NETMARK: A Schema-less Extension for Relational Databases for Managing Semi-Structured Data Dynamically

David A. Maluf
NASA Ames Research Center, Mail Stop 269-4
Moffett Field, California, USA
maluf@ptolemy.arc.nasa.gov

and

Peter B. Tran
QSS Group, Inc.
NASA Ames Research Center, Mail Stop 269-4
Moffett Field, California, USA
pbtran@mail.arc.nasa.gov

Abstract. Object-Relational database management system is an integrated hybrid cooperative approach to combine the best practices of both the relational model utilizing SQL queries and the object-oriented, semantic paradigm for supporting complex data creation. In this paper, a highly scalable, information on demand database framework, called NETMARK, is introduced. NETMARK takes advantages of the Oracle 8i object-relational database using physical addresses data types for very efficient keyword search of records spanning across both context and content. NETMARK was originally developed in early 2000 as a research and development prototype to solve the vast amounts of unstructured and semi-structured documents existing within NASA enterprises. Today, NETMARK is a flexible, high-throughput open database framework for managing, storing, and searching unstructured or semi-structured arbitrary hierarchal models, such as XML and HTML.

1 Introduction

During the early years of database technology, there were two opposing research and development directions, namely the relational model originally formalized by Codd [1] in 1970 and the object-oriented, semantic database model [2][3]. The traditional relational model revolutionized the field by separating logical data representation from physical implementation. The relational model has been developed into a mature and proven database technology holding a majority stake of the commercial database market along with the official standardization of the Structured Query Language (SQL)[1] by ISO and ANSI committees for a user-friendly data definition language (DDL) and data manipulation language (DML).

The semantic model leveraged off from the object-oriented paradigm of programming languages, such as the availability of convenient data abstraction mechanisms, and the realization of the *impedance mismatch* [4] dilemma faced between the popular object-oriented programming languages and the underlining relational database management systems (RDBMS). Impedance mismatch here refers to the problem faced by both

[1] The Structured Query Language (SQL) is the relational standard defined by ANSI (the American National Standard Institute) in 1986 as SQL1 or SQL-86 and revised and enhanced in 1992 as SQL2 or SQL-92.

database programmers and application developers, in which the way the developers structure data is not the same as the way the database structures it. Therefore, the developers are required to write large and complex amounts of object-to-relational mapping code to convert data, which is being inserted into a tabular format the database can understand. Likewise, the developers must convert the relational information returned from the database into the object format developers require for their programs. Today, in order to solve the impedance mismatch problem and take advantage of these two popular database models, commercial enterprise database management systems (DBMS), such as Oracle, IBM, Microsoft, and Sybase, have an integrated hybrid cooperative approach of an *object-relational model* [5].

In order to take advantage of the object-relational (OR) model defined within an *object-relational database management system* (ORDBMS) [5][6], a standard for common data representation and exchange is needed. Today, the emerging standard is the *eXtensible Markup Language* (XML) [7][8] known as the next generation of HTML for placing structure within documents. Within any large organizations and enterprises, there are vast amounts of heterogeneous documents existing in HTML web pages, word processing, presentation, and spreadsheet formats. The traditional document management system does not provide an easy and efficient mechanism to store, manage, and query the relevant information from these heterogeneous and complex data types.

To solve the vast quantities of heterogeneous and complex documents existing within NASA enterprises, NASA at Ames Research Center initially designed and developed an innovative schema-less, object-relational database integration technique and framework referred to hereby as *NETMARK*. Developed in early 2000 as a rapid, proof-of-concept prototype, NETMARK, today, is a highly scalable, open enterprise database framework (architecture) for dynamically transforming and generating arbitrary schema representations from unstructured and/or semi-structured data sources. NETMARK provides automatic data management, storage, retrieval, and *discovery* [17] in transforming large quantities of highly complex and constantly changing heterogeneous data formats into a well-structured, common standard.

This paper describes the NETMARK schema-less database integration technique and architecture for managing, storing, and searching unstructured and/or semi-structured documents from standardized and interchangeable formats, such as XML in relational database systems. The unique features of NETMARK take advantages from the object-relational model and the XML standard described above, along with an open, extensible database framework in order to dynamically generate arbitrary schema stored within relational databases, object-relational database management system.

