The Application Of Scripts To Deadlock Avoidance.

C N BLEWETT[*] and G J ERWIN[#]

* *Dept. of Accounting and Finance (Business Information Systems Section), University of Natal, King George V Ave., Durban 4001, South Africa.*
E-mail: BLEWETT@BIS.UND.AC.ZA
Tel 031-2602161,
Fax 031-2603292,

\# *Dept. of Accountancy (Business Information Systems Section), University of Durban-Westville, Private Bag X54001, Durban 4000, South Africa.*
E-mail: ERWIN@PIXIE.UDW.AC.ZA
Tel 031-8202444,
Fax: 031-8202429.

Abstract : We describe the prototype of an expert system software advisor for the lock manager of a database system. The software advisor, called EAGLE (**E**xpert **A**dvisor for **G**ranting **L**ocks **E**ffectively), is intended to become an embedded expert system within a database management system. EAGLE maintains a record of lock requests and lock status within a database management system as an application processes transactions.Eag uses this dynamic lock data to avoid the granting of locks which could lead to a future deadlock. The sequence of lock requests and lock grantings is held as a script(s). EAGLE uses its collected record of lock request sequences to matchagainst sterotypical lock event sequences (script base) and to learn to avooid such sequences in future. As EAGLE gains expereince of lock event sequences leading to deadlock, it recognises patterns which have led to deadlock, and avoids granting locks which would repeat a previous deadlock-inducing sequence of lockss, thereby reducing the occurrence of deadlock. EAGLE treats the deadlock problem as a plan recognition issue rather than a problem resolution issue. We describe the general design of EAGLE, present some results from the EAGLE prototype implementation and discuss planned enhancements to EAGLE.

Keywords: database management system, deadlock, locking, expert system, scripts, learning, plan recognition

Computing Review Category: Expert system, database, knowledge representation.

Introduction

Deadlock is the combination of five conditions within a database system [8], namely,

•	lockout.	A process (transaction) obtains exclusive control of a resource.
•	concurrency.	Transactions compete for exclusive control of multiple resources.
•	completion.	A transaction is allowed to hold resources until all processing for that transaction is completed.
•	additional request.	A transaction can request additional resources while holding exclusive control of other resources.
•	circular wait.	A circular chain (cycle) of two or more transactions exists such that each transaction holds a resource requested by the next transaction in the chain.

Each transaction forming this deadlock state awaits the release of exclusive lock(s) on resources held by another transaction. Each transaction prevents at least one other transaction from proceeding further because it holds an exclusive lock(s) on a resource(s) required by another transaction. Each transaction blocks another and each transaction is blocked by another. Typically, there is a cycle of transactions waiting for resources held by another transaction, and, eventually, a transaction waiting for resources held by the first transaction. Deadlock is usually detected by noticing that there is a cycle of transactions awaiting resources.

The deadlock state and its treatment are well-described. See, for example, [1], [4], [6] and [8].

As a database management system processes transactions, locks are requested, set and released on resources. Each lock setting action is preceded by a lock request from the application through the database management system's lock manager (LOMAN). LOMAN, after inspection of current locks on resources, eventually grants the lock request or refuses it.

So, a LOMAN deals with the following locking-related tasks:

- request for a lock on a specific resource
- inspection of current locks
- setting (granting) of a lock
- refusal of a lock request (this may be after some waiting period)
- notification of the success/failure of the lock request to the application.
- release of a lock (unlock).

A typical sequence of locking activities could be:

- request a lock on resource *n*
- inspect resource *n* to see if it is able to be locked
- grant or refuse the lock request
- record the result of the lock request

This sequence of lock activities within a database management system is a locking event sequence. In a typical application, the lock activities (request lock/unlock on a specific resource) in a locking event sequence happen in a consistent sequence. For example, if a database application is updating a database of banking accounts with deposits and withdrawals, then every transaction requests locks and releases locks in the same sequence. The content of locking event sequences for each specific application is predictable; a stereotype. Many multi-user applications process more than one type of application concurrently, often using the same database management system. So, the neat stereotypical content of locking event sequences is not guaranteed when a mixture of applications and mixture of types of transactions (add, delete, alter, inspect) is in progress.

