ABSTRACT

TOWARDS HYPERMEDIA SUPPORT IN DATABASE SYSTEMS

by
Anirban Bhaumik

The general goal of our research is to automatically generate links and other hypermedia related services to analytical applications. Using a dynamic hypermedia engine (DHE), the following features have been automated for database systems. Based on the database's relational (physical) schema and its original (non-normalized) entity-relationship specification links are generated, database application developers may also specify the relationship between different classes of database elements. These elements can be controlled by the same or different database application, or even by another software system. A DHE prototype has been developed and illustrates the above for a relational database management system.

The DHE is the only approach to automated linking that specializes in adding a hyperlinks automatically to analytical applications that generate their displays dynamically (e.g., as the result of a user query). The DHE's linking is based on the structure of the application, not keyword search or lexical analysis based on the display values within its screens and documents. The DHE aims to provide hypermedia functionality without altering applications by building "application wrappers" as an intermediary between the applications and the engine.

TOWARDS HYPERMEDIA SUPPORT IN DATABASE SYSTEMS

by
Anirban Bhaumik

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

January 2001

APPROVAL PAGE

TOWARDS HYPERMEDIA SUPPORT IN DATABASE SYSTEMS

Anirban Bhaumik

Dr. Michael Bieber — Date
Associate Professor of Computer and Information Science, NJIT

Dr. Vincent Oria — Date
Assistant Professor of Computer and Information Science, NJIT

Dr. Byoung-Kee Yi — Date
Assistant Professor of Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author:	Anirban Bhaumik
Degree:	Master of Science in Computer Science
Date:	January 2001

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ 2001

- Master of Science in Chemical Engineering
  New Jersey Institute of Technology, Newark, NJ, 1998

- Bachelor of Engineering in Chemical Engineering
  Regional Engineering College, Durgapur, India, 1996

Major: Computer Science

Presentations and Publications:

"Computer Aided Cognitive Tools for Teaching and Implementing Clean Manufacturing" at the National Science Foundation, Technology Reinvestment Project, Engineering Education Innovators' Conference, April 8, 1997, Washington DC.

To my Parents, and Suchi, thanks for being there.

ACKNOWLEDGMENT

The author would like to thank his advisor Dr. Michael Bieber for his guidance. Special thanks are also due to my colleagues in the Collaborative Hypermedia Laboratory at the New Jersey Institute of Technology: Mr. Roberto Galnares, Mr. Firas Aljallad, Ms. Deepti Dixit and Ms. Aparna Reddy.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Motivation

Database queries typically return results in a plain text format. Some applications on the World Wide Web generate link anchors for database elements, but these anchors normally hold a single link to the most obvious destination for the dominant type of user.

An element within a database application maybe considered a potential starting point for information exploration. Each element may have multiple links, each representing a different relationship (schema-based or otherwise). The ability to explore a piece of information in more detail would allow users to better understand that item, as well as analyze and view the various relationships that define these elements. Users may wish to explore around data values and symbols they see, labels on graphs or user input forms, options in pop-up lists, or even on the menu commands they can invoke.

To complicate the developer's job, users often have different mental models of an application and its underlying domain than the developer. Even when developers work closely with users, the end result might not be intuitive for all users or serve each user's individual tasks equally well. Many people visit a given application's screen aside from the most dominant type of user(s) for which it was developed. These include other users of the application, customer service representatives, company analysts, managers, trainees, people inside the company designing new databases or applications based on the current one, external analysts, and stockholders, among others. Each may be interested in different aspects of application elements, according to their current task-at-hand. Customization is one solution, but even so users often might wish to explore several different relationships from a given anchor, and therefore should have several links available.

The purpose of this research is to explore all aspects of hypermedia support for database applications, and is based on the experience of designing and developing the prototype Dynamic Hypermedia Engine (DHE). The DHE automatically generates anchors, sets of links and metadata within database applications, as well as supporting users with other types of hypermedia structuring, navigation and annotation functionality, including guided tours and annotation.

This work makes many contributions to both the database and hypermedia fields. Many database applications do not take as much advantage of hypermedia as they could. This chapter puts forth a series of opportunities for integrating hypermedia and database systems. As we shall describe in the next chapter, the DHE is the only tool that provides automated linking and hypermedia services based on the application structure (as opposed to search or lexical analysis), without altering applications. Thus it is uniquely suited to support databases and other analytical applications on the Web that generate the contents of their displays dynamically in response to user queries.

This work proceeds as follows. A Literature Review of other Hypermedia Engines and hypermedia support in databases is presented followed by an introduction the Dynamic Hypermedia Engine (DHE). The following chapters show how the DHE provides support to relational DBMS, database applications and enterprise-wide Data Warehouses. This research concludes by providing a direction for future Research in this field.

1.2 Literature Review

1.2.1 Hypermedia Engines

Several approaches exist for integrating hypermedia functionality into primarily non-hypermedia information systems. These include employing hypermedia data models (Campbell and Goodman 1988; Halasz and Schwartz 1994), hypermedia toolkits (Anderson 1996), link services (Pearl 1989; Davis et al. 1992; Anderson 1997), hyperbases (Leggett and Schnase 1994), hypermedia development environments (Nanard and Nanard 1995a; Marshall and Shipman 1995; Akscyn et al. 1988), open hypermedia systems (Whitehead 1997; Wiil 1997; Grønbæk and Trigg 1999), and independently executing hypermedia engines, such as the DHE.

Hypermedia engines execute independently of an application with minimal modifications to it, and provide the application's users with hypermedia support. Few approaches provide transparent hypermedia integration as our engine does. Notable projects include Microcosm's Universal Viewer, Freckles and the OO-Navigator and SFX..

Microcosm's Universal Viewer (Davis et al. 1994) and Freckles (Kacmar 1993, 1995) seamlessly supports an application's other functionality but provides only manual linking. OO-Navigator comes the closest to our approach, providing a seamless hypermedia support for computational systems that execute within a single Smalltalk environment (Garrido and Rossi 1996; Rossi et al. 1996). This approach meets our goal of supplementing Smalltalk applications with hypermedia support without altering them. Our approach applies to both object-oriented and non object-oriented applications.

SFX's engine is very similar to DHE, but it only serves one specific environment. SFX dynamically generates anchors within the reference section of academic papers being displayed on the Web. Selecting these will lead to the original work within bibliographic databases (Van der Stemple, 1999a,b,c). DHE, in contrast, provides a generalized approach for linking and additional hypermedia functionality for most analytical applications.

1.2.2 Research in Hypermedia and Databases.

Several techniques have been proposed recently for the integration of hypertext and databases. Some of them address the issue of building hypertext structures over existing databases to provide more direct navigation through hyperlinks.

Hara and Botafogo (Hara et al 1994) use an SQL-like data definition language to map single relations or relational views to node types. A node type is similar in nature to an entity type, i.e., it models a real world object or concept in the hypertext conceptual schema defined over the database contents. Its specification includes the correspondences between relation attributes and node fields, as well as presentation information. At run time, a node type produces two kinds of nodes: a composite one for the whole relation, and a number of nodes corresponding to the tuples of the relation. The same language is used to define link types among node types. A WHERE-clause is used to constraint the creation of links during navigation.

In a similar approach, Falquet et al. (Falquet et al 1995, Falquet et al 1998) offer a declarative language to produce databases views composed of node and link schemas, accessed through the WWW. Each node schema is based on one object class or a set of inter-related object classes. The content of the node is composed of a subset of the attributes of the class(es).

Foreign keys to other classes constitute link types to the corresponding nodes. Two kinds of links are supported: *Reference* links are indented to offer navigation structure within the nodes, while *inclusion* links are indented to create nested structures (part-of relationships). In addition, the specification of the relational view includes presentation information. The above definitions form the input to a cgi-script that produces HTML pages for the end-user. The DHE would enhance the existing views specified through the database application.

The above approaches leave the original client application intact, introducing a new interface that provides hypermedia-based interaction with the database. On the other hand the DHE overlays linking facilities within the original user interface application by means of user interface wrappers.

Domenicus (Constantopoulos et al. 1996) is a hypermedia engine developed over a repository management system, called Semantic Index System (SIS). Domenicus offers hypermedia functionality (such as alphabetic lists, subject catalogs, guided tours, query cards, hyperlinks, image annotations, bookmarks and history), based on predefined queries over the information objects and their structure, managed by SIS. *Presentation Card Specifications* are executed at run-time to present objects or classes of objects stored in SIS, while *hyperlink classes* dynamically produce links during navigation. Different presentation models can coexist for the same repository instance, to fulfil the searching, browsing and updating requirements of different user groups. The DHE provides many of these features in a generic way over any application, allowing tours and indexes to contain elements from several systems. Also, in the DHE queries are only predefined to the extent that mapping rules hold skeleton queries for particular classes of database elements.

Other approaches suggest embedding database queries into HTML pages. For example, a mechanism offering cross-language variable substitution between HTML and SQL is the core of the *DB2 WWW Connection* system (Nguyen et al. 1996), which enables quick and easy construction of applications accessing relational databases from the Web. The developer creates *macros* that consist of HTML and SQL commands, written in distinct sections. The sections are tied together via variable substitution. Macros are stored at the Web server and are processed by cgi-scripts in order to get user input or produce output reports. The DHE does not store single database queries in the pages displayed on the Web. Instead we generate a list of several possible links for any element from specifications in the mapping rules.

Instead of providing hypertext functionality for a specific database, Geldof (Geldof 1996) uses an abstract page definition language to construct templates embodying presentation guidelines for terms of an ontology; a conceptualisation containing objects, concepts and relationships among them, that are presumed to exist in some area of interest. Actual information sources are linked separately with the terms of the ontology, using a definition language as well. CGI-scripts in Perl are computed to dynamically generate the HTML pages returned to the user for browsing. While this approach adds a certain level of automated linking to aid navigation, the DHE provides a generally larger set of links based solely on the database structure and entity-relationship schemas, as well as metadata. The DHE, however, does not provide customized templates for domain-specific navigational contexts. The DHE might integrate well with Geldof's approach to provide an additional level of functionality.

Moreover, many products have been released recently that aim to interface RDBMS and Web servers (Frank 1995). The solutions employed in these products require huge programming effort in SQL or a scripting language. The ease of integration with the DHE depends on how easy it is to parse application displays to identify the elements of interest, and to specify the commands to return to the application in the mapping rules. If the application has an API or marks the elements in the displays (as should become the custom as XML becomes more prevalent), building the application wrapper should be relatively easy.

The approaches presented above presuppose the hypertext designer's insight into the intrinsic semantics of the relational structure. A different approach was proposed by Papadopoulos et al (1996). Instead of relying on the relational schema of the database, a more semantically enriched Extended Entity-Relationship (EER) schema is semi-automatically produced, by incorporating a reverse engineering methodology. Hypertext views, consisting of node and link types, can be defined over the EER schema, while the SQL queries to instantiate them at run-time where automatically created, based on mapping information gathered during the reverse engineering process. Currently the DHE requires people to enter the entity-relationship schemas manually to the Database Schema Mapper Module (see 3.5). This application could help to automate this process, and perhaps provide additional relationships, which the DHE could display for database applications.

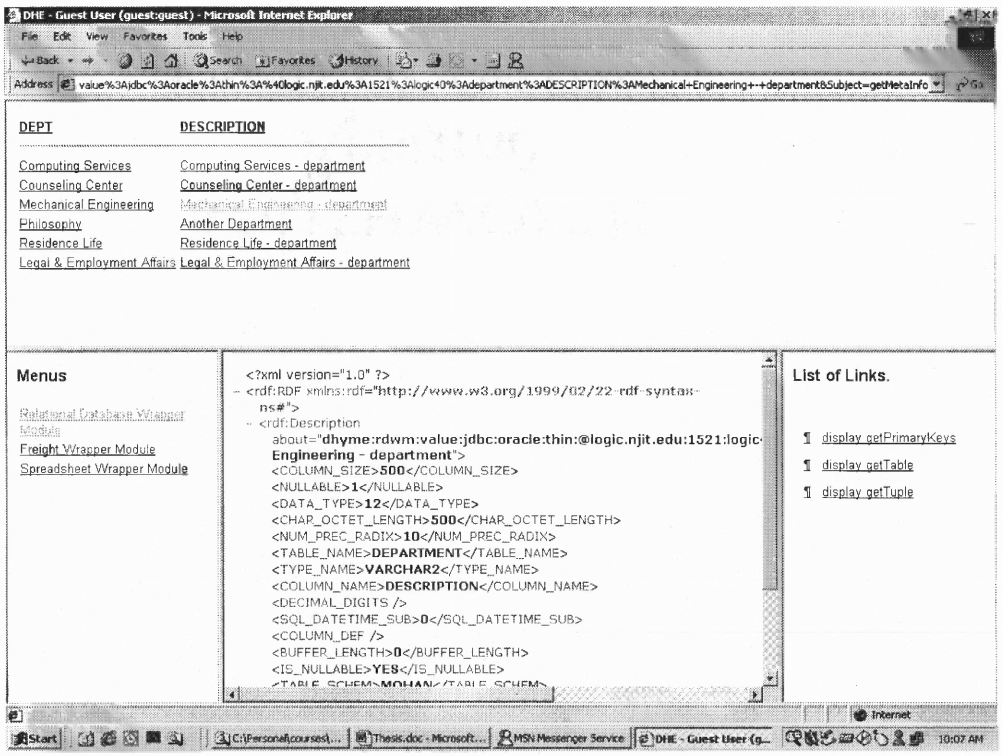
While the hypermedia paradigm embodies an approach to structure and navigate information, it has several shortcomings. In particular, few hypermedia systems have focused on methodologies for information storage and retrieval. Database systems, on the contrary, are only concerned with storage and retrieval of information based on a formal model. They exhibit powerful methodologies for information storage, and effective indexing and querying. Furthermore they provide facilities such as transaction management, concurrency and access control as well as locking mechanisms.

The DHE should be able to streamline a company's software development efforts in several ways. It automatically supplements the organization's applications with hypermedia links, structuring, navigation and annotative functionality, and metadata. It also implements inter-application linking, as the university department example shows. Mapping rules can point to any accessible application. DHE also can speed the development of applications. Developers can offload link management, navigational structures (such as guided tours), user preference management and other features to DHE.

CHAPTER 2

THE DYNAMIC HYPERMEDIA ENGINE

A Web-based prototype of the Dynamic Hypermedia Engine (DHE) has been developed, which redesigns an older PC-based prototype (Bieber 1999). Figure 1 shows a screenshot of a database query result in the main frame. The DHE has added anchors to all parts of the query result, including the field names at the column heads. The user has clicked on "Counseling Center - department," resulting in metadata for the element in the bottom center frame and a list of links in the bottom right-hand frame. Selecting any link will generate an SQL query to create the appropriate result. Currently the list of links includes only database structural links, such as finding the primary keys for this element. The bottom left-hand frame contains menus for any integrated application or DHE internal module. Links represent relationships and relationships have "meta-information" as well. Selecting the "¶" next to any link will provide metadata and a list of links for it. The DHE's next release will provide these for menu items as well. The metadata frame currently displays the full Resource Description Framework (RDF) record. The DHE's next release will preformat the metadata for display. Future versions will also filter and rank the links and metadata based on the user task and preferences.


DHE - Guest User (guest:guest) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address
Menus
List of Links.
Start
C:\Personal\courses\...
Thesis.doc - Microsoft...
MSN Messenger Service
DHE - Guest User (g...
Internet
10:07 AM

value%3Ajdbc%3Aoracle%3Athin%3A%40logic.njit.edu%3A1521%3Alogic40%3Adepartment%3ADESCRIPTION%3AMechanical+Engineering+-+department&Subject=getMetaInfo

DEPT	DESCRIPTION
Computing Services	Computing Services - department
Counseling Center	Counseling Center - department
Mechanical Engineering	Mechanical Engineering - department
Philosophy	Another Department
Residence Life	Residence Life - department
Legal & Employment Affairs	Legal & Employment Affairs - department

Relational Database Wrapper Module
Freight Wrapper Module
Spreadsheet Wrapper Module

```
  <?xml version="1.0" ?>
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
   ns#">
 - <rdf:Description
     about="dhyme:rdwm:value:jdbc:oracle:thin:@logic.njit.edu:1521:logic
     Engineering - department">
     <COLUMN_SIZE>500</COLUMN_SIZE>
     <NULLABLE>1</NULLABLE>
     <DATA_TYPE>12</DATA_TYPE>
     <CHAR_OCTET_LENGTH>500</CHAR_OCTET_LENGTH>
     <NUM_PREC_RADIX>10</NUM_PREC_RADIX>
     <TABLE_NAME>DEPARTMENT</TABLE_NAME>
     <TYPE_NAME>VARCHAR2</TYPE_NAME>
     <COLUMN_NAME>DESCRIPTION</COLUMN_NAME>
     <DECIMAL_DIGITS />
     <SQL_DATETIME_SUB>0</SQL_DATETIME_SUB>
     <COLUMN_DEF />
     <BUFFER_LENGTH>0</BUFFER_LENGTH>
     <IS_NULLABLE>YES</IS_NULLABLE>
     <TABLE_SCHEM>MOHAN</TABLE_SCHEM>
```

¶ display getPrimaryKeys
¶ display getTable
¶ display getTuple

Figure 1 : Screenshot of the DHE

As this demonstrates, link generation in the DHE does not result from any type of lexical analysis. Our focus is not on the display content of the link anchor, but rather on the application elements underlying each anchor. A "mapping rule" encodes each relationship found between two elements of interest at the "class level". For example, suppose an application display shows the name of a university department. Departments generally have professors and courses taught (based on the standard entity-relationship diagram within a database system), as well as a Web page, an annual budget (within the accounting system), hires-in-progress (within the personnel system), a location on a map (within a geographic information system), etc. Individual mapping rules contain an algorithm or computation (set of commands) leading to the appropriate component in these respective systems. When the user selects a particular department, the DHE constructs these commands with the actual department instance selected and sends them to the appropriate destination system, which then retrieves— or more often generates— the resulting page. For example, one mapping rule could state that an element of type "department" would be related to an element of type "annual budget" through a relationship with the semantic type "annual budget for" and with a parameterized command to retrieve annual budgets from the accounting system. Developers may take advantage of this to integrate database applications with other applications without altering their contents; they only have to add new mapping rules for the relevant element types.

The DHE executes concurrently with database management systems, database applications, and other applications such as the accounting system, providing automated link generation and other hypermedia functionality without altering them. Developers write an independent application "wrapper" and a set of mapping rules for each. Note that once a wrapper is written and the mapping rules are specified for each type of application (geographic information system, relational database management system, accounting package, etc.), the DHE will support all instances of that application in the future (new maps, database contents, budget sheets, etc.).

The DHE executes as follows. Applications or their wrappers connect to DHE through World Wide Web components, such as Servlets and JavaServer pages. It intercepts all messages passing between the application and its user interface, and uses the mapping rules to map each appropriate element of the message to a hypermedia anchor. The DHE's Web browser wrapper merges these anchors into the document being displayed and passes the resulting HTML document through the Web component servlet to the user's Web browser. When the user selects an anchor, the browser wrapper passes it to DHE, which returns a list of possible links (one for each appropriate relationship as determined by the mapping rules) and metadata. If the user selects a DHE link (e.g., to add an annotation or stage in a guided tour), the DHE processes it entirely. If the user selects a relationship with a destination in a known application, the DHE infers and instantiates the appropriate SQL queries or other application commands from the relationship's mapping rule and passes them to the target application for processing. If the user selects a user-created annotation or tour, etc., the DHE retrieves it. Thus the DHE automatically provides all hypermedia linking (as well as navigation) to applications, which remain hypermedia-unaware and in fact often entirely unchanged.

Figure 2 shows the DHE's logical engine architecture. Some major components are described here. The others are described on our project Web site (http://space.njit.edu:8001). The current prototype has been developed in Java. XML is the messaging format used for intramodule communication, and RMI is the underlying protocol used to transfer these messages. While the browser wrapper currently produces HTML documents for display, a future version will produced XML documents, which take advantage of the Web's new XLink, and XPointer standards to handle anchors and links.

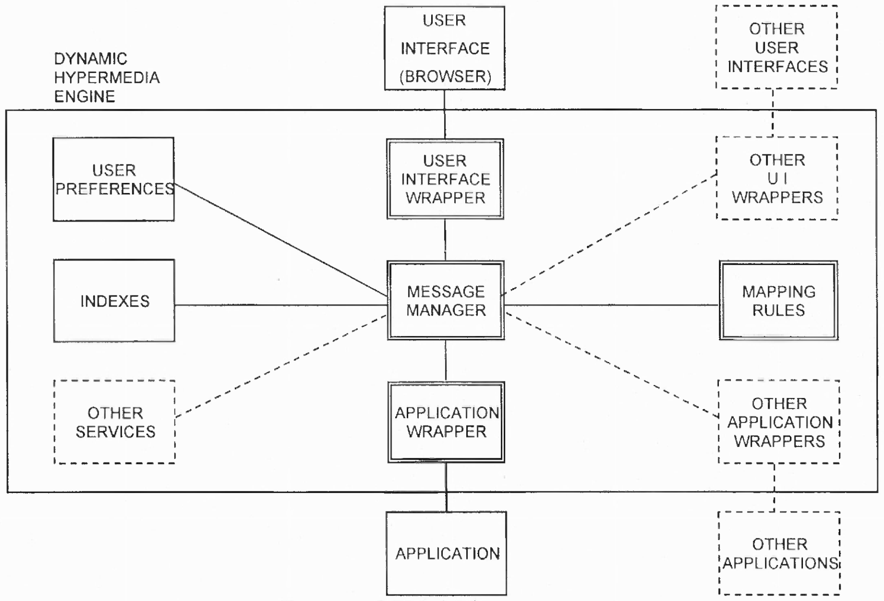


Figure 2: DHE Architecture

User Interface Wrappers serve three important functions: First, they translate the DHE's internal messages from DHE's standard format to a format the browser (or other User Interface or UI) can process, and vice versa. Second, they handle communication between the engine and the UI. Third, they implement any functionality DHE requires from the UI (e.g., maintaining parameters), which the UI cannot provide itself.

The *Message Manager Module* enables the communication between all DHE modules, routing all DHE internal messages.

The *Mapping Rules Module* maps the application data and relationships to hypermedia objects at run-time. The Mapping Rules Module maps the element instances in the virtual document to global element types (classes), and infers all relevant relationships (links) and metadata for the given element classes. These links and metadata are passed in messages to the UI Wrapper for display.

Application Wrappers, like user interface wrappers, manage the communication between DHE and their application systems, such as database applications and DBMS. They translate user requests from DHE's internal format to the application's programming interface (if any). They receive output from the application, convert it to the DHE format, mark the elements for the mapping rules module, and send it to DHE for eventual display on the UI.

Other Hypermedia Functionality: A series of other service modules will be implemented in future versions. Most will implement various kinds of hypermedia structuring, navigation and annotation functionality (Bieber et al. 1997; Conklin 1987; Nielsen 1995). Hypermedia structuring functionality includes local and global information overviews; node, link and anchor typing; as well as keywords, attributes and metadata on all of these. Navigation functionality includes structure-based query, sophisticated history-based navigation and bi-directional linking. Annotation functionality includes adding user-declared links, comments and bookmarks to dynamically-generated documents and displays.

CHAPTER 3

HYPERMEDIA SUPPORT FOR RELATIONAL DATABASE MANAGEMENT SYSTEMS

3.1 Introduction to the Relational Database Wrapper Module

One of the first tasks in providing hypermedia support is to intercept messages between the computational and user interface (UI) portions of the DMIS.

In the case of a Relational Database Management System (RDBMS), it provides only the computational portion. The UI portion is the responsibility of the database application. The RDBMS provides a standard way of requesting computational services (i.e. store, retrieve and analyze data) by means of Structured Query Language (SQL) statements. Database applications send SQL statements to an RDBMS. The RDBMS then executes these statements and returns the results of these statements back to the application, which then displays these results in a UI intuitive to its domain.

