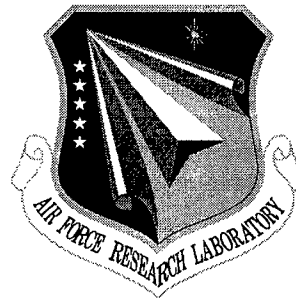


AFRL-IF-RS-TR-1998-75
Final Technical Report
May 1998



HIGH PERFORMANCE ACTIVE DATABASE MANAGEMENT ON A SHARED-NOTHING PARALLEL PROCESSOR

University of Florida

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. A0-D959

19980713 009

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

DTIC QUALITY INSPECTED 1

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

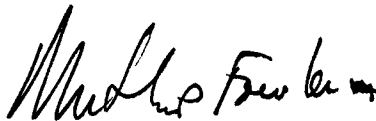
AFRL-IF-RS-TR-1998-75 has been reviewed and is approved for publication.

APPROVED:



JOSEPH P. CAVANO
Project Engineer

FOR THE DIRECTOR:



NORTHROP FOWLER, III, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/ITB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

HIGH PERFORMANCE ACTIVE DATABASE
MANAGEMENT ON A SHARED-NOTHING
PARALLEL PROCESSOR

Eric N. Hanson

Contractor: University of Florida
Contract Number: F30602-96-1-0190
Effective Date of Contract: 1 June 1996
Contract Expiration Date: 31 May 1997
Program Code Number: 6E20
Short Title of Work: High Performance Active
Database Management
Period of Work Covered: Jun 96 - May 97

Principal Investigator: Eric N. Hanson
Phone: (352) 392-2691
AFRL Project Engineer: Joseph P. Cavano
Phone: (315) 330-4033

Approved for public release; distribution unlimited.

This research was supported by the Defense Advanced Research
Projects Agency of the Department of Defense and was monitored
by Joseph P. Cavano, AFRL/ITB, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE May 1998	3. REPORT TYPE AND DATES COVERED Final Jun 96 - May 97		
4. TITLE AND SUBTITLE HIGH PERFORMANCE ACTIVE DATABASE MANAGEMENT ON A SHARED-NOTHING PARALLEL PROCESSOR		5. FUNDING NUMBERS C - F30602-96-1-0190 PE - 62301E PR - H137 TA - 00 WU - P1		
6. AUTHOR(S) Eric N. Hanson				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Florida CISE Department Gainesville FL 32611-6120		8. PERFORMING ORGANIZATION REPORT NUMBER N/A		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington VA 22203-1714		10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-1998-75		
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Joseph P. Cavano/IFTB/(3135) 330-4033				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) A new type of system was designed for testing trigger conditions and running trigger actions outside of a data base management system. This system processes triggers asynchronously, after triggering updates have committed in the source database.				
14. SUBJECT TERMS Active databases, triggers, asynchronous processing, data base management, parallel			15. NUMBER OF PAGES 36	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

High Performance Active Database Management on a Shared-Nothing Parallel Processor

Abstract

A new type of system for testing trigger conditions and running trigger actions outside of a DBMS was designed as part of this project. Such a system is called an *asynchronous trigger processor* since it processes triggers asynchronously, after triggering updates have committed in the source database. The architecture of a prototype asynchronous trigger processor called TriggerMan is described. TriggerMan is designed to be able to gather updates from a wide variety of sources, including relational databases, object-relational databases, legacy databases, flat files, the web, and others. TriggerMan achieves the ability to gather updates from so many sources using an extensible data source mechanism. TriggerMan can make use of the asynchronous replication features of commercial database products to gather updates. When cooperating with a source DBMS with direct support for asynchronous replication, TriggerMan can gather updates in an efficient and robust manner. TriggerMan supports simple, single-table (single data-source) triggers, as well as sophisticated multiple-table (multiple-data-source) triggers. It also will support temporal triggers using an extensible temporal function mechanism. Moreover, TriggerMan takes advantage of parallelism for high performance, utilizing a shared-nothing model of parallel computation.

1. Introduction

The main goal of this project was to develop technology to support efficient trigger processing for shared-nothing parallel database management systems. Mid-way through the one-year period of the grant, the design and implementation focus of the project shifted away from developing trigger extensions for the Paradise parallel DMBS, toward developing an independent system, an "asynchronous trigger processor" called TriggerMan [Hans97b]. This shift was brought on because it was not possible for us to get full source code for the Paradise DBMS from Wisconsin. Focusing effort on TriggerMan instead of Paradise was fully consistent with the original goals of the project, since TriggerMan is also a parallel trigger processing system. In fact, the shift was

beneficial in terms of the potential for technology transfer, since the prototype we develop will be usable with existing information systems applications built using current commercial database systems. The rest of this final report covers the design of TriggerMan, our goals for its future implementation, and plans for developing technology to support parallel asynchronous trigger processing.

There has been a great deal of interest in active database systems over the last ten years. Many database vendors now include active database capability (triggers) in their products. Nevertheless, a problem exists with many commercial trigger systems as well as research efforts into development of database triggers. Most work on database triggers follows the event-condition-action (ECA) rule model. In addition, trigger conditions are normally checked and actions are normally run in the same transaction as the triggering update event. In other words, the so-called immediate binding mode is used. The main difficulty with this approach is that if there are more than a few triggers, or even if there is one trigger whose condition is expensive to check, then update response time can become too slow. A general principle for designing high-throughput transaction processing (TP) systems put forward by Jim Gray can be paraphrased as follows: *avoid doing extra work that is synchronous with transaction commit* [Gray93]. Running rules just before commit violates this principle.

Moreover, many advances have been made in active database research which have yet to show up in database products because of their implementation complexity, or because of the expense involved in testing sophisticated trigger conditions. For example, sophisticated discrimination networks have been developed for testing rule conditions [Hans96]. In addition, techniques have been developed for processing temporal triggers (triggers whose conditions are based on time, e.g. an increase of 20% in one hour). Neither of these approaches has been tried in a commercial DBMS.

In this report, the author proposes a new kind of system called an *asynchronous trigger processor*, or ATP. An ATP is a system that can process triggers asynchronously, after updates have committed in a source database, or have been completed in some other data source. Processing triggers asynchronously avoids slowing down update transactions with trigger processing logic. Moreover, since an ATP could be used with many different source DBMSs, the effort to develop the trigger processing code could be amortized over use with more applications. Really, an arbitrary application program can be used to transmit descriptions of database updates (update descriptors) to the ATP, and triggers can be processed on top of these update descriptors. The ability to process triggers based on updates from many different sources can help make it economically viable to implement sophisticated trigger processing code.