We classify deadlock treatments as:

Ignore deadlock

Impose some external system interference on the assumed deadlock situation to resolve the delay due to the assumed deadlock. For example, after 10 seconds with no further activity in a transaction, interrupt the

transaction and restart it. A useful analogy is a person travelling through a malaria-infested area. No periodic visits are paid to the doctor. Even should the traveller exhibit the symptoms of malaria, no attempt is made to visit a doctor. Finally when the traveller can no longer proceed because of his illness, he purchases some medication from the pharmacy in an attempt to rectify the problem.

Deadlock Detection and Resolution

Allow transactions to proceed, but continually check the lock status of all resources for all current transactions for deadlock. Once a deadlock is detected, resolve it by "choosing a victim" for restart, hoping (planning) that the deadlock state will now disappear. Continuing the analogy of the traveller, (s)he would travel un-perturbed through the danger areas. However, (s)he would periodically visit the doctor to check for malaria infection. If malaria is detected the doctor would prescribe appropriate medication.

Deadlock Prevention

Design the interaction of concurrent transactions so that it is impossible for deadlock to occur. Usually this is achieved by severely reducing the interleaving (concurrency) of transactions. The (analogical) traveller, with extreme caution, would carefully plan his route so as to ensure that (s)he never enters any area where malaria could possibly exist. No possibility of malaria infection.

Deadlock Avoidance

Allow transactions to proceed, but continually evaluate the potential of deadlock occurring, before deciding on whether to grant a lock request. Not as safe as the preventative (analogical) traveller, the avoidance traveller would continue to travel the roads until (s)he arrived at an area where (s)he anticipated that it was likely that (s)he could contract malaria. At this point (s)he would decide to find an alternative route where malaria would be less likely.

There are two major ways in which deadlock avoidance is approached:

> *A Priori Deadlock Avoidance*
> Belik [2] outlines an *a priori* approach to deadlock avoidance. In this approach, avoidance of potential deadlocks takes place, based on pre-stored knowledge of the content and order of lock-requests. In this case our (analogical) traveller would highlight those areas on the road-map where malaria could be contracted. While haphazardly driving around the country, he would continually check his map to ensure (s)he was not approaching a malaria area. If (s)he found that (s)he was approaching an infected area, (s)he would choose another route.
>
> *Dynamic Deadlock Avoidance*
> Our approach to deadlock avoidance does not require pre-stored knowledge of lock-request sequences. Avoidance of potential deadlocks takes place, based on experience accumulated from observations of previous similar lock event sequences. The dynamic deadlock avoidance (analogical) traveller would set out on a journey with no details of dangerous malaria areas, besides some simple guidelines that marshy areas could have malaria, and so on. While haphazardly traversing the country (s)he may decide to avoid an area which fits such a generic description. However, as (s)he unwittingly enters danger areas, and contracts malaria, (s)he learns to identify and avoid these dangerous areas in future.

In this paper we describe our new dynamic deadlock avoidance approach. We treat the deadlock problem as a *plan recognition* issue, see [3], rather than as a *problem resolution* issue. Our dynamic avoidance approach attempts to identify whether the "plan" of current lock event sequences is to reach the goal of deadlock, and, when so recognised, avoid deadlock prior to its occurring.

Script-based knowledge representation

A script is a knowledge representation scheme for representing time-based events.

"A script is a finite set of events of some duration and importance oriented towards a goal." [12], Schank and Abelson [13], state "... a script, ... is a stereotypical representation of a sequence of actions oriented toward attaining some goal."

A script-based knowledge representation in an expert system stores data about sequences of actions (events) which have the following characteristics:

- duration — a beginning, and an end,
- dependency — certain events must occur before/after others, and
- stereotypical aspects — the event sequences are often predictable, well-understood, and commonly used.

Locking event sequences within a database management system exhibit, *inter alia*, these same characteristics. Each locking event (activity) has starting and finishing times; each event (except the first and last events) has a predecessor(s) and a successor(s); and, the combination of events in sequence usually forms a well-ordered, stereotypical and understood series of events. The "goal" in our context is the attainment of deadlock.

Scripts were initially formulated, see [13], [14] and [15], to provide structures and manipulative ability for the handling of time and dependency-based events in the domain of "understanding" natural language, a particular field of artificial intelligence. This work defined primitive operations within natural language, and developed a formal methodology for representing the semantic content of sentences and stories.