We have developed a service module, the Relational Database Wrapper Module (RDWM) that accepts requests to, execute SQL statements on the underlying database, and retrieve metadata of an element. It will also provide a UI allowing users to execute SQL statements and view results, metadata and all the relationships of the data affected by the SQL statement. This UI provides a hypertext-enriched view of the data stored in the database and also acts as the debug or test interface for our module.

Once a document containing the results of the original request has been created, our module identifies and marks all "elements of interest". An "element of interest" is an entity that may have a relationship with another entity (element) or may have metadata of its own.

This document is then sent to the DHE Message Manager, which routes it to an intermediate module called the Mapping Rules Module (MRM). The MRM maintains a list of Mapping Rules, which are representations of the various inter-relationships between elements in a domain. It retrieves all the Mapping Rules that each marked-up "element of interest" may have and creates hyperlinks corresponding to each one of them.

The modified XML document containing the results of the original SQL statement (or the requested metadata) and hyperlinks is returned to the gateway, which then sends it onward to the database application. The database application thus receives hypertext-enriched result of a SQL statement from the RDBMS, and is free to interpret it in a manner suitable to its domain.

For example when the user uses the UI to execute a SQL statement a message containing the SQL statement is sent from the UIW to the RDWM (by way of the Message Manager). The RDWM executes the SQL statement on the underlying database, identifies the "elements of interest", marks them up and sends back a response containing this to the module that had sent the original request (i.e. the UIW). The Message Manager routes this message to the Mapping Rules Module, which applies the appropriate Mapping Rules and forwards the response to the UIW (as before, by way of the Message Manager).

The Message Flow is as:

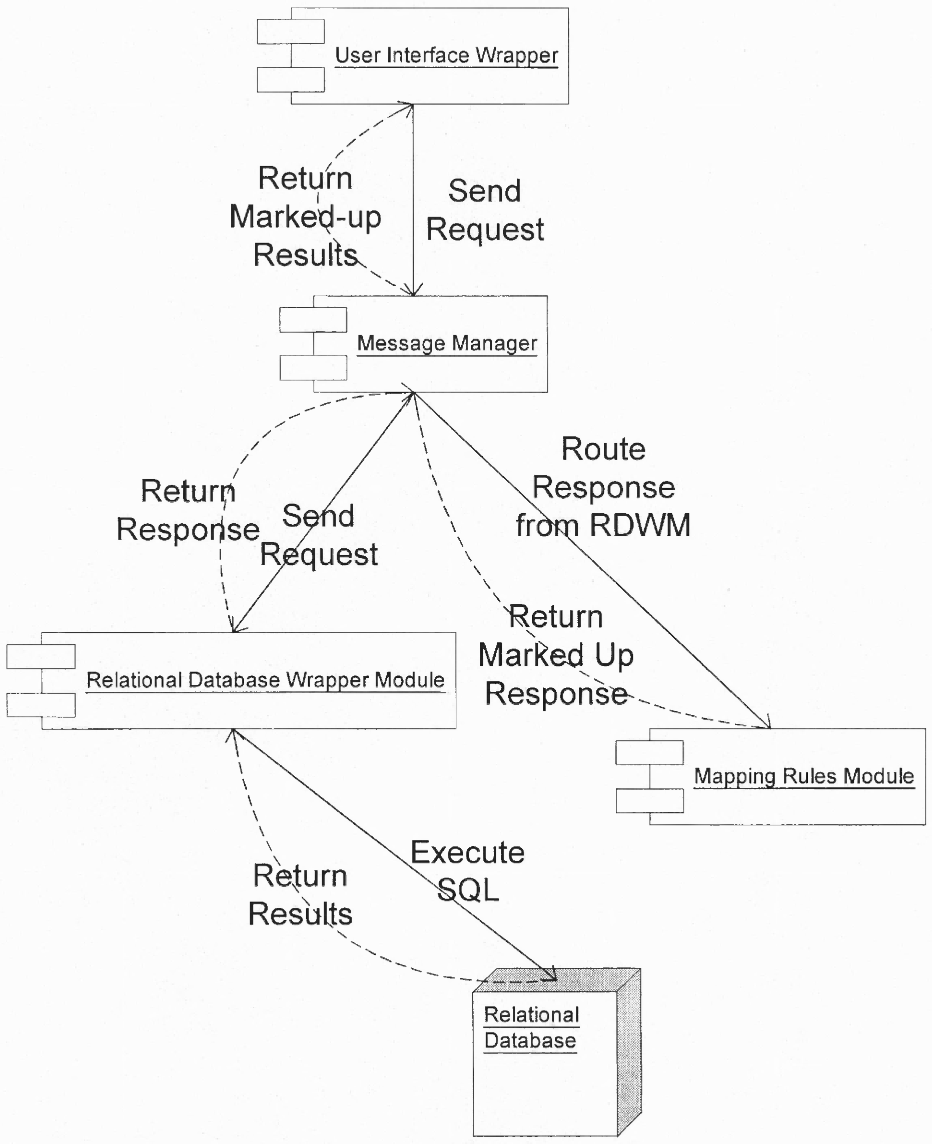


Figure 3: Relational Database Wrapper Module Message Flow

3.2 Elements of Interest in Database Systems.

3.2.1 Element Types.

The following types of elements exist in the RDBMS context:

- Columns.
- Tables.

- Indices.
- Stored Procedures.
- Catalogs.
- Schema.
- Drivers.
- Users.
- User Rights.
- Table and Column Privileges.
- JDBC Types.
- The RDBMS Product itself.

Any instance of the above types can be uniquely identified, have metadata, and have one or more relationships associated with it. Because any user may be interested in exploring that type of object in terms of its metadata or relationships, the RDWM must mark each of its instances as an "element of interest".

3.2.2 Marking an Element of Interest.

Marking an item involves providing its unique identifier and its element type. When the results of the SQL statement is returned from the RDBMS, the RDWM must parse through this document and locate all elements and mark all instances with locator tags. An element's locator tag references its unique identifier and its type.

Later on as the Message containing this document makes its way to the Mapping Rules Module it will use the element type information to find relationships and metadata for that element. If an element has at least one relationship or piece of metadata, then the Mapping Rules Module will specify that the UI Wrapper make it into a hyperlink.

3.2.3 Element Identifiers.

When the user follows a hyperlink, the action is passed on to the underlying DMISW (in our case the RDWM), as mentioned previously this action may either be a request for the hyperlinked element's metadata or the user may be trying to follow a relationship with another element in which case it would be another SQL statement that has to be executed. In either case the element must be uniquely identified.

Like other DHE modules the RDWM uses the Uniform Resource Identifier (URI) syntax (Berners-Lee 1998) to define its element identification scheme. The generic URI syntax specifies that the name of the scheme must be specified followed by a colon (which acts as the delimiter), and followed by the scheme-specific part.

```
<scheme name>:<scheme-specific-part>
```

The DHE URI scheme is named "dhyme" (Dynamic Hypermedia Engine), and all URI's must also contain the module name which identifies the domain the element belongs to.

For elements belonging to the RDWM domain we use the following syntax.

```
dhyme:rdwm:<element type>:<Database JDBC URL>:<element specific part>
```

The JDBC (Java Database Connectivity) URL of the database provides a way of identifying it so that the appropriate database driver will recognize it and establish a connection with it. (White 1999)

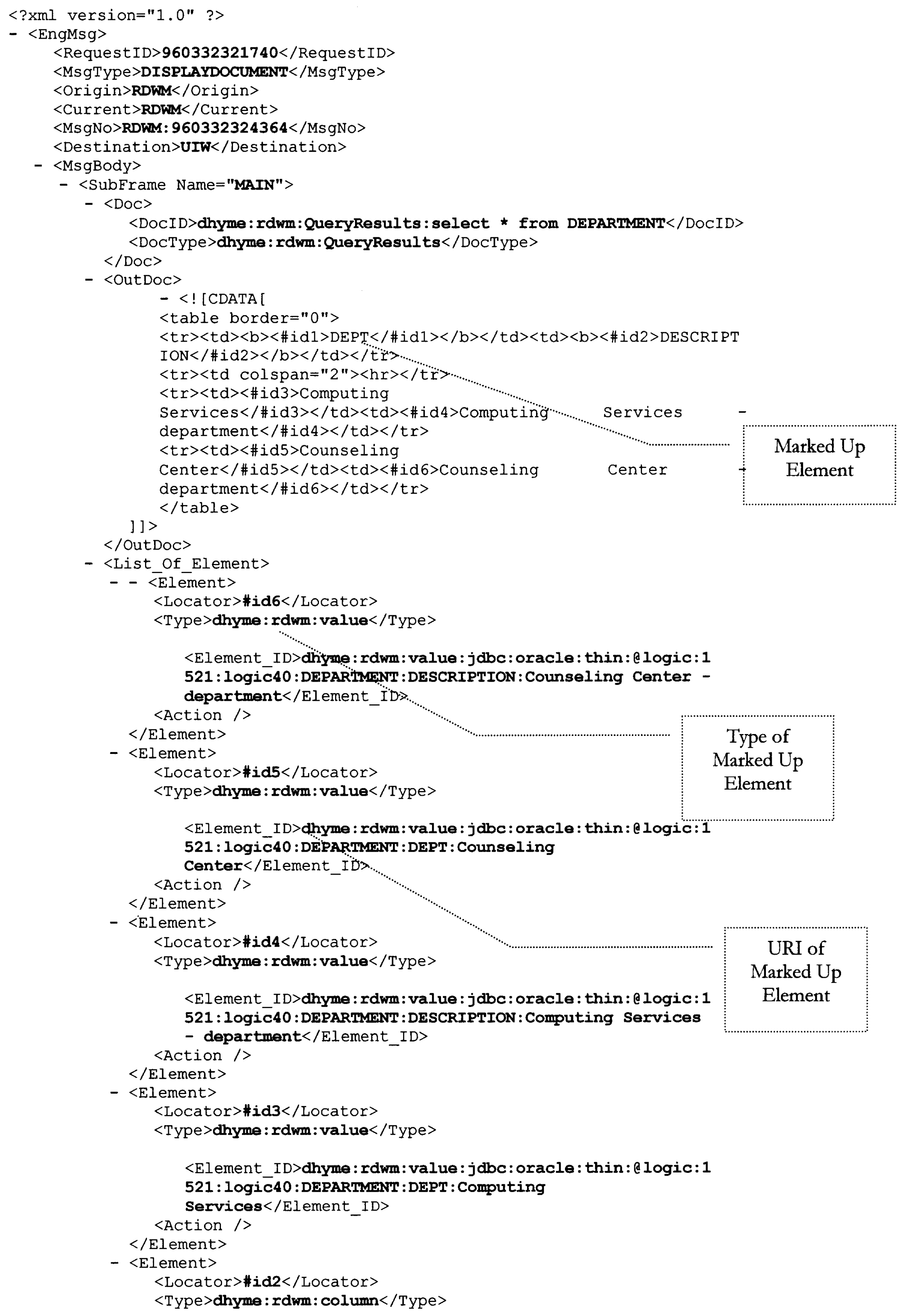
Thus a column called "DEPT" in the table "DEPARTMENT" in the database with the JDBC URL "jdbc:oracle:thin:@logic:1521:logic40" would have the following URI.

```
dhyme:rdwm:column:jdbc:oracle:thin:@logic:1521:logic40:DEPARTMENT:DEPT
```

Because of the case insensitive nature of SQL statements this scheme too, is case insensitive.

3.2.4 Marked up Message.

The result of the SQL statement "select * from department" on the database the JDBC URL "jdbc:oracle:thin:@logic:1521:logic40" is as.



```
<?xml version="1.0" ?>
- <EngMsg>
    <RequestID>960332321740</RequestID>
    <MsgType>DISPLAYDOCUMENT</MsgType>
    <Origin>RDWM</Origin>
    <Current>RDWM</Current>
    <MsgNo>RDWM:960332324364</MsgNo>
    <Destination>UIW</Destination>
  - <MsgBody>
    - <SubFrame Name="MAIN">
      - <Doc>
          <DocID>dhyme:rdwm:QueryResults:select * from DEPARTMENT</DocID>
          <DocType>dhyme:rdwm:QueryResults</DocType>
        </Doc>
      - <OutDoc>
            - <![CDATA[
            <table border="0">
            <tr><td><b><#id1>DEPT</#id1></b></td><td><b><#id2>DESCRIPT
            ION</#id2></b></td></tr>
            <tr><td colspan="2"><hr></tr>
            <tr><td><#id3>Computing
            Services</#id3></td><td><#id4>Computing      Services      -
            department</#id4></td></tr>
            <tr><td><#id5>Counseling
            Center</#id5></td><td><#id6>Counseling       Center       -
            department</#id6></td></tr>
            </table>
          ]]>
        </OutDoc>
      - <List_Of_Element>
        - - <Element>
              <Locator>#id6</Locator>
              <Type>dhyme:rdwm:value</Type>

                <Element_ID>dhyme:rdwm:value:jdbc:oracle:thin:@logic:1
                521:logic40:DEPARTMENT:DESCRIPTION:Counseling Center -
                department</Element_ID>
              <Action />
            </Element>
        - <Element>
              <Locator>#id5</Locator>
              <Type>dhyme:rdwm:value</Type>

                <Element_ID>dhyme:rdwm:value:jdbc:oracle:thin:@logic:1
                521:logic40:DEPARTMENT:DEPT:Counseling
                Center</Element_ID>
              <Action />
            </Element>
        - <Element>
              <Locator>#id4</Locator>
              <Type>dhyme:rdwm:value</Type>

                <Element_ID>dhyme:rdwm:value:jdbc:oracle:thin:@logic:1
                521:logic40:DEPARTMENT:DESCRIPTION:Computing Services
                - department</Element_ID>
              <Action />
            </Element>
        - <Element>
              <Locator>#id3</Locator>
              <Type>dhyme:rdwm:value</Type>

                <Element_ID>dhyme:rdwm:value:jdbc:oracle:thin:@logic:1
                521:logic40:DEPARTMENT:DEPT:Computing
                Services</Element_ID>
              <Action />
            </Element>
        - <Element>
              <Locator>#id2</Locator>
              <Type>dhyme:rdwm:column</Type>
            <Element_ID>dhyme:rdwm:column:jdbc:oracle:thin:@logic:
            1521:logic40:DEPARTMENT:DESCRIPTION</Element_ID>
         <Action />
       </Element>
     - <Element>
         <Locator>#id1</Locator>
         <Type>dhyme:rdwm:column</Type>

            <Element_ID>dhyme:rdwm:column:jdbc:oracle:thin:@logic:
            1521:logic40:DEPARTMENT:DEPT</Element_ID>
         <Action />
       </Element>
     </List_Of_Element>
   </SubFrame>
 </MsgBody>
</EngMsg>
```

Figure 4: Marked up Message

3.3 Mapping Rules and Metadata for Relational Databases.

DHE specifies relationships based on the element type. The twelve types of elements noted above are interrelated. These inter-relationships are depicted below.

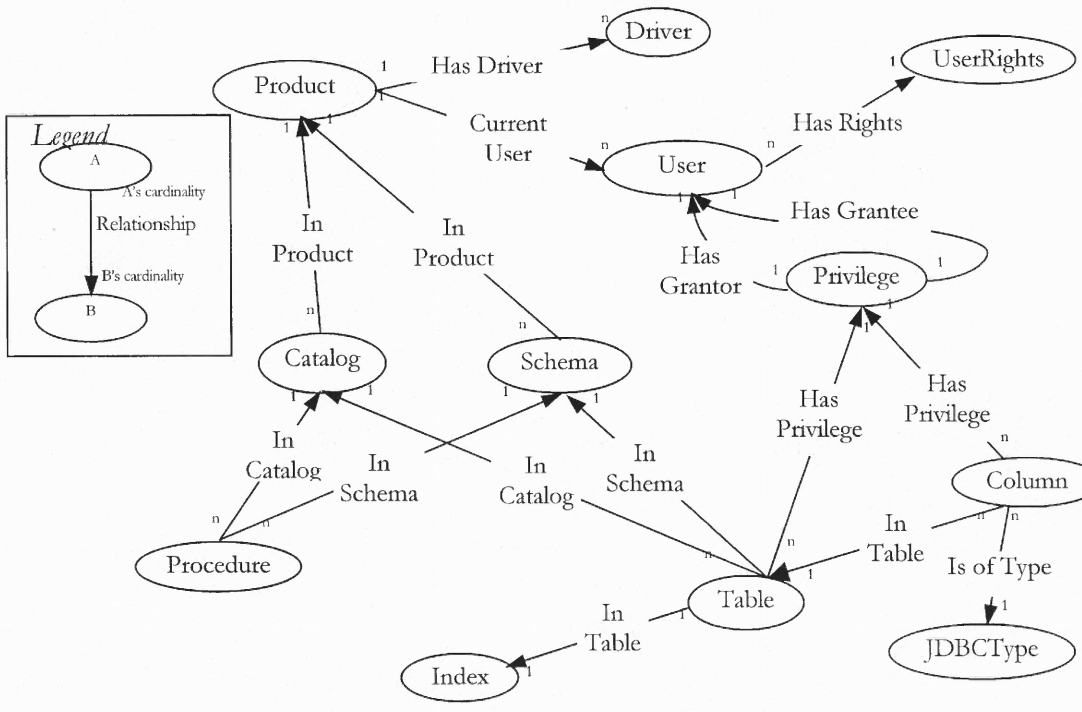


Figure 5: Relationships between Element Types

Each of these relationships are reflexive i.e., if a column has a "in table" relationship with a table then the table has a relationship called "has columns" with the column.

A mapping rule for each of the above relationships may be specified. The DHE (specifically the Mapping Rules Module) would then use the mapping rule to generate a link for each of the instances of the participating elements.

For example all instances of columns would have a hyperlink corresponding to the "in table" relationship that it has with a table. This link would contain the appropriate SQL command to generate the contents of the mapping rule's endpoint, in this case the table that this column belongs to. Wan and Bieber (Wan 1996)(Bieber 1997) have given several mapping rules (called "bridge laws") for relational databases.

3.3.1 Mapping Rules for Relational Database Management Systems.

Wan and Bieber (Wan 1996) have identified the following classes of Mapping Rules for databases:

Object Mapping Rules: These map the contents of database objects, as well as schemata and ER diagrams.

Structure Mapping Rules: These map database objects to their structural containers, for example a mapping rule for columns in a table, tables in a catalog etc. These would correspond to the "In Object" relationships mentioned in Figure # 5.

Operation Mapping Rules: These map SQL queries. Frequently used specific queries maybe mapped, and users who follow link generated from this mapping rule would retrieve a document that contained the results of the SQL statement being executed.

Schema-based Mapping Rules: These mapping rules map relationships between entities as defined in the database schema.

Meta-information Mapping Rules: Meta-information Mapping Rules define reference links to database object statistics such as field types, size, table size, referential constraints etc.

3.3.1.1 Mapping Rules examples.

The following Mapping Rules have been implemented in the current DHE prototype.

- Structure Mapping Rules:

- Mapping Rule$_{getTable}$ maps tables to a set of records.
- Mapping Rule$_{getTuple}$ maps the tuple a database value belongs to.

- Schema-based Mapping Rules:
  - Mapping Rule$_{getPrimaryKeys}$ maps the primary keys of a table.
  - Mapping Rule$_{getForeignKeys}$ maps the foreign keys in a table.
- Meta-information Mapping Rule:
  - MappingRule$_{getMetaInfo}$ maps the meta-information of a Database Object.

3.3.2 Metadata for Relational Database Management Systems.

Metadata is data about data. The key purpose of metadata is to facilitate and improve the retrieval of information (Miller 1998). Other uses of Metadata include providing semantic information about the data, encoding information (i.e. how to interpret the data), relationships with other resources etc. (Waugh 1997).

In a relational database, metadata is the representation of the objects defined in that database - specifically, the definitions of its tables, columns and business rules and transformational algorithms implemented as stored procedures and triggers (Gardner 1998).

The RDWM retrieves this metadata on demand. It is passed the URI of the element whose metadata is being asked for. It then uses the Java Database Connectivity (JDBC) API to retrieve information relevant to that element. The RDWM uses the Resource Description Framework (RDF) to model the metadata, and the RDF/XML serialization syntax to format it for transfer to other modules.

3.3.2.1 The Resource Description Framework.

RDF is a framework for describing and interchanging metadata it defines metadata in terms of "Resources", where each resource is any entity that can be uniquely identified, i.e., has a URI, and is defined by its properties (Lassila 1999).

Hence all "elements of interest" are resources and have associated metadata. Resources and elements are thus interchangeable in the DHE context and the element identifier corresponds to its URI. Thus there exists a Resource for every element type mentioned in section 3.2.1.

RDF also defines a "Property Type" as a resource that has a name and can be used to define a resource property. Consider the statement "The student's name is John Doe". "Student" is the resource, "name" is the property type and "John Doe" is the property.

3.3.2.1.1 RDF Schema for Databases.

However the semantics of the statement in the previous section is meaningful to the reader if and only if (s)he knows what the property type "name" actually means. RDF provides a schema mechanism, which allows authors (i.e. those who define what the resources and property types are) to define the terms that will be used in RDF statements and to give specific meaning to them. The RDF schema may thus be considered to be the dictionary to the RDF vocabulary in use (Brickley 2000).

To represent RDBMS resources in RDF, an RDF schema containing the various property types used by the RDWM has been defined in Appendix A.

3.3.2.1.2 RDF/XML Serialization Syntax

A formal syntax representing this metadata model is required to store instances of this model into machine-readable files and to communicate these instances among applications. XML is this syntax. RDF imposes formal structure on XML to support the consistent representation of semantics (Miller 1998).

For example consider the following model that represents metadata about a column called "Description" in the table "Department" on the Oracle instance on "logic".

