MIMSY: A System For Analyzing Time Series Data in the Stock Market Domain

William G. Roth*
Tandem Computers, Cupertino, CA.

Raghu Ramakrishnan
University of Wisconsin, Madison

Praveen Seshadri †
University of Wisconsin, Madison

Abstract

This paper describes a real–world application built on top of the CORAL deductive database system. This application is meant to demonstrate the power of CORAL not only as a deductive database but also as a generic extensible database system. Mimsy is a stock market historical reporting system that can answer questions about daily stock market pricing data. The paper describes the Mimsy system, and issues related to its design and implementation.

1 Overview

CORAL [RSS92] is an extensible deductive database system developed at the University of Wisconsin. While providing all the functionality of a logic programming environment, CORAL also provides the functionality of a general–purpose extensible database system [RSSS93]. The system includes an interpreter capable of processing a declarative rule–based language and an imperative environment where CORAL can be accessed from within a host language(C++ [ES90]). This environment is geared toward the building of non–trivial database applications.

Mimsy is a system for asking questions about stock market data, and has been built on top of CORAL. The goal is to provide an SQL–like language specifically geared toward the application of querying stock market data. Mimsy is inspired by a commercial application sold by Logical Information Machines, Inc., called MIM[1] [Lew92][2]. Like MIM, Mimsy allows the user to execute ad hoc queries against the stock market data. It should be noted that Mimsy is at once both more powerful and less powerful than MIM. For example, MIM has an extensive set of date primitives, whereas Mimsy has only a rudimentary

The work of the authors was supported by a David and Lucile Packard Foundation Fellowship in Science and Engineering, a Presidential Young Investigator Award with matching grants from DEC, Tandem and Xerox, and NSF grant IRI-9011563.

*The work of this author was performed while he was at the University of Wisconsin, Madison, and was partially supported by the grants listed above.

†The addresses of the authors are Tandem Computers, 10100 Tantau Avenue, Mail Stop 251-04, CA 95014, USA, and Computer Sciences Department, University of Wisconsin, Madison, WI 53706, USA. The authors email addresses are roth_bill@tandem.com, raghu@cs.wisc.edu and praveen@cs.wisc.edu.

[1] Logical Information Machines and MIM are registered trademarks of Logical Information Machines, Inc.

[2] "Mimsy" is meant to be thought of as an adjective meaning MIM–like.

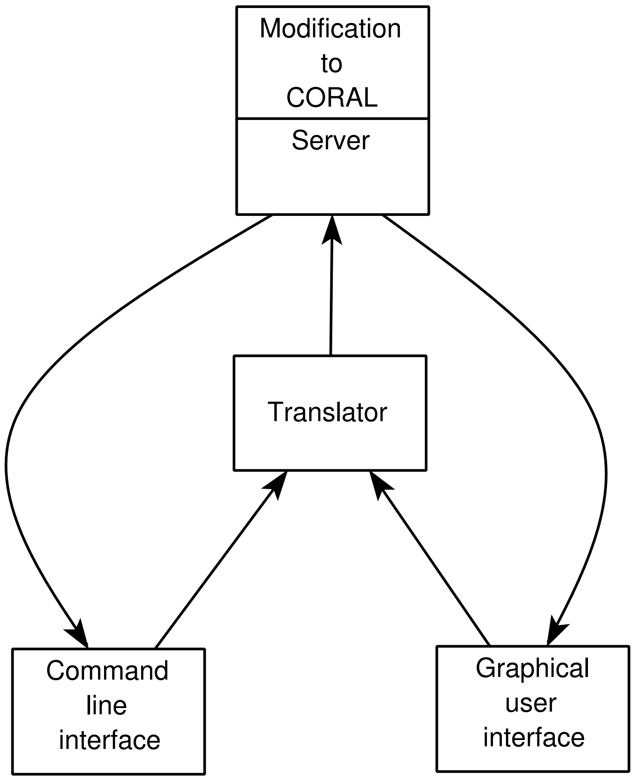

Figure 1: The design of the Mimsy system

subset. However, MIM's range of extensibility is rather limited, while Mimsy's extensibility has the full range of expressiveness of declarative CORAL, by virtue of the fact that it is built on top of CORAL.

Mimsy consists of three logical components.

- an interface that accepts queries in the form of the Mimsy query language.
- a translator that converts the Mimsy query into corresponding CORAL queries.
- the Mimsy server that processes the CORAL queries and returns the answer back to the interface for display to the user.