We are currently developing an ATP called TriggerMan as a vehicle for investigating issues related to asynchronous trigger processing. A simple subset of the functionality discussed in this report has been implemented. We are actively doing the detailed design and implementation of the more advanced features of TriggerMan.

Part of the motivation for TriggerMan has been the surge in popularity of asynchronous replication features in commercial database systems. In actual practice, to achieve replication of data in a distributed DBMS, most database customers greatly prefer asynchronous replication to a synchronous replication policy based on distributed transactions using two-phase commit. The reason for this is that update availability and response time are both better with asynchronous replication. This was a motivating

factor behind the choice to move to an external, asynchronous trigger processor, which also would avoid slowing down updates. Furthermore, as shall be explained later, the update capture mechanism built in to asynchronous replication systems can be used to send update descriptors to an ATP.

With respect to related research, many active database systems have been developed, including POSTGRES, HiPAC, Ariel, the Starburst rule system, A-RDL, Chimera, and others [Wido96]. In addition, there has been a notion of "de-coupled" rule condition/action binding mode for some time, as introduced in HiPAC [McAr89]. However, the implicit assumption regarding de-coupled rule condition evaluation and action execution was that the DBMS itself would still do the needed work. This report outlines an alternative architecture that would off-load rule condition testing and action execution to a separate system.

Ultimately, the proposed TriggerMan/ATP architecture will provide an "active information server" capability that can support a wide variety of applications. This architecture will be able to support different tasks that involve monitoring of information sources, filtering of data, and selective propagation of information. TriggerMan can be used to augment traditional data management applications, as well as support new, distributed, heterogeneous information systems applications.

TriggerMan will be extensible in a number of ways, including the ability to add new data sources, new data types, and new temporal functions. To handle extended data types, the approach used will be similar to that used in object-relational database systems such as Informix Universal server.

2. The TriggerMan Command Language

Commands in TriggerMan have a keyword-delimited, SQL-like syntax. TriggerMan supports the notion of a connection to a remote database or a generic data source program. A connection description for a remote database contains information about the host name where the database resides, the type of database system running (e.g. Informix, Oracle, Sybase, etc.), the name of the database server, a userid, and a password. A single connection is designated as the default connection. There can be multiple data sources defined for a single connection. Data sources can be defined using this command:

```
define data source [connectionName.]sourceName [as localName]
[ ( attributeList ) ]
[ propertyName=propertyString,
...
propertyName=propertyString ]
```

Suppose a connection "salesDB" had been defined on a remote database called "sales." An example data source definition for the table "sale" in the sales database might look like this:

```
define data source salesDB.sale as sale
```

This command would read the schema from the salesDB connection for the "sale" table to gather the necessary information to allow triggers to be defined on that table.

Triggers can be defined using the following command:

```

create trigger <triggerName> [in setName] [-inactive]
from fromList
[on eventSpec]
[start time timePoint]
[end time timePoint]
[calendar calendarName]
[when condition]
[group by attr-list]
[having group-condition]
do action

```

Triggers are normally eligible to run as soon as they are created if a triggering event occurs. However, if the **-inactive** flag is specified the trigger remains ineligible to run until it is enabled later using a separate **activate trigger** command. The **start time** and **end time** clauses define an interval within which the trigger can be eligible to run. In addition, the name of a calendar object can be specified as part of a trigger. A calendar indicates "on" and "off" time periods. For example, a simple business calendar might specify "on" periods to be Monday through Friday from 8:00AM to 5:00PM. A trigger with a calendar is only eligible to be triggered during an "on" period for the calendar. In addition, whether a trigger is eligible to be triggered is determined by the logical AND of the eligibility criteria determined by (1) whether the trigger is active or not, (2) whether the current time is between the start time and end time, and (3) whether the associated calendar is in an "on" time period.

Triggers can be added to a specific trigger set, otherwise they belong to a default trigger set. The **from**, **on**, and **when** clauses are normally present to specify the trigger condition. Optionally, **group by** and **having** clauses, similar to those available in SQL [Date93], can be used to specify trigger conditions involving aggregates or temporal functions. Multiple remote tables (or other data streams) can be referenced in the **from** clause. This allows multiple-table triggers to be defined.

An example of a rule, based on an **emp** table from a database for which a connection has been defined, is given below. This rule sets the salary of Fred to the salary of Bob:

```

create trigger updateFred
from emp
on update emp.salary
when emp.name = "Bob"
do execSQL "update emp set salary=:NEW.emp.salary where emp.name=
'Fred'"

```

This rule illustrates the use of an **execSQL TriggerMan** command that allows SQL statements to be run against data source databases. The **:NEW** notation in the rule action (the **do** clause) allows reference to new updated data values, the new **emp.salary** value in this case. Similarly, **:OLD** allows access to data values that were current just before an update. Values matching the trigger condition are substituted into the trigger action using macro substitution. After substitution, the trigger action is evaluated. This procedure binds the rule condition to the rule action.

An example of a more sophisticated rule (one whose condition involves joins) is as follows. Consider the following schema for part of a real-estate database, which would be imported by TriggerMan using **define data source** commands:

```
house(hno,address,price,nno,spno)
salesperson(spno,name,phone)
represents(spno,nno)
neighborhood(nno,name,location)
```

A rule on this schema might be "if a new house is added which is in a neighborhood that salesperson Iris represents then notify her," i.e.:

```
create trigger IrisHouseAlert
on insert to house
from salesperson s, house h, represents r
when s.name = 'Iris' and s.spno=r.spno and r.nno=h.nno
do raise event NewHouseInIrisNeighborhood(:NEW.h.hno, :NEW.h.address)
```

This command refers to three tables. The **raise event** command used in the rule action is a special command that allows rule actions to communicate with the outside world [Hans97]. Application programs written using a library provided with TriggerMan can register for events. When triggers raise events, the applications registered for the events will be notified. Applications can run on machines running anywhere on the network that is reachable from the machine where TriggerMan is running.

3. System Architecture

The general architecture of the TriggerMan system is illustrated in Figure 1. Each box in this diagram represents a system component. These components can run on the same machine or different machines. Most components are single processes. The exception to this is the TriggerMan server component, which has a parallel internal structure, consisting of a number of virtual processors, or *vprocs*. The *vprocs* communicate with each other via message passing. Hence, the TriggerMan server code can be made to run with little modification on shared-memory multiprocessors, shared-nothing machines, and collections of SMP machines connected by an interconnect. The first parallel implementation is designed to run on an SMP. The *vproc* concept has been used before successfully in the implementation of parallel DBMS software, such as the Teradata system [Witk93].

