©1996 Copyrights reside with the original authors who may be contacted directly

ISBN 0-620-20568-7

Cover printed by Natal Printers (Pty) Ltd, Pietermaritzburg
Copying by the Multicopy Centre, University of Natal, Pietermaritzburg
Binding by Library Technical Services, University of Natal, Pietermaritzburg

The views expressed in this book are those of the individual authors
FOREWORD

This book is a collection of papers presented at the National Research and Development Conference of the Institute of Computer Scientists and Information Technologists, held on 26 & 27 September, at the Interaction Conference Centre, University of Natal, Durban. The Conference was organised by the Department of Computer Science and Information Systems of The University of Natal, Pietermaritzburg.

The papers contained herein range from serious technical research to work-in-progress reports of current research to industry and commercial practice and experience. It has been a difficult task maintaining an adequate and representative spread of interests and a high standard of scholarship at the same time. Nevertheless, the conference boasts a wide range of high quality papers. The program committee decided not only to accept papers that are publishable in their present form, but also papers which reflect this potential in order to encourage young researchers and to involve practitioners from commerce and industry.

The organisers would like to thank IBM South Africa for their generous sponsorship and all the members of the organising and program committees, and the referees for making the conference a success. The organisers are indebted to the Computer Society of South Africa (Natal Chapter) for promoting the conference among its members and also to the staff and management of the Interaction Conference Centre for their contribution to the success of the conference.

On behalf of the Organising Committee
Vevek Ram
Editor and Program Chair
Pietermaritzburg, September 1996
Organising Committee

Conference General Chairs
Mr Rob Dempster and Prof Peter Warren (UNP)

Organising Chair
Dr Don Petkov (UNP)

Secretariat
Mrs Jenny Wilson

Program Chair
Prof Vevek Ram (UNP)

Program Committee
Prof Peter Wentworth, Rhodes
Dr Milan Hajek, UDW
Prof Derek Smith, UCT
Prof Anthony Krzesinski, Stellenbosch
Dr Don Petkov, UNP
Mr Rob Dempster, UNP
Prof Peter Warren, UNP
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>i</td>
</tr>
<tr>
<td>Organising Committee</td>
<td>ii</td>
</tr>
<tr>
<td>List of Contributors</td>
<td>vi</td>
</tr>
<tr>
<td><strong>Keynote Speaker</strong></td>
<td></td>
</tr>
<tr>
<td><em>The Role of Formalism in Engineering Interactive Systems</em></td>
<td>1</td>
</tr>
<tr>
<td>M D Harrison and D J Duke</td>
<td></td>
</tr>
<tr>
<td><strong>Plenary</strong></td>
<td></td>
</tr>
<tr>
<td><em>Industry-Academic-Government Cooperation to boost Technological Innovation and People Development in South Africa</em></td>
<td>15</td>
</tr>
<tr>
<td>Tjaart J Van Der Walt</td>
<td></td>
</tr>
<tr>
<td><em>Checklist support for ISO 9001 audits of Software Quality Management Systems</em></td>
<td>17</td>
</tr>
<tr>
<td>A J Walker</td>
<td></td>
</tr>
<tr>
<td><em>The IS Workers, they are a-changin'</em></td>
<td>29</td>
</tr>
<tr>
<td>Derek Smith</td>
<td></td>
</tr>
<tr>
<td><strong>Research</strong></td>
<td></td>
</tr>
<tr>
<td><em>Examination Timetabling</em></td>
<td>35</td>
</tr>
<tr>
<td>E Parkinson and P R Warren</td>
<td></td>
</tr>
<tr>
<td><em>Generating Compilers from Formal Semantics</em></td>
<td>43</td>
</tr>
<tr>
<td>H Venter</td>
<td></td>
</tr>
<tr>
<td><em>Efficient State-exploration</em></td>
<td>63</td>
</tr>
<tr>
<td>J. Geldenhuys</td>
<td></td>
</tr>
<tr>
<td><em>A Validation Model of the VMTP Transport Level Protocol</em></td>
<td>75</td>
</tr>
<tr>
<td>H.N. Roux and P.J.A. de Villiers</td>
<td></td>
</tr>
<tr>
<td><strong>Intelligent Systems</strong></td>
<td></td>
</tr>
<tr>
<td><em>Automated Network Management using Artificial Intelligence</em></td>
<td>87</td>
</tr>
<tr>
<td>M Watzenboeck</td>
<td></td>
</tr>
<tr>
<td><em>A framework for executing multiple computational intelligent programs using a computational network</em></td>
<td>89</td>
</tr>
<tr>
<td>H L Viktor and I Cloete</td>
<td></td>
</tr>
<tr>
<td><em>A Script-Based prototype for Dynamic Deadlock Avoidance</em></td>
<td>95</td>
</tr>
<tr>
<td>C N Blewett and G J Erwin</td>
<td></td>
</tr>
<tr>
<td><em>Parallelism: an effective Genetic Programming implementation on low-powered Mathematica workstations</em></td>
<td>107</td>
</tr>
<tr>
<td>H. Suleman and M. Hajek</td>
<td></td>
</tr>
<tr>
<td><em>Feature Extraction Preprocessors in Neural Networks for Image Recognition</em></td>
<td>113</td>
</tr>
<tr>
<td>D Moodley and V Ram</td>
<td></td>
</tr>
</tbody>
</table>
## Real-Time Systems

**The real-time control system model - an Holistic Approach to System Design**  
T Considine  

**Neural networks for process parameter identification and assisted controller tuning for control loops**  
M McLeod and VB Bajic  

