Diverse: A User Agent for Accessing External Databases

Dennis Heimbigner
Department of Computer Science
University of Colorado
Boulder, Colorado 80309

July 26, 1985

ABSTRACT

Diverse is a system that is capable of sophisticated access to external database systems. The requirements for such a system are outlined and the Diverse architecture is proposed for meeting those requirements. Diverse allows a user to indicate the information that is of interest and then it automatically retrieves the requested data. It is capable of integrating data from several external sources. The Diverse system is built upon a relational database system for storing data, a knowledge base about the possible states of external databases, and a syntax-directed database system for extracting data from the output of an external database. The detailed structure of Diverse may be characterized as a hierarchical collection of planning systems. Each level of the hierarchy has a knowledge base that encodes the structure of that level in the form of a graph. Associated with each level is a planning procedure that finds appropriate paths through the graph. The path is then used to invoke the appropriate procedures for manipulating information at that level. There are 3 levels of the planning hierarchy: (1) Universal Relations, (2) Media Access, and (3) Database Access. Diverse allows a user for formulate a relational query describing the information of interest. Diverse takes this query and determines which external databases can provide the requested information. For each such external database, Diverse automatically connects to it, provides appropriate input commands, collects appropriate output data, and finally disconnects. After doing this for each external database, Diverse calculates and presents the answer to the user.

Categories and Subject Descriptors: H.2.5 [Heterogeneous Databases], H.3.5 [On-Line Information Services] H.4.1 [Office Automation] H.4.3 [Communications Applications] I.2.1 [Applications and Expert Systems]

General Terms: Algorithms, Languages, Management.

Additional Key Words and Phrases: Office Automation, User Agent, Database, External Database, Heterogeneous Database, Universal Relation, Syntax-directed Parsing, Planning.

1. INTRODUCTION

The trend in office automation seems to be towards large numbers of essentially autonomous workstations networked with larger mainframes. In such a situation, accessing and integrating information from diverse sources is a fundamental problem. These sources consist of the various databases available throughout the network, but also the externally available commercial systems such as Dialog, Compuserve, The Source, and Dow-Jones. The term "micro-mainframe access" is sometimes used in reference to this problem because the external databases tend to reside on the large mainframes but the users of the data have their analysis tools on the workstations.

Each of the available databases may have a different command structure and a different model for structuring data (the database model). Thus, a user must know or learn many different languages in order to access various systems. Further, once the external data is captured in a workstation, it must be converted into a form that is usable by the tools available in that workstation. This conversion is complicated by the fact that most data is only available in text form. That is, external databases tend to assume that they are accessed by people rather than computers and so all of their output is in the form of human readable formatted text.

At the moment, typical workstations access external data in one of two modes: terminal emulation or tight integration. The first mode is to emulate a terminal calling into the system. The terminal emulator can capture the raw text output and place it in file on the workstation. While it does provide access, it provides no help in understanding the structure of the data. It may be difficult to extract the relevant parts from the mass of collected text. Additional programs must be constructed (usually by hand) to convert the data to a form usable by a local data manager.

Tight integration of the external databases is the second mode of access. In this mode, specialized programs execute on the workstation and on the external database

(host) computer. The local and host programs can communicate with each other to transfer data. Each "side" of the transfer knows about the other and about the structure of the data being passed. While useful, such a system has a number of deficiencies:

(1) The external database must be capable of running, at user request, a specific host program. Often this is impossible because the external database interface makes no provision for programmatic interfaces.

(2) The set of integrated databases is usually only a small subset of the total set of available databases. No help is provided for these other, non-integrated, databases.

(3) The workstation side of the integrated system may be only accessible in the context of some vendor supplied program. This often means that it is impossible to use other analysis tools.

In light of the limitations of these two approaches, it is reasonable to search for some better alternative: a system that is capable of more sophisticated access than terminal simulation, but that is also less restrictive than tight integration. This alternative should possess sufficient flexibility to operate in a heterogeneous environment. It should be capable of accessing a wide variety of databases, and it should be extendible to new databases as they become available. The goal of this paper, then, is to outline the requirements for such a system, and describe an implemented architecture that is capable of meeting those requirements. For convenience, I will refer to that system by the name Diverse (referring to its ability to access data from diverse sources).

2. REQUIREMENTS

The principle characteristics of an external data system are autonomy and heterogeneity. Such a system often is a commercial venture over which a user has no con-

trol. It is free to maintain its data in any way it wishes, and it is not required to provide special interfaces to any of its users. As a consequence, Diverse must provide its own, local, capabilities for integration, independent of the external data provider.

Since the external systems are autonomous and heterogeneous, it is very unlikely that they will all adopt identical, or even similar, interfaces and data structures. Of course, there is always some pressure to standardize, but historical inertia can often prevail. For example, all of the systems mentioned above (Compuserve, etc.) have radically different command languages. In fact, some of them have several different languages in common use depending upon which database is accessed. Thus, Diverse must be able to deal with the variety of command languages provided by these external databases. Diverse takes the approach of providing a common structure into which the external system is mapped. This map must encompass not only the structure of data, but also the procedures for accessing the data.

Additionally, Diverse must provide guidance and abstraction for the various activities involved in accessing data. It must provide step-by-step sequencing of these activities, it must provide knowledge about the external system, and it must allow the user to augment his knowledge about existing systems or newly available ones.

Finally, Diverse must be capable of combining data from several external databases. Suppose the user requests that the news headlines for the top ten best performing stocks be placed into the local database No one external database can can answer this question directly. But, a combination of data from the Compuserve Microquote system and data from the Dow-Jones system can provide the answer. Diverse must be prepared to take the user's request, figure out that it must access the list of names from Microquote, and then it must input those names to Dow-Jones to collect the relevant headlines.

3. AN ARCHITECTURE FOR DIVERSE

Diverse operates under three major assumptions:

(1) It is assumed that the only interface to the external database is through text (i. e., the same interface available to a person using that database).

(2) It is assumed that all external data can be mapped into some standard database model; Diverse uses the relational model. The previous assumption (text interface) makes this a reasonable choice since it forces all of the data to come out in terms of strings and numbers.

(3) It is assumed that only read access is required for external databases.

Diverse allows a user for formulate a relational query describing the information of interest. Diverse takes this query and determines which external databases can provide the requested information. For each such external database, Diverse automatically connects to it, provides appropriate input commands, collects appropriate output data, and finally disconnects. After doing this for each external database, Diverse calculates and presents the answer to the user.