In the current TriggerMan system that runs on an SMP, *vprocs* are software objects that live in the same address space. Each *vproc* owns multiple threads, including:

- a *matching thread* that processes update descriptors arriving from data sources to see if trigger conditions are satisfied,
- a *command server thread* that handles requests from client applications, and
- *rule action execution threads* to run rule actions.

The number of *vprocs* is normally made equal to the number of real processors in the system. In the current implementation, single-table triggers are allocated to different

vprocs in a round-robin fashion to allow parallel condition testing. A special client application called the Console allows a user to start the system, shut down the system, create triggers, define data sources, and run other commands supported by TriggerMan. Multiple threads, spread across the vprocs, allow parallel testing of trigger conditions on an SMP machine.

Two libraries that come with TriggerMan allow writing of client applications and data source programs. These libraries define the TriggerMan *client application programming interface* (API) and the TriggerMan *data source API*. The console program and other application programs use client API functions to connect to TriggerMan, issue commands, register for events, and so forth. Data source programs, such as a generic data source that sends a stream of update descriptors to TriggerMan, or

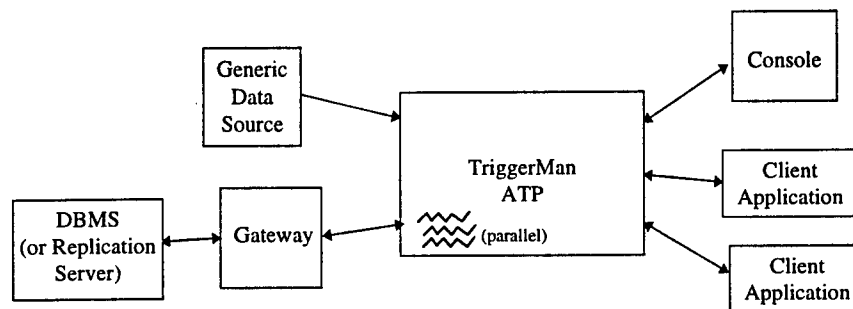


Figure 1 Architecture of the TriggerMan asynchronous trigger processor.

a DBMS gateway program that gathers updates from a DBMS and sends them to TriggerMan, can be written using the data source API.

4. Data Source Design

A flexible strategy is being designed to gather streams of update descriptors or other messages from data sources. A simple, generic data source could be an application that sends a stream of new data values to TriggerMan. Such a generic data source, as illustrated in Figure 1, would be a program written using the data source API. A more sophisticated data source could gather a stream of update descriptors from a database by cooperating with the replication services provided by the DBMS. E.g. with Sybase, a gateway program, as shown in Figure 1, could be written using the TriggerMan data source API and the Sybase replication API [Syba96]. This Gateway program would transmit update descriptors received from the Sybase replication server and propagate them to TriggerMan. A different gateway program could be written for each potential DBMS that might serve as a data source. For databases for which no replication service exists, a gateway program could be written that would query the database periodically and compare the answers to the queries to produce update descriptors to send to TriggerMan [Chaw96]. Alternatively, the gateway could trap inserts, updates and deletes using simple triggers in the source DBMS. Reading the database log is another alternative, but it is not usually realistic because DBMS vendors normally have a

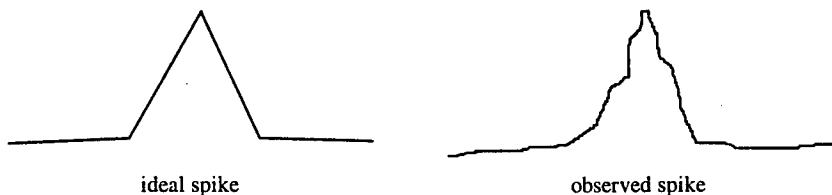
proprietary log format that other systems are not allowed to read, since the vendor reserves the right to change the log format.

TriggerMan will maintain catalogs and other persistent state information using a transactional DBMS. To preserve transaction semantics, the first approach to handling the stream of updates from a DBMS will be to apply the updates in TriggerMan, and run the resulting trigger actions, in commit order. The Sybase replication server, for example, presents updates in commit order, making this strategy feasible. The maximum transaction ID handled so far by TriggerMan will be recorded along with updates to TriggerMan's internal state information in a single transaction. If this transaction fails, the updates will be rolled back and will be re-applied later. Maintaining the maximum transaction ID applied so far will make sure TriggerMan does not forget to handle a transaction from the primary database.

An issue that will be addressed in the future is how to deal with high update rates in the data source databases. If updates are taking place at a high rate in the source DBMS, TriggerMan might not be able to keep up with the source if it must handle the updates in commit order. This is because it might not be possible to get enough concurrency or parallelism in the ATP if the updates are handled serially. Possible solutions to this problem will be considered, such as relaxing the requirement to handle updates in commit order (for database data/data sources) or in the order of arrival (for generic data sources). Piggybacking multiple update descriptors in a single message to the TriggerMan server and in a single broadcast to the vprocs may also make it possible to handle higher update rates.

5. Temporal Trigger Support

Temporal triggers are triggers whose conditions are based on changes in a value or set of values over time. For example, a temporal trigger could be defined to fire if the sales from a particular store rise by more than 20% in one month. Prior work on temporal triggers has focused on logic-based trigger languages [Sist95]. The difficulty with these languages is that the user must specify the trigger in a logic-based notation, and logic-based languages with quantifiers may be difficult for typical application developers to master. Moreover, certain kinds of temporal conditions may be quite useful, yet be extremely difficult or impossible to specify using temporal logic. For example, one might envision a temporal trigger that would fire if the price of a stock had a "spike" in value, where the definition of "spike" is based on some application-specific mathematical criteria, such as "the average root mean square difference on a point-by-point basis between the actual sequence of values (curve) and an ideal spike is less than a threshold value." An example of an ideal spike and what an observed spike might look like is given below:



The capability to detect a spike based on mathematical criteria would be much easier to express using an algorithmic language like C, C++, Java, or FORTRAN than using temporal logic. Moreover, temporal functions written in C, for example, may be able to evaluate temporal trigger conditions much more efficiently than the equivalent temporal logic-based condition evaluator.

Rather than use a temporal-logic-based language, we propose to use a set of basic temporal functions, including **increase**, **decrease** and several others, as well as temporal aggregates such as the **sum** and **count** of values over a certain time window. The benefit of using temporal operators to specify trigger conditions is that they are declarative, and relatively simple to understand – you say what you want, not now to achieve it. The implementation of the basic temporal functions will be provided as a standard part of the system. In addition, a *temporal function extensibility mechanism* is being developed to allow sophisticated application developers to write code to implement new temporal functions and register this code with the TriggerMan system. The extension code will be dynamically linked by the TriggerMan server when needed.

