in the integrated schema, but no *fixed* value may be present.
6. If an attribute is present in both initial schemas, and has two different *default* values, it may not have a *default* value in the integrated schema.
7. If an attribute is *fixed* in one schema and has a *default* value in the second, and the two values are the same, the attribute may retain the value as a *default* value in the integrated schema.

5. Implementation: XML Schema Integration

The global schema GSEM is obtained after applying the rules of integration explained in this paper on the local schemas GSE1 and GSE2 for the airline example. Figure 11 shows the graphical representation of the global schema GSEM.

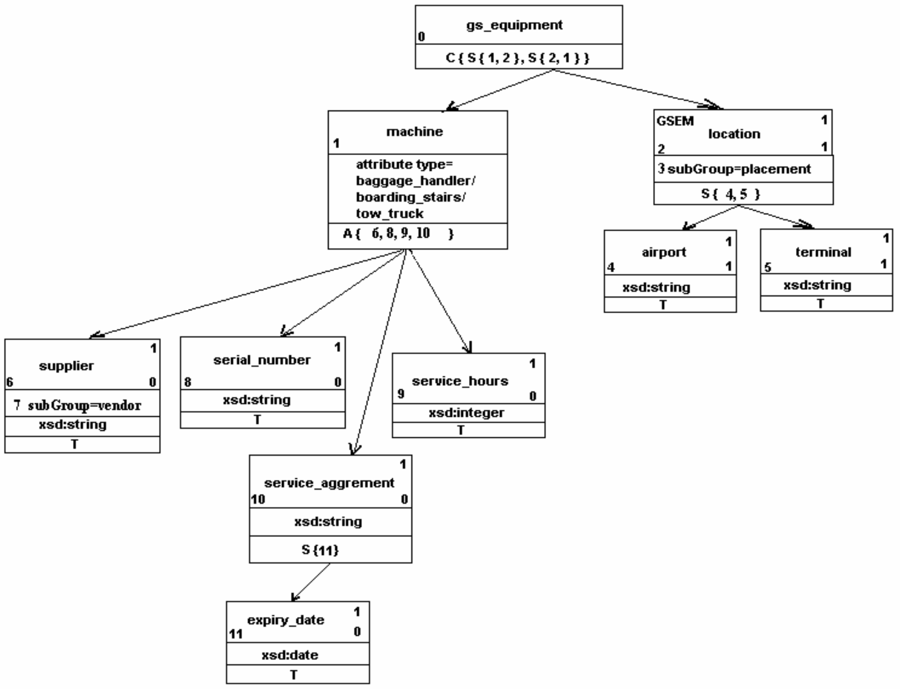


Figure 11. Graphical representation of the global schema GSEM

The implementation of the integration system consists in creating a node list that has a node for each unique element definition from each local schema .xsd file. Each non-terminal node has an associated list of direct children. Both the Node list and Child list have been implemented as arrays, which grow dynamically and can be fixed in size once insertion is complete. Beginning with the root element (root Node), the definition of each global element is created by first applying an appropriate child integration policy based on its Node type. This method is then applied to each of the child Nodes that are part of its new definition, until all of the terminal nodes (data containing elements) for that particular branch of the global schema's tree are integrated in a depth-first approach. The implementation of the integration rules requires four classes of methods - *get*, *set*, *compare* and *recombination* methods. The *get* methods retrieve the value of a particular object, e.g. a node, a datatype, an attribute or a simpler object like the String name for a given element. The *set* methods assign a new value to a particular object. The *comparison* methods compare the values of two similar objects, e.g. the names of two nodes. The *recombination* methods create new childlists and nodelists for the merged schema. Once the global schema is complete, it is output as a new .xsd file. The integration of schemas outlined in this paper has been implemented in Java using the Apache Xercers Parser. The GUI is capable of displaying two views - graphical view and the Schema view.

The merged Schema GSEM obtained by integrating the local schemas GSE1 and GSE2 (shown in figure 13) for the airline example is given in figure 12 below (its implementation is shown in Figure 14).

```
<?xml version="1.0"?>
<schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
targetNamespace="http://www.GSEMexample.org"
elementFormDefault="qualified"
xmlns:GSEM="http://wwwGSEMexample.org
xmlns:GSE2="http://wwwGSE2example.org >
<element name ="gs_equipment”>
        <complexType>
                <choice>
                        <sequence>
```

```
                                        <element ref="GSEM:machine" minOccurs="1" maxOccurs="1"/>
                                         <element ref="GESM:location" minOccurs="1" maxOccurs="1" />
                                   </sequence>
                                   <sequence>
                                        <element ref="GESM:location" minOccurs="1" maxOccurs="1" />
                                        <element ref="GSEM:machine" minOccurs="0" maxOccurs="1"/>
                                   </sequence>
                              </choice>
          </complexType>
     </element>
     <element name ="machine">
       <complexType>
          <all>
           <element name="supplier" type="xsd:string" minOccurs="0" maxOccurs="1" />
           <element name="serial_number"  type="serial_number_type" minOccurs="0" maxOccurs="1" />
          <element ref=”GSEM:service_agreement" minOccurs="1" maxOccurs="1" />
          <element name="service_hours" type="decimal" minOccurs="0" maxOccurs="1" >
          <element name="vendor" type="xsd:string" minOccurs="0" maxOccurs="1" >
         </all>
         <xsd:attribute name="type" use="optional">
              <xsd:simpleType>
                   <xsd:restriction base="xsd:string">
                             <xsd:enumeration value="baggage_handler"/>
                             <xsd:enumeration value="boarding_stairs"/>
                             <xsd:enumeration value="tow_truck"/>
                   </xsd:restriction>
              </xsd:simpleType>
         </xsd:attribute>
      </complexType>
    </element>
    <xsd:simpleType name=”serial_number_type”>
         <xsd:union>
                   <xsd:string>
                   <xsd:positiveInteger>
         </xsd:union>
    </xsd:simpleType>
    <element name ="service_agreement” >
          <complexType>
               <sequence>
                    <element name="expiry_date" type="xsd:date" minOccurs="0" maxOccurs="1" />
              </sequence>
          </complexType>
     </element>
     <element name ="location" substutionGroup =”GESM:placement”>
          <complexType>
                   <sequence>
                        <element name="airport" type="xsd:string" minOccurs="1" maxOccurs="1" />
                        <element name="terminal" type="xsd:string" minOccurs="1" maxOccurs="1" />
                    </sequence>
          </complexType>
     </element>
     <element name ="placement">
           <complexType>
               <sequence>
                    <element name="airport" type="xsd:string" minOccurs="1" maxOccurs="1" />
                    <element name="terminal" type="xsd:string" minOccurs="1" maxOccurs="1" />
                 </sequence>
          </complexType>
     </element>
     </schema>
```

Figure 12. Global Schema (GSEM)

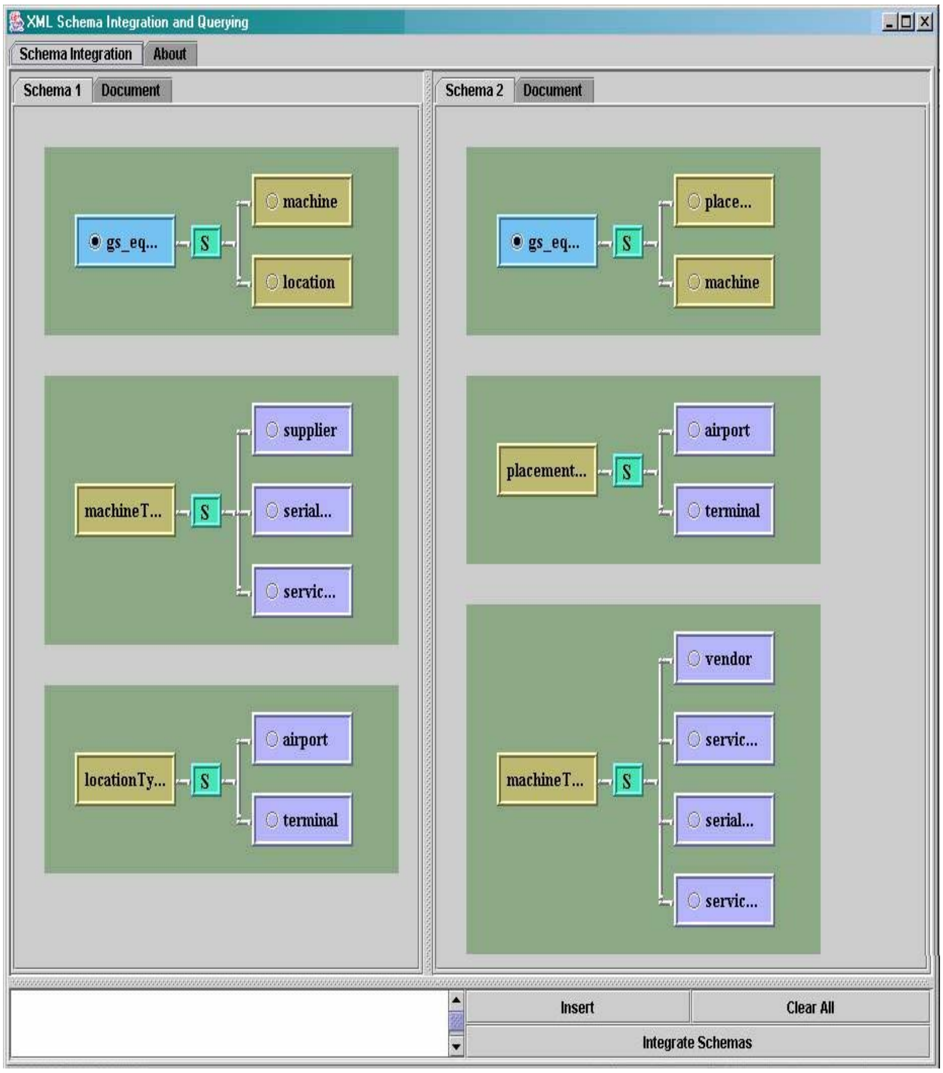
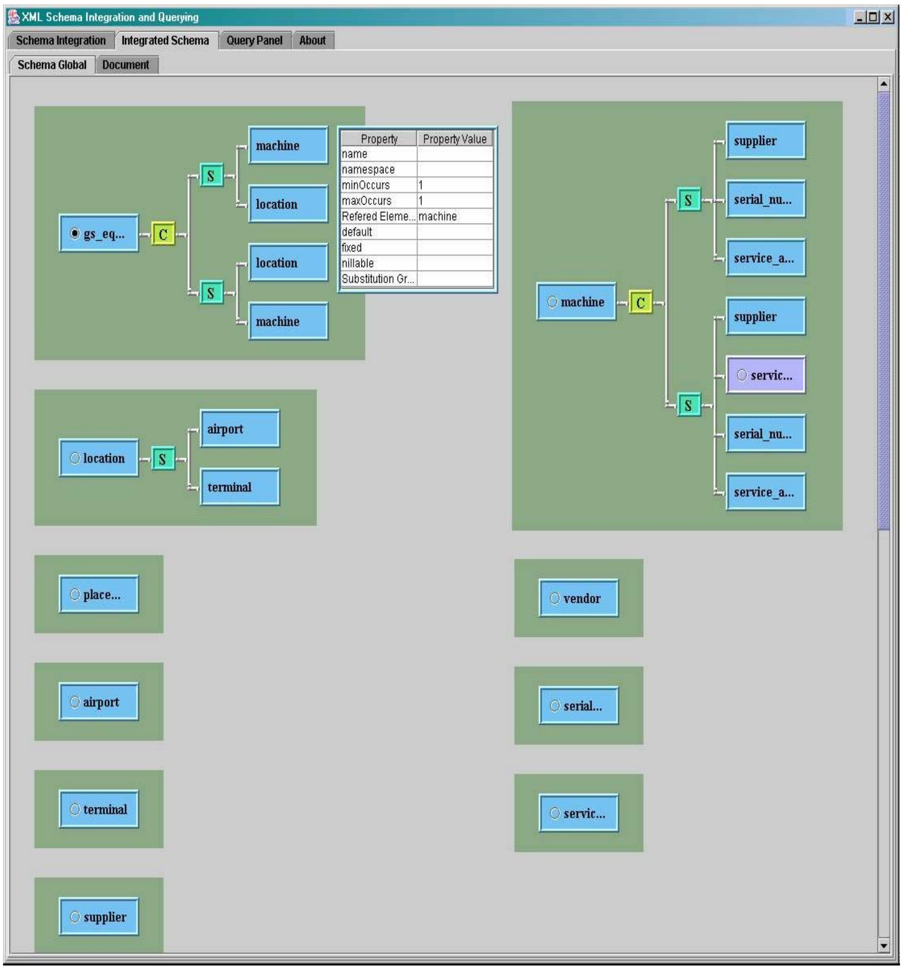