The script concept is suitable for describing, storing, and searching for sequences of actions with attendant states. Script structures and manipulations have been applied to a variety of contexts (see, for example, [3], [5], [7], [9], [10] and [11]), and are appropriate for the representation of event sequences within the context of sequences of lock activities within a database management system.

Script-based knowledge representation and lock requests

The script-based knowledge representation is a specific application of the frame knowledge representation. Each frame's slot definitions are assigned for the specific application under consideration [16].

Thus, each frame in our dynamic avoidance system contains slots for:

Script Name
Process Roles
Resource Props
Expected Event
Critical Event
Critical Event Response
Sequence Description
Response
Activation Level
Utilisation Level
Minimum Activation Level
Maximum Activation Level

The script name uniquely identifies each script while the process roles and resource props identify the number of transactions and resources involved in the deadlock. The expected event is used to monitor the next expected event for this script. Any objections must take place before the critical event, after which objections to deadlock are ineffective. The sequence description describes the stereotypical event sequence and the response indicates the result of a complete match against this sequence description. The various activation and utilisation levels are

used to dynamically monitor and control the activation of the script.

EAGLE IMPLEMENTATIONS

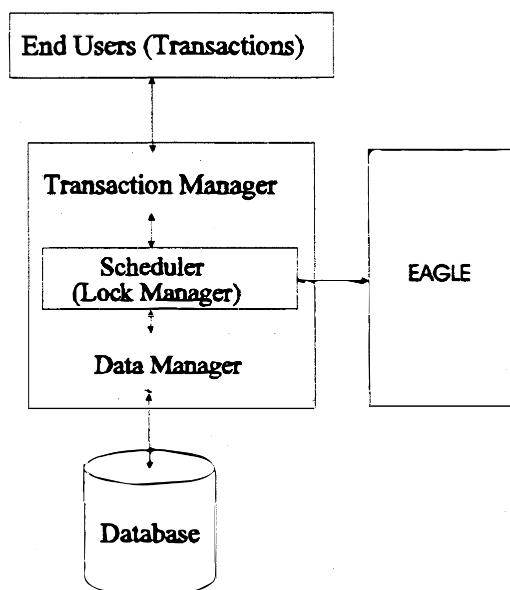
We developed several EAGLE prototypes. Various prototypes included an algorithmic implementation (version 1), a rule-based implementation (version 2) and a script-based implementation (version 3). EAGLE Version 3 (v3) is described in this paper.

The EAGLE v3 prototype is still under evaluation. The EAGLE v3 system was developed in CA-Clipper and Common LISP. CA-Clipper was used for simulation of database activities, and Common LISP was used for the MATCHER (see below).

EAGLE v3

EAGLE v3 is the script-based implementation of our dynamic deadlock avoidance scheme. Incoming lock requests (lock event sequences) are passed from LOMAN to EAGLE where they are matched against stored stereotypical event sequences, representing various patterns of lock events which lead to deadlock situations. Should an event (lock or unlock request) in an event sequence cause a match to occur with a stored stereotype, the lock or unlock request is rejected. If no match occurs the lock event is approved.

The following diagram depicts an overview of the EAGLE operating system environment -



The EAGLE system interacts, as an external advisor, with the lock manager of the DBMS. In the simulated environment, created for testing the system, the tasks of the Transaction Manager and Data Manager are not required. However, the simulated operation of users and the lock management operation of the scheduler are required and are therefore included in the simulated environment.

LOMAN receives the transactions from the transaction module. LOMAN checks the lock table (if a lock request is received) to determine if the requested resource is available, *i.e.* the resource is not currently locked by another transaction. If the lock request is approved by LOMAN, the lock request details (transaction and resource(s) involved) are passed to EAGLE for further approval. EAGLE attempts to match the event sequence against stereotypical event sequences, to determine the probability of deadlock occurring, based on previous experience.

EAGLE's decision to approve or reject a lock request, based on a computed probability of deadlock occurring, is returned to LOMAN. If EAGLE approves the granting of the lock, then LOMAN grants the lock to the transaction. If EAGLE denies the lock request, then LOMAN denies the lock request. If denied, the transaction is forced to re-request the lock, at which time if LOMAN approves the request, EAGLE will re-evaluate the lock request. The primary task of the EAGLE system is to advise LOMAN on the granting of lock requests.