As mentioned earlier, the TriggerMan trigger language supports temporal condition specification through the use of the **group by** and **having** clauses familiar to users of SQL. For example, the following trigger will fire when there is an increase or decrease of more than 20% in the price of IBM stock in a six month period.

```
create trigger BigIBMchange
from stock
when stock.symbol = "IBM"
having increase(stock.price, "20%", "6 mo") or
    decrease(stock.price, "20%", "6 mo")
do raise event BigChange ("IBM")
```

The above trigger can be generalized to all "technology" stocks by introducing a **group by** clause and modifying the **when** clause, as follows:

```
create trigger BigTechStockChange
from stock
when stock.category = "technology"
group by stock.symbol
having increase(stock.price, "20%", "6 mo") or
    decrease(stock.price, "20%", "6 mo")
do raise event BigChange (:NEW.stock.symbol)
```

The **group by** capability is powerful since it allows triggers for multiple groups to be defined using a single statement.

Temporal functions can return boolean values (temporal predicates) and scalar values, such as integers and floating point numbers. These types of temporal functions can be composed in the **having** clause to form compound temporal conditions. For example, the following temporal function might compute a moving average of a value over a time window of width `window_size`:

moving_avg (expr>window_size)

The following example shows how this function could be used in conjunction with the **increase** function to detect when the 10-day moving average of the price of Oracle stock increases by more than 15%:

```
create trigger ORACLE_TREND
from stock
when stock.symbol = "ORCL"
having increase(moving_avg(stock.price, "10 days"), "15%")
do ...
```

Values of temporal aggregates computed in the **having** clause will sometimes need to be used in the trigger action. The trigger language needs a way to support this form of condition/action binding. When it is not ambiguous, the name of the temporal aggregate function can be used in the trigger action, as in the following example:

```
create trigger HighYearlySales
from sale
group by sale.spno
having sum(sale.amount, "1 yr") > 1000000
do execSQL "append to highSales(:NEW.sale.spno, :NEW.sum, Date())"
```

If the same function appears multiple times, the **as** operator can be used to bind names to the values produced by those functions, e.g.:

```
create trigger HighSalesAndCommssions
from sale
group by sale.spno
having sum(sale.amount, "1 yr") as s1 > 500000
and sum(sale.commission, "1 yr") as s2 > 50000
do execSQL "append to highSales(:NEW.sale.spno, :NEW.s1, :NEW.s2,
Date())"
```

5.1. Adding New Temporal Functions

The temporal function mechanism in TriggerMan is designed to be extensible. A new temporal operator can be defined using this notation:

```
define temporal function returnType funcName (argumentDefinition)
dynamicLinkLibraryName functionPrefix
```

The `dynamicLinkLibraryName` is the name of the dynamic link library (DLL) where relevant functions are kept. The DLL consists of compiled C code. The `functionPrefix` is the prefix of the name of all functions that are relevant to the temporal operator being defined. A temporal function's argument list can specify normal arguments, as well as initialization arguments used to initialize the state of the temporal function. Initialization arguments are preceded by the keyword **init**. A default value can also be provided for **init** arguments. For example, a new function to compute an exponential average could be

registered with the system like this (the C:\tmanlib\expavg.dll file is a dynamic link library):

```
define temporal function double expavg (double newValue,  
    init double multiplier = 0.9) "C:\tmanlib\expavg.dll" "double_expavg"
```

Also, functions can be overloaded. For example, expavg can be re-implemented for different types, such as float, and the system will automatically use the right function depending on the data types with which it is called.

Functions with the proper formats and naming conventions need to be available in the dynamic link library to implement a temporal function. This kind of extensibility technique is similar to that used in extensible database systems such as POSTGRES [Ston90] and the Informix Universal Server [Info97]. To test the condition of a temporal trigger, state information related to the temporal trigger condition must be maintained. For example, state could simply be a number and a multiplier for a simple exponential average. For most temporal triggers involving temporal aggregates such as **increase**, **decrease** etc., the state of the trigger will be a time series. In addition, if there is a **group by** clause, one piece of state information (normally a time series) must be maintained for each group. The functions required in the DLL used to implement a temporal operator must be able to create a temporal trigger state object, delete the object, update the state of the object based on the arrival of new data or the passage of time, and get the current value of the temporal operator.

We use the C language to define the interface to a temporal operator's state object. C is used instead of C++ or Java [Arno96] because (1) we want very high performance, meaning that fully compiled code is required, ruling out Java, and (2) because dynamic linkers in Windows NT and Solaris (Sun Unix) support dynamic linking of C much better than dynamic linking of C++. In addition, we provide a reusable time-series abstract data type implemented in C for use by developers of new temporal functions, as well as for our own internal use.

Full details of the formats of the C functions required in a DLL to define the behavior of a temporal operator are not given here. A general description of the functions required (but not a complete list) is as follows:

constructor	A function to build the internal state object for a temporal trigger (or a single group of a temporal trigger in case a group by clause is used). Usually the state object will contain a time series.
destructor	A function to free the space used by a temporal trigger state object when it is no longer needed (e.g. when a trigger is dropped).
provide new history value	A function to take a new value and time stamp to update the history information (usually a time series) contained in the state object
provide new current time value	A function to provide a new value of the current time. This can be thought of as a "clock tick" function. The state object should be updated as needed based on this new time value (e.g. to trim off some of the oldest history values that have moved out of the time window of interest). This function will be called periodically for some operators, such as "holds(emp.salary,

“>30000”, “2 years”)”. With “holds” and some other temporal operators, it may be necessary to fire even if no new history value arrives.

get current value Get the current value of the temporal operator (the result may be true/false or some other data type).

A protocol for firing temporal triggers needs to support both triggering when update events occur, and triggering when timers expire. The interface above allows TriggerMan to trigger both on update events and on timer expiration, as needed.

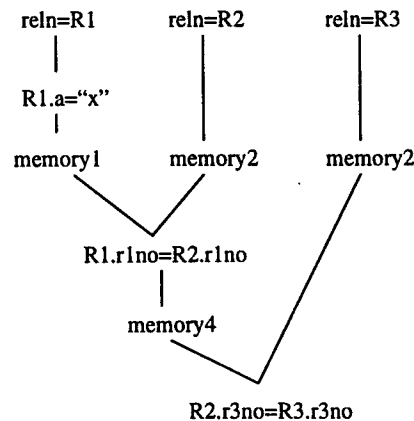
6. Support for Multiple-Table and Parallel Trigger Condition Testing

Allowing triggers to have conditions based on multiple tables greatly increases the power and expressiveness of the trigger language. However, efficiently testing multiple-table trigger conditions is a challenging problem. As part of prior work, we have developed an optimized multiple-table trigger condition testing mechanism known as the Gator network, a generalization of the TREAT and Rete networks used for rule condition testing in production rule systems such as OPS5 [Hans95]. We are investigating strategies that will allow development of a parallel version of the Gator network, as well as cost models and caching strategies specific to the environment of an asynchronous trigger processor. These new cost models and caching strategies are needed since TriggerMan runs in an separate address space and possibly on a separate machine from the DBMS or other data sources.