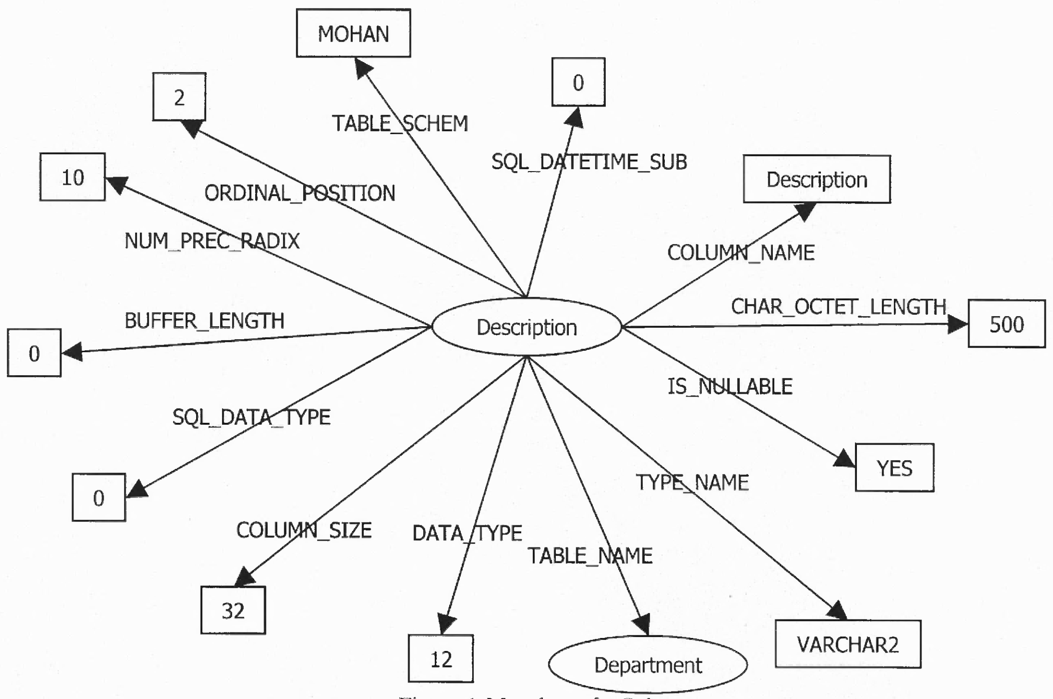


Figure 6: Metadata of a Column

The above model in RDF/XML serialization syntax is:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description
about="dhyme:rdwm:column:jdbc:oracle:thin:@logic:1521:logic40:Departmen
t:Description">
<ORDINAL_POSITION>2</ORDINAL_POSITION>
<CHAR_OCTET_LENGTH>500</CHAR_OCTET_LENGTH>
<SQL_DATETIME_SUB>0</SQL_DATETIME_SUB>
<TABLE_SCHEM>MOHAN</TABLE_SCHEM>
<DECIMAL_DIGITS></DECIMAL_DIGITS>
<COLUMN_NAME>Description</COLUMN_NAME>
<NUM_PREC_RADIX>10</NUM_PREC_RADIX>
<COLUMN_SIZE>500</COLUMN_SIZE>
<TABLE_CAT></TABLE_CAT>
<COLUMN_DEF></COLUMN_DEF>
<REMARKS></REMARKS>
<IS_NULLABLE>YES</IS_NULLABLE>
<DATA_TYPE>12</DATA_TYPE>
<BUFFER_LENGTH>0</BUFFER_LENGTH>
<SQL_DATA_TYPE>0</SQL_DATA_TYPE>
<TABLE_NAME>Department</TABLE_NAME>
<TYPE_NAME>VARCHAR2</TYPE_NAME>
</rdf:Description>
```

```
</rdf:RDF>
```

Figure 7: RDF/XML Syntax

3.3.2.2 Architecture and Design of the Metadata Retrieval Subsystem

The Metadata Retrieval Subsystem is composed of two broad components.

- A set of utility classes responsible for storing Metadata and representing them in the RDF/XML serialization syntax, these classes comprise the "RDF Package". These classes maybe used outside the RDWM and indeed outside the DHE.

- A set of classes extending the functionality provided in the RDF package for use in the RDBMS context.

3.3.2.2.1 The RDF Package

The RDF Package essentially consists of two core classes the "`Resource`" class and the "`ResourceProperty`" class. These classes are present in the Java package "`dhyme.metadata.rdf`".

3.3.2.2.1.1 Object Oriented Analysis and Design

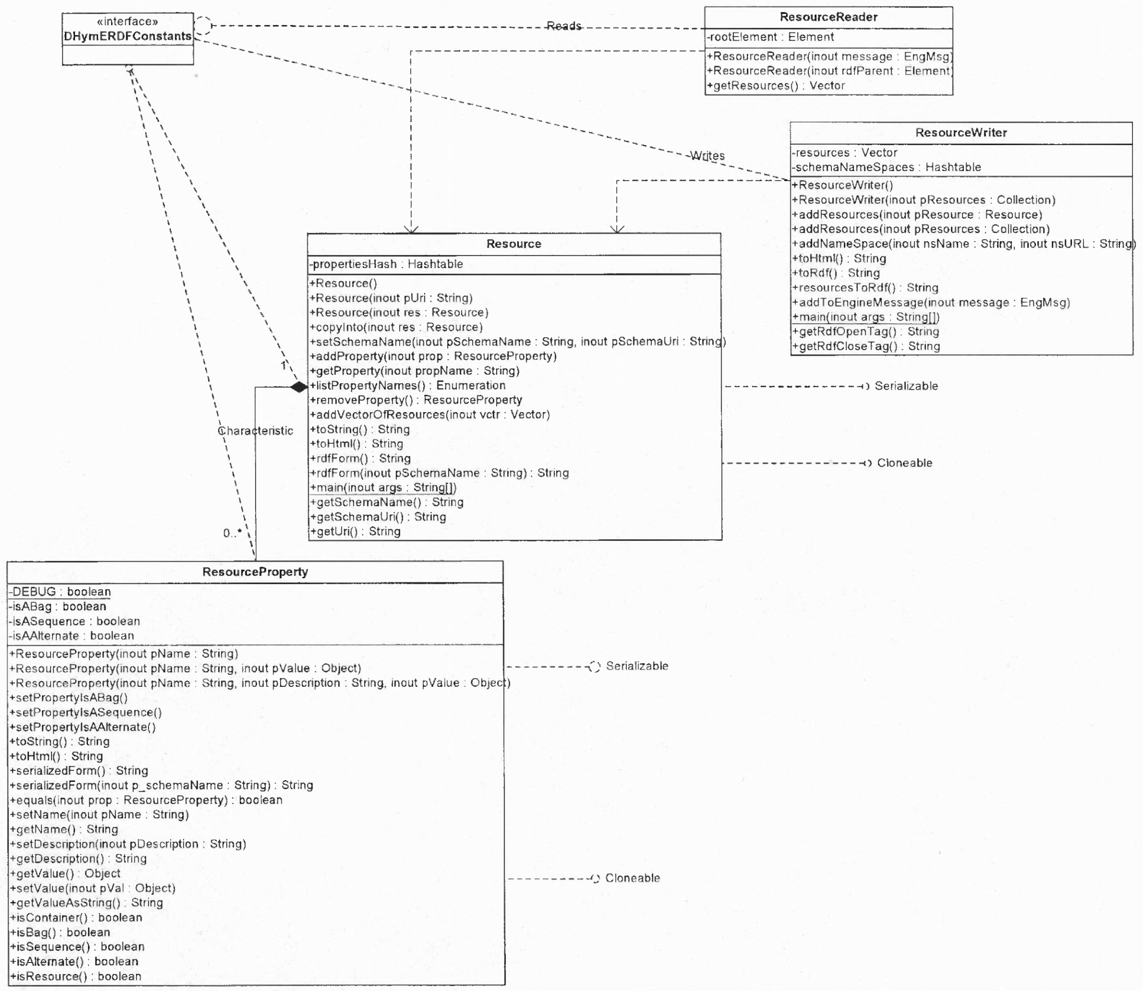

«interface»
DHymERDFConstants
Reads
Writes
ResourceReader
-rootElement : Element
+ResourceReader(inout message : EngMsg)
+ResourceReader(inout rdfParent : Element)
+getResources() : Vector
ResourceWriter
-resources : Vector
-schemaNameSpaces : Hashtable
+ResourceWriter()
+ResourceWriter(inout pResources : Collection)
+addResources(inout pResource : Resource)
+addResources(inout pResources : Collection)
+addNameSpace(inout nsName : String, inout nsURL : String)
+toHtml() : String
+toRdf() : String
+resourcesToRdf() : String
+addToEngineMessage(inout message : EngMsg)
+main(inout args : String[])
+getRdfOpenTag() : String
+getRdfCloseTag() : String
Resource
-propertiesHash : Hashtable
+Resource()
+Resource(inout pUri : String)
+Resource(inout res : Resource)
+copyInto(inout res : Resource)
+setSchemaName(inout pSchemaName : String, inout pSchemaUri : String)
+addProperty(inout prop : ResourceProperty)
+getProperty(inout propName : String)
+listPropertyNames() : Enumeration
+removeProperty() : ResourceProperty
+addVectorOfResources(inout vctr : Vector)
+toString() : String
+toHtml() : String
+rdfForm() : String
+rdfForm(inout pSchemaName : String) : String
+main(inout args : String[])
+getSchemaName() : String
+getSchemaUri() : String
+getUri() : String
Serializable
Cloneable
Characteristic
1
0..*
ResourceProperty
-DEBUG : boolean
-isABag : boolean
-isASequence : boolean
-isAAlternate : boolean
+ResourceProperty(inout pName : String)
+ResourceProperty(inout pName : String, inout pValue : Object)
+ResourceProperty(inout pName : String, inout pDescription : String, inout pValue : Object)
+setPropertyIsABag()
+setPropertyIsASequence()
+setPropertyIsAAlternate()
+toString() : String
+toHtml() : String
+serializedForm() : String
+serializedForm(inout p_schemaName : String) : String
+equals(inout prop : ResourceProperty) : boolean
+setName(inout pName : String)
+getName() : String
+setDescription(inout pDescription : String)
+getDescription() : String
+getValue() : Object
+setValue(inout pVal : Object)
+getValueAsString() : String
+isContainer() : boolean
+isBag() : boolean
+isSequence() : boolean
+isAlternate() : boolean
+isResource() : boolean
Serializable
Cloneable

Figure 8: Class diagram of the "`dhyme.metadata.rdf`" package.

3.3.2.2.1.2 Features

- Each `Resource` Object has 0..n `ResourceProperty` Objects.
- Each `ResourceProperty` defines an attribute of the `Resource`, in turn a `Resource` Object is defined by the `ResourceProperty` Objects it "has".
- A `Resource` must have a Uniform Resource Identifier (URI). A URI uniquely identifies the `Resource` and by extension the `ResourceProperty` Objects it possesses.
- A `ResourceProperty` may be another `Resource` or a collection (referred to in RDF syntax as a "container") of `Resource` Objects. This collection may be a set of unordered `Resource` Objects (referred to in RDF syntax as a "bag") or it may be a set of ordered `Resource` Objects (referred to in RDF syntax as a "sequence"), or it may be a set of alternates, i.e. each `Resource` Object may substitute the other (referred to in RDF syntax as a "alternate" or "alt").
- A `Resource` Object has the ability to add, retrieve and remove `ResourceProperty` Objects.
- A `Resource` Object has the ability to express itself in the complete RDF-XML "serialization" syntax.
- A `Resource` Object has the ability to express itself in HTML and in Plain Text as an aid to debugging and for viewing metadata on browsers.
- `ResourceWriter` is a utility class used to "write" a collection of Resources into an Engine Message or express them in the RDF/XML serialization syntax.
- `ResourceReader` is a utility class that may be used to parse RDF resources from an Engine Message or from an RDF Element (i.e. the tag named rdf:RDF).

3.3.2.2.1.3 Recommended Usage

Create an instance of a `Resource` object and add/remove/change it's properties by adding/removing `ResourceProperty` Objects to its collection of `ResourceProperty` Objects (as maintained in an internal `Hashtable`). The ability to serialize a `Resource` into it's proper RDF/XML format is available by means of calls to `rdfForm()` and `rdfForm(schemaName)` on it's instance.

Another approach is to subclass `Resource` class and then "load" `ResourceProperty` Objects in the constructor of the subclass. An example of this Usage is provided by the RDWM metadata retrieval subsystem.

3.3.2.2.2 The RDWM Metadata Retrieval Subsystem

This subsystem is an extension of the RDF package applied to the Relational Database Domain. The classes that comprise this subsystem are present in the "`dhyme.metadata.database`" package.

3.3.2.2.2.1 Object Oriented Analysis and Design

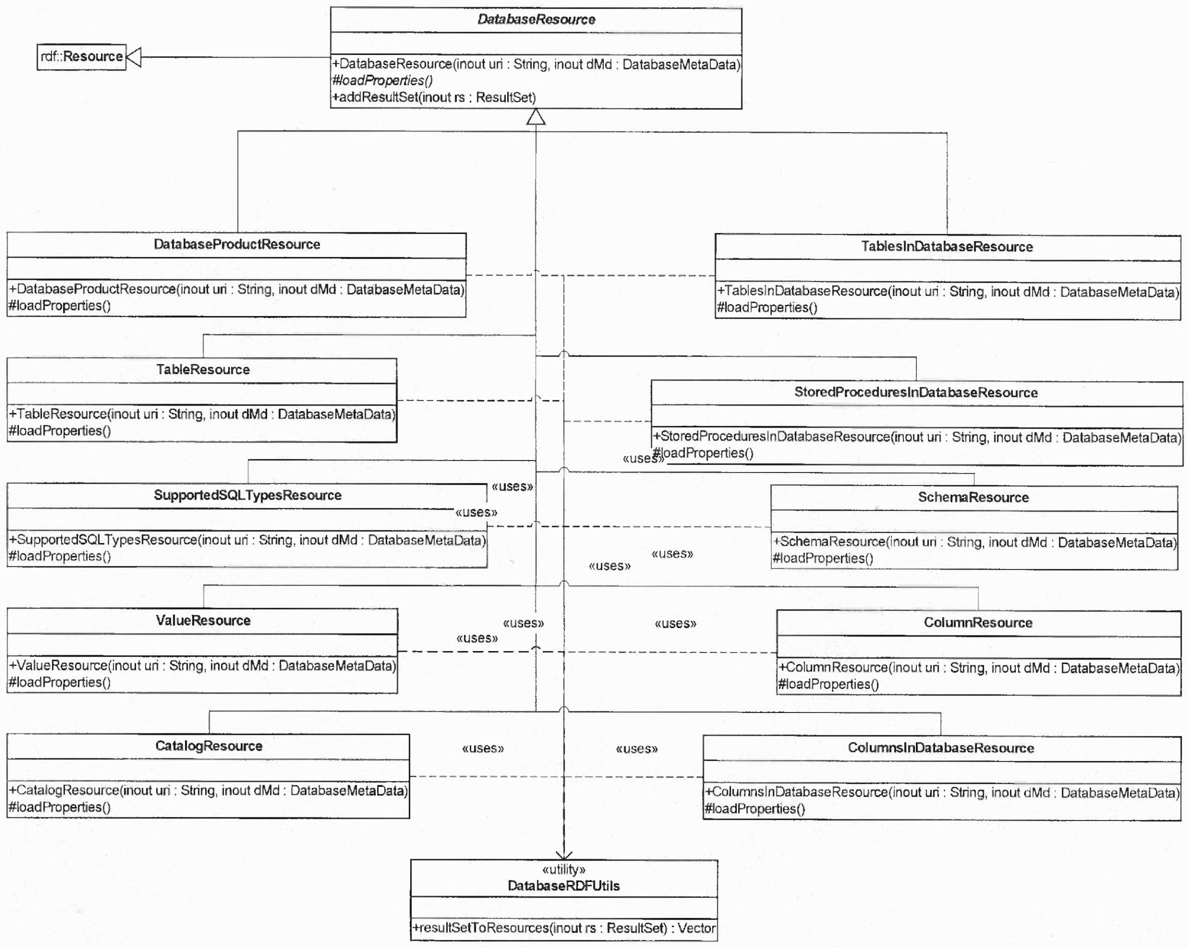

DatabaseResource
+DatabaseResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
+addResultSet(inout rs : ResultSet)
rdf::Resource
DatabaseProductResource
+DatabaseProductResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
TablesInDatabaseResource
+TablesInDatabaseResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
TableResource
+TableResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
StoredProceduresInDatabaseResource
+StoredProceduresInDatabaseResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
SupportedSQLTypesResource
+SupportedSQLTypesResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
SchemaResource
+SchemaResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
ValueResource
+ValueResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
ColumnResource
+ColumnResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
CatalogResource
+CatalogResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
ColumnsInDatabaseResource
+ColumnsInDatabaseResource(inout uri : String, inout dMd : DatabaseMetaData)
#loadProperties()
«uses»
«utility»
DatabaseRDFUtils
+resultSetToResources(inout rs : ResultSet) : Vector

Figure 9: Class diagram of the "`dhyme.metadata.database`" package.

3.3.2.2.2.2 Features

- The core of this package is an abstract class called `DatabaseResource`, which in turn extends the `Resource` class. `DatabaseResource` provides an abstract method called `loadProperties`, which takes as an argument a `java.sql.DatabaseMetadata` Object.

- It is then the responsibility of each subclass of `DatabaseResource` to extract whatever metadata is relevant to it from the `DatabaseMetadata` instance passed to it.

- For example the `ColumnResource` class is responsible for extracting all metadata related to Columns in the Database, the `TableResource` class is responsible for extracting all metadata related to Tables in the Database.

3.3.2.2.2.3 Recommended Usage

Instantiate the appropriate subclass of `DatabaseResource` by passing in an instance of a `DatabaseMetaData` object from the database that contains the Resource (Table, Column etc.), and use the inherited methods of the `Resource` class to obtain the metadata in RDF/XML serialized form.

3.4 The Relational Database Wrapper Module

3.4.1 Features.

The Relational Database Wrapper Module passes through the following states:

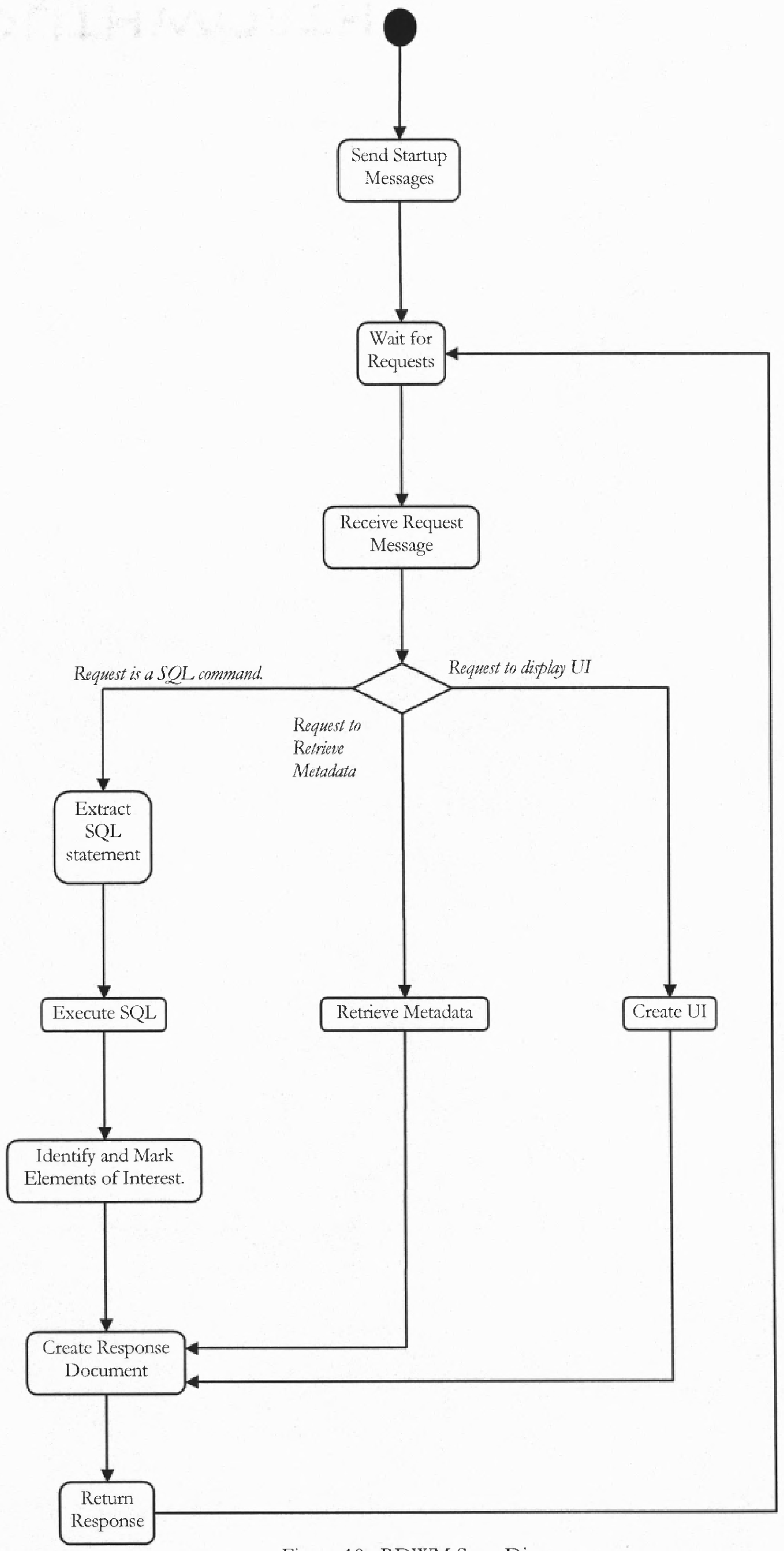


Figure 10 : RDWM State Diagram

3.4.1.1 Sending Startup Messages.

Because of the DHE's distributed nature, all modules must register themselves with the DHE Message Manager at startup time, so that the Message Manager knows which modules are currently active and able to receive and process requests.

3.4.1.1.1 Inputs.

3.4.1.1.1.1 PARAMETERS

- **The Module-ID**. This identifies the module and hence must be unique for every instance of this (and all other) module(s). The value of this is dependent on the Message Manager's mechanism of registration. The Message Manager assigns a unique Module ID whenever any module tries to register itself. This module then proceeds to use this value as the remote object identifier when it binds itself to the local RMI Registry.

- **IP address.** Identifies the machine the RDWM is running on. This is required so that the Message Manager is able to perform an RMI "lookup" on this module.

3.4.1.1.2 Processing

Like all DHE modules the RDWM extends the `Module` class in the `dhyme.gateway.module` package. It thus inherits the `startMe` method. An invocation of this method with the URL of the Message Manager module sends the required Startup Messages.

The `startMe` method automatically discovers the IP address of the RDWM and sends this information to the Message Manager.

3.4.1.1.3 Outputs

Registration is essential for any DHE module to send and receive messages. Thus if registration fails, the RDWM exits with a message informing the user that registration has failed.

3.4.1.2 Receiving Request Messages

A module waits until it receives a request to perform a service. The services the RDWM provides are:

- Interact with the RDBMS.
- Retrieve metadata for an element
- Generate a user input form to enter SQL statements.

Like all DHE modules the RDWM receives requests by an invocation of `rcvMsg` a remote method it inherits from the `Module` class. The argument to this method is the Request Message itself.

The `rcvMsg` method goes on to call the method `processMsg`, which is an abstract method in the `Module` class. Like all DHE modules the RDWM provides an implementation of this method.

3.4.1.2.1 Inputs

An XML Message. This XML Message must contain the following two parameters:

- The source of the request, which will be used to determine the destination of the response of the RDWM.
- The action to be performed by the RDWM.

A sample Message:

```
<..prolog..>
<EngMsg>
  <RequestID>967145925626</RequestID>
  <MsgType>FOLLOWLINK</MsgType>
  <Origin>UIW</Origin>
  <Current>BLEM</Current>
  <MsgNo>UIW:967145925675</MsgNo>
  <Destination>RDWM</Destination>
  <User_name>guest:guest:Guest User</User_name>
  <MsgBody>
    <Subject>select</Subject>
....... other elements
```

```
  </MsgBody>
</EngMsg>
```

The source of the message is defined in the element `Origin`. The action to be performed by the RDWM is defined in the element `Subject`.

The values of `Subject` is a command to execute a SQL statement, and will have one of the following values:

`Select`	Execute a select statement
`Insert`	Execute an insert statement.
`Update`	Execute an update statement.
`Delete`	Execute a delete statement.
`Alter`	DDL (Data Definition Language) command
`Create`	DDL command

A sample message to execute a SQL Select statement.

