Data Integration in a Three-Layer Mediation Framework

Qasem Kharma, Raimund K. Ege, Onyeka Ezenwoye, Li Yang
Secure System Architecture Laboratory
School of Computer Science
Florida International University, Miami, FL 33199
{qkhar002|ege|oezen001|lyang03}@cs.fiu.edu

Abstract

Our distributed mediation architecture employs a layered framework of presence, integration, and homogenization mediators. In order to find a mediation path from a client request to data sources, a Distributed Hash Table (DHT) algorithm is deployed in the integration layer. A designated global-mediator in the integration layer initiates the keyword-based matching decomposition of the request with the use of the DHT. It generates an Integrated Data Structure Graph (IDSG), creates association and dependence relations between nodes in the IDSG, and then it generates a Global IDSG (GIDSG). GIDSG is used to stream data from the mediators in the homogenization layer where they connect to the data sources. The architecture is dynamic, scalable, and does not have any central point of failure. In this paper we present our research on the use of the GIDSG for the integration of data in our three-layer mediation architecture.

Keywords: mediator, middleware, software architecture, integration, P2P, DHT.

1. Introduction

The proliferation of modern information systems has enabled access to a multitude of disparate but often related information. This information - in the form of multimedia data - is stored on and accessed from various kinds of heterogeneous devices. There is a need for mediators [20] that harmonize and present the information available in heterogeneous data sources. This harmonization comes in the form of identification of semantic similarities in data while masking their syntactic differences. Relevant and related data is then integrated and presented to a higher layer of applications. The sourcing, integration, and presentation of information can then be seen as logically separated mediator roles and forms the basis for the three-layer mediator architecture [5, 13, 12].

The research reported on here is a part of our ongoing effort to define and build a multi-layered mediator architecture that will provide a dynamic and scalable framework for information delivery. The architecture is based on three layers; presence, integration and homogenization. The high-level goal of the presence layer is acceptance of requests (queries) from clients and the presentation of the results of those queries. The intermediate level goal of this layer is to make sure the quality of service (QoS) criteria of these requests is met [12]. The main steps taken to achieve this include the monitoring and advertising the QoS parameters of the client/query, the election of a global-mediator for the query, caching and buffer of result stream and if necessary the manipulation of the results to suit the desired QoS. The data interchange language between our mediators is XML. Queries are converted to XML in the presence layer before the search; results are converted back from XML to the desired format in this layer. The decomposition of the XML query, its distribution (search) and integration of the results is done at the integration layer. The third layer, homogenization, is where connection to actual data sources is established. Data from these heterogeneous data sources are converted from their individual data formats to a common data language of the mediators, in this case XML. The mediators in this layer act as wrappers to the data sources. Figure 1 depicts the framework. The integration layer consists of mediators that successively decompose a XML request into smaller XML requests that are closer to the data sources that are served-up by the homogenization layer.

The focus of this paper is the integration layer. The integration layer represents a special kind of knowledge which is the composition/decomposition of XML schemas and routes. Instead of maintaining a central schema repository server which manages and handles all schemas, we opt for a distributed search mechanism that uses the mediators in the integration layer as nodes in a Distributed Hash Table (DHT).

DHT algorithms can be classified into three categories [2]: Skiplist-like routing algorithms such as the Chord algorithm [18], Routing-in-Multiple dimensions algorithms such as the CAN algorithm [14], and Tree-like algorithms such as the Pastry algorithm [16]. DHT algorithms can also be classified according to their basic routing geometries [7], such as tree, hypercube, butterfly, ring, XOR, and hybrid. The general idea of DHT is that each node maintains
information about its neighbors in the system. No node has all the information, and some information is duplicated so when a node fails, the whole system will not fail.

Section 2 covers the related work in the mediation and data integration in particular. Section 3 describes the three-layer architecture and the data integration process. In section 4, we use an example to demonstrate the schema generation, distribution and integration in the architecture. Section 5 concludes the paper.

2. Related Work

A lot of work has been done on mediation systems [6, 19, 21, 15, 11, 10, 9]. As stated in [11, 9], most of these architectures however are centralized, in that, there is a single mediator through which query decomposition, result integration and access to heterogeneous sources is achieved. Like our architecture, some [11, 19, 21] mediator architectures are distributed and mediators are able to access and communicate with each other. [21] is a two-tier mediation model that comprises a homogenization and integration layer with mediators in each that playing similar roles as in our architecture. [11] on the other hand does not have any restrictions on mediator functions as each mediator can play the role of homogenization and/or integration. There is also no restriction as to the number of mediator tiers. [11] and [21] employ a similar integration process for homogenized sources [11].