A command line interface is provided for interactive querying, and a graphical point-and-click interface for composing queries is also available. Utilities for sorting and graphing the results of queries are provided. This paper, however, focuses on the Mimsy language and implementation issues. The interface is described in detail in [Rot93].

The following points are worth emphasizing:

1. Mimsy can deal with significant amounts of data and quite complex queries, providing interactive responsive time. The recursive query capabilities of CORAL are necessary for expressing many natural concepts (e.g., a "bull market", "consecutive peaks", etc.) in this domain.
2. It demonstrates the power of CORAL's extensibility features.

3. It illustrates the use of CORAL in a client-server mode.
4. It is a substantial application that was completed primarily by one graduate student over one semester. We believe that this is considerably less than the effort that would be involved in building a similar package from scratch without the use of a system such as CORAL.

2 The Language

The basic structure of a Mimsy query is as follows:

```
select <something>
when <something-else>
save as <something-to-be-saved>
```

Both the `when` and the `save as` part of the query are optional. The `select` clause determines what data will be returned to the user. The `when` clause determines for which dates the data in the `select` clause will be shown. The `save as` clause specifies that the answers should be stored in a relation instead of displaying the results of the query on the screen. We discuss how the saved relation can be used in subsequent Mimsy queries in Section 6.2.

The data operated on by Mimsy is called a *series*. A series is a vector of price data. A series can be thought of as a binary relation with the first column specifying the date or time index, and the second column specifying the value of the series at that index. In Mimsy, a series is identified by its ticker symbol and an identifying attribute. For example, `close of abc` represents the close series for the stock whose ticker symbol is `abc`.

Operations, known as *aggregates*, can be applied to series to produce values. Aggregates and a time range are applied to base series. For example `the 30 day average of close of abc` represents the 30 day moving average of the stock whose ticker symbol is abc.

2.1 The Select Clause

In the Mimsy language the `select` portion of the query is comprised of one or more select attributes separated by commas. A select attribute is composed of a select expression followed by an optional `repeated` clause. A select expression is a series, aggregate on a series, or an arithmetic expression involving a series or an aggregate. Each select expression can be offset by a time period. Examples of the select clause are shown below, that demonstrate the use of the `repeat` clause and the `offset` clause.

```
select close of ibm;
select 3 * close of ibm repeated for 10 days;
select the 4 day average of close of ibm + 1;
select close of ibm - close of ibm 1 day ago
    repeated from the previous 2 days to
    the next 3 days;
```

2.2 The When Clause

The `when` clause is a list of predicates, separated by logical connectives that choose the dates for which the data in the `select` clause is to be shown. The `when` clause comes in 3 flavors: relational operator clauses, `change` clauses, and `crosses` clauses. All of these clauses can be further modified by a `condition interval`, which is a date range for which the condition must hold true.

The relational operator clause tests one select expression against another for a specific date. For example:

```
select close of abc when close of abc > 12;
select close of abc when close of def > close of b;
select close of abc when close of abc > close of b * 6;
```

For these queries, the implication is that the condition holds over 1 day. For a longer date range, the condition interval can be used. For example:

```
select close of xrx when close of xrx > 12 over 3 days;
```

The `change` clause tests whether a select expression is up or down over some period.

```
select close of abc when close of abc is up at least 10;
select close of abc when close of abc is down more than close of b;
select close of abc when close of abc is up more than 10%;
select close of abc when close of x is not down;
```

The `crosses` clause tests when one series crosses another. A cross occurs when, for example, one stock's price moves above the price of another when the first stock was smaller or equal in price to the second stock in the previous time period. For example:

```
select close of abc when close of abc crosses close of b;
```

This query will select the `close of abc` on all dates when the `close of abc` crosses above or below the `close of b`. The query can also specify either "above" or "below" in the `crosses` clause to specify which type of cross is of interest. For example:

```
select close of abc
when close of abc crosses above
    the 30 day average of close of abc;
```

2.3 Aggregates

Aggregates compute a single answer from the time range and series that are given as arguments. The aggregates included in the system are shown in Figure 1. The `average` aggregate computes a moving average of a series for the time period specified. The definition of `min`, `max` and `sum` are the same as in SQL. `Move` represents the total change in price of the series from the beginning of the time period to the end of the time period. `Pcmove` represents the total percentage change in price of the series from the beginning of the time period to the end of the time period.

All aggregates are used as follows, using `average` as an example:

#	Name	Mimsy Aggregate name
1	Average	average
2	Max	max
3	Min	min
4	Sum	sum
5	Move	move
6	Percentage Move	pcmove

Table 1: Mimsy aggregates

```
select the 5 day average of close of ibm
when the 3 day average of ibm is up more than 10%;
```

This query will show the 5 day moving average of the close of IBM on each day when the 3 day average rises by more than 10 percent in 1 day.

