Jānis Bārzdiņš (Ed.)

Databases and Information Systems

Proceedings of the
Third International Baltic Workshop
Rīga, April 15-17, 1998

Volume 1

Third International Baltic Workshop on

Databases and Information Systems

organised by

University of Latvia
Riga Information Technology Institute
Riga Technical University
Latvian Academic Library

sponsored by

a/s DATI (sponsor general)
Baltic Fund of VLDB Endowment
Soros Foundation Latvia
Latvian Science Council
SIA Datorikas instituts
Institute of Mathematics and Computer Science, University of Latvia

supported by

Information Technology Committee, Baltic Council of Ministers
European Association of the National Computer Societies (CEPIS)
Latvian National Organisation of automatics
Information Systems Audit and Control Association, Latvia Chapter

Jānis Bārzdiņš (Red.)

Datu bāzes un informatīvās sistēmas

Trešā starptautiskā Baltijas konference
Rīga, 1998. gada 15.-17. aprīlis

Rakstu krājums

1. sējums

Rīga 1998

Jānis Bārzdiņš (Ed.)

Databases and Information Systems

Proceedings of the
Third International Baltic Workshop
Rīga, April 15-17, 1998

Volume 1

Rīga 1998

UDK - p 681.5 (063)
Da 810

Jānis Bārzdiņš (Ed.)

Databases and Information Systems
Preprinted Proceedings of the
Third International Baltic Workshop
Rīga, Latvia, April 15-17, 1998
Volume 1

Publishing:

Institute of Mathematics and Informatics,
University of Latvia
29, Raiņa Blvd.
Rīga, LV-1459, Latvia
Phone: +371 722 4730
Fax: +371 782 0153
E-mail: imcs@mii.lu.lv

Latvian Academic Library
10, Rūpniecības Str.
Rīga, LV-1235,
Latvia
Phone: +371 710 6206
Fax: +371 710 6202
E-mail: acadlib@lib.acadlib.lv

Typesetting: Camera-ready by authors
Printing: Latvian Academic Library

ISBN 9984-538-27-3

Advisory Committee

Janis Bubenko, jr.
Arne Sølvberg

Programme Chair

Janis Barzdins

Organising Chair

Juris Borzovs

Programme Committee

Alfs Berztiss
Janis Bicevskis
Sjaak Brinkkemper
Albertas Caplinskas
Klaus R.Dittrich
Hans-Dieter Ehrich
Janis Grundspenkis
Hele-Mai Haav
Leonid Kalinichenko
Vjatcheslav Katkov
Vasyl Kostyrko
Pericles Loucopoulos
Mihhail Matskin
Julie A.McCann
Edward F.Miller, jr.
Boriss Mishnevs
Jorgen Fischer Nilsson
Algirdas Pakstas
Bronius Paradauskas
Jaan Penjam
Jaan Poial
Keng Siau
Kazimiers Subieta
Eugenijus Telesius
Janis Tenteris
Jaak Tepandi
Bernhard Thalheim
Enn Tyugu
Naoki Yonezaki
Benkt Wangler

Additional Referees

G.Barzdins
I.Dimitromanolaki
A.Georgalas
N.Georgalas
A.Geppert
T.Grotehen
S.Hakkarainen
D.Jonscher
A.Kalnins
V.Khoo
P.Louridas
L.Memuraite
T.Meyer
K.Podnieks
A.Vavouras
N.Yoshiura

Organising Committee

Maris Alberts
Rihards Balodis
Janis Bicevskis
Janis Grundspenkis
Mara Jakobsone
Askolds Kalis
Ahto Kalja
Edvins Karnitis
Marite Kirikova
Saulius Maskeliunas
Tarvi Martens
Bruno Martuzans
Juris Mikelsons
Janis Osis
Uldis Sukovskis
Maris Treimanis
Olegas Vasilecas
Juris Viksna
Maris Vitins
Lolita Zeltkalne

Preface

It was indeed a great idea of Janis Bubenko, jr. and Arne Sølvberg to organise Baltic Workshops on databases and information systems. These workshops have become a real incentive for development of regional research and boost of international cooperation.

The first workshop was held in Trakai, Lithuania in 1994, the next was in Tallinn, Estonia in 1996, and now you are welcome to Riga to the Third International Baltic Workshop DB&IS'98.

The DB&IS'98 is organised back-to-back with the Second International Exhibition and Conference on Information Technologies and Telecommunications. Having separate programme and organising committees, DB&IS'98 and IT&T'98 shared a joint plenary session and some informal events. IT&T'98 traditionally is more governmentally and commercially oriented, allowing DB&IS'98 strongly concentrate on academic issues. Consequently, most of the papers that are not so research oriented and refer more to the national information infrastructure and submitted to the IT&T'98. As a result, DB&IS'98 has become more research oriented, and, to our mind, it is welcome. Back-to-back events allow visitors from other countries to get a wide insight into IT issues in Baltics during one visit.

The International Programme Committee had representatives from 16 countries. They received 46 submissions for main workshop and 6 submissions for the Doctoral Consortium from 17 countries. Each workshop paper was reviewed by three referees. As a result, 39 papers were accepted. The accepted papers as well as invited speakers contributions are included in the workshop proceedings.

The papers present original results in a wide variety of topics, such as data base performance aspects, information retrieval, security, testing, concurrency, metamodels, application generation, object-oriented approaches and knowledge representation. The invited talks review new trends in software engineering, education, software testing tools, quantum computation.

The Workshop is preceded by one-day Doctoral Consortium. The workshop's full programme as well as back-to-back exhibition and IT&T conference offer an excellent chance for researchers and practitioners to share their ideas and experiences on databases and information systems and their interrelations.

We express our warmest thanks to all authors who contributed to the workshop. We are grateful to members of the Programme Committee and the additional referees for carefully reviewing the submissions. We'd also like to thank the organising team, especially Juris Vīksna for big technical work.

We express our very special thanks and deep gratitude to our sponsor general that covered almost half of the workshop budget - a/s DATI, the biggest software company in East Europe. Donations by Baltic Fund of VLDB Endowment, Soros Foundation Latvia, Latvian Science Council, SIA Datorikas Instituts, and Institute of Mathematics and Computer Science, University of Latvia are also acknowledged. Our thanks go also to Latvian Fund for Education and its Special Programme for Education, Research and Culture that supported the workshop a lot.

Last, but not least, we thank all participants, who really made the workshop.

Juris Borzovs
Organising Chair

Jānis Bārzdiņš
Programme Chair

Contents

Challenges in Information Systems Engineering

Janis A. Bubenko jr
Kungl Tekniska Högskolan and Stockholm University
Department of Computer and Systems Science
Electrum 230, S-164 40, Kista, Sweden
janis@dsv.su.se

Abstract

During my professional life in the field of Information Systems Engineering, my perception of what are the crucial problems of the field have gradually changed. In the sixties and seventies my interest was mainly focused on engineering problems of limited scope which could be expressed and handled formally. Since the eighties I have, by participation in a number of industrial projects where methodologies for information and business engineering were exposed in tough industrial settings, come to realise that the informal parts of business and information system development perhaps are even more significant in order to guarantee the success of a development project. The informal issues are certainly more difficult to understand, to appreciate, to do research in, and to teach.

In this talk I will examine the area linking business engineering and information systems engineering. The growing competitive importance of information systems for running the business, indicates the need of both better understanding the business, and deriving information system requirements from this understanding. A good information system can be a competitive asset of the business, while a system not conforming to the business needs can severely hinder running of the business. This is why we now can observe the beginning of a paradigm shift from the more technology oriented Information Systems Engineering approaches to frameworks which focus on modelling of "the enterprise", "the business", including its business goals and rules. The importance of establishing explicit links between the development of business objectives and strategies and information system development, has been recognised.

This paradigm shift imposes a number of new challenges for an organisation. Gone are the days when information systems were designed and delivered by the IT-department with a minimum of requirements acquisition and consultations with the users and stakeholders. Instead we are in an era where active user participation in the system development life-cycle is considered natural as well as necessary. But user participation is not without problems. Methodologies claiming active user involvement have often not thought about that users are not used to and have difficulties in understanding detailed structured and formal descriptions (as well as the need for them) which, on the other

hand, are needed for IS design. Some of the challenges, therefore, are related to balancing the informal contributions and way of working of the participating users versus the more formal design products needed and generated by the IS designers. Challenges related to this problem concern the way we are managing the process of requirements acquisition and user involvement. Our experiences also show the need for re-examining what kind of special and new skills are needed in the business and requirements engineering phases of a system development project.

Contents of an Information Systems Engineering Education

Arne Sølvberg
Information Systems Group
Faculty of Physics, Informatics and Mathematics
The Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

Abstract

When we talk about computers, software and information systems we do not distinguish clearly between the words science and technology. We talk about computer science and software engineering as were they the same. This is unfortunate. Many of us get into difficulties when asked to define what we mean by these terms. Our confusion becomes even more embarrassing when we try to explain words like information technology, information science, and requirements engineering. This carries over to disagreements on what the content of our teaching should be, and to which faculty we should be associated. The words also carry political content when research funds are distributed. If we are a science we have to compete with mathematics and physics. If we are technology we must associate ourselves with industry. We would be better off if we could agree on what is characteristic to our discipline. What follows is intended as a contribution to this end.

Some differences between science and technology

There is a profound difference between science and technology. In natural science the aim is to describe the natural phenomena. The descriptions may change and improve over time, but the laws that govern the object of study are always the same. Natural science is analytical because its object is given and the goal is to give a general description. The scientist is an intelligent observer, and nothing more. If we use the word science to characterise our discipline we put ourselves in the role of the observer whose task it is to describe the phenomenon of information systems.

Technology is very different. It creates phenomena that have never existed before. This is achieved either through intuition or through trial and error. Good technology is logical and systematic, but needs not be theoretical. Technology is synthetic and constructive.

There are two major approaches to formulating technical problems: (1) What is it possible for me to make that is new? (2) What is it that I have made? These two questions are often circular.

Technology studies itself and what it has brought forward, and what the future possibilities are, and this changes continuously. Technology seeks to solve tasks that itself has created. Everything that takes place is of course constrained by the laws that govern the relationships among the relevant phenomena, be they natural laws, mathematical laws, or physiological, psychological and social "laws". So close contact between technology and science supports the improvement of the technology.