Our architecture is a three-layer model that consists of the presence, integration, and homogenization layers. Our architecture does not only accommodate heterogeneous data sources but also with the aid of the presence layer mediators adapts to the heterogeneous nature of the client devices by taking into account various QoS issues of the client. [11] is a peer mediation system much like ours but unlike our model, it does not employ the use of the DHT in the distribution of source schema and peer lookup.

Most of the aforementioned frameworks use trees or graphs to integrate heterogeneous data sources and hide unrelated detail from the integration process. Our framework uses the IDSG to integrate schema (structure).

[1] proposes a middleware that is based on order label tree to integrate heterogeneous data. The authors introduced formal description of the correspondence between the tree nodes by using two predicates: "is", which links similar real world entities, and "concat", which is a standard concatenation. Moreover, a rule-based language was introduced to define the correspondences among heterogeneous data sources.

[3] describes structural recursion functions on labeled trees that can be used for unstructured data. The defined functions were also applied to cyclic structure. [3] can be considered as fundamental theories of using tree/graph in data integration.

[8] integrates XML-based sources Information Integration Agents(IIAs). Unlike our system, it uses inference to generate global view. In general, [8] and our architecture covert XML schemas/DTDs to their equivalent trees and integrate those trees by running some operation.

3. Three-layer mediator architecture

The three-layer mediator architecture consists of the presence, integration, and homogenization layers. The main objective of designing the three-layer architecture is to design a scalable, dynamic, fault-tolerant, secure system in which work load is distributed over chains of connected mediator. A system that is able deal with the heterogeneous nature of data sources as well as that of the client devices that access them.

![Figure 1: Three-Layer Mediator Architecture](image)

3.1 The Architecture

Mediators form a virtual database between client and data stores [11]. The path from the client to the desired data source(s) will comprise a series of mediators. This path forms a tree with which the data is integrated. Our system has been designed to give a high degree of autonomy to the data sources. This gives the data stores the freedom to join and leave the federation of mediated databases as they wish. This also allows the individual data stores to modify, maintain their content and schemas independently. The system thus exhibits a behavior that is similar to peer-to-peer (P2P) architectures. Due to the dynamic nature of the topology, this path from the client to the data stores that forms the mediation tree cannot be static but must be dynamically constructed during the search.
This dynamic construction also allows the mediators to form a path that best meets the QoS requirements of the client application.

Our research is focused on a dynamic mediation architecture that attempts to homogenize low layer data sources while meeting the QoS requirements of the heterogeneous client devices at the top layer. The presence layer is the interface to the client, which can be any computing device such as a PC, a PDA, or any special purpose devices. Caching and buffering data streams are some of the functions performed in this layer. The integration layer functions include analyzing queries, finding appropriate data sources, and forming the Integration Data-Structure Graph. In the homogenization layer, translation of heterogeneous data sources into XML format is done [5, 13, 12].

Within this architecture, we differentiate between three kinds of mediators. They are, the presence-mediators deployed in the presence layer, the mediator-composers deployed in the integration layer, and mediator-connectors in the homogenization layer. For the rest of this writing, we will refer to mediator-composers and mediator-connectors as composers and connectors, respectively. The client first connects to a presence-mediator which will perform presence layer functions. The presence-mediator will elect a special kind of composer called the global-mediator; this global-mediator will be responsible for the particular query for which it was elected. It will be responsible for composing the path from the client to the data sources that represents the tree for the composition of the query result. A new global-mediator is elected for every new request based on predefined QoS criteria [12].

At the lowest level of the mediator hierarchy, connectors connect to the actual databases and are the interface through which these data sources are accessed. Unlike composers, connectors will not play any role in routing a request. They map the local database schemas to XML schemas and convert their data according to those XML schemas.

Upon receipt of a query, the global-mediator forwards the request to other composers in order to find the results. It uses the DHT, which is implemented in the composer, to determine which composers to send the queries to [12]. More than one composer will need to cooperate to handle a single request. Once the desired connector, which maps the requested data, is reached, the Global Integrated Data-Structure Graph (GIDSG) will be composed by the global-mediator, and this tree will be used to integrate data from multiple sources and accessing those sources.