Figure 13: XML Schemas parsed and shown as a DOM tree in GUI

6. Query Rewriting using Semantic Mapping Rules

While integration has long been an active research area, the constraint-mapping problem we study in this paper has not been addressed thoroughly. We specifically address the semantic mapping of the constraints, or analogously the translation of vocabulary. In contrast, other efforts have mainly focused on generating query plans that observe the native grammar restrictions (such as allowing conjunctions of two constraints, disallowing disjunctions, etc.) [9].

A query can be viewed as a Boolean expression of constraints of the form [ElementName Op Value]. These constraints constitute the "vocabulary" for the query, and must be translated to constraints understood by the target source. In general, we have to map attributes, convert data values, and transform the operators. We define a query to be a set of conjunctive constraints that selects or identifies one or more elements of an XML document. The general problem of the semantic mapping of elements (e.g. mapping Author to Creator) is a major barrier to a distributed search over very different XML documents. Achieving the best translation is challenging because the sources support different constraints for formulating queries, and often these constraints cannot be precisely translated. For instance, a query [score> 8] might be "perfectly" translated as [rating > 0.8] at some site, but can only be approximated as [grade = A] at another [10].



Property	Property Value
name	
namespace	
minOccurs	1
maxOccurs	1
Refered Eleme...	machine
default	
fixed	
nillable	
Substitution Gr...	

Figure 14: The integrated Global Schema

In this section, we propose mapping rules and strategies that can be applied during the integration process and query rewriting strategies. The methodology used in this paper relies on rules to indicate relations between the elements of two different XML documents and thus, how a query should be rewritten to accommodate these relations. Also the strategy relies on rules to indicate what groups of constraints need to be mapped as a unit, and what functions must be executed to actually transform element values. A rule matches a set of conjunctive constraints and specifies translation, similar to pattern matching. The goal of the mapping rules is to translate query constraints into ones that are understood and supported in the local source. Query translation must rely on human expertise to define what constraints may be interrelated, and how to translate basic semantic units. In the case of constraint mapping, it is critical to note that query mapping is not simply a matter of translating each constraint separately. Some constraints can be inter-dependent and must be handled together. A mapping rule is used to convert a global query constraint into one that can be understood by the local source. The head (left hand side) of the rule consists of constraint patterns and conditions to match the original constraints of the global query. The tail (to the right of the →) consists of functions for converting value formats and an emit: clause that specifies the corresponding constraint to be used for the local source. An example mapping rule looks like the following:

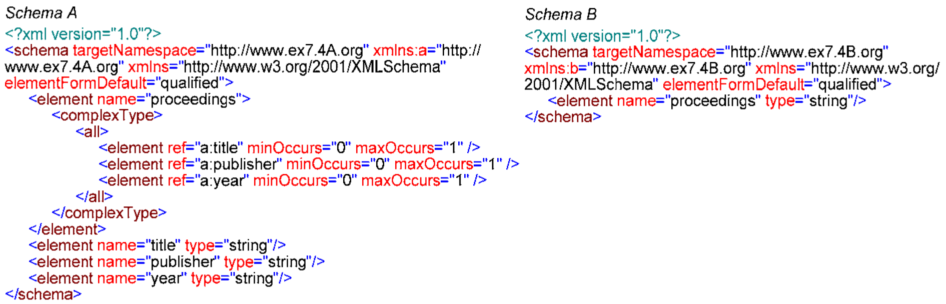
[Distance D] → D2 = ConvertDistanceToKm(D);
emit: [Distance D2]

Within the mapping rule it is possible that a function that may be used to convert constraint values, combine constraints values or to determine if a specific condition is met. In the case of the example above, the ConvertDistanceToKm function is used to convert a distance value into kilometers, which are the appropriate units for the local source. It will be necessary that there be human interaction for the creation of these functions to ensure correctness and completeness. Subsequently the functions can be stored in a repository and called from the repository the next time the function is required.

It is possible that semantic mapping rules be defined during integration. By creating these mapping rules the task of interpreting queries and rewriting them for the local schemas becomes easier. The proposed query rewriting methodology defines the mapping rules within the integration rules, some of which might require user interaction. For a detailed list of integration rules see section 4.2. In some of these integration cases it is appropriate to create mapping rules, which deal with the semantic difference between the local sources. A query is a Boolean expression of constraints. Constraints constitute the vocabulary for the query and are translated to constraints, which can be understood by the target source. Constraints have the form [ELEMENT value]. In this case, the equality operator is implied; however, a constraint is not limited to only this single operator. In general, query rewriting includes two main tasks. The first task is to determine if an element in a given query constraint is available in each local source. The second task is the mapping element names, converting data values, etc. according to the semantic mapping rules established for each source. Each constraint may not be able to be mapped individually as the constraints could be dependent on one another.

The mapping rules can be generated during the integration process and a mapping table created. The mapping table assists in the query rewriting process as it keeps track of all the elements and attributes. It contains a list of all the elements that exist in the global schema. For each element in the table, it records the attributes, element references, mapping rules, namespaces and data locations where XML fragments or documents may be found when applying the query.

Let us consider the local schema A and Schema B in Figure 15. The element "proceedings" is represented as a non-terminal element in Schema A and as a terminal element in Schema B. This exhibits a structural conflict when the two elements have to be merged in the global schema as the elements in the two local schema have the same name but different structure. The conflict could be resolved through user interaction by selecting the non-terminal element or the terminal element "proceedings". The global schema in Figure 16 shows that the structural conflict was resolved by selecting the non-terminal element "proceedings" and the global schema in Figure 17 shows that the structural conflict was resolved by selecting the terminal element "proceedings".