```
……. . other elements
<MsgBody>
    <Subject>select</Subject>
    <BLid></BLid>
    <Parameter>
      <ParamName>Submit</ParamName>
      <ParamValue>Execute Query</ParamValue>
    </Parameter>
    <Parameter>
      <ParamName>query</ParamName>
      <ParamValue>select * from publishers</ParamValue>
    </Parameter>
  </MsgBody>
```

All the above values of `Subject` require that a `Parameter` tag and a `ParamName` tag having the value `query` exist, and that it have an associated non-null `ParamValue` tag; which is the SQL statement that will be executed by the RDWM.

The value of the Subject tag may also be a command to retrieve the Metadata associated with a given Database Element (As defined in Section 3.3.2)

`getMetaInfo`	Gets the metadata associated with a given RDBMS element.

A sample `getMetaInfo` Message:

```
……. . other elements
<MsgBody>
    <Subject>getMetaInfo</Subject>
    <BLid></BLid>
    <Parameter>
      <ParamName>Type</ParamName>
      <ParamValue>dhyme:rdwm:value</ParamValue>
    </Parameter>
    <Parameter>
      <ParamName>Element_ID</ParamName>

<ParamValue>dhyme:rdwm:value:jdbc:oracle:thin:@logic.njit.edu:1521:logi
c40:booksauthors:ISBN:1-56-884454-9</ParamValue>
    </Parameter>
  </MsgBody>
```

For a `getMetaInfo` message, two parameter tags are required.

- One Parameter tag must have a `ParamName` with the value `Type` and an associated non-null `ParamValue`, which contains the type of the element whose metadata is being requested.

- The other Parameter tag must have a `ParamName` with the value `Element_ID` and an associated non-null `ParamValue` that contains the URI of the Element whose metadata is being requested.

Two other values of `subject`, which do not perform any action on the underlying RDBMS, are also supported.

`List`	Lists all the values of `subject` that are supported by the RDWM.
`Display`	Displays a User Interface to enter a Data Manipulation Statement.

3.4.1.3 Extracting and Executing SQL statements

Like all DHE modules the RDWM uses an XML parser to extract the action to be performed and SQL statement (if any) from the request message. If the request is to execute an SQL statement then the UI Wrapper will have embedded the actual SQL in a request message. (This is its default procedure for user input forms.) The RDWM then uses a persistent pool of JDBC (Java Database Connectivity) Connection Objects, to execute a SQL statement on, or retrieve metadata from the RDBMS.

3.4.1.3.1 Input

An XML message containing the SQL statement to be executed. As mentioned in section 3.4.1.2.1 there are 6 cases the RDWM may receive a message containing a SQL statement to be executed, and in each case the SQL statement is in the `Parameter` tag of the message.

3.4.1.3.2 Processing

The Java Programming Environment provides an API for performing RDBMS related tasks. This API, known as the Java Database Connectivity (JDBC) API, creates a programming-level interface for communicating with all relational databases in a uniform manner similar in concept to Microsoft's Open Database Connectivity (ODBC). The JDBC API is based on the X/Open SQL Call Level Interface, the same basis as that of ODBC. The core JDBC API consists of three interfaces: `Connection`, `Statement`, and `ResultSet`. A typical operation would establish a connection with a database get an instance of a `Connection`, then execute a query against that database using a `Statement` object, which would then return a `ResultSet`.

3.4.1.3.2.1 Connection Overhead

However the overhead time for establishing a database connection is typically around 1-3 seconds. This is the time it takes to locate the database server, establish a connection with it, and exchange login information. For applications where the database query times are large, this overhead is probably too small a fraction of the overall turn-around-time to be a critical issue. However when the application has to perform numerous short-term queries, database connection overhead can become a serious issue.

3.4.1.3.2.2 Connection Pool Oriented Architecture

The solution to this problem is to create a pool of persistent (reusable) database connections to be used by various application components as needed. This pool of connections is created and managed by a separate process or thread, the **connection broker**. An application component requests the broker to hand it a database connection. In addition, the broker manages the pool of connections, watches for locked or corrupted connections, logs events and performs other housekeeping tasks. Once the application component has completed its database request, the connection must be returned to the pool for reuse.

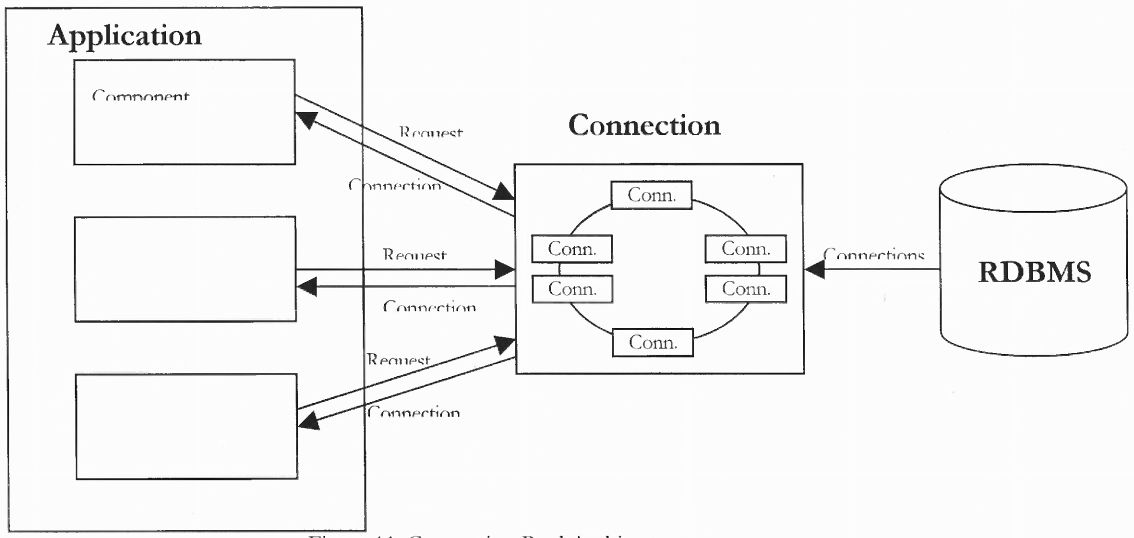


Figure 11: Connection Pool Architecture

3.4.1.3.2.3 Connection Pool Usage

The connection broker is in the class `DBBroker` in the package `dhyme.utils.dbBroker`. The connection broker is a *singleton*, i.e. only one instance of it ever exists in the life of the application/VM. When the application initializes, it must initialize the broker. Different application components then proceed to request the broker for a connection and once they are done with it they return this connection back to the broker.

- During Application Initialization:

  ```
  DBBroker.init(database parameters);
  ```

- When a Component needs a Connection:

  Connection conn = DBBroker.getInstance().getConnection();

  ……………do something with the connection

  DBBroker.`getInstance().returnConnection(conn);`

3.4.1.3.2.4 Broker Architecture

3.4.1.3.2.4.1 REQUESTING AND RETURNING CONNECTIONS

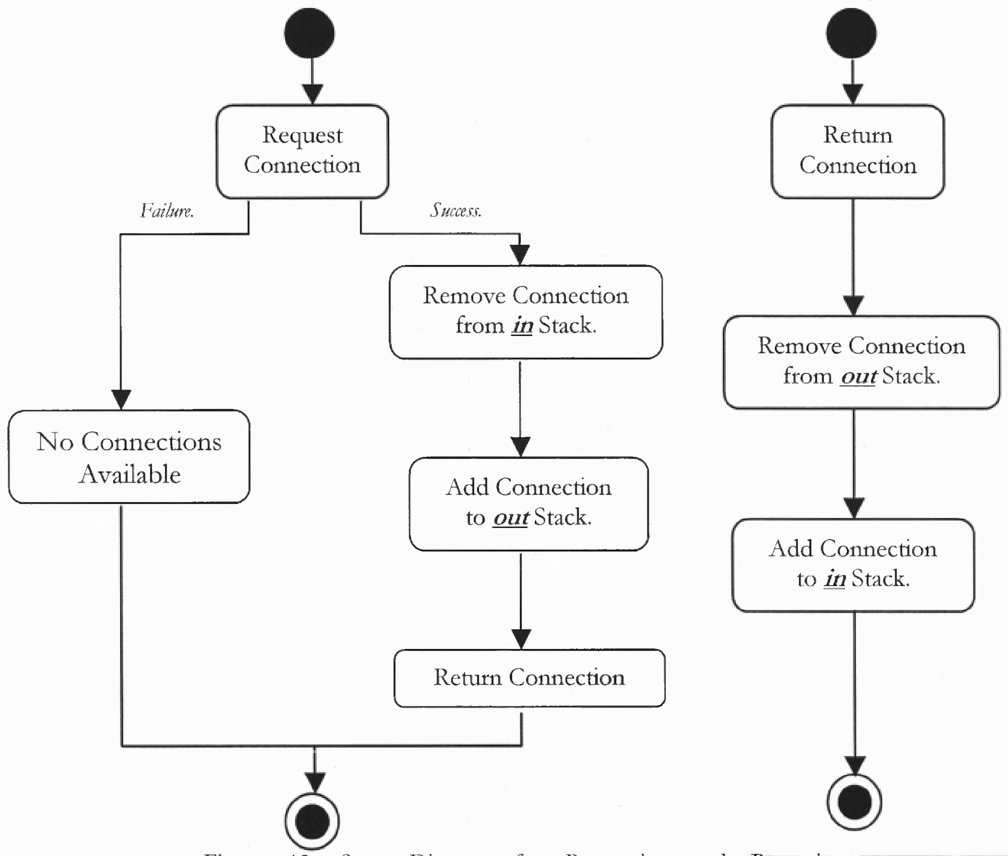

Request Connection
Failure.
Success.
Remove Connection from *in* Stack.
No Connections Available
Add Connection to *out* Stack.
Return Connection
Return Connection
Remove Connection from *out* Stack.
Add Connection to *in* Stack.

Figure 12: State Diagram for Requesting and Returning Connections.

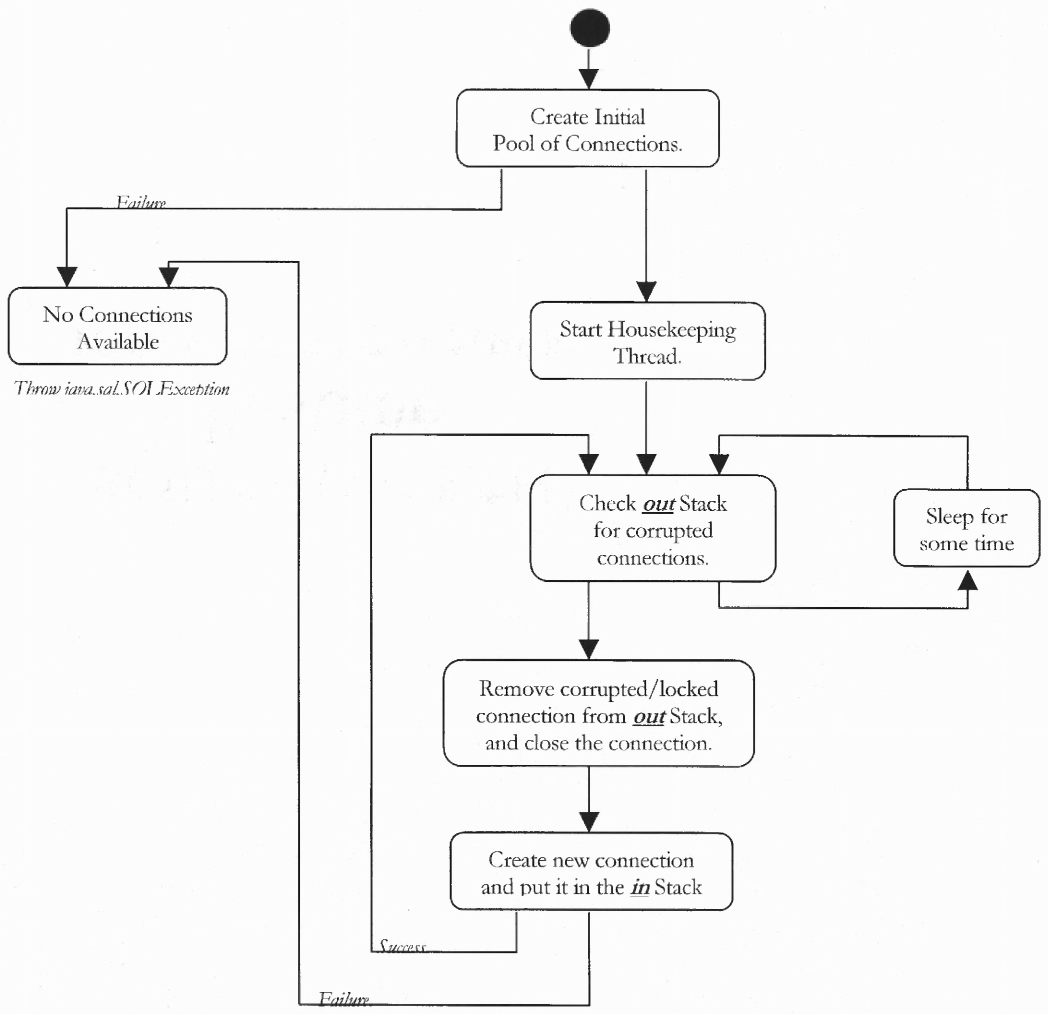
The Broker uses two Stacks to maintain the connections, the ***in*** Stack represents those connections not yet handed out to an application component. The ***out*** Stack represents those connections currently being used by various application components.

When a component requests a connection, and if one is not available (i.e. not in the ***in*** Stack), instead of returning immediately (via a `java.sql.SQLException`), the broker waits a while and then looks again to see if a Connection is available. This process is repeated a number of times. The amount of time to wait for and the number of times this process will be repeated are parameters and are passed into the broker during initialization.

3.4.1.3.2.4.2 BROKER HOUSEKEEPING

It is possible that after the connection has been handed out to application components, it may have become corrupted during use and the application may not be able to return it back to the pool. In such cases the broker should be able to remove that connection from the pool, and replace with a new uncorrupted connection.

To manage this housekeeping operation a `PoolManager` has been developed.


Create Initial Pool of Connections.
Failure
No Connections Available
Throw java.sql.SQLException
Start Housekeeping Thread.
Check ***out*** Stack for corrupted connections.
Sleep for some time
Remove corrupted/locked connection from ***out*** Stack, and close the connection.
Create new connection and put it in the ***in*** Stack
Success
Failure
Figure 13: Broker Housekeeping State Diagram

Deciding whether or not a connection is corrupted/locked is however not a trivial matter. A rudimentary way of deciding is to keep count of the length of time the connection has been in the ***out*** Stack. If this amount of time is extraordinarily "long" (i.e. greater than a max amount, passed in to the broker during initialization) and there are `SQLWarnings` for this connection, that would mean this connection is corrupted/locked, and hence should be closed immediately. A side effect of this is that the application component utilizing this connection would get a `java.sql.SQLException` when it tries to execute queries or access a `ResultSet`, because the underlying resources would be closed.

3.4.1.3.2.4.3 OBJECT ORIENTED ARCHITECTURE

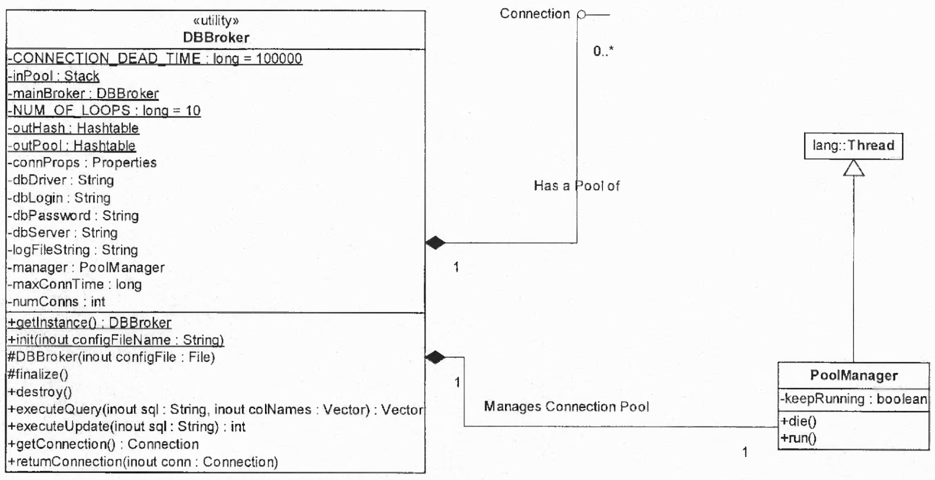


Figure 14: Class Diagram of the `dhyme.utils.dbBroker.DBBroker` Class.

3.4.1.4 Identifying an Element of Interest

The Mapping Rules Module maps elements in the RDBMS domain into the hypertext environment. In order to perform this mapping, the Mapping Rules Module needs to identify which elements have any Mapping Rules or metadata associated with it. The RDWM (and all DMISWs) thus has to "mark" those elements.

As mentioned in section 3.2.1, there are 12 kinds of elements in the RDWM context and any instance of that is marked up.

3.4.1.4.1 Input

The `ResultSet` from a query.

3.4.1.4.2 Processing

A `ResultSetMarker` class has been developed that iterates through the records returned in a `ResultSet`, formats those records into an HTML table for display and marks up each element with a `locator` tag. This HTML table is written into a `DisplayDoc` message for display in the UIW.

3.4.1.4.3 Output

The Response as an XML message with all the required elements marked up with `locator` tags, and the types and the URIs of these elements in the `ListOfElements` tag.

3.4.1.4.4 Format of Respons

A `DisplayDoc` message.

3.4.1.5 Returning the Response

All DHE Modules extend the Module class and thus inherit the `sendMsgToModule` method. To send a message to any other DHE module, the source module must specify the ultimate destination of the module in the message, and call that method with the message as the argument.

3.4.2 Object Oriented Analysis

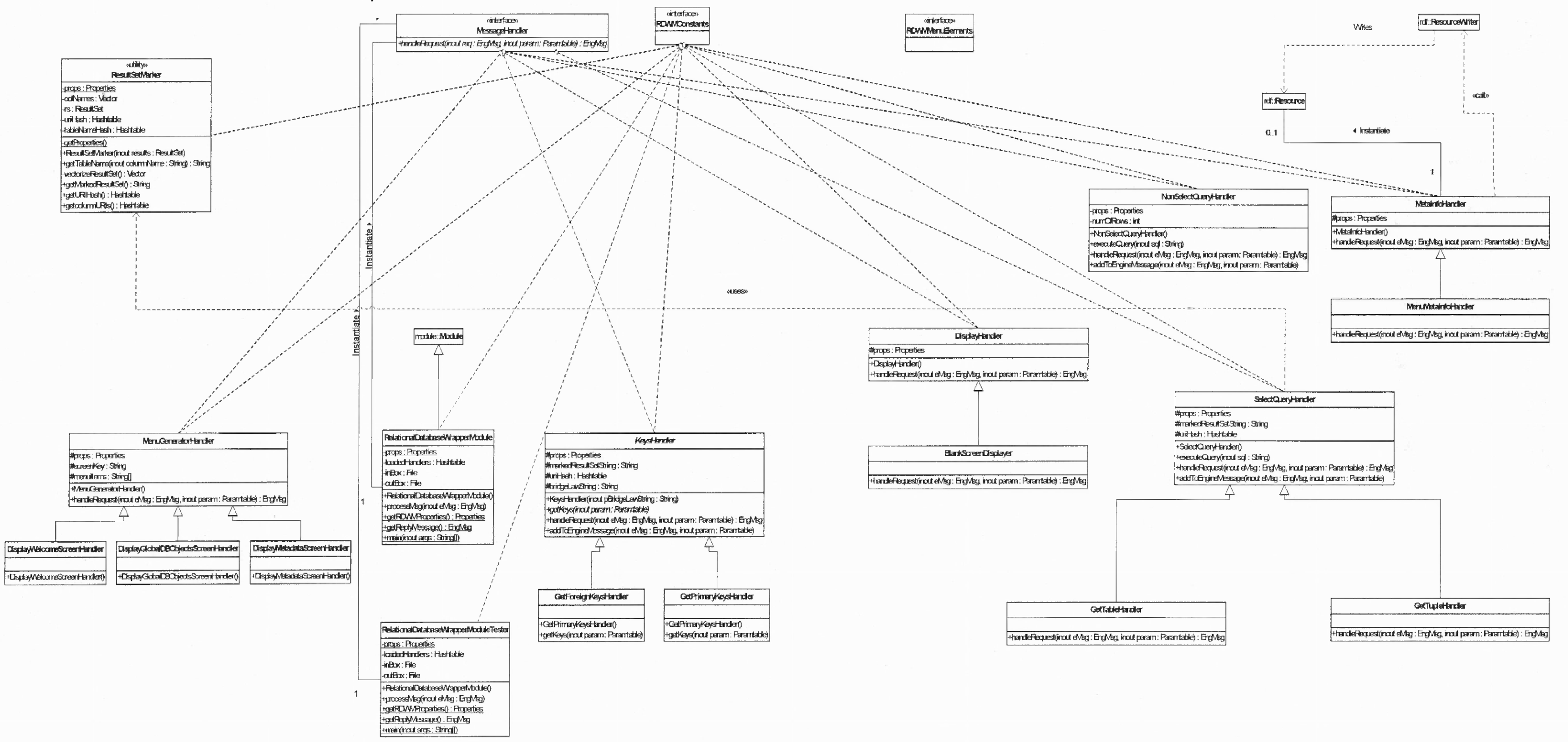

«interface»
MessageHandler
+handleRequest(inout req : EngMsg, inout param : Paramtable) : EngMsg
«interface»
RDWMConstants
«interface»
RDWMenuElements
Writes
rdf::ResourceWriter
rdf::Resource
«call»
0..1
Instantiate
1
«utility»
ResultSetMarker
-props : Properties
-colNames : Vector
-rs : ResultSet
-uriHash : Hashtable
-tableNameHash : Hashtable
-getProperties()
+ResultSetMarker(inout results : ResultSet)
+getTableName(inout columnName : String) : String
-vectorizeResultSet() : Vector
+getMarkedResultSet() : String
+getURIHash() : Hashtable
+getcolumnURIs() : Hashtable
Instantiate
Instantiate
«uses»
NonSelectQueryHandler
-props : Properties
-numOfRows : int
+NonSelectQueryHandler()
+executeQuery(inout sql : String)
+handleRequest(inout eMsg : EngMsg, inout param: Paramtable) : EngMsg
+addToEngineMessage(inout eMsg : EngMsg, inout param : Paramtable)
MetaInfoHandler
#props : Properties
+MetaInfoHandler()
+handleRequest(inout eMsg : EngMsg, inout param : Paramtable) : EngMsg
MenuMetaInfoHandler
+handleRequest(inout eMsg : EngMsg, inout param : Paramtable) : EngMsg
module::Module
DisplayHandler
#props : Properties
+DisplayHandler()
+handleRequest(inout eMsg : EngMsg, inout param : Paramtable) : EngMsg
BlankScreenDisplayer
+handleRequest(inout eMsg : EngMsg, inout param : Paramtable) : EngMsg
SelectQueryHandler
#props : Properties
#markedResultSetString : String
#uriHash : Hashtable
+SelectQueryHandler()
+executeQuery(inout sql : String)
+handleRequest(inout eMsg : EngMsg, inout param : Paramtable) : EngMsg
+addToEngineMessage(inout eMsg : EngMsg, inout param : Paramtable)
MenuGeneratorHandler
#props : Properties
#screenKey : String
#menuItems : String[]
+MenuGeneratorHandler()
+handleRequest(inout eMsg : EngMsg, inout param : Paramtable) : EngMsg
DisplayWelcomeScreenHandler
+DisplayWelcomeScreenHandler()
DisplayGlobalDBObjectsScreenHandler
+DisplayGlobalDBObjectsScreenHandler()
DisplayMetadataScreenHandler
+DisplayMetadataScreenHandler()
RelationalDatabaseWrapperModule
-props : Properties
-loadedHandlers : Hashtable
-inBox : File
-outBox : File
+RelationalDatabaseWrapperModule()
+processMsg(inout eMsg : EngMsg)
+getRDWMProperties() : Properties
+getReplyMessage() : EngMsg
+main(inout args : String[])
1
KeysHandler
#props : Properties