A parallel Gator network capability will be developed for TriggerMan to allow multiple table rule condition testing. A related approach has been successfully used to perform parallel rule condition matching in main-memory production systems using a Rete network [Forg82] organized as a global distributed hash table, or GDHT [Acha92]. The idea behind this strategy is as follows. A Gator network tree structure will be constructed for each trigger. Gator networks consist of nodes to test selection and join conditions, plus “memory” nodes that hold sets of tuples matching one or more selection and join conditions. For example, consider the following table schemas, trigger definition, and one possible Gator network for the trigger:

R1(r1no,a,b)
R2(r1no,r3no)
R3(r3no,c,d)

create trigger T1
from R1, R2, R3
when R1.r1no=R2.r1no
and R2.r3no=R3.r3no
and R1.a = “x”
then do ...



In this Gator network, memory1 logically contains the result of

```
select * from R1 where R1.a= "x"
```

Similarly, memory4 logically contains the result of

```
select * from R1, R2 where R1.a= "x" and R1.r1no=R2.r1no
```

In addition, memory nodes can be either stored or virtual. A stored node is like a materialized view. It actually contains the specified tuples. A virtual node is like a real view. It only contains a predicate defining which tuples should qualify. It does not contain the real tuples. Memory nodes at the leaf level, drawn at the top of a Gator network, are called alpha memory nodes. Inner memory nodes, holding join results, are called beta memory nodes. Only alpha memory nodes can be virtual.

A detailed discussion of how discrimination networks can be used for multiple-table trigger condition testing on a single processor can be found in [Hans96]. An outline of our approach to implementing a Gator network in parallel on a shared-nothing machine consisting of a set of vprocs is as follows:

1. The selection predicates will be allocated round-robin among the processors.
2. The tree shape of the network will be replicated on every vproc. The contents of alpha and beta nodes will not be replicated.
3. Stored memory nodes will be horizontally partitioned across the vprocs on a single attribute, normally a join attribute, using hash partitioning.
4. Stored memory nodes will be cached in their entirety in memory in TriggerMan on first use. Virtual memory nodes will never be cached. An LRU replacement strategy will be used with an entire memory node as the cache replacement granularity. (More sophisticated caching policies that allow caching a subset of a memory node are being considered).
5. For other attributes for which a fast access path is required to a memory node, such as additional join attributes or the primary key, parallel secondary hash partitions will be created. These are analogous to secondary indexes in a single-processor DBMS. For a memory node $N(\underline{nno}, X, \dots)$ if the node has a primary partition on X , then a secondary partition on the nno (primary key) field can be formed by creating a table $N_nno_index(nno, vproc_number)$ and doing a primary hash partition of N_nno_index on nno . A row in N_nno_index tells which vproc contains the tuple with a particular value of nno . This allows deletion of a tuple in N given its key (nno) value with two point-to-point messages, one to look up the processor where the tuple lives using N_nno_index , and one to actually delete the tuple.
6. Pattern matching will operate as follows:
 - a. When a tuple update arrives, a description of the update will be packaged as a "token" and will be broadcast to all the vprocs. A token contains a tuple or an old/new tuple pair, along with a tag describing what kind of operation was performed (insert, delete or update). If the operation is an update, then the identification of which fields were updated will also be included in the token.
 - b. Each vproc will test the token against its local collection of selection predicates. Event conditions from the **on** clause and regular selection

conditions from the **when** clause will both be treated logically as selection conditions on tokens.

- c. For each matching selection predicate, the token will be forwarded onward down the network. For a single-table trigger, a match against the selection predicate for the trigger causes the trigger action to fire. The trigger action is executed once for each matching tuple, on the processor where the match is detected.
- d. For a multiple-table trigger, a match of a token against a selection predicate causes an update to the memory node below the selection predicate, and then the token is joined to a neighboring memory node (see below for a discussion of parallel joins of tokens to memory nodes). Resulting joining pairs of tuples (intermediate tokens) are then propagated down the network. When a token arrives at the bottom of the network (the P-node) the trigger action is run for that token.

In general, the following operations may need to be performed on a memory node: (1) insert a tuple, (2), delete a tuple and (3) join a tuple to the node. Each of these operations can be performed using one or two point-to-point messages from one vproc to another based on the hash partitioning scheme used. Consider memory2 from the Gator network shown earlier. Tuples may need to be inserted into or deleted from memory2. In addition, a tuple may sometimes be joined to memory2. This requires finding all the memory2 tuples such that `memory2.r1no=CONSTANT` for some constant value extracted from the `r1no` field of the tuple being joined to memory2. In this example, memory2 would be hash partitioned across the vprocs on the `r1no` column. Hence, to find all memory2 tuples satisfying `memory2.r1no = CONSTANT`, a vproc must do the following. First, use the hash function `h` used to define the partition of memory2 to find `h(CONSTANT)`, yielding a value `P`, a vproc number. Any memory2 tuples that join to the current tuple must be stored on vproc `P` due to the way memory2 is partitioned. Send the tuple being joined to memory2 to vproc `P`. The join can then be completed locally on vproc `P`.

An additional issue is how to deal with non-equi joins. These are joins for which the join predicate is something other than the `=` operator. For example, the following trigger has a non-equi join condition:

```
create trigger NonEquiJoinExample
from R1, R2
when R1.X >= R2.lowerBound and R1.X < R2.upperBound
do ...
```

For triggers like this, it is not possible to define a partitioning that can be used effectively to speed up join processing. Hence, in cases like this, broadcasts will be used to join a token to a memory node in parallel. Memory nodes will still be horizontally partitioned.

It is a good idea to avoid broadcasts and use a point-to-point messaging scheme instead when possible. This will avoid unnecessary CPU utilization. With point-to-point messaging, a parallel speedup can still be obtained because multiple tokens can be processed simultaneously on different processors.

We have outlined the basic pattern matching strategy for select/join trigger conditions. Fully detailed algorithms for all aspects of join condition testing and temporal condition testing are left for a future paper.