The detailed structure of Diverse may be characterized as a hierarchical collection of planning systems. Each level of the hierarchy has a knowledge base that encodes the structure of that level in the form of a graph. Associated with each level is a planning procedure that finds appropriate paths through the graph. The path is then used to invoke the appropriate procedures for manipulating information at that level.

There are 3 levels of the planning hierarchy:

Universal Relation:

The first level consists of sets of attributes that are in turn combined into relations. The user accesses these relations by using a universal relation interface. Such an interface allows the user to specify a query in terms of the result attri-

butes and the input attributes without having to deal with the joining of relations that might be necessary to answer the query.

Media Access:

The second level of the planning hierarchy encapsulates information about the communication media that may be involved in accessing a particular database. Media knowledge is broken into two pieces. First, there is knowledge about the underlying carrier (e. g., telephones or ethernet). Second, there is knowledge about the network commands provided by such systems as Telenet or Tymshare. Planning at this level involves (1) executing the appropriate commands to set up the carrier and to physically connect to the Network, and (2) sending network commands to connect to the desired external database.

Database Access:

The third level of the hierarchy finds and extracts data from the external database. It plans the (database specific) commands necessary to maneuver in the external database.

In addition to the planning system, Diverse has two other components. These are in some sense orthogonal to the hierarchy in that they are used by several of the levels:

(1) The syntax-directed database system is used to interpret command results and to extract the desired data from these results.

(2) A relatively conventional relational database is used to store intermediate and final results.

Figure 1 outlines the interconnections among the various components of the system.

The rest of this paper describes the detailed operation of each component. Examples will deal with financial information from the Dow-Jones News Retrieval System (DJNS) and (to a minor extent) the Compuserve Microquote system. The medium used is the Telenet network. Dow-Jones provides, among other things, access to news

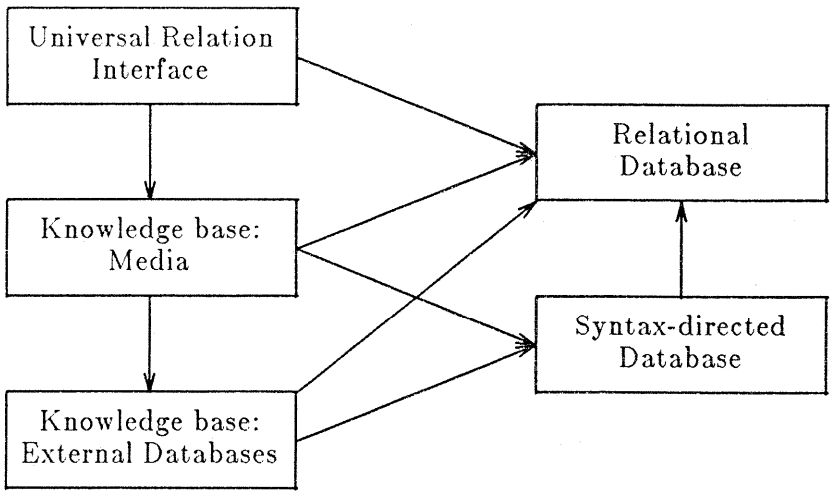

Figure 1. Diverse Component Level Architecture

headlines and stories about specified companies. These headlines are indexed through the stock exchange name for that company. Figure 2 shows sample output from a request for the headlines for the "cbu"[1] company. The line (in bold) after "ENTER QUERY" requests page one of headlines about the company "cbu".

4. RELATIONAL DATABASE

The relational database component of Diverse is a standard relational database. It manages relations with attributes over the domain of strings. The relational database component is the foundation tying the rest of the system together. The relations connect a user query with specific external databases. The syntax directed database connects external database output with specific attributes in the relational database. The output from an external database is stored into the relational database.

[1]The name of a company is represented by its stock exchange name. So "cbu" is the name for Commodore Business Machines.

DOW JONES NEWS IS BEING ACCESSED

ENTER QUERY
.cbu 01
N CBU 01/02
AV 01/09 COMMODORE INTL ANALYSIS -2-
(DJ)
AU 01/09 SOURCES SAY RUMOR OF POSSIBLE
(DJ) COMMODORE PRICE CUT HURTS STK
AT 01/07 ATARI, COMMODORE INTRODUCE
(WJ) HIGHER-PRICED COMPUTERS
AS 12/26 HEARD ON STREET: ANALYSTS
(WJ) PICK COMPUTER SHAKEOUT WINNERS

Figure 2. Sample Dow-Jones output.

4.1. Attribute Values

The relational database provides two levels for data storage. First, it provides storage for conventional tuples stored in relation. Second, it provides for temporary storage of values for attributes. The latter is used to hold both input and output values resulting from access to an external database. These values are stored in a *value list* is associated with each attribute in the database. The value list stores an ordered list of values (e. g., strings).

4.2. User Relations and External Databases

It is important to understand the connection between the external databases and the set of relations provided in the user interface. Each relation is associated with an external database system. That relation is derived in the sense that its contents reflect the data in the external database. For example the relation

Headlines(company,headline)

is associated with the Dow-Jones system and its content is taken from output such as figure 2. Figure 3 shows the instantiated relation that is derived from figure 2.

Each relation is directional, which means that some of its attributes are tagged as *input* attributes, and the rest as *output* attributes. In terms of the underlying external database, this means that at some point, one can send the input attribute values, and expect to receive a screen of information that contains the corresponding output attribute values (there may be more than one output for a given input). In effect, the input attributes together form a key for the relation. For the Headlines relation, the input attribute is company, and the output attribute is headline.

4.3. Universal Relation Interface

The user interface for Diverse is a simple variant of the Universal Relation (UR) interface for database systems [Korth 84, Maier 83]. For our purposes, the interface consists of small relations (corresponding to objects in the UR model) and a collection of attributes. The user frames a request in terms of the available attributes, but without worrying about which relations contain those attributes. By considering attributes independent of the relations, the user is freed from the specification of most relational join operations.

The principal problem for UR systems is converting a query against attributes into a series of selects, projects, and joins on the underlying relations. To simplify the problem, Diverse currently restricts the form of queries by disallowing tuple variables and restricting comparisons.

company	headline
cbu	AV 01/09 COMMODORE INTL ANALYSIS ...
cbu	AU 01/09 SOURCES SAY RUMOR OF POSSIBLE...
cbu	AT 01/07 ATARI, COMMODORE INTRODUCE...
cbu	AS 12/26 HEARD ON STREET: ANALYSTS...