The precondition header instantiates a script by reference to the main precondition of the script [13]. The precondition in the deadlock context, is the existence of concurrent processes. This condition is necessary for the possibility of deadlock to exist at all.

The instrumental header occurs when reference is made to two or more contexts, where at least one may be considered instrumental to the others [13]. In the deadlock context, an instrumental header is the existence of a competitive lock request by a transaction, on an already locked resource. This would serve as a stronger indication of deadlock than the precondition header.

The activation system is also responsible for activating the Learner when deadlocks occur, to record new scripted sequences.

The following figure depicts the appearance of LOMAN and the Rule Component as they interact. LOMAN monitors the incoming lock requests as shown in the EVENT column. Should a deadlock occur, LOMAN will detect (D) and resolve (R) it. The Rule Component would then activate the Learner to generate an appropriate script. The Rule Component of EAGLE monitors the event sequences and activates the MATCHER to determine whether any matches have occurred. The second column indicates the number of scripts which are instantiated where the computed similarity metric is below the activation level (see Matcher/Analuzer). The activation level is the value the computed similarity metric must equal or exceed in order to activate the script. The third column shows the number of scripts which do match the current event sequence but where the critical event (see Matcher/Analyzer) has past, making objection to a lock request futile. In this situation nothing can be done to avoid deadlock, but deadlock may not occur. The fourth column displays, when objecting, the name of the script causing the objection. The final column shows EAGLE's response to the current event.

Lock Manager/Rule Component

(1/0) EAGLE STATUS REPORT [2/10] - EAGLE ON - RDM ON B/No. 2

EVENT	Less than AL (S/G)	Past CE (S/G)	Object SFS	OBJECT
T02-R01	2/0	0/0	-	N
T02-R02	2/0	0/0	-	N √
T01*R02	2/0	0/0	-	N
T01*R01	2/0	0/0	-	N
T03+R01	0/0	0/0	-	N
T04+R02	0/0	0/0	-	N
T01-R02	2/0	0/0	-	N
T04*R02	2/0	0/0	-	N
T01-R01	2/0	0/0	-	N √
T03*R01	2/0	0/0	-	N
T04+R01	0/0	0/0	-	N
T03+R02	0/0	0/0	-	N(D)
T04-R02	0/0	0/0	-	N(R)
T03*R02	0/0	0/0	-	N

The Rule component is therefore primarily responsible for interfacing with LOMAN and activating the Matcher/Analyzer and Learner, when necessary.

ii. Script Base Component

The seccond component of EAGLE is the script base. It stores the stereotypical event sequences which have been learnt through previous experience (see Learner component).

A script to represent the sequence progression of three processes towards deadlock, would contain the following;

Script Name	S_P3R3_0
Process Roles	?PA?PB?PC
Resource Props	?RA?RB?RC
Expected Event	(LOCK ?PA ?RA)
Critical Event	(LOCK ?PC ?RC)
Critical Event Response	(DENY LOCK)
Sequence Description	((LOCK ?PA?RA)(LOCK ?PB?RB)(WAIT ?PB?RA) (LOCK ?PC?RC)(WAIT ?PC?RB)(WAIT ?PA?RC))
Response	(DEADLOCK=TRUE)
Activation Level	0.500
Utilisation Level	0
Minimum Activation Level	0.500
Maximum Activation Level	0.660

Each script is identified by its name, with other slots containing information relevant to the activation and instantiation of the script.

Two types of scripts are stored in the script base, namely, *specific* scripts and *generic* scripts. *A Specific Script* describes event sequences in the context of the current operational environment, *i.e.* a specific script is not applicable in another context. A specific script records all lock, unlock, and wait events of both deadlock participants and non-deadlock participants. A specific script is therefore specific to a certain context. For example, specific scripts generated for deadlocks occurring between 3 transactions in a 10 transaction environment would be different to those generated for deadlocks occurring between 3 transactions in a 5 transaction environment.

The following is an example of a specific script;

S_P2R2_3
?PC?PA
?RB?RA
(LOCK ?PA ?RA)
(LOCK ?PC ?RB)
(DENY LOCK)
((LOCK ?PA ?RA)(WAIT ?PB ?RA)(LOCK ?PC ?RB)(WAIT ?PC ?RA)(WAIT ?PA ?RB))
(DEADLOCK = TRUE)
0.530
0
0.600
0.600

Note that (WAIT ?PB ?RA) is a non-participating event in the deadlock sequence, but provides the necessary contextual detail to instantiate this script.