Schema A

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.ex7.4A.org" xmlns:a="http://
www.ex7.4A.org" xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <element name="proceedings">
        <complexType>
            <all>
                <element ref="a:title" minOccurs="0" maxOccurs="1" />
                <element ref="a:publisher" minOccurs="0" maxOccurs="1" />
                <element ref="a:year" minOccurs="0" maxOccurs="1" />
            </all>
        </complexType>
    </element>
    <element name="title" type="string"/>
    <element name="publisher" type="string"/>
    <element name="year" type="string"/>
</schema>
```

Schema B

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.ex7.4B.org"
xmlns:b="http://www.ex7.4B.org" xmlns="http://www.w3.org/
2001/XMLSchema" elementFormDefault="qualified">
    <element name="proceedings" type="string"/>
</schema>
```

Figure 15. Local schemas

The semantic mapping rules that map the constraints from the global schema onto the local schemas for the two cases of global schema in Figure 16 and Figure 17 would take the following form:

```
MappingRule1: [m:title T] → T1 = fn:resolve-QName("b:proceedings", $element) ;
          emit: [contains(T1/, T)]
MappingRule2: [m:publisher P] → P1 = fn:resolve-QName("b:proceedings", $element);
          emit: [contains(b:proceedings/, P)]
Mapping Rule3: [m:year Y] → Y1 = fn:resolve-QName("b:proceedings", $element);
          emit: [contains(b:proceedings/, Y)]
MappingRule4: [m:proceedings/title] → PT = fn:resolve-QName("b:proceedings", $element);
          emit: [substring-before(PT, m:proceedings/publisher)]
MappingRule5: [m:proceedings/publisher] → PP = fn:resolve-QName("b:proceedings", $element);
          emit: [substring-before(PP, m:proceedings/year) ∧ substring-after(PP, m:proceedings/title)]
MappingRule6: [m:proceedings/year] → PY = fn:resolve-QName("b:proceedings", $element);
          emit: [substring-after(PY, m:proceedings/publisher)]
MappingRule7: [m:proceedings] → PS1 = fn:resolve-QName("a:proceedings/title", $element) ∧
                                 PS2 = fn:resolve-QName("a:proceedings/publisher", $element) ∧
                                 PS3 = fn:resolve-QName("a:proceedings/year", $element);
          emit: [string-join(PS1, PS2, PS3)]
MappingRule8: [contains(m:proceedings, $a) ] → S1 = concat("a:proceedings","/title") ∨
                                                  S1 = concat("a:proceedings","/publisher") ∨
                                                  S1 = concat("a:proceedings","/year");
               emit: [S1= $a]
```

The constraint [m:title T] in MappingRule1 means m:title = T, i.e. the "title" element from the global namespace "m" equals a value given by "T". The right side of the → in MappingRule1 is a conversion function fn:resolve-QName("b:proceedings", $element) which returns a xs:QName (i.e. an expanded qualified name [6]) whose namespace URI is specified by the namespace binding corresponding to the prefix "b" and whose local name is "proceedings". The emit: clause specifies the corresponding constraint on the local schema B which in MappingRule1 returns the function contains(T1/, T), which tests if the substring denoted by T1, i.e. value of the element "b:proceedings/" contains the substring denoted by T, i.e. value of the element "m:title". MappingRule2 and MappingRule3 are similar to MappingRule1. MappingRule4 states that the value of the element "m:proceedings/title" in the global schema is mapped to a function substring-before() in the local schema B and emits a function substring-before(). The function substring-before($arg1, $arg2) returns the substring value of $arg1 that precedes the substring value of $arg2. In MappingRule4 the function substring-before(PT, m:proceedings/publisher) returns the substring of "proceedings" element in schema B that precedes the substring for the value of "publisher". The function substring-after($arg1, $arg2) returns the substring value of $arg1 that follows the substring value of $arg2. In MappingRule5 the function substring-after(PP, m:proceedings/title) returns substring of "proceedings" element in schema B that follows the substring value of "title". MappingRule 5 and MappingRule6 are similar to MappingRule4. MappingRule7 states that the value of the element "m:proceedings" in the global schema is mapped to a function string-join() which joins the strings for elements "title", "publisher" and "year" in schema A. MappingRule8 states that a constraint on "proceedings" element in the global schema is a function contains() and is mapped to a string with a value given by the path a:proceedings/title, or a:proceedings/publisher, or a:proceedings/year in schema A and emits the constraint with either of the three paths equal to the string value $a in schema A. The function contains(m:proceedings, $a) checks if the string value $a is contained in the string value of the element "proceedings". All the functions defined in the mapping rules in this section have been taken from the in-built functions listed in XQuery 1.0 standard of W3C [24].

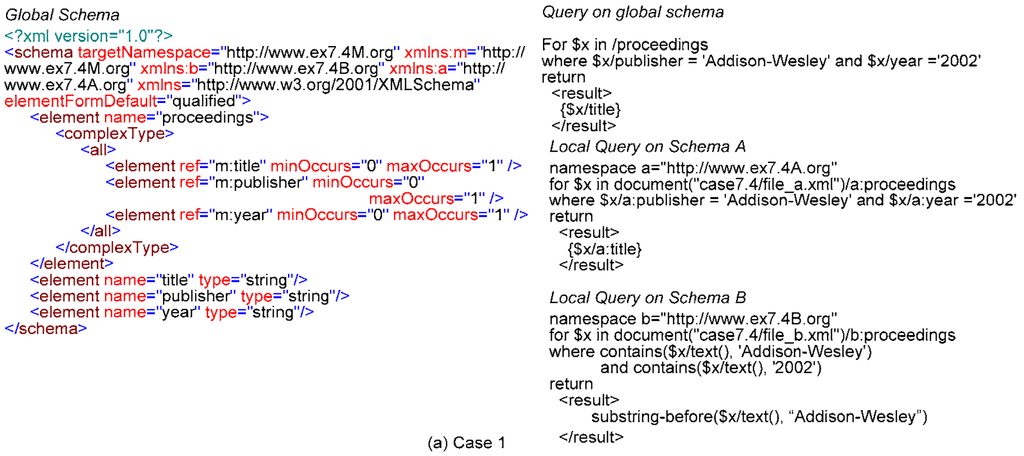
A mapping table can be created from the mapping rules. Table 3 shows a mapping table generated for the mapping rules for the global schemas given in Figure 16 and Figure 17.

Table 3: Mapping table for global schemas in Figure 16 and Figure 17

Constraint on the global schema	**QName in local schema**	**Constraint in local schema**
m:title = T	b:proceedings	contains(b:proceedings /, T)
	a:title	a:title = T
m:publisher = P	b:proceedings	contains(b:proceedings /, P)
	a:publisher	a:publisher = P
m:year = Y	b:proceedings	contains(b:proceedings /, Y)
	a:year	a:year = Y
m:proceedings/title	b:proceedings	substring-before(b:proceedings, m:proceedings/publisher
	a:proceedings/title	a:proceedings/title
m:proceedings/publisher	b:proceedings	substring-before(b:proceedings, m:proceedings/year) ∧ substring-after(b:proceedings, m:proceedings/title)
	a:proceedings/publisher	a:proceedings/publisher
m:proceedings/year	b:proceedings	substring-after(b:proceedings, m:proceedings/publisher)
	a:proceedings/year	a:proceedings/year
m:proceedings	a:proceedings/title, a:proceedings/publisher, a:proceedings/year	string-join(a:proceedings/title, a:proceedings/publisher, a:proceedings/year)
	b:proceedings	b:proceedings
contains(m:proceedings, $a)	a:proceedings/title, a:proceedings/publisher, a:proceedings/year	a:proceedings/title = $a, a:proceedings/publisher = $a, a:proceedings/year = $a

Query Rewriting: Queries on the global schema are rewritten on the local schemas using the mapping tables. Let us consider a query on the global schema shown in Figure 16 which states "return the title where the publisher is Addison-Wesley and the year is 2002". To rewrite the query on the local schema A and schema B, the global constraints must be mapped to constraints on the local schemas using Table 3. The constraints on the global schema are mapped to the local schemas as shown below:

[m:publisher = "Addison-Wesley"] → [a:publisher = "Addison-Wesley"]
[m:publisher = "Addison-Wesley"] → [contains(b:proceedings/, "Addison-Wesley")]
[m:year = "2002"] → [a:year = "2002"]
[m:year = "2002"] → [contains(b:proceedings/, "Addison-Wesley")]
[m:proceedings/title] → [a:proceedings/title]
[m:proceedings/title] → [substring-before(b:proceedings, "Addison-Wesley")]



Global Schema

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.ex7.4M.org" xmlns:m="http://
www.ex7.4M.org" xmlns:b="http://www.ex7.4B.org" xmlns:a="http://
www.ex7.4A.org" xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <element name="proceedings">
        <complexType>
            <all>
                <element ref="m:title" minOccurs="0" maxOccurs="1" />
                <element ref="m:publisher" minOccurs="0"
                                                maxOccurs="1" />
                <element ref="m:year" minOccurs="0" maxOccurs="1" />
            </all>
        </complexType>
    </element>
    <element name="title" type="string"/>
    <element name="publisher" type="string"/>
    <element name="year" type="string"/>
</schema>
```

Query on global schema

```
For $x in /proceedings
where $x/publisher = 'Addison-Wesley' and $x/year ='2002'
return
  <result>
    {$x/title}
  </result>
```

Local Query on Schema A

```
namespace a="http://www.ex7.4A.org"
for $x in document("case7.4/file_a.xml")/a:proceedings
where $x/a:publisher = 'Addison-Wesley' and $x/a:year ='2002'
return
  <result>
    {$x/a:title}
  </result>
```

Local Query on Schema B

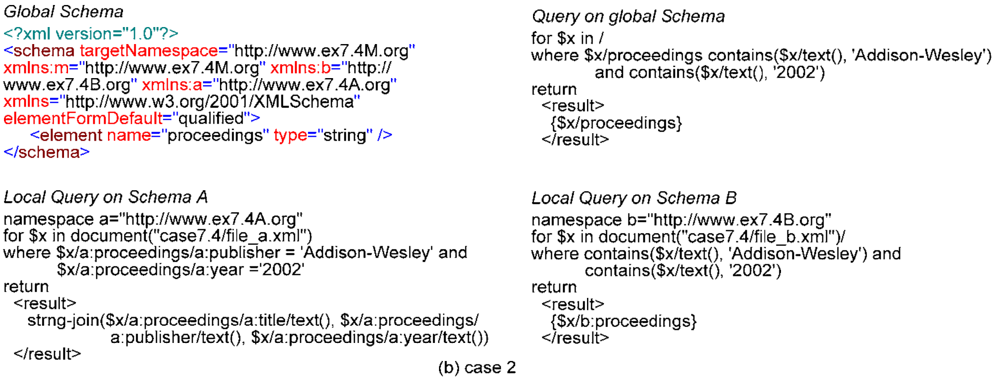
```
namespace b="http://www.ex7.4B.org"
for $x in document("case7.4/file_b.xml")/b:proceedings
where contains($x/text(), 'Addison-Wesley')
          and contains($x/text(), '2002')
return
  <result>
        substring-before($x/text(), "Addison-Wesley")
  </result>
```

(a) Case 1

Figure 16. Global Schema and example query for case 1

Consider a query on the global schema shown in Figure 17 which states "return the proceedings by Addison-Wesley in 2002". The constraints on the global schema are mapped to the local schemas as shown below:

[m:proceedings] → [string-join(a:proceedings/title, a:proceedings/publisher, a:proceedings/year)]
[m:proceedings] → [b:proceedings]
[contains(m:proceedings, "Addison-Wesley")] → [a:proceedings/publisher = "Addison-Wesley"]
[contains(m:proceedings, "Addison-Wesley")] → [contains(b:proceedings, "Addison-Wesley")]
[contains(m:proceedings, "2002")] → [a:proceedings/year = "2002"]
[contains(m:proceedings, "2002")] → [contains(b:proceedings, "2002")]



Global Schema

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.ex7.4M.org"
xmlns:m="http://www.ex7.4M.org" xmlns:b="http://
www.ex7.4B.org" xmlns:a="http://www.ex7.4A.org"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <element name="proceedings" type="string" />
</schema>
```