Figure 3. Sample Headlines Relation

In Diverse queries are assumed to be of the form:

Retrieve $A_1, A_2, ...A_n$
Where Predicate

The A_i are attribute names and the predicate is a series of comparisons tied together with "and" and "or" operators. A comparison is between an attribute and a constant.

In practice, Diverse requires that all comparisons be equality comparisons with constants. This restriction is motivated by the capabilities of the external databases. Most of them cannot answer a request, for example, of the form "give me all the stocks with prices greater than $50"; it is a query involving inequalities. Given a more intelligent external database, it is fairly easy to extend the class of Diverse queries to match.

Given a query, it is necessary to deduce the set of underlying relations that will be needed to produce an answer. We explicitly assume the weak equivalence solution [Ullman 82] in order to limit the number of relations involved in answering the query. Thus, only relations that have attributes involved in the query will be used in constructing the answer.

Diverse deduces the set of involved relations by performing a restricted depth-first search through the hypergraph [Maier 83] of relations. The result is a set of non-cyclic paths, which means that (1) the same relation is never used more than once, and that (2) no attribute is used to derive itself. At the end of the algorithm, each returned path will contain, in order, a sequence of relations that can be accessed to answer the query. This path starts with the given attributes, and ends with the target attributes. These relations will be connected by attributes common to two or more of the objects in the path.

Given the set of paths, Diverse chooses the first one in the set of paths. A more comprehensive analyzer should use some criteria for picking one. Possible criteria might be to minimize the number of external databases to be accessed, or to minimize

path length.

4.4. An Example

As an example, consider the following two relations:

(1) Headlines(company,headlines). This is a relation that can be constructed from data available in the Dow-Jones system.

(2) Reports(report_id,company). This relation is available from the Compuserve Micro-quote financial reporting system. Micro-quote makes available a number of "reports" listing, for example, the top 10 highest percentage gain stocks in the last day or week. This relation is intended to capture that information by associating a report_id with the companies listed in that report[2].

Note that the universal relation for this database is

U(company,headlines,report_id).

Suppose that the following query is posed against that universal relation:

```
Retrieve headlines
where report_id = "stocks with highest gain"
```

Actual operation of the algorithm is a bit more complicated than will be shown below since it checks for cycles and can try to derive multiple attributes at once. In this example, there is in fact only one path through this set of relations. Informally, it is calculated as follows:

(1) Find a relation whose output attribute is "headlines". The only possibility is the relation "Headlines". Mark this relation as the last element in the accumulated path.

(2) Since the attribute "headlines" has been derived, discard it and attempt to derive the input attributes of the just chosen relation, namely, the attribute "company".

[2]Note that this is an example of the limitations of current external databases. In effect, Micro-quote

(3)

Add the relation "Reports" to the the path, discard "company", and try to derive "report_id".

(4) Note that "report_id" is given in the predicate, so quit and return the ordered path [Reports, Headlines].

Thus, to answer our query, we must access Reports, with report_id as input, and a set of companies as output. Then we access Headlines with those companies as input, and the set of headlines as output.

4.5. Query Answering

Given the list of relations, the result is formed, in principle, by joining the relations, applying the selection criteria, and projecting out the target attributes. In fact, Diverse performs the joins in stages. The reason for this is that the output values from one relation must be used as inputs to the next relation (and hence the next external database).

Once a path is constructed, it is the duty of Diverse to use it as a plan for accessing some set of external databases. As described above, every relation in the user interface is associated with an external database. So it is possible to translate the path of object relations into a series of references to the associated external databases. Diverse takes each relation in turn and accesses the external database to fill in the corresponding relation.

Given a relation, Diverse establishes initial value lists for its input attributes. For the first relation accessed, the initial values come from the query predicate. Successive relations take their values from preceding relations. Note that this implies sequential processing. It is possible that in some cases Diverse could perform accesses

---

has a number of special built-in queries. It would be better if it let the user formulate more general queries.

in parallel, but it is not clear how much time would be saved. In any case, the current version of Diverse performs strictly sequential processing.

When a relation is to be filled in, the value lists for its input attributes are initialized. The external database is contacted (as described below) and the output attribute lists for the relation are filled. A value list for an output attribute may be used in later relations as an input attribute, but it will never be changed once it is filled.

After the final attribute of a relation is filled in, a procedure called a "generator" (section 6.3) is invoked to convert the value lists to actual tuples for the relation. The tuples are then stored in the local relational database. When the final relation has been calculated, the appropriate selects, projections, and joins can be performed to produce the desired answer.

5. REPRESENTING KNOWLEDGE ABOUT EXTERNAL DATABASES

The heart of diverse is a collection of knowledge bases concerning the operation of various communications media and about the operation of various external databases. This knowledge is used to perform the automatic access to the external databases. The access is broken into two related steps. First, knowledge about a communications medium is used to connect to (and later disconnect from) the external database. Once the connection is made, the knowledge about the external database is invoked to extract the relevant data.

In both cases (medium and database), the knowledge base uses a model based on the notion of state machine with inputs and output. The knowledge base records the known states of an external system. It also records the appropriate inputs that can drive the system from state to state, and the outputs that are produced when moving from one state to another.

The reason the model works is that, by design, external systems such as Dow-Jones or Telenet operate in a classical state machine fashion:

(1) The user is presented with a screen of information representing an initial state.

(2) The user provides some input. This might be a baud-rate, a menu selection, or it might be specific data such as the name of a stock (when dealing with a financial database). This step corresponds to the specification of a transition.

(3) The system accepts the input and provides a new screen of information. this corresponds to moving to a new state.

Thus, a user drives the system from state to state by providing input data at each step.

As with any state machine, there is an initial state (entered automatically on connect or login) and a final state (entered on logout or disconnect). It is possible for a user to re-enter states any number of times. It is important also to note that entering a new state may cause output of useful information such as the current price for a stock. Typically, a user will provide input data (e. g., the stock name) at some state, and some number of states later (usually the next state), the system will provide some useful output (e. g., the stock price).

Each external database (and medium) is modelled in the knowledge base by a graph (see figures 4 and 5). The nodes of this graph correspond to the possible states of the external database. Each node is typed to indicate its purpose. Each graph is assumed to have a unique *root* node which is its start state. There may be any number of *final* states, although there is usually only one. All other states are typed as *intermediate*. For the media graphs (such as for telenet, figure 5), one node may be marked as a *database* state. This means that when this node is reached, it is time to recursively invoke some external database graph to continue operation. When that external database graph is finished, traversal of the medium's graph continues.