Information Systems Engineering and classical technology

It seems to be quite safe to propose that our discipline is more technological than scientific. Consequently it should be treated as a technological discipline in teaching as well as in research. It is nevertheless different from the classical technologies of, e.g., bridge building and process control. Classical technologies are based on natural sciences. Ours is not.

The classical definitions of the concepts of technique and technology are: (1) Technique is the physical means that we use to satisfy wishes, and (2) Technology is the discipline of the means and of the methods for using the means.

The basic definitions are constrained to "physical means". Information systems use physical means to handle information, be they disk drives, data transmission lines, computers, displays and paper, to mention a few. Even the very abstract phenomenon of software may be seen as physical means, because software translates to electrical currents in a computer, which are physical phenomena, and because software manifests itself as physical interactions with its environment, e.g., control signals to process regulators, and symbols on computer screens.

In everyday language the word technique is frequently used also for non-physical means, e.g., reasoning technique, proof technique and negotiation technique. We would nevertheless hesitate to talk about negotiation technology to mean the discipline of negotiation techniques. Physical means are too closely associated to the concept of technology to permit such use of the word. So we would reserve negotiation technology to mean software systems for the support of negotiations, e.g., CSCW-systems.

Components of Information Systems Engineering Education

This panel is concerned about the contents of an appropriate education for information systems engineers, that is, the contents of an appropriate knowledge profile for people who build and re-build information systems. The concept of information systems is fairly well understood, even if different disciplines argue on the relative importance of its many features. Information systems are usually understood to be computer assisted systems for collecting, storing, transforming and distributing information that is to serve as basis for the operation of a larger system in which the information system is contained. The "larger system" is usually assumed to be populated by humans, who interact with the computerised parts of the information system, which are seen to be the technical "means". The human "information processors" are sometimes seen to belong to the information system, but may also be viewed to be external to the information system. The concept of "information" is generally understood to comprise data plus interpretation.

A characteristic feature of information systems is their abstract nature. Classical engineering is concerned with natural phenomena which may be touched, felt and viewed. The models of classical engineering are closely related to "touchables" and may often be visualised in easily recognisable forms. Information systems are different. Information systems engineers must consequently be trained to work with abstractions in a more profound way than their colleagues in other branches of technology. Basic mathematical training is an important part of a curriculum, because this improves the abstraction skills.

Desirable knowledge components in a curriculum must comprise training in abstract thinking and modelling, knowledge about computers, about "information processing man", about the relationship between humans and computers, about the "larger system" and its relationship to the information system, and about the techniques for building, maintaining and operating information systems. A central question in the formation of an information systems engineering curriculum is which of these relationships to emphasise the most. Is the relationship to the computer more important than the relationship to the "larger system"? What is the "larger system"? Is it manufacturing, or is it service industries like banking or insurance, or may be all of them? Is it organisation in general? If information systems engineering shall be considered a discipline in its own right there must be enough commonalties among information systems in the various "larger systems". Which are these? Certainly the computer is a common denominator.

Five views of computers in curricula

Computers provide the core technical means to realise information systems. But the computers provide the most important enabling techniques in many other fields as well. The various disciplines have their own views of which aspects of computer are the most important. There are (at least) five different views that influence the formation of curricula. They are: (1) the computer as a phenomenon, (2) the computer as calculator (3) the computer as a system component, (4) the computer as a tool for human activities, and (5) computerised information as a phenomenon.

(1) The computer as phenomenon view requires a knowledge profile which is sufficient for understanding the inner workings of the computer itself. The curriculum comprises electronics, VLSI, digital technique, computer architecture, operating systems, telecommunication, software engineering, database management, graphics, and data centre operations.

(2) The computer as calculator view sees computational power as the most important feature. Mathematics, visualisation and simulation belong to the core subjects. The design of powerful parallel computers to crunch larger and larger data volumes is an area of central concern.

(3) The computer as component view is concerned with computers embedded in technical systems. Control engineering is a major field for this view. The desired knowledge profile is more concerned with the workings of the encompassing technical system than with the computer, which is mainly seen as a supporting systems component.

(4) The computer as a tool for human activities view is concerned with the computer as a tool for the information processing human being, and comprises individual PC-tools as well as

administrative data processing. Management information systems is within the bounds of this view.

(5) The computerised information as a phenomenon view is concerned with the concept of information, with information content, semantics and pragmatics as central parts. Connections to linguistics, multi-media and entertainment are important. The computer is seen mostly as an information processing machine. Its capacity for symbol manipulation and transmission is at the centre of interest, and the relationship between symbols and people (pragmatics) is seen as the most interesting challenge.

Contents of Information Systems Engineering education

Technological education always has an eye to the market place. So also for Information Systems Engineering. IS engineers must be able to design, build and operate information systems. Education must be fitted to this end.

Information systems engineers must be able to implement. Most of the means are based on computers. The IS engineer must therefore have satisfactory knowledge of the workings of computers and telecommunication systems. So the computer as a phenomenon is an obvious part of the curriculum. So is project management.

A high capacity for abstract thinking is a must. We should train our students accordingly. Basic mathematical training is a good vehicle to this end. Training in formal IS modelling is an integral part of the curriculum.

Information systems comprise the interplay between computers and humans. The computer as a tool for human activities is central to information systems engineers. Human-computer interfaces are important as well as subjects concerned with human behaviour, e.g., psychology, sociology, technology transfer and management. Teaching is important because of the very large investments in user training that are associated with information systems operations.

The interplay between computers and information content is increasingly important, e.g., in entertainment. Computerised information as a phenomenon becomes a more important educational subject in the future than what it has been in the past.

Information systems integrate computer resources and human resources in order to support information processing in some "larger" system. The curriculum can not cover every kind of "larger" system and should mostly be concerned with general techniques in organisation and management.

The weight given to the different aspects will differ over time, and depend on which markets one wishes to educate for. An engineering approach to education should be followed, with much emphasis on learning by doing. Individual projects and group projects should play a large role in the curriculum, and projects should rather be concerned with making information systems than to study them.

Software Testing Tools: Planning for the Millennium

Edward Miller

Software Research, Inc.

This talk by a senior software technologist and someone who has been building and selling software test tools for over two decades, will focus on test tools currently on the market and on the direction test tools appear to be developing.

The talk will survey current test tool features, methods, strong and weak points, availability and other factors, and will also include assessment of "best use" in a typical software development scenarios.

Quantum Computers and Computer Science

Rūsiņš Freivalds

Department of Computer Science
University of Latvia
Raiņa bulvāris 29
LV-1459, Rīga, Latvia
rusins@cclu.lv

Abstract

In 1982 R. Feynman predicted the possibility to use effects of quantum mechanics for building future computers. Quantum circuits contradicting the classical physics have been designed and even implemented. Quantum algorithms allow computation of discrete logarithms and factorization of integers in polynomial time. Hence cryptography will be changed radically.

Information is physical, so is computation

We are not used to think of computation in physical terms. We consider the inputs and the outputs as abstract quantities. However before any computation the inputs are encoded in a physical system. This can take several radically different forms depending on the computing device: voltage potentials at the gates of a transistor on a silicon microchip, beads on the rods of an abacus, nerve impulses on the synapse of a neuron, etc. The computation itself consists of a set of instructions (referred to as an *algorithm* carrieed out by means of a physical process. Completion of the algorithm yields a result that can be reinterpreted in abstract terms: we observe the physical system (for instance, measure the voltage) and conclude that the result is , say,5.

Theory of computation has always been a theoretical field completely detached from physics. Nevertheless the pioneers such as Turing, Church, Post and Gödel were able to capture the correct physical picture by their intuition alone. Other people falsely assumed that Theory of Computation was based on purely abstract foundations. The famous Church-Turing thesis "Every function computable by algorithms is a recursive function" has survived six decades and most people believe that this thesis will survive in the future as well, but to be more safe we are to add "until the algorithms are performed by devices disobeying the laws of the classical physics".

In this century physicists have discussed the foundations of their theories much more than, say, computer scientists. No wonder that the Church-Turing thesis has been

under permanent physical attack for a long time. For an early discussion of this topic, see M. Davis [5, p. 11] :

"... how can we ever exclude the possibility of our presented, some day (perhaps by some extraterrestrial visitors), with a (perhaps extremely complex) device or "oracle" that "computes" a non computable function?"

A main theme of R. Landauer's work has been the connections between computation and physics; see, for example, his 1967 article [10] "*Wanted: a physically possible theory of physics*." As D. Deutsch puts it more recently [6, p. 101] :

"The reason why we find it possible to construct, say, electronic calculators, and indeed why we can perform mental arithmetic, cannot be found in mathematics or logic. The reason is that the laws of physics 'happen to' permit the existence of physical models for the operations of arithmetic such as addition, subtraction and multiplication. If they did not, these familiar operations would be non-computable functions. We might still know of them and invoke them in mathematical proofs (which would presumably be called 'non constructive') but we could not perform them."

A turning point in this discussion came in 1982. Nobel prize winner physicist Richard Feynman asked in [7] what effects can have the principles of quantum mechanics, especially, the *principle of superposition* on computation. He gave arguments showing that it might be computationally expensive to simulate quantum mechanics on classical computers. This observation immediately leads to a conjecture predicting enormous advatages to quantum computers versus classical ones. R. Feynman left open even the crucial question whether or not quantum computers can compute any functions non-computable by classical (deterministic) computers. P. Benioff [2] gave early arguments on Feynman's problem, and later D.Deutsch [6] introduced the commonly used notion of the quantum Turing machine and proved that quantum Turing machines compute exactly the same recursive functions as ordinary deterministic Turing machines do.

On the other hand microminiaturization of computer chips has lead to a situation when quantum mechanical effects can no more longer be neglected.

Hilbert space quantum mechanics

Quantization has been introduced by Max Planck in 1900 [11]. Planck assumed a *discretization* of energy. That was a bold step in a time of the predominant continuum models of classical mechanics.

