Lean Middleware

David A. Maluf
NASA Ames Research Center
MS 269/3
Moffett Field, CA 94035
(001)-650-604-0611
David.A.Maluf@nasa.gov

David G. Bell
USRA RIACS
NASA Ames Research Center
MS 269/3, Moffett Field CA 94035
(001)-650-604-500
Dbell@email.arc.nasa.gov

Naveen Ashish
USRA RIACS
NASA Ames Research Center
MS 269/3, Moffett Field CA 94035
(001)-650-604-2822
Ashish@email.arc.nasa.gov

ABSTRACT
This paper describes an approach to achieving data integration across multiple sources in an enterprise, in a manner that is cost efficient and economically scalable. We present an approach that does not rely on major investment in structured, heavy-weight database systems for data storage or heavy-weight middleware responsible for integrated access. The approach is centered around pushing any required data structure and semantics functionality (schema) to application clients, as well as pushing integration specification and functionality to clients where integration can be performed "on-the-fly".

Categories and Subject Descriptors
H.2.5 [**Heterogeneous Databases**]: Data integration, middleware, integrated access, economically scalable, – *new integration paradigm*

General Terms
Algorithms, Management, Design, Economics

Keywords
Middleware, Data Integration, XML, Economical

1 INTRODUCTION
Seamless integrated access to multiple, distributed, and heterogeneous information sources has been and continues to be a challenge for large organizations and enterprises. Data integration systems and integration middleware [3] that address this problem have been around for several years. The current middleware technology however, requires significant investment in 'heavy-weight' middleware for an application of any scale or requirements. For each integration application we need to define schemas or views for each source, and reconcile the schemas or form integrated global schemas or views to facilitate the integration. This approach, unfortunately, causes the IT cost for integration applications to increase linearly with the application size as shown in Fig 1. Our experience with building data integration systems in the NASA enterprise has led us to an approach to data integration that is cost-effective, economically scalable, and flexible. The experience of building data integration systems and applications for a variety of NASA problems ranging from aviation information systems to program management systems to NASA enterprise wide business analysis tools, tells us that the needs of different data integration applications are often very diverse. Applications might require data integration across anywhere from a handful of information sources to literally hundreds of sources. The data in any source could range from a few tables that could well be stored in a spreadsheet to something that requires a sophisticated DBMS for storage and management. The data could be structured, semi-structured, or unstructured. Also, the query processing requirements for any application could vary from requiring just basic keyword search capabilities across the different sources to sophisticated structured query processing across the integrated collection.

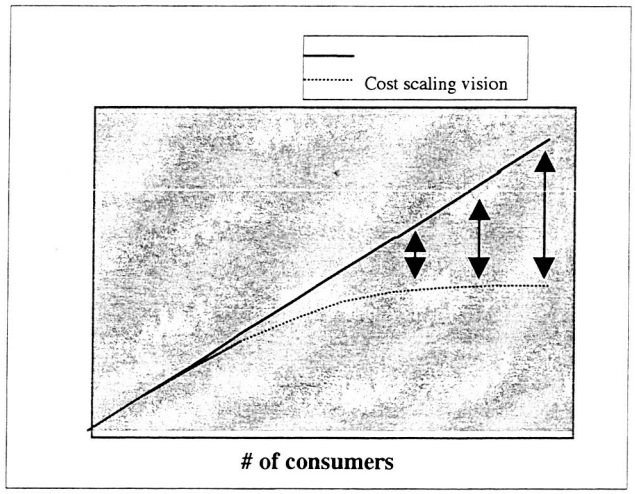

Fig 1. Costs of data integration

In this paper we present an approach to data integration that is cost-effective and scalable. The approach is based on key insights (outlined in the following section) that permit an integration approach that is more flexible and *adaptable* to different integration applications. The vast majority of integration applications at NASA have been built over data in documents, spreadsheets, reports and presentations as input. Also, the query

ACM SIGMOD '05, June 14–16, 2005, Baltimore, MD, USA.

processing (and result composition) requirements have largely or completely been focused on extracting particular sections from documents [1], composing new documents with sections from other multiple documents, or doing keyword based searches on documents. Our approach has thus been to develop a data management and integration system optimized for the above capabilities and avoiding investing in functionalities that are unnecessary for the common applications.