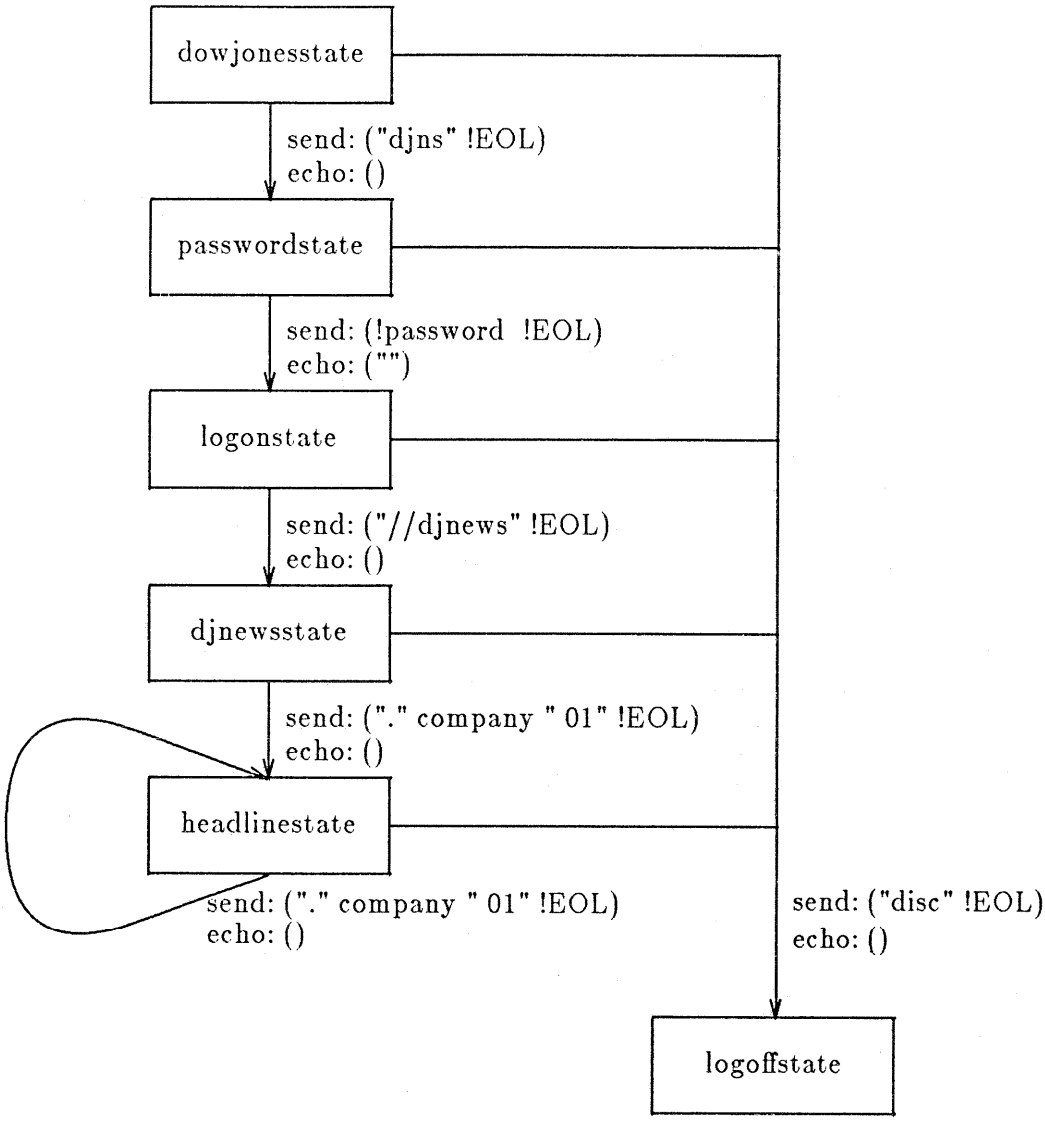

Figure 4. Dow-Jones Graph

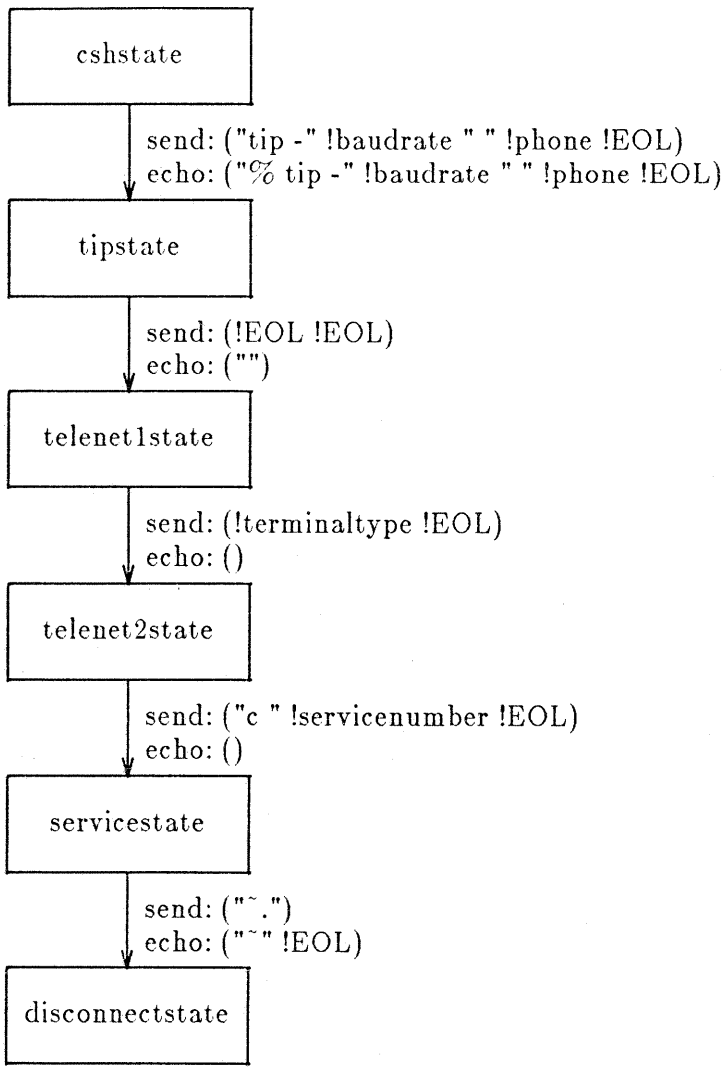

Figure 5. Telenet Graph

Associated with each node is a collection of edges leading to other nodes of the graph. The graph may contain cycles, and in practice it usually does (see figure 4, for example). Traversing an edge of this graph involves sending some string as input to the external database. Associated with each edge is a specification of that input and how it is to be constructed. the specification consists of a concatenation of three kinds of elements:

(1) constants strings,

(2) parameters, which are system dependent values such as baud rate, phone numbers, and passwords, and the machine dependent end of line,

(3) attributes, which are values taken from the attributes in the relational database. It is through these that database values are provided as input to the external database. Attribute references are normally only present in the graphs for external databases. The communications graphs do not use them.