Mathematically all quantum mechanical entities are represented by objects of Hilbert spaces. A *Hilbert space* is a linear vector space over the field of complex numbers (with vector addition and scalar multiplication), together with a complex function for the scalar product. It seems that there have never been satisfactory explanations why complex numbers are used (but not, say, quaternions or more exotic number fields). The simplest

explanation might be that physicists have developed such a theory, and it works, while theories based on real numbers only do not explain all the experiments.

Quantum mechanics differs from the classical physics very much. It suffices to mention *Heisenberg's uncertainty principle* asserting that one cannot measure both the position and the impulse of a particle simultaneously precisely. There is a certain trade-off between the accuracy of the two measurements. Another well-known distinction of quantum mechanics from the classical physics is the impossibility to measure any object without changing the object.

Quantum information theory

The fundamental atom of information is the quantum bit, henceforth abbreviated by the term *qbit*. Classical information theory is based on the classical bit as fundamental atom. This classical bit, henceforth called *cbit*, is in one of two classical states *t* often interpreted as "true") and *f* (often interpreted as "false"). In quantum information theory the most elementary unit of information is the *quantum bit,* henceforth called *qbit*. To explain it, we first discuss a *probabilistic* counterpart of the classical bit, which we call here *pbit* . It can be *t* with a probability α and *f* with probability β, where $\alpha+\beta=1$. A *qbit* is very much like to *pbit* with the following distinction. For a *qbit* α and β are not real but complex numbers with the property $\|\alpha\|^2+\|\beta\|^2=1$.

The classical and the quantum mechanical concepts of information differ from each other in several aspects. A classical bit remains unchanged, no matter by what methods it is inferred. It obeys classical logic. It can be copied.

A quantum bit may appear different, depending on the method by which it is inferred. Every measurement destroys the quantum bit transforming it into one of the classical bits with probability equal to square of the module of the amplitude. Quantum bits cannot be copied or "cloned". Classical tautologies are not necessarily satisfied in quantum information theory. Quantum bits obey quantum logic.

Every computation done on qbits is performed by means of unitary operators. One of the simplest properties of these operators shows that such a computation is reversible. The result always determines the input uniquely. It may seem to be a very strong limitation for such computations. Luckily this is not so. It is possible to embed any irreversible computation in an appropriate environment which makes it reversible. For instance, the computing agent could keep the inputs of previous calculations in successive order.

The following features of quantum computers are important (but far from the only characteristic features of them):

- Input, output, program and memory are represented by qbits.
- Any computation (step) can be represented by a unitary transformation of the computer as a whole.

- Any computation is reversible. Because of the unitarity of the quantum evolution operator, a deterministic computation can be performed by a quantum computer if and only if it is reversible.
- Unless classical, qbits are context-dependent. That is, their value may depend on the method by which they have been inferred, and on the co-measured qbits.
- Measurements may be carried out on any qbit at any stage of the computation. But, unless classical, a qbit cannot be measured by a single experiment with arbitrary accuracy. The computation process and the measurement have to be repeated in order to obtain sufficient statistics.
- Quantum parallelism: during a computation (step), a quantum computer proceeds down all coherent paths at once. If managed properly, this may give rise to speedups.

In order to appreciate quantum computation, one should make proper use of the above features - quantum parallelism, unerasability of information, non-copying, context-dependence and impossibility to directly measure the atoms of quantum information, the qbits, related to quantum indeterminism.

Quantum bit teleportation

Quantum information cannot be measured in general but it can be *teleported* from one place to another. *Teleportation* is a term used by science fiction authors to describe the possibility to send a person via telephone or a similar communication system destroying the original and building an identical copy for the receiver. This would allow (by opinion of the authors) a travel at a speed faster than light. Assume
Alice wishes to send a qbit to Bob. Additionally, Alice and Bob have two identical qbits of information (however these qbits are absolutely unrelated to the information being transmited). It was shown in [3] that Alice can send to Bob only two classical bits of information, and this suffices for Bob to restore the original qbit. This result contradicts the classical information theory developed by C.Shannon since the amplitudes in the qbit being transmitted can be arbitrary complex numbers (not only 0 or 1).

Quantum information theory (*in statu nascendi* only) is a modification of Shannon's theory like Einstein's relativistic mechanics is a generalization and modification of Newtonian mechanics. For instance, it is proved that *k* qbits can transmit at most *k* classical bits of information. However this does not rule out the possibility that *k* classical bits are encoded into 1 quantum bit of information, and it is up to the receiver which classical bit to learn in a suitable process of measurement[15].

Quantum bit teleportation was the first but not the last example of quantum algorithms which clearly contradict classical physics. This is why the scientific world was so surprised when in 1997 a team of physicists in University of Inbruck implemented

quantum bit teleportation in a practically working circuit. This result of the experiment was reported in *Nature* (December 11, 1997) and later reported in practically all the media.

Unexpected quantum algorithms

Public-key cryptography is considered a modern branch of science using results of the Theory of Complexity of Computation for data protection. The methods of public-key cryptography are based on the assumption that nobody will be able to compute discrete logarithms or to factorize integers reasonably quickly. However Peter Shor [13] invented surprising polynomial-time quantum algorithms for computation of discrete logarithms and for factorization of integers thus challenging all the modern data protection systems. Now it is clear that building a quantum computer implies building a code-breaking machine as well.

It has been shown that there are other practically important problems that can be solved faster on a quantum computer than on any classical computer. An example of this is the quantum database search algorithm of L. Grover [9].

Where the advantages of quantum algorithms over deterministic ones lie? In contrast with the surprising efficiency of the above-mentioned quantum algorithms, there have not been *proved* time or memory space advantages of quantum algorithms over deterministic ones. The only theorems on advantages are proved for restricted algorithm classes, e.g. for finite automata. We can be proud in the University of Latvia that one of the most spectacular results in this area is ours we have proved the very first result on provable advantages of quantum algorithms over probabilistic ones. Namely, the language "the length of the input word is a multiple of p " can be recognized by a quantum 1-way finite automaton with $O(\log p)$ states while every deterministic and even probabilistic 1-way finite automaton needs p states [1].

Quantum cryptography

When quantum computers are built, the end of the contemporary public-key cryptography comes. Luckily, quantum cryptography is already at sight. Strangely enough, the pioneering work of S. Wiesner [16] was done as early as in late 1960's (however it remained unpublished till 1983). The security of these protocols would not be compromised even if the cheater had unlimited computing power, but in essentially all the cases it has not yet been ruled out that still more sophisticated use of quantum physics might defeat them.

Since then many quantum cryptographical protocols have been designed and even implemented. For instance, G. Brassard et al. [4] have described a complete protocol based on the transmission of polarised photons. They have shown that under the laws of

quantum physics, this protocol cannot be cheated by either party except with exponentially small probability.

References

[1] AMBAINIS, A., AND FREIVALDS, R. 1-way quantum finite automata: strengths, weaknesses and generalizations. e-print *http://xxx.lanl.gov/abs/quant-ph/9802062*.

[2] BENIOFF, P.A. Quantum mechanical Hamiltonian models of Turing machines. *Journal of Statistical Physics 29*, 3 (1982), 515-546.

[3] BENNETT, C.H., BRASSARD, G., CREPEAU, C., JOZSA, R., PERES, A., AND WOOTTERS, W.K. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters 70*, 13 (1993), 1895-1899.

[4] BRASSARD, G., CREPEAU, C., JOZSA, R., LANGLOIS, D. A quantum bit commitment scheme provably unbreakable by both parties. *Proc. FOCS'93* (1993), 362-371.

[5] DAVIS, M. *Computability and Unsolvability*. McGraw-Hill, New York, 1958.

[6] DEUTSCH, D. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Royal Society London A 400* (1985), 97-119.

[7] FEYNMAN, R.P. Simulating physics with computers. *Internat. J. Theoretical Physics 21* (1982), 467-488.

[8] FEYNMAN, R.P. Quantum mechanical computers. *Optics News 11* (February 1985), 11-20.

[9] GROVER, L. A fast quantum mechanical algorithm for database search. *Proc. STOC'96* (1996), 212-219. e-print *http://xxx.lanl.gov/abs/quant-ph/9605043*.

[10] LANDAUER, R. Wanted: a physically possible theory of physics. *IEEE Spectrum 4* 1967), 105-109.

[11] PLANCK, M. Ueber eine Verbesserung der Wienschen Spectralgleichung. *Verhandlungen der deutschen physikalischen Gesellschaft 2* (1900), 202.

[12] ROGERS, JR., H. *Theory of Recursive Functions and Effective Computability*. MacGraw-Hill, New York, 1967.

[13] SHOR, P.W. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium of on Foundations of Computer*

Science, Santa Fe, NM, Nov. 20-22, 1994 (November 1994), IEEE Computer Society Press.
e-print *http://xxx.lanl.gov/abs/quant-ph/9508027*.

[14] SVOZIL, K. Quantum algorithmic information theory. *Journal of Universal Computer Science 2*, 311-346 (1996). e-print *http://tph.tuwien.ac.at/~svozil/publ/qait.tex*.

[15] TA-SHMA, A., AND VAZIRANI, U. *How much information can one touch in k qbits?*
e-print *http://www.brics.dk/~salvail/aqip/abstlist.html*.

[16] WIESNER, S. Conjugate coding. *Sigact News 15*, 1, 1983, 78 - 88; original manuscript written circa 1970.

The COIL Project: A Common Object Interconnection Language to Support Database Integration and Evolution[1]

William J. McIver, Jr., Roger King, Richard M. Osborne, and Christian Och

Database Research Laboratory
Department of Computer Science
University of Colorado
Boulder, Colorado 80309-0430

e-mail: mciver@cs.colorado.edu, roger@cs.colorado.edu, rick@cs.colorado.edu

Abstract

The COIL Project is an effort to design a module interconnection language specifically for managing database integration and evolution. This project is a significant part of a larger effort to build a database integration and evolution environment called Sanctuary. COIL will make a significant contribution to module interconnection language research by providing interface symmetry, application-level module composition, and syntactic analysis capabilities for systems constructed with the Object Management Group's Common Object Request Broker Technology (CORBA). Since CORBA is the state-of-the-art in distributed, object-based technologies, and since it has been adopted by over 800 software vendors, software developers and end users, this research will have immediate and high impact. This paper presents a discussion of the motivation, background, and fundamental module interconnection language issues and goals for the COIL project.

