Open Database Connectivity Development of the Context Interchange System

by

Andy C. Y. Shum

Submitted to the Department of Electrical Engineering
and Computer Science in Partial Fulfillment of the
Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and
Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

December 17, 1996

© 1996 Massachusetts Institute of Technology. All Rights Reserved.

Author ....
Department of Electrical Engineering and Computer Science
December 15, 1996

Certified by ....
Michael D. Siegel
Thesis Supervisor

Accepted by ....
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

OF TECHNOLOGY

MAR 21 1997

Eng.

Open Database Connectivity Development of the Context Interchange System

by

Andy C. Y. Shum

Submitted to the
Department of Electrical Engineering and Computer Science

December 17, 1996

In Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Open Database Connectivity (ODBC) development is an extension to the Context Interchange System, a system that aims to provide semantic interoperability between heterogeneous system environments. The development of ODBC is a front end extension which provides users in the Windows working environment the capability to access the Context Interchange System using Windows applications. The primarily goal of the thesis is to present the design and implementation of the ODBC driver which serves as the bridge between the Context Interchange System and ODBC compliant applications. In the mean time, the development of the ODBC driver will serve as a guideline for future integration between common applications and the Context Interchange System.

Thesis Supervisor: Michael D. Siegel
Title: Principal Research Scientist, Sloan School of Management

Acknowledgments

I would like to thank my thesis advisor, Michael Siegel, and Professor Stuart Madnick for giving me the opportunity work in the Context Interchange project. I would like to express my gratefulness to them for their advice and support, especially during my hard time in the project. Thank you so much for having confidence in me and bearing with me! Also, I would like to thank the entire Context Interchange research group for providing such a great working environment. In particular, for Allen Moulton's help and insight in the driver development, Tom Lee's knowledge of the Context Interchange System, Jesse Jacobson's expertise in Excel, and Cheng Goh, Stephane Bressan, Tito Pena, Kofi Fynn and other group members' help and support.

Also, I would like to take this opportunity to thank my parents for their support throughout my years at MIT. Thank you so very much for giving me the freedom and opportunity to pursue my career and destiny.

Finally, I would like to thank all of my friends at MIT. I could not have make it through without you all! In particular, the "FOPA" member Felix Lo, Oliver Yip and Patrick Chan. Also the Class of 96 members Anita Chan, Jenny Lee and Vinci Leung. In addition, this year's senior Calvin Yuen, Julian Lee, Preston Li, Zhou Yu, Danny Yim, Chris Leung and Brian Lee. And last but not least, all the juniors, sophomores, and freshmen who know me. Thank you so much for your support. It could have been the worst and dull years of my life without you!. But now, it is definitely one of the most celebrated time of my life. Thank you very much!

Contents

1 Introduction **9**

2 Background **13**

2.1 Context Interchange System Architecture 13

2.1.1 First Tier (Front End)....................14

2.1.2 Second Tier (Middleware)....................15

2.1.1 Third Tier (Back End)....................18

2.2 Open Database Connectivity (ODBC)....................18

2.2.1 ODBC Defined....................19

2.2.2 Components of ODBC....................19

2.3 Hypertext Transfer Protocol (HTTP)....................21

3 Design of the ODBC Driver **23**

3.1 Role of the Driver....................23

3.2 Structure of the Driver....................24

3.2.1 General Architecture....................24

3.2.2 Internal Data Structure....................29

3.3 Interactions with the Context Mediator....................32

3.3.1 Comparisons between ODBC API & the Context Mediator Interface....................33

3.3.2 Operations between the ODBC Driver & the Context Mediator....................33

4 Implementation/Scenario **37**

4.1 Implemented ODBC Function Calls....................37

4.2 Migrating to the Second Prototype....................44

4.3 Algorithms and Tradeoffs....................45

4.3.1 Algorithms Analysis....................45

4.3.2 Tradeoffs....................47

4.4 Scenario of Retrieving Financial Data....................48

4.4.1 Single-datasource query scenario (First & Second Prototypes) 48
4.4.2 Multi-datasource query scenario (Second Prototype) 55

5 Conclusions **57**

5.1 Further ODBC Driver Development 57
5.2 Re-Design of the Context Mediator Interface 58
5.3 Full ODBC and Context Interchange System Integration 60
5.4 Future Context Interchange System Extensions 61
5.4.1 ActiveX 61
5.4.2 OLE DB 61
5.5 Conclusions 62

A Sample HTML formatted result file **63**

B ODBC function calls sequence **64**

B.1 MS Query Example 64
B.2 MS Excel Example (query using SQL.REQUEST functions) 66

C User's Manual **70**

C.1 Guide to create a new Excel spreadsheet using the Context Mediator-ODBC Driver 70
C.2 Complete List of Implemented ODBC API Function Calls 71
C.3 Specification of the ODBC Driver 72
C.4 Miscellaneous Instructions of the Driver 78

List of Figures

Figure 2.1-1: The Context Interchange System 13

Figure 2.2-1: ODBC architecture 20

Figure 2.3-1: Examples of HTTP/1.0 protocol 22

Figure 3.1-1: Overview of the ODBC Framework 23

Figure 3.2-1: General Structure of the Context Mediator-ODBC Driver 25

Figure 3.2-2: Internal data structure of the Context Mediator-ODBC driver 32

Figure 4.1-1: Essential components of the ODBC driver 38

Figure 4.3-1: The “pos_pair” array structure for retrieving data 46

Figure 4.4-1: ODBC calls and Context Mediator interface used in MS Query 50

Figure 4.4-2: Table names are shown after SQLTables is called 51

Figure 4.4-3: Column names of the chosen table is shown after SQLColumns 51

Figure 4.4-4: Inputting SQL query in the SQL box 52

Figure 4.4-5: Result of the SQL query from the Context Mediator is returned 52

Figure 4.4-6: ODBC function calls and Context Mediator interface for MS Excel 53

Figure 4.4-7: The MS Excel demo 55

Figure 4.4-8: Multi-datasource query example by MS Excel using the second Context Mediator prototype 56

List of Tables

Table 2.1-1: Context Mediator interface and its corresponding function[6] 15

Table 2.1-2 Results of the same query with different specified contexts 17

Table 3.3-1: ODBC Function Calls and their corresponding Context Mediator Interface 35

Table 4.4-1: Summary of the three scenarios given in this section 48

Table C.2-1: Complete list of implemented and non-implemented ODBC function calls 71

Chapter 1

Introduction

Over the last few years, the World Wide Web has experienced an exponential growth in its usage. Information of every conceivable kind is being made available on the Web with more being added every day[5]. People can not only view text formatted information from the Internet, but they can also enjoy the power of multimedia through pictures, video and audio clips from the Web. The Internet has indeed become one of the most essential resources for all kinds of information in the world.

However, as usage of the World Wide Web has been constantly increasing for the past few years, so has been the problem of trying to correctly interpret all the information. In the process of retrieving data from the Internet, or from any database, one major obstacle is to understand and unravel the true meaning of the information obtained. In other words, meaningful data exchange can only be accomplished when the data receivers and data providers have the same interpretation of the data. One obvious example is the ambiguity of a date such as "6/4/96". In the US, this means June 4, 1996; while in Europe, this means April 6, 1996.

The Context Interchange Project is designed to resolve the above problem by providing semantic interoperability between heterogeneous system environments. The key to the Context Interchange approach is the notion of *context* - the underlying meaning and interpretation of the data. In the Context Interchange System, it is the job of the Context Mediator to detect any semantic conflicts between the data sources and data receivers. When conflict occurs, the Context Mediator is also responsible for resolving context conflicts and transforming the data to become meaningful for the receivers. Note that in order for the Context Mediator to resolve semantic conflicts between data sources, each of the data sources accessible by the Context Mediator has to be registered; that is, the Context Mediator has to have information about the data sources' context before hand.

In the current Context Interchange System prototype, a web browser is the only front end (or client application) of the system. Structured Query Language (SQL) is employed by the Context Interchange System as the means of issuing queries to the data sources. In order to provide better interaction between users and the Context Interchange System, supporting another other front-end working environments becomes necessary. Since standard SQL queries are used in the Context Interchange System, using applications such as Microsoft Excel or Microsoft Access, which also use SQL queries, as the new front end for the Context Interchange System is an appropriate choice.

Microsoft Windows is no doubt the dominant client operating system in the contemporary computer world. Supporting users working in the Windows environment is indeed a natural choice for the Context Interchange System extension. As Open Database Connectivity(ODBC), developed by Microsoft, provides for flexible interoperation between applications and data sources, it was selected to be the cornerstone for the extension of the Context Interchange System. ODBC is designed to provide a standard interface which allows applications and data sources to shuttle data requests and results back and forth. By building the ODBC extension to the Context Interchange System, users will no longer be restricted to make use of the Context Interchange System merely through web browsers; users working in Windows will also be able to access the system through client applications, such as Microsoft Access or Microsoft Excel, or other ODBC compliant applications, such as Power Builder or Business Objects. Note that in these cases, users can perform manipulations on the returned data, unlike the display only capability of using the web browser.

The scope of this thesis is primarily focused on the development of the ODBC driver which provides the above accessibility to the Context Interchange System for users in the Windows environment. At the same time, the development of the ODBC driver will invoke many issues regarding the integration between common applications and the Context Interchange System. General problems as well as enhanced capabilities associated with the ODBC extension to the System will be investigated. In addition, the current development of accessibility from ODBC to the Context Interchange System will serve as

a guideline for future two-way traffic between ODBC and the System. The development will also be beneficial for future extensions to the Context Interchange project.

The rest of the thesis will be organized as follows. In Chapter 2, background information on the Context Interchange System architecture and the idea behind ODBC will be presented. Hypertext Transfer Protocol (HTTP) will also be discussed in this chapter. Chapter 3 will focus on the design of the ODBC driver. In particular, the general architecture of the driver, the internal data structure of the driver, as well as the interaction between the ODBC driver and the Context Mediator. The implementation of the driver, including the development of the second prototype, certain algorithms, and the general tradeoffs of the driver will be examined in Chapter 4. In addition, examples of retrieving data of financial information will be presented in this chapter. Chapter 5 will address issues and impacts regarding the ODBC development to the Context Interchange System. It will also discuss other related future development to the System. Finally, this chapter will summarize the ODBC development as a whole.

Chapter 2

Background

2.1 Context Interchange System Architecture

The Context Interchange System, which uses Structured Query Language (SQL) as the means of issuing query, defines a strategy for integrating data providers and data receivers, and in the mean time, resolving any semantic conflicts between the two. The system architecture is divided into three tiers, as shown in Figure 2.1-1. The first tier, also called the front end of the System, is the client application, namely the Web browser. The second tier is the context mediator, the heart of the Context Interchange System, which performs conflict detection, conflict resolution, query optimization and query execution.[3] The third and final tier is the back end of the system that consists of the gateway technology to the data sources, i.e., the relational databases and the Web.[6]

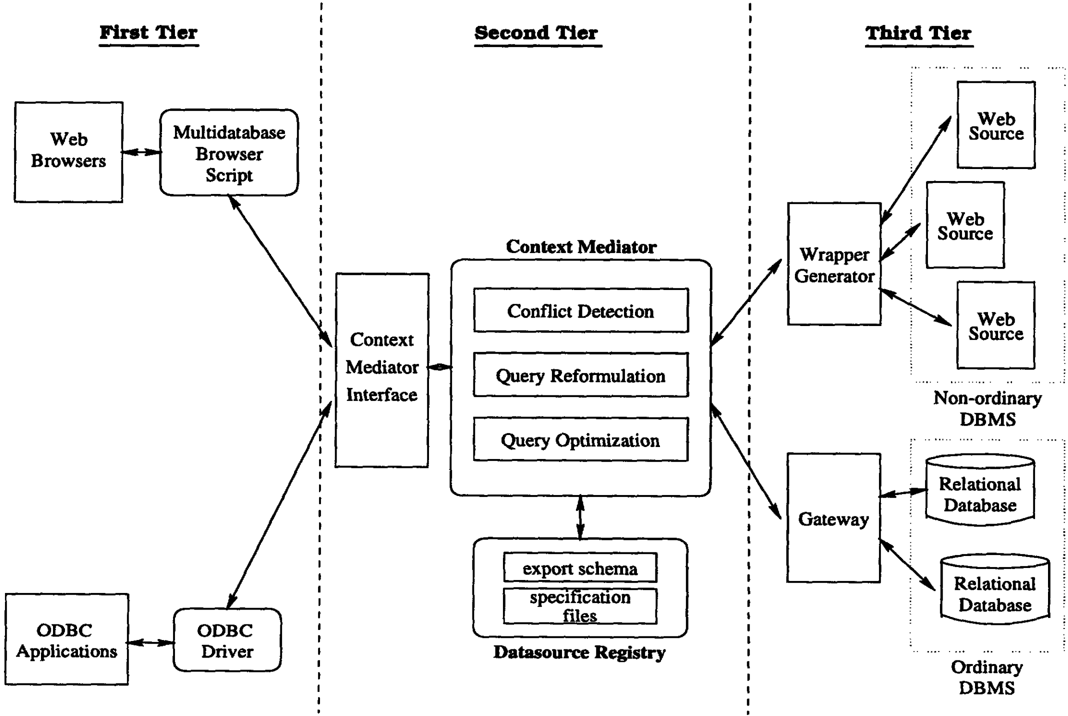

Figure 2.1-1: The Context Interchange System

Client applications (or data receivers) interact with the data sources through the Context Mediator by issuing standard SQL queries. The Context Mediator translates the SQL query sent by the client application into the source context if necessary and then sends it to the data source. When data return, the Context Mediator translates the data back into the receiver's context if necessary and then ships them back to the client application.

2.1.1 First Tier (Front End)

In the current prototype of the Context Interchange System, a web browser is the only client application of the system. In particular, the Multidatabase Browser [6][1] is an interface on the web that guides users to formulate an SQL query that is to be sent to the context mediator middleware.

The Multidatabase Browser operates as follows. First, the user has to select from a list of registered Context Interchange data sources which one(s) he would like to query from. Then, he needs to select the table within the chosen data source(s), as well as the attributes in the selected table(s). After that, the user is asked to place constraints on the attributes that he has selected. Finally, the user can choose his desired local context for the data, and choose whether to have the query undergo mediation or not. This step by step procedure facilitates users in and serves as a guideline for building an SQL query that is to be sent to the Context Interchange System.

In order for the client application to interact with the middleware service of the Context Mediator, a standard set of interfaces has to be defined. The current interface includes the following calls:

1. The Multidatabase Browser is a Web-based program developed by the members of the Context Interchange System.

Table 2.1-1: Context Mediator interface and its corresponding function[6]

API Functions	Functionality
SQLDataSources	Returns the list of registered data sources
SQLGetInfo	Returns information about the specified data source
SQLTables	Returns the available table names of a specified data source
SQLColumns	Returns the available column names of a specified table
SQLDescribeCol	Returns the datatype of a specified column
SQLExecDirect	Executes the SQL query and stores the results temporarily in a queue
SQLFetch	Returns a single row of result
SQLExtendedFetch	Returns the entire result set

By following the above interface, it is feasible for other client applications to access the Context Interchange System. In fact, this leads to the ODBC extension of the Context Interchange System, which will be described in detail in Chapter 3 and 4.

2.1.2 Second Tier (Middleware)

Lying in between the front end and the back end is the core of the Context Interchange System. Conflict detection, query optimization and query execution are all carried out in this middleware service.[3] Context conflict, if any, is detected according to the data source registry, while query optimization and execution are performed by the Context Mediator.

Data Source Registry

In order for a data source to be part of the Context Interchange System, the data source has to be registered. Otherwise, the context for that particular data source is not known by the System, and it is not possible for the System to apply any operation to that data source.

Note that in the Context Interchange System vocabulary, register means the creation of an export schema and a specification file for that particular data source.

The export schema defines the attributes (or columns) of each table for a specific data source. It is not necessary that the export schema contains all the available tables and its corresponding attributes. It can just contain tables and attributes that are supported by the Context Interchange System. In addition, a data type associated with each attribute is also present in the export schema. The datatype is in fact crucial to the ODBC extension of the System. (Please refer chapter 3 and 4.)

The specification file describes the actions that need to be executed by the Wrapper Generator (Section 2.1.3) in order to access and retrieve data from the data sources. On the one hand, if the data source is a relational database, it can be as simple as an instruction of submitting the input SQL query directly to the database gateway. If the data source is a Web site instead, the specification file includes information about the sequence of operations that need to be performed so as to retrieve the necessary data. A more detailed description of the Wrapper technology is discussed in section 2.1.3.

Context Mediator

Context conflict identification and resolution is carried out by the Context Mediator. However, users can choose whether to have mediation executed or not. If no mediation is required by the user, the SQL query submitted by the user will simply be sent to the target data source. If the data source is a relational database, the query is sent directly to the database gateway directly. On the other hand, if the data source is a Web site, the query is processed and a sequence of operations are performed by the Wrapper Generator according to the instructions indicated in the specification file. In any case, results from a database or from a web site are returned to the Context Mediator, and the Context Mediator then sends them back to the front end application.

In the case where mediation is desired, the Context Mediator first determines the context of the receivers and sources according to the registry information, and detects any semantic conflicts between the two. After this preliminary conflict detection stage,[2] the

Context Mediator converts the query into the source's context and performs query optimization.[3] The original query may be divided into a number of sub-queries. Then the query or sub-queries are sent to the Wrapper or to the database gateway, much like the simple case of no mediation. As all the results are gathered by the Context Mediator, the Context Mediator converts them back into the receiver's context, and finally, returns them to the client application.