7. Extensibility

TriggerMan is designed to be extensible. This will include support for new data types and operators, in addition to new temporal functions. Support for extended data types such as images, time series, web pages, text, etc. within a database management system has been supported in POSTGRES [Ston90] and Informix Universal Server [Info97] and is being included in other commercial database products. This feature is beginning to be used to add multimedia and object management capability to real-world database applications. The approach taken to support extensibility has been to define a dynamic link library of C functions with a certain format, including (1) constructor and destructor functions, (2) functions for translating an instance of a type from internal to external format, (3) functions for performing operations on instances of a type, (4) functions for estimating the cost of performing certain operations, etc. This library is then registered with the DBMS and dynamically linked when needed. As an example, an extended data type called Document could be created, and triggers could be defined on a stream of documents arriving from an intelligence-gathering source, implementing a form of selective dissemination of information using TriggerMan.

If possible, the approach to handling extended data types in TriggerMan will be to use the standard extensibility format used by Informix, and introduce commands to register new types with the system. If this is done, the same DataBlade modules used by Informix can be used by TriggerMan. Using the existing Informix type extension standard in TriggerMan is preferred to defining a new standard for extended types, since existing extended types that have already been implemented for Informix could be used with TriggerMan. However, we will define our own type extension module format if necessary. As part of the TriggerMan project, issues related to moving large objects between a DBMS and TriggerMan, caching the internal representation of large objects, and evaluating expensive predicates within the TriggerMan server will be investigated.

8. Performance of Initial Prototype

A version of the prototype TriggerMan server, consisting of roughly 20,000 lines of C++, is already operational. It implements single-table triggers, but has no persistent catalogs or ability to directly communicate with a DBMS via a replication server gateway. It supports parallelism using the concept of virtual processors (vprocs), making the code portable to both SMP and shared-nothing parallel computers. Performance tests were run on a dual-processor

```
for {set i 2} {$i <= 2700} {incr i 2} {
  createTrigger t$i in ts1 {
    from EMPLOYEE
    on {insert EMPLOYEE}
    when {EMPLOYEE.salary = [expr 2700 % $i]}
    do {} # no trigger action was run so only
  }      # condition testing would be timed
```

75Mhz Sun SPARCstation 20. Originally, the TriggerMan command language was implemented as an extension of Tcl. We have since implemented a command-language parser so Tcl is not a required component of TriggerMan, and commands have a more natural, SQL-like syntax. At the time the tests were done, the createTrigger command was implemented as an extended Tcl command. The performance tests were done in the following manner. A single EMPLOYEE data source was defined. A total of 1350 triggers were then created using the Tcl program shown inset.

The triggers created fire when the inserted employee has a salary equal to some constant. The [expr 2700 % \$i] expression is evaluated before the trigger is created. The % symbol is the modulo operator. There are 24 triggers that will fire when the inserted salary is zero. Triggers are allocated round-robin to the different vprocs. The tests were run first with one vproc, then with two. With one vproc, only one of the processors is used for rule condition testing. With two, both processors are used. The results are summarized in the following table:

Number of Processors Used (number of vprocs).	Average Condition Testing Time for All Triggers	Average Condition Testing Time Per Trigger
1	13.5 msec	10 μ sec
2	7.5 msec	5.6 μ sec

No selection predicate indexing strategy [Hans90,Hans96b] is currently used. A selection predicate index could dramatically increase performance for this example. This example shows that performance will be quite good for single data source triggers even when their conditions are not or cannot be indexed, as long as the number of triggers is not huge.

The TriggerMan code is now being ported to Windows NT, which will become our primary development platform. Code will be written in a way so that the system will be portable to NT and Solaris. Access to a database for storing TriggerMan's catalog information and other persistent state will be done using the ODBC interface so that TriggerMan will work with multiple different DBMS products.

9. Conclusion

The research outlined here seeks to develop principles that will allow the effective construction of asynchronous, or "outboard" trigger processing systems. A prototype ATP called TriggerMan is being implemented as a vehicle to explore asynchronous trigger processing issues and to validate the design approach introduced here. TriggerMan, or a system like it, could be useful in situations where current trigger systems are not. For example, TriggerMan could trigger on a stream of updates generated by a general application program that were never placed in any DBMS. Moreover, TriggerMan could place a trigger on two different data sources, one from a DBMS, and one from a program, performing an information fusion function. This type of function could be valuable in a number of heterogeneous information systems applications, e.g., in a chemical plant application, a trigger could correlate a stream of reactor vessel temperature and pressure values sent by an application with known dangerous combinations of temperature and pressure kept in a database, firing when it saw a dangerous combination. The main benefit of an ATP system is that it can allow sophisticated, "expensive" triggers (e.g. multiple-table and temporal triggers) to be

defined against a database and processed using the best available algorithms, without adversely impacting on-line update processing. This could greatly expand the benefits of trigger technology in demanding, update-intensive environments.

The results of the TriggerMan project could lead to a new type of system to support applications that need to monitor changes to information – an asynchronous trigger processor. In addition, an architecture for asynchronous trigger processing similar to the one described here could be incorporated directly into a DBMS. This would allow the benefits of asynchronous trigger processing, particularly good update response time *plus* sophisticated trigger processing capability, without the need to incur the cost of moving update descriptors across the boundary from the DBMS into another system. In addition, it would not be necessary to cross back to the DBMS to run trigger actions against database data. In summary, the work outlined here can help develop a new, useful kind of information processing tool, the ATP, and point the way to improvements in the active database capability of existing database management systems. In either case, it will become possible to develop powerful information monitoring applications more easily, and these applications will run with faster performance.

Bibliography

- [Acha92] Acharya, A., M. Tambe, and A. Gupta, "Implementation of Production Systems on Message-Passing Computers," *IEEE Transactions on Knowledge and Data Engineering*, 3(4), July, 1992.
- [Arno96] Arnold, K., J. Gosling, *The Java Programming Language*, Addison Wesley Longman, 1996.
- [Chaw96] Chawathe, S., A. Rajaraman, H. Garcia-Molina, and J. Widom, "Change Detection in Hierarchically Structured Information," *Proc. ACM SIGMOD Conf.*, 1996.
- [Date93] Date, C. J. And Hugh Darwen, *A Guide to the SQL Standard*, 3rd Edition, Addison Wesley, 1993.
- [Forg82] Forgey, C. L., Rete: "A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, vol. 19, pp. 17-37, 1982.
- [Gray93] Gray, Jim, *Transaction Processing, Concepts and Techniques*, Morgan Kaufmann, 1993.
- [Hans90] Hanson, Eric N., M. Chaabouni*, C. Kim and Y. Wang*, "A Predicate Matching Algorithm for Database Rule Systems," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pp. 271-280, Atlantic City, NJ, June 1990.
- [Hans96] Hanson, Eric N., "The Design and Implementation of the Ariel Active Database Rule System," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 1, pp. 157-172, Feb., 1996.
- [Hans96b] Hanson, Eric N. and Theodore Johnson, "Selection Predicate Indexing for Active Databases Using Interval Skip Lists," *Information Systems*, vol. 21, no. 3, pp. 269-298, 1996.
- [Hans95] Hanson, Eric N., S. Bodagala, M. Hasan, G. Kulkarni, J. Rangarajan, *Optimized Rule Condition Testing in Ariel Using Gator Networks*, University of Florida CISE Department TR 95-027, <http://www.cise.ufl.edu>, October 1995.