1. Motivation

Modern persistent applications typically run on top of several (if not hundreds) of distributed, heterogeneous databases (e.g., legacy databases and modern relational and object-oriented databases, as well as ad hoc systems). Integrating, evolving and maintaining such a set of databases is a huge problem which has traditionally been attacked in one of two ways: point-to-point solutions and complete integration. Point-to-point solutions consist of building a new set of wrappers or applications each time a new database needs to be accessed. Maintaining data consistency between a large set of heterogeneous databases in such an ad hoc manner is generally doomed to failure. Fully integrated systems solve many of the consistency and integrity issues associated with point solutions due to the fact that the database schemas are typically translated into a common model so that they may be merged into a common view. Unfortunately, for an application involving a large number of rapidly evolving databases (with new ones being frequently added), this solution is generally intractable. There are several systems that have either taken this approach, or at least require a substantial amount of schema integration; these include

1. This research is being funded under DARPA order B126 (Rome AF contract F30602-94-C-0253), under DARPA/NASA contract NAG2-862, and under NSF award IRI-9632595.

Pegasus [30], UniSQL [14], multidatabase systems (e.g., InterBase [20]), and Federated databases [10].

Sanctuary is a database integration and evolution environment for supporting exactly such large, heterogeneous environments [15, 16][1]. COIL (Common Object Interconnection Language) is a language being designed for use within Sanctuary that: allows the specification of points of interconnection between objects and data transformations across those points of interconnection; and which can support application level module composition and evolution (i.e. re-composition) for a broad spectrum of data integration, migration, and evolution problems. In particular, the Sanctuary project is motivated by the need to support the data integration and evolution needs of applications that are using legacy databases and are candidates for integration with more modern database systems (e.g., relational or object-oriented). Such environments typically have several criteria for integration. First, as new databases are added, relationships between the new databases and the existing legacy systems need to be defined. Second, in many cases, the ultimate goal is to eventually migrate the legacy data to newer systems in order to take advantage of the tools provided by such systems (e.g., transaction management, concurrency, modern query languages, etc.). And third, such environments generally need to be able to keep legacy applications running without rewriting existing code.

The unique contributions of Sanctuary to database application integration are its notion of interoperation through a notion we call explicit heterogeneity; application-level integration of existing data sources via dynamic interface discovery mechanisms; a technique for describing interconnections between existing data sources; and a mechanism for generating relatively lightweight constructs, called interoperation applets, which provide interoperability between applications and their data sources. These data sources could be legacy database management systems, object databases, or flat files. Furthermore, Sanctuary allows interoperation applets to evolve over time, as an applications and their data source relationships evolve.

Sanctuary has been applied successfully to a variety of data integration and evolution problems. It has been used to perform data migration from Unidata/VMARK CODASYL databases to O_2 object-oriented databases; support the building of new object-oriented applications atop old CODASYL applications; support the interoperation of the storage system of the WinWin application from USC and a CORBA-based system; and, support the integration of the object-oriented Catalyst software engineering environment with ODBC-compliant DBMSs [15, 16].

2. Background: The Sanctuary/COIL Architecture and Implementation

The Sanctuary approach is based on loosely coupling databases or other sources of persistent data into what might be thought of as lightweight "alliances" tailored for a specific application (or set of applications). At the heart of the Sanctuary approach is a notion called *explicit heterogeneity*. The idea is that component database systems are not forcibly integrated in their entirety into a massive, centralized view; and, database developers, tool developers, and users are not restricted to using an environment where constraints are defined and updates are specified only through this

1. Sanctuary was previously named Sybil.

centralized view. Rather, the component systems remain largely autonomous, and much of their interconnectivity is defined locally, between pairs or small sets of component databases. This approach avoids the ad hoc approach of point-to-point solutions. At the same time it does not involve the huge start-up costs usually associated with fully integrating a system. The design-time environment supports the creation of these alliances in the form of *interoperation* and *data migration applets*, each defining an interoperation or migration protocol between a specific pair of client and server interfaces. These applets can also be retrieved from the Sanctuary repository and reused, perhaps with modifications, when new interoperation and data migration problems arise. The result is a database integration and migration approach that is tractable, scalable, and evolvable. Data migration from one database to another (e.g. CODASYL to relational) is a special case of our interoperation applet mechanism, where the client application object using the applets are database servers and the applets are conduits for converting or reformatting, and transferring data from one database to another. These are called *data migration* applets.

The Sanctuary architecture, shown in *Figure 1*, is divided into three major segments: a compile-time environment, a run-time environment, and a repository. Application objects[1], which correspond to the traditional notion of applications, exist in an implicit fourth segment of the Sanctuary architecture. The compile-time segment of Sanctuary consists of tools which generate and store interoperation and data migration applets loaded by the run-time segment. The run-time segment of Sanctuary consists of services used by application objects to achieve interoperability with other application objects. Central to the entire architecture is the repository, which is used by both the compile-time and run-time segments to store and retrieve, respectively, interoperation and data migration applets.

Systems within Sanctuary's compile-time segment are used to process application interface specifications, and interoperation and data migration applet specifications. These tasks are performed, respectively, by the Sanctuary Interface Definition Language Compiler and the Sanctuary Connection Language Compiler.

Application interface specifications are used to define interfaces to application objects that will operate in the Sanctuary run-time environment. Application interface specifications are written in the Sanctuary Interface Definition Language (*S-IDL*). S-IDL is not a programming language; it is solely an interface language. S-IDL is the language used to describe the interfaces of the server application objects that client application objects in the Sanctuary run-time environment call. Viewed another way, S-IDL is used to describe the interfaces provided by the implementations of those server application objects. Implementations of application objects whose interfaces are defined in S-IDL are written in some programming language. Public operations, attributes, types, constants, and exceptions made available by these application objects are presented to the outside world in the form of files containing S-IDL expressions. S-IDL files are processed by the S-IDL Compiler. The S-IDL Compiler generates programming language-specific stubs, which are then stored in the Interface Repository segment of the Sanctuary Repository. These S-IDL stubs are then used by the Connection Language Compiler for the interface sections of the applet code it

1. The term "Application Object" is used here as defined in the Object Management Group's Object Management Architecture Reference Mode, meaning objects specific to particular commercial products or end user systems. See [22] for further details.

generates. The stubs also represent the actual interfaces called by the Proxy Service in the Sanctuary Run-time Services Environment when it invokes requests on application objects (on behalf of client application objects). The Proxy Service will be explained in greater detail below.

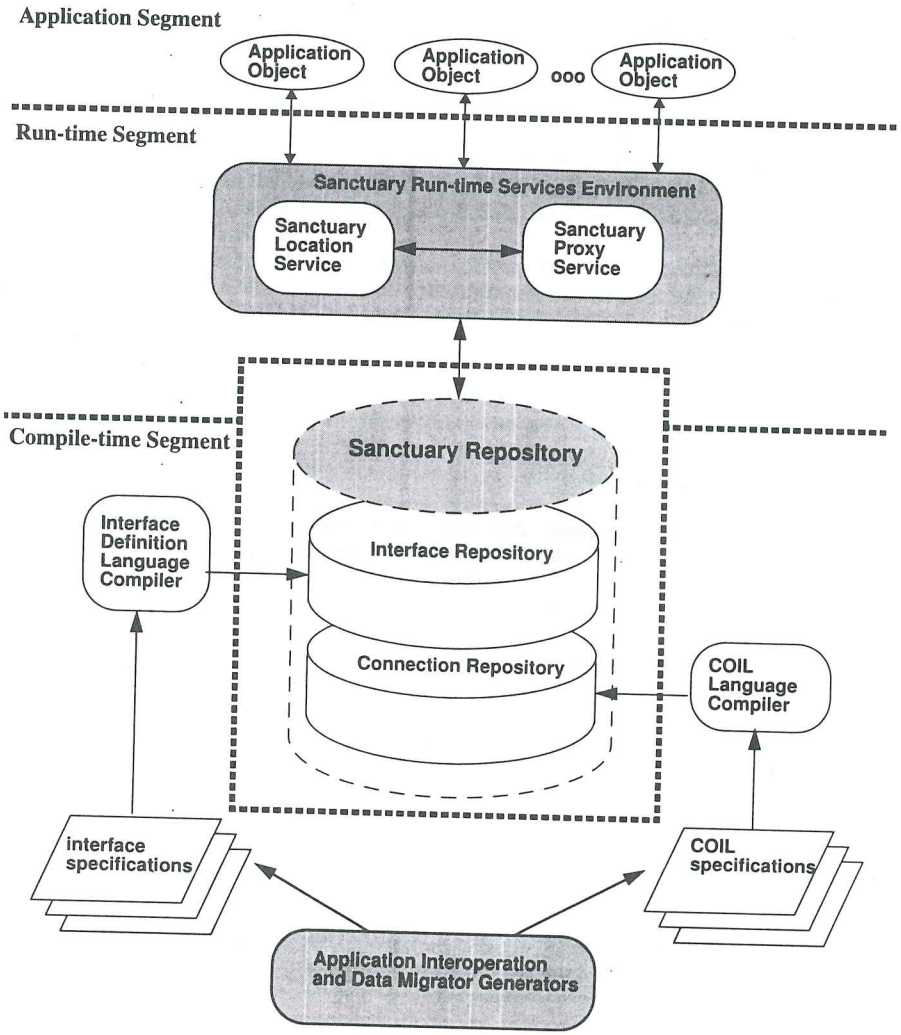

Figure 1. The Sanctuary/COIL Architecture.

S-IDL is compliant with the CORBA 2.0 IDL since application object interoperation is facilitated using CORBA facilities and services[1]. The S-IDL compiler is, therefore, currently "implemented" using an existing commercial CORBA 2.0 IDL compiler. S-IDL is expected to evolve to allow the expression of more information about application object interfaces, such as operational constraints and semantic information. To remain CORBA compliant, extensions will be implemented using special tags contained within CORBA IDL-compliant comments, which can be recognized by the S-IDL compiler and processed separately from the information required for stub generation.