#markedResultSetString : String
#uriHash : Hashtable
#bridgeLawString : String
+KeysHandler(inout pBridgeLawString : String)
+getKeys(inout param: Paramtable)
+handleRequest(inout eMsg : EngMsg, inout param: Paramtable) : EngMsg
+addToEngineMessage(inout eMsg : EngMsg, inout param : Paramtable)
GetForeignKeysHandler
+GetPrimaryKeysHandler()
+getKeys(inout param: Paramtable)
GetPrimaryKeysHandler
+GetPrimaryKeysHandler()
+getKeys(inout param : Paramtable)
GetTableHandler
+handleRequest(inout eMsg : EngMsg, inout param : Paramtable) : EngMsg
GetTupleHandler
+handleRequest(inout eMsg : EngMsg, inout param : Paramtable) : EngMsg
RelationalDatabaseWrapperModuleTester
-props : Properties
-loadedHandlers : Hashtable
-inBox : File
-outBox : File
+RelationalDatabaseWrapperModule()
+processMsg(inout eMsg : EngMsg)
+getRDWMProperties() : Properties
+getReplyMessage() : EngMsg
+main(inout args : String[])
1

Figure 15: RDWM Class Diagram

3.4.3 Functionality.

3.4.3.1 Architecture.

As shown in the class diagram, the RDWM itself receives and sends messages, it parses the message and determines what command to execute. As mentioned in section 3.4.1.2.1, the command may either be to execute a query, to display a UI or to retrieve metadata. Another class of commands maybe issued as a result of a user following link created from a Mapping Rule, as detailed in section 3.3.1. These commands are also dealt with in the exact same manner.

To process this command the RDWM uses "handlers" devoted for that purpose, for example, a `DisplayHandler` is responsible for allowing a user to enter a SQL statement for execution.

The following sections detail the processing involved in each of these "handlers", by a series of Collaboration diagrams that show the method calls in them. The arrows in each diagram signifies the direction of the method call, the arguments to that method (if any) is also labeled on each arrow.

3.4.3.2 Displaying the User Interface.

This is as a result of a `display` command being received by the RDWM.

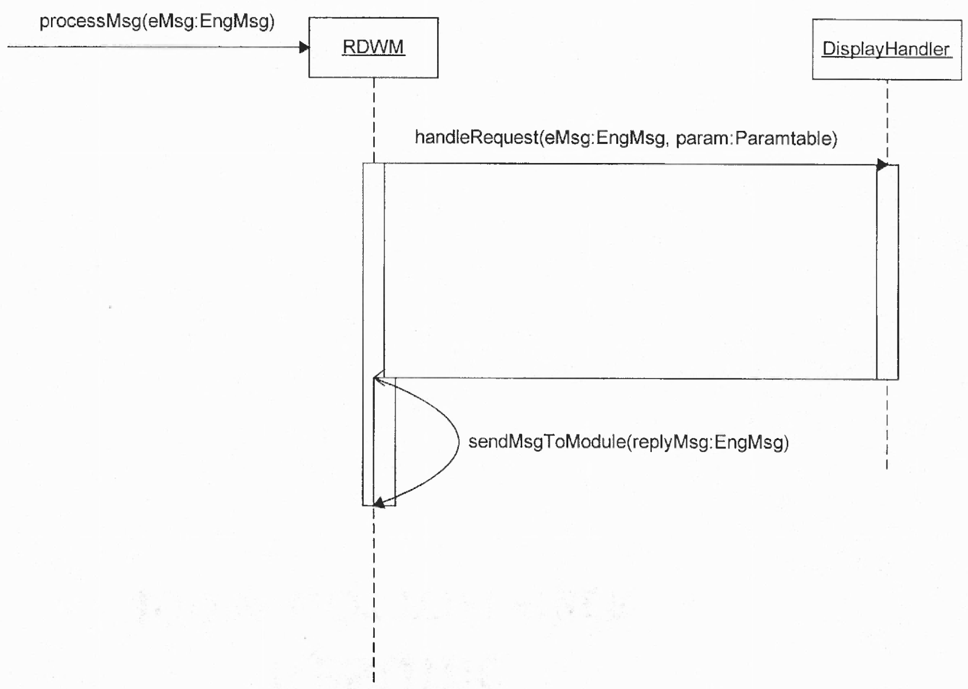

processMsg(eMsg:EngMsg)
RDWM
DisplayHandler
handleRequest(eMsg:EngMsg, param:Paramtable)
sendMsgToModule(replyMsg:EngMsg)

Figure 16: Displaying the UI - Collaboration Diagram

3.4.3.3 Executing a select statement.

This is as a result of a `select` command being received by the RDWM.

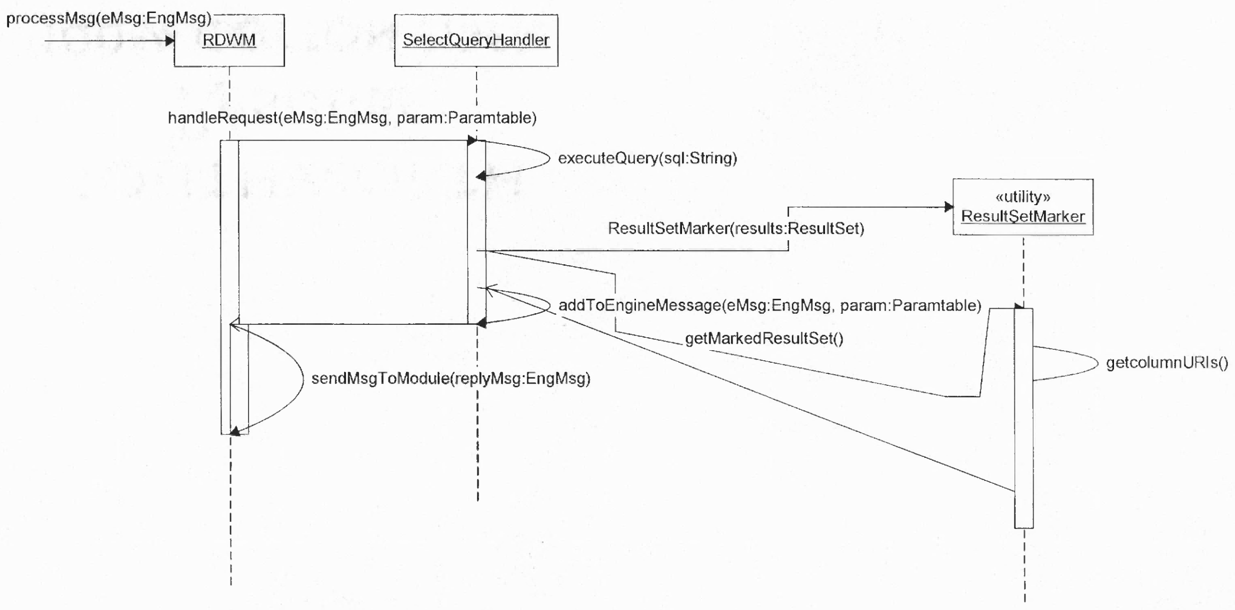

processMsg(eMsg:EngMsg)
RDWM
SelectQueryHandler
handleRequest(eMsg:EngMsg, param:Paramtable)
executeQuery(sql:String)
«utility»
ResultSetMarker
ResultSetMarker(results:ResultSet)
addToEngineMessage(eMsg:EngMsg, param:Paramtable)
getMarkedResultSet()
getcolumnURIs()
sendMsgToModule(replyMsg:EngMsg)

Figure 17: Executing a select statement - Collaboration Diagram.

3.4.3.4 Executing a non-select statement.

This is as a result of a `delete`, `insert`, `update` or `DDL` command having been issued from the UI.

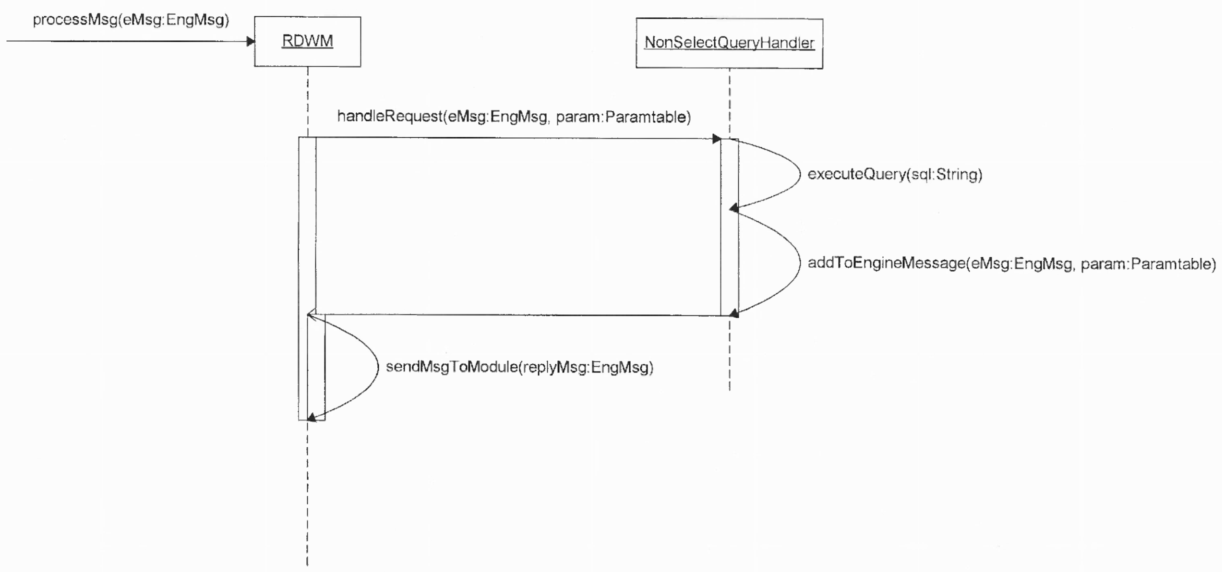

processMsg(eMsg:EngMsg)
RDWM
NonSelectQueryHandler
handleRequest(eMsg:EngMsg, param:Paramtable)
executeQuery(sql:String)
addToEngineMessage(eMsg:EngMsg, param:Paramtable)
sendMsgToModule(replyMsg:EngMsg)

Figure 18: Executing a non-select statement - Collaboration Diagram

3.4.3.5 Executing a Mapping Rule – Retrieve Primary Keys.

This is as a result of a user following through the "Get Primary Keys" Mapping Rule for a Column or Table.

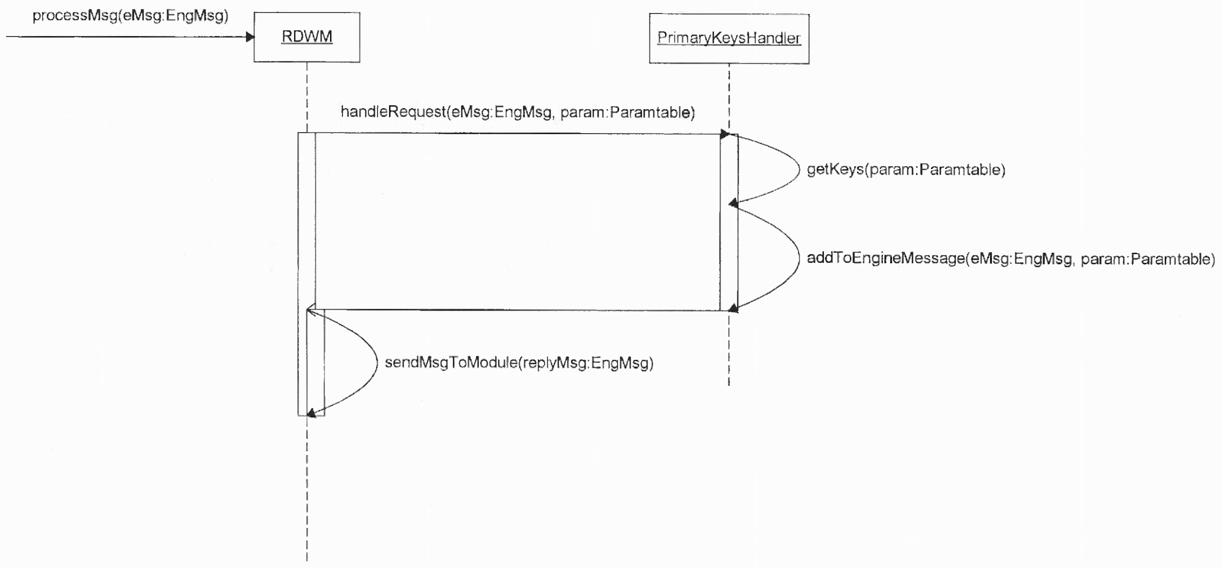

processMsg(eMsg:EngMsg)
RDWM
PrimaryKeysHandler
handleRequest(eMsg:EngMsg, param:Paramtable)
getKeys(param:Paramtable)
addToEngineMessage(eMsg:EngMsg, param:Paramtable)
sendMsgToModule(replyMsg:EngMsg)

Figure 19: Retrieving Primary Keys

3.4.3.6 Executing a Mapping Rule – Retrieve Foreign Keys.

This is as a result of a user following through the "Get Foreign Keys" Mapping Rule for a Column or Table.

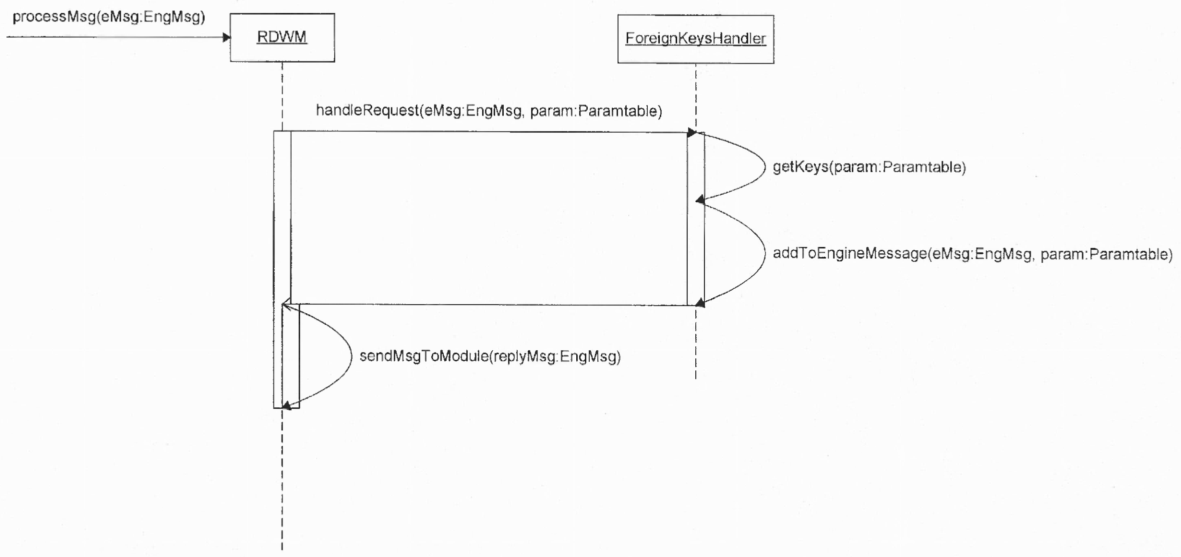

processMsg(eMsg:EngMsg)
RDWM
ForeignKeysHandler
handleRequest(eMsg:EngMsg, param:Paramtable)
getKeys(param:Paramtable)
addToEngineMessage(eMsg:EngMsg, param:Paramtable)
sendMsgToModule(replyMsg:EngMsg)

Figure 20: Retrieving Foreign Keys

3.4.3.7 Executing a Mapping Rule – Retrieve the Table a Column or a Value belongs to.

This is as a result of a user following through the "Get Table" Mapping Rule for a Column or Table.

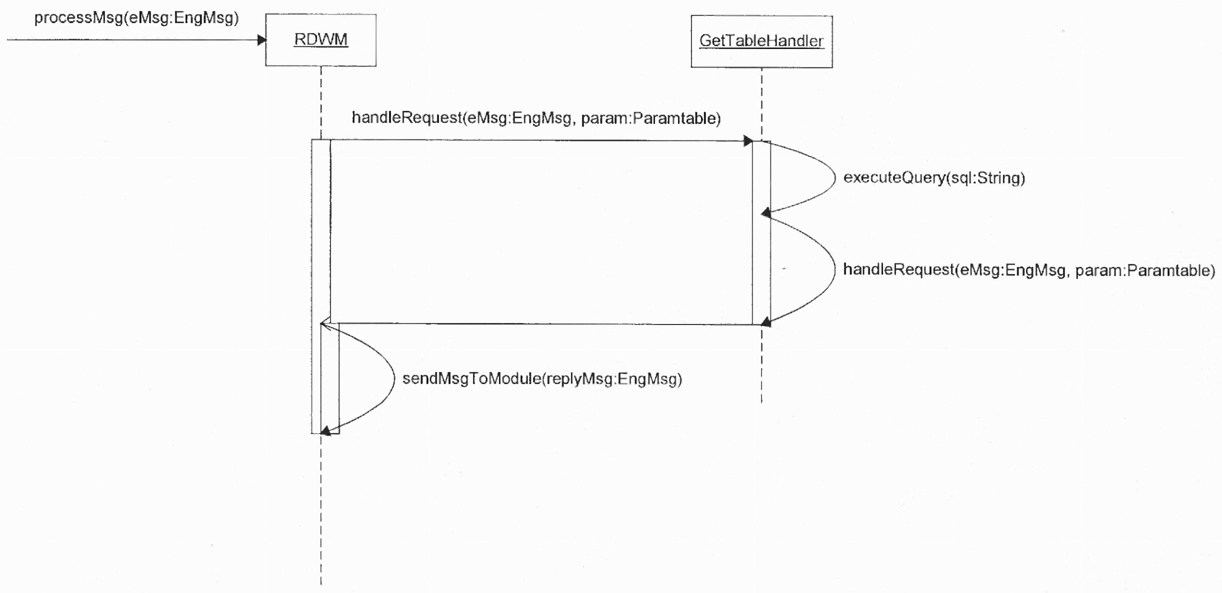

processMsg(eMsg:EngMsg)
RDWM
GetTableHandler
handleRequest(eMsg:EngMsg, param:Paramtable)
executeQuery(sql:String)
handleRequest(eMsg:EngMsg, param:Paramtable)
sendMsgToModule(replyMsg:EngMsg)

Figure 21: Retrieving a Table

3.4.3.8 Executing a Mapping Rule – Retrieve the Tuple a Value belongs to.

This is as a result of a user following through the "Get Tuple" Mapping Rule for a Database Value.

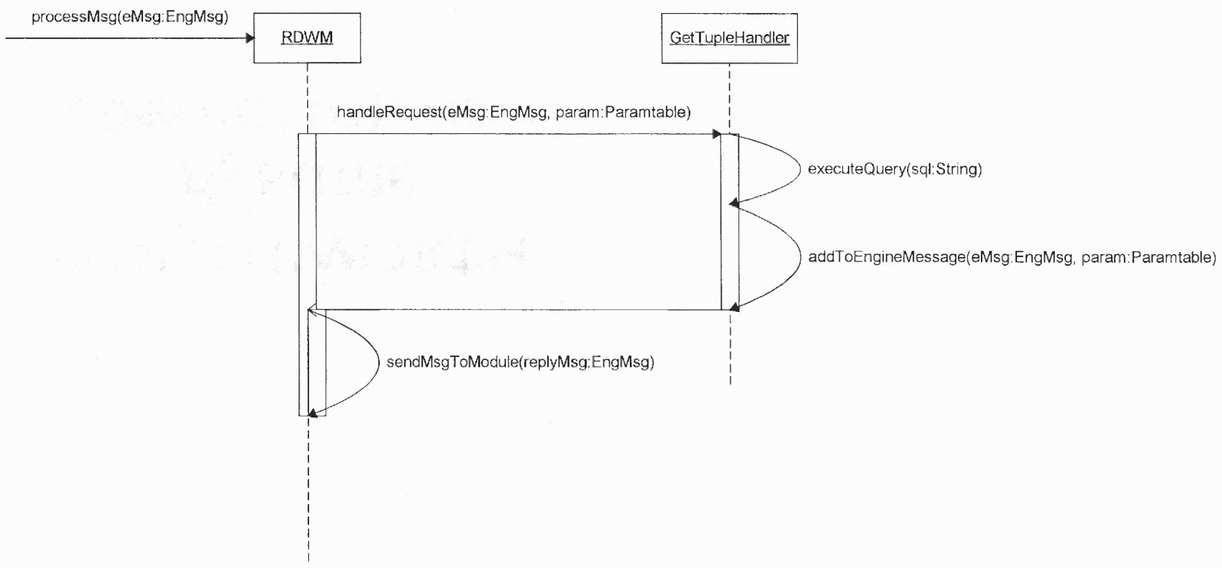

processMsg(eMsg:EngMsg)
RDWM
GetTupleHandler
handleRequest(eMsg:EngMsg, param:Paramtable)
executeQuery(sql:String)
addToEngineMessage(eMsg:EngMsg, param:Paramtable)
sendMsgToModule(replyMsg:EngMsg)

Figure 22: Retrieving a Tuple.

3.4.3.9 Retrieving Metadata.

This is as result of a user retrieving metadata for an element.

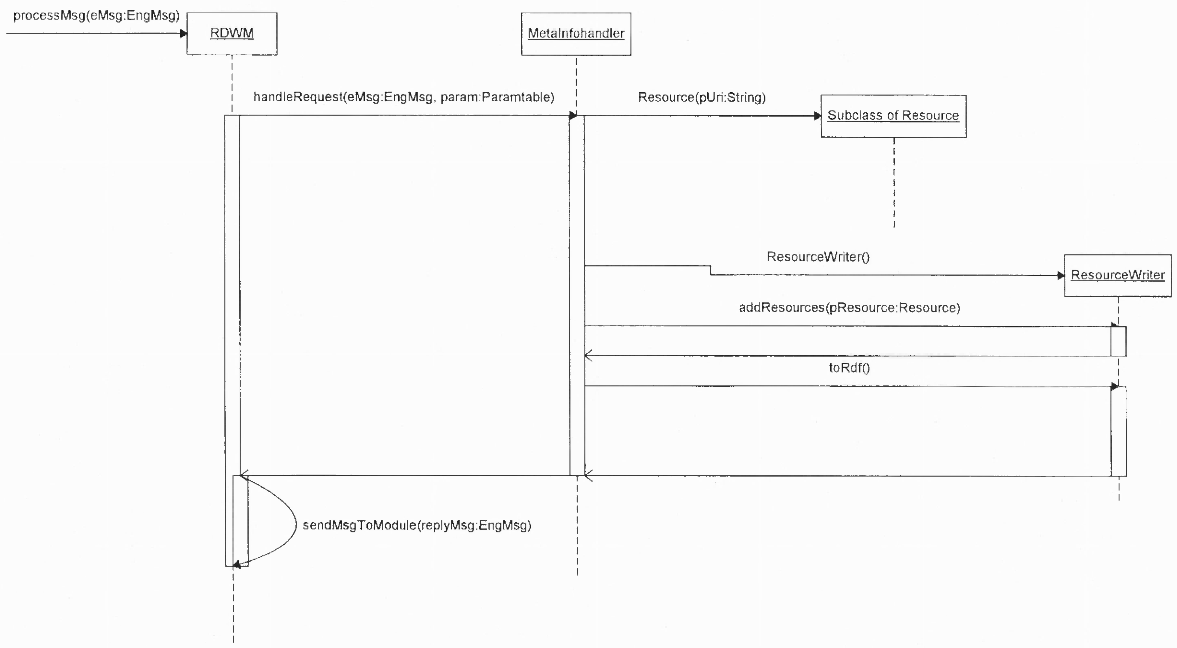
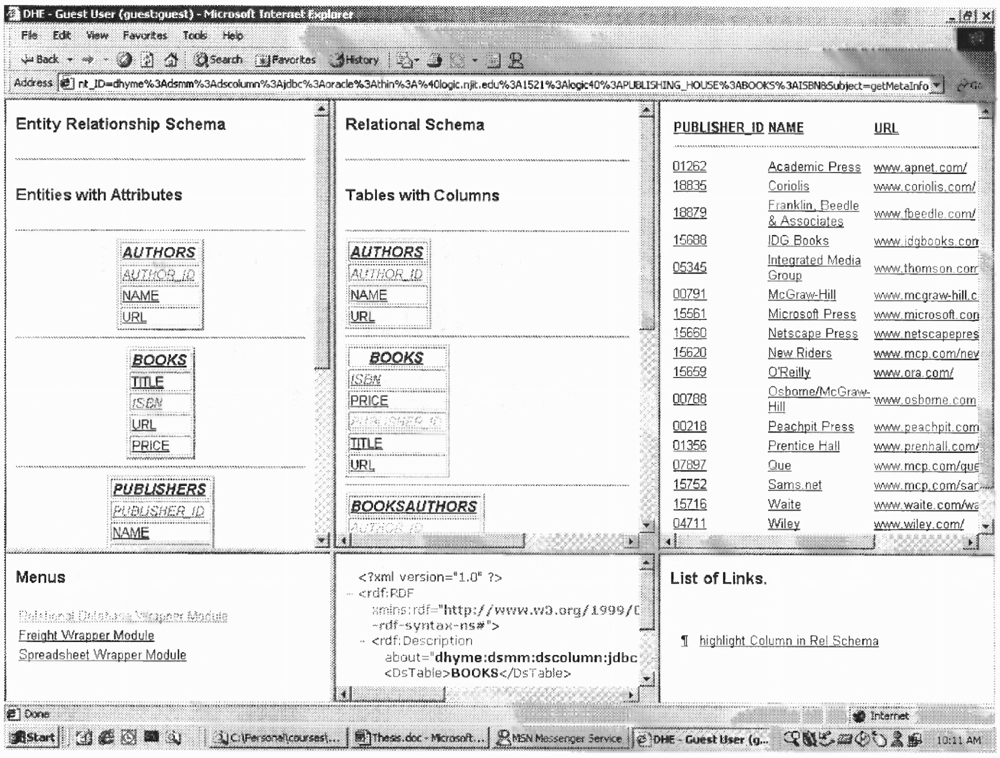


Figure 23: Retrieving Metadata

3.5 Enhanced Links through a Database Schema Wrapper

Most database applications provide no contextual information about the underlying schema of the database from which query results were retrieved. The DHE utilizes a dedicated Database Schema Mapper Module to add value to database applications by making this information explicit.



DHE - Guest User (guest:guest) - Microsoft Internet Explorer

Address: nt_ID=dhyme%3Adsmm%3Adscolumn%3Ajdbc%3Aoracle%3Athin%3A%40logic.njit.edu%3A1521%3Alogic40%3APUBLISHING_HOUSE%3ABOOKS%3AISBN&Subject=getMetaInfo

Entity Relationship Schema

Entities with Attributes

AUTHORS: *AUTHOR_ID*, NAME, URL

BOOKS: TITLE, *ISBN*, URL, PRICE

PUBLISHERS: *PUBLISHER_ID*, NAME

Relational Schema

Tables with Columns

AUTHORS: *AUTHOR_ID*, NAME, URL

BOOKS: *ISBN*, PRICE, *PUBLISHER_ID*, TITLE, URL

BOOKSAUTHORS: *AUTHOR_ID*

PUBLISHER_ID	NAME	URL
01262	Academic Press	www.apnet.com/
18835	Coriolis	www.coriolis.com/
18879	Franklin, Beedle & Associates	www.fbeedle.com/
15688	IDG Books	www.idgbooks.con
05345	Integrated Media Group	www.thomson.corr
00791	McGraw-Hill	www.mcgraw-hill.c
15561	Microsoft Press	www.microsoft.con
15660	Netscape Press	www.netscapepres
15620	New Riders	www.mcp.com/nev
15659	O'Reilly	www.ora.com/
00788	Osborne/McGraw-Hill	www.osborne.com
00218	Peachpit Press	www.peachpit.com
01356	Prentice Hall	www.prenhall.com/
07897	Que	www.mcp.com/que
15752	Sams.net	www.mcp.com/sar
15716	Waite	www.waite.com/wa
04711	Wiley	www.wiley.com/

Menus

Relational Database Wrapper Module
Freight Wrapper Module
Spreadsheet Wrapper Module