For example, consider two of the many financial data providers, Worldscope and Disclosure. The providers have different contexts for their data. In Worldscope, all the financial data such as net income and total assets are stored in 1000's of units and in US dollars. In Disclosure, all the figures are stored in single units and the currency depends upon the location of incorporation. So, if a user wants to find out the net income and total assets of a German company called "Daimler Benz Corporation" from the Disclosure data source (DiscAF) and to display the result in Worldscope context, the Context Mediator will go through the above described mediation and return the data in the appropriate context. In particular, the conflicts to be resolved between Disclosure and Worldscope are the company name (Daimler Benz Corp vs. Daimler-Benz AG), the unit difference (1 vs. 1000) and the currency difference (Deutsche Marks vs. US Dollars). The table below summarizes the results of the query with the two different specified contexts.

Table 2.1-2 Results of the same query with different specified contexts

Context	Mediation	Net Income	Total Assets
Disclosure[a]	No	615,000,000	1,073,740,000
Worldscope[b]	Yes	412,050[c]	719,407

a. Query: *select discaf.net_income, discaf.total_assets from discaf where discaf.company_name = 'Daimler Benz Corp'*(in Disclosure context)
b. Query: *select discaf.net_income, discaf.total_assets from discaf where discaf.company_name = 'Daimler-Benz AG'*(in Worldscope context)
c. Since Worldscope context has the amount in 1000 units and currency in US dollars, the result of 615000000*(0.67/1000) = 412050. (1 DEM = 0.67 USD). Similarly for Total Assets.

2. Notice that context conversion may also occur in the remaining stages if necessary.
3. Query Optimization is still an on-going research in the Context Interchange Project.

2.1.1 Third Tier (Back End)

The back end of the Context Interchange System consists of the data sources and their corresponding gateways. For relational databases, the gateway simply presents the SQL query to the database. However, for SQL query going to a web site, it has to go through the Wrapper Generator for data retrieval.

Wrapper Generator

The Wrapper Generator is designed to incorporate data from Web sites into the Context Interchange System. The Wrapper Generator can be thought of as a high-level interpreter[6]. It has the capability to extract what the client application requests, and perform the necessary operations to obtain data from the Web source according to the information in the specification file. By following the instructions of the specification file, the Wrapper sends the necessary HTTP commands to the web site step by step in order to obtain the required web pages. The wrapper then extracts the required data from the HTML text of the Web page. A specification file wizard is currently under development in the Context Interchange project to not only cope with changes in registered web sources, but it will also help users to "wrap" new web data sources. (For more information on the Wrapper technology, please refer Chapter 3 of [6].)

2.2 Open Database Connectivity (ODBC)

In the traditional database world, an application usually can only perform a specific database task with a specific database management system (DBMS). When the application needs to access data from another DBMS, it is necessary to install another specific software or even hardware package. *Open Database Connectivity,* commonly known as ODBC, resolves the above problem. Since there are and always will be many viable communication methods, data protocols, and data source capabilities in the coming future, the ODBC solution is to allow different technologies to be used by defining a standard interface for both the applications and the data sources.[10]

2.2.1 ODBC Defined

ODBC is designed to provide maximum interoperability - it is defined as a method of communication between as many applications and data sources as possible. It allows applications to be developed without targeting a specific data source. ODBC does not restrict an application to use one database type and one database server. Moreover, it does not limit the means of communications to the data sources.[12]

In order to fulfill the above requirements, the application and the data source must agree upon a common method to access the data. This agreement is accomplished by the establishment of the ODBC interface. This standard set of function calls allows applications to access data in different DBMS using SQL as the standard for accessing data. To use ODBC, applications must use the ODBC API function calls as defined. In addition, the SQL statements must comply to the ODBC SQL syntax. The database side of this open connectivity is provided by drivers - dynamic-link libraries (DLL) which an application can invoke to gain access to a particular data source through a particular communication method.[10] The drivers convert the ODBC API function calls into calls supported by the requested data sources, and in the mean time, transform the ODBC SQL syntax into syntax accepted by the requested data sources. The ODBC driver is the crucial component of the ODBC framework. It plays the role of a bridge between the applications and the data sources.

2.2.2 Components of ODBC

The ODBC architecture is shown in Figure 2.2-1. There are four components in the architecture:[10][12]

- Application - Calls the ODBC API functions to issue SQL query and retrieve data.
- Driver Manager - Loads the ODBC drivers and direct function calls to them.
- Driver - Processes the ODBC API function calls, submits the SQL query to the data source and returns result back to the application. If necessary, performs conversion of the calls and SQL statements into the data source syntax.
- Data Source - Includes the desired data itself, and its associated operating system,

DBMS, and corresponding network platform (if any).

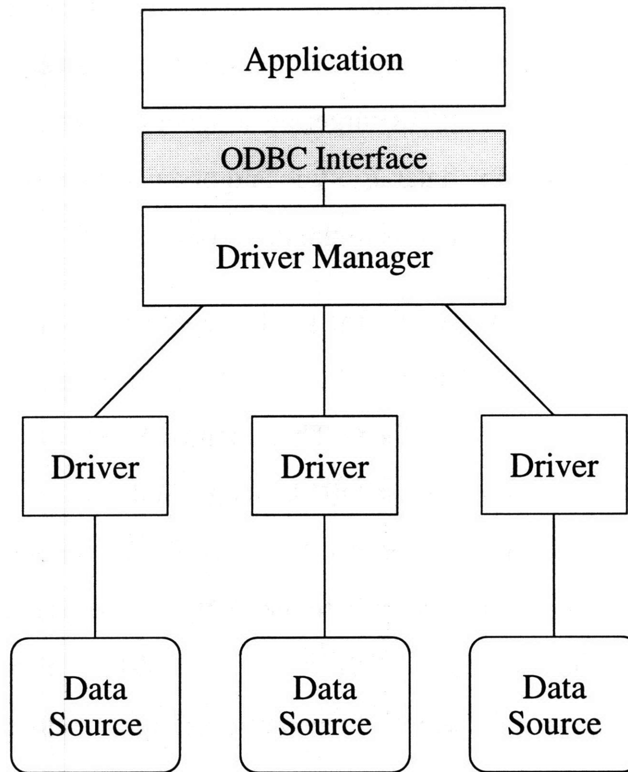

Figure 2.2-1: ODBC architecture

ODBC Drivers

There are two types of drivers in the ODBC world -- single-tier and multiple-tier. Single-tier refers to a driver that processes both the ODBC calls and the SQL statements. The driver directly processes the SQL statement and retrieves data from the data source. Multiple-tier drivers, on the other hand, only process the ODBC calls. They do not process the SQL statement; instead, they ship the SQL statements to the data source and let the data source process the SQL requests. Results from the data source are passed back to the driver, which in turn returns to the application.

ODBC defines conformance levels for drivers in two areas: the ODBC API and the ODBC SQL grammar (which includes also the ODBC SQL data types).[10] Applications can easily determine if the driver supports its functionality according to the driver's conformance level. A driver can also be developed according to the conformance level,

without targeting to support a specific application. Further discussion on ODBC drivers is carried out in Chapter 3 and 4 -- the design and implementation of the Context Mediator-ODBC driver.

2.3 Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems.[1] HTTP has been in use by the World Wide Web since 1990.[1] The version which this section describes is HTTP/1.0.

The HTTP protocol consists of a number of parameters such as the HTTP version, Uniform Resource Identifier (URI), Product Token and so on. Specifically, a request message from a client to a server includes the method to be applied to the resource, the identifier of the resource, the protocol version in use, and the optional request header fields. The syntax of the HTTP/1.0 protocol request is shown as follows[1]:

Method <space> Request-URI <space> HTTP-Version <carriage return line feed>

Request Header (Optional)

Note that the first three parts of the protocol have to be within the first line of the message. The optional request header lies in the second line. An actual example of the HTTP protocol is shown in Figure 2.3-1. Notice in the figure, there are two protocols of different syntax. The first one uses the absolute-URI in the request-URI. This can only be used when the request is being made to a proxy. The most common form of request-URI, however, is in the form of absolute path, in which the request is sent directly to the origin server or gateway (The bottom sample in Figure 2.3-1). The client has to create a TCP connection on port 80 (common HTTP port number) of the host before sending the request. The Context Mediator-ODBC driver will use this second type of protocol to request information from the Context Mediator. Refer later chapters for details.

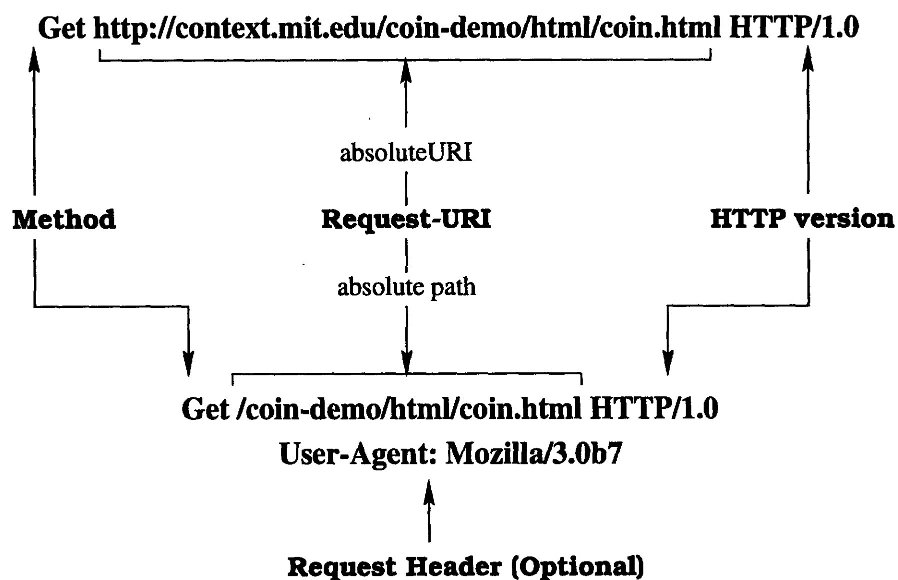

Figure 2.3-1: Examples of HTTP/1.0 protocol

The three essential building blocks for the development of the ODBC driver have been introduced in the above three sections. The following two chapters will described the design and implementation of the Context Mediator-ODBC driver.

Chapter 3

Design of the ODBC Driver

3.1 Role of the Driver

As shown in Figure 3.1-1, the driver is the middleware between the ODBC compliant applications and the data sources. Each driver's role differs from one another. A single-tier driver processes the SQL statement and retrieves data from the data source; while a multiple-tier driver only processes the ODBC calls and passes the SQL statement to the data source. However, all drivers still handle the basic ODBC function calls of making the connection to the data sources, executing the SQL queries, fetching data from the data sources, and returning data to the client applications in the appropriate data type.

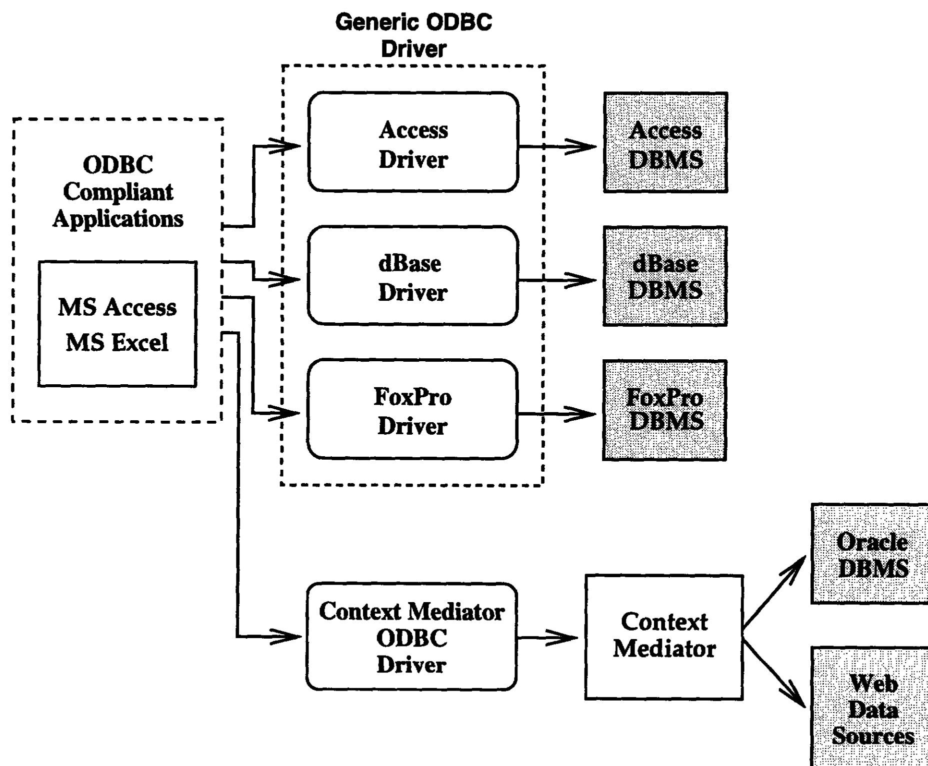

Figure 3.1-1: Overview of the ODBC Framework

The Context Mediator-ODBC driver belongs to the multiple-tier driver category. That means the Context Mediator-ODBC driver process the SQL statement and then sends the processed SQL statement to the data source to retrieve the data. The ODBC compliant applications submit SQL statements to the Context Mediator-ODBC driver. The driver then translates the SQL statements into the Context Mediator syntax, retrieves the results from the Context Mediator, processes the results, and finally sends the data back to the client applications. The Context Mediator-ODBC driver is designed to treat the entire Context Interchange System as one data source. From the Context Interchange framework perspective, the driver and the ODBC compliant applications together can also be regarded as the new front end of the Context Interchange System.

3.2 Structure of the Driver

In this section, the structure of the Context Mediator-ODBC driver will be discussed. First, the general architecture of the driver will be introduced. Then the internal data structure design of the driver will be discussed.

3.2.1 General Architecture

There are three fundamental operations that an ODBC driver carries out - connecting to the data source, executing the SQL statement, and retrieving results from the data source. Since there are about one hundred different ODBC API function calls in the ODBC family, organizing the driver according to the functionality of each function call is a natural choice for designing the ODBC driver. As shown in Figure 3.2-1, the Context Mediator-ODBC driver is divided into eleven parts, where each part corresponds to certain functionality of the driver.[10] Detailed description of the role of each part of the driver follows.

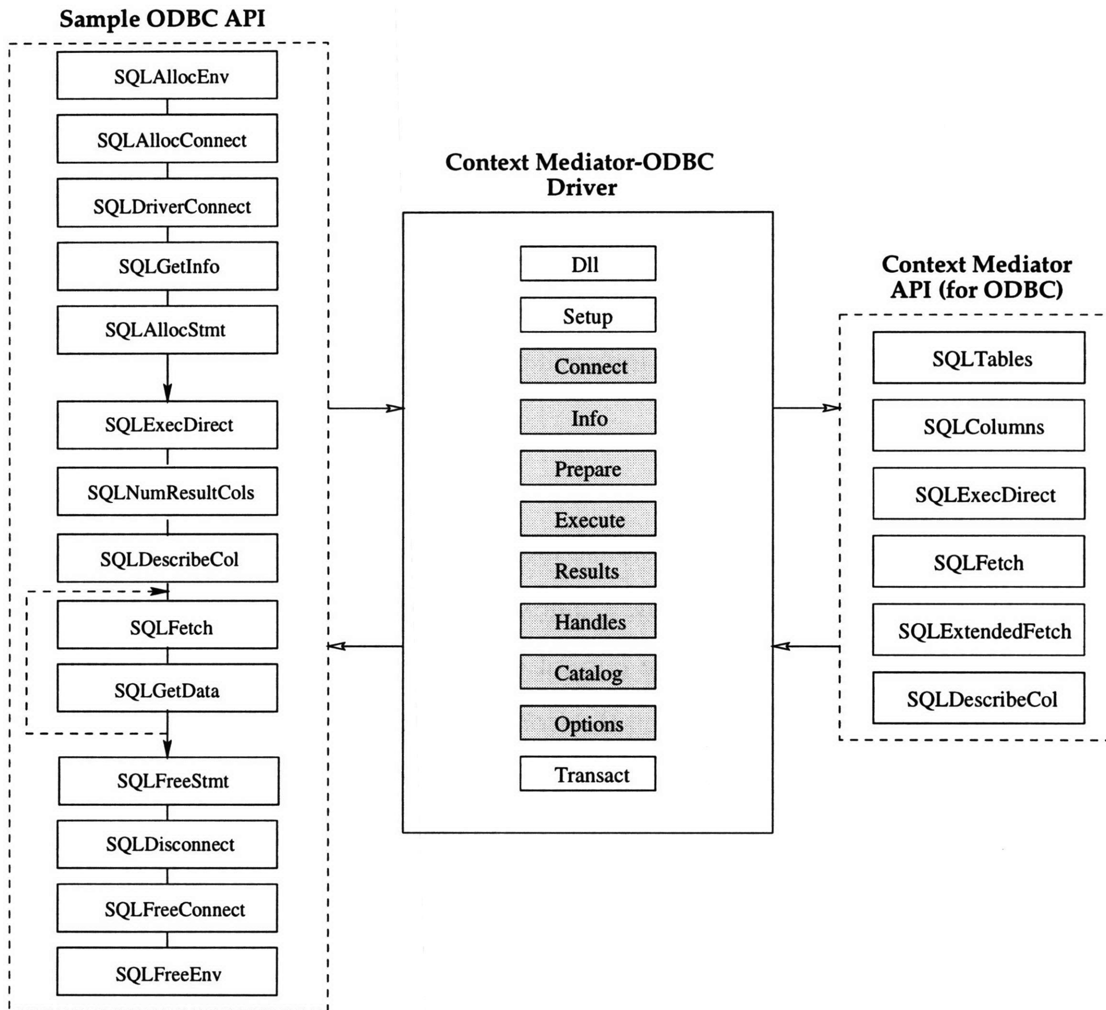

Figure 3.2-1: General Structure of the Context Mediator-ODBC Driver

Dll

As mentioned in Section 2.2.1, ODBC drivers are dynamic-link libraries which applications invoke to gain access to a particular data source. The role of the Dll part is to construct the driver into a DLL. Since this part of the driver is standard to all ODBC drivers, further discussion of how to setup the DLL will not be carried out in this paper. For further information, refer to [10] and the Microsoft ODBC Software Development Kit (SDK).