**Reference Model for the Process Control Domain of Application**  
N Dhhevcharan, A L Steenkamp and V Ram  

## Database Systems

**The Pearl Algorithm as a method to extract information out of a database**  
J W Kruger  

**Theory meets Practice: Using Smith’s Normalization in Complex Systems**  
A van der Merwe and W Labuschagne  

**A Comparison on Transaction Management Schemes in Multidatabase Systems**  
K Renaud and P Kotze  

## Education

**Computer-based applications for engineering education**  
A C Hansen and P W L Lyne  

**Software Engineering Development Methodologies applied to Computer-Aided Instruction**  
R de Villiers and P Kotze  

**COBIE: A Cobol Integrated Environment**  
N Pillay  

**The Design and Usage of a new Southern African Information Systems Textbook**  
G J Erwin and C N Blewett  

**Teaching a first course in Compilers with a simple Compiler Construction Toolkit**  
G Ganchev  

**Teaching Turing Machines: Luxury or Necessity?**  
Y Velinov  

## Practice and Experience

**Lessons learnt from using C++ and the Object Oriented Approach to Software Development**  
R Mazhindu-Shumba  

**Parallel hierarchical algorithm for identification of large-scale industrial systems**  
B Jankovic and VB Bajic
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information Technology and Organizational Issues</td>
<td></td>
</tr>
<tr>
<td><em>A cultural perspective on IT/End user relationships</em></td>
<td>243</td>
</tr>
<tr>
<td>A C Leonard</td>
<td></td>
</tr>
<tr>
<td><em>Information Security Management: The Second Generation</em></td>
<td>257</td>
</tr>
<tr>
<td>R Von Solms</td>
<td></td>
</tr>
<tr>
<td><em>Project Management in Practice</em></td>
<td>267</td>
</tr>
<tr>
<td>M le Roux</td>
<td></td>
</tr>
<tr>
<td><em>A Case-Study of Internet Publishing</em></td>
<td>271</td>
</tr>
<tr>
<td>A Morris</td>
<td></td>
</tr>
<tr>
<td><em>The Role of IT in Business Process Reengineering</em></td>
<td>285</td>
</tr>
<tr>
<td>C Blewett, J Cansfield and L Gibson</td>
<td></td>
</tr>
<tr>
<td><strong>Abstracts</strong></td>
<td></td>
</tr>
<tr>
<td><em>On Total Systems Intervention as a Systemic Framework for the</em></td>
<td>299</td>
</tr>
<tr>
<td><em>Organisation of the Model Base of a Decision Support Systems</em></td>
<td></td>
</tr>
<tr>
<td>Generator</td>
<td></td>
</tr>
<tr>
<td>D Petkov and O Petkova</td>
<td></td>
</tr>
<tr>
<td><em>Modular Neural Networks Subroutines for Knowledge Extraction</em></td>
<td>300</td>
</tr>
<tr>
<td>A Vahed and I Cloete</td>
<td></td>
</tr>
<tr>
<td><em>Low-Cost Medical Records System: A Model</em></td>
<td>301</td>
</tr>
<tr>
<td>O A Dami and T Seipone</td>
<td></td>
</tr>
<tr>
<td><em>A Methodology for Integrating Legacy Systems with the Client/Server</em></td>
<td>302</td>
</tr>
<tr>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td>M Redelinghuys and A L Steenkamp</td>
<td></td>
</tr>
<tr>
<td><em>Information Systems Outsourcing and Organisational Structure</em></td>
<td>303</td>
</tr>
<tr>
<td>M Hart and Kavatzandis</td>
<td></td>
</tr>
<tr>
<td><em>The relational organisation model</em></td>
<td>304</td>
</tr>
<tr>
<td>B Laauwen</td>
<td></td>
</tr>
<tr>
<td><em>The Practical Application of a New Class of Non-Linear Smoothers for</em></td>
<td>305</td>
</tr>
<tr>
<td><em>Digital Image Processing</em></td>
<td></td>
</tr>
<tr>
<td>E Cloete</td>
<td></td>
</tr>
<tr>
<td><em>A Technology Reference Model for Client/Server Software Development</em></td>
<td>306</td>
</tr>
<tr>
<td>R C Nienaber</td>
<td></td>
</tr>
<tr>
<td><em>The Feasibility Problem in the Simplex Algorithm</em></td>
<td>307</td>
</tr>
<tr>
<td>T G Scott, J M Hattingh and T Steyn</td>
<td></td>
</tr>
<tr>
<td><strong>Author Index</strong></td>
<td>309</td>
</tr>
</tbody>
</table>
List of Contributors

Vladimir B Bajic  
Centre for Engineering Research,  
Technikon Natal,  
P O Box 953  
Durban 4000

Ruth de Villiers  
Department of Computer Science and  
Information Systems  
UNISA  
Box 392, Pretoria, 0001

C N Blewett  
Department of Accounting  
University of Natal  
King George V Avenue  
Durban 4001

G J Erwin  
Business Information Systems  
University of Durban-Westville  
Private Bag X54001  
Durban 4000

Justin Cansfield  
Department of Accounting  
University of Natal  
King George V Avenue  
Durban 4001

G Ganchev  
Computer Science Department  
University of Botswana  
PBag 0022  
Gaberone, Botswana

Tom Considine  
Apron Services (Pty) Ltd  
P O Johannesburg  
International Airport  
1600

J Geldenhuys  
Department of Computer Science  
University of Stellenbosch  
Stellenbosch  
7700

Eric Cloete  
School of Electrical Engineering  
Cape Technikon  
Box 652  
Cape Town

Louise Gibson  
BIS, Dept Accounting & Finance  
University of Durban  
Pvt Bag X10  
Dalbridge 4014

I Cloete  
Computer Science Department  
University of Stellenbosch  
Stellenbosch  
7600

Mike Hart  
Department of Information Systems  
University of Cape Town  
Rondebosch  
7700

O A Daini  
Department of Computer Science  
University of Botswana  
Gaborone  
Botswana

M. Hajek  
Department of Computer Science  
University of Durban-Westville  
Pvt Bag X54001  
Durban 4000

Nirvani Devcharan  
Umgeni Water  
Box 9  
Pietermaritzburg  
3200

A C Hansen  
Dept of Agricultural Engineering  
University of Natal  
Private Bag X01  
Scottsville 3209