2 BACKGROUND

2.1 Object-Relational DBMS

The object-relational model takes the best practices of both relational and object-oriented, semantic views to decouple the complexity of handling massively rich data representations and their complex interrelationships. ORDBMS employs a data model that attempts to incorporate object-oriented features into traditional relational database systems. All database information is still stored within relations (tables), but some of the tabular attributes may have richer data structures. It was developed to solve some of the inadequacies associated with storing large and complex multimedia objects, such as audio, video, and image files, within traditional RDBMS. As an intermediate hybrid cooperative model, the ORDBMS combined the flexibility, scalability, and security of using existing relational systems along with extensible object-oriented features, such as data abstraction, encapsulation, inheritance, and polymorphism.

In order to understand the benefits of ORDBMS, a comparison of the other models need to be taken into consideration. The 3x3 database application classification matrix [6] shown in Table 1 displays the four categories of general DBMS applications—simple data without queries (file systems), simple data with queries

(RDBMS), complex data without queries (OODBMS), and complex data with queries (ORDBMS). For the upper left-handed corner of the matrix, traditional business data processing, such as storing and managing employee information, with simple normalized attributes, such as numbers (integers or floats) and character strings, usually needs to utilize SQL queries to retrieve relevant information. Thus, RDBMS is well suited for traditional business processing; but this model cannot store complex data, such as word processing documents or geographical information. The lower right-handed corner describes the use of persistent object-oriented languages to store complex data objects and their relationships. The lower right-handed corner represents OODBMS, which either have very little SQL-like queries support or none at all. The upper right-handed corner with the dark gray colored cell is well suited for complex and flexible database applications that need complex data creation, such as large objects to store word processing documents, and SQL queries to retrieve relevant information from within these documents. Therefore, the obvious choice for NETMARK is the upper right-handed corner with the dark gray colored cell as indicated in Table 1.

Query	**RDBMS** (Traditional Business Data Processing)	**ORDBMS** (NETMARK)
No Query	**File Systems** (Simple Text Editors)	**OODBMS** (Persistent OO Languages)
	Simple Data	**Complex Data**

Adapted from M. Stonebraker, "Object-Relational DBMS - The Next Wave", Informix Software (now part of the IBM Corp. family), Menlo Park, CA

Table 1: Database Application Classification Matrix

The main advantages of ORDBMS, are scalability, performance, and widely supported by vendors. ORDBMS have been proven to handle very large and complex applications, such as the NASDAQ stock exchange, which contains hundreds of gigabytes of richly complex data for analyst and traders to query stock data trends. In terms of performance, ORDBMS supports query optimization, which are comparable to RDBMS and out performs most OODBMS. Therefore, there is a very large market and future for ORDBMS. This was another determining factor for using ORDBMS for the NETMARK project. Most ORDBMS supports the SQL3 [9] specifications or its extended form. The two basic characteristics of SQL3 are crudely separated into its "relational features" and its "object-oriented features". The relational features for SQL3 consist of new data types, such as large objects or LOB and its variants. The object-oriented features of SQL3 include structured user-defined types called *abstract data types* (ADT) [10][11] which can be hierarchical defined (inheritance feature), invocation routines called methods, and REF types that provides reference values for unique row objects defined by *object identifier* (OID) [11] which is a focus of this paper.

2.2 Large Objects

ORDBMS was developed to solve some inadequacies associated with storing large and complex data. The storage solution within ORDBMS is the large object data types called LOBs [9][11]. There are several LOB variants, namely binary data (BLOB), single-byte character data set (CLOB), multi-byte character data

(NCLOB), and binary files (BFILE). BLOBs, CLOBs, and NCLOBs are usually termed *internal LOBs* [11], because they are stored internally within the database to provide efficient, random, and piece-wise access to the data. Therefore, the data integrity and concurrency of external BFILEs are usually not guaranteed by the underlining ORDBMS. Each LOB contains both the data value and a pointer to the data called the *LOB locator* [9]. The LOB locator points to the data location that the database creates to hold the LOB data. NETMARK uses LOBs to store large documents, such as word processing, presentation, and spreadsheet files, for later retrieval of the document and its contents for rendering and viewing.