Setup

The Setup part of the driver is responsible for interaction with the ODBC installer. The ODBC installer functions in setting up the ODBC driver onto the client machine. It also allows the driver to control the management of the data sources. The Setup does not directly modify the ODBC ".ini"[4] file which the ODBC driver manager depends upon; instead, it links with the ODBC installer library which in turn modifies the ".ini" file. Note that the code of this part of the driver comes directly from the ODBC Software Development Kit. Again, for further information about Setup, see [10] or the ODBC SDK.

Connect

The role of the Connect part is to establish the connection between the driver and the data source, i.e. in this case, between the Context Mediator-ODBC driver and the Context Mediator. Since the front end of the current Context Interchange System prototype is a web browser, all interactions between the front end and the middleware of the System is based on HTTP protocol. Hence, communication between the ODBC driver and the Context Mediator is also HTTP protocol based. Therefore, the main role of the Connect part of the driver is to establish the HTTP window socket connection between the client computer (where the ODBC driver is located) and the Context Mediator. The Connect part also allocates storage space for the ODBC environment and connection using the underlying internal data structure. The storage is used to store information about the environment and the connection. (A more thorough description of the design of the internal data structure of the Context Mediator-ODBC driver is shown in Section 3.2.2.) The handles to these storage areas are returned back to the application, where the application uses them to identify its level of actions to the driver. For discussion of handles, see Section 3.2.2.

Info

The Info part mainly deals with information about the driver and the data source. In other words, ODBC compliant applications ask for information not only about the driver, but

4. This is the file used by the ODBC administrator to identify the available ODBC drivers.

also the data sources. The two main ODBC API function calls that deal with such information are SQLGetFunctions and SQLGetInfo. The former indicates which conformance level (refers Section 2.2.2) the driver belongs to and what other functions it is capable of; while the latter gives back various kinds of information about the data source a well as the driver. Since the Context Mediator-ODBC driver is a single-tier driver, information about the Context Mediator (data source) is obtained from the driver as well. Please refer Chapter 21, 22 of [10] for further information.

Prepare

When requested by the application, the driver has to allocate memory storage for information about the SQL statements. The handles of these storage areas are then returned back to the application. The Prepare part of the driver corresponds to these ODBC API function calls, which "prepares" memory storage within the driver to store information obtained later from the data source. Similar to the information about the environment and the connection, the information obtained is also stored in the internal data structure of the statement. (Again, please refer Section 3.2.2 for details.) The Prepare part also "prepares" the input SQL statements for execution. It is responsible for parsing and translating the SQL statement into the Context Mediator syntax, and storing it under the "Statement" internal data storage (Section 3.2.2) for execution.

Execute

The Execute part of the driver is also responsible for processing the SQL statement issued from the application and sending it to the data source. The processing part, however, depends on whether the SQL statement has already been processed or not[5] in the Prepare part of the driver. If indeed the processing is necessary, the SQL statement is parsed and translated into the appropriate syntax (a HTTP protocol) the same way as it is done in the Prepare part of the driver. Then the processed HTTP protocol is sent to the Context Mediator. (A more detailed discussion of the execution of the SQL statement will be

5. Different ODBC functions calls are used depending upon whether the function SQLPrepare in the Prepare part has been called. Refer to "Execute" part of Section 4.1 for further explanation.

presented in Section 3.3.2.)

Results

The Results part of the driver corresponds to a number of different retrieval methods of the result sets from the data source. The most important piece of the Results part is to retrieve the result set returned from the Context Mediator after the execution of the SQL statement. In addition, the Results part of the driver also obtains information about the results such as the number of columns and the data type of each column. It is also responsible for parsing the data and returning them to the application in the appropriate format. (Further discussion of the retrieval of data from the Context Mediator will be carried out in Section 3.3.2.)

Handles

The Handles part of the driver deals primarily with the internal data structure of the driver. It contains a number of function calls which are not exposed to the ODBC interface. That is, the Handles part includes routines used only internally by the driver. The main purpose of these internal function calls is to provide data abstraction between the handles from the client applications and the internal data structure pointers from the ODBC driver. In other words, client applications do not have any information about the internal data structure of the driver. Client applications only give the handles to the driver and get the handles from the driver. It is the responsibility of the driver itself to identify the correct internal data structure to which the handles correspond.

Catalog

The Catalog part of the driver includes the ODBC API function calls that are used to obtain information about the data sources system tables. It is used to identify the names and special privileges (if any) of the tables. It also provides information about each table and its corresponding column attributes by requesting information from the Context Mediator registry.

Options

The Options part of the driver is responsible for setting and getting options about the driver. There are two categories of options that are manipulated in the ODBC driver - the connection and the statement. And there are many different parameters associated with the connection and the statement respectively. For each specific option parameters, please refer to [10] for details.

Transact

The Transact part of the driver consists of only the SQLTransact function call. According to the input parameter, the driver will either commit or rollback all active operations within a connection or transaction. Refer to [10] for a more thorough description.

3.2.2 Internal Data Structure

As mentioned in the previous section, the application interacts with the ODBC driver through the use of handles. Handles are pre-defined by ODBC to identify the storage for different information. There are three kinds of handles: environment handles, connection handles, and statement handles. An application uses a single environment handle; it must request this handle prior to connecting to a data source. The application must also request a connection handle prior to connecting to a data source. Each connection handle is associated with an environment handle; but an environment handle can have multiple connection handles associated with it. Similarly, a connection handle can have multiple statement handles associated with it; but each statement handle can only correspond to exactly one connection handle. (Figure 3.2-2) Notice that the statement handle must be requested prior to submitting SQL requests.

The internal data structure of the Context Mediator-ODBC driver is designed for storing of information for the statement, connection, and environment. Each handle corresponds to the pointer to the data structure of either the environment, connection or statement. The environment structure includes a list of pointers to its associated connection structure, while the connection structure in turn has a list of pointers to all the

statements in that connection. The following paragraphs will examine the structures of the environment, connection and statement in more depth.

Environment

The environment structure is the simplest of the three internal data structures in the Context Mediator-ODBC Driver. An environment handle is associated with one environment structure. The environment structure in turn has a pointer back to the handle. Otherwise, the driver will not be able to hand the corresponding handle back to the client application. In addition, the environment structure contains four other pointers which correspond to the first connection and statement structures, as well as two spare connection and statement structures. The two spare connection and statement pointers are used to locate those connection and statement structures which have just completed their actions. By keeping pointers to these unused structures, the driver is able to allocate storage more efficiently for new connection and statement requests.

Connection

The connection structure is not as simple as the environment structure. It does not merely include pointers back to the connection handle like the environment structure, but it also has its internal data structure for storing information about the connection. The essential part of the structure is the storing of the information of the windows socket opened on the client machine. Since the HTTP protocol is issued by the Context Mediator-ODBC driver to the Context Mediator, the HTTP window socket is stored whenever the connection to the Context Mediator is requested. This socket information is later used in the driver to send the actual HTTP protocol SQL statements to the Context Mediator. Moreover, “sockaddr_in” is also used as the storage of the socket information for the Context Mediator. Finally, an error message string is also present in the connection structure. It is used for storing error messages related to any error occurring during the connection process.

Statement

Similar to the environment and connection structures, the statement structure also has a pointer to its corresponding handle. Furthermore, the statement data structure includes a pointer back to the connection data structure. This is necessary because some ODBC API function calls under the statement handle need to retrieve information of the HTTP window socket stored in the current connection data structure in order to send the HTTP protocol. The rest of the data structures are used for storing all other information about the current statement.

Once the SQL statement is sent to the driver, the driver parses it and translates it into the appropriate syntax (HTTP protocol string) for execution. There are three different calls to the Context Mediator and the statement data structure includes three such storage strings for these HTTP protocols. Second, the result that is returned back from the Context Mediator is written to a text file because the result set is of unknown size. That is, instead of storing the result in memory (which might not be sufficient because of the size of the result set), writing the result in a text file is the most robust way. Thus, the statement structure of the driver stores the "result_file" name as well. Third, owing to the fact that the result sent back from the Context Mediator is in HTML format, the driver has to parse the result set in order to return data to the client application in the appropriate format. Hence, the statement structure is designed to have two strings holding the header of the result and the current row of the result respectively. Moreover, an array of "pos_pair" structure is designed in such a way that it indicates the starting point of the actual data in the string and its actual length. Fourth, the client application can pre-allocate memory storage for the data before fetching the result. The driver thus includes a structure called "bind_col" to accommodate this operation. Fifth, the statement structure needs to identify what kind of result set is the current one in order to perform the appropriate retrieval. Finally, the statement structure also has an error message string for storing any error message about an error occurring in any statement related process. Each part the internal data structure of the driver and their relationships are shown in the following Figure 3.2-2.

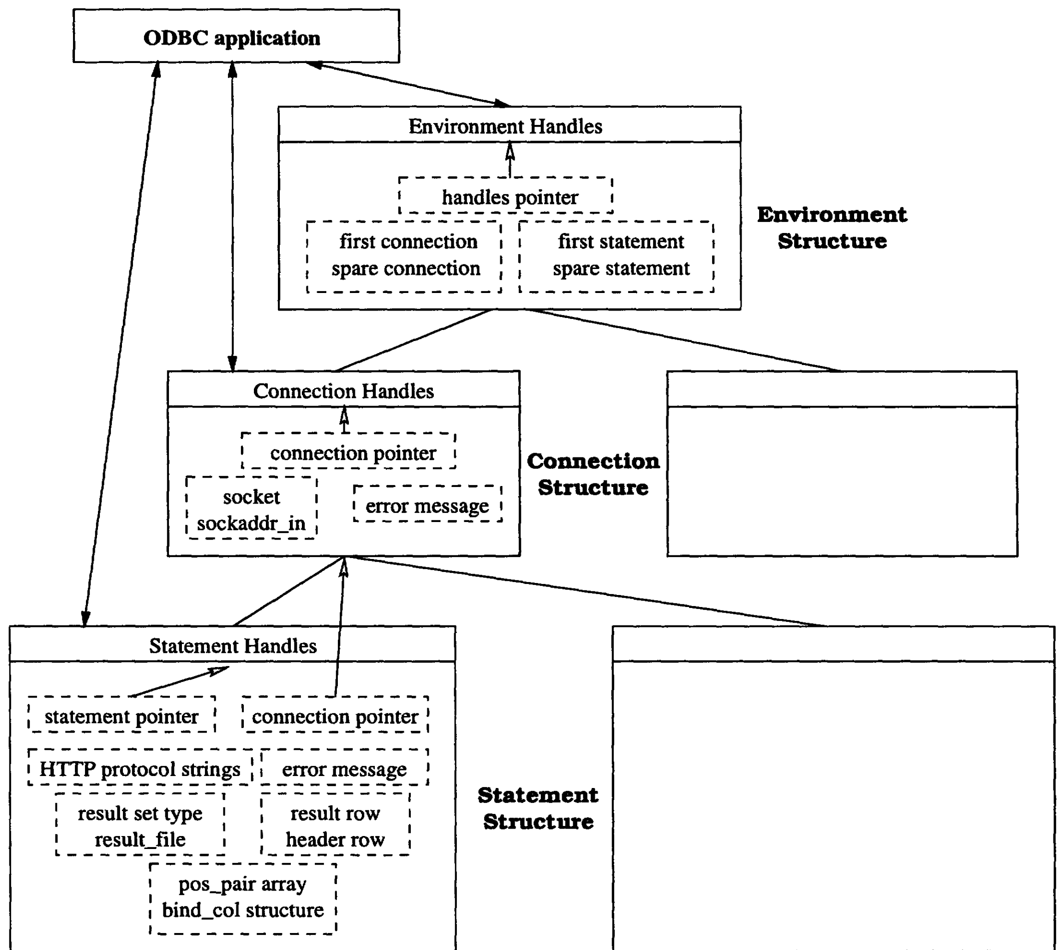

Figure 3.2-2: Internal data structure of the Context Mediator-ODBC driver

3.3 Interactions with the Context Mediator

In the previous section, the general architecture and the internal data structure of the driver was introduced. However, the previous section does not include any discussion about the underlying ODBC API function calls under each individual part of the driver and their corresponding role within the driver. In particular, it does not address any actual interactions between the Context Mediator-ODBC driver and the Context Mediator. In this section, the actual operations between the ODBC driver and the Context Mediator will be discussed.

3.3.1 Comparisons between ODBC API & the Context Mediator Interface

The basic functionality of each of the Context Mediator interfaces is summarized as shown in Table 2.1. This interface, as described in [6], mimics some of the basic ODBC function calls. Likewise, the functionality of each Context Mediator interface call is the same as its corresponding one in ODBC. The Context Mediator interface is designed to allow other front end applications to gain access to the system. The underlying activities behind the scene of the interface will not be discussed any further in this paper.

The Context Mediator interface is a subset of the full sets of ODBC API function calls. It only has eight out of the hundred function calls of the ODBC API. Yet the Context Mediator interface does possess the two essential function calls of the ODBC API, *SQLExecDirect* and *SQLExtendedFetch*. The former is the execution of the SQL statement, while the latter is the retrieval of the entire result set after execution of the SQL statement. The following section will explain how the actual operations to the Context Mediator are designed and carried out within the Context Mediator-ODBC driver.

3.3.2 Operations between the ODBC Driver & the Context Mediator

There are a number of ODBC function calls of the Context Mediator-ODBC driver that have direct interactions with the Context Mediator. These are function calls that send HTTP protocol requests and wait for the results to return from the Context Mediator. Notice that for each of the function calls to manipulate the HTTP protocol, the HTTP socket has to be present. The HTTP socket is established in the connection to the Context Mediator, which is stored under the connection data structure. In order to allow function calls under the statement handle to get this HTTP socket from the connection data structure so as to send the HTTP protocol to the Context Mediator, it is necessary for the statement data structure to contain a pointer to its corresponding connection structure (Section 3.2.2).

As mentioned before, the basic functionality of each Context Mediator interface call is more or less the same as the corresponding ODBC function call. However, the Context Mediator interface functions are HTTP protocols. So the job of the Context Mediator-ODBC driver is to translate requests from the client application into the appropriate HTTP protocol. The remainder of the section will describe how the ODBC function calls operate with the Context Mediator. The three fundamental processes of making connection, sending request, and retrieving result will be carried out.

Making a Connection

In establishing the connection, the driver does not physically connect the Context Mediator-ODBC driver to the Context Mediator. Instead, the design of the driver is to setup the HTTP window socket on the client machine to make it ready for sending HTTP protocol to where the Context Mediator is situated. In doing so, later executions of the HTTP protocols will not require any socket establishment. The executions can therefore use the already established socket directly. Hence, the driver is more efficient in the sense that it no longer needs to establish the socket connection every time an HTTP protocol is sent.

Sending Request

The sending of the SQL statement involves two major states. The first state is to transform the input SQL statement into the HTTP protocol as well as into the appropriate Context Mediator interface syntax. The second state is to physically send the protocol to the Context Mediator. Similarly, for retrieving information about a specific registered data source, the driver also needs to go through two states, the setup of the HTTP protocol and the execution of the HTTP protocol. Depending upon the ODBC function call, the setup process of the HTTP protocol is similar. In Table 3.1 below, two examples of the ODBC function calls and their corresponding Context Mediator interface function are shown. Note that the first SQLExecDirect is the one that translates the SQL statement[6], while the

6. The SQL query is: *select discaf.company_name, discaf.net_sales from discaf where discaf.company_name = 'DAIMLER BENZ CORP'*

SQLDescribeCol is the one that retrieves information (the data type) about the data source.

Table 3.3-1: ODBC Function Calls and their corresponding Context Mediator Interface

ODBC Function	Context Mediator Interface
SQLExecDirect	http://context.mit.edu/cgi-bin/Inc/Dev/router.cgi? ODBC=SQLExecDirect &datasource=Disclosure &sql= select+discaf.company_name,discaf.net_sales+ from+discaf+ where+discaf.company_name+%3D'+DAIMLER+BENZ+CORP'
SQLDescribeCol	http://context.mit.edu/cgi-bin/Inc/Dev/router.cgi? ODBC=SQLDescribeCol &datasource=Disclosure &table=DiscAF &column=COMPANY_NAME

However, the second state of the process -- sending the actual HTTP protocol to the Context Mediator -- is the same for all the ODBC function calls. The driver makes use of the HTTP window socket established during the connection phase and the built-in "winsock" function "send" to do the execution.