P J A de Villiers  
Department of Computer Science  
University of Stellenbosch  
Stellenbosch  
7700

J M Hattingh  
Department of Computer Science  
Potchefstroom University for CHE  
Potchefstroom 2520
V Ram
Department of Computer Science and Information Systems
University of Natal
PBag x01
Scottsville 3209

H. Suleman
Department of Computer Science
University of Durban-Westville
Pvt Bag X54001
Durban 4000

Melinda Redelinghuys
Department of Computer Science and Information Systems
UNISA
Box 392
Pretoria, 0001

A Vahed
Department of Computer Science
University of Western Cape
Private Bag X17
Bellarive 7530

Karen Renaud
Computer Science and Information Systems
UNISA
Box 392
Pretoria, 0001

A Van der Merwe
Computer science and Information Systems
UNISA
P O Box 392
Pretoria, 0001

H N Roux
Department of Computer Science
University of Stellenbosch
Stellenbosch
7700

Tjaart J Van Der Walt
Foundation for Research and Development
Box 2600
Pretoria, 0001

T G Scott
Department of Computer Science
Potchefstroom University for CHE
Potchefstroom
2520

K Vavatzandis
Department of Information Systems
University of Cape Town
Rondebosch
7700

T Seipone
Department of Computer Science
University of Botswana
Gaborone
Botswana

Y Velinov
Dept Computer Science
University of Natal
Private Bag X01
Scottsville 3209

Derek Smith
Department of Information Systems
University of Cape Town
Rondebosch
7700

H Venter
Department of Computer Science
University of Port Elizabeth
Box 1600
Port Elizabeth 6000

Anette L Steenkamp
Department of Computer Science and Information Systems
UNISA
Box 392
Pretoria, 0001

H L Viktor
Computer Science Department
University of Stellenbosch
Stellenbosch
7600

T Steyn
Department of Computer Science
Potchefstroom University for CHE
Potchefstroom 2520

R Von Solms
Department of Information Technology
Port Elizabeth Technikon
Private Bag X6011
Port Elizabeth 6000
A J Walker
Software Engineering Applications
Laboratory
Electrical Engineering
University of Witwatersrand
Johannesburg

P Warren
Computer Science Department
University of Natal
P/Bag X01
Scottsville 3209

Max Watzenboeck
University of Botswana
Private Bag 0022
Gaberone
Botswana
A Script-based prototype for Dynamic Deadlock Avoidance

C N Blewett* and G J Erwin

Abstract

Expert systems apply Artificial Intelligence (AI) techniques to an application area, aiming (usually) to mimic the behaviour of a human expert. However, there are some AI techniques which can be used to improve the internal performance of an existing application, not necessarily currently performed by a human. In this paper, we present further research and results of EAGLE (External Advisor for Granting Locks Expertly), an expert system advisor for the lock manager in a database system. By matching lock event sequences received from the lock manager against stored scripted deadlock sequences, EAGLE is able to identify unfolding deadlock sequences. By using this Dynamic Deadlock Avoidance (DDA) approach, EAGLE is able to avoid deadlocks before they occur. Currently, no ideal solution exists to the deadlock problem. Solutions vary in terms of the number of waits for access to resources and the number of deadlock occurrences. By utilising AI techniques, DDA offers a new way of treating the deadlock problem. In this paper we describe the design of EAGLE and present the results, in terms of the number of waits and number of deadlock occurrences, occurring with DDA compared with Deadlock Detection and Resolution.

Keywords

database system, deadlock, expert system, scripts, knowledge representation, machine learning, plan recognition

1. Introduction

"Deadlock is a situation in a resource allocation system in which two or more processes are in a simultaneous wait state, each one waiting for one of the others to release a resource before it can proceed." Deadlock may occur within database, communication and distributed systems using locking as a concurrency control strategy [3]. The deadlock state and its treatment are well-described. See, for example, [1], [5], [7], [10], [12] and [22].

Blewett and Erwin [6] describe four categories of deadlock treatment: viz. ignore deadlock, deadlock detection and resolution, deadlock prevention and deadlock avoidance.

Detection with recovery allows the system to enter a deadlock state and then recover from it. Prevention ensures that a deadlock will never happen. Avoidance should also ensure no deadlocks, but, can only do that by using a priori data about the locking activities of transactions (resource consumers) [3].

The dynamic deadlock avoidance (DDA) approach described in this paper does not require pre-stored knowledge of lock request sequences, but attempts to notice potential deadlocks based on experience accumulated from observations of the previous locking event sequences which led to deadlock. DDA allows deadlock to occur, records the conditions which led to the deadlock, then watches for similar, deadlock-producing conditions in future.

DDA treats the deadlock problem as a plan recognition issue [4], rather than as a problem resolution issue. The DDA approach attempts to identify whether the "goal" of the running "plan" of current locking event sequences is deadlock. When this goal is recognised, DDA OBJECTs to further locking activities which could lead to a future deadlock.

In the following sections we first describe the overall design of our DDA-based prototype, called EAGLE (External Advisor for Granting Locks Expertly), incorporating script-based knowledge representation [18], [19] and [20], in a centralised resource allocation (database) system. We then discuss the implementation of the EAGLE prototype, EAGLE's components, and test runs under simulated conditions within a database system. We describe the sets of test data used to assess the impact of EAGLE on the occurrence of deadlock under those simulated conditions and present some preliminary results from those test runs.
2. 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." [17]. Schank and Abelson [18] 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 events with the following characteristics: duration: a start and a finish; dependency: certain events occur before/after others, and stereotypical aspects: event sequences are often predictable and well-understood.

The script-based knowledge representation is a specific application of the frame knowledge representation. Each frame's slot definitions are constructed to apply to the specific application under consideration [21].