The following sections contain a description of this approach. The data storage approach is first described followed by a description of the system architecture. This is followed by a description of the query processing capabilities. Some integration applications built using this system are then described, concluding with a discussion on the relationship with other integration approaches.

2 A COST-EFFECTIVE and SCALABALE INTEGRATION APPROACH

The need for middleware and integration technology is inevitable, more so with the shifting computing paradigms as noted in [11]. The investment in schema management per new source integrated and heavy-weight middleware are reasons why user costs increases directly with the user benefit (as shown in Fig 1.), with the investment going to the middleware IT product and service providers. What is beneficial to end users however are integration technologies that truly demonstrate economies of scale as envisioned in Fig 1.

How is a cost-effective and scalable integration approach achieved ? Note first that the high IT cost of integration is often due to completely unnecessary investment in formally structuring information (with schemas) and using heavy-weight, "one-size-fits-all" integration middleware for any integration application. We begin by eliminating some tacit assumptions that seem to be holding for data integration technology, namely:

- *Data must always be stored and managed in DBMS systems*

  Actually, requirements of applications vary greatly ranging from data that can well be stored in spreadsheets, to data that does indeed require DBMS storage.

- *The database must always provide for and manage the structure and semantics of the data through formal schemas*

  Alternatively, the "database" can be nothing more than intelligent storage. Data could be stored generically and imposition of structure and semantics (schema) may be done by clients as needed.

- *Managing multiple schemas from several independent sources and interrelationships between them, i.e., "schema-chaos" is inevitable and unavoidable.*

  Alternatively, any imposition of schema can be done by the clients, as and when needed by applications.

- *Clients are too light-weight to do any processing. Thus a significant component of integration across sources must be, in a sense, "pre-compiled" and loaded into a centralized mediation component.*

  This assumption is based on a 1960s paradigm where clients had almost negligible computing power. Clients of today have significant processing power and sophisticated functionality can well be pushed to the client side.

As correctly noted in a recent Gartner research note [5], knowledge workers will continue to introduce new technologies and tools faster than enterprises can support them and the challenge thus is to strike a balance between the consolidated technologies that the enterprise can manage and support along with giving the knowledge workers new tools and capabilities they constantly crave for.

2.1 The NETMARK Data Storage and Integration Approach

We now describe NETMARK, a system that was designed by eliminating the above assumptions and based on the following tenets:

- The database will be nothing more than an intelligent storage component that stores the data but does not impose a formal structure or semantics is the form of schemas on the data. In other words it is "schema-less".
- Any imposition of schema on the data will be done at the client side.
- Any required integration across multiple sources will be done at the client and on the fly.

The following sub-sections contain a description of NETMARK. We describe data storage in NETMARK followed by a description of query capabilities and integrated access in NETMARK. It is only possible to provide a brief overview of the system and its functionality in this paper. The reader is referred to [6] and [7] for recent and detailed technical descriptions of the NETMARK system.

A high-level architectural overview of the NETMARK system is provided in Fig 2. The 'NETMARK XML Store' is the data storage component of the system. We begin with a description of data storage and management in the NETMARK XML Store as this is central to other aspects such as query processing described shortly after.

2.1.1 Data Storage and Management

NETMARK is designed to effectively store and manage structured data as well as semi-structured data found in documents, web-pages and spreadsheets. Structured data storage and management database systems have been around for several years, relational database systems (RDBMSs) for decades and object-relational database systems (ORDBMSs) in the last decade. For managing semi-structured data, we have seen a significant amount of activity in building XML data management systems in the last several years. The XML data management systems fall into two broad categories. One is based on an approach of building an XML data management system over a relational data management system [10]. Any XML documents to

[1] We use the term 'documents' to include documents in formats such as Word, PDF, HTML, XML or others, spreadsheets, presentations in powerpoint, reports, etc., henceforth

be stored are "shredded" into relational tables and stored as relational data. The other approach, called the native XML approach, is based on storing XML documents and structures in underlying tree structures corresponding to the XML documents [2]. Note that both approaches are "schema-centric" and "schema-dependant" in that the structure of the data stored in the database system depends on the structure of the XML document being stored.