3 The Translator

Once a query has been received by either the command line interface or the graphical interface, the string is sent to the translator. If the query string acquired from the interface parses correctly, it is translated into CORAL commands and sent to the server over a socket connection. The result of the query, when received from the server, is read from the socket and sorted, and then displayed either textually or graphically.

The overall task of the translator is simple. It generates CORAL rules for the `when` clause so that all its results are collected into one temporary relation. This relation corresponds to the list of all dates that satisfied all the conditions in the `when` clause. These are the dates for which the `select` clauses will be evaluated. Separate abstract syntax trees are genarated for the `select` and `when` clauses, and are subject to various transformations. These transformations handle date ranges, and boolean expressions. Optimizations to handle aggregates and REPEAT clauses are also performed. Finally, the CORAL code corresponding to the Mimsy query is generated and sent to the server. The details of the translation are fairly complex and are described in [Rot93].

4 Series Data

There are currently 10 base series each defined for 165 stocks going back 5 years. The data for these series comes from the University of Chicago's Center for Research in Security Prices (CRSP). We have actually used a small fraction of the available data, which covers over 1600 stocks, going back over 20 years. It is important to note that CORAL only brings the data relevant to stocks in a given query into main memory; this loading is done very efficiently. Considering the entire data set supplied by CRSP will not affect performance significantly.

The series and their keywords in Mimsy are shown in Table 2. Series 1–5 are base series. That is, they are are actual values describing some property of the stock. Series 6–10 are composite series that represent some kind calculation performed by CRSP on the base data. These composite series are pre–calculated and stored as base data.

#	Name	Mimsy Sequence name
1	Close	close
2	High	high
3	Low	low
4	Volume	volume
5	Shares outstanding	shares
6	Beta	beta
7	Beta return	betaret
8	Capitalization	cap
9	Return	ret
10	Standard deviation	stdev

Table 2: Base series defined in Mimsy

5 The Server

The server is an iterative, connection–oriented server constructed as an extension to the basic CORAL system. Operationally, the server front–end merely listens to a socket, treats whatever comes over the socket as commands to the CORAL system, and returns the result back to the originator. However the underlying CORAL database engine has been extended to perform efficiently in this application domain. A new type of relation, called a *fast array relation*, has been added to support stock data, and all of the series aggregates have been coded directly as CORAL "built–in" relations.

The server first loads the date translation table, which translates regular dates into date integers, and the CORAL program to handle the date offsets. Next, the server loads in the relation that defines the catalog of all valid series, and initializes the corresponding relations. The server then creates the `dates` relation, which is a unary relation that contains a date identifier for every valid date in the system[3]. In the next step, the server loads in semantic information that will be sent to and used by both the command line interface and the graphical interface. These catalogs include the currently defined series, properties of series and aggregate definitions.

5.1 Fast Array Relations

One of the classic problems of dealing with stock market data is that it does not fit very well into the relational model. In the relational data model, every "object" is treated as a table of an unspecified (though potentially large) number of columns. The cost of maintaining this generality often makes accessing stock market, and indeed all time–series, both space and time inefficient. Some Wall Street investment banks have even been known to use environments like APL [GR84, Ive62] that are geared toward dealing with vectors and matrices, and eschew the use of RDBMSs completely. Since CORAL is a general purpose deductive database system, using the basic system would result in similar inefficiencies.

In order to allow CORAL to operate efficiently on time–series data, a new type of relation has been added to CORAL which facilitates fast access of sequence data. *FastArrayRelation* is a subclass of the CORAL Relation class and was added to the CORAL class hierarchy using the extensibility features provided by the system. A filename is passed to the FastArrayRelation when it is created specifying the location of the data file. The data is loaded into the file the first time the relation is iterated over. The data is stored in the file as a binary array, so that it can be loaded directly into memory without any translation. It is assumed that the data in the file is in the host machine's numeric format.

[3] Note that the stock trading dates are not sequentially increasing. Holidays and weekends interrupt the sequentiality.

The first column is an integer index. The second column is the value for that index, and is either a double precision floating point number or an integer. The first column of the relation is not actually stored. When the data is loaded, the file header information contains the number of elements in the array and the lower bound. Using this information, the first column can be synthesized. It is assumed that FastArrayRelations are read–only, since the stock data is relatively static.[4] The second column is retrieved by a simple array access.