Scripts were initially formulated, [18], [19] and [20], to provide structures and manipulative ability for the handling of time and dependency-based events in the domain of "understanding" natural language. This work established conceptual dependency operators within natural language as a formal methodology for representing the semantic content of sentences and stories. Scripts are 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, [4], [9], [11], [14], [15] and [16].

3. Representing locking event sequences as scripts

Locking event sequences exhibit, inter alia, the same characteristics as scripts. 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. In the EAGLE context, we regard the "goal" of these locking events sequences as the attainment of deadlock. EAGLE formats locking event sequences as scripts, and stores those scripts which resulted in deadlock.

Each frame (script) in DDA 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 and maximum activation level.

The script name uniquely identifies each script. The process roles and resource props identify the number of transactions and resources involved in the deadlock. The expected event is used to look for the next expected event for a script. Any objection to the granting of a lock request must take place before the critical event. The sequence description describes a stereotypical event sequence and the response is the result of a match against this sequence description. The various activation and utilisation levels monitor and control the activation of the script.

4. EAGLE Implementation

The EAGLE system was developed in the Clipper and Common LISP. Clipper was used for the simulation of database activities, and Common LISP was used for the Matcher/Analyser which generates stereotypical event sequences and compares locking event sequences against stored stereotypical event sequences. EAGLE is the script-based implementation of the DDA scheme. Incoming lock requests (as part of locking event sequences) are passed from the lock manager (LOMAN) to EAGLE, where they are matched against stored stereotypical locking event sequences, representing various patterns of lock events which lead to deadlock situations. Should an event (lock or unlock request) in a locking event sequence cause a match to occur with a stored stereotype script, the lock or unlock request is rejected. If no match occurs, the lock event is approved. Fig. 1 below shows an overview of the EAGLE operating environment.
LOMAN receives transactions from the Transaction Manager. LOMAN checks its 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 inspection. EAGLE adds the current lock request to the script for that transaction. EAGLE then matches the current content of the locking event sequence against stereotypical scripts to estimate the probability of a future deadlock occurring.

EAGLE calculates a similarity metric (SM) or belief [17] for each instantiated script to determine the degree of relevance of a script to an observed locking event sequence. This is similar to Benoit et al.'s [4] SCAN system, where an overall likelihood is computed for each of the scripts. EAGLE either denies or approves the lock request based on the computed SM. 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 EAGLE system initially has no stored scripts describing deadlock situations stored. As a result, all "early" lock requests are granted. When a deadlock occurs, the contents of that locking event sequence are passed to the Learner sub-system of EAGLE. Here, the current locking event sequence is converted into both a specific stereotypical event sequence (one which contains lock, unlock, and wait tasks) and into a generic event sequence (one which contains lock and wait tasks of only the transactions involved in the deadlock). As further deadlocks occur, so the number of stored scripts increases, building a Script Base. The Matcher inside EAGLE continues to match fresh locking event sequences against the Script Base. If a match is made, EAGLE OBJECTs to the current request, and returns that OBJECTion to LOMAN. EAGLE has four main components, as depicted below in Fig 2.
Consider each component in turn;

* **Rule Component.** The rule component has two sub-components, namely the Communication Interface and the Activation System. The Communication Interface is responsible for passing messages and responses between LOMAN, the Matcher/Analyzer and the Script Base. The Activation System is responsible for activating the Matcher/Analyzer when the Precondition Header (at least 2 concurrent processes in operation) and the Instrumental Header (at least one lock request for a resource required by another transaction) are satisfied.

* **Matcher/Analyzer Component.** The second component of EAGLE is the Matcher/Analyzer, based on the IPS architecture of [4]. The Matcher/Analyzer is activated by the Rule Component. The Matcher/Analyzer receives the current locking event sequence from the Communication Interface of the Rule Component. The Matcher/Analyzer compares the current locking event sequence against stereotypical locking event sequences stored in the Script Base. If the conditions for an OBJECTion are met, the Matcher/Analyzer will return an OBJECT response to the Rule Component, and the transaction lock request is not granted. Fig. 3 below depicts an OBJECT being made based on a successful script instantiation. If no other transaction has made any requests, the same transaction may attempt to re-request the resource. This component also converts actual locking event sequences into stereotypical format.

* **Script Base Component.** The third component of EAGLE is the Script Base. The Script Base holds the stereotypical locking event sequences which have been encountered previously. There are two types of stereotypes, namely, a specific stereotype and a generic stereotype. A Specific Stereotype describes a stereotypical event sequence in the context of...
the current operational environment, *i.e.* a specific stereotype is not applicable in another context. A specific stereotype records all lock, unlock, and wait events of deadlock participants and non-deadlock participants. A *Generic Stereotype* describes scripts in a domain-independent context. No details of non-participating transactions' events or unlock requests are recorded in the generic stereotype. Generic stereotypes are useful in situations where specific stereotypes have not yet been defined.

* Learner Component. The Learner component is activated by the Rule Component whenever LOMAN's deadlock detector detects a deadlock. To detect deadlocks, LOMAN constructs wait-for-graphs which are analysed for cycles showing the presence of deadlock. The deadlock detection technique is similar to Topological Sorting [3], using a graph reduction approach with an adjacency list representation of the graph. The entire list is scanned for cycles. The details of the event sequence leading to the detected deadlock are converted into specific and generic stereotypical forms and stored in the Script Base. The utilisation of the scripts recorded in the Script Base is continually monitored. The Activation Level of the script is adjusted to tune the script to an effective level of performance. Over-utilisation of a script can lead to unnecessary system interference, and under-utilisation of a script wastes system resources. Scripts failing to perform within the defined parameters, after adjustments have been made, are removed from the Script Base.