A Generic Script describes deadlock in a domain-independent context. No details of non-participating transactions' events or unlock requests are recorded in the generic script, i.e. they describe "classic" deadlock situations between 3 or more participating processes. Generic scripts do not describe 2 process deadlocks for the reasons outlined above. Generic scripts are useful in situations where specific scripts have not yet been defined. The following is an example of a generic script representation of a classic 3 process deadlock situation;

G_P3R3_0
?PA?PB?PC
?RA?RB?RC
(LOCK ?PA ?RA)
(LOCK ?PC ?RC)
(DENY LOCK)
((LOCK ?PA ?RA)(LOCK ?PB ?RB)(WAIT ?PA ?RB)(LOCK ?PC ?RC)(WAIT ?PB ?RC)
(WAIT ?PC ?RA))
(DEADLOCK = TRUE)
0.530
0
0.5
0.67

The script base is the foundation of the EAGLE system. The next section describes the Matcher/Analyzer component, which using both specific and generic scripts from the script base, attempts to identify potential deadlock situations.

(iii) Matcher/Analyzer Component

The third component of EAGLE is the Matcher/Analyzer (M/A). On being activated by the Rule Component, and receiving the current event sequence, the M/A attempts to match the event sequence against stored scripts.

The M/A is activated when the activation conditions, as determined by the rule component, have been met. The M/A receives the current event sequence from the Communication Interface of the rule component. The Matcher's primary task is to match the current event sequence against scripts stored in the script base. The Analyzers task is to decide whether to object to the current event.

Matcher

Matching of event sequences is performed against both the specific and generic script bases. Before instantiation is attempted, the scope of the match is reduced by checking the Process Roles and Resource Props of the scripts. For example, a script with 3 process roles, ?PA?PB?PC would not be instantiated with an event sequence which only had two participating transactions.

After identifying the candidate scripts from the script base, the matcher attempts to instantiate the scripts with the current event sequence. Consider the following event sequence;

(T01*R01)(T02*R02)(T02+R01)(T03*R03)(T03+R02)(T01+R03)

Each event (clause) contains constants, such as T01, R01 etc. The sequence description contains events, e.g.

((LOCK ?PA ?RA)(LOCK ?PB ?RB)(WAIT ?PB ?RA)(LOCK ?PC ?RC)
(WAIT ?PC ?RB)(WAIT ?PA ?RC))

that consist of pattern variables that need to be bound to the constants during instantiation. The task of matching, is to find every event that matches the pattern contained in the script.

In this multiple-clause pattern, matching is performed against all clauses in the pattern simultaneously, i.e. the same variables are bound to the same constants in each instantiation. So, all bindings after each match (one pattern clause against one event sequence clause) have to be merged to find consistent bindings. All consistent bindings are stored and returned as the result.

The instantiated script (which includes the expected event and the critical event) is then passed to the Analyzer.

Analyzer

The Analyzer is similar to the plan analyzer of Benoit [3]. The Analyzer seeks to examine the matched scripts in order to identify progression (plans) towards deadlock. The Analyzer attempts to recognise the plan of an event sequence by computing a similarity metric (SM) for each instantiated script. This is done by computing the number of bindings made, as a percentage of the total number of variables in the script, e.g.

The computed SM for the event sequence

(T01*R01)(T02*R02)(T02+R01)(T03*R03)

producing the instantiated script

((LOCK T01 R01)(LOCK T02 R02)(WAIT T02 R01)(LOCK T03 R03)
(WAIT T03 R02)(WAIT T01 R03))

would be

8/12 = 0.66

The SM is then compared to the activation level (AL) of the script. The AL is initially set at 0.500. This is adjusted, depending on script utilisation levels (see below). If the SM is greater or equal to the AL, and the current event is not past the critical event, then the CE response (deny lock) i.e. OBJECT, is returned to the Rule Component. The CE is that event that once it has occurred makes avoidance of deadlock impossible. In the sequence;

((LOCK ?PA ?RA)(LOCK ?PB ?RB)(WAIT ?PB ?RA)(LOCK ?PC ?RC)
(WAIT ?PC ?RB)(WAIT ?PA ?RC))

the CE is (LOCK ?PC ?RC). After this point, no more locks are placed, and so no action can be taken to avoid deadlock. Therefore a request by event (T03*R03) instantiated as (LOCK T03 R03), would be refused because the SM (0.66) is greater than the AL (0.50), and the CE is not yet past.