2.3 Structuring Documents with XML

XML is known as the next generation of HTML and a simplified subset of the Standard Generalized Markup Language (SGML)[2]. XML is both a semantic and structured markup language [7]. The basic principle behind XML is simple. A set of meaningful, user-defined tags surrounding the data elements describes a document's structure as well as its meaning without describing how the document should be formatted [12]. This enables XML to be a well-suitable meta-markup language for handling loosely structured or *semi-structured data*, because the standard does not place any restrictions on the tags or the nesting relationships. Loosely structured or semi-structured data here refers to data that may be irregular or incomplete, and its structure is rapidly changing and unpredictable [12]. Good examples of semi-structured data are web pages and constantly changing word processing documents being modified on a weekly or monthly basis.

XML encoding, although more verbose, provides the information in a more convenient and usable format from a data management perspective. In addition, the XML data can be transformed and rendered using simple *eXtensible Stylesheet Language* (XSL) specifications [8]. It can be validated against a set of grammar rules and logical definitions defined within the *Document Type Definitions* (DTDs) or *XML Schema* [14] much the same functionality as a traditional database schema.

2.4 Oracle ROWIDs

ROWID is an Oracle data type that stores either physical or logical addresses (row identifiers) to every row within the Oracle database. Physical ROWIDs store the addresses of ordinary table records (excluding indexed-organized tables), clustered tables, indexes, table partitions and sub-partitions, index partitions and sub-partitions, while logical ROWIDs store the row addresses within indexed-organized tables for building secondary indexes. Each Oracle table has an implicit pseudo-column called ROWID, which can be retrieved by a simple SELECT query on the particular table. Physical ROWIDs provide the fastest access to any record within an Oracle table with a single read block access, while logical ROWIDs provide fast access for highly volatile tables. A ROWID is guaranteed to not change unless the rows it references is deleted from the database.

The physical ROWIDs have two different formats, namely the legacy *restricted* and the new *extended* ROWID formats. The restricted ROWID format is for backward compatibility to legacy Oracle databases, such as Oracle 7 and/or earlier releases. The extended format is for Oracle 8 and later object-relational releases. This paper will only concentrate on extended ROWID format, since NETMARK was developed using Oracle 8i (release 8.1.6). For example, the following displays a subset of the extended ROWIDs from a NETMARK generated schema. It is a generalized 18-character format with 64 possibilities each:

AAAAAA | BBB | CCCCCC | DDD

[2] The Standard Generalized Markup Language (SGML) is the official International Standard (ISO 8879) adopted by the world's largest producers of documents, but is very complex. Both XML and HTML are subsets of SGML.

The extended ROWIDs could be used to show how an Oracle table is organized and structured; but more importantly, extended ROWIDs make very efficient and stable unique keys for information retrievals, which will be addressed in the following subsequent section below (3.3).

3 THE NETMARK APPROACH

Since XML is a document and not a data model per se, the ability to map XML-encoded information into a true data model is needed. The NETMARK approach allows this to occur by employing a customizable data type definition structure defined by the NETMARK SGML parser to model the hierarchical structure of XML data regardless of any particular XML document schema representation. The customizable NETMARK data types simulate the Document Object Model (DOM) Level 1 specifications [15] on parsing and decomposition of element nodes. The SGML parser is more efficient on decomposition than most commercial DOM parsers, since it is much more simpler as defined by node types contained within configuration files. The node data type format is based on a simplified variant of the *Object Exchange Model* (OEM) [13] researched at Stanford University, which is very similar to XML tags. The node data type contains an object identifier (node identifier) and the corresponding data type. Traditional 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 traditional 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.