Since the physical interface is usually through some sort of terminal emulation program (see section 7), any input string will be echoed to the output. The echo is handled by associating another specification with the edge describing the expected echo. If not echo specification is given, then the input specification is re-used. After the input is sent, characters are read from the output and matched with the echo specification until the whole echo is absorbed. It should be noted that the echo may not match the input if the local system has turned off or modified the echo. Password input is an example. The input string is the password, but it has a null string as its echo.

In figures 4 and 5, the edges are marked with the send and echo specifications surrounded by parenthesis. Constant strings are in double quotes, attributes are names, and parameters are names preceded by an exclamation point.

After an edge is traversed, the destination node of that edge becomes the current state of the graph traversal. At the same time, the external database is producing a screen full of text (separate from the echo) in response to the input string. Each state has a grammar associated with it that is used to parse the output produced by the external database. The detailed operation of this parse is described in section 6. For now, it is sufficient to note that the parse is used for two purposes. First, a successful parse verifies that the proper state was reached. Second, the parse may extract data from the text and place it in the database.

Given a graph, such as the graph for Dow-Jones or for Telenet, diverse must plan a set of partial paths through the associated graph. Given two nodes, diverse can easily calculate all paths between them, and so its job involves (1) picking the end-points of the path, and (2) choosing one of those paths.

For the communications graph, the traversal is straightforward:

(1) Find any path (there is usually only one) from the root of the graph to the node of type *database*.

(2) Recursively traverse the designated database graph and then return to finish the communications graph.

(3) Choose any path from the database node to the final node of the communications graph.

Traversing a graph for the database proper is a bit more complicated. This involves choosing three path segments.

(1) Find a path from the root to a *desired* node. This will consume one of the associated values from each input attribute.

(2) Find a path from the desired node back to that desired node. This path will be used only if the input attributes have more than one associated value.

(3) Find a path from the desired node to the final node.

Choosing the desired node is performed as follows:

(1) Take the set of all nodes producing at least one of the output attributes of the relation to be calculated from this external database. A node produces an attribute if its grammar can assign a value to that attribute.

(2) Form all paths from the root node to each of the producing nodes found in step 1.

(3) Examine each path from step 2 to see if it is *suitable*. Of all the suitable paths, arbitrarily choose one.

A path is considered suitable if it meets the following criteria:

(1) The path should consume exactly the set of input attributes for the relation of interest.

(2) The path should produce exactly the set of output attributes.

(3) The path should consume all input attributes before it produces any output attribute.

These criteria are also used to choose a path from the desired node back to the desired node.

5.1. An Example Continued

Suppose that the query of section 4.4 has been decomposed, relation Reports has been filled in, and now relation Headlines is to be obtained from the Dow-Jones external database. Dow-Jones uses the Telenet network as its communications media (review figures 4 and 5[3]).

The traversal algorithm is applied first to the Telenet graph to find a path from the root state (the one labeled "cshstate") to the database state (the one labeled "ser-

[3]Note that both graphs are only partial; There are many more states than shown in the figures.

vicestate"). This path is traversed by sending the appropriate commands. The first part of figure 6 shows actual output from Diverse describing the path to be followed, the input specification, and the actual inputs sent to traverse that path. After this path is traversed, the Dow-Jones graph is invoked and paths are traced for it. When that graph finishes, the remainder of the Telenet graph is traversed to reach its final state (last part of figure 6).

When the Dow-Jones graph is invoked, the algorithm must first find the *desired* state. Without going into details, the only state that produces an attribute is the one labeled "headlinestate", and it produces the attribute "headlines". The only edges that consume an attribute are (1) the edge from the state labeled "djnewsstate" to the state labeled "headlinestate, and (2) the cyclic edge from headlinestate back to headlinestate. Both consume the attribute "company". Given this information, our algorithm will pick headlinestate as the desired state.

Now our algorithm will find a path from the root ("dowjonesstate") to the desired state ("headlinestate"). This will consume one value from the company attribute value list and append a set of headlines to the value list for the headlines attribute (see next section for details of this extraction). If there are more values left in the company attribute, then the algorithm will follow the cyclic path from headlinestate back to headlinestate. It will repeat this until all company attribute values have been consumed. Then to end the traversal, the algorithm will follow the path from the headlinestate to the final state (labeled "logoffstate"). Figure 7 shows the actual traversal assuming that two company values ("eex" and "cbu") are available.

6. SYNTAX-DIRECTED DATABASE SYSTEM

A key problem for Diverse concerns the extraction of data from the external database. It must, somehow, obtain the data from the external database and then transform it into a format that can be manipulated by the local database system.

```
Following path:
cshstate => tipstate ("tip -" !baudrate " " !phone !EOL)
tipstate => telenet1state (!EOL !EOL)
telenet1state => telenet2state (!terminaltype !EOL)
telenet2state => servicestate ("c " !servicenumber !EOL)
sending: tip -1200 3376060
sending:  n   n
sending: d1
sending: c 60942

Following path:
servicestate => disconnectstate ("~.")
sending: ~.
```

Figure 6. Telenet Graph Paths

```
Following path:
dowjonesstate => passwordstate ("djns" !EOL)
passwordstate => logonstate (!password !EOL)
logonstate => djnewsstate ("//djnews" !EOL)
djnewsstate => headlinestate ("." company " 01" !EOL)
sending: djns
sending: XXXXXXXXXX
sending: //djnews
sending: .eex 01

Following path:
headlinestate => headlinestate ("." company " 01" !EOL)
sending: .cbu 01

Following path:
headlinestate => logoffstate ("disc" !EOL)
sending: disc
```

Figure 7. Dow-Jones Graph Paths

Diverse is constrained in its options because the external databases are, by definition, autonomous. This means that Diverse must use whatever standard interface those systems provide, namely text strings. The syntax-directed database component is responsible for extracting database information from the textual output of an external data-

base.