Retrieving the Result

The process of retrieving results also includes two stages. The first stage is to apply the built-in winsock function "recv" and the already established HTTP window socket to receive the result from the Context Mediator. In the case that the driver fetches the result after an SQL statement execution, the SQLExtendedFetch function of the Context Mediator interface is called and the entire result set is sent to the ODBC driver. The second stage is the parsing of the result. Since the result returned from the Context Mediator is in HTML format, the driver has to parse the result so that the actual data can be shipped to the client application. In order to accomplish that, as described in the statement internal data structure in Section 3.2.2, the driver uses the "pos_pair" structure to pre-identify the meaningful data position and length of the result before any actual

fetching. Note that the "pos_pair" internal data structure is solely used for parsing result sets from the execution of the SQL queries. Other results about the data source information are parsed directly within the function call since the results are much simpler in format. As in SQLDescribeCol, a simple parser is used to parse the returned HTML formatted result to obtain just the data type string[7]. For the example in Table 3.3-1, the data type of the column "COMPANY_NAME" in table "DiscAF" is returned.

7. The data type is obtained from the registry information on the Context Mediator side.

Chapter 4

Implementation/Scenario

This chapter is divided into four sections. The first three sections deal with the implementation of the Context Mediator-ODBC driver. In the first section, the implemented ODBC function calls will be described in detail. The second section will address the development of the second prototype of the Context Mediator-ODBC driver. In the third section, a few primary algorithms employed by the driver and certain tradeoffs in the design and implementation of the driver will be discussed. Finally, the fourth section of the chapter uses three scenarios to illustrate how the ODBC compliant applications, the Context Mediator-ODBC driver, and the Context Mediator interact with each other.

4.1 Implemented ODBC Function Calls

As mentioned in the previous chapter, there are about a hundred different ODBC function calls in the ODBC framework. Due to the limitation of time in this thesis project, the Context Mediator-ODBC driver is not a fully implemented driver. That is, not all of the hundred ODBC function calls are implemented. Since the targeting ODBC compliant application is Microsoft Excel, the Context Mediator-ODBC driver is designed and implemented to handle all the necessary ODBC function calls required by Excel. The following will describe each of the crucial implemented function calls and discuss any important issue related to its implementation. Figure 4.1-1 below shows the ODBC function calls in the three major parts of the driver (Connect, Prepare/Execute, Results) that have actual interactions with the Context Mediator. A complete list of the implemented and non-implemented ODBC API function calls is shown in Appendix C.2.

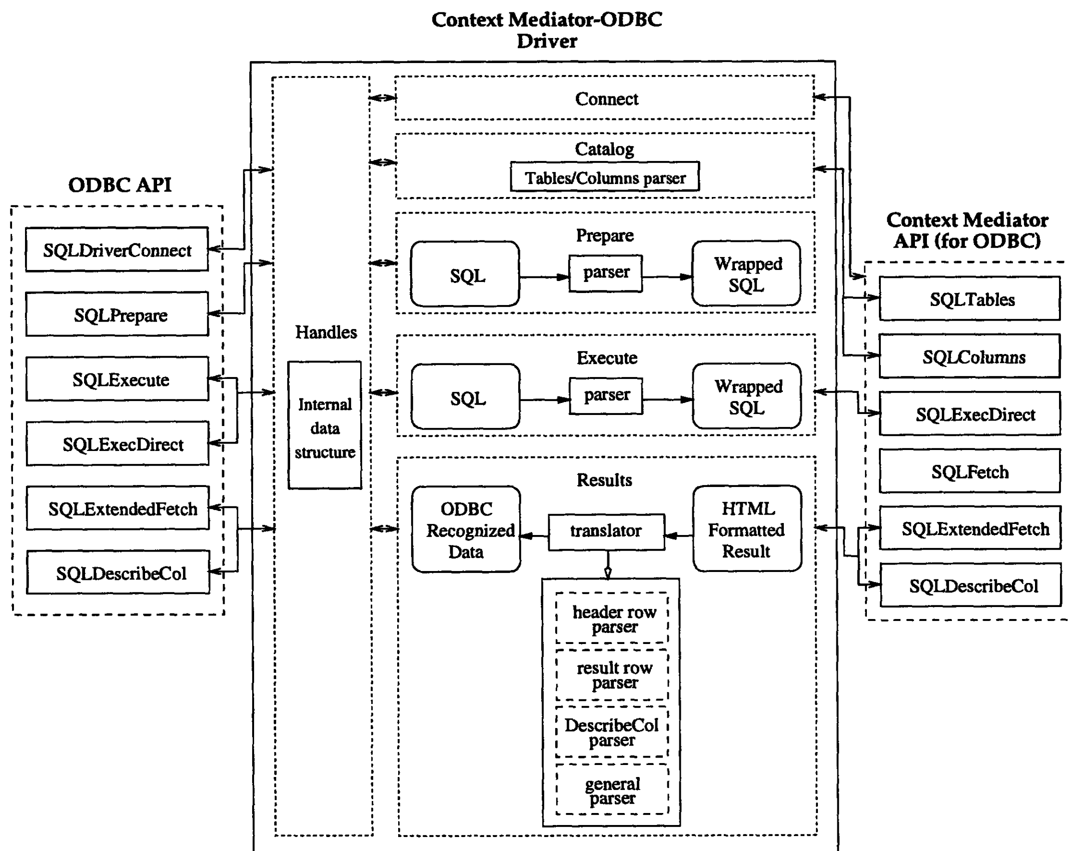

Figure 4.1-1: Essential components of the ODBC driver

Connect

In the Connect part of the driver, the SQLAllocEnv, SQLAllocConnect, SQLFreeEnv and SQLFreeConnect are implemented. These four functions are crucial in the ODBC driver since they are responsible for the allocation and deallocation of the storage space of the internal data structure. In particular, the SQLAllocEnv and SQLAllocConnect make use of the internal functions under the Handles package of the driver to initialize the storage. In contrast, SQLFreeEnv and SQLFreeConnect deallocate (or free) the storage space also using the internal functions under the Handles package.

In addition, the Connect part also includes the function SQLDriverConnect. Its role in the Context Mediator-ODBC Driver is to setup an HTTP socket connection between the client computer and the Context Mediator. The implementation of the HTTP window socket connection is based on [7]. Once the HTTP windows socket is established, the socket is stored under the Connection internal data structure for later use. SQLDisconnect

is also implemented. It is responsible for disconnecting the client application from the targeting data source. In the Context Mediator-ODBC driver, SQLDisconnect closes the HTTP windows socket and clears the storage of the socket.

Prepare

The Prepare part of the driver consists of another level of storage allocation and deallocation function calls. SQLAllocStmt and SQLFreeStmt are implemented to handle the storage space of the Statement level. SQLAllocStmt and SQLFreeStmt use the Handles package internal functions to setup and clear the storage space respectively. Note that SQLAllocStmt also initializes the statement data structure, since the data storage is used for storing information during the execution and fetching processes. In particular, the HTTP protocol SQL strings, the name of the result_file[8], result header and result row strings, the pos_pair array, the bind_col structure, the result set indicator, as well as the error message string are all initialized in the SQLAllocStmt function. On the other hand, SQLFreeStmt, depending on the input parameters from the ODBC application, either deallocates the storage entirely (including the removal of the result file), unbinds the memory storage between the application and the driver, or re-initializes the Statement structures (which also removes the previously used result file).

In addition, SQLPrepare is implemented in the Context Mediator-ODBC driver. Its sole function is to parse and translate the input SQL statement into the Context Mediator syntax, and then store it in the Statement structure. For example, the following SQL query:

select networth.Last from Networth.networth where networth.Ticker = 'IBM'

which queries the latest stock price of IBM from the Networth web site, is translated into an HTTP protocol SQL statement (called wrapped SQL statement)[9]:

Get /cgi-bin/Inc/Dev/router.cgi?ODBC=SQLExecDirect&datasource=Networth&sql= select+networth.Last+from+networth+where+networth.Ticker+%3D+'IBM' HTTP/1.0

User-Agent: Mozilla/3.0b7

---

8. The result_file name is a temporary file name randomly generated by the driver.
9. Please refer Section 4.3.1 for the parser procedure, algorithm and analysis.

Since the Context Mediator is not a proxy server, the HTTP protocol syntax used is the second one shown in Section 2.3. The window socket established in SQLDriverConnect is already an HTTP socket with the target referring to the hostname (context.mit.edu) of the Context Mediator, as such, the protocol only needs the absolute path in the request-URI. While building the "SQLExecDirect" HTTP protocol, two other HTTP protocols, namely SQLExtendedFetch and the partial SQLDescribeCol are also built. The SQLExtendedFetch call is the one that actually retrieves the result set from the Context Mediator after the execution of the SQL query; while SQLDescribeCol is used later to obtain data type of the returned column.

Execute

The Execute part has two function calls implemented: SQLExecute and SQLExecDirect. Note that SQLPrepare has to be called prior to SQLExecute. It is because SQLPrepare is used to "prepare" the input SQL query for SQLExecute to execute. There is no input parameter in SQLExecute that allows the application to input the SQL statement. Therefore, upon execution, SQLExecute retrieves the HTTP window socket information from the Connection data structure to setup the actual socket, applies the built-in winsock function "send" to send the wrapped SQL statement HTTP protocol stored previously in the statement structure to the Context Mediator, uses the "recv" winsock function to retrieve any result from the Context Mediator, and finally stores the result in the temporary result string. It then writes the result string to the randomly generated text file indicated by "result_file". (Refer to Section 4.3 for explanation) Notice that the above process actually includes two HTTP protocols. The first set of "send" and "recv" is associated with the SQLExecDirect interface of the Context Mediator, of which the query is executed within the Context Mediator. Only an "OK" flag is sent back to the driver. It is not until the second set of "send" and 'recv" of the "SQLExtendedFetch" Context Mediator interface is executed has the actual result set for the SQL query been sent back to the driver and written to the result file.

On the other hand, SQLExecDirect is a combination of the functions of SQLPrepare and SQLExecute altogether. That means SQLPrepare is not called prior to SQLExecDirect. SQLExecDirect, which accepts SQL query as the input parameter, parses and translates the input SQL statement as in SQLPrepare, and then sends the wrapped SQL statement HTTP protocol to the Context Mediator and receives result from it as in SQLExecute described above.

Results

The ODBC function calls implemented in the Results part of the driver are described as follows.

SQLNumResultCols: This is the function that finds out the total number of columns in the result set. To do so, the function first parses the result_file text to retrieve the header row by searching for the table tag "<TABLE BORDER>" in the HTML formatted result set. (Please refer Appendix A for the sample result file) Then it parses the header row (which is the next line after the table tag) with the use of the pos_pair structure array to store each column name position and its corresponding length for later use in SQLColumns and SQLDescribeCol. In the mean time, it also counts the number of columns in the result set, which is the objective of this function.

SQLDescribeCol: This function is used to describe the data type of a particular column. Given the column number as one of its parameters, SQLDescribeCol fetches the column name of the specific column from the pos_pair array structure and concatenates it to the partial SQLDescribeCol HTTP protocol created in SQLPrepare or SQLExecDirect. Again, winsock functions "send" and "recv" are used to handle the HTTP protocol. As the result from the Context Mediator is returned, SQLDescribeCol parses the result set to find the data type and translates it into the corresponding ODBC SQL data type.

SQLBindCol: Given the column number and storage space from the application, this function "binds" the application storage space with the corresponding data structure of the

driver (called aBind_col). In other words, the storage space of the application is pointed to this internal data structure. Thus as the real data is stored in the aBind_col structure, it is actually equivalently to sending the data back to the application since the application storage space refers to this driver storage too.

SQLFetch: This function does not only fetch the result set after the execution of the SQL statement, but it also fetches different kinds of result sets such as from SQLTables, SQLColumns, SQLGetTypeInfo, etc. The fetching of the latter category of result sets are similar to the former one, but with a much simpler parser. Thus the focus of this section is not on the implementation of this portion of SQLFetch. Instead, it will focus on how to retrieve data from the much complicated result set of SQL query.

An indicator called "result_set_row" in the Statement structure is used to represent the current row number of the result set. By locating the table header row of the result set and using the "result_set_row" indicator, the current result row of data is found. Then, SQLFetch parses the entire result row and stores the information of the actual data position and its length in the pos_pair array structure. If SQLBindCol has been called, each data of the current row of result is stored in the aBind_col structure of the driver, where the application has previously "bind" its storage space to this structure as described in the above "Prepare" section. Note that SQLFetch also checks to see whether the current row of the result is the end of the table or is invalid, and then returns the end of data or error parameter value accordingly. In addition, if SQLBindCol has not been called, the current row of result will be kept in the result_row string of the Statement structure for later retrieval through SQLGetData.

SQLGetData: This function, upon the input of the column number, uses the pos_pair array information to retrieve the required data from the result_row string which holds the current row of the result. The retrieved data is then returned to the application.

Handles

The handles part contains the so called Handles package internal functions solely designed

for dealing with the internal data structure and the handles from the client application. It has the function CEnv_New for allocating memory storage of the new environment structure, CEnv_Delete for removing the environment storage, CEnv_FromHandle and CEnv_GetHandle for referring the environment handle to the pointer of the corresponding environment data structure and vice versa. Similarly, same sets of functions for the connection and statement structures are implemented as well. In addition, there are other sub-routines implemented that deal with the initialization and cleanup processes for each structure.

Catalog, Info, and Options

These three parts of the driver consist of functions that deal with information and functionality of the data source and the driver. For function calls in the Info and Options parts, they are primarily functions with a fixed number of parameter values related to each specific functionality of the driver. Hence, the implementation is straight forward in the sense that the functions are implemented so that they return the appropriate pre-defined ODBC value accordingly.

Functions in the Catalog part, which consists of function calls such as SQLTables, SQLColumns and so on, mainly deal with the data source. Similar to the SQLDescribeCol function, HTTP protocol is sent to the Context Mediator "registry" to retrieve available table names and column names of the specified data source and table respectively. Note that similar to SQLExecute, the returned result set is written to the file for later retrieval. Again, refer to Section 4.3 for justification.

Dll, Setup, and Transact

ODBC function calls in these three parts of the driver will not be discussed. Dll and Setup are implemented according to the ODBC Software Development Kit and they are outside the scope of this thesis. The transact part contains only a single function call SQLTransact which is not yet implemented in the Context Mediator-ODBC driver.

4.2 Migrating to the Second Prototype

As the Context Interchange System has evolved into the next prototype, so has the Context Mediator-ODBC driver. Modifications have been made in order for the driver to support the new and improved Context Mediator. However, the use of the new prototype temporarily pushes the ODBC driver to abandon the original Context Mediator ODBC interface and some of its functionality. Section 5.2 will address the issue and discuss the impact and possible re-engineering aspects on the Context Mediator interface in more detail. This section will focus only on the changes made in the new design and implementation of the driver.

In general, there are not many changes on the ODBC driver due to the new Context Mediator prototype. The HTTP windows socket setup and the parsing of the returned result set remain the same as the first prototype. Changes are made on the parsing and translating process of the input SQL statement, and the process of sending HTTP protocol and receiving the returned result.

First of all, for the original driver prototype and Context Mediator interface, it is not possible for the application to specify the desired "context" in the input SQL query. This is a major limitation for the ODBC development of the Context Interchange System. In order to solve the problem, the new Context Mediator prototype allows direct execution on the Context Mediator underlying engine in which the "context" of the query can be specified as a flag in the request-URI of the HTTP protocol. (However, in doing so, it violates the abstraction level on the Context Mediator. A proper solution will be suggested in section 5.2) Thus the parser and translator of the Context Mediator-ODBC driver are modified such that they can parse and translate the new SQL query which includes the new "context" tag.

The second modification of the Context Mediator is that it is no longer necessary to issue two HTTP protocols (SQLExecDirect and SQLExtendedFetch of the Context Mediator interface) in order to execute and fetch data from the Context Mediator. The second prototype of the ODBC driver only sends one HTTP protocol of the wrapped SQL

statement to the Context Mediator and receives the result set directly after the call. This process is indeed much more efficient than the original process of two HTTP protocols.

The following example is the same SQL query as described in Section 4.1. It also gets the latest stock price of the ticker "IBM" from the Networth web site, but using the second prototype interface:

Get /coin/proto-0/client/eclipse-agent?query=
select+networth.Last+from+networth+where+networth.Ticker+%3D+'IBM'
&context=c_ws HTTP/1.0

User-Agent: Mozilla/3.0b7

4.3 Algorithms and Tradeoffs

This section will examine and analyze certain important algorithms employed by the Context Mediator-ODBC driver. Furthermore, tradeoffs of these algorithms and any other kinds of tradeoffs of the ODBC driver will also be discussed.

4.3.1 Algorithms Analysis

The first algorithm discussed is the parser and translator for converting standard SQL query into the HTTP protocol with Context Mediator syntax. The parser and the translator are employed in the SQLPrepare and SQLExecDirect function calls. When presented with the SQL statement, the parser first parses the SQL statement into three separate parts: "select" portion, "from" portion, and "where" portion. This process takes O(n) time, where n refers to the length of the SQL query. For each individual portion, the translator replaces all the "white space" by "+", and "=" by "%3D" by going through the string. For the "from" portion, the parser also extracts the datasource name (which is a required part of the Context Mediator interface syntax) from the string. The time taken for each of these processes is also in the order of O(n). The last step is to concatenate these three strings into one, and then concatenate this wrapped SQL query with other HTTP protocol parameters and the Context Mediator syntax parts. Since there are a fixed number of concatenation steps and each step takes constant time, the entire concatenation process is

in the order of O(1). Therefore, the entire process for the parser and translator altogether is considered as in O(n). Notice that the above algorithm uses a fixed number of temporary strings with length less than or equal to the length of the final HTTP protocol wrapped SQL statement. Thus, the memory usage is also in the order of O(n).