5.2 Aggregates

Mimsy aggregates are written to recognize opportunities for efficient execution. This is done by having the aggregate cache its arguments and some of its computations from the previous invocation. Note that forward motion of time is always assumed when an aggregate is being iterated across a range of dates. For example, if the `average` aggregate is called on day 1 through day 20 of a series, it will cache the number of days in the range, the name of the portfolio or series, and the sum of the values over the range. If the next invocation is for day 2 through day 21 of the series, to compute the average it is sufficient to get the value for the previous lower bound of the range, subtract it from the sum, get the value for new upper bound for the range, add it to the sum and divide by the number of days in the series. If the next invocation is not for the next day in the window, the cached values are discarded and the computation must proceed as if called on the first day of a range. The aggregates `min` and `max` are optimized similarly. The effect of having aggregates optimized in this way is that the aggregates will recognize many opportunities for optimizations. This obviates the need for any higher level optimizations other than sorting the data before the aggregates are executed.

6 Extensibility

Mimsy utilizes and demonstrates the extensibility of the underlying CORAL system. However, Mimsy itself is also extensible at all levels, so that it is possible to enhance the system with more data, further functionality and even greater query expressivity. This section discusses more how extensions to the query language are handled.

6.1 Extending the Server

Users can define a new series in one of two ways.

Data can be added to the server by placing a properly formatted file in the data directory. Tools are provided to appropriately format the data. To add a derived series to the server via a CORAL module, the module should be loaded in by the server during startup. In both cases, information about the new schema should be added to the server catalogs.

Definitions of new aggregates are added to the server in a similar fashion. An aggregate is defined as follows, using "average" as an example:

```
seqaverage(BeginDate,EndDate,Series,Answer).
```

[4] A sub-class of FastArrayRelations that allows inserts too is provided for the case of a time–series that needs to be saved permanently, or temporarily.

The aggregate computes the appropriate value from `BeginDate` to `EndDate` for the series or portfolio and returns the result in `Answer`. There are two ways to define new aggregates, as new built–ins to be compiled into the server, or as CORAL modules to be loaded into the server at startup. The catalog of aggregate definitions needs to be updated to include the new aggregate. Some of these aggregate definitions may involve the use of recursion (for example, a moving average), and the ability of CORAL to specify recursive rules is useful in this regard.

6.2 Extending the Language

It is likely that a user will want to pose some queries not readily expressible in the Mimsy language, but that can be expressed in the underlying CORAL language. For instance, in the stock market domain, there are a variety of queries that are recursive in nature (for example, determining maximal montonic trends, or local minima and maxima). These queries are easily expressible in CORAL since it supports recursive rules(the appendix has a couple of examples of such queries). We provide a way to define CORAL predicates and extend the Mimsy language to make these new definitions accessible; this *extensibility* allows a sophisticated user to customize the language. Indeed, this is also the mechanism used to refer to relations produced by the *save as* clause (see Section 2).

Extensibility in the Mimsy language is provided by allowing strings found at certain places in the grammar to be passed uninterpreted to the server. Also, variables in a query that refer to variables in extensibility strings are allowed to appear in `select` expressions. A special variable, "`Date`", can appear in the argument string. In the translation, the string `Date` will be replaced by the variable that represents the current date under consideration. For example, suppose we had a predicate "extend" that returned some value. For example,

```
select close of abc,"new_series(Date,Val)",Val
when "extend(Date,X)" and X > 12.
```

would be translated into:

```
whendates(D) += dates(D),extend(D,X),X>12.
?whendates(D),a_close(D,D0),new_series(D,Val),D1=Val,
   dateuntranslate(D,X),print(X,D0,D1).
```

The extensibility strings can contain any valid CORAL code, and are interpreted to a very limited extent by the Mimsy front–end. Some knowledge of the translation process of Mimsy to CORAL is therefore necessary to effectively use this feature.

7 Conclusion

In developing Mimsy, we have tried to demonstrate that CORAL provides support for realistic database applications. The extensibility features of CORAL are heavily utilized to ensure efficient performance. The design of the Mimsy system itself involved some interesting language issues and implementation decisions. While we have touched upon them briefly, they are discussed in detail in [Rot93]. The system is currently being extended to handle portfolio queries, requiring significant extensions both of the language and the implementation.