### 3.2 Handling a request in the three-layer architecture

When a presence-mediator receives a request, it starts an election to choose the most suitable composer to act as global-mediator to that request. The presence-mediator then converts the client request into XML document and forwards the XML document to the global-mediator. After that, the presence-mediator will wait until it receives a response from the global-mediator.

The global-mediator will coordinate with other mediator-composers to find path(s) from the global-mediator to mediator-connector(s) which map the required data. First, the global-mediator will break the request into tokens, a sequence of digits and characters. It will then try to match these tokens with some tags (elements' identifiers) in the stored XML schema. Recall that the tags in the integration layer are distributed and maintained by the DHT. After the decomposition process ends, the coordinated composers will return the corresponding XML trees and their connectors' addresses.

The composers which decompose the schema will contact the connectors to get the best sub-tree(s) corresponding to its XML schema. Since the data may be distributed over many connectors, the composers will integrate those sub-trees into one tree. The basic idea in the integration is to find the corresponding trees which are stored in the connectors and, then, to create dependency relations between the nodes in the retrieved trees based on the mapping rules stored in the connectors. At the end of this process, the global-mediator will have the Global Integrated Data-Structure Graph (GIDSG) and the IP addresses of the corresponding connectors.

### 3.3 Using a DHT in the architecture

All composers need to cooperate in order to find the connector(s) to the desired data source(s). In order to find the connector(s), the route from the global-mediator through composers can be found using DHT instead of having a central repository of the connectors' XML schemas. Although it is possible to use one of the aforementioned DHT algorithms by defining what values will be mapped, we elected to build a hybrid algorithm of Chord and Pastry: this new algorithm maintains some features of both but adds important Quality of Service (QoS) criteria.

Unlike CFS [4] which is a file storage for blocks based on Chord, and PAST [17] which is a file storage for files based on Pastry, the mediator does not distribute the data in the data sources among the composers. The mediator system needs only to distribute pointers to the data which will be accessed through connectors. Hence, the first level of security is implemented in the connector, so only clients with right permissions can retrieve the XML schemas from the connectors and then access the data through the connectors.

Composers are distributed on a logical ring, like Chord. Unlike Chord, the composers maintain successor list, predecessor list, finger table, and a cache. This cache contains information about composer with links to recently accessed connectors. The cache is useful because, although
this mediation architecture is flexible, mediators should be
domain specific. In a specific domain therefore, there will
be some keywords that are frequently used in queries. For
instance, in a medical environment, words like patient,
name, xray, insurance, and so on will be repeated
frequently. The cache maintains a short list of the keywords
with the highest frequency of occurrence in queries.

In our architecture, each composer maintains some
keywords which are tags in XML schemas stored in some
connectors. When a system administrator adds a new data
source by starting a new connector, the connector will
convert the schema for its data source into an XML
schema. This schema is then sent to a composer. The
receiving composer will convert the XML schema into its
corresponding tree. Next, a hash function is used to map the
XML tags (elements) which are now nodes in the tree onto
keys which will be distributed over the peers.

Besides the schema of the data source, all connectors
will also contain their mapping rules as XML documents.
These mapping rules will be used to create associations
among tokens from different schemas. This process is
explained by an example in the next section.

4. Example

Connectors generate XML schema (shown here in tree
fashion). Composers then distribute the nodes of those trees
among the composers using a DHT algorithm. Let us
assume that in a mediation system we have two data
sources and six composers. On top of each data source,
there must be a connector, so there are two connectors
(Connector1 and Connector2). Connector1 maintains the
schema in Figure 2 which has the tree representation of
Figure 3 and all the mapping rules, and Connector2
maintains the schema in Figure 4.

For the sake of simplifying the example, we assume that
the composers are distributed over a logical ring in which
each composer knows its successor node. When a composer
receives a request, it will either find the requested
keywords in its DHT or else it forwards the keywords to its
successor.

The mapping rules are represented in XML documents
which contain the following information; the target data
source and the destination data source, the attributes, and
operation on attributes. For instance, Figure 5 represents
mapping rule for the “name” keyword for data source 1 into
the concatenation of the firstname and the lastname in the

3.4 Integration Process

When a new connector joins the system, the connector
will submit its XML schema to a composer. The composer
will tokenize the XML schema and run the hash function
on these tokens. Then, it will add those tokens as new keys
into the DHT. The values of the elements and attributes
fields in XML documents are used as tokens. For instance,
given a class record: ssn, name, case (diagnosis, test),
address, the equivalent XML schema would be as in Figure
2.