5. Test runs

A simulated transaction environment was developed in which EAGLE operated. A series of simulated transactions was run in order to test and evaluate EAGLE's impact on the occurrence of deadlock. Simulated transactions occurred in two situations, *viz.* (i) EAGLE active, *i.e.* Dynamic Deadlock Avoidance (DDA), and (ii) EAGLE inactive, *i.e.* Deadlock Detection (DLD). [5] identified the following assumptions for evaluating a concurrency control algorithm, *viz.* "All transactions require the same number of locks, all data items are accessed with equal probability, and all locks are write locks. The transactions use Strict 2PL: data items are locked before they are accessed, and locks are released only after all transactions commit (or abort). The database is centralised...". See [8] for discussion of these assumptions.

5.1 Simulation parameters

Several parameters were used within the transaction simulations. Considering each parameter in turn:

* **Transaction Size.** This is the number of resources processed by transactions. Except for run 8, all transactions accessed the same number of resources. Varying the transaction size alters the number of possible deadlock situations, and hence the number of scripts required to represent the stereotypical deadlock situations.

* **Multiprogramming Level (MPL).** The MPL is the number of transactions that are processed concurrently in a run. The higher the MPL the higher the potential resource conflict, and consequently the higher the probability of deadlock. The MPL parameter varies from 2 to 5 transactions.

* **Resource Contention.** This is the proportion of the resources used by all participating transactions. A value of 1.0 means that all transactions require all resources, resulting in higher resource contention. A value of 0.5 means that only half of the resources required are common to all transactions, so resulting in lower resource contention. This parameter allows for larger transaction sizes with lower resource contention. This parameter was set to 1.0 except in Run 8 and Run 12. In Run 8 it was set at 0.5 and in Run 12 to 0.2. This allowed for a low conflict situation between 4 concurrent transactions.

* **Activation Level.** This is the degree of similarity required (measured between 0, no similarity, and 1, exact match) between an observed locking event sequence and a stored script to activate the script. The activation level of all new scripts is initially at 0.5. This level is adjusted as scripts are utilised.

* **Clean Level.** This parameter determines when un/under-utilised scripts are removed from the Script Base. The clean level is the number of activations of the Matcher/Analyser after which un/under-utilised scripts are removed. The clean level was set at 200. Decreasing the level below 200 caused many scripts to be removed from the database that had not yet been given sufficient opportunity to be activated. Increasing the clean level above 200 caused the number of scripts in the Script Base to grow too large, thus resulting in long matching times and multiple OBJECT ceilings (see below). The clean level was set at 200.

* **OBJECT Ceiling.** In order to avoid an EAGLE-lock (deadlock-like) situation arising, a limit was set for the number of consecutive OBJECTions. If no such limit is set, it is possible that no transactions will proceed, because all transactions match a script in the Script Base. The OBJECT ceiling limits the number of consecutive OBJECTions that will be permitted. Once this ceiling value is reached, any further lock request is automatically granted. Setting the OBJECT ceiling too low *e.g.* 2, means that many valid OBJECTions are ignored. However, setting the OBJECT ceiling too high *e.g.* 10, slows the forward progress of transactions. An OBJECT ceiling of 5 was found to be an effective value, providing a trade-off between ignoring valid OBJECTions and impeding transaction progress.

* **Leamer Component.** The Learner component is activated by the Rule Component whenever LOMAN's deadlock detector detects a deadlock. To detect deadlocks, LOMAN constructs wait-for-graphs which are analysed for cycles showing the presence of deadlock. The deadlock detection technique is similar to Topological Sorting [3], using a graph reduction approach with an adjacency list representation of the graph. The entire list is scanned for cycles. The details of the event sequence leading to the detected deadlock are converted into specific and generic stereotypical forms and stored in the Script Base. The utilisation of the scripts recorded in the Script Base is continually monitored. The Activation Level of the script is adjusted to tune the script to an effective level of performance. Over-utilisation of a script can lead to unnecessary system interference, and under-utilisation of a script wastes system resources. Scripts failing to perform within the defined parameters, after adjustments have been made, are removed from the Script Base.

5. Test runs

A simulated transaction environment was developed in which EAGLE operated. A series of simulated transactions was run in order to test and evaluate EAGLE's impact on the occurrence of deadlock. Simulated transactions occurred in two situations, *viz.* (i) EAGLE active, *i.e.* Dynamic Deadlock Avoidance (DDA), and (ii) EAGLE inactive, *i.e.* Deadlock Detection (DLD). [5] identified the following assumptions for evaluating a concurrency control algorithm, *viz.* "All transactions require the same number of locks, all data items are accessed with equal probability, and all locks are write locks. The transactions use Strict 2PL: data items are locked before they are accessed, and locks are released only after all transactions commit (or abort). The database is centralised...". See [8] for discussion of these assumptions.

5.1 Simulation parameters

Several parameters were used within the transaction simulations. Considering each parameter in turn:

* **Transaction Size.** This is the number of resources processed by transactions. Except for run 8, all transactions accessed the same number of resources. Varying the transaction size alters the number of possible deadlock situations, and hence the number of scripts required to represent the stereotypical deadlock situations.

* **Multiprogramming Level (MPL).** The MPL is the number of transactions that are processed concurrently in a run. The higher the MPL the higher the potential resource conflict, and consequently the higher the probability of deadlock. The MPL parameter varies from 2 to 5 transactions.

* **Resource Contention.** This is the proportion of the resources used by all participating transactions. A value of 1.0 means that all transactions require all resources, resulting in higher resource contention. A value of 0.5 means that only half of the resources required are common to all transactions, so resulting in lower resource contention. This parameter allows for larger transaction sizes with lower resource contention. This parameter was set to 1.0 except in Run 8 and Run 12. In Run 8 it was set at 0.5 and in Run 12 to 0.2. This allowed for a low conflict situation between 4 concurrent transactions.

* **Activation Level.** This is the degree of similarity required (measured between 0, no similarity, and 1, exact match) between an observed locking event sequence and a stored script to activate the script. The activation level of all new scripts is initially at 0.5. This level is adjusted as scripts are utilised.