Should a match occur, and the Analyzer idenify an avoidable deadlock, the Matcher/Analyzer will return an OBJECT to the rule component, and the transaction request will not be granted. The followin figure shows the Matcher matching the event sequence,

(T01*R01)(T02*R02)(T02+R01)(T03*R03)

against the script;

((LOCK ?PA ?RA)(LOCK ?PB ?RB)(WAIT ?PB ?RA)(LOCK ?PC ?RC)
(WAIT ?PC ?RB)(WAIT ?PA ?RC)),

and then objecting because the SM exceeds the AL, and the CE (LOCK ?PC?RC) is not yet past.

Matcher/Analyzer

The following figure shows the processing taking place in the Lock Manager/Rule Component modules. The fourth event (T03*R03) as requested by LOMAN is objected to by the Matcher/Analyzer based on script S_P3R3_0. The result is that processing of the three transactions completes without any deadlocks occurring.

Analyzer

(2/0) EAGLE STATUS REPORT [1/1] - EAGLE ON - RDM ON B/No. 8

EVENT	Less than AL (S/G)	Past CE (S/G)	Object SES	OBJECT
T01*R01	0/0	0/0	-	N
T02*R02	0/0	0/0	-	N
T02+R01	0/0	0/0	-	N
T03*R03	0/0	0/0	S_P2R2_0	Y(A)
T01*R03	1/0	0/0	-	N
T01-R01	1/0	0/0	-	N
T02*R01	1/0	0/0	-	N
T02-R02	1/0	0/0	-	N
T01-R03	1/0	0/0	-	N √
T03*R03	1/0	0/0	-	N
T02-R01	1/0	0/0	-	N √
T03*R02	1/0	0/0	-	N
T03-R03	1/0	0/0	-	N
T03-R02	1/0	0/0	-	N √

Lock Manager/Rule Component

iv Learner Component

The fourth component of the EAGLE system is the Learner component. The Learner component is activated by the Rule Component whenever LOMAN's deadlock detector detects a deadlock. The details of the event sequence leading to the deadlock are passed to the Learner for analysis.

The Learner has 2 sub-components, viz. SESREC and GSESREC.

a. SESREC is responsible for recording specific scripted sequences. On receiving a deadlock sequence, SESREC converts the event sequence into the stereotypical sequence description format, e.g. T01*R01 becomes (LOCK ?PA?RA). All events (locks, unlocks and waits) are recorded from the first successfully placed lock[1], of a transaction involved in the deadlock, up until the deadlock is detected. After checking to see that the sequence is not already stored in the script base, SESREC then identifies the details of the remaining slots of the script, i.e. Script Name, Process Roles, Resource Props, Expected Event, CE, CE Response, Activation Response, AL, UL, Minimum AL and Maximum AL. The script is then recorded in the script base.

b. GSESREC is responsible for recording generic scripted sequences. GSESREC follows the same procedure as SESREC, however it only includes deadlock participating transactions, and LOCK and

[1] An unsuccessfully placed lock is one which is released later on as a result of a rollback.

UNLOCK requests in the sequence description. Should the sequence description represent a 2 transaction/2 resource deadlock or an up-front locking deadlock (see script base section) the script will not be recorded.

The four components of the EAGLE system interact in an attempt to identify potential deadlocks based on stored previous expereince. Once identified, EAGLE objects to events which are recognised as participating in a plan which could result in deadlock. However, should a deadlock occur, the experience gained from the deadlock is stored in the script base in an attempt to avoid future occurrences of the same deadlock.