Query on global Schema

```
for $x in /
where $x/proceedings contains($x/text(), 'Addison-Wesley')
          and contains($x/text(), '2002')
return
  <result>
    {$x/proceedings}
  </result>
```

Local Query on Schema A

```
namespace a="http://www.ex7.4A.org"
for $x in document("case7.4/file_a.xml")
where $x/a:proceedings/a:publisher = 'Addison-Wesley' and
          $x/a:proceedings/a:year ='2002'
return
  <result>
    strng-join($x/a:proceedings/a:title/text(), $x/a:proceedings/
                a:publisher/text(), $x/a:proceedings/a:year/text())
  </result>
```

Local Query on Schema B

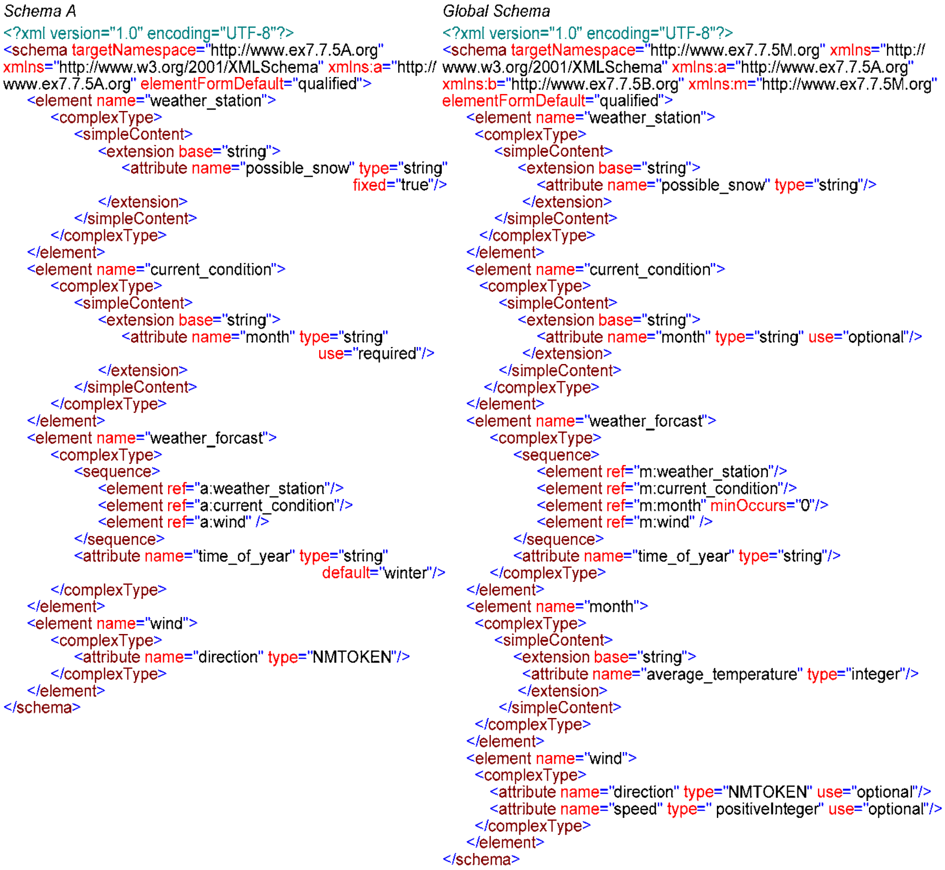
```
namespace b="http://www.ex7.4B.org"
for $x in document("case7.4/file_b.xml")/
where contains($x/text(), 'Addison-Wesley') and
          contains($x/text(), '2002')
return
  <result>
    {$x/b:proceedings}
  </result>
```

(b) case 2

Figure 17. Global Schema and example query for case 2

6.1 Querying the attributes

The same techniques and strategies used while creating mapping rules for XML Schema elements can be used for XML attributes. Mapping rules can be generated to deal with the semantic differences and likenesses of attributes. These attribute mapping rules can have the same syntax as element mapping rules. Users can determine relationships among attributes, such as synonyms, and write mapping rules accordingly. When writing a query based on the global schema, the user must be aware of the possibility that data contained in an attribute may be the same as the data that is stored in an element. For example, in one local schema, an element may have the same name as an attribute which appears in another local schema as shown in Figure 18.

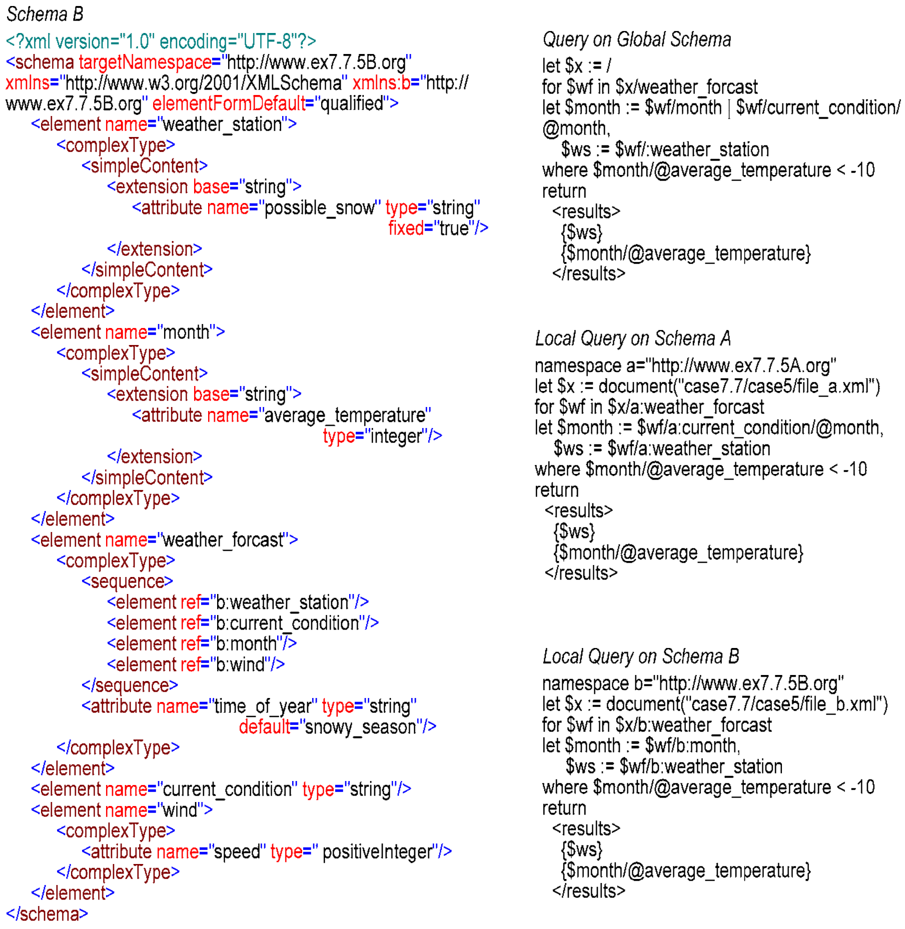


Schema A

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.ex7.7.5A.org"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:a="http://
www.ex7.7.5A.org" elementFormDefault="qualified">
   <element name="weather_station">
      <complexType>
         <simpleContent>
            <extension base="string">
               <attribute name="possible_snow" type="string"
                                                   fixed="true"/>
            </extension>
         </simpleContent>
      </complexType>
   </element>
   <element name="current_condition">
      <complexType>
         <simpleContent>
            <extension base="string">
               <attribute name="month" type="string"
                                                use="required"/>
            </extension>
         </simpleContent>
      </complexType>
   </element>
   <element name="weather_forcast">
      <complexType>
         <sequence>
            <element ref="a:weather_station"/>
            <element ref="a:current_condition"/>
            <element ref="a:wind" />
         </sequence>
         <attribute name="time_of_year" type="string"
                                                default="winter"/>
      </complexType>
   </element>
   <element name="wind">
      <complexType>
         <attribute name="direction" type="NMTOKEN"/>
      </complexType>
   </element>
</schema>
```

Global Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.ex7.7.5M.org" xmlns="http://
www.w3.org/2001/XMLSchema" xmlns:a="http://www.ex7.7.5A.org"
xmlns:b="http://www.ex7.7.5B.org" xmlns:m="http://www.ex7.7.5M.org"
elementFormDefault="qualified">
   <element name="weather_station">
    <complexType>
       <simpleContent>
          <extension base="string">
             <attribute name="possible_snow" type="string"/>
           </extension>
       </simpleContent>
     </complexType>
   </element>
   <element name="current_condition">
     <complexType>
        <simpleContent>
          <extension base="string">
             <attribute name="month" type="string" use="optional"/>
           </extension>
        </simpleContent>
      </complexType>
   </element>
   <element name="weather_forcast">
      <complexType>
         <sequence>
            <element ref="m:weather_station"/>
            <element ref="m:current_condition"/>
            <element ref="m:month" minOccurs="0"/>
            <element ref="m:wind" />
         </sequence>
         <attribute name="time_of_year" type="string"/>
      </complexType>
   </element>
   <element name="month">
    <complexType>
       <simpleContent>
         <extension base="string">
          <attribute name="average_temperature" type="integer"/>
          </extension>
       </simpleContent>
    </complexType>
   </element>
   <element name="wind">
    <complexType>
      <attribute name="direction" type="NMTOKEN" use="optional"/>
      <attribute name="speed" type=" positiveInteger" use="optional"/>
    </complexType>
   </element>
</schema>
```