- [Hans97] Hanson, Eric N. et al., "Flexible and Recoverable Interaction Between Applications and Active Databases," *VLDB Journal*, 1997 (accepted).
- [Hans97b] Hanson, Eric N. and Samir Khosla, "An Introduction to the TriggerMan Asynchronous Trigger Processor," Proceedings of the 1997 Workshop on Rules in Database Systems (RIDS '97), Skovde, Sweden, June, 1997.
- [Info97] "Informix Universal Server," <http://www.informix.com>
- [McCa89] "McCarthy, Dennis R. and Umeshwar Dayal, "The Architecture of an Active Data Base Management System," *Proceedings of the ACM SIGMOD Conference on Management of Data.*, Portland, OR, June, 1989, pp. 215-224.
- [Oust94] Ousterhout, John, *Tcl and the Tk Toolkit*, Addison Wesley, 1994.
- [Sist95] Sistla, Prasad A. and Ouri Wolfson, "Temporal Triggers in Active Databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 3, June, 1995, pp. 471-486.
- [Ston90] Stonebraker, Michael., Larry Rowe and Michael Hirohama, "The Implementation of POSTGRES," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 7, March, 1990, pp. 125-142.
- [Syba96] Sybase Replication Server Technical Overview, Sybase Inc., 1996.
- [Wido96] Widom, J. and S. Ceri, *Active Database Systems*, Morgan Kaufmann, 1996.
- [Witk93] Witkowski, A., F. Carino and P. Kostamaa, "NCR 3700 - The Next-Generation Industrial Database Computer," *Proceedings of the 19th VLDB Conference*, Dublin, Ireland, 1993.

DISTRIBUTION LIST

addresses	number of copies
JOSEPH P. CAVANO AFRL/IFTB 525 BROOKS ROAD ROME, NY 13441-4505 5	10
ERIC HANSON UNIVERSITY OF FLORIDA CISE DEPARTMENT GAINESVILLE, FL 32611-6120	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
RELIABILITY ANALYSIS CENTER 201 MILL ST. ROME NY 13440-8200	1
ATTN: GWEN NGUYEN GIDEP P.O. BOX 8000 CORONA CA 91718-8000	1

AFIT ACADEMIC LIBRARY/LDEE
2950 P STREET
AREA B, BLDG 642
WRIGHT-PATTERSON AFB OH 45433-7765

1

ATTN: GILBERT G. KUPERMAN
AL/CFHI, BLDG. 248
2255 H STREET
WRIGHT-PATTERSON AFB OH 45433-7022

1

ATTN: TECHNICAL DOCUMENTS CENTER
DL AL HSC/HRG
2698 G STREET
WRIGHT-PATTERSON AFB OH 45433-7604

1

AIR UNIVERSITY LIBRARY (AUL/LSAD)
600 CHENNAULT CIRCLE
MAXWELL AFB AL 36112-6424

1

US ARMY SSDC
P.O. BOX 1500
ATTN: CSSD-IM-PA
HUNTSVILLE AL 35807-3801

1

TECHNICAL LIBRARY D0274(PL-TS)
SPAWARSYSCEN
53560 HULL STREET
SAN DIEGO CA 92152-5001

1

NAVAL AIR WARFARE CENTER
WEAPONS DIVISION
CODE 48L000D
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100

1

SPACE & NAVAL WARFARE SYSTEMS CMD
ATTN: PMW163-1 (R. SKIANO)RM 1044A
53560 HULL ST.
SAN DIEGO, CA 92152-5002

2

SPACE & NAVAL WARFARE SYSTEMS 1
COMMAND, EXECUTIVE DIRECTOR (PD13A)
ATTN: MR. CARL ANDRIANI
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200

COMMANDER, SPACE & NAVAL WARFARE 1
SYSTEMS COMMAND (CODE 32)
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200

CDR, US ARMY MISSILE COMMAND 2
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSMI-RD-CS-R, DOCS
REDSTONE ARSENAL AL 35898-5241

ADVISORY GROUP ON ELECTRON DEVICES 1
SUITE 500
1745 JEFFERSON DAVIS HIGHWAY
ARLINGTON VA 22202

REPORT COLLECTION, CIC-14 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

AEDC LIBRARY 1
TECHNICAL REPORTS FILE
100 KINDEL DRIVE, SUITE C211
ARNOLD AFB TN 37389-3211

COMMANDER 1
USAISC
ASHC-IMD-L, BLDG 61801
FT HUACHUCA AZ 85613-5000

US DEPT OF TRANSPORTATION LIBRARY 1
FB10A, M-457, RM 930
800 INDEPENDENCE AVE, SW
WASH DC 22591

AWS TECHNICAL LIBRARY 1
859 BUCHANAN STREET, RM. 427
SCOTT AFB IL 62225-5118

AFIWC/MSY
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

1

SOFTWARE ENGINEERING INSTITUTE
CARNEGIE MELLON UNIVERSITY
4500 FIFTH AVENUE
PITTSBURGH PA 15213

1

NSA/CSS
K1
FT MEADE MD 20755-6000

1

ATTN: DM CHAUHAN
DCMC WICHITA
271 WEST THIRD STREET NORTH
SUITE 6000
WICHITA KS 67202-1212

1

AFRL/VSOS-TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004

1

ATTN: EILEEN LADUKE/D460
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

1

OUSD(P)/DTSA/DUTD
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

2

SOFTWARE ENGR'G INST TECH LIBRARY
ATTN: MR DENNIS SMITH
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213-3890

1

USC-ISI
ATTN: DR ROBERT M. BALZER
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292-6695

1

KESTREL INSTITUTE
ATTN: DR CORDELL GREEN
1801 PAGE MILL ROAD
PALO ALTO CA 94304

1

ROCHESTER INSTITUTE OF TECHNOLOGY
ATTN: PROF J. A. LASKY
1 LOMB MEMORIAL DRIVE
P.O. BOX 9887
ROCHESTER NY 14613-5700

1

WESTINGHOUSE ELECTRONICS CORP
ATTN: MR DENNIS BIELAK
ELECTRONICS SYSTEMS GROUP
P.O. BOX 746, MAIL STOP 432
BALTIMORE MD 21203