```
<?xml version="1.0" ?>
- <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/0
    -rdf-syntax-ns#">
  - <rdf:Description
      about="dhyme:dsmm:dscolumn:jdbc
      <DsTable>BOOKS</DsTable>
```

List of Links.

¶ highlight Column in Rel Schema

Figure 24 : Screenshot of the DSMM

As shown above the three frames of the Database Schema Wrapper will subdivide the main frame of the DHE. The leftmost frame contains the database query results as before. The middle frame shows the relational (physical) schema behind the query results. The rightmost frame shows the original, non-normalized entity-relationship schema corresponding to the query results.

The DB Schema Mapper (DSM) runs in conjunction with the Relational Database Wrapper Module. As mentioned in section 3.3.2, the RDWM provides metadata by examining the physical schema and returns attributes such as names of columns, data types etc. To this metadata the DSM adds schematic information for the values retrieved from a database query.

The schematic information about a particular database has to be entered through a user interface by a system developer or administrator at the time the system is being designed or integrated with DHE. At runtime when a query has been issued, the DSM checks its internal database to see if the schematic information is available. If it is, then it allows the user the ability to view the underlying E-R schema as well as the relational schema of the database from which the query result was retrieved.

The DSM receives messages either from the RDWM or from the User Interface Wrapper. The RDWM passes query result sets through the DSM to add schematic information. The DSM parses the original query to get the table name. It then queries its own internal database to see if it contains schema information for that table. If so, the DSM generates the E-R and relational schemas, marks whichever elements of interest each contains, and sends a message to the UI Wrapper to display these together with the regular query results.

The UI Wrapper sends the DSM messages when a user follows a DSM-related link. For example, when the user selects a table and chooses a link to highlight that table in the relational schema. The mapping rule corresponding to that link sends the appropriate command to the DSM. The DSM must follow its internal mapping from the RDWM URI scheme to the DSM URI scheme to identify the corresponding table element in the relational schema. Then the DSM creates a new display where it indicates that the UI Wrapper should highlight certain elements.

When parsing the RDWM's query results, the DSM marks the following as elements of interest in the corresponding E-R schema, and includes the properties shown as parameters:

1. E-R Database Schema
    - Name (of the database in the DSM internal database)
2. Entities.
    - Name.
    - Type (Weak Entity or Normal Entity).
3. Relationship.
    - Name

- Type (identifying relationship or weak entity relationship)

4. Attributes

    - Name

    - Type:

        1. Composite or Simple

        2. Multi-valued or Single-valued

        3. Stored or Derived

        4. Key or not

The mapping rules capture the inter-relationships amongst each of these elements. Given any particular database, entity, relationship or attribute in the E-R schema, the DSM will find the corresponding database, entities, relationships and attributes related to it in the E-R schema. Additional mapping rules will find the corresponding elements in the relational schema and the query result currently being displayed.

When parsing the RDWM's query results, the DSM marks the following as elements of interest in the corresponding relational schema, and includes the properties shown as parameters, with corresponding relationships:

1. Relational Database Schema

    - Name

2. Relation

    - Name

3. Attribute

- Primary Key
- Foreign key

4. Referential Integrity Costraint

- Relation name where the foreign key resides
- Relation name where the foreign key references
- Attribute name of the foreign key in the residing relation
    - Attribute name of the primary key of the referenced table.

CHAPTER 4

PROVIDING SUPPORT TO DATABASE APPLICATIONS

One of the roles Relational Database Wrapper Module (RDWM) as envisioned in the original architecture was to provide hypermedia enriched database services to existing as well as completely new Database Driven Applications.

4.1 Support to existing Database Applications

Developers can retrofit existing database applications to work with DHE. This section begins by describing such an integration. Then we describe the different kinds of links and metadata that DHE provides and conclude with a brief description of a system we have integrated with the DHE.

4.1.1 Motivation

Database applications that need to provide a hypertext-enriched view of data interact with the RDBMS via the RDWM. Such a view enables the user or application and database designers to view the underlying relationships that are not easily perceived, moreover by following links and retrieving metadata they gain an insight on the domain that the application serves.

4.1.2 Architecture

One of the first tasks in providing hypermedia support is to intercept messages between the computational and user interface (UI) portions of the application (Bieber and Kacmar 1995). In the case of a Relational Database Management System (RDBMS), the application comprises only a computational portion. The RDBMS provides a standard way of requesting computational services (i.e., store, retrieve and analyze data) by means of Structured Query Language (SQL) statements. The RDWM does this and provides its own interface for entering SQL queries and displaying their results.

Database applications build a customized interface and possibly a larger set of functionality around a RDBMS. Database applications, therefore, are responsible for their own UI. Database applications send SQL statements to their RDBMS. The RDBMS then executes

these statements and returns the results of these statements back to the application, which then customizes and displays them.

If the UI displays are easy to parse, i.e., a developer could easily figure out which elements are in each display screen and how to pass back database and application commands to the application, then one could write an application wrapper that intercepts all displays and redirects them to DHE. This would satisfy our goal of providing automated hypermedia functionality with minimal change to the application. The DHE's UI would then display the enhanced screens with hypermedia anchors as shown in Figure 1.

In general, Web-based database applications (Internet storefronts, catalogs etc.) have a clear distinction between their UI and the database; usually the UI uses a middleware to communicate with the database. The DHE may supplement or replace this middleware. However most legacy database applications (client server or mainframe based) usually don't have such a clear distinction between their UI and the underlying database; most often the data and presentation is commingled. Integration with the DHE could be much more difficult in this case.

In the DHE architecture each database application would be a DMIS, which would have its corresponding DMIS Wrapper (DMISW). There are two integration points with an existing Database Application.

4.1.2.1When the DMIS needs database services

This would be similar to supporting new database applications. The DMIS would route all database requests to the wrapper, which would pass it onto the rest of the Engine. Once the database request has been serviced, a hypermedia-enriched result of the database request would be provided to the DMIS, which could then use the UIW to display it. (See section 4.2 for more details)

4.1.2.2 When the DMIS interacts with its UI

All messages between the computational portion and the UI portion of the application would be intercepted by the Wrapper, which would route it to the engine to add hypermedia functionality to it and use the UIW to display the enriched UI (Chiu 1997; Bieber and Kacmar 1995). Assuming that a clear distinction exists between the computational and the interface portions of the application, and that the application allows these messages to be intercepted (Bieber and Kacmar 1995).

4.1.3 Case Study

This section describes the integration of a database application with the DHE.

4.1.3.1 Problem Description

The New Jersey Department of Transportation has an extensive database that contains commodity (coal, vegetable oil etc.) flow information between various counties in NJ and various zones in the Northeastern US. A rudimentary web interface to this database exists; we have created a DHE module that supplements this system and provides enhanced metadata, exposes the various interrelationships between the "elements of interest" in the system and provides additional functionality.

4.1.3.2 Architecture

The Freight Database Wrapper (FDW) is a DHE module that acts as the wrapper to the Freight system. Like other application wrappers described in previous sections, the FDW provides a gateway to application specific commands. Users may view the system through the DHE's User Interface and view reports on commodity flows between counties and zones, via a set of Menu Items and Mapping Rules.

The FDW may also use the RDWM to completely bypass the existing web based freight system and access the underlying database directly. This serves a twofold purpose:

- Metadata that is currently not provided by the system can be extracted directly from the database by means of SQL statements executed by the RDWM.

- New mapping rules maybe formulated, these mapping rules would correspond to SQL statements that would be executed on the freight database, thus providing additional functionality not available in the existing system.

The FDW may also be used in conjunction with the DSM, providing users with a view of the internal schema of the freight system's database.

4.2 Support to Applications being developed

This section describes how database applications could be developed quickly and easily if designed to take advantage of the DHE's infrastructure. A system that is being built using the DHE is also analyzed.

4.2.1 Motivation

In this role the DHE will provide access to a relational database for applications that need it. All requests to a database i.e., SQL statements that need to be executed on a database will be routed to the RDWM, which will execute the statement and return the hypermedia enriched results to the application.

This will allow any application developer to quickly develop a database driven application that is already enriched with hypermedia. The application developer will be able to use hypertext functionality (HTF) to visualize relationships between entities in the application's domain, augment applications with, annotation and navigation functionality and other metadata that would normally be hidden inside the Relational Database. (Bieber and Vitali 1997)

4.2.1.1 Scalable Application

Most database applications are tied to a certain database schema, which must be known when the application is being designed. However by retrieving metadata and relationships (as a "list of links") it is possible to create a generic database application that dynamically generates a UI corresponding to a database schema at runtime.

Thus instead of building a customized database application such as "Inventory" or "Purchase Order Tracking" where the UI and screen flow must be decided at the application design stage, it may be possible to build an application that utilizes the "list of links" to create a screen flow and metadata to generate the UI.

Because the application is not tied to a specific database schema, it is naturally scalable. Any time the Data Model or underlying "Business Rule" changes and a change has to be made to the schema, the application itself does not have to be modified, a change to an existing Mapping Rule or addition of a new one will suffice. This is in stark contrast to conventional application where a costly reengineering effort must be undertaken.

4.2.2 Architecture

To integrate with DHE, an application normally routes all database access requests to its application wrapper. In this case, because the application is being developed from the ground up, this wrapper maybe a part of the application itself. The wrapper (or wrapper portion of the application) would pass a DHE-formatted XML message to the RDWM to perform any database services requested by the application— usually the execution of a SQL statement or retrieval of metadata from the RDBMS being wrapped by the RDWM.

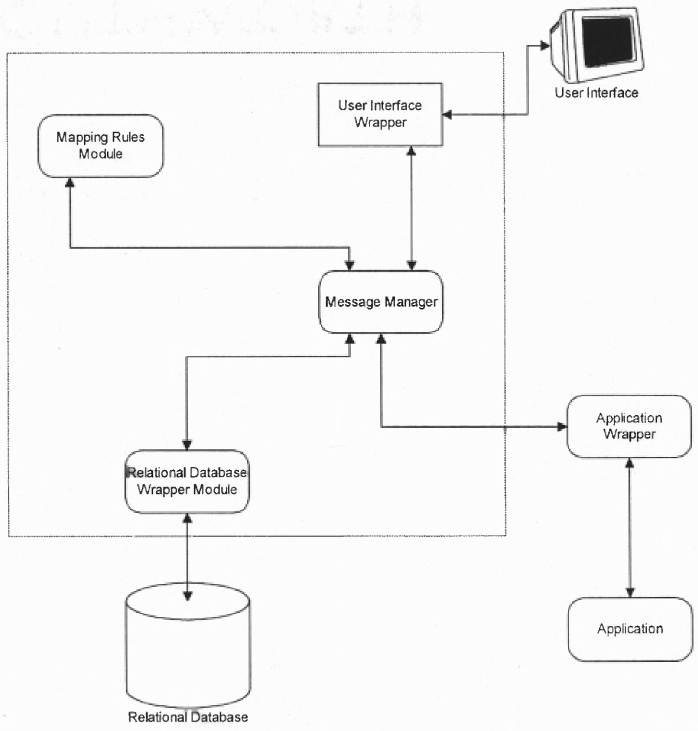


Figure 25: Providing Support to New Database Applications – Architecture

RDWM would still be responsible for marking up any query results, passing the hypermedia-enriched document is then routed back to the application wrapper.

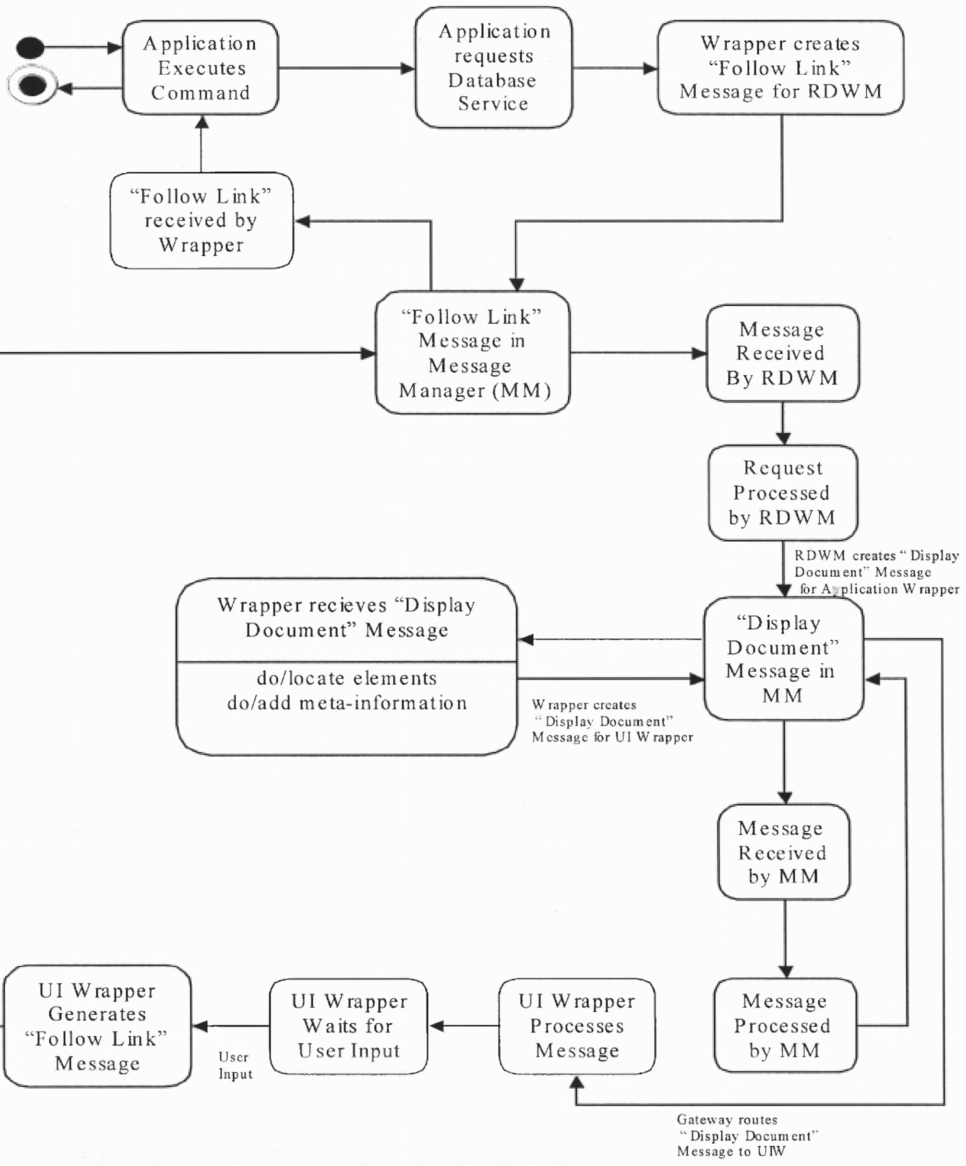

Application Executes Command
Application requests Database Service
Wrapper creates "Follow Link" Message for RDWM
"Follow Link" received by Wrapper
"Follow Link" Message in Message Manager (MM)
Message Received By RDWM
Request Processed by RDWM
RDWM creates "Display Document" Message for Application Wrapper
Wrapper recieves "Display Document" Message
do/locate elements
do/add meta-information
"Display Document" Message in MM
Wrapper creates "Display Document" Message for UI Wrapper
Message Received by MM
Message Processed by MM
UI Wrapper Processes Message
UI Wrapper Waits for User Input
UI Wrapper Generates "Follow Link" Message
User Input
Gateway routes "Display Document" Message to UIW

Figure 26: State Diagram

At this point the Application Wrapper may take the enriched document and translate it to the native application's native User Interface, and display it there. However, should the application developer decide to use DHE's user interface instead of writing its own, then the Application Wrapper could then add any additional content and pass the final display document to the Mapping Rules Module via the Message Manager. If it adds additional, non-database elements, then the application wrapper should mark these up too, so they also may be made into anchors. Each of these additional elements types will require its own mapping rules for determining links and metadata. Many application commands can be moved into the link's mapping rules.

The application could also take advantage of other DHE services, such as the menu manager for displaying application specific menus and the user preference manager for managing users' sessions, login, profiles etc., not to mention the other hypermedia functionality that all applications receive.

4.2.3 Case Study

This section describes a proposal to develop a student paper review information system using the DHE.

4.2.3.1 Problem Description

As part of their coursework students must review a published paper, typically students would email the instructor the bibliographic reference to the article they would like to review and the instructor would approve or reject the request. If rejected the student would have to resubmit his/her request with a new article. If approved the student would review the article and email it to the instructor. The instructor would then grade the review and post it on the class message board. Because of the volume of email an instructor receives it becomes difficult for him/her to quickly approve or reject a request, because (s)he must go through previously approved requests and make sure that this article has not been approved for review by another student.

To ameliorate this situation we propose to develop a DHE module that would allow a student to request approval, check the status of the request, post a link to the review once completed and to check his/her grade, once the review is corrected. The instructor would be able to view pending requests, approve a request, view submitted reviews, assign grades, view reports on approved articles, submitted reviews etc. The instructor would also be able to perform administrative tasks such as adding students, deleting old reviews etc.

4.2.3.2 Architecture

The DHE only allows authenticated users access to the various modules (RDWM, etc.), if a user has not logged in, he/she is presented with a login screen, where a valid DHE username and password must be entered. At the start of semester, the instructor creates user accounts for all students. When a student logs in he/she presented with a menu, customized to the privileges that their user group has, in this case the student may have access only to this application and is not able to "play around" with other DHE modules.

Thus the paper review application module does not have to manage customized menus or user authentication issues; the DHE provides this service. Once a user has been authenticated any subsequent requests (i.e. view grades, approve a review request etc.) will always contain the username and the group the user belongs to. The application can thus perform or reject the action based on the privileges the user may have. The application developer may also register customized menus based on user groups; thus users belonging to the "student" group will not see menus for administrative tasks.

The DHE would also provide database access to this application. For example when the instructor to wishes to view a report on all approved articles not yet submitted, he/she would click on the appropriate Menu Item and a message would be issued to the application module with a request to generate this report. The application module would then issue a message to the RDWM containing the appropriate SQL statement, the RDWM would execute the statement and return a document containing the results of the statement, with the "elements of interest" marked up. At this point the application module may simply forward this message to the UIW and it will be displayed in the instructor's browser with hyperlinks in them. The instructor may follow this follow hyperlink to get more metadata for this element, and view a list of mapping rules. For example if the element is an article, the metadata may be the complete bibliographic reference to the article and a link to the article itself, and a mapping rule maybe a list of students who have requested this article

4.2.4 Advantages of using the DHE's infrastructure

The DHE thus speeds up the development process by freeing the developer from having to program user management, menu management, database access and above all hyperlink creation. All the developer would have to do is:

- Register a set of mapping rules corresponding to the additional functionality desired (i.e. view a list of students who have requested a specific article, view a list of pending approval requests for a given section etc.).

- Create a set of display screens.

- Formulate a set of SQL statements corresponding to each of the main commands (i.e. retrieve grades for a student, display submitted reviews etc.)

- Write a DHE Module that receives requests for each of the main commands and sends the appropriate SQL statement to the RDWM.

CHAPTER 5

USING THE DHE AS A DATA WAREHOUSING SOLUTION

This section proposes integrating a data warehousing application within the DHE infrastructure, one of our future research topics. This would provide hypermedia support to data warehousing applications, as well as facilitate linking among data warehouses and other applications.

5.1 Introduction

A data warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management decision-making (Inmon 1996). A data warehouse is a repository of information built using data from diverse, and often departmentally isolated, application systems within an organization so this data can be modeled and analyzed by managers (Johnson 1999; Inmon 1996). Data warehouses usually are customized for a particular enterprise. Most vendors offer platforms on which enterprise data warehouses (or smaller datamarts) may be built.

5.1.1 Data Warehouse Metadata

Data warehouse architectures integrate a metadata repository that contains:

- Administrative metadata: source databases and their contents; gateway description; warehouse schema, view and derived data definitions; dimensions and hierarchies; predefined queries and reports; data mart locations and contents; data partitions; data extraction, cleaning, transformation rules, default values; data refresh and purge rules; user profiles, user groups; security

- Business metadata: business terms and definitions; ownership of data; charging policies

- Operational metadata: data lineage (history of migrated data and sequence of transformation applied); currency of data: active, archived, purged; monitoring information: warehouse usage statistics, error reports, audit trails

5.1.2 Motivation

The DHE maybe used top offer a complete end-to-end solution with the added benefit of obtaining hypertext functionality without any additional effort.

Data warehouses are typically used for on-line analytical processing (OLAP). The key structure of a data warehouse always contains some element of time and some dimension hierarchies. OLAP queries are complex. They involve grouping and aggregation. A single OLAP query can lead to several closely related queries (Chaudhuri 1997). The visualization of an OLAP query result using DHE will involve links between data from one hierarchy level to the other and links SQL subqueries contained in the OLAP query. In addition, an OLAP query can result in a large collection of data with several dimensions. In the rest of this section we concentrate on the loader module.

5.2 Functionality

The data warehouse has two broad functions:

- Accessing the data from the data warehouse.
- Loading the data from the operational systems into the data warehouse.

Like all Data Warehouses a DHE based Data Warehouse will use a DBMS as the underlying store, thus functionality of accessing data from a data warehouse will be similar to any other database application and has been dealt in previous sections.

5.2.1 The Loader Module.

To load data into the data warehouse a loader module will have to be designed and developed. The Loader Module will be like any other DHE module, and will perform the following functions:

5.2.1.1 Mapping data from Operational Systems to the Data Warehouse

The operational systems store data in their own structure, encoding etc. this has to be mapped to the data warehouse's format which is consistent across all operational systems.

5.2.1.2 Extracting Metadata from Operational Systems.

Metadata is the road map or blueprint to the data in the data warehouse, and needs to be operational. Also, metadata needs to be preserved for analysis once it has been loaded (Gardner 98). Metadata may include:

- The structure of the operational data.
- Relationships in the operational data.
- Other user-defined metadata.

5.2.1.3 Eliminating Noise

Operational data may contain data irrelevant to the warehouse (i.e., is noise), this data needs to be eliminated before loading.

5.2.1.4 Architecture

The Loader Module is supplied a template (an XSL stylesheet) that maps data from the operational systems' format to the data warehouse's format. It processes the template and maps data from the operational system to the data warehouse. Any data not specified in the template is noise and will be eliminated.

Once the extraction process is complete, the Loader Module sends a message to the RDWM containing the data to be loaded as well as the metadata. The RDWM then loads this into the data warehouse. Figure 27 describes the extraction process.

An argument could be made that a Loader Module is not required and the DHE is simply used to access data from the warehouse. However this approach would not allow the DHE to retrieve metadata from the operational systems. Moreover a complete end-to-end solution requires that we provide a Loader Module.

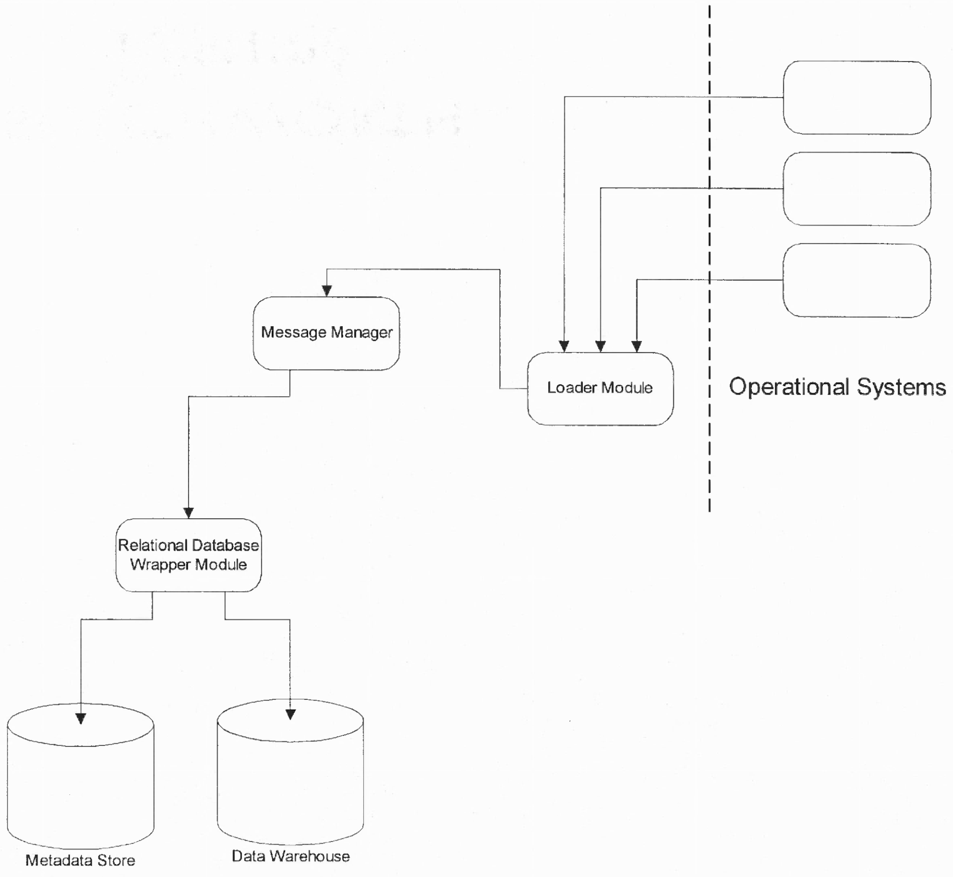


Figure 27: DHE Data Warehouse Loader Module

CHAPTER 6

FUTURE RESEARCH PLANS - INTEGRATING DATA MINING

6.1 Introduction

Currently DHE determines links from the mapping rules. Because the person who develops the application wrapper also writes the mapping rules at the same time, the types of relationships DHE finds are known ahead of time. Data mining brings the opportunity of a new kind of dynamic linking. Data mining searches large databases for relationships and global patterns and relationships that are not immediately obvious, such as a relationship between patient data and their medical diagnosis (Holsheimer 1994).

Data mining tools discover these relationships or models at runtime as opposed to design time. Thus, the DHE must request the Data Mining Tool to discover the relationships for an element of interest at runtime, and then use these discovered relationships to create hyperlinks.

Of course, in addition, the DHE could provide hypertext functionality to commercial Data Mining Tools. A commercial data mining tool would have a wrapper written for it, just as with any other application. Oracle Corporation's "Darwin"™ is one of the most well known commercial Data Mining tools. It provides a comprehensive API (in C++) to access its data mining functions, it would be easy to write a wrapper that would be able to interface with it. (Tamayo 1997), and thus seems an ideal candidate to develop a DHE wrapper for.

6.2 Mapping Rules for Data Mining

As mentioned in previous sections the DHE uses logical rules called mapping rules to provide hypertext functionality to the components of the DMIS. Mapping Rules map the objects defined in the DMIS such as models, relationships etc. to objects in the hypermedia elements such as links etc. (Bieber 1995)

These Mapping Rules need to be defined when the DMIS Wrapper is being developed which means the models, relationships that are going to be mapped should be known. Normally this is not a problem since the relationships in the various elements/nodes/entities in a DMIS are well known and determined when the DMIS is being designed.

The problem with a Data Mining Tool is that it discovers these relationships or models at runtime as opposed to design time. Which means the Mapping Rules in this case cannot be defined and entered into the Engine when the Data Mining Wrapper is being developed.

One way of solving this would be define a generic set of Mapping Rules for Data Mining, which in turn would generate other Mapping Rules dynamically (Two step Mapping Rules).

6.2.1 Two Step Mapping Rules.

For each element in a document generated by the DMIS (i.e. the Data Mining Tool), the Mapping Rules module would ask it to generate Relationships for elements of the same type.

For example, consider that the Data Mining Tool is operating on a mass of data related to Automobile Tire Replacements that contain information on the customer who returned the tire(s), the reason for return, the type of vehicle etc. all accessible via the tire's serial number, and it operates on another mass of data relating the tire to the place of manufacture, the batch number and the composition of the components used. When asked to return possible relationships for a tire it would automatically return correlations between the point of manufacture, the vehicle the automobile tire was placed on and the composition of the components. The Mapping Rules Module would then generate Mapping Rules for each of these relationships and display them as hyperlinks allowing the user to quickly and navigate these relationships and obtain an overall view of the detailed data available to him/her.

APPENDIX A

RDF RESOURCE PROPERTIES USED BY THE RDWM

Legend:

- Resource – The type of the resource the property belongs to.
- Resource Property – The name of the property, also the tag used in the RDF/XML serialized metadata representation.
- Type – The type of the Property
- Range – Range of the property if any.
- JDBC Method – The JDBC API method used to retrieve this information
- RDF:Comments – Explanatory comments on the RDF/XML tag.
- JDBC Class – The JDBC API class the JDBC Method belongs to.

Resource	Resource Property	Type	Range	JDBC Method	RDF:Comments	JDBC Class
Catalog	NAME	String		getCatalogs()	Name of Catalog.	DatabaseMetaData
Column	CHAR_OCTET_LENGTH	int		getColumns()	Maximum number of bytes in the column, valid for char types only.	DatabaseMetaData
Column	COLUMN_PRIVILEGE	Privilege		getColumnPrivileges(), getTablePrivileges()	Privileges for this Column	DatabaseMetaData
Column	DECIMAL_DIGITS	int		getColumns()	Number of Fractional Digits.	DatabaseMetaData
Column	DEFAULT	String		getColumns()	Default Value for the column.	DatabaseMetaData
Column	IN_TABLE	Table		getColumns()	Which Table this Column belongs to.	DatabaseMetaData
Column	IS_NULLABLE	String	NO, YES, UNKNOWN	getColumns()	Whether or not a column can be null or not.	DatabaseMetaData
Column	IS_PSEUDO_COLUMN	String	Unknown, NotPseudo, Pseudo	getBestRowIdentifier()	Whether or not this is a pseudo columns or not. (eg. Oracle ROWID)	DatabaseMetaData
Column	JDBC_TYPE	JDBCType		getColumns()	JDBC data type.	DatabaseMetaData
Column	NAME	String		getColumns()	Column Name	DatabaseMetaData
Column	ORDINAL_POSITION	int		getColumns()	Index of the column in a table (first column is 1)	DatabaseMetaData
Column	RADIX	int		getColumns()		DatabaseMetaData
Column	RDF:COMMENTS	RDF:Comments		getColumns()	Explanatory Comment on the Column	DatabaseMetaData
Column	RDF:LABEL	RDF:Label		getColumnLabel()	Suggested Column Title for Column	ResultSetMetaData
Column	SIZE	int		getColumns()	Column Size. For char or date types this is the max. number of characters; for numeric or decimal types this is the precision	DatabaseMetaData
Driver	IDENTIFIER_QUOTE_STRING	String		getIdentifierQuoteString()	String used to quote SQL Identifiers.	DatabaseMetaData
Driver	IS_NULL_PLUS_NON_NULL_NULL	boolean	true/false	nullPlusNonNullIsNull()	Whether concatenation of a NULL value and a non-NULL value returns a NULL value.	DatabaseMetaData
Driver	MAX_CONNECTION	int		getMaxConnections()	Maximum number of connections that can be made to the database through this driver instance.	DatabaseMetaData
Driver	NAME	String		getDriverName()	JDBC Driver Name	DatabaseMetaData
Driver	VERSION	String		getDriverVersion()	JDBC Driver Version	DatabaseMetaData