Schema B

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.ex7.7.5B.org"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:b="http://
www.ex7.7.5B.org" elementFormDefault="qualified">
    <element name="weather_station">
        <complexType>
            <simpleContent>
                <extension base="string">
                    <attribute name="possible_snow" type="string"
                                                    fixed="true"/>
                </extension>
            </simpleContent>
        </complexType>
    </element>
    <element name="month">
        <complexType>
            <simpleContent>
                <extension base="string">
                    <attribute name="average_temperature"
                                           type="integer"/>
                </extension>
            </simpleContent>
        </complexType>
    </element>
    <element name="weather_forcast">
        <complexType>
            <sequence>
                <element ref="b:weather_station"/>
                <element ref="b:current_condition"/>
                <element ref="b:month"/>
                <element ref="b:wind"/>
            </sequence>
            <attribute name="time_of_year" type="string"
                                  default="snowy_season"/>
        </complexType>
    </element>
    <element name="current_condition" type="string"/>
    <element name="wind">
        <complexType>
            <attribute name="speed" type=" positiveInteger"/>
        </complexType>
    </element>
</schema>
```

Query on Global Schema

```
let $x := /
for $wf in $x/weather_forcast
let $month := $wf/month | $wf/current_condition/
@month,
   $ws := $wf/:weather_station
where $month/@average_temperature < -10
return
  <results>
   {$ws}
   {$month/@average_temperature}
  </results>
```

Local Query on Schema A

```
namespace a="http://www.ex7.7.5A.org"
let $x := document("case7.7/case5/file_a.xml")
for $wf in $x/a:weather_forcast
let $month := $wf/a:current_condition/@month,
   $ws := $wf/a:weather_station
where $month/@average_temperature < -10
return
  <results>
   {$ws}
   {$month/@average_temperature}
  </results>
```

Local Query on Schema B

```
namespace b="http://www.ex7.7.5B.org"
let $x := document("case7.7/case5/file_b.xml")
for $wf in $x/b:weather_forcast
let $month := $wf/b:month,
     $ws := $wf/b:weather_station
where $month/@average_temperature < -10
return
  <results>
   {$ws}
   {$month/@average_temperature}
  </results>
```

Figure 18. Querying the attributes

The local schemas show that there exists a 'month' element in schema B while in schema A there is a 'month' attribute that occurs in the 'current_condition' element. Both the 'month' element and attribute contain the same data, but is represented in a different form; this must be taken into consideration when writing a query in order to retrieve all possible data for the 'month' element or attribute where ever it may appear in the various data locations. The global schema obtained after integrating local schema A and schema B is given in Figure 18 When rewriting the query based on the global schema in terms of the local schemas, all elements require binding with their respective namespace URI through the use of a prefix. The mapping rules in this case will assign the respective namespace URI to the elements and attributes in the local schema A and schema B. Some of the elements defined in the local schemas are *empty* elements where the element does not contain any data. It may however contain attributes as in the element "wind". Consider the query "return the weather station and month where the average temperature is colder than -10" on the global schema. The query takes into consideration the month being an element and as an attribute. The query can be rewritten by applying the mapping rules that assign the namespace URI to the elements and attributes in the local schemas. The query on global schema and the corresponding local queries on schema A and schema B are shown in Figure 18

7. Query Rewriting: Implementation

The XQuery 1.0 data model defines the information in an XML document that is available to an XQuery processor [4]. XQuery 1.0 is built on the same principles as the XPath 2.0 data model. The type system of

XQuery 1.0 is based on XML Schema. A query that is posed to XQuery processor along with the global schema, on which the query is based, must be rewritten on the local XML documents validated by their respective local schemas. The local XML documents and the corresponding schemas are available on the system storage. The query rewriting process requires a mapping file that contains all the mapping rules and necessary information to translate the global query into local queries.

The mapping file is generated during the process of schema integration. The necessary information with respect to each element such as namespace, prefix, data location, root status and attributes are stored in the mapping table through the integration data model. This facilitates the starting point for query rewriting as we must retain all necessary information with respect to each schema element prior to integration as the global schema does not hold specific local schema information which is required when remapping the global query in terms of the local queries. For instance, when two elements are being integrated where they are equivalent but may exist with different names, i.e. synonyms, the mapping table will record the namespace URI and the prefix associated with it. The mapping table must also show in what XML document(s) the element exists and it must generate a rule which confirms the element name as retained in the global schema. This rule for example is the substitution group rule. When remapping the global query, the element name must be substituted with its synonym.

```
Create_query_instance(URI LocalSource, string MappingRules, string GlobalQuery) {

    Boolean   docAdded = false   // document location status

  While more elements exist in GlobalQuery {
      get next element

      //root elements
      if RootElement = True { //root column
          if docAdded  = False { // check if document location has been added to the root element
                  for each document instance  {
                        add document("location")   // add doc location from "data" column in the mapping table
                  }
                 docAdded = true
          }
       }

      // binding elements to local namespace
      if URI matches prefix-uri column
           bind prefix to element   // prefix is LHS in the prefix-uri column.

      For each rule in MappingRules
      {
                If LHS of rule matches this element
               {
                   Compute function on RHS of rule
                    Return constraint in emit clause  //Emit clause is explained in section 6.
                }
       }
   }
}
```

Figure 19. Query Rewriting Algorithm

The query rewriting application consumes the mapping file which is an XML representation of all the elements in the global schema and mapping rules. The query rewriting algorithm on a local source identified by its URI is given in Figure 19.

The method create_query_instance(LocalSource, MappingRules, GlobalQuery) creates an instance of the query on the local schema identified by its URI. Mapping rules and the global query are passed as parameters into the method as strings. When rewriting the global query for a local schema where the targetNamespace URI matches the URI for a given xmlns, the query instance is identified by its prefix. The schema fragment below shows that targetNamespace and it's URI matches the URI assigned to xmlns:a, therefore the query instance is referred to as instance 'a'.

```
<schema targetNamespace="http://www.7.6.3A.org"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:a="http://www.7.6.3A.org"
        elementFormDefault="qualified"/>
```

Each element in the query string is parsed and checked if it the root element. If it is the root element a document("location") is added for each document instance which identifies the XML data to be queried. The element is bound to the local namespace URI through its prefix. The mapping rules string is parsed and each rule is checked for that element. For every rule that is applicable to the element, the right hand side of the mapping rule is computed and the constraint given by the emit clause is returned.

For each element in the global schema, the mapping file lists all namespaces, referenced namespaces, namespace prefix and the corresponding URI's. The element <book> in the schema fragment below is not a root element. The namespace element illustrates that we are rewriting the query for query instance 'b' as indicated by the value of the 'prefix' attribute. The URI is also shown as well as the attribute called 'refonly'. The 'refonly' attribute is required whenever there is a child element of <namespace>, namely <referenced>.

```
<element name="book" root="false">
    <namespace prefix="b" uri="http://www.7.6.3B.org" refonly="false">
        <referenced>
            <namespace prefix="a" uri="http://www.7.6.3A.org"/>
        </referenced>
    </namespace>
  </element>
```

The only child element that may occur for the element <namespace> is <referenced>. The <referenced> element may contain one or more (1+) <namespace> child elements. The referenced namespace element contains the prefix and URI for any referenced element. The following XML fragment shows two book child elements where one element is imported.

```
<text>
    <book>Object Oriented Programming</book>
    <a:book>MVC Architecture</a:book>
</text>
```

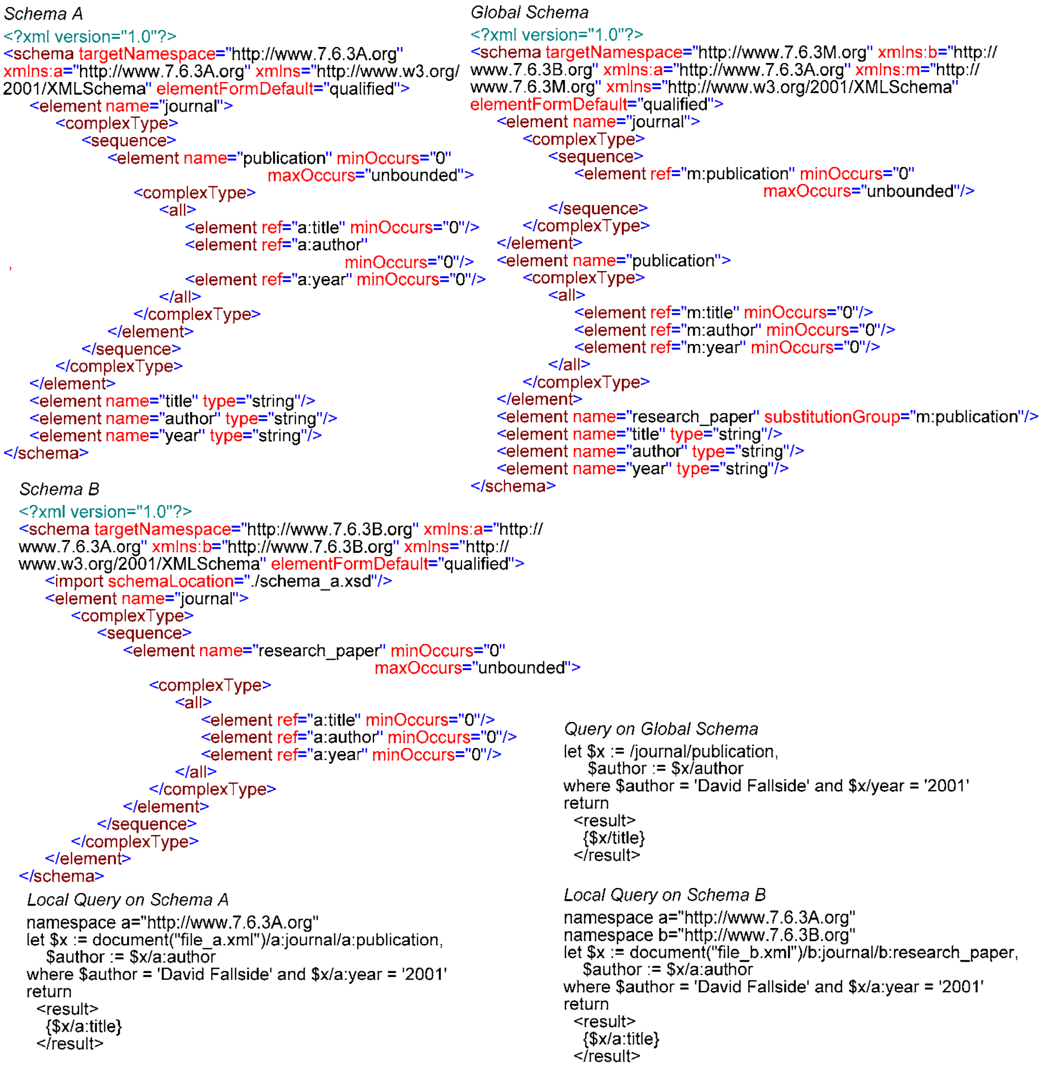
Finally, the mapping file also holds the information necessary to facilitate substitution groups. The substitution group information appears in the form of an element and is a sibling to the <namespace> element as an immediate child of the <element> element. An example is as follows:

```
<element name="book" root="false">
    <namespace prefix="b" uri="http://www.7.6.3B.org"/>
    <sub_group_ptr subelement="publisher"/>
</element>
```

Once the mapping file is consumed by the digester, the XML elements and attributes are stored as Java objects in a tree like structure. The parser then analyzes the XQuery and ensures its syntax is correct. The parser moves along the query string and binds each element with the correct namespace prefix depending on the target namespace for the query being rewritten. If there are two distinct namespace prefix's that must bind with a single element, the query rewriter systematically takes care of this action. Similarly, if an element must be substituted for another element of some other name (substitution Group), the query rewriter also handles this while binding the element with its corresponding namespace prefix. The result is a rewritten global query that is transformed into a local query and this local query applies to one of the various local schemas. The query will return the data from respective data sources.

The query rewriter is based on the SAX parser and the time it takes to rewrite the queries is much faster than it would take with the DOM parser. This ensures that the query rewriter is fast and robust. The implementation is completely written in Java and may be run as an installed program or deployed as a web based servlet.

We illustrate the use of the mapping table in the query rewriting process using the example query presented in Figure 20.



Schema A

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.7.6.3A.org"
xmlns:a="http://www.7.6.3A.org" xmlns="http://www.w3.org/
2001/XMLSchema" elementFormDefault="qualified">
    <element name="journal">
        <complexType>
            <sequence>
                <element name="publication" minOccurs="0"
                                            maxOccurs="unbounded">
                    <complexType>
                        <all>
                            <element ref="a:title" minOccurs="0"/>
                            <element ref="a:author"
                                                  minOccurs="0"/>
                            <element ref="a:year" minOccurs="0"/>
                        </all>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
    <element name="title" type="string"/>
    <element name="author" type="string"/>
    <element name="year" type="string"/>
</schema>
```

Global Schema

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.7.6.3M.org" xmlns:b="http://
www.7.6.3B.org" xmlns:a="http://www.7.6.3A.org" xmlns:m="http://
www.7.6.3M.org" xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <element name="journal">
        <complexType>
            <sequence>
                <element ref="m:publication" minOccurs="0"
                                             maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>
    <element name="publication">
        <complexType>
            <all>
                <element ref="m:title" minOccurs="0"/>
                <element ref="m:author" minOccurs="0"/>
                <element ref="m:year" minOccurs="0"/>
            </all>
        </complexType>
    </element>
    <element name="research_paper" substitutionGroup="m:publication"/>
    <element name="title" type="string"/>
    <element name="author" type="string"/>
    <element name="year" type="string"/>
</schema>
```

Schema B

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.7.6.3B.org" xmlns:a="http://
www.7.6.3A.org" xmlns:b="http://www.7.6.3B.org" xmlns="http://
www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <import schemaLocation="./schema_a.xsd"/>
    <element name="journal">
        <complexType>
            <sequence>
                <element name="research_paper" minOccurs="0"
                                               maxOccurs="unbounded">
                    <complexType>
                        <all>
                            <element ref="a:title" minOccurs="0"/>
                            <element ref="a:author" minOccurs="0"/>
                            <element ref="a:year" minOccurs="0"/>
                        </all>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
</schema>
```

Query on Global Schema

```
let $x := /journal/publication,
    $author := $x/author
where $author = 'David Fallside' and $x/year = '2001'
return
  <result>
   {$x/title}
  </result>
```

Local Query on Schema A

```
namespace a="http://www.7.6.3A.org"
let $x := document("file_a.xml")/a:journal/a:publication,
   $author := $x/a:author
where $author = 'David Fallside' and $x/a:year = '2001'
return
  <result>
   {$x/a:title}
  </result>
```

Local Query on Schema B

```
namespace a="http://www.7.6.3A.org"
namespace b="http://www.7.6.3B.org"
let $x := document("file_b.xml")/b:journal/b:research_paper,
   $author := $x/a:author
where $author = 'David Fallside' and $x/a:year = '2001'
return
  <result>
   {$x/a:title}
  </result>
```

Figure 20. Querying synonym elements

The query on the global schema is stated as "return all the publications of David Fallside in the year 2001".

Table 4. Mapping Table

Element	Rule	Root	Ref(only)	Attribute	Prefix – URI	Data
title		false	b → a		a : http://www.7.6.3A.org	a → file_a.xml b → file_b.xml
author		false	b → a		a : http://www.7.6.3A.org	a → file_a.xml b → file_b.xml
year		false	b → a		a : http://www.7.6.3A.org	a → file_a.xml b → file_b.xml
publication	substitutionGroup → b:research_paper	false			a : http://www.7.6.3A.org	a → file_a.xml
research_ paper		false			b : http://www.7.6.3B.org	b → file_b.xml
journal		true			a : http://www.7.6.3A.org b : http://www.7.6.3B.org	a → file_a.xml b → file_b.xml

The mapping table for the global schema generated during integration process is given in Table 4. The query rewriter parses the query and arrives at the /journal node. It immediately looks up the namespaces in which the element exists. From the 'Prefix-URI' column in the mapping table, it finds that two query instances must be created. The two instances reference the URI. For simplicity in this explanation, we will use the prefix as our identifier and call the query instances 'a' and 'b'. Upon reaching the 'journal' element, we notice that the element root status is 'true' therefore we must add to the XQuery expression, the document location. The location of the XML document(s) may be found in the 'Data' column. The first query instance 'a' is transformed in the following manner. Immediately following the /journal element is the /publication element. The query rewriter takes both elements and begins to transform them into qualified elements. The first line of the query is transformed to:

```
let $x := document("file_a.xml")/a:journal/a:publication,
```

The /author element which is the next qualified element is bound to the prefix from the 'Prefix-URI' column in the mapping table. The second line of the query is translated to:

```
$author := $x/a:author
```

Similarly the remaining elements in the query instance "a" are bound to the proper prefix and the query remapped as:

```
namespace a=http://www.7.6.3A.org
Let $x := document("file_a.xml")/a:journal/a:publication,
    $author := $x/a:author
Where $author = 'David Fallside' and $x/a:year = '2001
Return
<Result>
{$x/a:title}
 </Result>
```

For the query instance "b" the /journal element requires a document location (since it is the root element) and a prefix. The prefix 'b' will bind with the elements as indicated in the 'Prefix-URI' column. For the remapping of the first element, we will have the following:

```
Let $x := document("file_b.xml")/b:journal/publication
```

The /publication element has a mapping rule associated with it given in the "Rule" column of the mapping table. The rule states that the element /publication must be substituted for the element 'research_paper' within the instance 'b'. Consequently, the first line of the query is rewritten as follows:

```
Let $x := document("file_b.xml")/b:journal/b:research_paper,
```

The /author element exists within “b” but is a referenced element as indicated by the “Ref(only)” column in the mapping table, which comes from some other URI – namely, a : http://www.7.6.3A.org. In this case the /author element will bind with the prefix ‘a’ as the element is imported. For the elements /year and /title, the mapping table shows that they both exist in “b” as reference elements that must take the namespace prefix of ‘a’. The final remapped query for query instance “b” is then rewritten as:

```
Let   $x := document("file_b.xml")/b:journal/b:research_paper,
 $author := $x/a:author
Where $author = 'David Fallside' and $x/a:year = '2001'
Return
  <Result>
    {$x/a:title}
  </Result>
```

7.1 XML Data Integration

The whole implementation of this system is divided into four main categories – schema integration phase, mapping table construction, querying phase and result integration. These modules as shown in Figure 24 and here we briefly provide an overview of the system modules and then present some details on the querying module. Sun Microsystems Java is used to implement the system to exploit the cross-platform independence feature of the language. Apache Xerces XML Parser is used to parse XML documents. IPSI XQuery engine is used to run the XQuery queries on the XML documents. Many other tools such as TextPad, XMLSPY, JBuilder are used for constructing the data sets and as a development environment. A set of classes are created to model the XML documents, the XSDM notation of XML schema documents, and the XQuery queries. The implementation has the following modules as shown in Figure 21. Here we briefly provide an overview of the system modules and then presented some details on the querying module.