1

AFIT/ENG
ATTN:TOM HARTRUM
WPAFB OH 45433-6583

1

THE MITRE CORPORATION
ATTN: MR EDWARD H. BENSLEY
BURLINGTON RD/MAIL STOP A350
BEDFORD MA 01730

1

UNIV OF ILLINOIS, URBANA-CHAMPAIGN
ATTN: DR MEHDI HARANDI
DEPT OF COMPUTER SCIENCES
1304 W. SPRINGFIELD/240 DIGITAL LAB
URBANA IL 61801

1

HONEYWELL, INC.
ATTN: MR BERT HARRIS
FEDERAL SYSTEMS
7900 WESTPARK DRIVE
MCLEAN VA 22102

1

SOFTWARE ENGINEERING INSTITUTE
ATTN: MR WILLIAM E. HEFLEY
CARNEGIE-MELLON UNIVERSITY
SEI 2218
PITTSBURGH PA 15213-38990

1

UNIVERSITY OF SOUTHERN CALIFORNIA
ATTN: DR W. LEWIS JOHNSON
INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY/SUITE 1001
MARINA DEL REY CA 90292-6695

1

COLUMBIA UNIV/DEPT COMPUTER SCIENCE
ATTN: DR GAIL E. KAISER
450 COMPUTER SCIENCE BLDG
500 WEST 120TH STREET
NEW YORK NY 10027

1

SOFTWARE PRODUCTIVITY CONSORTIUM
ATTN: MR ROBERT LAI
2214 ROCK HILL ROAD
HERNDON VA 22070

1

AFIT/ENG
ATTN: DR GARY B. LAMONT
SCHOOL OF ENGINEERING
DEPT ELECTRICAL & COMPUTER ENGRG
WPAFB OH 45433-6583

1

NSA/DFC OF RESEARCH
ATTN: MS MARY ANNE OVERMAN
9800 SAVAGE ROAD
FT GEORGE G. MEADE MD 20755-6000

1

AT&T BELL LABORATORIES
ATTN: MR PETER G. SELFRIDGE
ROOM 3C-441
600 MOUNTAIN AVE
MURRAY HILL NJ 07974

1

ODYSSEY RESEARCH ASSOCIATES, INC.
ATTN: MS MAUREEN STILLMAN
301A HARRIS B. DATES DRIVE
ITHACA NY 14850-1313

1

TEXAS INSTRUMENTS INCORPORATED
ATTN: DR DAVID L. WELLS
P.O. BOX 655474, MS 238
DALLAS TX 75265

1

TEXAS A & M UNIVERSITY
ATTN: DR PAULA MAYER
KNOWLEDGE BASED SYSTEMS LABORATORY
DEPT OF INDUSTRIAL ENGINEERING
COLLEGE STATION TX 77843

1

KESTREL DEVELOPMENT CORPORATION
ATTN: DR RICHARD JULLIG
3260 HILLVIEW AVENUE
PALO ALTO CA 94304

1

DARPA/ITO
ATTN: DR KIRSTIE BELLMAN
3701 N FAIRFAX DRIVE
ARLINGTON VA 22203-1714

1

NASA/JOHNSON SPACE CENTER
ATTN: CHRIS CULBERT
MAIL CODE PT4
HOUSTON TX 77058

1

SAIC
ATTN: LANCE MILLER
MS T1-6-3
PO BOX 1303 (OR 1710 GOODRIDGE DR)
MCLEAN VA 22102

1

STERLING IMD INC.
KSC OPERATIONS
ATTN: MARK MAGINN
BEECHES TECHNICAL CAMPUS/RT 26 N.
ROME NY 13440

1

NAVAL POSTGRADUATE SCHOOL
ATTN: BALA RAMESH
CODE AS/RS
ADMINISTRATIVE SCIENCES DEPT
MONTEREY CA 93943

1

HUGHES AIRCRAFT COMPANY
ATTN: GERRY BARKSDALE
P. O. BOX 3310
BLDG 618 MS E215
FULLERTON CA 92634

1

SCHLUMBERGER LABORATORY FOR
COMPUTER SCIENCE
ATTN: DR. GUILLERMO ARANGO
8311 NORTH FM620
AUSTIN, TX 78720

1

MOTOROLA, INC.
ATTN: MR. ARNOLD PITTLER
3701 ALGONQUIN ROAD, SUTE 601
ROLLING MEADOWS, IL 60008

1

DECISION SYSTEMS DEPARTMENT
ATTN: PROF WALT SCACCHI
SCHOOL OF BUSINESS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CA 90089-1421

1

SOUTHWEST RESEARCH INSTITUTE
ATTN: BRUCE REYNOLDS
6220 CULEBRA ROAD
SAN ANTONIO, TX 78228-0510

1

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
ATTN: CHRIS DABROWSKI
ROOM A266, BLDG 225
GAITHSBURG MD 20899

1

EXPERT SYSTEMS LABORATORY
ATTN: STEVEN H. SCHWARTZ
NYNEX SCIENCE & TECHNOLOGY
500 WESTCHESTER AVENUE
WHITE PLAINS NY 20604

1

NAVAL TRAINING SYSTEMS CENTER
ATTN: ROBERT BREAU/CODE 252
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3224

1

CENTER FOR EXCELLENCE IN COMPUTER-
AIDED SYSTEMS ENGINEERING
ATTN: PERRY ALEXANDER
2291 IRVING HILL ROAD
LAWRENCE KS 66049

1

DR JOHN SALASIN
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

1

DR BARRY BOEHM
DIR, USC CENTER FOR SW ENGINEERING
COMPUTER SCIENCE DEPT
UNIV OF SOUTHERN CALIFORNIA
LOS ANGELES CA 90089-0781

1

DR STEVE CROSS
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891

1

DR MARK MAYBURY
MITRE CORPORATION
ADVANCED INFO SYS TECH; G041
BURLINGTON ROAD, M/S K-329
BEDFORD MA 01730

1

MR SCOTT FOUSE
ISX
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE C 91361

1

MR GARY EDWARDS
ISX
433 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

1

DR ED WALKER
BBN SYSTEMS & TECH CORPORATION
10 MOULTON STREET
CAMBRIDGE MA 02238

1

LEE ERMAN
CIMFLEX TEKNOLEDGE
1810 EMBACADERO ROAD
P.O. BOX 10119
PALO ALTO CA 94303

1

DR. DAVE GUNNING
DARPA/ISD
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

1