The run-time segment contains Sanctuary's Run-time Services Environment. This environment is used by pairs of application objects that wish to interoperate. It allows them to locate, bind with, and run interoperation applets appropriate to the interfaces in the pair. These tasks are performed through the cooperation of two services within the Run-time Services Environment: a *Location Service* and a *Proxy Service*. The environment and its operation are depicted in *Figure 2*. Both services are implemented as a CORBA 2.0-compliant application objects, which allows them to be used by application objects running on a distributed and heterogeneous platforms. This makes the Sanctuary's run-time services virtually platform-independent.

The Location Service is used by application objects to locate other application objects that provide services they need. In the context of database integration, this will normally mean client applications locating database servers. The Location Service maintains a registry of application objects providing services and any associated interoperation applets that might be used to request services from those objects. The process involving the Location Service is shown in *Figure 2*. In step 1, a service application object, *S1*, registers its capabilities and network location with the Location Service[2]. In step 2, the Location Service is queried by a client application object, *C1*, looking for a particular service, *S1*. The Location Service in step 3 then attempts to locate an *S1* service and any associated *C1/S1* interoperation applets by performing a lookup in the Sanctuary Repository. If the desired service is located in the repository, the Location Service then, in step 4, requests the Proxy Service to load and initialize its *C1/S1* interoperation applet as a proxy object. The Proxy Service then loads the applet and initializes it in step 5. This proxy object is then "connected" with the real server application object, *S1* in step 6. Client application object *C1* can now communicate with server application object *S1* via the *C1/S1* interoperation applet running within the proxy object managed by the Proxy Service.

The Location Service is implemented using an implementation of the OMG Trading Service defined in the CORBA 2.0 Common Object Services Specification. The OMG trading object service facilitates the offering of services by application objects. Client application objects can then query a trading object in an attempt to discover instances of a desired service[3]. The CORBA IDL specification for the Location Service interface will resemble the following:

```
import "Sanctuary.idl"
interface SanctuaryLocationService {
    SanctuaryProxyObject FindMatching(in SanctuaryQueryObject query_expression);};
```

1. See [21] for details on CORBA IDL.
2. This action is called "exporting" in CORBA terminology.
3. For further information on the CORBA Trading Object Service see [21].

The Proxy Service facilitates communication between application objects via an interoperation or data migration applet. It does so by creating a proxy object for each applet used. Client and server application objects then communicate via the applet using the interface presented to them by the proxy object. The process involving the Proxy Service is also shown in *Figure 2.* Once the interoperation applet has been loaded and initialized within a proxy object (step 5), the proxy object is connected with its real server application object (step 6) and a reference to the proxy object is returned to the client (step 7). At this stage in the process, the client application object can begin sending requests to the server application object.

The proxy object in which the interoperation applet, *C1/S1*, is running presents an interface (defined previously in S-IDL) containing methods which the client can invoke on the server. Such an interface might resemble the following (in S-IDL):

```
import "Sanctuary.idl"
interface Customer : SanctuaryProxyObject {
    attribute string<12> ssn;
    attribute string<50> name;
    attribute string<80> address1;
    attribute string<80> address2;
};
```

Step 8 depicts the client, *C1*, invoking a method on server, *S1*, via the interoperation applet *C1/S1*. The interoperation applet, *C1/S1*, then receives the invocation, performs any operations or transformations on the request and passes it to the real server application object, *S1*, through a method invocation in step 9. Any results returned from the method invocation on *S1* are then passed to the interoperation applet in step 10. The applet then in step 11 performs any specified operations or transformations on the results and passes them back to the client application object, *C1*. The operations and transformation performed by the interoperation applet are previously specified in the Connection Language and compiled to produce the applet.

3. Research Issues in the COIL Project

A Sanctuary interoperation applet defines a conversion between two specific application object interfaces. Data migration applets, as discussed above, are a special type of interoperation applet. Specifications for both types of applets are to be written (or generated) in a "connection language" called COIL (Common Object Interconnection Language). Specifications in this language will be processed by a *COIL* compiler. COIL will generate program code to implement each interoperation construct specified in a Sanctuary Connection Language file. It will then store each implementation in the Connection Repository segment of the Sanctuary Repository.

Sanctuary was initially designed with a modest "connection language," called *Sconn*, for supporting the definition of unidirectional and bidirectional connections between application objects. SConn generated the applet code required to perform transformations required to make two objects interoperate. Code generated by Sconn was then to be stored in the Connection Repository segment of the Sanctuary Repository. Run-time services allow incoming requests for

interoperation applets to be mapped to the code in the Connection Repository that will perform the interoperation. Sconn was designed to operate on interface specifications defined in S-IDL. It could recognize S-IDL types, implicitly perform standard C++-style narrowing and conversions between a number of types, and support operations common arithmetic, string, relational, boolean, floating point conversion, and aggregate functions found in most imperative languages.

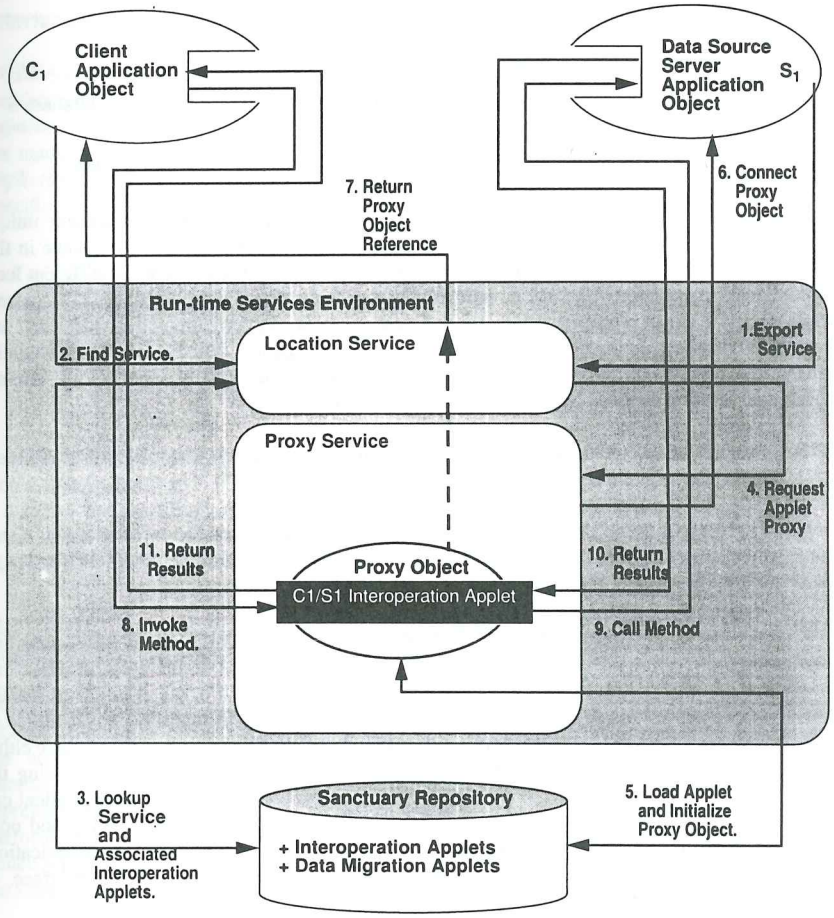

Figure 2. The Process of client/server interoperation in the Sanctuary/COIL architecture.

The COIL project will make a fundamental advancement within the interconnection area of the Sanctuary project. COIL will provide richer support for data integration in the face of software

evolution than currently provided by SConn. It will replace SConn -- a language that simply allows the specification of points of interconnection and data transformations across those points -- with a language that can support application level[1] module composition and evolution (i.e. re-composition) for a broad spectrum of data integration, migration, and evolution problems. The approach in this endeavor is to apply best practices in module interconnection language (MIL) research within a new and state-of-the-art context, the Object Management Group's (OMA) Common Object Request Broker Architecture (CORBA).

Research problems in the COIL project exist along two dimensions:

1. the development of module interconnection languages to support the domain of database integration and evolution problems;
2. and the development of a module interconnection language framework for CORBA.

Database integration and evolution problems in multidatabase environments present unique challenges in module interconnection research. First, a module interconnection language in this context must provide support during module composition for the integration of the different local database language interfaces that are constituents of a multidatabase environment [2, 17]. Second, module composition in a multidatabase environment is likely require support for traditional and advanced transaction models [6] for inter-module communication. Finally, production-quality database integration, migration, and evolution mechanisms should ideally support fault-tolerant "switch-over" to new module configurations or "composites" [29].

CORBA presents unique challenges in module interconnection research. The Object Management Architecture defines the environment within which CORBA exists [23]. The OMA carries with it a far more complex collection of standard mechanisms, services, and facilities than contexts to which earlier MILs have been applied. These resources make it possible to offer much richer module interconnection mechanisms than heretofore possible. For example, the CORBA Trader service might be leveraged to provide interface-matching capabilities.

At the same time, the existence of the OMA adds complexity to the module interconnection problem. Module interconnection within a CORBA context may involve the management of not only application object dependencies, as provided by traditional MILs, but it may *require* the management of dependencies and behaviors of CORBA services and facilities. An example of this complexity is CORBA's event channel service. It allows both servers and clients to take either consumer or producer roles. Event channels support both push and pull models, allowing the clients and servers using a channel to decide who initiates the event communication. A client can become a push supplier or a pull consumer. The events service also permits blocking and non-blocking push and pull; fan-in and fan-out of event flows; and typed event communication, allowing suppliers to call operations on consumers through some mutually-agreed interface. In addition, notification and callback services are provided between servers and clients [22].

1. The OMG-specific term "Application Level" is used specifically to indicate that COIL is concerned with application-level interconnection.

4. Conclusion: The COIL Project Research Goals

This project is informed and motivated by a broad spectrum of module interconnection language work. The goals for the COIL project are to create a language that can give support in the following areas:

Interface Symmetry

CORBA's interface definition language lacks a symmetry recognized as a fundamental requirement in module interconnection languages [9]. An asymmetry exists in its interface definition language, as in most programming languages, where the interfaces *provided* by objects are made explicit, but the interfaces *required* by objects are encapsulated within objects and are *implicitly* represented by method invocations in the code. It is expected, therefore, that COIL specifications will have the following basic structure.

```
// <COIL specification>
module <module_name> {
    // Resources provided by this module.
    interface <interface_name> {
        <interface_specification>;
    };

    // Resources required by this module.
    dependencies{
        <dependencies_specification>;
    };
};
```

Many module interconnection language paradigms exist from which COIL can draw to create a language model that is symmetric in this respect [2,4,5,7,9,17,20,26,31].