* **Clean Level.** This parameter determines when un/under-utilised scripts are removed from the Script Base. The clean level is the number of activations of the Matcher/Analyser after which un/under-utilised scripts are removed. The clean level was set at 200. Decreasing the level below 200 caused many scripts to be removed from the database that had not yet been given sufficient opportunity to be activated. Increasing the clean level above 200 caused the number of scripts in the Script Base to grow too large, thus resulting in long matching times and multiple OBJECT ceilings (see below). The clean level was set at 200.

* **OBJECT Ceiling.** In order to avoid an EAGLE-lock (deadlock-like) situation arising, a limit was set for the number of consecutive OBJECTions. If no such limit is set, it is possible that no transactions will proceed, because all transactions match a script in the Script Base. The OBJECT ceiling limits the number of consecutive OBJECTions that will be permitted. Once this ceiling value is reached, any further lock request is automatically granted. Setting the OBJECT ceiling too low *e.g.* 2, means that many valid OBJECTions are ignored. However, setting the OBJECT ceiling too high *e.g.* 10, slows the forward progress of transactions. An OBJECT ceiling of 5 was found to be an effective value, providing a trade-off between ignoring valid OBJECTions and impeding transaction progress.

* **Leamer Component.** The Learner component is activated by the Rule Component whenever LOMAN's deadlock detector detects a deadlock. To detect deadlocks, LOMAN constructs wait-for-graphs which are analysed for cycles showing the presence of deadlock. The deadlock detection technique is similar to Topological Sorting [3], using a graph reduction approach with an adjacency list representation of the graph. The entire list is scanned for cycles. The details of the event sequence leading to the detected deadlock are converted into specific and generic stereotypical forms and stored in the Script Base. The utilisation of the scripts recorded in the Script Base is continually monitored. The Activation Level of the script is adjusted to tune the script to an effective level of performance. Over-utilisation of a script can lead to unnecessary system interference, and under-utilisation of a script wastes system resources. Scripts failing to perform within the defined parameters, after adjustments have been made, are removed from the Script Base.
Fig. 4 below summarises the main simulation parameters used:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAN_SIZE</td>
<td>The resource size of a transaction, varies from 2 to 5.</td>
</tr>
<tr>
<td>MPL</td>
<td>Multi-programming level, varies from 2 to 5 transactions.</td>
</tr>
<tr>
<td>RESOURCE CONTENTION</td>
<td>Proportion of resources used by all participating transactions, varies from 0.2 to 1.0.</td>
</tr>
<tr>
<td>ACTLEVEL</td>
<td>Activation Level - The degree of similarity necessary to activate a script. The initial ActLevel is 0.50.</td>
</tr>
<tr>
<td>OBJCEIL</td>
<td>The OBJECT ceiling is the maximum number of consecutive OBJECTions allowed. Set at 5.</td>
</tr>
</tbody>
</table>

Fig. 4. EAGLE simulation run parameters.

5.2 Test data

Twelve sets of test data were used, with varying MPLs and TRAN_SIZEs. As performance was not being measured, it was not necessary for transactions to perform any actual processing/updating of a real database. Simulated transactions acquire locks and then, on completion, release all acquired locks. The MPL varies between 2 and 5 concurrent transactions, and the TRAN_SIZE between 2 and 5 resources. These values were chosen for the following reasons:

* Simulation Time. The runs were constructed to test the algorithm under varying degrees of resource contention. The MPL and TRAN_SIZEs were therefore “selected so as to jointly yield a region of operation which allows the interesting performance effects to be observed without necessitating impossibly long simulation times” [1]. This is particularly true in the simulated environment, when DDA is active, as all lock requests have to be passed to the Matcher/Analyser for evaluation. As the Matcher/Analyser was implemented in interpreted LISP, increasing the MPL or TRAN_SIZE results in more complex matches and extremely long elapsed times for simulation runs.

* Rarity of Conflict. Besides the problem of long simulation run times, with high MPLs and large TRAN_SIZEs, “conflicts become very rare, and when conflicts are rare, all concurrency control algorithms perform alike” [1].

* Heavy Conflict Workloads. The chosen MPLs and TRAN_SIZEs yield a relatively high mean number of deadlocks as a result of high levels of conflict (measured by mean number of waits). [2] indicates that a high deadlock rate is 100 per hour, and the simulation runs can exceed that figure. For example, run 4 resulted in approx. 300 deadlocks in one hour. These “non-negligible conflict levels ... facilitate understanding how the algorithms will perform under heavy conflict workloads, or when 'hot spots' exist in the database.” [1].

* Realistic levels. A MPL of 5 is comparable to the general workload of a conventional order entry system [13].

5.3 Nature of the Test

Each of the 12 sets of data was processed 200 times (except for 3 runs where this was not possible because of the long simulation times involved). The data sets of data were processed 200 times to produce sufficiently large sample sizes and so minimise the effects of outliers. Each set of data was processed both with EAGLE on (DDA) and then with EAGLE off (DLD). The transactions are passed to LOMAN from the Transaction module in a random, interleaved manner. LOMAN, after approving a lock event, passes the event sequence to EAGLE for approval. EAGLEs OBJECT or NO OBJECT response is then returned to LOMAN.

6. The Simulation results

There were 12 sets (runs) of transaction data constructed with various combinations of MPL, TRAN_SIZE and RESOURCE CONTENTION. Each of the 12 runs was executed (simulated) 200 (usually) times (cycles), with random selection of the sequence in which transactions were selected for forward progress. Three of the runs were not done for 200 cycles, because of the infeasible (long) simulation elapsed times involved. Each set of data was processed both with EAGLE on (DDA) and with EAGLE off (DLD). The number of locks granted, waits (denied locks), unlock requests, deadlock occurrences etc. were recorded in a report file. Results from the twelve simulation runs are summarised in Fig. 5 below. The first four rows (RUN NO., MPL, TRAN_SIZE, RESOURCE CONTENTION) describe the run. The SAMPLE SIZE is the number of times (cycles) the same set of transactions was processed. The last 4 rows contain the MEAN number of DEADLOCKS and the MEAN number of WAITS per cycle of transactions for both the EAGLE runs (DDA) and the runs without EAGLE (DLD).
As an illustration, in Runs 2 and 3, MEAN WAITS using DDA is less than MEAN WAITS using DLD. In Runs 4 and 11, MEAN WAITS using DDA is more than MEAN WAITS using DLD. In Runs 2 and 3, MEAN DEADLOCKS occurring with DDA is less than that occurring with DLD. However, in Runs 6 and 11 the MEAN DEADLOCKS is more with DDA than when DLD is used.