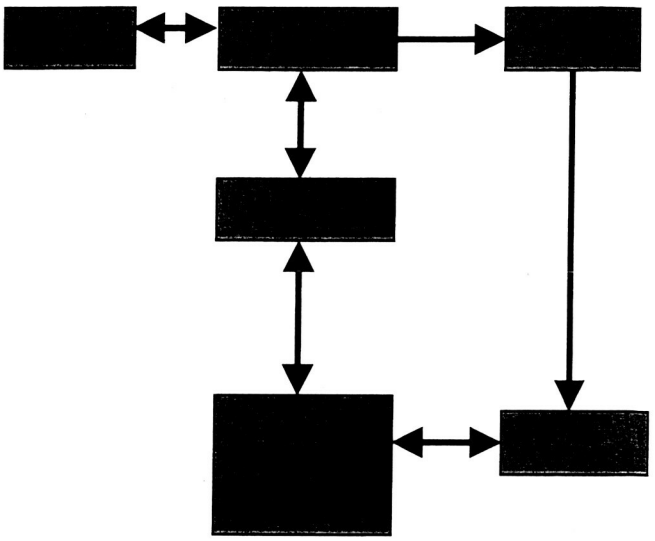

Fig 3. Netmark System Architecture

In NETMARK, we first convert any documents to be stored, originally in any format such as MS Word, PDF, HTML, Excel spreadsheets and others to XML. This conversion is done automatically using a library of parsers for various document formats. These parsers essentially take hints from the formatting information in a document to extract what are called *Contexts* in that document. Contexts can be thought of as sections or sub-sections in a document. For instance, for this paper, the "Abstract", "Introduction", or "Data Storage and Management" sections can be thought of as contexts. The data within a section (or context) is referred to as *Content*, for that context. So the text in the abstract section is the context for the "Abstract" context. This context and content information for a document is captured in XML, for instance for this paper, the XML representation would look as shown in Fig 4.

```
.....
<Context>Abstract</Context>
<Content> This paper describes an ... </Content>
<Context>The NETMARK Data .... </Context>
<Content> We now describe NETMARK .. </Content>
<Context>Data Storage and Management</Context>
<Content>NETMARK is designed to .... </Content>
```

Fig 4. Document in XML

We will discuss query processing with context and content shortly after, and continue with describing document storage. Each document is converted to XML with context and content information as illustrated above and then stored in the NETMARK XML Store. In NETMARK we store the XML documents as relational tables in an underling ORDBMS. Approaches such as [10] define different relations for different XML element types. The NETMARK storage scheme however uses the *same* relational tables to represent and store *any* XML document type. The NETMARK 'SGML parser' (Fig 3.) decomposes the XML (or even HTML) documents into its constituent nodes and dynamically inserts them into two primary database tables—namely, XML and DOC—within a NETMARK generated schema. The descriptions of the XML and DOC tables along with their respective relationships are listed in Fig 5. The SGML parser is governed by five different node data types, which are specified in the HTML or XML configuration files passed by the daemon. The five NETMARK node data types and their corresponding node type identifier as designated in the NODETYPE column of the XML table are as follows: (1) ELEMENT, (2) TEXT, (3) CONTEXT, (4) INTENSE, and (5) SIMULATION [2]. These tables are stored in an underling Oracle ORDBMS.