The experience of developing Mimsy offers evidence of the power of deductive systems. The total programming effort was about two man-months, of which about one month was spent on developing the graphical interface. The actual design of Mimsy and acquiring familiarity with CORAL took about a month. We note that the primary implementor of Mimsy, W.G. Roth, was relatively unfamiliar with many of the implementation details, such as relation interfaces, when he began this project. While he did have some familiarity with aspects of CORAL, this was probably offset by the fact that the C++ interface was in a state of flux during the course of this project. In any case, while such considerations are hard to quantify, it is fair to note that Mimsy was designed and developed in about a semester primarily by one programmer.

The CORAL system provided considerable support. First, the C++ interface facilitated the definition of a number of functions needed to support the Mimsy language. Second, the extensibility of CORAL was used to represent series data effficiently while retaining a relational view. Third, the ability to pose ad-hoc queries relies upon CORAL to parse and evaluate rules; this allowed us to focus on high-level issues relevant to Mimsy rather than details of query processing. Finally, the full power of CORAL rules can be used to define additional functions to add to the power of Mimsy, and some of the pre-defined functions are indeed defined in this way.

8 Appendix

CORAL rules defining the N-day Moving Average of a Series

```
% These rules use the moving average for the previous period to
% incrementally compute the moving average for the next period.

% series(SeriesName, Date, Value)
% movavg(SeriesName, NoOfDays, Date, Value)

movavg(Sname,N,Day1,A) :-
        t2(Sname,N,Day1,N,V), A=V/N.

movavg(Sname,N,D1+1,A) :-
        movavg(Sname,N,D1,A1),
        series(Sname,D1,Old),
        series(Sname,D1+N,New), A=A1+(New-Old)/N.

% t2 just computes the average for the first N days.

t2(Sname,N,Day1,1,V) :-
        from(Day1),
        series(Sname,Day1,V).

t2(Sname,N,D,M1+1,V1+V2) :-
        t2(Sname,N,D,M1,V1), M1<N,
        series(Sname,D+M1,V2).
```

CORAL rules defining the Runs(maximal monotonic trends) of a Series

```
% This program analyses a series and generates a result series
% that describes maximal decreasing/non-decreasing "runs" in the
% original series.

run(Sname,Date,Dir,max(<L>)) :- trend(Sname,Date,Dir,L).

% for each date, trend computes the longest current inc/dec (possibly
% non-maximal) "run"

% the following rules deal with the case when the "observed trend" (inc/dec)
% continues on the current date.

trend(Sname,Date,nondec,L+1) :-
        trend(Sname,Date,nondec,L), series(Sname,Date+L,V1),
        series(Sname,Date+L+1,V2), V2>=V1.

trend(Sname,Date,dec,L+1) :-
        trend(Sname,Date,dec,L), series(Sname,Date+L,V1),
        series(Sname,Date+L+1,V2), V2<V1.

% the following rules deal with a "change in direction"

trend(Sname,D1,nondec,1) :-
        trend(Sname,Date,dec,L), series(Sname,Date+L,V1),
        series(Sname,Date+L+1,V2), V2>=V1.

trend(Sname,D1,dec,1) :-
        trend(Sname,Date,nondec,L), series(Sname,Date+L,V1),
        series(Sname,Date+L+1,V2), V2<V1.

trend(Sname,Day1,nondec,1) :-
        from(Day1), series(Sname,Day1,V1), series(Sname,Day1+1,V2), V2>=V1.

trend(Sname,Day1,dec,1) :-
        from(Day1), series(Sname,Day1,V1), series(Sname,Day1+1,V2), V2<V1.
```

References

[ES90] Margaret A. Ellis and Bjarne Stroustrup. *The Annotated C++ Reference Manual.* Addison-Wesley, Reading, Massachusett, 1990.

[GR84] Leonard Gilman and Allen J. Rose. *APL: An interactive approach.* John Wiley & Sons Inc., New York, third edition, 1984.

[Ive62] Kenneth E. Iverson. *A programming language.* John Wiley & Sons Inc., New York, 1962.

[Lew92] Peter H. Lewis. A Fast Way to Discover Patterns in Vast Amounts of Data. *The New York Times*, pages 16–17, August 23 1992.

[Rot93] William Gibson Roth. MIMSY:A System for Analysing TimeSeries Data in the Stock Market Domain. Master's Thesis, University of Wisconsin, Madison, 1993.

[RSS92] Raghu Ramakrishnan, Divesh Srivastava, and S. Sudarshan. CORAL: Control, Relations and Logic. In *Proceedings of the International Conference on Very Large Databases*, 1992.

[RSSS93] Raghu Ramakrishnan, Divesh Srivastava, S.Sudarshan, and Praveen Seshadri. Implementation of the CORAL deductive database system. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1993.