A moving average provides a visual indication of the effect of introducing DDA into simulated locking activity. A moving average of the number of occurrences of deadlock per transaction cycle was calculated for each run. The moving average period was set to 20. The moving average chart for one of the Runs (Run 8) is shown in Fig. 6 below. The impact of EAGLE with DDA, relative to the "normal" Deadlock Detection and Resolution (DLD) without EAGLE, is clearly (visually) apparent. RUN 8 began with DLD processing and no "advice" or learning from EAGLE, with an average number of deadlocks per transaction cycle of 0.52. Then, beginning at transaction cycle 201, EAGLE began to "advise" LOMAN on the granting of locks, and to learn about deadlock-inducing sequences of lock activities. The average number of deadlocks quickly decreased to 0.33.

### Table 1: Summary of Results

<table>
<thead>
<tr>
<th>RUN NO.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO. TRANS (MPL)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>NO. RESOURCES</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>RESOURCE UK</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAMPLE SIZE</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>110</td>
<td>200</td>
<td>50</td>
<td>200</td>
</tr>
<tr>
<td>MEAN WAITS (DLD)</td>
<td>1.56</td>
<td>2.13</td>
<td>2.84</td>
<td>5.82</td>
<td>20.2</td>
<td>436</td>
<td>10.8</td>
<td>3.08</td>
<td>12.6</td>
<td>8.23</td>
<td>243</td>
<td>6.23</td>
</tr>
<tr>
<td>MEAN WAITS (DDA)</td>
<td>1.76</td>
<td>193</td>
<td>2.35</td>
<td>6.05</td>
<td>19.4</td>
<td>456</td>
<td>11.0</td>
<td>2.55</td>
<td>11.9</td>
<td>7.83</td>
<td>253</td>
<td>6.05</td>
</tr>
<tr>
<td>MEAN DEADLOCKS (DLD)</td>
<td>0.52</td>
<td>0.73</td>
<td>0.87</td>
<td>1.65</td>
<td>6.73</td>
<td>1.19</td>
<td>3.64</td>
<td>0.52</td>
<td>3.91</td>
<td>2.33</td>
<td>7.04</td>
<td>1.36</td>
</tr>
<tr>
<td>MEAN DEADLOCKS (DDA)</td>
<td>0.57</td>
<td>0.58</td>
<td>0.61</td>
<td>1.65</td>
<td>6.23</td>
<td>1.23</td>
<td>3.62</td>
<td>0.33</td>
<td>3.63</td>
<td>2.09</td>
<td>7.4</td>
<td>1.26</td>
</tr>
</tbody>
</table>

**Fig. 5. Results of EAGLE simulation runs**

**Fig. 6. Run 8: Moving average of deadlock occurrences without EAGLE, then with EAGLE.**
However in Run 9, where the average number of deadlocks with deadlock detection was 3.91, the introduction of EAGLE does not seem to cause any noticeable perturbation in the number of deadlocks. After EAGLE is activated the number of deadlocks decreases marginally to 3.63. Figure 7 shows the moving average for Run 9.

**RUN 9 - MOVING AVERAGE**

**NUMBER OF DEADLOCKS**

![Graph showing the moving average of deadlock occurrences without EAGLE and with EAGLE.](image)

*Fig. 7. Run 9: Moving average of deadlock occurrences without EAGLE and with EAGLE.*

The simulation runs results indicate that, in certain situations, EAGLE reduces the number of deadlocks and waits occurring, while in other situations it seems to have no noticeable affect.

### 7. Evaluation of Results

Two sets of transactions were processed with the different deadlock treatment techniques (DDA and DLD). Other deadlock treatments exist [1], [3], [5], [10], [12], [13] and [22], but this study was restricted to a direct comparison between DDA and DLD. The effectiveness of DDA compared with DLD was measured with two of the "most important performance factors for locking" viz, "deadlock rate" (number of deadlocks) and "lock wait time" (number of waits). The third factor (lock conflict probability) is the primary measure of resource contention, which was not part of our investigation. All three factors "are interrelated" [13].

The null hypothesis was established as:

\[ H_0: \text{There is no decrease in the criteria measures (number of deadlocks and number of waits) occurring with DDA compared with DLD.} \]

Alternate hypotheses were also tested, but are not reported in this paper. The Mann-Whitney U-Test was selected for (most of the) testing for significant difference [8]. Because of the nature of some runs, it was not meaningful to draw any statistical conclusion. For example, Run 1 was a 2 transaction, 2 resource simulation, so DDA could not avoid deadlock dynamically. All statistical tests were evaluated initially at a 95% level of confidence.

For example, Figure 8 shows the results of the Mann-Whitney U-Test for Run 8.
The average rank of the DDA sample for the number of deadlocks (181) is lower than the average rank of the DLD sample for the number of deadlocks (220). The computed test statistic $Z$ is 3.94227 and the p value is $4.04\times10^{-5}$. The null hypothesis can be rejected because the p value ($4.04\times10^{-5}$) is lower than the set level of significance (0.05). It can be concluded that the number of deadlocks occurring under DDA is less than the number occurring under DLD.

In this run, at the 95% confidence interval the null hypothesis $H_0$ was rejected.

However, for many of the runs (such as Run 9), the null hypothesis was not rejected.

Figure 9 presents a summary of the means and significance levels for each of the 12 runs.