1. **XSDM – XML Schema Data Model.** Given any XML Schema document, the system first construct the proposed XSDM data model using the help of graphical representation of the XSchema.
2. **XML Schema Integration.** Given any two XML Schemas and ontology, the XML Schema files and the ontology file are parsed, and the XML schemas are shown graphically using the XSDM graphical notation along with their XML documents. Next, the system finds the corresponding data elements and attributes and resolve conflicts such as data type constraints, structural conflicts and key conflicts from the local schemas, and the user’s input is validated based on the ontology available. Once the conflicts are resolved, the integrated schema is generated. The robustness of the integrated schema is displayed by validating the local documents using the global schema.
3. **Generating Local Queries.** The global query on the integrated schema is read from the user and the local queries are generated by resolving the conflicts that arrive due to global predicates. The generated local queries are used to query the local XML documents that are validated with the local schemas.
4. **XML Data Integration.** The integration process use the correspondences information. In cases where a global predicate exists, the data need to be integrated and then queried again using the global query to obtain the result. The correctness of the integrated result obtained is demonstrated by querying a subset of the whole data available using a global query. The results obtained can be stored in a persistent storage.

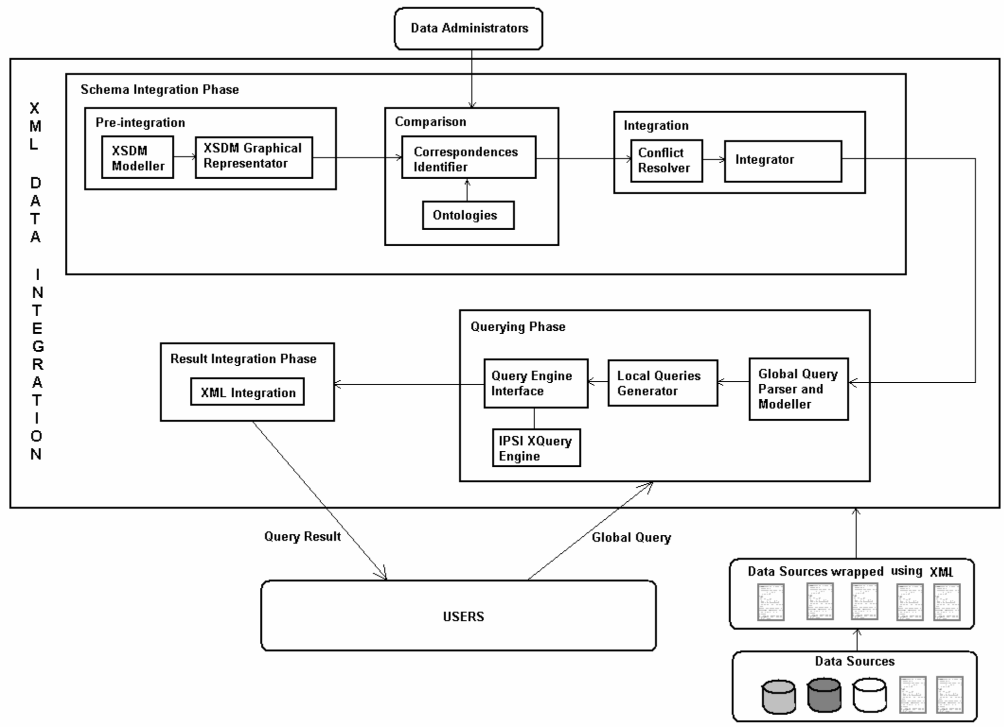
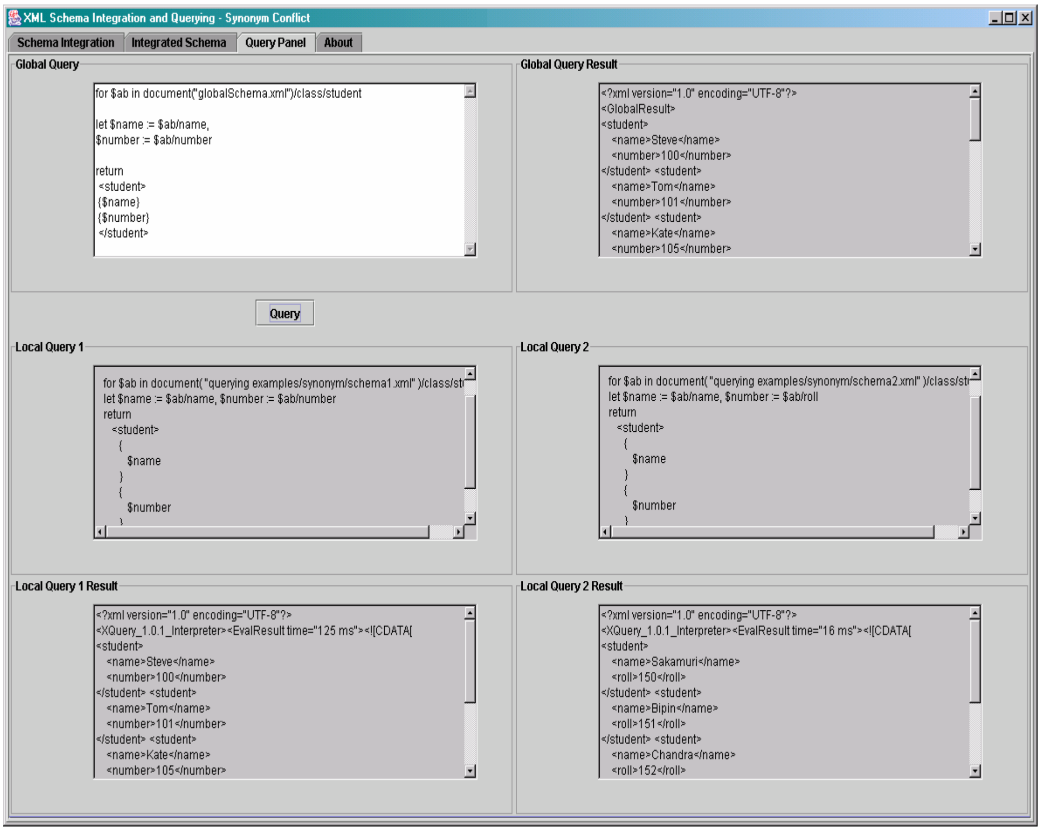


Figure 21. Implementation Level XML Integration System Architecture

In the querying phase, the user inputs the query on the integrated view of the schemas. The query is entered into the system using the "Querying Panel" shown in Figure 22. The query entered by the user is parsed and rewritten as local queries and are executed on the local XML documents. The obtained results are parsed and are integrated. This is the only phase where the integration of the XML data is performed. Generally, the size of the query results, which is small as compared to the huge size of the actual data and knowledge of the local schemas make the process of XML data integration feasible. To integrate the XML data, the ontology is used and the data administrators' input received during the schema integration phase are used to resolve the conflicts. For example, in few cases, where a substitution element is present, while rewriting the global query into local queries these elements name in the local queries should be changed. After getting the results back, these element names should be converted back to the original names used in global query. Most of the XML data integration methodology is similar to the schema integration except in the case of global predicates. In case of global predicates, we need to get the result from local queries, integrate the results and then query the integrated result with global predicate. The integrated XML data is validated with the integrated schema. A sub module that allows the user to store the result of his query to any persistent storage is made available.



Global Query

```
for $ab in document("globalSchema.xml")/class/student

let $name := $ab/name,
$number := $ab/number

return
 <student>
{$name}
{$number}
 </student>
```

Global Query Result

```
<?xml version="1.0" encoding="UTF-8"?>
<GlobalResult>
<student>
   <name>Steve</name>
   <number>100</number>
</student> <student>
   <name>Tom</name>
   <number>101</number>
</student> <student>
   <name>Kate</name>
   <number>105</number>
```

Local Query 1

```
for $ab in document( "querying examples/synonym/schema1.xml" )/class/st
let $name := $ab/name, $number := $ab/number
return
  <student>
   {
     $name
   }
   {
     $number
   }
```

Local Query 2

```
for $ab in document( "querying examples/synonym/schema2.xml" )/class/st
let $name := $ab/name, $number := $ab/roll
return
  <student>
   {
     $name
   }
   {
     $number
   }
```

Local Query 1 Result

```
<?xml version="1.0" encoding="UTF-8"?>
<XQuery_1.0.1_Interpreter><EvalResult time="125 ms"><![CDATA[
<student>
   <name>Steve</name>
   <number>100</number>
</student> <student>
   <name>Tom</name>
   <number>101</number>
</student> <student>
   <name>Kate</name>
   <number>105</number>
```

Local Query 2 Result

```
<?xml version="1.0" encoding="UTF-8"?>
<XQuery_1.0.1_Interpreter><EvalResult time="16 ms"><![CDATA[
<student>
   <name>Sakamuri</name>
   <roll>150</roll>
</student> <student>
   <name>Bipin</name>
   <roll>151</roll>
</student> <student>
   <name>Chandra</name>
   <roll>152</roll>
```

Figure 22. Querying Panel

8. Conclusions

The problem of heterogeneous data integration is solved using XML Schema by mapping the local XML schema to an integrated view of the various data sources. Unlike other systems that were developed which use an integrated view which should be first created and then mapped, in this research the integrated view is generated by using a minimal user input. The mapping process is made automatic and the integration process is scalable in the sense that any number of schemas can be integrated. The user queries on the integrated view are handled by the mechanism of rewriting the queries and executing them on local data sources. The query rewriting phase does not need any user input. We have design and developed a detailed graphical representation strategy for XML Schema and a data model for XML Schema has been defined. We have presented a query rewriting mechanism using semantic mapping for XML Schema integration. The rewriting of queries onto the local schemas requires mapping rules to remove the semantic differences between the local schemas. We have presented the mapping rules and strategies to rewrite a global query into queries on local XML documents validated by local schemas. We have also discussed the implementation of the system.