The SDDB and the knowledge base are connected by a set of grammars. Each state node in any graph in the knowledge base is associated with one of the grammars. As described previously, Diverse traverses a graph by constructing input strings that drive the external database from state to state. Whenever an input is sent, Diverse moves to the next state in the graph. It then begins parsing the external database output using the grammar associated with that state. This grammar serves two purposes. First, it may be used simply to verify that the external database is in the expected state. The grammar for such a test usually consists of a search for a series of strings characteristic of that external database state.

The second kind of grammar both verifies the state and extracts attribute values from the output. During the parse of the output, substrings of the parse tree are collected and stored in the value list for some specified attribute.

6.1. Parser Details

Diverse uses a simple, recursive, parser with backtrack. It expects an attributed grammar [Knuth 68] that consists of a series of productions. Each production has a left side, which is a non-terminal, and a set of alternative right sides. A right side consists of a sequence of patterns. The set of possible patterns is heavily influenced by SNOBOL [Griswold 71]. The patterns are taken from the following possibilities:

String constant

- succeeds if the current input matches a specified string constant. This pattern is denoted by a string enclosed in double quotes.

Nonterminal

- succeeds if the current input matches an instance of the pattern represented by some specified non-terminal. This pattern is denoted by the non-terminal name.

Class

- succeeds if the current input character is one of the characters in the specified list of characters. This pattern is denoted by "any(<string>)".

Length

- succeeds by matching the next n characters of the input. This pattern is denoted by "len(<integer>)".

End of line

- succeeds if the current input is an end of line. This pattern is denoted by "↓".

Arbitrary

- succeeds when it eventually locates an instance of some pattern sequence in the input string. This pattern is denoted by "arb(<pattern-list>)".

Not - succeeds when reaches a point in the input that does not match a specified pattern sequence. This pattern is denoted by "~(<pattern-list>)".

The last two patterns must be used with care since they will not stop until they succeed. If, for example, *arb* is used to search for some string, and that string has been corrupted by noise, then the parse will loop indefinitely looking for that string.

6.2. Extracting Attribute Values

An attribute may optionally be associated with each pattern occurring in a right side of a production. Every time that this pattern is successfully matched, the matched string is inserted into the value list associated with the attached attribute.

Figure 8 shows an example grammar for parsing headlines about specific companies. The grammar is broken into two parts. The part beginning with the "*:" defines productions that are common to more than one grammar. The part beginning with the "headlinestate:" defines the grammar to be invoked whenever the state labeled "headlinestate" is entered. It may be instructive to compare the grammar to

figure 2, which represents data that it is intended to parse. Note the occurrence of the attribute tag in the production for "headlineseq". Such tags have the form

(attribute = pattern).

When the parse is completed successfully, each substring of the input that is matched by the attributed pattern is located. These substrings are copied and placed in the value lists of the corresponding attribute. In figure 8, this means that all substrings matching the pattern "headline" in the context of a "headlineseq" will be assigned to the value list of the attribute "headlines".

```
*:
        EOL ::= "↓" ;
        spaces ::= arb( ~( any("        ")));
        spaces_or_eols ::= arb( ~( any("↓")));
        digit ::= any("0123456789") ;
        letter ::= any("ABCDEFGHIJKLMNOPQRSTUVWXYZ") ;
    .
headlinestate:
    headlineresponse ::= companyhead headlineseq EOL "      ";
    companyhead ::= spaces exchange spaces tickername
            spaces pagespec spaces EOL;
    headlineseq ::=
            (headlines = headline)
            headlineseq
            | /*empty*/;
    headline ::= headline1 headlinext;
    headline1 ::= spaces headindex spaces date headtext
        spaces "DELETED" EOL;
    headlinext ::= spaces headindexext headtext
    |;
    headindex ::= letter letter ;
    headindexext ::= spaces "(" letter letter ")";
    headtext ::= arb( EOL);
    date ::= digit digit "/" digit digit ;
    exchange ::= any("NAO");
    tickername ::= len(3);
    pagespec ::= digit digit "/" digit digit ;
    .
```

Figure 8. Example Grammar

6.3. Generators

At various points in the parse, special functions, termed generators, are invoked to convert the value lists of a set of attributes into complete tuples to be stored into the local relational database. In this way, results collected during the traversal are available for later user manipulation without the overhead involved in invoking the external database again.

The attribute value lists by themselves do not constitute a database relation. They must be converted into tuples and inserted into the relational database. Generators are functions for converting the value lists of a specified set of attributes into tuples.

Generators are constructed from attribute names composed with two possible primitive functions: cross product and dot product. Typically, two attributes are combined with dot product when they are in one to one correspondence. Cross product is usually used when two attributes have a one to many relationship Often, one of the attributes has a single element in its value list.

Figure 9(a,b,c) shows a separate example of the use of generators. Figure 9a indicates the stock price history for the company with ticker name CBU. When this screen is parsed using an appropriate grammar, it will fill the value lists of the relevant attributes: Company, Dates, Prices (see figure 9b). Suppose that the accumulated data is intended to fill in the relation described in figure 9c. The appropriate generator for converting figure 9b to figure 9c may be defined as

Company X (Dates • Prices).

This generator pairs the Dates and Prices on a one to one basis using dot product. Then, each of these pairs is combined with the company to produce the final result shown in figure 9c.

CBU

Date	Price
1/1/75	$52
2/1/75	$48
.	.
.	.
.	.
12/1/75	$57

Figure 9a. Sample screen output.

Company = { CBU }

Dates = { 1/1/75, 2/1/75, ... 12/1/75 }

Prices = { $52, $48, ...$57 }

Figure 9b. Value lists after parsing.

QUOTES	Company	Dates	Price
	CBU	1/1/75	$52
	CBU	2/1/75	$48
	.		
	.		
	CBU	12/1/75	$57

Figure 9c. Relation schema and contents.

7. COMMUNICATIONS

The communications subsystem is responsible for connecting to some physical carrier and then connecting to some particular external database. This involves two steps. First, a channel must be created over which commands can be sent from Diverse to the appropriate medium. Second, commands must be sent over this channel to set up the connections to networks and external databases.