The second algorithm to be discussed is the way the result set parser parses the HTML formatted result set returned from the Context Mediator. Given that the result set returned from the Context Mediator is in Hypertext format, the parser exploits the HTML tag to locate the actual data position. In addition, it also makes use of an array of "pos_pair" structure to store the beginning position (iStart) and the length (iLength) of the actual data. Suppose the current result row or the header row is located and is stored as a string. By going through the current result row or header row once, the parser locates the beginning of the first data after a specific HTML tag and stores it in 'iStart'. Then it counts the number of characters of the current data until it reaches the beginning of another specific HTML tag. The length is stored in 'iLength'. The above process repeats and the second data beginning position and length is stored in the second element of the pos_pair structure array. The parsing continues until it reaches the end of the row. Since the algorithm goes through the string only once, the process again is in the order of O(n).

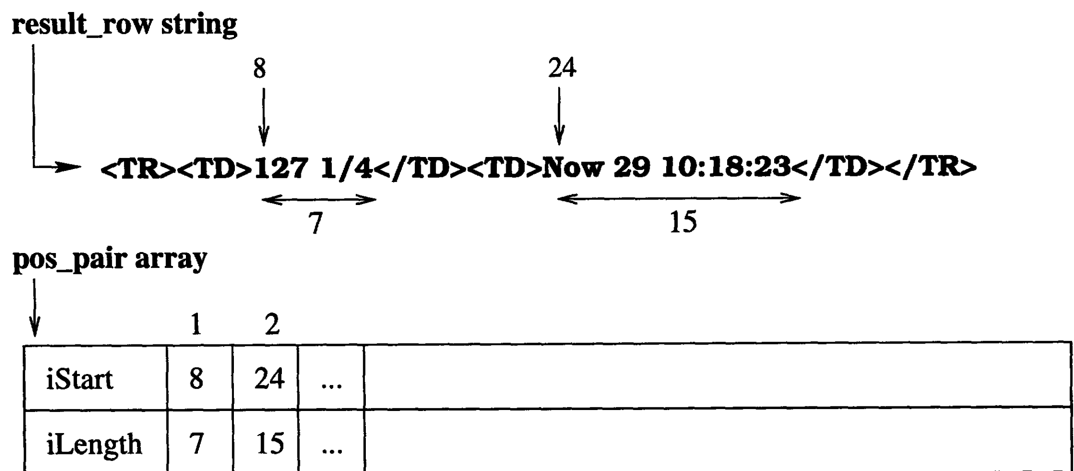

Figure 4.3-1: The "pos_pair" array structure for retrieving data

4.3.2 Tradeoffs

In the design and implementation of such a complicated driver, there are always different approaches in solving the problems and meeting the requirements. Tradeoffs between different approaches are inevitable. The following will discuss the pros and cons for two of the crucial design and implementation approaches in the Context Mediator-ODBC driver.

First of all, in the process of parsing the current result row or header row of the result set, the driver uses an array of pos_pair structure to locate the actual data position and length by scanning through the row string only once. By having the actual data position and length stored, later retrieval no longer needs to search through the string again, thus making the retrieval process simpler and more efficient. On the other hand, another method of retrieving data is to search through the string for each desired data. This is of course not as efficient as the previous method, especially when the string is long. But the advantage is that it does not need memory for storing the array of pos_pair structure. Nonetheless, the first method of using pos_pair to store the pre-located data is chosen over the second method. It is not likely for the driver to retrieve result set of more than twenty columns, thus the memory used by the array of pos_pair is not a limiting factor for the driver. The driver can afford the tradeoff of spending more memory to compensate the efficiency of the driver.

The second kind of tradeoff is exactly the contrary to the one described above. Here, the driver is designed to sacrifice efficiency by conserving memory and maintaining robustness. After sending the SQL query to the Context Mediator, the driver uses the winsock "recv" function to receive the result set back from the Context Mediator. In the current design, the entire result set is written to a text file stored in the hard disk of the client computer. Later retrieval of data from the result set is read from this text file. Of course, as one may argue, this is not efficient since writing and reading from memory is about six order of magnitude faster than writing and reading from hard disk. However, the justification of using disk storage over memory storage is that it is more robust. Since the result set returned from the Context Mediator may vary from a few rows to multiple

thousands or even hundred thousands of row, it is possible for the client machine to run out of memory. Furthermore, since reading and writing on disk is already in the order of millisecond, the driver should not suffer any noticeable speed deficiency relative to the network latency which is in the vicinity of a tenth of a second. Therefore, robustness is chosen over efficiency in this case.

4.4 Scenario of Retrieving Financial Data

In order to better understand the interaction between the ODBC compliant application, the Context Mediator-ODBC Driver and the Context Mediator, three examples are given below to illustrate the entire data retrieval process. Samples of retrieving financial data are shown in the following sections.

Table 4.4-1: Summary of the three scenarios given in this section

Scenario	ODBC Application	Context Mediator Prototype
Single-datasource	MS Query	First prototype
	MS Excel	Second prototype
Multi-datasource	MS Excel	Second prototype

4.4.1 Single-datasource query scenario (First & Second Prototypes)

There are a number of ODBC compliant applications in the current ODBC family, such as MS Excel, MS Access, MS Query, Powerbuilder, Lotus 1-2-3, Lotus Approach, and etc. There are two prototypes of the Context Mediator for single data source query, two examples are used to demonstrate the difference and capability of the driver. And two different ODBC compliant applications are employed to illustrate the two examples.

First Prototype Example

The first example demonstrates the capability of the first Context Mediator prototype. MS Query (a Query-By-Example (QBE) like application) is the chosen ODBC application. As

in Figure 4.4-1, the ODBC function calls and Context Mediator interface that have been used in the retrieving process are shown. Notice that the ODBC function calls are simplified such that it can be better understood by the readers. For details, refer to Appendix B.1.

In summary, the entire process can be divided into several stages. The first six function calls (refer to Figure 4.4-1) can be regarded as the "setup" phase. In particular, memory allocations of the three internal data structures and the setup of the HTTP window socket take place. Then, the two catalog function calls SQLTables and SQLColumns are called to retrieve information about the data source. The third and fourth phase are the query execution stage and result retrieval stage respectively. Wrapped HTTP protocol SQL statement is executed and the result set is returned (SQLExecDirect) in the third phase; while the result set is processed and sent back to the client application (SQLFetch) in the fourth phase. The fifth and last phase, which consists of the last four function calls, is the "termination" stage where all the memory storages associated to the above process is cleared. The following paragraphs will go through the process using MS Query as the example.

As MS Query is opened, the new query button is clicked. First it asks the user to choose the desired data source from a list of already installed ODBC supported data sources. In this case, the user chooses the one called Context Mediator-ODBC and the setup phase takes place. SQLTables is executed after the selection of the data source and a list of available table names is shown (Figure 4.4-2). The user is then asked to choose one or more tables from the list of tables. As soon as a table is chosen, SQLColumns is executed. Column names, or attributes of the selected table are displayed (Figure 4.4-3).

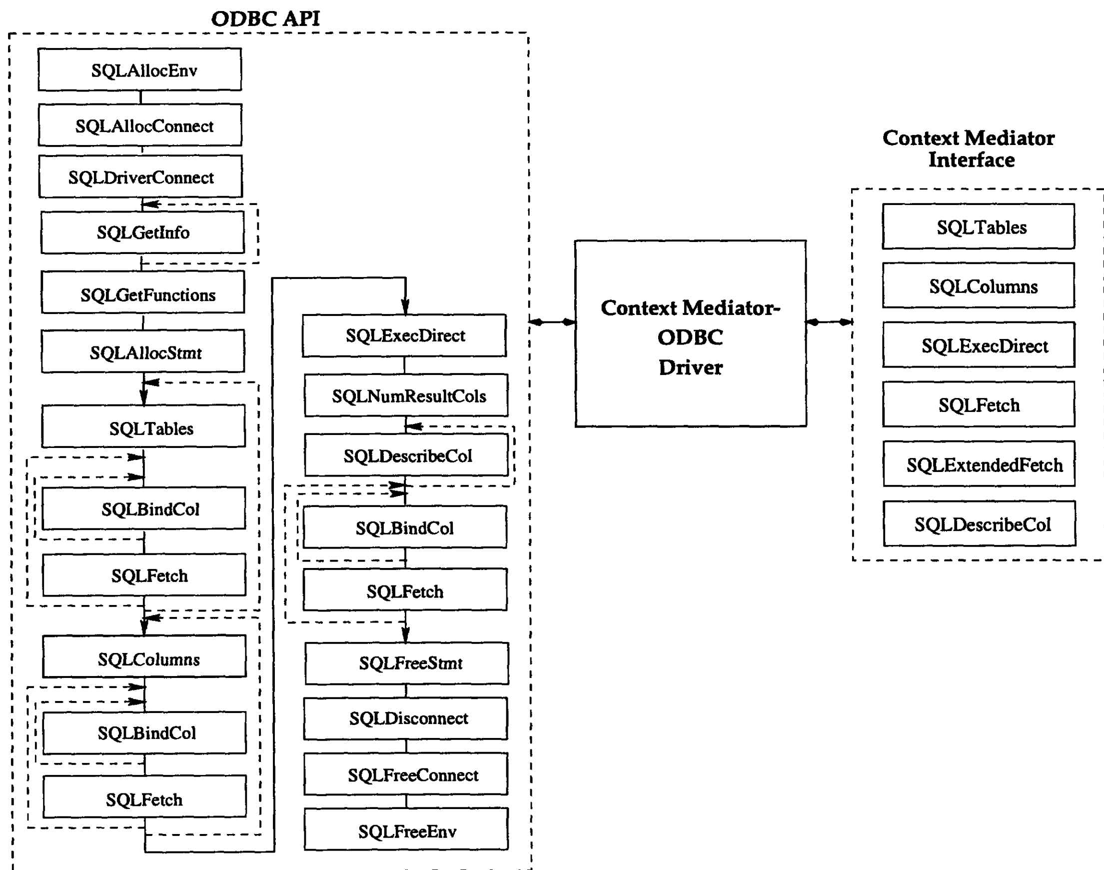

Figure 4.4-1: ODBC calls and Context Mediator interface used in MS Query

Users are then asked to input the SQL query. The same example of querying the latest stock price of IBM from the Networth web site as in section 4.1 is used. User types the same query into the SQL query box of MS Query (Figure 4.4-4), and the result is returned and displayed as shown in Figure 4.4-5. ("153 7/8" in the bottom of the figure is the returned stock price) This is the completion of the query execution and result retrieval stages. Finally, as MS Query closes, the "termination" phase is triggered and the memory storage will then be cleaned up.

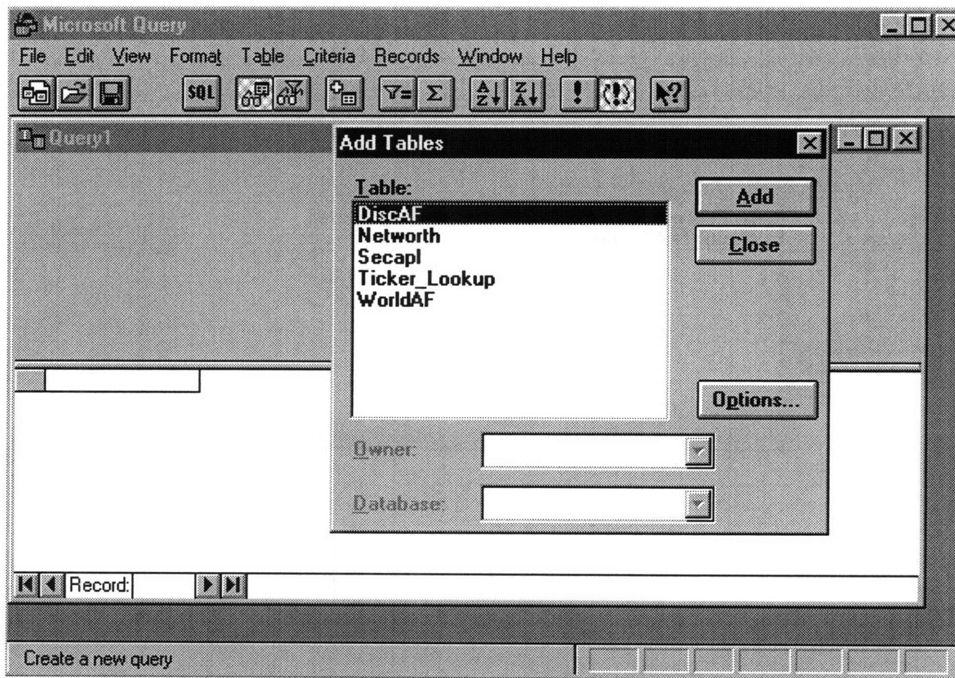

Figure 4.4-2: Table names are shown after SQLTables is called

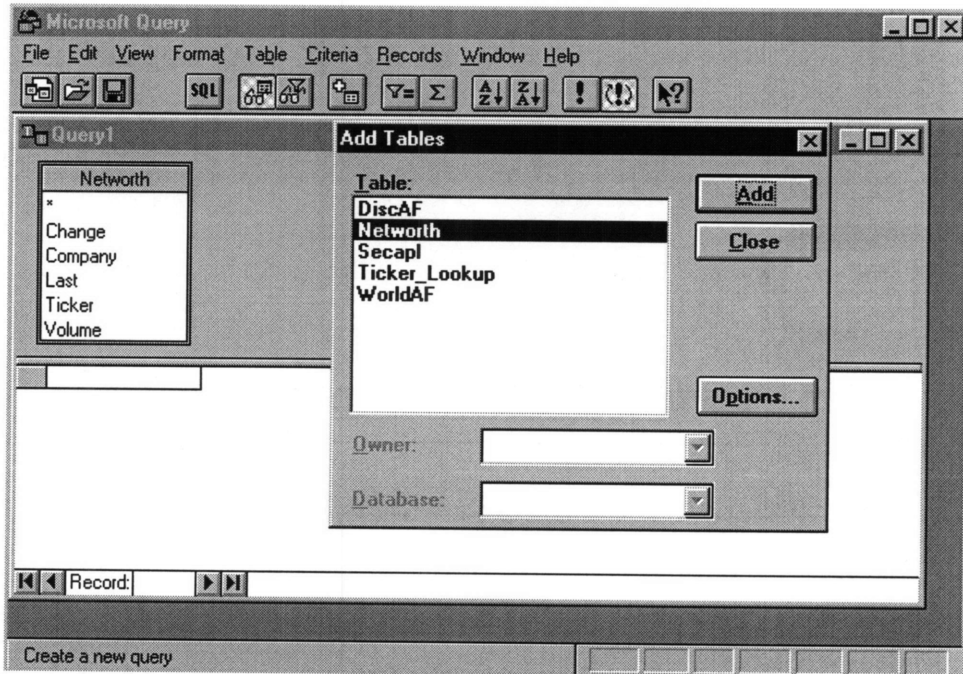

Figure 4.4-3: Column names of the chosen table is shown after SQLColumns

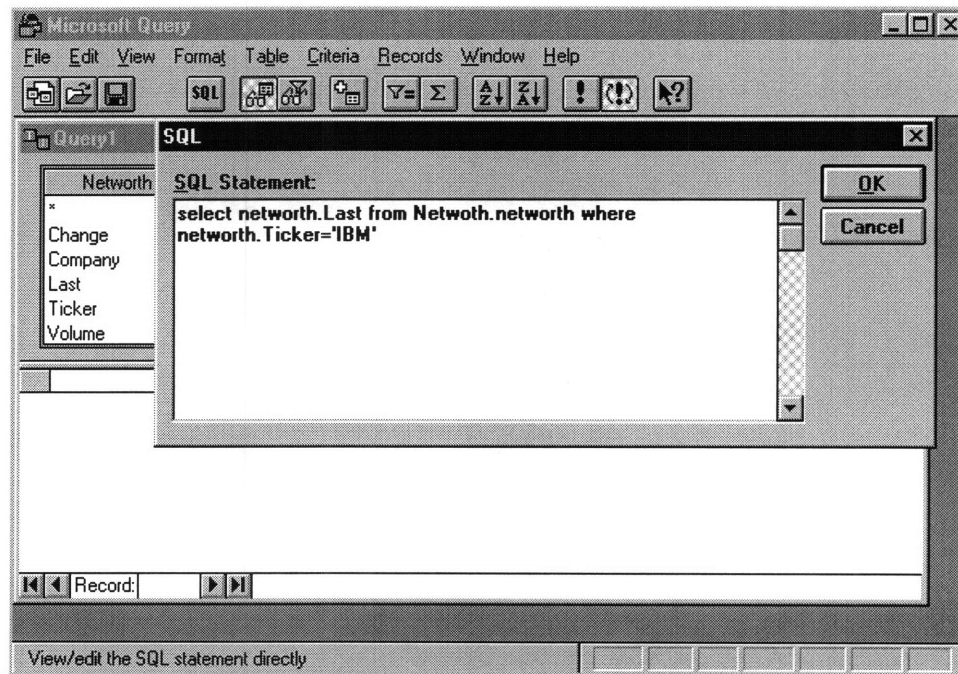

Figure 4.4-4: Inputting SQL query in the SQL box

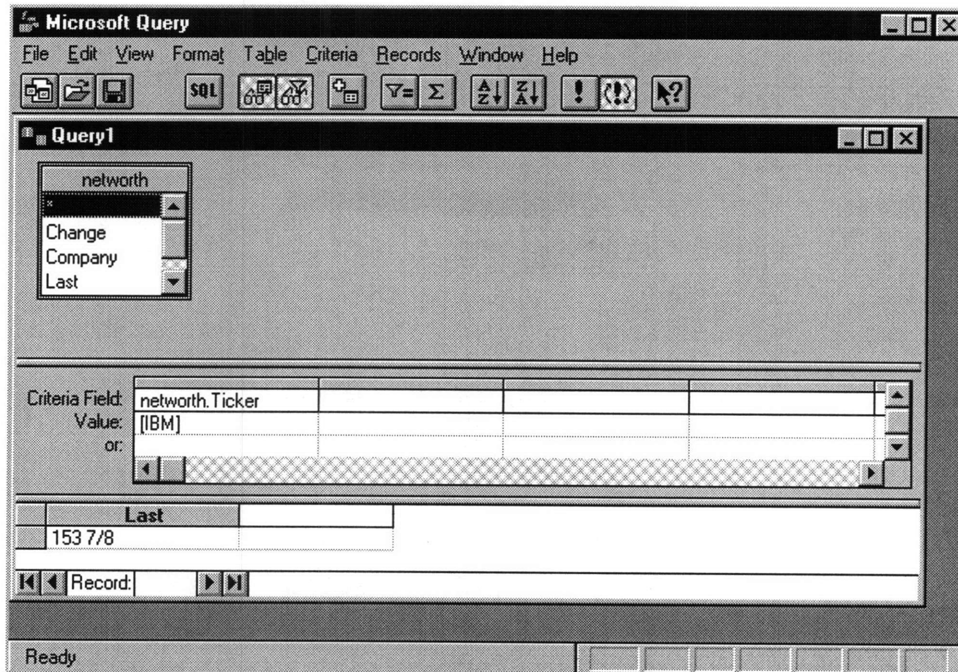

Figure 4.4-5: Result of the SQL query from the Context Mediator is returned

Second Prototype Example

MS Excel is the ODBC compliant application used to demonstrate the new capability of

the second prototype of both the Context Mediator and the Context Mediator-ODBC driver. In particular, the new capability of the ODBC driver is that it allows the user to input the desired "context" as part of the input SQL query. Moreover, it allows the user to specify the location of the Context Mediator so that the change in location of the Context Mediator will not result in any internal ODBC driver modification. Figure 4.4-6 summarizes the ODBC function calls used by MS Excel. (Refer to Appendix B.2 for details) Notice that the ODBC function calls used in the Excel example differ from that of the MS Query. But in general, the basic ODBC function calls of allocating storage, setting up connection, sending SQL statement, retrieving returned result, and deallocating storage are the same.