Object-relational mapping from XML to relational database schema models the data within the XML documents as a tree of objects that are specific to the data in the document [14]. In this model, element type with attributes, content, or complex element types are generally modeled as classes. Element types with parsed character data (PCDATA) and attributes are modeled as scalar types. This model is then mapped to the relational database using traditional object-relational mapping techniques or via SQL3 object views. Therefore, classes are mapped to tables, scalar types are mapped to columns, and object-valued properties are mapped to key pairs (both primary and foreign). This mapping model is limited since the object tree structure is different for each set of XML documents. On the other hand, the NETMARK SGML parser models the document itself (similar to the DOM), and its object tree structure is the *same* for all XML documents. Thus, NETMARK is designed to be *independent* of any particular XML document schemas and is termed to be *"schema-less"*.

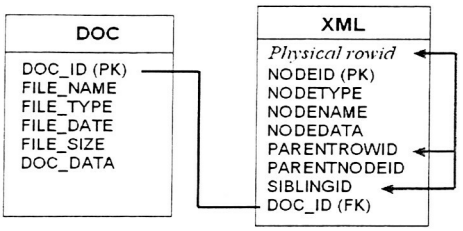

Fig 5. NETMARK Generated Schema

[2] We skip the details on what the different node types are

Note that we have now provided a means to generically store any XML or HTML document without requiring a new schema for a new document (type). We have also captured the context and content information in each document. We must also mention that we have exploited the feature of physical row-ids in Oracle for very fast traversal between nodes that are related.

2.1.2 *NETMARK System*

We outlined the NETMARK system architecture and process flow in Fig 3. above. Users insert new documents (in any format such as Word, PDF, HTML, XML or others) into NETMARK by simply dragging the documents into a (NETMARK) desktop folder. The 'NETMARK DAEMON' periodically picks up these documents passes them onto the 'SGML Parser', which converts the documents into XML. The XML documents are then stored in the 'NETMARK XML Store' in a schema-less manner, as described above. Communication between the user folders and the NETMARK server is done using Web DAV [12] which is a set of extensions to the HTTP protocol which allows users to collaboratively edit and manage files on remote web servers.

Clients and applications can access and query data through the 'NETMARK Extensible APIs' using a variety of protocols based on J2EE, RMI, and ODBC. Users can access NETMARK documents by simple HTTP requests, in fact HTTP provides an extremely simple yet powerful mechanism for users and clients to access NETMARK

2.1.3 *Querying Data in NETMARK*

We now look at data querying capabilities in NETMARK, centered around the notions of context and content. A key capability is that of *context search.* A context search query, such as `Context=Introduction` [3] will return the content portion in the 'Introduction' sections (the text in the Introduction section) in *all* the documents in a document collection. NETMARK also provides for result composition and formatting where we can use XSLT [15] to specify the format of the query results before presenting to the user. For instance we could specify a context search for "Technology Gap" and specify that the integrated results be presented in a new document. This is illustrated in Fig 6. Users can also specifying *content searches* which are essentially keyword searches that return all documents containing the specified search terms. For instance, a content query such as `Content=Shuttle` will return all documents that contain the term 'Shuttle' anywhere in the document. One can also combine context and content searches, for instance a query such as `Context=Technology Gap& Content=Shrinking` returns the "Technology Gap" contexts (sections) of all documents where the term 'Shrinking' occurs *within the Technology Gap context (section).*

[3] This is not the precise query syntax and we do not think it essential to use the formal and precise Netmark query syntax here

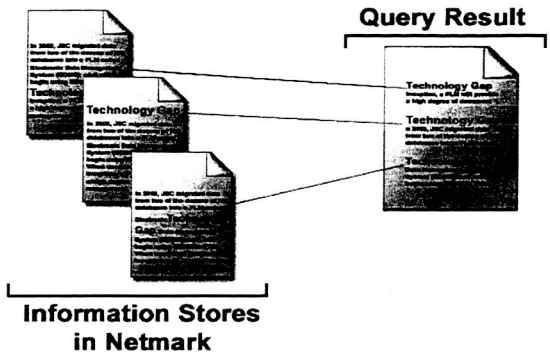

Fig 6. Context Search

The Netmark query language is a language called XDB Query [7]. XBD Query allows for posing the context and content kinds of queries over XML documents, as illustrated above. We will not go into the query syntax details here but the key features are that context and content search specifications are appended to a URL that is sent to NETMARK. In thus URL we may also specify an XSLT stylesheet which specifies how the results are to be formatted and composed into a new document. Fig 7. provides an illustration of using XDB Query to query the data in NETMARK and then using XSLT to format the results. XSLT transformation is done using the Xalan XSLT processor [13].