NETMARK is even flexible to handle more than just XML. It is also a SGML-enabled, open enterprise database framework. The term SGML-enabled means NETMARK supports both HTML and XML sets of tags through a set of customizable configuration files utilized by the NETMARK SGML parser for dynamically generating arbitrary database schema as shown in section (3.2) and in Figure 2. The NETMARK SGML parser decomposes either the HTML or XML document into its constituent nodes and inserted the nodes as individual records within Oracle tables. This dynamic schema representation and generation without requiring to write tedious and cumbersome SQL scripts or having to depend on experienced database administrators (DBAs) saves both time and valuable resources. Thus, this makes the storage model of NETMARK a general-purpose HTML or XML storage system.

3.1 Architecture

The NETMARK architecture comprises of the distributed, *information on demand* model, which refers to the "plug and play" capabilities to meet high-throughput and constantly changing information management environment. Each NETMARK modules are extensible and adaptable to different data sources. NETMARK consists of (1) a set of interfaces to support various communication protocols (such as HTTP, FTP, RMI-IIOP and their secure variants), (2) an information bus to communicate between the client interfaces and the NETMARK core components, (3) the daemon process for automatic processing of inputs, (4) the NETMARK keyword search on both document context and content, (5) a set of extensible application programming interfaces (APIs), (6) and the Oracle backend ORDBMS.

The three core components of NETMARK consist of the high-throughput information bus, the asynchronous daemon process, and the set of customizable and extensible APIs built on Java enterprise technology (J2EE) and Oracle PL/SQL stored procedures and packages [11]. The NETMARK information bus allows virtually three major communication protocols heavily used today—namely HTTP web-based protocol and its secure variant, the File Transfer Protocol (FTP) and its secure variant, and the new Remote Method Invocation (RMI) over Internet Inter-Orb Protocol from the Object Management Group (OMG) Java-CORBA standards—to meet the information on demand model. The NETMARK daemon is a unidirectional asynchronous process to increase performance and scalability compared to traditional synchronous models, such as Remote Procedure Call (RPC) or Java RMI mechanisms. The NETMARK set of extensible Java and PL/SQL APIs are used to enhance database access and data manipulation, such as a robust Singleton database connection pool for managing check-ins and checkouts of pre-allocated connection objects.

3.2 Universal Process Flow

The NETMARK closed-loop universal process flow is shown in Figure 1. The information bus comprises of an Apache HTTP web server integrated with Tomcat Java-based JSP/Servlet container engine. It waits for incoming requests from the various clients, such as an uploaded word processing document from a web browser. The bus performs a series of conversion and transformation routines from one specific format to another using customized scripts. For instance, the NETMARK information bus will automatically convert a semi-structured Microsoft Word document into an inter-lingual HTML or XML format. A copy of the original word document, the converted HTML or XML file, and a series of dynamically generated configuration files will be handed to the NETMARK daemon process.

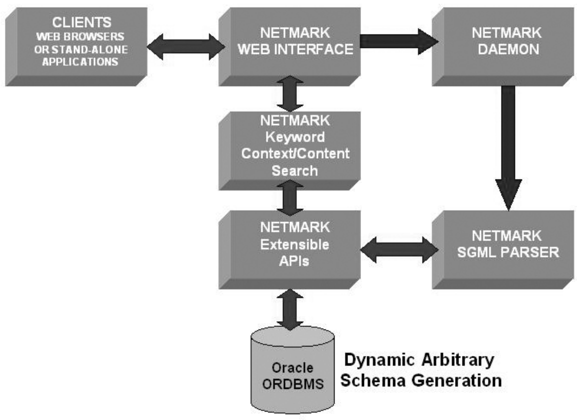
CLIENTS
WEB BROWSERS OR STAND-ALONE APPLICATIONS
NETMARK WEB INTERFACE
NETMARK DAEMON
NETMARK Keyword Context/Content Search
NETMARK Extensible APIs
NETMARK SGML PARSER
Oracle ORDBMS
Dynamic Arbitrary Schema Generation

Figure 1: NETMARK Universal Process Flow

The daemon process checks for configuration files, the original processed files, and notifies the NETMARK SGML parser for decomposition of document nodes and data insertion. The daemon has an automatic logger that outputs both successful and event errors by date and time stamps with periodical archival and cleanup of log files. The daemon accepts three types of configuration files—(1) the request file, (2) the HTML/XML configuration file, and (3) the metadata configuration file. The request file is required by the daemon to proceed to process the correct information, whereas the HTML/XML configuration file and the metadata file are optional. If there is no HTML/XML configuration file provided to the daemon, a default configuration file located on the server is used. If there is no request file, the daemon issues an appropriate error message, logs the message to the log files for future reference, performs cleanup of configuration files, and waits for the next incoming request.