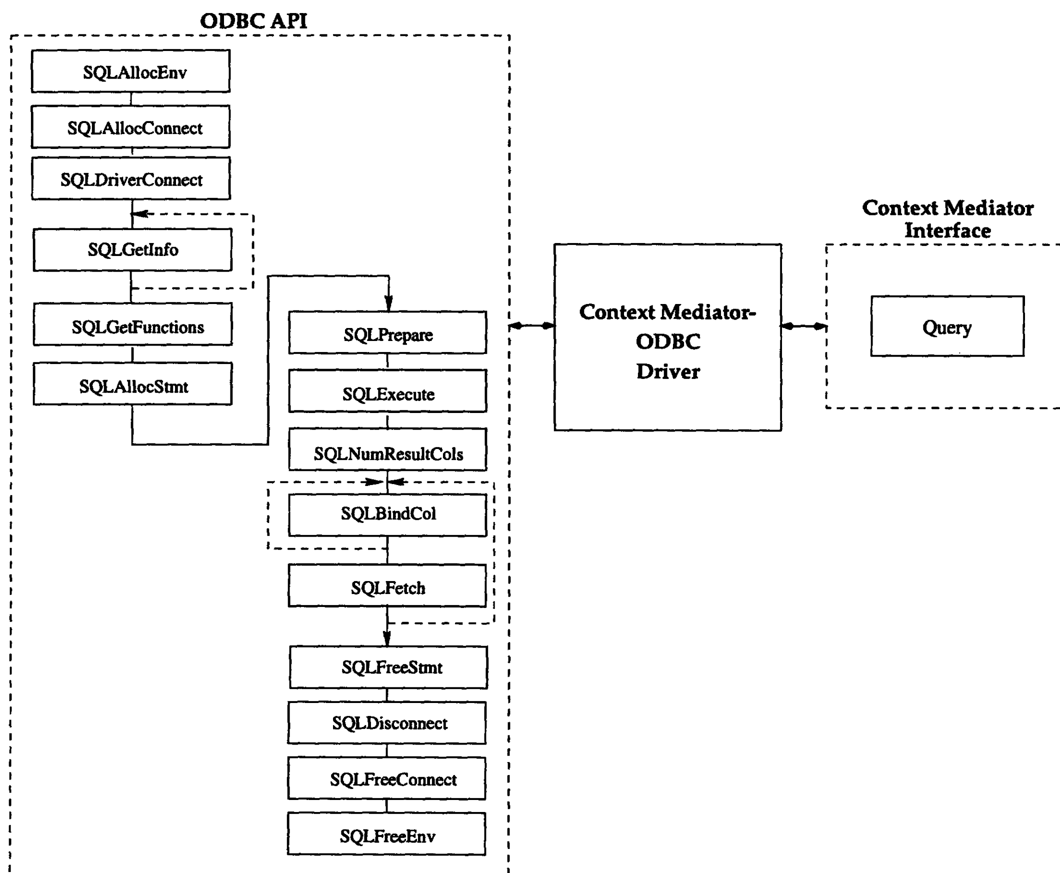

Figure 4.4-6: ODBC function calls and Context Mediator interface for MS Excel

This Excel example does not only illustrate the capability of specifying "context"[10], but it also demonstrates the Context Interchange System capability of querying ordinary DBMS such as Oracle database and non-ordinary DBMS such as the wrapped web pages. The example obtains information about a portfolio through a number of queries to the Oracle database and the web based on the Ticker symbol.

There are a total of ten queries in this spreadsheet example. They can be divided into four major parts. Three queries in column A use the Ticker symbol in the "Ticker" column to find the Company names through the Disclosure database.[11] Three other queries under the "Net Income and Total Assets" column in turn use the Company names obtained from the previously three queries to find the net income and total assets from also the Disclosure database.[12] Notice that the "context" of the spreadsheet is specified as "Worldscope", which is not the original context of the data source Disclosure. Thus, all the data returned from the Context Mediator has undergone mediation. Other than getting information from an ordinary database, this Excel example also obtains data from web sites. Three queries use the Tickers to find out the corresponding current stock price (the prices are fifteen minutes delayed) of each Ticker through the Security APL web site.[13] Finally, all the monetary amount are converted from US Dollars into the desired currency by the latest exchange rate queried from the Olsen web site[14] according to the specified currency code (Figure 4.4.7). Notice that each of the ten queries goes through the same set of ODBC function calls as shown in Figure 4.4.6.

---

10. Refer to Appendix C.1 for syntax of how to include the desired context in the input SQL query.
11. The query is: *select Ticker_Lookup.COMP_NAME from Ticker_Lookup where Ticker_Lookup.TICKER = <ticker name>* (refer Appendix C.1 for the syntax detail)
12. The query is: *select DiscAF.NET_INCOME, DiscAF.TOTAL_ASSETS from DiscAF where DiscAF.COMPANY_NAME = <company name>*
13. The query: *select secapl.Last, secapl.Date from secapl where secapl.Ticker = <ticker name>*
14. The query is: *select olsen.Rate from olsen where olsen.Expressed = <currency symbol> and olsen.Exchanged = <currency symbol>*

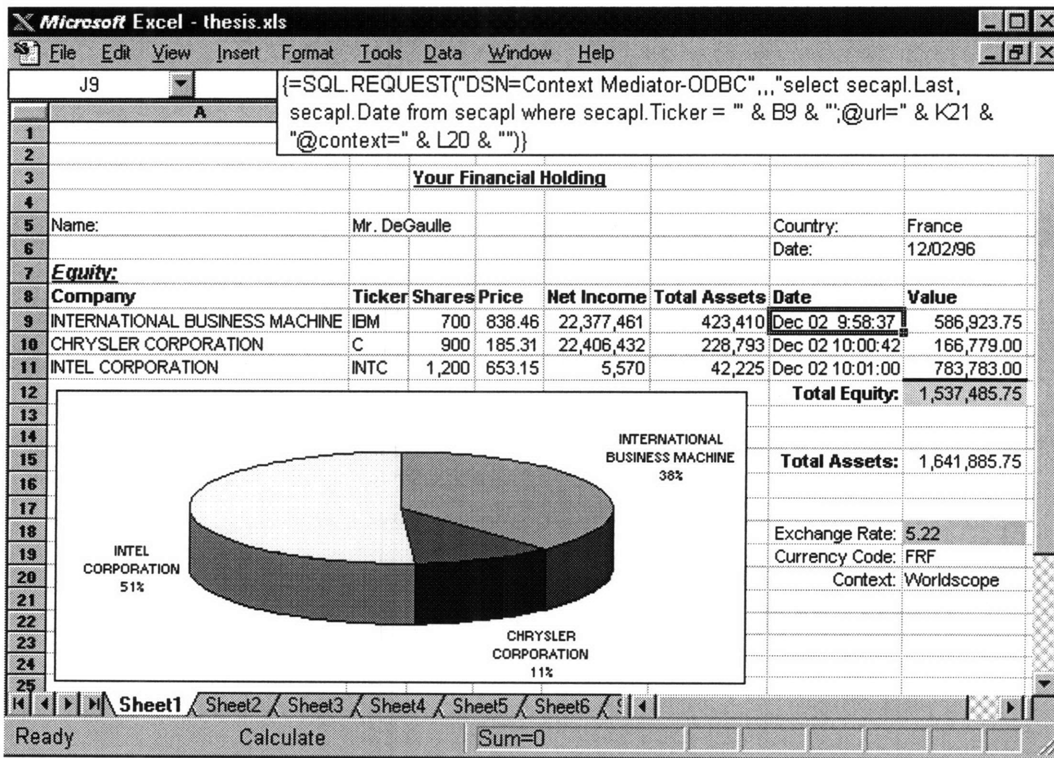

Figure 4.4-7: The MS Excel demo

4.4.2 Multi-datasource query scenario (Second Prototype)

The two examples above have illustrated the Context Mediator capabilities of querying both databases and web sites. However, each of the above query fetches data from only a single data source. The following example demonstrates that the Context Interchange System is capable of handling multi-datasource queries.

There is only one query in this example. The query requests information of the current stock price and total number of shares in the market of the companies where the companies have net income greater than 2.5 million. The information of the stock price is queried from the web site (secapl) which uses only ticker symbol as the searching index. While company names of the companies that have net income greater than 2.5 million are queried from database (WorldAF) which does not have ticker symbol as one of its attributes. Therefore, the query uses an auxiliary data source (Ticker_lookup) to get the

tickers for the companies of interest. Notice that the data sources are in different context, so mediation is performed as well. The result and the query itself are shown in Figure 4.4.8 below.[15]

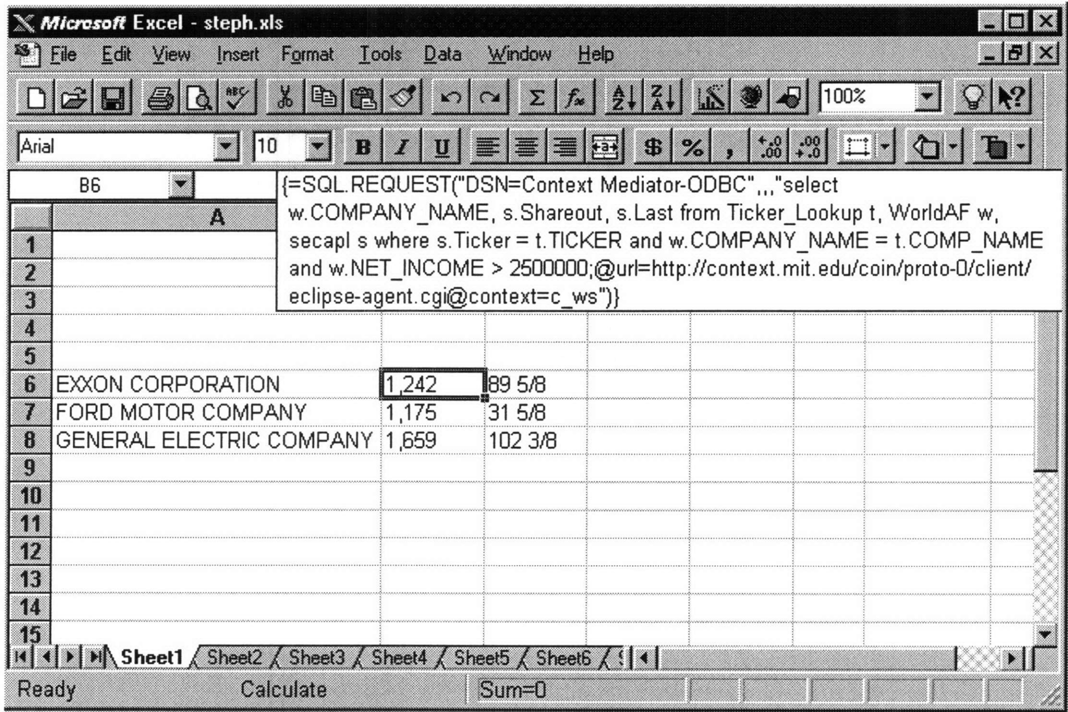

Figure 4.4-8: Multi-datasource query example by MS Excel using the second Context Mediator prototype

15. Notice that the query uses "variable assignment" to replace the datasource name in the "from" clause: w stands for WorldAF, t stands for Ticker_Lookup and s stands for secapl.

Chapter 5

Conclusions

In this chapter, future development relating to the extensions of the Context Interchange System will be discussed. In particular, further ODBC development will be suggested in order to enhance the capability of the Context Mediator-ODBC driver. In addition, proposal of a better and more complete interface for the Context Mediator will be raised. This chapter will also discuss issues about the full integration between the Context Interchange System and the ODBC framework. Moreover, it will investigate other possible future extensions to the Context Interchange System. Finally, the last section of the chapter will summarize the entire ODBC development project.

5.1 Further ODBC Driver Development

As described in the previous chapter, the Context Mediator-ODBC driver is not a fully implemented ODBC driver. That means that not all ODBC function calls are implemented. The limitation is due to the time constraint of the thesis project. The short term goal of the driver is to support MS Excel as the target ODBC compliant application. As of now, the ODBC driver can indeed support MS Excel as the client application. In addition, MS Query is also able to use the Context Mediator-ODBC driver to access the Context Interchange System.

Since the driver is implemented to support MS Excel, the implemented ODBC function calls are the ones that are being used by Excel. Furthermore, some of the ODBC function calls behave differently according to the input values of the parameters from the application. Such function calls are implemented only to handle the corresponding input parameter values. Thus in order to support other ODBC compliant applications such as Power Builder and MS Access, function calls used by these applications have to be implemented. In addition, the new applications might call the functions differently by

inputting different parameter values, thus those input parameter values have to be taken care of too.

Consequently, in order to have a fully implemented ODBC driver, further development of the Context Mediator-ODBC driver is necessary. The future development includes the implementation of all the currently not supported ODBC function calls. (Refer to Appendix C.3) All function calls should be implemented to support the entire set of their corresponding input parameter values as well. This process will require further investigation of the capability of the driver and the Context Interchange System since most of the parameter values are related to the specific functionality of the driver and characteristic of the data source. The current ODBC driver is able to recognize data types of "char" and "number" given from the Context Mediator and convert them to the corresponding ODBC data types. Further development of the driver should include the support of all other ODBC data types. (Refer to Appendix D of [6] for details about ODBC data types) In addition, the ODBC driver may require certain modifications of the design due to newly added implementation. For instance, in implementing SQLRowCount, the internal data structure might have to be modified so as to store the number of rows in the result sets.

5.2 Re-Design of the Context Mediator Interface

In the development of the first prototype of the Context Mediator-ODBC driver, the Context Mediator interface that has been used is described in Table 2.1 of Section 2.1.1. The interaction between the Context Mediator-ODBC driver and the Context Interchange System is rather straightforward since the Context Mediator interface mimics the ODBC calls. Therefore, each Context Mediator interface call corresponds to the same ODBC function call. In other words, ODBC function calls, such as SQLExecDirect, correspond to the same calls in the Context Mediator interface; and so do SQLDescribeCol, SQLColumns, etc.

The Context Mediator seems to work fine until concurrency and transaction problem arises. Before the ODBC extension of the Context Interchange System came into place,

the Multidatabase Browser was the only front end application. The Multidatabase Browser program is used to guide the user to build and execute an SQL query. SQLExecDirect and SQLExtendedFetch are called consecutively. SQLExecDirect executes the SQL query and stores the result in temporary storage within the Context Mediator. SQLExtendedFetch retrieves the result from the temporary storage and returns it to the application. The Context Mediator-ODBC driver also retrieves the results in the same fashion. The concurrency and transaction problem arises when two users executes different queries concurrently. The second SQLExecDirect call might be executed between the first SQLExecDirect and SQLExtendedFetch calls. Thus the temporary storage for the first result set will be overwritten by the execution of the second SQLExecDirect call. The first SQLExtendedFetch call will get the result of the second query, but not the first. The problem is that the Context Mediator interface does not deal with multi-users. Ever since the development of the ODBC driver, the chances of having the above problem occurred becomes very high.