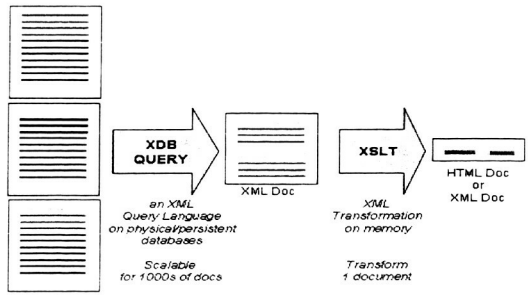

Fig 7. XDB Query search and transformation process

2.1.4 *Processing Queries Internally*

Note that any context query essentially maps to the (implicit) schema for a document or set of documents. The keyword-based context and content search is performed by first querying the text index for the search key. Each node returned from the index search is then processed based on its designated unique ROWID. The processing of the node involves traversing up the tree structure via its parent or sibling node until the first context is

found. The context is identified via its corresponding NODETYPE. The context refers to here as a heading for a subsection within a HTML or XML document, similar to the <H1> and <H2> header tags commonly found within HTML pages. Thus, the context and content search returns a subsection of the document where the keyword being searched for occurs. Once a particular CONTEXT is found, traversing back down the tree structure via the sibling node retrieves the corresponding content text. The search result is then rendered and displayed appropriately.

2.1.5 Accessing Multiple Data Sources

In the above sections we have elaborated on query processing over data in the NETMARL XML Store. NETMARK can also provide integrated query access to multiple information sources that may be distributed at other locations. This is done through a simple declarative process where an administrator creates a 'Databank' for an application. The databank specifies what sources are to be queried when a user fires a query to that application (databank). A source that is queried need not necessarily have XML or even Context+Content searching capabilities. However NETMARK 'augments' the query capability in that it uses whatever query and search capabilities are available at the source and then does further processing required. For instance a source we integrated in one of the NASA applications is the NASA Lessons Learned Information Server[4]. A look at the search interface shows that this source allows only "Content search" kinds of queries. For a query such as `Context=Title&Content=Engine`, NETMARK will pass on to the original source whatever portions of the query it can process (in this example the original source can at least process the content portion of the query which is retrieving documents that contain the word 'Engine'). Further processing is then done in NETMARK where NETMARK then extracts the 'Title' sections from only those documents that contain the word 'Engine' in the 'Title' section, from amongst the initial results returned by the original server. All this is of course abstracted from the end user. For each data source that is accessed, an administrator will have to look at the query capabilities of that source and engineer what query processing can be used from the source and what must further be augmented by Netmark. Also an arbitrary number of sources may be specified in any Databank and any query to that Databank (application) will ultimately go to all the sources specified.

Having outlined the key features and functionality of NETMARK we summarize the distinguishing characteristics of this system and approach:

- We are able to provide sophisticated query facilities, such as Context+Content search or even full-fledged XML querying, over any information repository (that may otherwise have limited or no query capabilities) with NETMARK.
- We can access multiple distributed information sources simultaneously.
- Integration can be specified (and executed) at the *client* side by specifying databanks. Thus integration can be done on-the-fly.
- Middleware requirements are reduced to needing just a thin router capability across the various information sources.
- The approach is highly scalable and flexible in that we can take arbitrary numbers of sources and compose applications that access one or more sources amongst these as and when required.