If the daemon can read the request file from the incoming request directory, it locks the file and extracts the name-value pairs from the request file for further processing. After extraction of the relevant attribute values, the request file is unlocked and a child process is spawned to process the incoming files. The child process locks the request file again to prevent the parent process from reprocessing the same request file and calling the SGML parser twice to decompose and insert the same document. The child process then calls the NETMARK SGML parser with the appropriate flag options to decompose the HTML or XML document into its constituent nodes and insert the nodes into the specified database schema. After the parsing and insertion completes, the source, result, and metadata files along with its corresponding configuration files will be cleanup and deleted by the daemon.

The NETMARK SGML parser decomposes the HTML or XML 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 Figure 2. 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.

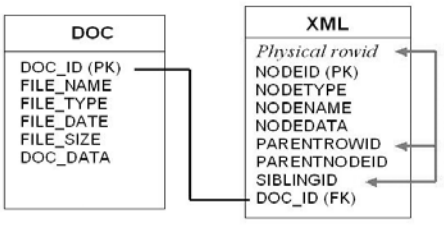

Figure 2: NETMARK Generated Schema

The node type identifier is a single character data type inserted by the SGML parser to the XML table for each decomposed XML or HTML nodes. The node type identifiers will be used in the keyword-based context and content search.

The XML table contains the node tree structure as specified by the rules governed by the HTML or XML configuration files being used by the SGML parser to decompose the original HTML or XML documents into its constituent nodes. The DOC table holds the source document metadata, such as FILE_NAME, FILE_TYPE, FILE_DATE, and FILE_SIZE. Each NETMARK generated schema contains these two primary tables for efficient information retrievals as explained in the subsequent section (3.3). In order to store, manipulate, and later on retrieve unstructured or semi-structured documents, such as word processing files, presentations, flat text files, and spreadsheets, NETMARK utilizes the LOB data types as described in section (2.2) to store a copy of each processed document. In Figure 2 both the XML and the DOC table utilize CLOB and BLOB data types, respectively within the NODEDATA attribute for the XML table and the DOC_DATA column for the DOC table.

3.3 NETMARK Keyword Search

There are two ways that the Oracle database performs queries—either by a costly full table scan (with or without indexes) or by ROWIDs. Since a ROWID gives the exact physical location of the record to be retrieved by a single read block access, this is much more efficient as the database table size increases. As implied in the earlier section (2.4), ROWIDs can be utilized to very efficiently retrieve records by using them as unique keys. The NETMARK keyword search takes advantage of the unique extended ROWIDs for optimizing record retrievals based on both context and content. The keyword-based search here refers to finding all objects (elements or attributes) whose tag, name, or value contains the specified search string.