The above problem is solved in the development of the second prototype of the Context Mediator. The second prototype creates different processes called "agents" to handle each SQL query execution. Moreover, it merges the SQLExecDirect and SQLExtendedFetch calls into a single call so that there is no more transaction problem to be taken care of.

However, this prototype does not provide any interface for getting information about the result set. Function calls such as SQLDescribeCol, SQLColumns and SQLTables are not currently supported in the current prototype of the Context Mediator. The Context Mediator-ODBC driver is not currently capable of supporting these function calls on the ODBC side yet. The complete development of the ODBC driver will require the operation of these functions.

It is necessary for the current Context Mediator prototype to provide a more complete interface for obtaining information about the result set. A standard Context Mediator interface should be developed in the near future. The interface will not only enable further development of the ODBC extension, but it will also provide a guideline for other possible

extensions of the Context Interchange System, such as the development of a new browser (front end) program using Java. The suggested interface should include calls that give back information about the available data sources, tables within each data source, columns within each table, data type of each column, and any other general information about the data source.

5.3 Full ODBC and Context Interchange System Integration

In the previous two sections, further development of the ODBC driver and re-design of the Context Mediator interface have been discussed. Future completion of the above two parts will result in a seemingly completed integration between the Context Interchange System and the ODBC framework.

The above ODBC development of the Context Interchange System represents just one side of the integration. The scope of the current ODBC development is that the Context Interchange System is treated as one of the many ODBC supported Database Management Systems (DBMS). In other words, the current development is a front end extension of the Context Interchange System. The client is the ODBC application, and the server is the Context Interchange System.

There is another side of the Context Interchange System and ODBC integration. Instead of having ODBC as the client and the Context Interchange System as one of the DBMS, the reverse can be true. That is, ODBC can serve as one of the DBMS supported by the Context Interchange System.

The "complete" integration between the Context Interchange System and ODBC will be enormous in terms of the amount of underlying information in the entire system. By establishing the Context Interchange System as one of the supported ODBC data sources, the System does not only provide the most popular software products in the contemporary computer world as the new front end, but the System itself also offers the ODBC world with another gigantic data resource since the System supports both ordinary and non-

ordinary DBMS. By having the ODBC world as one of the data sources, the Context Interchange System is also tremendously enriched with ODBC supported data resources. When the integration is complete, the entire integrated system will become a colossal information repository with the ability of having semantic interoperability between the data sources.

5.4 Future Context Interchange System Extensions

There are many possible extensions for the Context Interchange System other than the developed ODBC extension described in this thesis. For example, one of the desired capability of the Context Interchange System is to display the data retrieved from the Context Mediator on a spreadsheet. One possible solution is to provide Microsoft's ActiveX capability for the Context Interchange System.

5.4.1 ActiveX

ActiveX is a set of technologies that enables software components to interact with one another in a network environment, regardless of the language in which they were created.[8] One capability of ActiveX is that it enables users to view non-HTML documents, such as Microsoft Excel or Word files, through Web browsers. It is very practical for users in the UNIX working environment. Through the ODBC extension, users in the Windows operating system can now use MS Excel to retrieve data from the Context Interchange System. However, users in the UNIX world are not able to use any Windows based spreadsheet to view the data. Thus, the extension of ActiveX will enable UNIX users to use ordinary web browser to view the data through the MS Excel spreadsheet. There are obviously many issues that require more in-depth explorations and investigations on the capabilities of ActiveX.

5.4.2 OLE DB

Another possible extension to the Context Interchange System is the support of OLE DB.

OLE DB is a set of interfaces developed by Microsoft whose goal is to enable applications to have uniform access to data stored in DBMS and non-DBMS information systems.[2] Popular data access API's such as ODBC imposes high "entry bar"[2] for data providers by requiring the data to be exposed in certain way, like SQL for ODBC. OLE DB lowers the "entry bar" by requiring them to implement only the functionality native to their data storage. In short, OLE DB provides a way for any type of data storage to expose its data in a standard, tabular form.[9] Consequently, the OLE DB interface comprises an industry standard for data access and manipulation that can ensure consistency and interoperability in a heterogeneous world of data and data types. Note that as stated in [9], OLE DB is not developed to replace the ODBC industry-standard data access interface. ODBC will continue to provide a unified way to access relational data sources as part of the OLE DB specification. OLE DB should be a good resource for the development because its infrastructure is similar to the Context Interchange System. For further information about OLE DB, refer to [2] and [9].

5.5 Conclusions

The ODBC development is successfully developed as the new front end of the Context Interchange System. The ODBC driver now enables MS Excel and MS Query applications to access the System. However, the developed ODBC driver is not a fully implemented one. Further ODBC function calls development are necessary in order for the driver to support all ODBC compliant applications. The ODBC extension has also brought about other related aspects that provide the Context Interchange research group the opportunity to further investigate.

Appendix A

Sample HTML formatted result file

HTTP/1.0 200 Document follows
Date: Fri, 29 Nov 1996 15:40:40 GMT
Server: NCSA/1.4.1
Content-type: text/html

<HTML><HEAD>
<!-- Created by: Tito Pena, 04-Nov-1996 -->
<TITLE>Multi-DB Result</TITLE>
</HEAD>
<H1>Multi-DB Result</H1>
<BODY><HR>
Original query asked in context c_ws:<P><STRONG> select secapl.Last, secapl.Date from secapl where secapl.Ticker = 'IBM'; </STRONG>
<P>

Number of rows: 1

<TABLE BORDER>
<TR><TH>Last</TH><TH>Date</TH></TR>
<TR><TD>127 1/4</TD><TD>Nov 29 10:18:23</TD></TR>
</TABLE>

</BODY></HTML>

Appendix B

ODBC function calls sequence

B.1 MS Query Example[16]

SQLAllocEnv
SQLAllocConnect
SQLGetInfo
SQLDriverConnect
SQLError
SQLGetFunctions
SQLGetInfo
    SQL_CURSOR_COMMIT_BEHAVIOR
SQLGetInfo
    SQL_CURSOR_ROLLBACK_BEHAVIOR
SQLGetInfo
    SQL_DATA_SOURCE_NAME
SQLAllocStmt
SQLGetInfo
    SQL_ACTIVE_STATEMENTS
SQLGetInfo
    SQL_DATA_SOURCE_READ_ONLY
SQLGetInfo
    SQL_DRIVER_NAME
SQLGetInfo
    SQL_SEARCH_PATTERN_ESCAPE
SQLGetInfo
    SQL_CORRELATION_NAME
SQLGetInfo
    SQL_NON_NULLABLE_COLUMNS
SQLGetInfo
    SQL_QUALIFIER_NAME_SEPARATOR
SQLGetInfo
    SQL_FILE_USAGE
SQLGetInfo
    SQL_SEARCH_PATTERN_ESCAPE
SQLGetInfo
    SQL_QUALIFIER_TERM
SQLGetInfo
    SQL_DATABASE_NAME
SQLFreeStmt
SQLFreeStmt
SQLGetInfo
    SQL_MAX_OWNER_NAME_LEN
SQLFreeStmt

---

16. This sample file is simplified. It does not contain all the parameter values. Please refer to the Excel sample in the following Section B.2 for similar actual parameter values.

SQLFreeStmt
SQLTables
SQLBindCol
SQLBindCol
SQLBindCol
SQLFetch
SQLFetch
SQLFetch
SQLFetch
SQLFetch
SQLFreeStmt
SQLFreeStmt
SQLGetInfo
    SQL_IDENTIFIER_QUOTE_CHAR
SQLGetTypeInfo
SQLBindCol
SQLBindCol
SQLBindCol
SQLBindCol
SQLBindCol
SQLFetch
SQLFreeStmt
SQLFreeStmt
SQLGetInfo
    SQL_ACTIVE_STATEMENTS
SQLAllocStmt
SQLGetInfo
    SQL_TABLE_TERM
SQLGetInfo
    SQL_OWNER_TERM
SQLGetInfo
    SQL_QUALIFIER_TERM
SQLColumns
SQLBindCol
SQLBindCol
SQLFetch
SQLFetch
SQLFetch
SQLFetch
SQLFetch
SQLFreeStmt
SQLFreeStmt
SQLSpecialColumns
SQLBindCol
SQLFetch
SQLFreeStmt
SQLFreeStmt
SQLFreeStmt
SQLFreeStmt
SQLAllocStmt
SQLFreeStmt
SQLColumns
SQLBindCol
SQLBindCol

SQLFetch
SQLFetch
SQLFetch
SQLFetch
SQLFetch
SQLFreeStmt
SQLFreeStmt
SQLSpecialColumns
SQLBindCol
SQLFetch
SQLFreeStmt
SQLFreeStmt
SQLFreeStmt
SQLFreeStmt
SQLSetStmtOption
SQLExecDirect
SQLSetStmtOption
SQLNumResultCols
SQLDescribeCol
SQLColAttributes
SQLDescribeCol
SQLColAttributes
SQLBindCol
SQLBindCol
SQLFetch
SQLFetch
SQLFreeStmt
SQLFreeStmt
SQLFreeStmt
SQLDisconnect
SQLFreeConnect
SQLFreeEnv

B.2 MS Excel Example (query using SQL.REQUEST functions)[17]

SQLAllocEnv (create the Environment strcuture)
    0x01000000
    SQL_SUCCESS
SQLAllocConnect (create the Connection structure)
    0x01000000
    0x01010000
    SQL_SUCCESS
SQLGetInfo (get information about the driver)
    0x01010000
    SQL_DRIVER_ODBC_VER
    [5]02.10
    6
    162
    SQL_SUCCESS

17. Refer to Appendix C.1 for details.

```
SQLDriverConnect    (connect to the driver)
    0x01010000
    0xC40B0000
    [25]DSN=Context Mediator-ODBC
    SQL_NTS
    [0]
    600
    0
    SQL_DRIVER_NOPROMPT
    SQL_SUCCESS
SQLError            (check if there is any error)
    NULL
    0x01010000
    NULL
    [5]00000
    0
    [0]
    512
    0
    SQL_NO_DATA_FOUND
SQLGetFunctions     (find the supported function calls by the driver)
    0x01010000
    0
    FALSE
    TRUE
    . . . [18]
    FALSE
    SQL_SUCCESS
SQLGetInfo          (get information about the driver)
    0x01010000
    SQL_CURSOR_COMMIT_BEHAVIOR
    SQL_CB_CLOSE
    2
    NULL
    SQL_SUCCESS
SQLGetInfo          (get information about the driver)
    0x01010000
    SQL_CURSOR_ROLLBACK_BEHAVIOR
    SQL_CB_CLOSE
    2
    NULL
    SQL_SUCCESS
SQLAllocStmt        (create the Statement structure)
    0x01010000
    0x01010001
    SQL_SUCCESS
SQLPrepare          ("prepare" the wrapped HTTP protocol SQL query)
    0x01010001
    [150]select secapl.Last, secapl.Date from secapl where secapl.Ticker = 'IBM';@url=http://con-
        text.mit.edu/coin/proto-0/client/eclipse-agent.cgi@context=c_ws
    SQL_NTS
```

18. There is a TRUE/FALSE value for each ODBC function call. They are omitted in here.

```
    SQL_SUCCESS
SQLExecute          (execute the query: send the HTTP protocol, retrieve and store the result set)
    0x01010001
    SQL_SUCCESS
SQLNumResultCols    (count the number of columns in the returned result set)
    0x01010001
    2
    SQL_SUCCESS
SQLColAttributes    (set the size of each column)
    0x01010001
    1
    SQL_COLUMN_TYPE
    UNUSED
    UNUSED
    UNUSED
    30
    SQL_SUCCESS
SQLColAttributes    (set the size of each column)
    0x01010001
    2
    SQL_COLUMN_TYPE
    UNUSED
    UNUSED
    UNUSED
    30
    SQL_SUCCESS
SQLBindCol          (bind the application storage space to the "aBind_col statement structure)
    0x01010001
    1
    SQL_C_CHAR
    0x7D349100
    255
    0x74349100
    SQL_SUCCESS
SQLBindCol          (bind the application storage space to the "aBind_col statement structure)
    0x01010001
    2
    SQL_C_CHAR
    0x95359100
    255
    0x8C359100
    SQL_SUCCESS
SQLFetch            (fetch the result and store each data in the correct location of "aBind_col")
    0x01010001
    SQL_SUCCESS
<BOUND>             (i.e. since the application has storage space "binded", the data is actually
    1               returned to the application)
    SQL_C_CHAR
    [7]159 1/2
    255
    7
<BOUND>
    2
    SQL_C_CHAR
```

```
    [15]Nov 27  1:50:29
    255
    15
SQLFetch             (fetch the result again; but here detect the end of the result set)
    0x01010001
    SQL_NO_DATA_FOUND
SQLFreeStmt          (delete the statement structure and all its associated storage)
    0x01010001
    SQL_DROP
    SQL_SUCCESS
SQLDisconnect        (destroy the "socket" stored in the connection structure)
    0x01010000
    SQL_SUCCESS
SQLFreeConnect       (delete the connection structure)
    0x01010000
    SQL_SUCCESS
SQLFreeEnv           (delete the environment structure)
    0x01000000
    SQL_SUCCESS
```

Appendix C

User's Manual

C.1 Guide to create a new Excel spreadsheet using the Context Mediator-ODBC Driver

First of all, given that the input SQL statement includes the "url" location of the Context Mediator and the desired "context", it is therefore necessary to use a built-in function of Excel that would accept non-standard SQL syntax and grammar. As shown in the examples in Section 4.4.2 and 4.4.3, the Excel built-in function used is called *"SQL.REQUEST"*.

SQL.REQUEST contains five parameters, but the only two needed are the first parameter "connection_string" and the fourth parameter "query_text"[19]. Connection_string specifies the data source to be connected. It should be exactly the same name as listed in the data source list of the ODBC Administrator. In this specific case, it should be called 'DSN=Context Mediator-ODBC', where "DSN=" stands for "Data Source Name". The query_text will contain the SQL query, including the url tag and the context tag as shown in Figure 4.4.7. This entire query_text string will be sent directly to the ODBC driver. Notice that the syntax for the SQL query is as follows:

<SQL query>;<url tag><context tag>

where <url tag> has the form "@url=<url string>" and <context tag> has the form "@context=<context name>".

To attach a string in another cell into the query, the syntax " & <cell number> & " is used. For instance, to replace the string "IBM" in

"select secapl.Last from secapl where secapl.Ticker = 'IBM'",

with the string in cell B8, user should use

"select secapl.Last from secapl where secapl.Ticker = '" & B8 & "'

In doing so, user will be able to modify just the content of a single cell to have the effect of modifying the query. This is especially useful when a number of queries use the same string, such as the "Ticker" symbol string in the example shown in Figure 4.4.7.

Since a query usually returns more than a single column and row of data, it is necessary for the Excel spreadsheet to have enough (or more than enough) cells to display the data. Therefore, user should first highlights the cells (technically, in the Excel context, it is called specifying the array of cells) where the data is to be returned, then inputs the SQL.REQUEST function with the appropriate connection_string and query_text into the formula bar. Note that the formula bar corresponds to every cells that have been highlighted. In other words, all the highlighted cells have the SQL.REQUEST function. Finally, user has to press <Ctrl><Shift><Enter> to execute the query. The following summarizes all the steps described above:

---

19. Second parameter: *Output_ref* - ignore when input from a worksheet; Third paramenter: *Driver_prompt* - indicate whether the driver dialog prompt is displayed; Fifth parameter: *Column_names_logical* - omitted means the column names are not displayed.

1. Highlight the cells where the data will be returned.
2. Go to the formula bar to input the SQL.REQUEST function by either pressing the function button (fx) or type in "=SQL.REQUEST(...)"
3. Input the first and fourth parameter values connection_string and query_text respectively. Note that even though the second and third parameters are omitted, commas (",") still have to be put after each empty parameter.
4. Press <Ctrl><Shift><Enter> to execute the query.[20]

C.2 Complete List of Implemented ODBC API Function Calls

The following table identifies all the implemented and non-implemented ODBC API function calls in the Context Mediator-ODBC Driver.

Table C.2-1: Complete list of implemented and non-implemented ODBC function calls

Driver Part	Implemented ODBC Functions	Non-implemented ODBC Functions
Connect	SQLAllocEnv SQLAllocConnect SQLDriverConnect SQLDisconnect SQLFreeConnect SQLFreeEnv	SQLConnect SQLBrowseConnect
Info	SQLGetInfo SQLGetTypeInfo SQLGetFunctions	
Prepare	SQLAllocStmt SQLFreeStmt SQLPrepare	SQLBindParameter SQLDescribeParam SQLParamOptions SQLNumParams SQLSetScrollOptions SQLSetCursorName SQLGetCursorName
Execute	SQLExecute SQLExecDirect	SQLNativeSql SQLParamData SQLPutData SQLCancel

20. A complete example: *=SQL.REQUEST("DSN=Context Mediator-ODBC",,,"select secapl.Last from secapl where secapl.Ticker = '" & B8 & "';@url=http://context.mit.edu/coin/client/proto-0/eclipse-agent.cgi@context=c_ws")* {Note that the url string and the context string can also be replaced by cells which contain the strings. (refer query shown in Figure 4.4-7}

Table C.2-1: Complete list of implemented and non-implemented ODBC function calls

Driver Part	Implemented ODBC Functions	Non-implemented ODBC Functions
Results	SQLNumResultCols SQLDescribeCol SQLColAttributes SQLBindCol SQLFetch SQLGetData SQLMoreResults SQLError	SQLRowCount SQLSetPos SQLExtendedFetch
Catalog	SQLTables SQLColumns SQLSpecialColumns	SQLStatistics SQLTablePriviledges SQLColumnPriviledges SQLPrimaryKeys SQLForeignKeys SQLProcedures SQLProcedureColumns
Options	SQLSetConnectOption	SQLSetStmtOption SQLGetConnectOption SQLGetStmtOption
Transact		SQLTransact

C.3 Specification of the ODBC Driver

There are twelve C files in the ODBC driver, namely:

1. Catalog.c (671[21])
2. Connect.c (363)
3. Dll.c (64)
4. Execute.c (598)
5. Handles.c (375)
6. Info.c (187)
7. Options.c (67)
8. Prepare.c (495)
9. Results.c (852)
10. Setup.c (646)
11. Transact.c (28)
12. Handles.h (103)

All ODBC function calls specification of each part of the driver is listed below.