While providing this type of interface symmetry, COIL must support unique features presented by CORBA such as its events service and its call-back model for method invocations. To the best of the authors' knowledge, this latter feature is fairly unique in the context of module interconnection languages. In general, COIL must support bi-directional connections which represent one-to-one mappings between two interfaces, or a set of reversible functions which perform undo logic defined as part of a client/server interaction; and, unidirectional connections which represent either functional transformations or functional compositions of multiple interface parameters.

Application Level Module Composition

COIL must be capable of expressing the semantics of software system composition over collections of CORBA application objects. For example, it should be possible to express the

composition of a set of client application objects with multiple application server objects associated with their underlying database systems. CORBA is designed for integrating objects, not whole applications [11]. A plethora of good research on creating module interconnection languages for existing systems can be drawn upon to achieve this goal [2,4,5,7,9,17,20,26,31]. These approaches must be re-contextualized and extended to deal with the unique challenges presented by database integration and evolution problems and CORBA.

Syntactic Analysis

COIL will be based on a formal syntactic model of a CORBA-based module interconnection language. This will provide a consistent and meaningful framework in which axioms can be developed to reason about structural aspects of module composition. COIL can draw upon work such as [4] and [5] here.

Module Evolution Analysis

COIL must be capable of expressing the essential nature of future structural and functional changes produced by modifications to software systems composed with it. This capability will allow axioms to be developed to identify and reason about the impacts of module re-composition, and tools to be developed which can assist developers in addressing those impacts. For example, this capability would provide support for preserving CORBA facilities and services dependencies, as discussed in the previous section. COIL can draw upon work such as [4] here. This and other approaches must be re-contextualized and extended to deal with the unique challenges presented by database integration and evolution problems and CORBA.

Module Behavior Analysis

COIL must also be capable of expressing behavioral constraints and requirements of software modules. Module composition requires not only that interfaces match, but that the interface provides its services using certain behaviors. For example, one module may require that the module to which it interfaces uses non-blocking behavior, or that it marshals data according to certain capacity constraints. Specification and analysis of software system behavior has been dealt with in [12] and [13]. Again, these approaches must be re-contextualized and extended to deal with the unique challenges presented by database integration and evolution problems and CORBA, as discussed in the previous section. Support for other special types of module behavior such as temporal constraints might also be built into COIL. The DAMSEL project is an example of specialized module interconnection support; in their case, support for event-driven, multimedia applications [25].

Fault-tolerant Module Interconnection

The COIL environment must be capable of supporting fault-tolerant "switch-over" to new module configurations or "composites". The very purpose of COIL (and of Sanctuary) is to support real-world database integration and evolution problems. Such problems involve operational, legacy systems whose functionality remains critical to an organization up to the very moment switch-

over to a new system or configuration takes place. To the authors' knowledge, MIL research has not focused on this problem. Work in this area has focused on the development of communication processes, transaction mechanisms and structures which can be used to perform on-line upgrades and the logical interchange of resources [29]. Again, these approaches must be re-contextualized and extended to deal with the unique challenges presented by database integration and evolution problems and CORBA.

References

1. T. Ball, J. R. Larus: Optimally Profiling and Tracing Programs. TOPLAS. Vol. 16, No. 4. pp. 1319-1360. 1994.

2. O. Bukhres, A. K. Elmagarmid, and E. Kuhn. Advanced Languages for Multidatabase Systems. Chapter in "Object-Oriented Multidatabase Systems", A.K. Elmagarmid, O. Bukhres (eds), Prentice-Hall. 1994.

3. R. Cattell. The Object Database Standard: ODMG-93. Morgan Kaufmann Publishers, San Mateo, California, 1993.

4. T. R. Dean and J. R. Cordy. A Syntactic Theory of Software Architecture. IEEE Transactions on Software Engineering. Vol. 21, No. 4, April 1995.

5. T. R. Dean and D. A. Lamb. A Theory Model Core for Module Interconnection Languages. External Technical Report ISSN-0836-0235-94-370. Department of Computing and Information Science, Queen's University, Kingston, Canada. October 1994.

6. A.K. Elmagarmid (ed.). Database Transaction Models for Advanced Applications. Morgan Kaufmann Publishers. 1991.

7. J. A. Goguen. LIL -- A Library Interconnect Language. In Report on Program Libraries Workshop. pp. 12-51. Menlo Park, California. SRI International. October 1983.

8. R. W. Gray, V. P. Heuring, S. P. Levi, A. M. Sloane, W. M. Waite. "Eli: A Complete, Flexible Compiler Construction System", CACM, 35 (February, 1992) 121-131.

9. P. Hall and R. Weedon. Object Oriented Module Interconnection Languages. Technical Report. Department of Computing. Open University, Milton Keynes, England. 1992.

10. D. Heimbigner and D. McLeod. A federated architecture for information management. ACM Trans. on Office Information Systems, 3(3), pages 253-278. July 1985.

11. INRIA: Projet Sirac. Olan : un environnement pour la programmation par composants. http://sirac.inrialpes.fr/Sirac/olan-francais.html. December 1997.

12. P. Inverardi and A.L. Wolf. Formal Specification and Analysis of Software Architectures Using the Chemical Abstract Machine Model. IEEE Transactions on Software Engineering. Vol. 21, No. 4, April 1995.

13. A. Isazadeh, D. A. Lamb, G. H. MacEwen. Viewcharts: A Behavioral Specification Language for Complex Systems. External Technical Report ISSN-0836-0227-95-388. Department of Computing and Information Science, Queen's University, Kingston, Canada. October 31, 1995.

14. W. Kelley, S. Gala, W. Kim, T. Reyes, and B.Graham. Schema architecture of the UniSQL/M

multidatabase system. In W. Kim, editor, Modern Database Systems, pages 621-648. Addison-Wesley, 1995.

15. R. King, M. Novak, and C. Och. Sybil: Supporting heterogeneous database interoperability with lightweight alliances. In The Third International Workshop on Next Generation Information Technologies and Systems, 1997.

16. R. King, C. Och, and R. Osborne. Sybil: Evolving software persistence layers. Software Engineering News, 1997.

17. E. Kuhn. Multidatabase Language Requirements. Proceedings of the 3rd International RIDE-IMS Workshop. IEEE. 1993.

18. J. Melton. In ISO/ANSI Working Draft Database SQL (SQL3). 1993.

19. Microsoft Corporation. ODBC 3.0 Specification. http://www.microsoft.com/odbc/docs/odbc.hlp. 1997.

20. J. Mullen, O. Bukhres, and A. Elmagarmid. Interbase: A multidatabase system. In O. Bukhres and A. Elmagarmid, editors, Object-Oriented Multidatabase Systems, pages 652-683. Prentice-Hall, 1996.

21. The Object Management Group. The Common Object Request Broker: Architecture and Specification. Revision 2.0. http://www.omg.org. July 1995.

22. The Object Management Group. Common Facilities Architecture. Revision 4.0. http://www.omg.org. November 1995.

23. The Object Management Group, Object Management Architecture, Revision 3.0. http://www.omg.org. 1995.

24. The Object Management Group. Who is OMG? http://www.omg.org/about/whorwe.htm. 1997.

25. P. Pazandak and J. Srivastava. The Language Components of DAMSEL: An Embedable Event-driven Declarative Multimedia Specification Language. Presented At SPIE's Photonics East '95 , First International Symposium on Technologies and Systems for Voice, Video, and Data Communications, Pittsburg, PA. 1995.

26. M. Purtilo. The POLYLITH Software Bus. ACM Transactions of Programming Languages and Systems. Vol 16, No 1, pp 151-174. January 1994.

27. T. W. Reps. The Use of Program Profiling for Software Testing. GI Jahrestagung. 4-16. 1997.

28. P. N. Robillard, D. Coupal. F. Coallier. Profiling Software through the Use of Metrics. SP&E. Vol. 21, No. 5. pp. 507-518. 1991.

29. L. Sha, R. Rajkumar, and M. Gagliardi. A Software Architecture for Dependable and Evolvable Industrial Computing Systems. Technical Report CMU/SEI-95-TR-005. Software Engineering Institute, Carnegie Mellon University. July 1995.

30. M. Shan, R. Ahmed, J. Davis, W. Du, and W. Kent. Pegasus: A heterogeneous information management system. In W. Kim, editor, Modern Database Systems. Addison-Wesley, 1995.

31. W.J. Tracz. Parameterized Programming in LILEANNA. Ph.D Thesis. Stanford University. August 1997.

Systematic Change Management in Dimensional Data Warehousing

R. Bliujute S. Saltenis G. Slivinskas C. S. Jensen

Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E, DK–9220 Aalborg Øst, DENMARK
{rasa, simas, giedrius, csj}@cs.auc.dk

Abstract

With the widespread and increasing use of data warehousing in industry, the design of effective data warehouses and their maintenance has become a focus of attention. Independently of this, the area of temporal databases has been an active area of research for well beyond a decade. This article identifies shortcomings of so-called star schemas, which are widely used in industrial warehousing, in their ability to handle change and subsequently studies the application of temporal techniques for solving these shortcomings.

Star schemas represent a new approach to database design and have gained widespread popularity in data warehousing, but while they have many attractive properties, star schemas do not contend well with so-called slowly changing dimensions and with state-oriented data. We study the use of so-called temporal star schemas that may provide a solution to the identified problems while not fundamentally changing the database design approach. More specifically, we study the relative database size and query performance when using regular star schemas and their temporal counterparts for state-oriented data. We also offer some insight into the relative ease of understanding and querying databases with regular and temporal star schemas.