It is possible to use the XML Integration process given in this paper to integrate multiple DTDs. This possibility exists because the capabilities and expressiveness of the document type definition structure are essentially a subset of those of the XML Schema structure. Such an integrated DTD schema can be output in either DTD or XML Schema format. As well, the process can be used to integrate a DTD with an XML Schema and vice versa. This is again possible because of the fact that the DTD is a subset of the XML Schema. However, such integration would need to be output as an XML Schema as a given XML Schema may contain structures that cannot be expressed in a DTD format such as the *all*, namespaces, and most datatypes.

The XML documents may not have an associated DTD or XML Schema as it is not mandatory for such documents to have a schema file attached to it. It would be interesting to extract the XML Schema from such documents and integrate the XML data from these local XML Schemas and the integrated document to be validated through the global schema. We are looking into ways to maintain the currency of the global schema in case the local schemas change.

Acknowledgement: The authors would like to thank Bipin Sakamuri, Eric Chaudhry, Louise Lane for implementing the system and suggestions for improvement during this research and system implementation over the period of two years. The authors would also like to thank all the reviewers who helped improving the contents in the paper.

References

[1] S. Adali, K. Candan, Y. Papakonstantinou and V.S. Subramanian, Query Caching and Optimization in Distributed Mediator Systems, In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Canada, June 1996, 25(2), pp. 137-148.

[2] R. Behrens, A Grammar Based Model for XML Schema Integration, In Proc. British National Conference on Databases (BNCOD), 172-190, Exeter, UK, July 3-5, 2000.

[3] C. Bontempo, DataJoiner for AIX, IBM Corporation, 1995.

[4] S. Boag, D. Chamberlin, M.F. Fernandez, D. Florescu, J. Robie and J. Simeon, XQuery 1.0: An XML Query Language, W3C Working Draft, 15 November 2002. http://www.w3c.org/TR/xquery/.

[5] C. Batini, M. Lenzerini and S. B. Navathe, A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys, Vol. 18, No. 4, December 1986.

[6] P.V. Biron and A. Malhotra, Eds. XML Schema Part 2: Datatypes Second Edition - *W3C Recommendation* 28 October 2004. http://www.w3.org/TR/xmlschema-2/

[7] V. Chritophides, S. Cluet and J. Simon, On Wrapping Query Languages and Efficient XML Integration, In Proc. ACM SIGMOD, Dallas, Texas, 2000.

[8] K.C. Chang and H. Garcia-Molina, Conjunctive Constraint Mapping for Data Translation, In Proceedings of the 3rd ACM International Conference on Digital Libraries, Pittsburgh, PA, USA, June 1998, pp. 49-58.

[9] K.C. Chang and H. Garcia-Molina, Mind You Vocabulary: Query Mapping Across Heterogeneous Information Sources, In Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA, June 1999, pp. 335-346.

[10] K.C. Chang and H. Garcia-Molina, Approximate Query Translation Across Heterogeneous Information Sources, In Proceedings of 26th International Conference on Very Large Data Bases (VLDB), Cairo, Egypt, September 2000, pp. 566-577.

[11] Moh, Chuang-Hue, Ee-Peng Lim and Wee-Keong Ng, Re-engineering Structures from Web Documents, Proceedings of the Fifth ACM Conference on Digital Libraries, June 2-7, 2000, San Antonio, TX, USA.

[12] A. Doan, P. Domingos and A. Halevy, Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach, ACM SIGMOD 2001 Electronic Proceedings, Santa Barbara, California, USA, May 2001.

[13] D. Draper, A.Y. Halevy and D.S. Weld, The Nimble Integration Engine, Industrial Track Paper, In ACM SIGMOD 2001 Electronic Proceedings, Santa Barbara, California, USA, May 2001.

[14] L. Ekenberg and P. Johannesson, Conflictfreeness as a Basis for Schema Integration. SdeLphi at Telia Research AB. NUTEK project SISI. Sweden. 1995.

[15] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman and J. Widom, The TSIMMIS project: Integration of heterogeneous information sources, Journal of Intelligent Information Sytems, 8(2) March 1997, pp. 117-132.

[16] M.A. Hernandez, R.J. Miller, L. Haas, L. Yan, C.T.H. Ho and X. Tian, Clio:A Semi-automatic Tool for Schema Mapping, System Demonstration, ACM SIGMOD 2001 Electronic Proceedings, Santa Barbara, California, USA, May 2001.

[17] L.H. Haas, R.J. Miller, B. Niswanger, M.T. Roth, P.M. Schwarz and E.L. Wimmers, Transforming Heterogeneous Data with Database Middleware: Beyond Integration, IEEE Data Engineering Bulletin, 22(1):31-36, 1999.

[18] M. Lamprecht, Air Canada's IT Integration, April 2001.
http://www.gsetoday.com/issues_by_month/articles/090346.html

[19] A.Y. Levy, A.O. Mendelzon, Y. Sagiv and D. Srivastava, Answering Queries Using Views, In Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), San Jose, California, May 1995, pp. 95-104.

[20] P.A. Larson and H.Z. Yang, Query transformation for PSJ-queries. In Proceedings of the 13th International Conference on Very Large Data Bases (VLDB), Brighton, England, September 1987, pp. 245-254.

[21] R.J. Miller, Using Schematically Heterogeneous Structures, Proc. of the ACM SIGMOD Intl. Conf. on the Management of Data, 27(2):189-200, Seattle, WA, USA, June 1998.

[22] R.J. Miller, L.M. Haas and M. Hernandez, Schema Mapping as Query Discovery, In Proceedings of the 26th International Conference on Very Large Databases (VLDB), Cairo, Egypt, September 2000, pp. 77-88

[23] R.J. Miller, M.A. Hernandez, L.M. Haas, L. Yan, C.T.H. Ho, R. Fagin and L. Popa, The Clio Project: Managing Heterogeneity, SIGMOD Record, 30(1), March 2001, pp. 78-83.

[24] A. Malhotra, J. Melton and N. Walsh, Eds. XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Candidate Recommendation 3 November 2005. http://www.w3.org/TR/xpath-functions/

[25] Y. Papakonstantinou, H. Garcia-Molina and J. Widom, Object Exchange Across heterogeneous Information Sources, Proc. IEEE Conf. on Data Engineering, 251-260, Taipei, Taiwan, 1995.

[26] R. Pottinger and A.Y. Halevy, MiniCon: A Scalable Algorithm for Answering Queries Using Views, VLDB Journal, 10(4): 270-294, 2001.

[27] Christine Parent and Stefano Spaccapietra. Issues and Approaches of Database Integration. *Communications of the ACM*. Vol. 41, No. 5, pp. 166-178. 1998.

[28] Y. Papakonstantinou and V. Vassalos. Query Rewriting for Semistructrued Data, In Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA, June 1999, 455-466.

[29] S. Ram and V. Ramesh, Schema Integration: Past, Present and Future, In A. Elmagarmid, M. Rusinkiewicz and A. Sheth, editors, Management of Heterogeneous and Autonomous Database Systems, Morgan-Kaufmann, San Mateo, CA, 1998.

[30] M.T. Roth and P.M. Schwarz, Don’t Scrap it, Wrap it! A Wrapper Architecture for Legacy Data Sources, In Proceedings of 23rd International Conference on Very Large Databases (VLDB), Athens, Greece, August 1997, pp. 266-275.

[31] Tukwila Data Integration System, http://data.ca.washington.edu/integration/tukwila/index.htm

[32] H.S. Thompson, D. Beech, M. Maloney and N. Mendelsohn, Eds. XML Schema Part 1: Structures Second Edition - *W3C Recommendation*. 28 October 2004. http://www.w3.org/TR/xmlschema-1/

[33] Tomasic, L. Rascid and P. Valduriez, Scaling Heterogeneous Databases and the Design of Disco In Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS), Hong Kong, IEEE Computer Society, May 1996, 449-457.

[34] L. Yan, R.J. Miller, L. M. Haas and R. Fagin, Data-Driven Understanding and Refinement of Schema Mappings, ACM SIGMOD 2001 Electronic Proceedings, Santa Barbara, California, USA, May 2001.

Biography

	Sanjay Kumar Madria received his Ph.D. in Computer Science from Indian Institute of Technology, Delhi, India in 1995. He is an Associate Professor, Department of Computer Science, at University of Missouri-Rolla, USA. He has published more than 120 Journal and conference papers in the areas of XML data management, mobile databases, and sensor data security. He guest edited WWW Journal and Data and Knowledge Engineering Sp. Issues on Web data management. He has served as General Chair, Program Chair, PC member of various database conferences and workshops and reviewer for many reputed database journals such as IEEE TKDE, IEEE Computer, IEEE TMC, ACM Internet Computing among others. He was invited keynote speaker in Annual Computing Congress in Canada and invited speaker in Conference on Information Technology. He is IEEE Senior Member.
	Kalpdrum Passi received his Ph.D. in Parallel Numerical Algorithms from Indian Institute of Technology, Delhi, India in 1993. He is an Associate Professor, Department of Mathematics & Computer Science, at Laurentian University, Ontario, Canada. He has published many papers on Parallel Numerical Algorithms in international journals and conferences. He has collaborative work with faculty in Canada and US and the work was tested on the CRAY XMP’s and CRAY YMP’s. He has more recently working in XML integration technology. He is a member of the ACM and IEEE Computer Society.
	Sourav S Bhowmick, received his Ph.D. in computer engineering in 2001. He is currently Associate Professor in the School of Computer Engineering, Nanyang Technological University. He also holds the position of Singapore-MIT Alliance (SMA) Fellow. His current research interests include XML data management, biological data management, interface-driven data management, web data management and semistructured data mining. He has published more than 90 papers in major international database conferences and journals such as VLDB, ICDE, WWW, TKDE, and DKE. Sourav is serving as a PC member of various database conferences and workshops and reviewer for various database journals. He is also serving as a program chair/co-chair of several international workshops in biological and XML data management. He is a member of the editorial boards of several international journals. He has co-authored a book entitled “Web Data Management: A Warehouse Approach" (Springers Verlag, 2003). Sourav is a member of ACM and an affiliated member of IEEE.