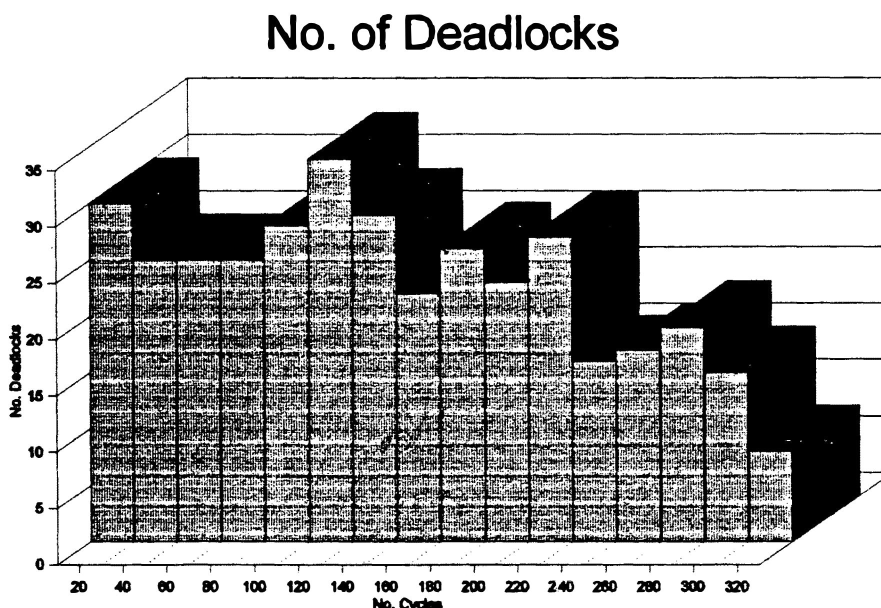
TEST RUNS

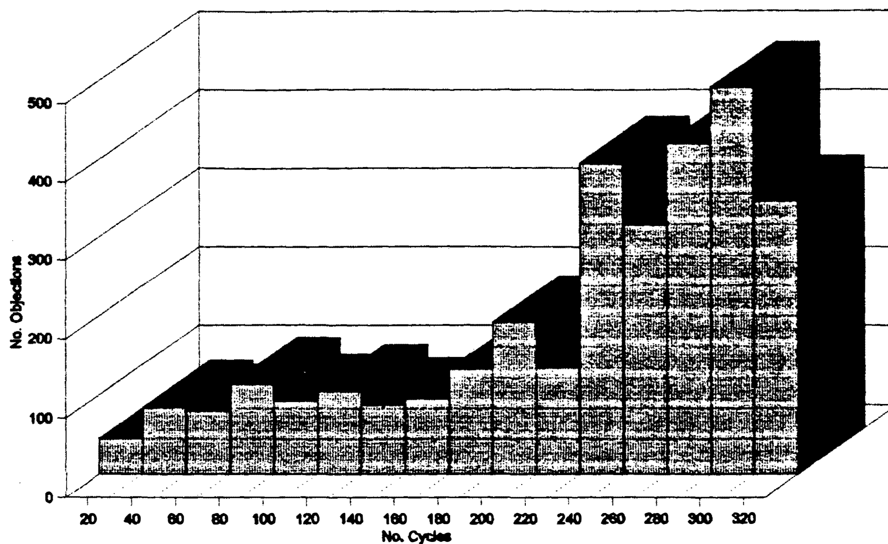
Various test runs have been performed with the EAGLE system. An event sequence generator generates transactions within specified parameters. This allows for either potentially high occurrence of deadlock situations or low occurrence of deadlock situations to be tested. Furthermore, deadlocks can occur between two transactions and two resources, or, in more complex situations, between *n* transactions and *m* resources, where *n* and *m* can assume the values 2 to infinity. We show below some results from one of our test runs, with three transactions and two resources:

Transaction 1	Transaction 2	Transaction 3
T01*R01	T02*R02	T03*R02
T01*R02	T02*R01	T03*R01
T01-R01	T02-R02	T03-R01
T01-R02	T02-R01	T03-R02

Three transactions (T01, T02, T03) are competing for two resources (R01, R02). Two-phase locking is used by all transactions. Each transaction requires a lock (*) on the resource before processing can proceed, and then unlocks (-) the resource when processing is completed.

These three transactions were processed, in a random order, 320 times. The original script base was empty, but over time was populated, as deadlocks were observed and recorded in the script base. As a result, the number of objections to lock requests from EAGLE increases while the number of deadlocks decreases. The following two graphs summarise the data produced.





These results indicate, in the above situation, how EAGLE is able to reduce the number of deadlocks occurring at the expense of an increased number of objections.