1 Introduction

Data warehousing [3] is one of the fastest growing segments of the database management market, which, in turn, is one of the biggest markets for software. The area of data warehousing has grown out of a need for decision support that was not met by existing database management technologies. Without data warehousing, the data of an enterprise resides in different databases that belong to systems aimed at supporting the day-to-day business of the enterprise. These databases are often termed operational data stores, and the systems are termed on-line transaction processing systems (OLTP systems) or operational systems. Paraphrasing Kimball [7], OLTP systems are designed to meet the needs of someone who *turns* the wheels of an enterprise. As a result, these systems do not meet the needs of someone who *watches* the wheels and tries to find the ways of making them turn even better. The objective of data warehousing is to meet this need by enabling enterprises to exploit the data that is entered into their operational data stores for decision support, allowing them to base business decisions on careful analyses of accumulated data capturing the past performance of the enterprise.

To enable such analyses, a data warehouse system typically employs its own hardware and software and is separate from the operational systems. Data is extracted from the different operational data stores, is integrated, and is loaded into the data warehouse

database, where it is then accumulated to make the database contain data up to 5–10 years old. The process of loading operational data into the warehouse is typically performed at regular intervals, e.g., every night. This architecture implies that the life cycle of the data warehouse consists of two alternating phases: the relatively short loading phase, where new data is inserted, and the querying phase, where data is not updated, only queried.

Data in data warehouses must be organized so that it is possible to query and analyze it on-line in the ways required by the analysts. It should be easy to formulate and execute queries. Most often these queries will not have a large answer set, but their execution can involve scanning huge amounts of data. Naturally, the requirement is to make these queries run as fast as possible.

To fulfill these goals, the organization of data in a data warehouse must be considered very carefully. Dimensional data modeling [7] is one of the main techniques used for this purpose, and star schemas serve as the means of representing dimensional data in relational databases. While being widely used and possessing desirable features, star schemas do not contend well with so-called *slowly changing dimensions* [7] and with state-oriented data, which occur when we want to record information that remains valid for a duration of time, e.g., when we want to record account balances in a bank.

This paper studies the use of techniques from temporal databases [12] for solving these problems. It describes temporal star schemas and provides a case-based, empirical comparison of temporal star schemas with regular star schemas, considering database size and query performance, as well as the ease of formulating queries. The temporal star schemas considered here capture valid time [6, 11], the time when the facts stored in the database are true in the modeled reality, as an opposite to transaction time, which concerns when the facts are current in the database.

The novel notion of temporal star schemas was introduced in a white paper by Leep Technology, Inc. [9]. While the performance of various representation schemas for temporal data has been the object of study (e.g., [1]), we are not aware of any works that have studied the size, query performance, or user-friendliness of temporal star schemas.

The paper is outlined as follows. Section 2 introduces star schemas, the distinction between state-oriented and event-oriented data, and identifies problems that motivate this study. Section 3 then describes the proposed solution, namely temporal star schemas, covering both state-oriented and event-oriented data. Having presented the solution, Section 4 proceeds to describe the settings for the experimental comparison of temporal and regular star schemas. Sections 5 and 6 compare database size and query performance, respectively. Finally, Section 7 concludes the paper.

2 Star Schema Problems

As a precursor to presenting the identified problems with regular star schemas, these are introduced, and we consider the different data representations that are used for event-oriented and state-oriented data.

2.1 Star Schemas

A star schema is a collection of tables where one central table, the fact table, contains foreign keys to all other tables, the dimension tables, which do not contain any foreign keys. Fact table rows typically contain, in addition to their foreign key references, one or more measured values, typically numerical, that describe some business process. A row in a dimension table contains data, typically textual, that describes aspects of rows in the fact table. The presence of a time dimension that describes when the measured values are in effect is a defining property of a data warehouse [3].

Let us take as an example a company that has a chain of stores that sell various products. Each row in the fact table models a sale of a product and records the quantity sold. Dimension tables Product, Store, and Time describe the products being sold, the stores where products are sold, and the times when products are sold. The star schema is shown in Figure 1. A fact table row may thus store the information that "on April 28, store X sold 200 items of product Y."

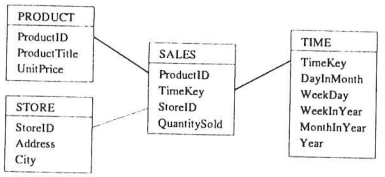

Figure 1: Sample Star Schema

The *granularity* of the measured values in the fact table rows is product by store by day. A single row then records all sales of a single product from a single store that occurred on a single day. Other possible granularities for the time dimension include minute, hour, week, and month. The choice of granularity affects the level of detail for the recorded information and the size of the database.

There is a difference between the attributes in fact versus dimension tables. The non-key attribute values in the fact table usually are numerical and vary continuously, e.g., a store *each day* sells *various* quantities of its products. In contrast the dimension table attribute values, such as store addresses and product titles, are textual, discrete, more or less stable, and serve as constraints in the business analyst's queries. However, sometimes the value of some attributes from the dimension tables change, and methods are needed to ensure that data in the database remains correct and consistent. Such dimensions, where some attribute values may change, are called slowly changing dimensions and are discussed in detail in Section 2.2 below.

We distinguish between two orientations of data: event-oriented data and state-oriented data. Examples of *event-oriented* data could be various types of buys, sales, inventory transfers, and financial transactions. The store example above thus concerns event-oriented data. Examples of *state-oriented* data include, e.g., prices, account balances, and inventory levels. The event and state models are complementary in the sense that having

events, we can construct states that span between events; and likewise, having states we can construct events, representing the changes from one state to another.

The data in the star schema is *represented* either as events or as time-series, depending on the data orientation type. Event-oriented data is represented as events, i.e., each row in the fact table corresponds to an event in real life. State-oriented data can be represented as events, where a fact table row represents the event that caused the state of the object to change; or state-oriented data can be represented as time-series, where each row in the fact table represents a state at some fixed point in time, independently from the state changes of that object (this corresponds to sampling). We will revisit these issues in Section 2.3.

2.2 Problem 1: Ad-hoc Handling of Slowly Changing Dimensions

In the previous section, we assumed implicitly that all the dimension tables are independent of one another. However, this is not true: it can easily be seen that some, or most, dimensions are dependent on the omnipresent time dimension. For example, the prices of products in the product dimension may change over time. We term such time-dependent dimensions *slowly changing* [7].

Their presence leads to potential consistency and correctness problems, which should be handled. In the example, if the price of a product changes, the change must somehow be re"ected in the Product dimension, to represent the correct price for all sales recorded in the fact table of the product. All fact table rows inserted before the change should still refer to the old price, while the new fact table rows (inserted after the change) should refer to the new price.

As proposed by Kimball [7], we can use three methods to handle slowly changing dimensions. Each of these methods provides a different degree of fidelity. Here, we describe only the most used method. In the description, we use a state-oriented star schema with a fact table, termed Balance, recording account balances and with dimension tables Time, Customer, and Account recording the times, customer informations, and account informations of the account balances, respectively. Assume that customer John Smith changed his address on January 15, 1997.

The most popular method for handling slowly changing dimensions proposes to create an additional dimension row at the time of the change with the new attribute value, getting an old description and a new description, see Figure 2.

Balance

	GK	Bal
1/01	1	100
1/10	1	200

Customer

GK	Cld	Name	Address
1	1	John	Ulavej 1

Balance

	GK	Bal
1/01	1	100
1/10	1	200
1/20	2	300

Customer

GK	Cld	Name	Address
1	1	John	Ulavej 1
2	1	John	Ritavej 5

Figure 2: Method Two: Balance and Customer Tables Before and After Changes

We create a new row in the Customer dimension for John Smith where the Address field has the new value. The new row must also have a unique key value. We cannot use, e.g., the social security numbers of customers as the key values because a customer such as John Smith has the same social security number independently of his address. There

is a need for creating an artificial, generalized key (GK), which then requires special handling and consumes additional space in the data warehouse.

Another shortcoming is that the newly updated values in dimensions cannot be used on older facts and vice versa, which is sometimes meaningful. For example, the names of all bank branches may change, but there may still exist a need to track history in terms of the new branch names.

The described method is most generally applicable, but it, as the other two methods, has disadvantages, thus creating a need for a better solution to the problem of handling slowly changing dimensions.

2.3 Problem 2: Ineffective Handling of Change for State-oriented Data

We have mentioned that state-oriented data can be represented as either time-series or as events. Here, we identify potential data warehouse size and query performance problems that may arise from the management of state changes of state-oriented data for both representations.

Under realistic assumptions, the time-series representation inadvertently leads to loss of some information and repetition of other information. To see this, consider the banking example where we want to record the balances of all accounts. If we sample balances every hour then we will not have the complete account information because there can occur many transactions for a single account per hour. On the other hand, if we sample every minute then we will probably not loose information, but redundancy will be enormous because each account (there could be millions) will have a corresponding entry for each minute independently of whether the balance changed or not. In general, it can be difficult to choose a proper time granularity for the time-series representation, because some balances may change very often while others may change quite rarely. The time-series representation may be suitable when state transitions occur regularly in the real world.

With the event representation, there is a row in the fact table for each balance change. If some loss of information is acceptable, it is possible to record facts not for each event (change of a balance), but for several events together. For example, it may be adequate to record one fact for all changes to a balance per hour, independently of how many times the balance changed per hour, if it changed at least once. In the event representation, we also avoid redundant information because balances that do not change do not generate rows for the data warehouse.

However, the event representation creates query performance problems. With state-oriented data, it is likely that there frequently is a need to know the entire period during which a state persisted (e.g., the time when a balance remained at a constant value). We can efficiently find the Time dimension row giving the period's beginning, but finding the end of the period is neither simple nor cheap to compute.

In the next section, we will present the temporal star schema, which represents a possible solution to the problems mentioned in this section.

3 Temporal Star Schemas

Based on the previous section, we conclude that the problem of handling slowly changing dimensions does not have a fully satisfactory and conceptually clean solution using regular star schemas. In contrast, the temporal star schema systematically solves this problem. Besides, we have reasons to believe that the temporal star schema lets us achieve good query performance and a relatively compact data warehouse.

The idea behind the temporal approach to databases is simple: most of the things that we observe change in time, and the database system should capture the changes and represent the historical information in a natural way. The temporal star schema differs from the traditional one in its treatment of time. While the traditional star schema treats time as any other dimension, the temporal star schema omits the time dimension *table* and instead timestamps each row in every table of the schema, treating the fact table and the dimension tables equally with respect to time.