<table>
<thead>
<tr>
<th>Run</th>
<th>Wait Mean</th>
<th>DL Mean</th>
<th>DL Z stat</th>
<th>DL Sig.level</th>
<th>Wait Z stat</th>
<th>Wait Sig.level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.56</td>
<td>0.53</td>
<td>-0.8025</td>
<td>0.2111</td>
<td>-0.8598</td>
<td>0.1949</td>
</tr>
<tr>
<td>2</td>
<td>2.13</td>
<td>0.74</td>
<td>3.1404</td>
<td>0.0008</td>
<td>2.5193</td>
<td>0.0059</td>
</tr>
<tr>
<td>3</td>
<td>2.84</td>
<td>0.87</td>
<td>5.9195</td>
<td>1.62e-09</td>
<td>4.4928</td>
<td>3.52e-06</td>
</tr>
<tr>
<td>4</td>
<td>5.82</td>
<td>1.65</td>
<td>-0.3498</td>
<td>0.6367</td>
<td>-1.9613</td>
<td>0.9751</td>
</tr>
<tr>
<td>5</td>
<td>20.15</td>
<td>6.73</td>
<td>1.5781</td>
<td>0.0573</td>
<td>1.2622</td>
<td>0.1034</td>
</tr>
<tr>
<td>6</td>
<td>4.36</td>
<td>1.19</td>
<td>0.0981</td>
<td>0.4609</td>
<td>-0.5423</td>
<td>0.7062</td>
</tr>
<tr>
<td>7</td>
<td>10.8</td>
<td>3.64</td>
<td>0.0878</td>
<td>0.465</td>
<td>-0.3374</td>
<td>0.6321</td>
</tr>
<tr>
<td>8</td>
<td>3.08</td>
<td>0.52</td>
<td>3.9423</td>
<td>4.04e-05</td>
<td>4.0256</td>
<td>2.84e-05</td>
</tr>
<tr>
<td>9</td>
<td>12.65</td>
<td>3.91</td>
<td>-0.6492</td>
<td>0.7419</td>
<td>-0.7406</td>
<td>0.7705</td>
</tr>
<tr>
<td>10</td>
<td>8.23</td>
<td>2.33</td>
<td>1.7812</td>
<td>0.0374</td>
<td>0.4027</td>
<td>0.3436</td>
</tr>
<tr>
<td>11</td>
<td>24.32</td>
<td>7.04</td>
<td>-0.9991</td>
<td>0.8411</td>
<td>-1.2176</td>
<td>0.8883</td>
</tr>
<tr>
<td>12</td>
<td>6.23</td>
<td>1.36</td>
<td>1.211</td>
<td>0.1129</td>
<td>0.9812</td>
<td>0.1632</td>
</tr>
</tbody>
</table>

The first column is the run number, the second and third columns are the wait and deadlock means, and the last four columns show the deadlock and wait Z statistic and significance levels, respectively.

Out of the 12 test situations, $H_0$ was rejected in 4 of the runs and accepted in 8 runs. The shaded area shown in Fig. 10 shows the 4 runs where there was a significant decrease in the number of deadlocks.
The shaded area in Fig. 11 shows the 3 runs where there was a significant decrease in the number of waits.

The combined results show that only in runs 2, 3 and 8 there is both a significant decrease in the number of deadlocks and the number of waits after EAGLE is introduced. Further examination of the results indicates that DDA was not associated with a significant decrease in the number of deadlocks when MEAN DEADLOCKS was greater than 1, and, was not associated with a significant decrease in the number of waits when MEAN WAITS was above 3.08. Runs 2, 3 and 8 all have a significantly decreased number of deadlocks and waits associated with the use of DDA. The number of deadlocks is significantly decreased with confidence intervals in excess of 99.998%, and the number of waits is significantly decreased with confidence intervals in excess of 99.994%. DDA reduces the number of deadlocks and waits, compared with DLD, in low conflict situations, typified by a mean number of deadlocks below 1 and a mean number of waits below 3.08. See [8] for a fuller discussion of the simulation run results for various combinations of transactions, parameters and resources.

There are two possible reasons why DDA is effective in low conflict situations (characterised by a mean number of deadlocks below 1):

1. Multiple deadlock situations imply that there are multiple event sequences that can give rise to deadlock. Each of these event sequences is represented as a script in the script base. The larger the number of scripts, the higher the likelihood of a match occurring between an observed event sequence and a stored script, and hence the higher the likelihood of an OBJECT. An OBJECT ceiling is set to limit the number of sequential OBJECTs occurring.
It could therefore be possible that EAGLE becomes less effective in high conflict situations. Too many situations are identified where deadlock seems imminent, which results in multiple OBJECTs and consequently multiple ceilings. The result is that EAGLE is unable to effectively decrease deadlocks.

2. Another possible reason why EAGLE is ineffective when the number of deadlocks is more than 1, may be linked to the number of script patterns required to represent the multiple deadlock sequences. In high conflict situations more combinations of transaction processing can result in deadlock, and multiple deadlocks can occur within the processing of a single batch of transactions. So the number of scripts required to represent the multiple deadlock situations increases, as does the likelihood of an unfolding event sequence not matching any of the stored scripts.

In order to determine whether either of these factors affects the performance of DDA, more tests need to be run where parameters such as the ceiling level, activation level, and clean level are monitored.

8. Conclusion

Script-based knowledge representation has proved to be appropriate for treatment of potential deadlock situations based on learning from previous experience. EAGLE (DDA) significantly decreases the number of deadlocks and the number of waits in low conflict situations. However, DDA (currently) does not significantly decrease the number of deadlocks or the number of waits in high conflict situations. Further work is needed in dealing with high conflict situations, especially in terms of the matching techniques that are used. The higher the conflict situation, the more complex the matching process becomes. Future work needs to consider the use of other “expert” techniques within the lock manager, such as neural networks.

References


### 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/Analyzer 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.