Conclusion

Script-based knowledge representation has proved to be appropriate for treatment of potential deadlock situations based on learning from previous experience.

The authors are continuing to develop EAGLE and produce empirical data from a large variety of simulated runs. Future work will look at the overhead of EAGLE inside a database management system, other forms of pattern recognition, such as neural networks, and evaluation of conditions in which EAGLE reduces deadlock occurrences with the attendant reductions in usage of system resources. As with many schemes which seek to prevent or avoid deadlock, EAGLE can reduce the amount of interleaving of concurrent transactions. The parameterisation of many aspects of EAGLE will allow future work to explore the tradeoff between interleaving and deadlock treatment.

References

1. Agrawal R, Carey M J and McVoy L W, [1987], The Performance of Alternative Strategies for Dealing with Deadlock in Database Management Systems, *IEEE Transactions on Software Engineering*, Vol. SE-13, **12**, 1348-1363.
2. Belik, F. (1990) An Efficient Deadlock Avoidance Technique. IEEE Transactions on Computers, Vol. 39, No. 7, July, 1990, pp. 882-888.
3. Benoit, J.W., Davidson, J.R., Hofman, E.J., Laskowski, S.J. and Leighton, R.R. (1987) Integrating plans and scripts : an expert system for plan recognition, in *Proceedings of the Third Annual Expert Systems in Government Conference*, IEEE Computer Society Press, Washington, pp. 201-207.
4. Bernstein, P.A., Hadzilacos, V. and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*, Addison-Wesley Publishers Ltd., Reading, Massachusetts.
5. Chen, D.C. (1985) Progress in knowledge-based flight monitoring, in Proceedings of the Second Conference on Artificial Intelligence Applications, *The Engineering of Knowledge-Based Systems*, IEEE Computer Society Press, Washington, December, pp. 441-446.
6. Date, C.J. (1990). *An Introduction to Database Systems*, Volume I, Fifth Edition, Addison-Wesley Publishing Coy., USA, 1990.

7. Erwin. G.J., and Bowen, P.A.(1994). Describing the building procurement process using script-based knowledge representation. *South African Journal of Science*, Vol. 90, No. 10, October, 1994, pp. 543-546.
8. Everest, G.C. 1986. *Database Management : Objectives, System Functions and Administration*, McGraw-Hill, USA.
9. Larsson, J.E. and Persson, P. (1986) Knowledge representation by scripts in an expert interface, in *Proceedings of the American Control Conference*, pp. 1159-1162.
10. Laskowski, S.J. and Hoffman, C.N. (1987) Script-based reasoning for situation monitoring, in *Proceedings of AAAI-87*, July, pp. 819-823.
11. Markosian, L.Z., Rockmore, A.J., Keller, K.J. and Gaan, M.R. (1985) An expert system for air-to-air combat management, in *Proceedings of the Eighteenth Asimolar Conference on Circuits, Systems, and Computers*, pp. 112-116.
12. Mutchler, C.N. and Laskowski, S.J. (1990) Script matching and belief propagation, in *Proceedings of the 5th I.E.E.E. International Symposium on "Intelligent Control"*, September.
13. Schank, R.C. and Abelson, R.P. (1977) *Scripts, Plans, Goals and Understanding : An Inquiry into Human Knowledge Structures*. Lawrence Erlbaum Associates, Hillsdale, New Jersey.
14. Schank, R.C. and Riesbeck, C.K. (Eds.) (1981) *Inside Computer Understanding : Five programs plus miniatures*. Lawrence Erlbaum Associates, Hillsdale, New Jersey.
15. Schank, R.C. and Childers, P. (1984) *The Cognitive Computer : On language, learning and artificial intelligence*. Addison-Wesley, Reading, Massachusetts.
16. Waterman, D.A. (1986) *A Guide to Expert* Systems. Addison-Wesley, Reading, Massachusetts.

Acknowledgements

Support work for EAGLE was done by Karl Fischer, a Business Information Systems (Honours) student in the Department of Accounting and Finance, Business Information Systems Section, University of Natal, King George V Ave., Durban, South Africa, 4001.
LISP code for the MATCHER was developed by Anna Richter c/o Mr A. Richter, Dept. of Mechanical Engineering, University of Natal, King George V Ave., Durban, South Africa, 4001.