21. Approximate number of lines of code in the current file. The overall total is about 4500.

C.3.1 Catalog

SQLTables
Requires: Valid statement handle; the set of input parameters has to fall in one of the 4 cases that the function can handle. In particular, (1) szTableType is not null, (2) all four parameters are null, (3) szTableQualifier is null, and (4) szTableQualifier equals "%" and both szTableOwner and szTableName are nulls.
Modifies: The statement structure "curr_result_set"
Effects: According to one of the above (1) to (4) case, the function modifies the result set marker so that SQLFetch knows what kind of result set it is supposed to parse. Please refer to the code and ODBC 2.0 book for details about the functionality of the sets of parameters. Note that HTTP call is issued to retrieve registry information for case (1) and (2).
{ Partially completed implementation }

SQLColumns
Requires: Valid statement handle
Modifies: The statement structure "curr_result_set"
Effects: Modifies the result set marker to Column result and sends the HTTP call to retrieve column names of the specified table.
{ Partially completed implementation }

SQLSpecialColumns
Requires: Valid statement handle
Modifies: The statement structure "curr_result_set"
Effects: Changes the result set marker to retrieve special column result set. Since there are no special column in the Context Interchange data source, the result set is just empty.
{ Partially completed implementation }

SQLStatistics
Not Implemented.

SQLTablePriviledges
Not Implemented.

SQLColumnPriviledges
Not Implemented.

SQLPrimaryKeys
Not Implemented.

SQLForeignKeys
Not Implemented.

SQLProcedures
Not Implemented.

SQLPrcedureColumns
Not Implemented.

C.3.2 Connect

SQLAllocEnv
Requires: None
Modifies: The environment structure
Effects: Creates the new environment structure using the internal Handles package function.

SQLAllocConnect
Requires: Valid environment handle
Modifies: The connection structure
Effects: Creates the new connection structure under the current environment

SQLDriverConnect
Requires: Valid connection handle
Modifies: None
Effects: Sets up the http window socket by retrieving the url information in the "odbc.ini" file.

SQLDisconnect
Requires: Valid connection handle.
Modifies: The winsock structure in the connection structure
Effects: Closes and cleans up the winsock structure in the connection structure

SQLFreeConnect
Requires: valid connection handle
Modifies: The connection structure
Effects: Calls the handle package function to clear (delete) the connection data structure

SQLFreeEnv
Requires: Valid environment handle
Modifies: The environment structure
Effects: Calls the handle package function to clear (delete) the environment data structure

SQLConnect
Not Implemented.

SQLBrowseConnect
Not Implemented.

C.3.3 Dll

This is the part that constructs the driver into a DLL. This part used the same code given from the ODBC Software Development Kit.

C.3.4 Execute

SQLExecute
Requires: Valid statement handle, wrapped SQL statement is present in the statement structure
Modifies: The statement structure "curr_result_set" and the file referred by "result_file"
Effects: Sends the wrapped SQL statement ("Prepared_str") stored previously in the statement structure to the Context Mediator using the "socket" stored previously in the connection structure. Retrieves the result set and store it in the file indicated by "result_file". Also set the result set marker "curr_result_set" to the appropriate number.

SQLExecDirect
NOTE: This function still uses the first prototype of the Context Mediator for the sake of consistency with the COIN system's old registry. That is, it still requires the use of two separate Context Mediator interface calls "SQLExecDirect" and "SQLExtendedFetch" to get the data. Note also that the input SQL query has to be in the right format.
Requires: Valid statement handle, valid input ordinary SQL syntax query
Modifies: The statement structure variable "curr_result_set", "ExtendedFetch_str", "DescribeCol_str" and "result_file"; the "socket" of the connection structure
Effects: First it sets up the http window socket and store it in the connection structure. ("socket" and "server") Then it parses the input SQL query and translate it into the SQLExecDirect HTTP protocol. At the same time, SQLExtendedFetch HTTP protocol and SQLDescribeCol HTTP protocol are also created. Next, the ExecDirect call is sent to the Context Mediator. If the OK flag is returned, ExtendedFetch is sent and the results of the query is returned. The result is written to the text file (result_file) for storage and later retrieval. Finally, set the result set marker "curr_result_set" to the appropriate number. {To change this old prototype to the new prototype, one can simple change the parser into the new one.)

SQLNativeSql

Not Implemented.

SQLParamData
Not Implemented.

SQLPutData
Not Implemented.

SQLCancel
Not Implemented.

C.3.5 handles

The handles package contains all the internal functions that deal with the interaction between the "handles" from the application and the "internal data structure" from the driver. For details, refer to the code. A complete list of functions in the handles package is shown below.
Environment: CEnv_New (create new structure), CEnv_Delete (destroy structure), CEnv_FromHandle (given handle, return the structure pointer), CEnv_GetHandle (given pointer to structure, return handle), CEnv_Constructor (initialize structure), CEnv_Destructor (clean-up memory, etc.)
Connection: CDbc_New, CDbc_Delete, CDbc_FromHandle, CDbc_GetHandle, CDbc_Constructor, CDbc_Destructor
Statement: CStmt_New, CStmt_Delete, CStmt_FromHandle, CStmt_GetHandle CStmt_Constructor, CStmt_Destructor

C.3.6 Info

SQLGetInfo
Requires: Valid connection handle
Modifies: Output variable rgbInfoValue or pcbInfoValue
Effects: According to the input parameter values, output the appropriate rgbInfoValue (string output) or pcbInfoValue (number). Please refer to the ODBC 2.0 book for each parameter value details and SQL.H file for the pre-defined values.
{ Partially completed implementation }

SQLGetTypeInfo
Requires: Valid statement handle
Modifies: "curr_result_set" in the statement structure
Effects: Set the "curr_result_set" to the appropriate value. As of now, empty result set is used.
{ Partially completed implementation }

SQLGetFunctions
Requires: Valid connection handle
Modifies: pfExists parameter
Effects: Set the supported function of the driver in pfExists[i] to true, and all others to false. (i stands for each function call; refer SQL.H file for the pre-defined value of each function)
{ Partially completed implementation }

C.3.7 Options

SQLSetConnectOption
Requires: Valid connection handle
Modifies: "error_msg" of the connection structure
Effects: Only "error_msg" is changed when input parameter fOption is 103. This is used to solely an attempt to handle the call by Access. (that means further investigation is necessary)
{ Partially completed implementation }

SQLSetStmtOption
Not Implemented.

SQLGetConnectOption
Not Implemented.

SQLGetStmtOption
Not Implemented.

C.3.8 Prepare

SQLAllocStmt
Requires: Valid connection handle
Modifies: New statement structure
Effects: Creates the new Statement structure under the current connection

SQLFreeStmt
Requires: Valid statement handle
Modifies: The entire statement structure, result_file, or aBind_col
Effects: Depending on the input parameter, calls the handle package functions to clear (delete) the statement data structure and remove the result file, or just delete the result_file, or re-initialize the aBind_col array (i.e. unbind the "binded" storage)

SQLPrepare
Requires: Valid statement handle, valid input SQL query
Modifies: Statement structure's "Prepared_str", "socket" of the connection structure
Effects: Parse the input SQL query and translate it into the HTTP protocol. The query is then stored in "Prepared_str". In the process, the "url" of the target Context Mediator location is also parsed if present. It is then used to establish the HTTP window socket and store it under the connection structure "socket" and "server".

SQLBindParameter
Not Implemented.

SQLDescribeParam
Not Implemented.

SQLParamOptions
Not Implemented.

SQLNumParams
Not Implemented.

SQLSetScrollOptions
Not Implemented.

SQLSetCursorName
Not Implemented.

SQLGetCursorName
Not Implemented.

C.3.9 Results

SQLNumResultCols
Requires: Valid statement handle
Modifies: Statement structure's "parsed_header" (pos_pair array), "table_header"
Effects: Parsed the result file to find the table header row and store it in "table_header" string. Then parse through the entire string to find out each column name (header_name of the result set) and store its beginning position and corresponding length in the "parsed_header" array of pos_pair structure. At the same time, count the total number of columns and return it in 'pccol'

SQLDescribeCol

Requires: Valid statement handle, valid input parameter
Modifies: None
Effects: Using the input specified column number, fetch the column name from the "table_header" string using the "parsed_header". Then concatenate it with the "DescribeCol_str" stored previously in the statement structure. Use the http window socket in the connection structure's "socket" and send it to the Context Mediator and retrieve the result. Parse the returned html text to find the data type. Convert the data type into SQL data type recognized by ODBC and return to the application.

SQLColAttributes
Requires: Valid statement handle
Modifies: None
Effects: Returns a fixed size of the column. The current setting is 50 characters long.
{ Partially completed implementation }

SQLBindCol
Requires: Valid statement handle
Modifies: Statement structure's "aBind_col" array structure, "col_num" array
Effects: Make the input parameters of the specified column, which is in fact the storage space of the application, points to the corresponding field in the "aBind_col" structure. Refer to the code for further details. Also set the corresponding element of the "col_num" array to true.

SQLFetch
Requires: Valid statement handle
Modifies: "result_row", "row_number", "parsed_data"(pos_pair array), "aBind_col"
Effects: For results of the query, first locate the "table_header" and then fetch the current result row using the "row_number" counter. Increment "row_number" for next fetch. Store the current result row in "result_row" first. Then scan through the "result_row" string and use the "parsed_data" to store each data's starting position and corresponding length. Then, by using "col_num" array to identify the "binded" column, store the data in the corresponding field in the "aBind_col" array structure. This basically means returning the data back since the application storage space are pointing to the "aBind_col" structure. For result set other than the query result, simply parse the result text file to retrieve the appropriate data. The data is then stored in the appropriate field in the "aBind_col" structure as described above. For details, please refer to the code and comments.

SQLGetData
Requires: Valid statement handle
Modifies: None
Effects: According to the input column number, use the "parsed_data" pos_pair to locate the starting position and length of the desired data in the "row_result" string. Fetches the data and return it with its length.

SQLMoreResults
Requires: Valid statement handle
Modifies: None
Effects: If the "row_result" string is equal to the "table end" tag, return the value that there is no more data.

SQLError
Requires: Valid environment, connection and/or statement handle
Modifies: "error_msg" string of connection or statement structure
Effects: Depending on the input parameters which indicate the action level (i.e. whether it is statement, connection or environment), return the appropriate error message from the corresponding "error_msg" string and its size and state. Then re-initialize the "error_msg" string for later use.
{ Partially completed implementation - in particular, the correct state }

SQLRowCount
Not Implemented.

SQLSetPos
Not Implemented.

SQLExtendedFetch
Not Implemented.

C.3.10 Setup
This file include code that interact with the ODBC installer. It comes directly from the ODBC SDK.

C.3.11 Transact

SQLTransact
Not Implemented.

C.3.12 Handles.h
This is the file where the structures of the environment, connection, statement, as well as all other internal structures such as "pos_pair" and "aBind_col" etc. are declared. Also, the function calls in the handles package are declared in this file. Please refer to this file for details.

C.4 Miscellaneous Instructions of the Driver

C.4.1 Changing the Driver Implementation
In order to make modification to the files of the driver using the Microsoft Development Studio, one must open the workspace for the driver. It is currently called "test3" workspace. (and this name can of course be changed) After one has made changes to the files of the driver, one should first compile the modified file to make sure there is no bug in it. Before doing so, you should switch the project configuration into "cpltest3 - Win32 Release". This is where all the compilable files of the driver are. Note that if you accidentally click the build button, just click the stop button to stop it.

C.4.2 Instructions for Re-building the ODBC Driver DLL
Again, in order to build the driver using the Microsoft Development Studio, one must open the workspace for the driver. It is called "test3" workspace for now. Then make sure you are under the "test3 - Win32 Release" project configuration since this project configuration has the Makefile needed for building the driver. If the above process is done, press <shift><F8> or the build button to build the DLL for the driver. Note that if you accidentally click the build button, just click "no" when it prompts you to add the file and cancel the compiling process.

After that, the user should be able to find the latest version of the driver DLL in the sub-directory called ndebug32. Up to this point, the building process of the driver is done.

C.4.3 Instructions for Installing the ODBC Driver
Assume that you have finished building the latest version of the driver and you are in the sub-directory ndebug32 where the new driver DLL is.

1. Copy the driver file to the Drvsetup.kit sub-directory "setup32" (or the software shipping package).
2. Record the size of the new driver, and its modified date.
3. Open the "odbc.inf" file and go to the section of the driver specification.
4. Update the size and date of the driver in the lines "Driver" and "Setup".
5. Save and close the "odbc.inf" file.
6. Execute the file "Drvstp32.exe" file and follow instructions.

C.4.4 Testing the Driver
One should use the "ODBC Test" program to test the driver before using any ODBC compliant application like MS Excel or MS Query to field test the driver. The "ODBC Test" program allows user to test each ODBC

function call individually. Note that it is not until SQLDriverConnect is called and "SQL_SUCCESS" is returned has the ODBC driver been really connected. From then on, the driver code can really be tested.

All the available ODBC function calls can be found from the menu of the "Test" program. One should test the function calls in sequential order as if it is the real client application. This is necessary because some of the ODBC function calls rely on the success of the previous calls. Note that in order to find the appropriate order of ODBC function calls used, one can use the program "ODBC Spy" to find out exactly what have been called in the real situation. It is especially useful for finding out the input parameter values of the function calls used by the applications.

C.4.5 Troubleshooting with the Installed Driver

There are many possible reasons for the driver to malfunction. Here are a summary of the possible errors and ways to solve the problems.

- When one first install the driver, you have to make sure that the "Services" file under the Windows directory has the port 80 specified as http. This is a common problem when the driver does not seem to work at all. Note that even though you might have installed the driver a while ago and have made the above change in the 'Services" file before, you might want to re-check the file again since any re-installing of Windows 95 will result in getting the default "Services" file which does not have http port 80 specified.
- When you do not seem to get data back after the SQL statement is executed, go to Netscape and make sure the Internet is running and the Context Mediator engine is up.
- Another possible source for checking error is to go to the "temp" directory in "C" drive and check out the "trace.out" file generated by the driver. All the ODBC calls are recorded in the file and possible error message handled by the driver itself is written to the file for tracing purposes.
- The last thing is of course to try to trace the failure part of the driver using the "ODBC Test" program or modifying the driver code itself to debug.

Bibliography

[1] T. Berners-Lee, R. Fielding and H. Frystyk. *"Hypertext Transfer Protocol -- HTTP/ 1.0"*, HTTP Working Group, Internet-Draft, February 19, 1996.
URL: http://www.w3.org/pub/WWW/Protocols/HTTP/1.0/spec.html

[2] Jose A, Blakeley. *"Data Access for the Masses through OLE DB"*, SIGMOD 1996, Montreal, Canada, June 1996.

[3] Adil Daruwala, Chang Goh, Scott Hofmeister, Karim Hussein, Stuart Madnick, Michael Siegel. *"The Context Interchange Network Prototype"*, Sloan Working Paper #3797, The Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, February 1995.

[4] Cheng Hai Goh, Stuart Madnick and Michael Siegel. *"Context Interchange: Vacuuming the Challenges of Large-Scale Interoperable Database Systems in a Dynamic Environment"*, In Proceedings of the Third Int'l Conf on Information and Knowledge Management, pages 337--346. Gaithersburg, MD, November 1994.

[5] Cheng Hai Goh, Stuart E. Madnick and Michael D. Siegel. *"Ontologies, Contexts, and Mediation: Representing and Reasoning about Semantics Conflicts in Heterogeneous and Autonomous Systems"*, Sloan Working Paper #3848, The Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, August 1995.

[6] Marta Jakobisiak. *"Programming the Web - Design and Implementation of an Multidatabase Browser"*, MIT, May 1996.

[7] Ken Koviak. "*Windows Socket Programming*", Winter, 1995.
URL: http://www.csis.gvsu.edu/~erickson/NetworkingLab/NetworkingArchive/WindowsSockets/KenCoviak/winsockp.html

[8] Microsoft Corporation. "*Microsoft ActiveX Resources*", 1996.
URL: http://www.microsoft.com/activex/

[9] Microsoft Corporation. "*Microsoft OLE DB*", 1996
URL: http://microsoft.com/oledb/toc.htm

[10] Microsoft Corporation. "*ODBC 2.0 Programmer's Reference and SDK Guide*", Microsoft Press, 1994.

[11] "*The Context Interchange (COIN) Project at MIT*", March 22, 1995.
URL: http://rombutan.mit.edu/context.html#project-overview

[12] Bill Whiting, Bryan Morgan and Jef Perkins. "*Teach Yourself ODBC in 21 Days*", SAMS Publishing, First Edition, 1996.

[13] "*Windows Sockets - An Open Interface for Network Programming under Microsoft Windows*", Version 1.1, 20 January 1993.
URL: http://www.seed.net.tw/~seedw002/winsock.htm#ProgrammingWithSockets

6485-64