Besides the schema of the data source, all connectors
will also contain their mapping rules as XML documents.
These mapping rules will be used to create associations
among tokens from different schemas. This process is
explained by an example in the next section.

4. Example

Connectors generate XML schema (shown here in tree
fashion). Composers then distribute the nodes of those trees
among the composers using a DHT algorithm. Let us
assume that in a mediation system we have two data
sources and six composers. On top of each data source,
there must be a connector, so there are two connectors
(Connector1 and Connector2). Connector1 maintains the
schema in Figure 2 which has the tree representation of
Figure 3 and all the mapping rules, and Connector2
maintains the schema in Figure 4.

For the sake of simplifying the example, we assume that
the composers are distributed over a logical ring in which
each composer knows its successor node. When a composer
receives a request, it will either find the requested
keywords in its DHT or else it forwards the keywords to its
successor.

The mapping rules are represented in XML documents
which contain the following information; the target data
source and the destination data source, the attributes, and
operation on attributes. For instance, Figure 5 represents
mapping rule for the “name” keyword for data source 1 into
the concatenation of the firstname and the lastname in the
data source 2. The values of the attributes are considered as
tokens and are distributed over the composers.

Prior to distribution, a hash function is run over the
tokens to generate their key values. The generated keys of
the tokens are then distributed over the composers. The
range of keys assigned to each composer is determined by
the hash value of that composer’s unique identifier. This
identifier could be an IP address and the hash value is
generated by running a hash function on that IP address.
Thus, in our example (Figure 6) the composers contain
tokens with key values in their id ranges.  For instance,
Composer # 1000 contains tokens with key values (0,
1000]. In Figure 6, the tables contain three columns: the
tokens, the key values of the tokens and the connector ids.
The first column was added for clarity.  In practice this
column is not necessary.

Assume that the request in Figure 7 is generated by a
presence mediator. The values “name”, “diagnosis”, “test”,
and “address” are tokens that will be lookup in the
composers’ DHTs. If Composer # 5000  (in Figure 6) was
elected as a global mediator, then Composer # 5000 will
lookup its DHT for any of the tokens. If the only token
found is “address”, the rest of the tokens will be forwarded
to the next composer # 0000. The search process will
continue until all tokens are found, or the initiating
composer is reached. Table 1 shows the sequence of the
searching.

The global mediator will use this information to retrieve
the required schema from the connectors. In our example,
the global mediator will contact Connector1 and retrieve
the tree in Figure 3. Because some of the mapping rules
stored in Connector1 point to data source 2, the global
mediator will retrieve the schema in Figure 4 from
Connector2. Both schema of Figure 3 and Figure 4 will be
linked together using the mapping rules. As a result, the
Global Integrated Data Structure Graph GIDSG in Figure 8
will be generated. It will include all the necessary
associations.

<table>
<thead>
<tr>
<th>Seq</th>
<th>Composer key</th>
<th>Keyword value</th>
<th>Keyword key</th>
<th>Connector ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0000</td>
<td>name</td>
<td>5853</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>diagnosis</td>
<td>0365</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2000</td>
<td>test</td>
<td>1364</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5000</td>
<td>address</td>
<td>4563</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: the steps in which the tokens were
found and the destination connector(s).

In this paper we report a technique where a composer in
the 3-layer mediation architecture builds the Integrated
Data Structure Graph (IDSG) on-the-fly in the integration
layer. A special composer called Global Mediator adds
associations and refines the IDSG generating a Global
IDSG (GIDSG) which will be used to retrieve data from
connectors which are employed on top of data sources and
integrate the data. Then, integrated data will be sent to the
presence mediator to present the result to the client. In our
architecture the IDSG is dynamically built instead of
maintaining a global view, and the data integration process
is delayed to a later stage of the integration phase to
minimize network traffic in the system.

5. Conclusion

In this paper we report a technique where a composer in
the 3-layer mediation architecture builds the Integrated
Data Structure Graph (IDSG) on-the-fly in the integration
layer. A special composer called Global Mediator adds
associations and refines the IDSG generating a Global
IDSG (GIDSG) which will be used to retrieve data from
connectors which are employed on top of data sources and
integrate the data. Then, integrated data will be sent to the
presence mediator to present the result to the client. In our
architecture the IDSG is dynamically built instead of
maintaining a global view, and the data integration process
is delayed to a later stage of the integration phase to
minimize network traffic in the system.
Acknowledgements

This material is based on work supported by the National Science Foundation under Grant No. HRD-0317692.

References