The NETMARK keyword search is built on top of Oracle 8i *interMedia Text* index [11][17] for retrieving the search key, and it is based on the Object Exchange Model [13] researched at Stanford University as mentioned earlier in section (3). Oracle interMedia is also known as Oracle Text in later releases of Oracle 9i and formerly known as the ConText [16] data cartridge. Oracle interMedia text index creates a series of index tables within the NETMARK generated schema to support the keyword text queries. The interMedia text index is created on the NODEDATA column of the XML table as shown in Figure 2. The NODEDATA column is a CLOB data type (character data). As described in Figure 2, the NETMARK XML table is consisted of eight attributes (columns) plus one physical ROWID pseudo-column. Each row in the XML table describes a complete XML or HTML node. The main attributes being utilizing by the search are DOC_ID, NODENAME, NODETYPE, NODEDATA, PARENTROWID, and SIBLINGID from the XML table. The DOC_ID column is used to refer back to the original document file. As the name implies, NODENAME contains the name of the node; whereas NODETYPE, as described earlier in section (3.2), identifies the type of node it is and informs NETMARK how to process this particular node. Reiterating the five specialized node data types: (1) ELEMENT, (2) TEXT, (3) CONTEXT, (4) INTENSE, and (5) SIMULATION. TEXT is a node whose data are free text or blocks of text describing a specific content. An ELEMENT, similar to a HTML or XML element, can contain multiple TEXT nodes and/or other nested ELEMENT nodes. Within NETMARK search, CONTEXT is a parent ELEMENT whose children elements contain data describing the contents of the following sibling nodes. INTENSE is another CONTEXT, which itself contains meaningful data. SIMULATION is a node data type reserved for special purposes and future implementation. NODEDATA is an Oracle CLOB data type used to store TEXT data. PARENTROWID and SIBLINGID are used as references for identifying the parent node and sibling node, respectively, and are of data type ROWID.

The NETMARK keyword-based context and content search is performed by first querying 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.

4 CONCLUSION

NETMARK provides an extensible, schema-less, information on demand framework for managing, storing, and retrieving unstructured and/or semi-structured data. NETMARK was initially designed and developed as a rapid, proof-of-concept prototype using a proven and mature Oracle backend object-relational database to solve the vast amounts of heterogeneous documents existing within NASA enterprises. NETMARK is currently a scalable, high-throughput open database framework for transforming unstructured or semi-structured documents into well-structured and standardized XML and/or HTML formats.

5 ACKNOWLEDGEMENT

The authors of this paper would like to acknowledge NASA Information Technology-Based program and NASA Computing, Information, and Communication Technologies Program.

References

1. Codd, E. F.: A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, 13(6), (1970) 377-387
2. Hull, R. and King, R.: "Semantic Database Modeling: Survey, Applications, and Research Issues", ACM Computing Surveys, 19(3), (1987) 201-260
3. Cardenas, A. F., and McLeod, D. (eds), Research Foundations in Object-Oriented and Semantic Database Systems, New York: Prentice-Hall, (1990) 32-35
4. Chen, J. and Huang, Q.: Eliminating the Impedance Mismatch Between Relational Systems and Object-Oriented Programming Languages, Australia: Monash University (1995)
5. Devarakonda, R. S.: Object-Relational Database Systems – The Road Ahead, ACM Crossroads Student Magazine, (February 2001)
6. Stonebraker M.: Object-Relational DBMS: The Next Wave, Informix Software (now part of the IBM Corp. family), Menlo Park, CA
7. Harold, E. R.: XML: Extensible Markup Language, New York: IDG Books Worldwide, (1998) 23-55
8. Extensible Markup Language (XML) World Wide Web Consortium (W3C) Recommendation, (October 2000)
9. Eisenberg, A., and Melton, J.: SQL:1999, formerly known as SQL3, (1999)
10. ISO/IEC 9075:1999, "Information Technology—Database Language—SQL—Part 1: Framework (SQL/Framework)", (1999)
11. Loney, K. and Koch, G.: Oracle 8i: The Complete Reference, 10th edition, Berkeley, CA: Oracle Press Osborne/McGraw-Hill, (2000) 69-85; 574-580; 616-644; 646-663
12. Widom, J.: Data Management for XML Research Directions, Stanford University, (June 1999) http://www-db.stanford.edu/lore/pubs/index.html
13. Lore XML DBMS project, Stanford University (1998)
http://www-db.stanford.edu/lore/research/
14. Bourret, R.: Mapping DTD to Databases, New York: O'Reilly & Associates, (2000)
15. Wood, L. et al.: "Document Object Model (DOM) Level 1 Specification", W3C Recommendation, (October 1998)
16. Oracle8 ConText Cartridge Application Developer's Guide (Release 2.4), Princeton University Oracle Reference (1998)
17. Maluf, D. A. and Tran, P. B.: "Articulation Management for Intelligent Integration of Information", IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews, 31(4) (2001) 485-496