The channel is established using the Unix[4] 4.2 pseudo-teletype device. A pseudo-teletype has two parts: a terminal device and a master device. The terminal device acts like an ordinary terminal device, but it has the property that it is controlled by another program. Thus, output sent to the pseudo-terminal can be read by the controlling program. Conversely, the controlling program can send input to the pseudo-terminal. Diverse acts as the controlling program for a pseudo-terminal attached to a standard Unix command processor (shell) program. Thus, Diverse can send commands to, and receive output from, other programs.

Once the channel is established, Diverse must connect to the appropriate medium. Assuming that this is the phone system, Diverse must connect to a dial-out modem and cause it either to dial the phone number of an intermediate network (e. g., Telenet) or to dial the external database directly.

As a short-cut, Diverse uses an existing program, the Unix Tip program, to handle the modem operations. Tip is a terminal emulation package for Unix. It can be invoked with a phone number and a baud rate as arguments, and then Tip will will perform all necessary dialing and connection functions. Once Tip has a connection, it transparently passes character data between the user and the phone connection.

To see how Tip is invoked, examine the first line of figure 6. The initial state ("cshstate") assumes that the channel to the Unix shell is running. It send as input a string invoking Tip with a baudrate and an phone number as argument. A few lines down, the actual command is shown: "tip -1200 3376060". This creates a 1200 baud phone connection to Telenet. The graph then moves to the state called "tipstate", and from that point on, commands are passed through Unix, through Tip, and over the modem to Telenet.

---

[4] Unix is a trademark of AT&T Bell Laboratories.

For other media, the general rule is to use existing programs where possible. For ethernet, for example, the remote commands of Berkeley Unix 4.2 (rcp, rsh, etc.) are used to connect to the external database.

8. AN EVALUATION

A working prototype of Diverse is implemented, and it is is capable of converting simple requests into the complex sequence of actions needed to answer those requests from external data. The prototype is written in an object-oriented version of Lisp called Objtalk [Rathke 83]. Objtalk builds on Franz Lisp by adding classes, message passing, and multiple inheritance. The pseudo-teletype control is written as Lisp callable C procedures. The program consists of about 2200 lines of lisp, and 400 lines of C.

Sufficient experience has been gained with the prototype to assess how well the original goals were met and to see the parts that are not completely successful. First, the current architecture can deal with heterogeneous systems and the associated command languages very well. The key feature here is the use of formal parsing techniques to connect database attributes with attributed parse trees. This parsing approach is quite effective in handling the text output of external databases. Second, the use of a relational database as the common format plus the attribute driven state machine graphs allow Diverse to combine data from multiple sources in an effective manner. Finally, Diverse is successful in hiding the actual databases from the user. This is due to the use of the state graphs, the automatic planning features, and the use of the a relational database as the user interface.

On the minus side, telephone line noise turns out to be one of the more serious problems for Diverse. Spurious characters or dropped characters can completely foil the parsing performed by Diverse. Work is currently under way to augment the Diverse parser with some powerful error correction techniques taken from compiler

research [Rohrich 80]. Actually, preliminary results indicate that these compiler techniques can be used two handle two problems:

(1) Noise on telephone lines,

(2) Nonessential changes in the output format from the external database: messages of the day, for example.

One thing that these techniques cannot solve is the corruption of output data (i.e., corruption of table entries). It may be that the corruption does not prevent the parse from completing successfully, but the data are still incorrect. Correcting such errors requires Diverse to have more semantics about the data it is collecting. This may necessitate a departure from a strictly relational database model to some sort of semantic model [Hammer 81, King 85].

At the moment, it is difficult to augment the knowledge base to add new states to existing graphs or to add whole new databases. A simple language exists for specifying this information and transforming it into the necessary Lisp structures. It would be better if Diverse had an associated knowledge acquisition system. This would involve some sort of graphic script editing capability. Work on such a facility is in preliminary stages.

The current grammars for screens do not sufficiently capture all of the structure perceptible to people. Most screens have a specific format, such as a table, or a list, or a menu. A promising alternative to specifying the grammar is to specify a more structured format in a parameterized form and then let Diverse generate the appropriate grammar. Given this extra information, Diverse could possibly define the generator functions and also do better error recovery.

Diverse is currently capable of extracting data, but it is not capable of modifying its external databases. There are a number of difficult issues here involving concurrency and recovery for which there are no acceptable answers as yet.

9. RELATED WORK

Information retrieval researchers have produced the most interesting and advanced work that is comparable to Diverse. Here, information retrieval (IS) refers to systems that extract text based on a user supplied set of keywords. Although there are a number of such systems [Goldstein 78, Meadow 82, Preece 80][5] the most advanced system appears to be CONIT [Marcus 81,83]. CONIT was designed to provide a uniform interface to existing information retrieval systems via a standardized command language. An interesting feature of CONIT is its use of a rule-based (or production) system to interpret both user commands and external IS responses. A CONIT rule is invoked whenever its *match string* is found at the head of some input stream. When the rule is invoked, it can send messages to the user and the external system, and it can perform an arbitrary computation

CONIT and Diverse are oriented to different domains (keyword search versus database retrieval). From the published work, it would appear that CONIT is somewhat less transparent than Diverse. That is, a CONIT user must use explicit commands to connect and disconnect. By contrast, a Diverse user specifies relations and then Diverse picks the database and transparently connects and disconnects. The rule based approach of CONIT is better able to respond to unexpected sequences of input. Diverse's graphs, by contrast, enforce sequential operations and this can be awkward in certain circumstances. On the other hand, since rules are inherently dynamic, CONIT cannot easily pre-plan its access patterns. Diverse, though, can examine its graphs and, based on conditions, choose the best access plan.

Software engineering has also produced some work of interest. The RITA project (Rand Intelligent Terminal Access) [Waterman 78] and the CONSUL/CUE project [Kaczmarek 81,84] are attempts to provide intelligent agents for programmers. RITA

[5] See [Marcus 83] for a comprehensive review and bibliography.

was based upon a production system that attempted to learn from the activities of the programmer. It also could deal with external systems, such as the Arpanet. While RITA could provide, in theory, smart-terminal access to external databases, it does not appear that it could provide much help in interpreting and integrating data obtained from those external sources. CONSUL/CUE, like RITA, attempted to provide a common interface to computer programs (as opposed to databases). It relied on a natural language interface plus structures for representing external programs.