Index	CARDINALITY	int		getIndexInfo()	number of unique values in the index if TYPE is tableIndexStatistic then this is the number of rows in the table	DatabaseMetaData
Index	FILTER_CONDITION	String		getIndexInfo()	Filter condition, if any	DatabaseMetaData
Index	IN_TABLE	Table		getIndexInfo()	Which Table this Index belongs to.	DatabaseMetaData
Index	INDEX_QUALIFIER	String		getIndexInfo()	Index Catalog	DatabaseMetaData
Index	INDEX_TYPE	String	STATISTIC,CLUSTERED, HASHED, OTHER	getIndexInfo()	Type of Index	DatabaseMetaData
Index	IS_NON_UNIQUE	boolean	true/false	getIndexInfo()	Whether or not Index values can be non-unique	DatabaseMetaData
Index	NAME	String		getIndexInfo()	Name of the Index	DatabaseMetaData
Index	ORDINAL_POSITION	int		getIndexInfo()	Column sequence number within index	DatabaseMetaData
Index	PAGES	int		getIndexInfo()	When TYPE is tableIndexStatisic then this is the number of pages used for the table, otherwise it is the number of pages used for the current index	DatabaseMetaData
Index	SORT_SEQUENCE	String	ASC, DESC, NONE	getIndexInfo()	column sort sequence, "ASC" => ascending, "DESC" => descending, may be none if sort sequence is not supported	DatabaseMetaData
JDBCType	CAN_BE_AUTO_INCREMENT	boolean	true/false	getTypeInfo()	Whether or not this type can be used for an auto-increment value.	DatabaseMetaData
JDBCType	CAN_BE_MONEY_VALUE	boolean	true/false	getTypeInfo()	Whether or not this type can be a money value or not.	DatabaseMetaData
JDBCType	CASE_SENSITIVE	boolean	true/false	getTypeInfo()	Whether this type is case sensitive or not.	DatabaseMetaData
JDBCType	CREATE_PARAMS	String		getTypeInfo()	Parameters used in creating the type.	DatabaseMetaData
JDBCType	DBMS_NAME	String		getTypeInfo()	Local RDBMS Name for this JDBCType	DatabaseMetaData
JDBCType	JDBC_NAME	String	*from java.sql.Types*	getTypeInfo()	Name of the JDBC Type	DatabaseMetaData
JDBCType	LITERAL_PREFIX	String		getTypeInfo()	Prefix used to quote a literal.	DatabaseMetaData
JDBCType	LITERAL_SUFFIX	String		getTypeInfo()	Suffix used to quote a literal.	DatabaseMetaData
JDBCType	LOCAL_NAME	String		getTypeInfo()	Localized Version of the DBMS_NAME	DatabaseMetaData
JDBCType	MAX_SCALE	int		getTypeInfo()	Maximum Scale Supported.	DatabaseMetaData
JDBCType	MIN_SCALE	int		getTypeInfo()	Minimum Scale Supported.	DatabaseMetaData
JDBCType	NULLABLE	String	NO, YES, UNKNOWN	getTypeInfo()	Whether or not a column with this type can be null.	DatabaseMetaData
JDBCType	PRECISION	int		getTypeInfo()	Maximum Precision.	DatabaseMetaData
JDBCType	RADIX	int	10 or 2			DatabaseMetaData
JDBCType	SEARCHABLE	String	NO_WHERE, ONLY_WHERE_LIKE, ALL_WHERE_EXCEPT_WHERE_LIKE, ALL_WHERE	getTypeInfo()	Indicates whether it is possible to use a WHERE clasue based on this type.	DatabaseMetaData

JDBCType	UNSIGNED	boolean	true/false	getTypeInfo()	Whether or not this type is unsigned or not.	DatabaseMetaData
Privilege	CAN_GRANTEE_GRANT_PRIVILEGE	boolean	true/false	getColumnPrivileges(), getTablePrivileges()	Whether or not the grantee can grant access to others	DatabaseMetaData
Privilege	PRIVILEGE_GRANTEE	User		getColumnPrivileges(), getTablePrivileges()	The User who is granted the access.	DatabaseMetaData
Privilege	PRIVILEGE_GRANTOR	User		getColumnPrivileges(), getTablePrivileges()	The User who grants the access.	DatabaseMetaData
Privilege	PRIVILEGE_TYPE	String		getColumnPrivileges(), getTablePrivileges()	Type of Access (SELECT, INSERT, UPDATE, REFRENCES etc.)	DatabaseMetaData
Procedure	IN_CATALOG	Catalog		getProcedures()	Procedure Catalog	DatabaseMetaData
Procedure	IN_SCHEMA	Schema		getProcedures()	Procedure Schema	DatabaseMetaData
Procedure	NAME	String		getProcedures()	Name of the Procedure	DatabaseMetaData
Procedure	RDF:COMMENTS	RDF:Comments		getProcedures()	Explanatory Comment on the Procedure.	DatabaseMetaData
Procedure	TYPE	String	Unknown, NoResult, ReturnsResult	getProcedures()	Whether or not a Procedure returns a Result.	DatabaseMetaData
Product	CATALOG_SEPARATOR	String		getCatalogSeparator()	Separator between catalog and Table names	DatabaseMetaData
Product	CATALOG_TERM	String		getCatalogTerm()	Database vendor's term for Catalog	DatabaseMetaData
Product	CATALOGS	Bag of Catalogs		getCatalogs()	All Catalog Names available in the database (In no order)	DatabaseMetaData
Product	DEFAULT_TRANSACTION_ISOLATION	String	NONE, READ_UNCOMITTED, READ_COMMITED, REPEATABLE_READ, SERIALIZABLE	getDefaultTransactionIsolation()	Default Transaction Isolation Level. (By Isolation we mean atomicity of a Transaction)	DatabaseMetaData
Product	DRIVER	Driver		getDriverName()	JDBC Driver used to connect to the Database	DatabaseMetaData
Product	DOES_DDL_IN_TRANSACT_CAUSE_COMMIT	boolean	true/false	dataDefinitionCausesTransactionCommit()	Does a DDL in a Transaction cause a Commit?	DatabaseMetaData
Product	EXTRA_NAME_CHARACTERS	String		getExtraNameCharacters()	ASCII Special Characters that can be used in names (beyond a-z, 0-9 and _)	DatabaseMetaData
Product	IS_BLOB_IN_MAX_ROW_SIZE	boolean	true/false	doesMaxRowSizeIncludeBlobs()	Does the value returned by getMaxRowSize include LONGVARCHAR and LONGVARBINARY blobs?	DatabaseMetaData
Product	IS_CATALOG_AT_START	boolean	true/false	isCatalogAtStart()	Does the Catalog Name appear at the start of a fully qualified Table Name?	DatabaseMetaData
Product	IS_DDL_IGNORED_IN_TRANSACT	boolean	true/false	dataDefinitionIgnoredInTransactions()	Is a DDL ignored in a Transaction?	DatabaseMetaData
Product	IS_NULL_SORTED_AT_END	boolean	true/false	nullsAreSortedAtEnd()	Are Null values sorted at the end regardless of sort order?	DatabaseMetaData
Product	IS_NULL_SORTED_HIGH	boolean	true/false	nullsAreSortedHigh()		DatabaseMetaData
Product	IS_READ_ONLY	boolean	true/false	isReadOnly()		DatabaseMetaData
Product	MAX_BINARY_LITERAL_LENGTH	int		getMaxBinaryLiteralLength()	How many hex characters can you have in an inline binary literal?	DatabaseMetaData
Product	MAX_CATALOG_LENGTH	int		getMaxCatalogNameLength()	What's the maximum length of a catalog name?	DatabaseMetaData
Product	MAX_CHAR_LITERAL_LENGTH	int		getMaxCharLiteralLength()	Maximum number of characters allowed in a character literal.	DatabaseMetaData

Product	MAX_COLUMN_NAME_LENGTH	int		getMaxColumnNameLength()	Maximum number of characters allowed in a column Name.	DatabaseMetaData
Product	MAX_COLUMNS_IN_GROUP_BY	int		getMaxColumnsInGroupBy()	Maximum number of columns allowed in a GROUP BY clause.	DatabaseMetaData
Product	MAX_COLUMNS_IN_INDEX	int		getMaxColumnsInIndex()	Maximum number of columns allowed in an index.	DatabaseMetaData
Product	MAX_COLUMNS_IN_ORDER_BY	int		getMaxColumnsInOrderBy()	Maximum number of columns allowed in an ORDER BY clause.	DatabaseMetaData
Product	MAX_COLUMNS_IN_SELECT	int		getMaxColumnsInSelect()	Maximum number of columns allowed in a SELECT clause.	DatabaseMetaData
Product	MAX_COLUMNS_IN_TABLE	int		getMaxColumnsInTable()	Maximum number of columns allowed in a table.	DatabaseMetaData
Product	MAX_CURSOR_NAME_LENGTH	int		getMaxCursorNameLength()	Maximum number of characters that can be used in Cursor Name.	DatabaseMetaData
Product	MAX_INDEX_LENGTH	int		getMaxIndexLength()	Maximum number of bytes allowed in an index.	DatabaseMetaData
Product	MAX_PROCEDURE_NAME_LENGTH	int		getMaxProcedureNameLength()	Maximum number of characters allowed in a procedure name.	DatabaseMetaData
Product	MAX_ROW_SIZE	int		getMaxRowSize()	Maximum number of bytes allowed in a single row.	DatabaseMetaData
Product	MAX_SCHEMA_NAME_LENGTH	int		getMaxSchemaNameLength()	Maximum number of characters allowed in a schema name	DatabaseMetaData
Product	MAX_STATEMENT_LENGTH	int		getMaxStatementLength()	Maximum Number of characters allowed in an SQL statement	DatabaseMetaData
Product	MAX_STATEMENTS	int		getMaxStatements()	Maximum number of active statements to this database that may be open at the same time.	DatabaseMetaData
Product	MAX_TABLE_NAME_LENGTH	int		getMaxTableNameLength()	Maximum number of characters allowed in a table name.	DatabaseMetaData
Product	MAX_TABLES_IN_SELECT	int		getMaxTablesInSelect()	Maximum Number of Tables allowed in a SELECT clause	DatabaseMetaData
Product	MAX_USER_NAME_LENGTH	int		getMaxUserNameLength()	Maximum number of characters allowed in a user name.	DatabaseMetaData
Product	NAME	String		getDatabaseProductName()	Product Name for this database.	DatabaseMetaData
Product	NUMERIC_FUNCTIONS	String		getNumericFunctions()	Comma Delimited List of Math Functions	DatabaseMetaData
Product	PROCEDURE_TERM	String		getProcedureTerm()	Database vendor's preferred term for "procedure"	DatabaseMetaData
Product	SCHEMA_TERM	String		getSchemaTerm()	Database vendor's preferred term for "schema"	DatabaseMetaData
Product	SCHEMAS	Bag Of Schema		getSchemas()	Schema in the Database.	DatabaseMetaData
Product	SEARCH_STRING_ESCAPE_PATTERN	String		getSearchStringEscape()	String that can be used to escape "_" or "%" wildcards in the string search pattern used for catalog search parameters.	DatabaseMetaData
Product	SQL_KEYWORDS	String		getSQlKeywords()	Comma-separated list of keywords used by the database that are not SQL-92 keywords	DatabaseMetaData

Product	STORE_LOWER_CASE_IDENTIFIER	boolean	true/false	storesLowerCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in lower case?	DatabaseMetaData
Product	STORE_LOWER_CASE_QUOTED_IDENTIFIER	boolean	true/false	storesLowerCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in lower case?	DatabaseMetaData
Product	STORE_MIXED_CASE_IDENTIFIER	boolean	true/false	storesMixedCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in mixed case?	DatabaseMetaData
Product	STORE_MIXED_CASE_QUOTED_IDENTIFIER	boolean	true/false	storesMixedCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in mixed case?	DatabaseMetaData
Product	STORE_UPPER_CASE_IDENTIFIER	boolean	true/false	storesUpperCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in upper case?	DatabaseMetaData
Product	STORE_UPPER_CASE_QUOTED_IDENTIFIER	boolean	true/false	storesUpperCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in upper case?	DatabaseMetaData
Product	STRING_FUNCTIONS	String		getStringFunctions()	Comma-separated list of string functions.	DatabaseMetaData
Product	SUPPORT_ALTER_TABLE_WITH_ADD_COLUMN	boolean	true/false	supportsAlterTableWithAddColumn()	Is "ALTER TABLE" with add column supported?	DatabaseMetaData
Product	SUPPORT_ALTER_TABLE_WITH_DROP_COLUMN	boolean	true/false	supportsAlterTableWithDropColumn()	Is "ALTER TABLE" with drop column supported?	DatabaseMetaData
Product	SUPPORT_ANSI_92_ENTRY_LEVEL_SQL	boolean	true/false	supportsANSI92EntryLevelSQL()	Is the ANSI92 entry level SQL grammar supported? All JDBC Compliant™ drivers must return true.	DatabaseMetaData
Product	SUPPORT_ANSI_92_FULL_SQL	boolean	true/false	supportsANSI92FullSQL()	Is the ANSI92 full SQL grammar supported?	DatabaseMetaData
Product	SUPPORT_ANSI_92_INTERMEDIATE_SQL	boolean	true/false	supportsANSI92IntermediateSQL()	Is the ANSI92 intermediate SQL grammar supported?	DatabaseMetaData
Product	SUPPORT_CATALOG_IN_DML	boolean	true/false	supportsCatalogsInDataManipulation()	Can a catalog name be used in a data manipulation statement?	DatabaseMetaData
Product	SUPPORT_CATALOG_IN_INDEX_DEFN	boolean	true/false	supportsCatalogsInIndexDefinitions()	Can a catalog name be used in an index definition statement?	DatabaseMetaData
Product	SUPPORT_CATALOG_IN_PRIVILEGE_DEFN	boolean	true/false	supportsCatalogsInPrivilegeDefinitions()	Can a catalog name be used in a privilege definition statement?	DatabaseMetaData
Product	SUPPORT_CATALOG_IN_PROCEDURE_CALLS	boolean	true/false	supportsCatalogsInProcedureCalls()	Can a catalog name be used in a procedure call statement?	DatabaseMetaData
Product	SUPPORT_CATALOG_IN_TABLE_DEFN	boolean	true/false	supportsCatalogsInTableDefinitions()	Can a catalog name be used in a table definition statement?	DatabaseMetaData
Product	SUPPORT_COLUMN_ALIASING	boolean	true/false	supportsColumnAliasing()	Is column aliasing supported?	DatabaseMetaData
Product	SUPPORT_CONVERT	boolean	true/false	supportsConvert()	Is the CONVERT function between SQL types supported?	DatabaseMetaData
Product	SUPPORT_CORE_SQL	boolean	true/false	supportsCoreSQLGrammar()	Is the ODBC Core SQL grammar supported?	DatabaseMetaData
Product	SUPPORT_CORRELATED_SUBQUERIES	boolean	true/false	supportsCorrelatedSubqueries()	Are correlated subqueries supported?	DatabaseMetaData

Product	SUPPORT_DDL_AND_DML_IN_TRANSACTION	boolean	true/false	supportsDataDefinitionAndDataManipulationTransactions()	Are both data definition and data manipulation statements within a transaction supported?	DatabaseMetaData
Product	SUPPORT_DIFFERENT_TABLE_CORRELATION_NAMES	boolean	true/false	supportsDifferentTableCorrelationNames()	If table correlation names are supported, are they restricted to be different from the names of the tables?	DatabaseMetaData
Product	SUPPORT_EXPRESSIONS_IN_ORDER_BY	boolean	true/false	supportsExpressionsInOrderBy()	Are expressions in "ORDER BY" lists supported?	DatabaseMetaData
Product	SUPPORT_EXTENDED_SQL_GRAMMAR	boolean	true/false	supportsExtendedSQLGrammar()	Is the ODBC Extended SQL grammar supported?	DatabaseMetaData
Product	SUPPORT_FULL_OUTER_JOINS	boolean	true/false	supportsFullOuterJoin()	Are full nested outer joins supported?	DatabaseMetaData
Product	SUPPORT_GROUP_BY	boolean	true/false	supportsGroupBy()	Is some form of "GROUP BY" clause supported?	DatabaseMetaData
Product	SUPPORT_GROUP_BY_BEYOND_SELECT	boolean	true/false	supportsGroupByBeyondSelect()	Can a "GROUP BY" clause add columns not in the SELECT provided it specifies all the columns in the SELECT?	DatabaseMetaData
Product	SUPPORT_GROUP_BY_UNRELATED	boolean	true/false	supportsGroupByUnrelated()	Can a "GROUP BY" clause use columns not in the SELECT?	DatabaseMetaData
Product	SUPPORT_INTEGRITY_ENHANCEMENT_FACILITY	boolean	true/false	supportsIntegrityEnhancementFacility()	Is the SQL Integrity Enhancement Facility supported?	DatabaseMetaData
Product	SUPPORT_LIKE_ESCAPE_CLAUSE	boolean	true/false	supportsLikeEscapeClause()	Is the escape character in "LIKE" clauses supported?	DatabaseMetaData
Product	SUPPORT_LIMITED_OUTER_JOINS	boolean	true/false	supportsLimitedOuterJoins()	Is there limited support for outer joins?	DatabaseMetaData
Product	SUPPORT_MINIMUM_SQL_GRAMMAR	boolean	true/false	supportsMinimumSQLGrammar()	Is the ODBC Minimum SQL grammar supported? All JDBC Compliant™ drivers must return true.	DatabaseMetaData
Product	SUPPORT_MIXED_CASE_IDENTIFIER	boolean	true/false	supportsMixedCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case sensitive and as a result store them in mixed case?	DatabaseMetaData
Product	SUPPORT_MIXED_CASE_QUOTED_IDENTIFIER	boolean	true/false	supportsMixedCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case sensitive and as a result store them in mixed case?	DatabaseMetaData
Product	SUPPORT_MULTIPLE_RESULTSETS	boolean	true/false	supportsMultipleResultSets()	Are multiple ResultSets from a single execute supported?	DatabaseMetaData
Product	SUPPORT_MULTIPLE_TRANSACTIONS	boolean	true/false	supportsMultipleTransactions()	Can we have multiple transactions open at once (on different connections)?	DatabaseMetaData
Product	SUPPORT_NON_NULLABLE_COLUMNS	boolean	true/false	supportsNonNullableColumns()	Can columns be defined as non-nullable?	DatabaseMetaData
Product	SUPPORT_OPEN_CURSORS_ACROSS_COMMIT	boolean	true/false	supportsOpenCursorsAcrossCommit()	Can cursors remain open across commits?	DatabaseMetaData
Product	SUPPORT_OPEN_CURSORS_ACROSS_ROLLBACK	boolean	true/false	supportsOpenCursorsAcrossRollback()	Can cursors remain open across rollbacks?	DatabaseMetaData
Product	SUPPORT_OPEN_STATEMENTS_ACROSS_COMMIT	boolean	true/false	supportsOpenStatementsAcrossCommit()	Can statements remain open across commits?	DatabaseMetaData
Product	SUPPORT_OPEN_STATEMENTS_ACROSS_ROLLBACK	boolean	true/false	supportsOpenStatementsAcrossRollback()	Can statements remain open across rollbacks?	DatabaseMetaData
Product	SUPPORT_ORDER_BY_UNRELATED	boolean	true/false	supportsOrderByUnrelated()	Can an "ORDER BY" clause use columns not in the SELECT statement?	DatabaseMetaData

Product	SUPPORT_OUTER_JOINS	boolean	true/false	supportsOuterJoins()	Is some form of outer join supported?	DatabaseMetaData
Product	SUPPORT_POSITIONED_DELETE	boolean	true/false	supportsPositionedDelete()	Is positioned DELETE supported?	DatabaseMetaData
Product	SUPPORT_SCHEMAS_IN_DML	boolean	true/false	supportsSchemasInDataManipulation()	Can a schema name be used in a data manipulation statement?	DatabaseMetaData
Product	SUPPORT_SCHEMAS_IN_INDEX_DEFN	boolean	true/false	supportsSchemasInIndexDefinitions()	Can a schema name be used in an index definition statement?	DatabaseMetaData
Product	SUPPORT_SCHEMAS_IN_PRIVILEGE_DEFN	boolean	true/false	supportsSchemasInPrivilegeDefinitions()	Can a schema name be used in a privilege definition statement?	DatabaseMetaData
Product	SUPPORT_SCHEMAS_IN_PROCEDURE_CALLS	boolean	true/false	supportsSchemasInProcedureCalls()	Can a schema name be used in a procedure call statement?	DatabaseMetaData
Product	SUPPORT_SCHEMAS_IN_TABLE_DEFN	boolean	true/false	supportsSchemasInTableDefinitions()	Can a schema name be used in a table definition statement?	DatabaseMetaData
Product	SUPPORT_SELECT_FOR_UPDATE	boolean	true/false	supportsSelectForUpdate()	Is SELECT for UPDATE supported?	DatabaseMetaData
Product	SUPPORT_STORED_PROCEDURES	boolean	true/false	supportsStoredProcedures()	Are stored procedure calls using the stored procedure escape syntax supported?	DatabaseMetaData
Product	SUPPORT_SUBQUERIES_IN_COMPARISONS	boolean	true/false	supportsSubqueriesInComparisons()	Are subqueries in comparison expressions supported?	DatabaseMetaData
Product	SUPPORT_SUBQUERIES_IN_EXISTS	boolean	true/false	supportsSubqueriesInExists()	Are subqueries in 'exists' expressions supported?	DatabaseMetaData
Product	SUPPORT_SUBQUERIES_IN_INS	boolean	true/false	supportsSubqueriesInIns()	Are subqueries in 'in' statements supported?	DatabaseMetaData
Product	SUPPORT_SUBQUERIES_IN_QUANTIFIEDS	boolean	true/false	supportsSubqueriesInQuantifieds()	Are subqueries in quantified expressions supported?	DatabaseMetaData
Product	SUPPORT_TABLE_CORRELATION_NAMES	boolean	true/false	supportsTableCorrelationNames()	Are table correlation names supported?	DatabaseMetaData
Product	SUPPORT_TRANSACTIONS	boolean	true/false	supportsTransactions()	Are transactions supported?	DatabaseMetaData
Product	SUPPORT_UNION	boolean	true/false	supportsUnion()	Is SQL UNION supported?	DatabaseMetaData
Product	SUPPORT_UNION_ALL	boolean	true/false	supportsUnionAll()	Is SQL UNION ALL supported?	DatabaseMetaData
Product	SUPPPORT_POSITIONED UPDATE	boolean	true/false	supportsPositionedUpdate()	Is positioned UPDATE supported?	DatabaseMetaData
Product	SYSTEM_FUNCTIONS	String		getSystemFunctions()	Comma-separated list of System Functions.	DatabaseMetaData
Product	TABLE_TYPES	String		getTableTypes()	Comma-separated list of supported table types in this DBMS	DatabaseMetaData
Product	TIME_DATE_FUNCTIONS	String		getTimeDateFunctions()	Comma-separated list of time and date functions	DatabaseMetaData
Product	URL	String		getURL()	JDBC Url of this database.	DatabaseMetaData
Product	USER	User		getUserName()	User of this database.	DatabaseMetaData
Product	USES_LOCAL_FILE_PER_TABLE	boolean	true/false	usesLocalFilePerTable()	Does the database use a file for each table?	DatabaseMetaData
Product	USES_LOCAL_FILES	boolean	true/false	usesLocalFiles()	Does the database store tables in a local file?	DatabaseMetaData
Product	VERSION	String		getDatabaseProductVersion()	Version for this Database Product	DatabaseMetaData
Schema	NAME	String		getSchemas()	Name of Schema	DatabaseMetaData
Table	AUTO_UPDATED_COLUMNS	Bag of Columns		getVersionColumns()	All Columns that are automatically updated if a row is updated.	DatabaseMetaData

Table	COLUMNS	Bag of Columns		getColumns()	Columns in Table	DatabaseMetaData
Table	IN_CATALOG	Catalog		getTables()	Table Catalog	DatabaseMetaData
Table	IN_SCHEMA	Schema		getTables()	Table Schema	DatabaseMetaData
Table	NAME	String		getTables()	Name of the Table	DatabaseMetaData
Table	RDF:COMMENTS	String		getTables()	Explanatory Comment on the Table	DatabaseMetaData
Table	TABLE_PRIVILEGE	Privilege		getTables()	Privileges for this Table	DatabaseMetaData
Table	TABLE_TYPE	String	TABLE, VIEW, SYSTEM TABLE, GLOBAL TEMPORARY, LOCAL TEMPORARY, ALIAS, SYNONYM	getTables()	Type of the Table	DatabaseMetaData
User	NAME	String		getUserName()	Username of this User.	DatabaseMetaData
User	RIGHTS	UserRights		allProceduresAreCallabe(), allTablesAreSelectable()	Rights for this User	DatabaseMetaData
UserRights	ARE_PROCEDURES_CALLABLE	boolean	true/false	allProceduresAreCallabe()	Whether or not the current user has the rights to call al procedures	DatabaseMetaData
UserRights	ARE_TABLES_SELECTABLE	boolean	true/false	allTablesAreSelectable()	Whether or not the current user has the rights to call SELECT statements on all Tables	DatabaseMetaData

APPENDIX B

DATABASE CONNECTION POOL PERFORMANCE METRICS

To test the Database Connection Pool's performance and the fact that using a Connection Pool is indeed superior to creating a connection every time, an experiment was conducted on a Pentium II (266) with 128 MB RAM running WIN-NT Workstation 4.0 (SP5).

The Database the testing was performed on was MS-Access 97 (SR2). A tester program was written that spawned 10 threads. Each thread simultaneously executed a SELECT query (`select * from test`, the table had 6 columns and 250 rows) for a specified amount of time. This program then obtained the `ResultSet,` stepped through each row and converted each row into a `Hashtable` with the column names as the keys. This procedure was repeated both with and without the Connection Pool. Without the Pool a connection was created each time a query had to be executed.

The turnaround time (i.e. the time it took for executing the query, stepping through the `ResultSet` and converting each row into a `Hashtable` was recorded for each query. This time was then plotted for both the cases.

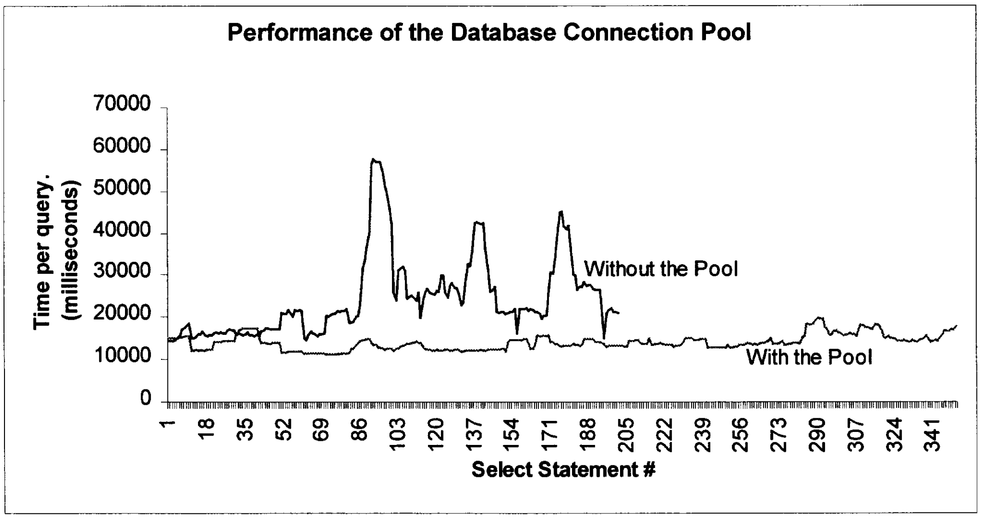


Figure 28: Database Connection Pool Performance at a Glance.

Summary Statistics

	Without the Database Connection Pool	With the Database Connection Pool
Time of Run (secs)	500	500
Number of Select Statements	201	351
Mean Turnaround Time (millis)	24490.40299	13993.73504
Maximum Turnaround Time (millis)	57783	19828
Minimum Turnaround Time (millis)	14301	11136
Median Turnaround Time (milllis)	21481	13800
Number of Threads	10	10

REFERENCES

Adler, A., Berglund, A., Caruso, J., Deach, S., Grosso, P., Gutentag, E., Milowski, A., Parnell, S., Richman, J., and Zilles S., *Extensible Stylesheet Language (XSL) Version 1.0*, W3C Working Draft 1, Mar. 2000, http://www.w3.org/TR/xsl/.

Agosti, M. and Smeaton A., Information Retrieval and Hypertext. Boston: Kluwer Academic Publishers.

Anderson, M., K. Data Scalability in Open Hypermedia Systems, Proceedings of ACM Hypertext '99 Conference, Darmstadt, Germany, pp. 27-36, Feb. 21-25 1991.

Balasubramanian, V., Bieber, M. and Isakowitz, T., A Case Study in Systematic Hypermedia Design, Information Systems Journal (forthcoming).

Bapat, A., Waesch, J., Aberer, K., and Haake, J., HyperStorM: an Extensible Object-Oriented Hypermedia Engine, Proceedings of ACM Hypertext Conference, Washington, D.C, pp. 203-214, Sep. 1996.

Berners-Lee T., Fielding R. and Masinter L., "Uniform Resource Identifiers (URI): Generic Syntax", Internet Engineering Task Force Request For Comments 2396, August 1998.

Bieber, M., Supplementing Applications with Hypermedia, Technical Report, New Jersey Institute of Technology, Information Systems Department, Mar. 1997.

Bieber, M., On Integrating Hypermedia into Decision Support and Other Information Systems, Decision Support Systems, vol. 14, pp. 251-267, 1995.

Bieber, M. and Kacmar, C., Designing Hypertext Support for Computational Applications, Communications of the ACM, vol. 38(8), pp. 99-107, 1995.

Bieber, M. and Vitali, F., Toward Support for Hypermedia on the World Wide Web, IEEE Computer, vol. 30(1), pp. 62-70, 1997.

Bieber, M. and Joonhee Y., Hypermedia: A Design Philosophy, ACM Computing Surveys (forthcoming).

Chaudhuri, S. and Dayal, U., An Overview of Data Warehousing and OLAP Technology. SIGMOD Record, vol. 26(1), pp. 65-74, 1994.

Chiramella, Y., and Kheirbek, A., An Integrated Model for Hypermedia and Information Retrieval, In Information Retrieval and Hypertext, M. Agosti, A. Smeaton (Eds), Kluwer, Amsterdam (NL), pp.139-176, 1998.

Chiramella, Y., Browsing and Querying: Two Complementary Approaches for Multimedia Information Retrieval, Proceedings of Hypertext – Information Retrieval – Multimedia, (HIM '97), Dortmund, Germany, pp. 9-26, 1997.

Chiu, C. and Bieber, M., A Generic Dynamic-Mapping Wrapper for Open Hypertext System Support of Analytical Applications, Proceedings of ACM Hypertext '97, ACM Press, Washington, D.C., pp. 218-219, Apr. 1997, http://www.cis.njit.edu/~bieber/pub/ht97/ht97-mac.html.

Christodoulou, S., Styliaras, G. and Papatheodourou, T., Evaluation of Hypermedia Application Development and Management Systems, Proceedings of ACM Hypertext '98, ACM Press, Pittsburgh, pp. 1-10, May 1998.

Constantopoulos, P., Theodorakis, M. and Tzitikas, Y., Developing Hypermedia Over an Information Repository, Proceedings of the 2nd Workshop on Open Hypermedia Systems, ACM Hypertext '96 Conference, Washington, DC., pp. 227-238, Sep. 1996.

Diáz, A., Isakowitz, T., Maiorana, V. and Gilabert, G., RMC: A Tool To Design WWW Applications. Proceedings of the Fourth International World Wide Web Conference, Boston, Dec. 1995.

Falquet, G., Guyot, J. and Prince, I., Generating Hypertext Views on Databases, CUI Technical Report No 101, University of Geneva, 1995.

Falquet, G., Guyot, J. and Nerima, L., Languages and Tools to Specify Hypertext Views on Databases, International Workshop webDB '98 selected papers, Valencia, Spain, Springer-Verlag LNCS 1590, Mar. 1998.

Frank, M., Database and the Internet, DBMS Magazine, vol. 8(13), pp. 39-47, Dec. 1995.

Fountain, A., Hall, W., Health, I. and Davis, H. C., Microcosm: An Open Model for Hypermedia with Dynamic Linking. Proceedings of the ACM European Conference on Hypertext, Paris, France, pp. 298-311, 1990.

Furner, J., Ellis, D. and Willett, P., The Representation and Comparsion of Hypertext Structures using Graphs , In Information Retrieval and Hypertext, M. Agosti, A. Smeaton (Eds), Kluwer, Amsterdam (NL), pp. 75-96., 1989.

Gardner, S. R., Building the Data Warehouse, Communications of the ACM, vol. 41(9), pp. 52-60, Sep. 1998.

Geldof, S., Hypertext generation from databases on the Internet, Proceedings of the 2nd Intl. Workshop on Applications of Natural Language to Information Systems (NLDB '96), Amsterdam, IOS Press, pp. 102-114, 1996.

Golovchinsky, G., "What the Query Told the Link: The Integration of Hypertext and Information Retrieval", Proceedings of Hypertext '97, pp. 67-74, Apr. 1997.

Grønbæk, K., and Trigg, R., Design Issues for a Dexter-Based Hypermedia System. Communications of the ACM, vol. 37(2): pp. 40-49, 1994.

Grønbæk, K., and Trigg, R., From Web to Workplace: Designing Open Hypermedia Systems, MIT Press., 1999.

Hara, Y. and Botafogo, R. A., Hypermedia Databases: A Specification and Formal Language, Proceeding of the Databases and Expert Systems Applications Conference (DEXA), Springer-Verlag LCNS 856, pp. 520-530, 1994.

Holsheimer, M. and Siebes, A, (Report CS-R9406) Data Mining, The Search for Knowledge in Databases, CWI, Amsterdam, ftp://ftp.cwi.nl/pub/CWIreports/AA/CS-R9406.ps.Z, 1994.

Inmon, W. H., Building the Data Warehouse, Second Edition, Wiley Comp., ISBN 0471-14161-5, USA, 1996.

Johnson, A. H., Data Warehousing, Computerworld, vol. 33(49), pp.74-75, Dec. 1999.

Isakowitz, T., Stohr, E. and Balasubramanian, P., RMM: A Methodology for Structuring Hypermedia Design. Communications of the ACM, vol. 38(8), pp. 34-44. Aug. 1995.

Lassila Ora and Swick Ralph R. (Editors), "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation 22, Feb. 1999.

Leggett, J. J., (ed.) Hypertext '93 workshop on hyperbase systems. Technical Report TAMU-HRL 93-009, Texas A&M University, 1993.

Nguyen, T., and Srinivasan, V., Accessing Relational Databases from the World Wide Web, Proceedings of the ACM SIGMOD Conference, pp. 529-540, 1996.

Papadopoulos, A., Vaitis, M. and Christodoulakis, D., Building Hypertext Interfaces to Existing Relational Databases. Proceeding of the 7th Intl. Conference on Database and Expert Systems Applications (DEXA '96), Springer-Verlag LCNS 1134, Zürich, Switzerland, pp. 276-288, 1996.

Schwabe, D. and Rossi, G., The Object-Oriented Hypermedia Design Model. Communications of the ACM, pp. 45-46, 1996.

Schwabe, D., Rossi, G., and Barbosa, S., Systematic Hypermedia Application Design with OOHDM. ACM Hypertext '96 Conference, New York, pp. 116-128, 1996.

Salton, G., Automatic Text Processing: : the transformation, analysis, and retrieval of Information, Reading, MA: Addison-Wesley, 1989.

Salton G., Allan J., Buckley C. and Singhal A., "Automatic Analysis, Theme Generation, and Summarization of Machine-Readable Texts", in Science, vol. 264, pp.1421-1426, 1994.

Savoy, J., Citation Schemes in Hypertext Information Retrieval. In Information Retrieval and Hypertext, M. Agosti, A. Smeaton (Eds), Kluwer, Amsterdam (NL), pp. 99-120, 1996.

Tamayo, P., Berlin, J., Dayanand, N., Drescher, G., Mani, D. R. and Wang C., Oracle Darwin Technical White Paper Darwin: A Scalable Integrated System for Data Mining, May 1997, http://www.oracle.com/datawarehouse/products/datamining/downloads/darwin-arch.html.

Wan, J., Integrating Hypertext into Information Systems through Dynamic Linking. Ph. D. dissertation, New Jersey Institute of Technology, Institute for Integrated Systems Research, Newark NJ 07102, 1996.

Wan, J. and Bieber, M., Providing Relational Database Management Systems with Hypertext. Proceedings of the Thirtieth Annual Hawaii International Conference on System Sciences, IEEE Press, Washington, D.C., vol. VI, pp. 160-166, Jan. 1997.

Wiil U. K. and Leggett, J. J., The HyperDisco Approach to Open Hypermedia Systems. ACM Hypertext Conference, Washington, pp. 140-48, 1996.

Yoo, J., Relationship Analysis. Ph.D. Dissertation, New Jersey Institute of Technology, CIS Department, 2000.

Yoo, J. and Bieber, M., Towards a Relationship Navigation Analysis. Proceedings of the 33rd Hawaii International Conference on System Sciences, IEEE Press, Washington, D.C., Jan. 2000.

Yoo, J. and Bieber, M., Finding Linking Opportunities through Relationship-based Analysis. Hypertext '00 Proceedings, San Antonio, ACM Press, Jun. 2000.