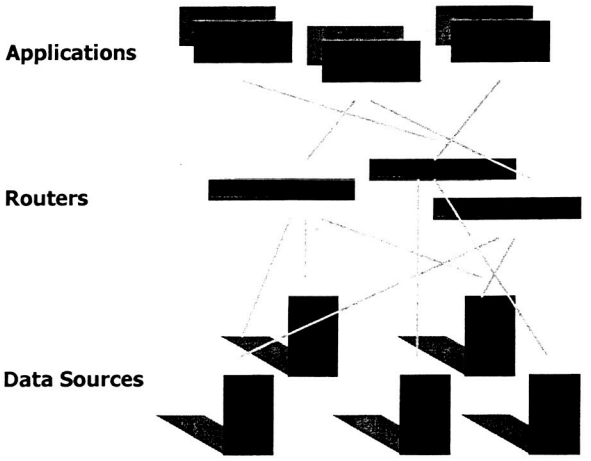

Fig 8. Highly scalable and flexible integration

3 NASA APPLICATION EXAMPLES

NETMARK has proven to be a highly flexible, nimble, and easy to assemble application framework for several integration applications that we have built using this framework at NASA. Table 1 contains a list of these applications along with the time that was taken to assemble them with NETMARK.

Table 1. NASA integration applications

Application Assembly Time	NASA Application
1 hour	Proposal Financial Management Risk Assessment
1 day	Integrated Budget Performance Document
1 week	Anomaly Tracking

We cannot describe all the above mentioned applications here, but all of them are centered around having to extract and integrate data from several heterogeneous and distributed documents to form either an integrated information system or an integrated document or report. For instance, the Proposal Financial

[4] http://llis.nasa.gov/

Management application is an information system for tracking proposal financial information for outgoing (NASA) proposals in response to a call for proposals. This allows querying of aggregated and statistical information about the proposals such as proposal numbers by NASA division type, dollar amounts requested etc. The application takes as input all the proposals (typically in formats such as Word or PDF) that have been submitted in response to a particular call. The Integrated Budget Performance Document (IBPD) is an integrated budget document which unifies previously disconnected budget documents. While manual assembly of the IBPD can take several weeks, NETMARK was used to extract and integrate information from thousands of NASA task plans containing the required budget information and compose an integrated IBPD document. Finally, Anomaly Tracking is an application that allows integrated querying of two NASA (web accessible) data sources that are essentially anomaly tracking databases. The application facilitates more sophisticated querying than provided by either original source and also facilitates simultaneous querying of both sources.

Clearly NETMARK has proven to be a scalable, fast, and flexible integration framework for all of the above NASA integration applications.

4 RELATED XML INTEGRATION APPROACHES

At this point we expect that a reader familiar with data integration systems is curious about a more accurate description of the 'integration glue' that NETMARK provides across various sources being integrated. The approach in other prominent XML mediation systems such as MIX [8] and Tukwila [4] (and industrial systems such as Enosys [9] and Nimble [1] based on these systems) follows from the Global-as-View (GAV) approach in previously developed mediation systems. Each information source is viewed as exporting an XML view (called a source view) of the data it contains. An integrated (global) view of the data is formed by defining an integrated view of the data over the individual data source views. This integrated view definition is done using XML query languages such as XQuery/Xpath [14]. In NETMARK, from a multiple source integration perspective, the focus has been on providing the capability to query multiple sources simultaneously. So for instance a Context query for "Budget" will pull out the 'Budget' sections from all documents in all sources for an application. If the Budget section happens to be referred to as 'Cost Details' in another source then, strictly speaking, in NETMARK we have to specify two Context queries (one for 'Budget' and one for 'Cost Details'). We do not have the luxury of defining a virtual "Budget" view and specifying a mapping that says that 'Cost Details' maps directly to 'Budget', as can be done in MIX and Tukwila systems. Indeed, virtual views can be more complicated. For instance we may want a virtual view called "Top Employees of NASA", which is a view across three information sources at three different NASA centers. Top Employees could be defined as say *employees* at *NASA Ames* with a *performance rating of excellent*, *personnel* at *NASA Johnson* with a *performance score of 2 or better*, and *employees* of *NASA Kennedy* with a *rating of very good or better*. Mediation frameworks such as [8] provide for defining such virtual views and then simply querying the Top Employees (virtual) view. In NETMARK we will end up asking three different queries (corresponding to the different NASA centers) which will go to the different information sources. Note however that the approach in [8] and [1] absolutely requires us to formally define schemas (source views) for the three information sources, define a virtual "Top Employees" view and specify the relationships between the virtual and source views. The NETMARK approach forces no such requirements. NETMARK will look at the source data (typically employee performance documents or spreadsheets) and automatically structure documents from each source, which users can then query. Our experience (and claim) is that the flexibility of not having to specify schemas and relationships between schemas, greatly outweighs any extra work that may need to be done in cases where having a virtual view would be useful.