Lochovsky and Tsichritzis at the University of Toronto also explored the problem of accessing external databases [Lochovsky 81]. Their emphasis was on the user interface, and it implicitly used a form of videotex (page-oriented) model. Under this framework, all data is cast into pages of data and its natural form is suppressed. While the user interface is important, The Diverse system attempts to provide more sophisticated understanding of the contents and structure of the external databases. It too casts much of that data into a fixed form (the relational model), but its use of text as common model may allow other structures to be used. Finally, and in contrast to Diverse, the Toronto work does not seem to promote integration of data from several sources.

Some distributed database systems should be mentioned to show the contrast with Diverse. Systems such as R* [Lindsay 80] and Multibase [Smith 81] also provide access to distributed data, but they require modification to the existing database managers or even a common database system (albeit "autonomous") at each computer. Diverse requires no such modifications or commonality.

The idea for Diverse grew from the author's previous work on a federated database [Heimbigner 82], which provided another architecture for combining existing databases, and his Syntax-directed database system [Heimbigner 84b], which applies the concept of parsing to provide a relational database interface to text files. A very

preliminary discussion of the requirements for Diverse was presented in [Heimbigner 84a].

10. SUMMARY

This paper presents the detailed architecture of the Diverse system, which is designed for sophisticated access to external database systems. Diverse allows a user to indicate the information that is of interest and then it automatically retrieves the requested data. It is capable of integrating data from several external sources. The Diverse system is built upon a relational database system for storing data, a knowledge base about the possible states of external databases, and a syntax-directed database system for extracting data from the output of an external database.

REFERENCES

[Goldstein 78] Goldstein, C. M., and Ford, W. H., "The User-Cordial Interface", *Online Review* 2(3):269-275 (1978).

[Griswold 71] Griswold, R. E., Poage, J. F., and Polonsky, I. P., *The SNOBOL4 Programming Language* (2nd Edition). Prentice-Hall, 1971.

[Hammer 81] Hammer, M. and McLeod, D., "Database Description with SDM: A Semantic Database Model", *ACM Transactions on Database Systems* 6(3):351-386 (September 1981).

[Heimbigner 84a] Heimbigner, D. M., "Towards an Integrated Environment for Accessing External Databases", *Proceedings of the Second ACM-SIGOA Conference on Office Information Systems,* Toronto, Canada, 25-27 June 1984.

[Heimbigner 84b] Heimbigner, D. M., "A Syntactic Database Model." Department of Computer Science Technical Report CU-CS-283-84, University of Colorado, Boulder, December 1984.

[Heimbigner 82] Heimbigner, D. M., and McLeod, D., "A Federated Architecture for Information Management", *ACM Transactions on Office Information Systems,* 3(7) (July 1985).

[Kaczmarek 81] Kaczmarek, T., Mark, W., and Wilczynski, D., "The Cue Project", Technical Report ISI/RS-83-1 (May 1983), Information Sciences Institute, Marina del Rey, CA.

[Kaczmarek 84] Kaczmarek, T., Mark, W., and Sondheimer, N., "The Consul/Cue Interface: An Integrated Interactive Environment", Technical Report ISI/RS-83-126 (April 1984), Information Sciences Institute, Marina del Rey, CA.

[King 85] King, R., and McLeod, D., "Semantic Database Models", in *Database Design.* Prentice Hall, 1985, S. B. Yao ed.

[Knuth 68] Knuth, D. E., "Semantics of Context-free Languages", *Mathematical Systems Theory* 2(2):127-145.

[Korth 84] Korth, H. F., Kuper, G. M., Feigenbaum, J., van Gelder, A, and Ullman, J. D., "System/U: A Database System Based on the Universal Relation Assumption", *ACM Transactions on Database Systems* 9(3):331-347 (September 1984).

[Lindsay 80] Lindsay, B., and Selinger, P. G., "Site Autonomy Issues in R*: A distributed database management system", Research report Rj2927, IBM Research Laboratory, San Jose, Ca, September 1980.

[Lochovsky 81] Lochovsky, F. H., and Tsichritzis, D. C., "Interactive Query Languages for External Data Bases", Computer Systems Research Group Technical Report, University of Toronto, March 1981.

[Maier 83] Maier, D. and Ullman, J. D., "Maximal Objects and the Semantics of Universal Relation Databases", *ACM Transactions on Database Systems* 8(1):1-14 (March 1983).

[Marcus 81] Marcus, R. S., Reintjes, J. F., "A Translating Computer Interface for End-User Operation of Heterogeneous Retrieval Systems. I. Design, II. Evaluation", *Journal of the American Society for Information Science* 32(4):287-317 (July 1981).

[Marcus 83] Marcus, R. S., An Experimental Comparison of the Effectiveness of Computers and Humans as Search Intermediaries", *Journal of the American Society for Information Science* 34(6):381-404 (November 1983).

[Meadow 82] Meadow, R. S., and Reintjes, J. F., "A Computer Intermediary for Interactive Database Searching. I. Design", *Journal of the American Society for Information Science* 33(5):325-332 (September 1981).

[Preece 80] Preece, S. E., and Williams, M. E., "Software for the Searcher's Workbench", *Proceedings of the 43rd ASIS Annual Meeting,* Volume 17, October 1980, pages 403-405.

[Rathke 83] Rathke, C., and Laubsch, J. H., "OBJTALK: Eine Erweiterung von LISP zum objektorientierten Programmieren", In *Objektorientierte Software- und Hardwarearchitekturen,* Teubner Verlag, 1983, H. Stoyan and H. Wedekind, ed., pp. 60-75,

[Rohrich 80] Rohrich, J., "Methods for the Automatic Construction of Error Correcting Parsers", *Acta Informatica* 13(2):115-139 (1980).

[Smith 81] Smith, J. M., Bernstein, P. A., Dayal, U., Goodman, N., Landers, T., Lin, K. W. T., and Wong, E., "Multibase: Integrating heterogeneous distributed database systems", Proceedings of the National Computer Conference (June 1981), AFIPS press, Arlington Va., pp. 487-499.

[Ullman 82] Ullman, J. D., *Principles of Database Systems* (2nd edition). Computer Science Press, 1982, pp.309-313.

[Waterman 78] Waterman, D. A., "Exemplary Programming in RITA", *Pattern-Directed Inference Systems,* pages 261-279, D. A. Waterman and F. Hayes-Roth (eds.), Academic Press 1978.

DIVERSE: A USER AGENT FOR ACCESSING EXTERNAL DATABASES

by

Dennis Heimbigner

CU-CS-306-85 July, 1985

University of Colorado, Department of Computer Science, Boulder, Colorado.