Also, particular attention has been paid to automated metadata extraction from the data. The bulk of enterprise data resides in documents (in formats such as word, PDF or html), in spreadsheets and presentations. We have developed parsers for a wide variety of document formats (such as Word, PDF, HTML, PowerPoint and others) that automatically structure and "upmark" a document into XML based on the formatting information in the document. Our experience shows that such automatic parses are extremely successful in parsing and structuring most enterprise documents quite accurately.

5 CONCLUSIONS

We have presented an integration framework that is cost effective and economically scalable. This has been achieved by ensuring that formal schema imposition on any data is there only to the extent that it needs to be (if at all). All integration functionality is pushed to the client. No mandatory heavy-weight integration middleware is required, rather the desired integration capabilities can be specified on the application side and on-the fly. The integration framework has been very successfully used to develop several NASA enterprise applications in a very cost-effective manner and within short time-frames.

6 REFERENCES

[1] Draper, D., Halevy, A.Y. and Weld:, D.S., The Nimble XML Data Integration System. *ICDE*, 2001, pp. 155-160.

[2] H.V.Jagadish, Khalifa, S., Chapman, A., Lakshmanan, L., Nierman, A., Paparizos, S., Patel, J., Srivastava, D., Wiwatwattanna, N., Wu, Y. and Yu, C., TIMBER: A Native XML Database, *VLDB Journal*, 11 (2002) 274-291.

[3] Halevy, A., Data Integration: A Status Report (Invited Talk). *German Database Conference (BTW)*, 2003.

[4] Ives, Z., Halevy, A. and Weld, D., An XML query engine for network-bound data, *VLDB Journal*, 11 (2002) 380-402.

[5] Knox, R., Grey, M., B.Burton, W.Andrews, G.Phifer, T, A., T.Eid, K.Harris, T.Bell, J.Lundy, W.Arevolo, D.M.Smith, D.Logan and L.Latham, Research Note: Predicts 2005: Support Improves for Knowledge Workers. Gartner, 2004.

[6] Maluf, D. and Tran, P., NETMARK: A Schema-Less Extension for Relational Databases for Managing

Semi-structured Data Dynamically. *ISMIS*, 2003, pp. 231-241.

[7] Maluf, D., Tran, P. and La, T., "An Extensible 'Schema-less' Database Framework for Managing High-Throughput Semi-Structured Documents. *IASTED, Applied Informatics*, 2003.

[8] Mukhopadhay, P. and Papakonstantinou, Y., Mixing Querying and Navigation in MIX. *ICDE*, 2002.

[9] Papakonstantinou, Y. and Vassalos, V., Architecture and Implementation of an XQuery-based Information Integration Platform, *Data Engineering Bulletin* (2002).

[10] Shanmugasundaram, J., Shekita, E., Kiernan, J., Krishnamurthy, R., Viglas, S., Naughton, J. and Tatarinov, I., A General Technique for Querying XML Documents using a Relational Database System, *SIGMOD Record*, 30 (2001) 20-26.

[11] Stonebraker, M., Too Much Middleware, *ACM SIGMOD Record*, 31 (2002) 97-106.

[12] Whitehead, J. and Goland, Y., WebDAV: A network protocol for remote collaborative authoring on the Web. *CSCW*, 1999.

[13] Xalan, http://xml.apache.org/xalan-j/.

[14] XQuery, http://www.w3.org/TR/xquery/.

[15] XSLT, http://www.w3.org/TR/xslt.