Adding detail to this design, we observe that event-oriented data and state-oriented data are represented differently in the temporal star schema. For event-oriented data, the event representation is used, meaning that each row in the fact table represents some event and has one timestamp, capturing the event occurrence time. For state-oriented data, we employ the state representation, meaning here that each row in the table describes some state and has two timestamps—the beginning and ending times of the period when the state persisted (see Figure 3).

Transaction Date	Transaction amount
1997/04/01	-100
1997/04/06	500

Begin Date	End Date	Account balance
1997/03/20	1997/04/01	10000
1997/04/01	1997/04/06	9900

Figure 3: Event and State Representations of Data

The event representation has the advantage of storing only one timestamp instead of two. Naturally this type of representation is well suited for event-oriented data and event-oriented queries. On the other hand, queries asking questions about states of some object in some periods of time will result in the comparison of timestamps from different rows and thus will execute slower. The state representation, while occupying more space, appears to be better suited for state-oriented data and state-oriented queries.

Taking a closer look at the data warehouse tables one can observe that dimension tables predominantly store state-oriented data. This implies having two timestamps in dimensions independently of whether the fact table represents events or states. For instance, in the banking example to the left in Figure 3, each fact table row represents one bank transaction and has one timestamp, while each Branch dimension row would have two timestamps. If some branch first became operational on July 14, 1990 and originally dealt only with savings accounts, but was upgraded to provide loans also on April 6, 1996, then the Branch dimension table will have two rows for that branch with timestamps July 14, 1990–April 6, 1996 and April 6, 1996–*now*.

Next, some dimension tables may not need timestamps at all. This occurs when a dimension models objects that do not change at all or when it does not describe real-

world objects. An example of the latter could be the inventory status dimension in an inventory tracking data warehouse. This dimension lists only the possible values of the inventory status (received, inspected, boxed, etc.)

When using a temporal star schema, the problems associated with the handling of slowly changing dimensions disappear. Generalized keys are not needed and state-oriented data can be represented as states, diminishing the potential query performance and database size problems that may occur having events or time-series representations in the regular star schema. It is worth noting that the temporal approach is more informative than any regular star schemas—unlike the non-temporal star schemas, it associates the *exact* dates of the changes with all the historical information.

The presence of period timestamps raises some new issues. First, the value *now* has to be represented in one way or another—we choose the maximum date (9999/12/31) for this purpose. Second, referential integrity constraints [8] have to be maintained. Third, the primary key of any table augmented with timestamps is the original key plus one or two timestamps. Joins of such tables become more complex: for the two rows to join, their original keys must match, and their timestamps must overlap.

4 Settings for Comparison of Star Schemas

The previous sections compared regular and temporal star schemas at the conceptual level. Next, data warehouse size and query performance are considered. We investigate state-oriented data, which can be represented in two ways using regular star schemas and in one way using temporal star schemas, leading to a comparison of three data warehouses: a regular time-series warehouse, a regular events warehouse, and a temporal states warehouse. Even if all three warehouses attempt to represent the same information, their sizes as well as the execution speeds of queries on them will generally differ. In order to present concrete numbers of size and speed measures and to give a more real feeling of the differences, we will base the experiments on a case study.

We choose a data warehouse for a bank. We assume that there are 50 branches, 200 different customer types, and 5 account types. The warehouse does not keep information on single accounts and customers because its intended use is for decision support. Thus we have Branch, CustomerType, Time (only in the regular warehouses), and AccountType dimensions. We choose "week" as the time granularity. Each fact table row shows a total sum of all account balances on one fixed day of the week (i.e., Friday) for a certain type of account opened at a certain branch and owned by a certain type of customer.

In order to be able to load appropriate data into all three warehouses, additional assumptions are necessary. We assume that 60% of all aggregated balances change per week, i.e., the fact tables in both the events warehouse and the states warehouse have 40% less entries than the fact table in the time-series warehouse, because the latter one does not exclude duplicate values. For instance, if the total balance of all customers older than 60 and living in Aalborg remains the same for three weeks, in the time-series warehouse fact

table, this will be represented by three rows, while in the events and states warehouses, there will be only one fact table row. We term the number 0.6 (which corresponds to 60%) the *actual change rate* (ACR).

We suppose that there are 7500 relevant combinations of AccountType, CustomerType, and Branch (5 account types · 150 customer types · 10 branches). One customer type is combined with only 10 branches of 50, because most likely those who live in, e.g., Aalborg never open accounts in the branches that are located in, e.g., Copenhagen. We assume that the AccountType and CustomerType dimensions do not change over time. The only slowly changing dimension is Branch.

The schemas of the two regular data warehouses are the same. In the temporal warehouse, the AccountType and CustomerType tables have the same schema as in the regular warehouses, but the Branch and the fact tables are different, and the Time table does not exist. The Branch tables in the regular warehouses contain a generalized key attribute, while in the temporal warehouse, the Branch table contains two timestamp attributes. The fact tables of the regular warehouses contain 4 foreign key attributes and a `Balance` attribute. The fact table of the temporal warehouse includes only 3 foreign keys (there is no Time table) in addition to two timestamp attributes and `Balance`.

We create an index on the fact table of each warehouse. For each of the regular warehouse fact tables, we create a primary index on the foreign key attributes, and for the temporal warehouse fact table we create an index on the foreign key attributes and `BeginDate`. For the temporal warehouse, we also experiment with primary index that additionally includes the `EndDate` attribute. SQL source code for creation of tables and indexes together with the size estimation of the attribute values of the tables can be found in reference [2]. The experiments were carried out using the Oracle DBMS (version 7.2.2.3).

5 Data Warehouse Size

We proceed to compare the sizes of the three warehouses. The data warehouse size is given by the sizes of the fact table, the dimension tables, and the indexes. Because dimension tables are generally very small [7] relative to the fact tables and have very little impact on the total size, we consider only the fact tables and indexes. The size of a table is determined by the number and size of the rows.[1]

The general formula for the computation of the size of a warehouse X is given next.

$$size(X) = rows(X) \cdot (FRL(X) + IEL(X)) \quad (1)$$

In the formula, X can be a temporal states warehouse (S), a regular events warehouse (E), or a regular time-series warehouse (TS). The function $rows(X)$ returns the number of rows in the fact table of warehouse X. If we assume that there are N rows in the time-series warehouse fact table, i.e., $rows(TS) = N$, then $rows(S) = rows(E) = \text{ACR} \cdot N$.

[1] For simplicity, we use numbers of rows and sizes of rows instead of computing the exact numbers of data blocks used in Oracle. Using data blocks leads to the same conclusions.

In Section 4, we stated that the actual change rate in the banking case is 0.6, that there are 7500 relevant combinations of AccountType, Branch, and CustomerType, and that we record data once per week. For one year, there will thus be 7500 · 52 = 390000 rows in the time-series warehouse fact table and 0.6 · 390000 = 234000 rows in the fact tables of events and states warehouses. Thus the regular schema with the events representation and the temporal schema are better suited for representing changes that occur *irregularly* in the real world. In such cases, it is difficult or impossible to choose an appropriate sampling rate (time granularity) for the regular schema with the time-series representation.

The remaining functions in Formula 1 compute the fact table row length and the index entry length for a warehouse. Formulas for these functions are given below, and quantities used in the formulas are defined in Table 1.

$$FRL(X) = \text{rh} + \text{NonFKL}(X) + \text{FKL}(X) + \text{n}(X) \cdot 1\text{byte}^2 \quad (2)$$

$$IEL(X) = \text{eh} + \text{rid} + \text{NonFKLIndex}(X) + \text{FKL}(X) + \text{nIndex}(X) \cdot 1\text{byte} \quad (3)$$

Notation	Definition
rh	row header size (3 bytes in Oracle)
NonFKL(X)	the sum of the sizes of the non-foreign key attributes in the fact table of warehouse X
FKL(X)	the sum of the sizes of the foreign key attributes in the fact table of warehouse X
n(X)	the number of fact table attributes in warehouse X
eh	index entry header size (2 bytes in Oracle)
rid	row identifier size (6 bytes in Oracle)
NonFKLIndex(X)	the sum of the sizes of the indexed non-foreign key attributes from the fact table of warehouse X
nIndex(X)	the number of attributes in the index defined on a fact table of warehouse X

Table 1: Definition of Notations

The fact table row length (Formula 2) is the same in the regular events and time-series warehouses, but differs for the temporal warehouse. The difference is caused by (1) the presence of two timestamp attributes in the fact table of the temporal warehouse (S), (2) the presence of a generalized key attribute—which references the slowly changing Branch dimension—in the fact tables of the regular warehouses (TS and E), and (3) the presence of the foreign key reference to the Time table in the fact tables of the regular warehouses (TS and E).

The sizes of the fact tables, indexes, and the total sizes of three data warehouses for the banking case are given in Table 2.

Structure	*size*(TS)	*size*(E)	*size*(S)
Fact table	390000 · 21b = 8Mb	234000 · 21b = 4.8Mb	234000 · 33b = 7.54Mb
Index	390000 · 20b = 7.6Mb	234000 · 20b = 4.57Mb	234000 · 24b = 5.48Mb
Total size	15.6Mb	9.37Mb	13.02Mb

Table 2: Data Warehouse Sizes for One Year of Data in the Banking Case Study

Two characteristics define index size: the number and the length of index entries. The number of index entries is the same as the number of fact table rows. The index

[2] For each attribute its length value has to be stored. In Oracle for all attributes with average length less than 250 bytes, 1 byte is enough to store the length value itself.

entry length is affected by (1) the space usage of the indexed date-type attributes for the temporal warehouse, (2) the presence of foreign-key references to slowly changing dimensions, and (3) the presence of a foreign-key reference to the Time dimension table for the regular warehouses.

Based on Formula 1 and the numbers provided in Table 2, we can determine the sizes for the three warehouses as a function of the actual change rate, see Figure 4. It can be seen that the states warehouse size is smaller than the time-series warehouse size when the actual change rate is smaller than 0.72 (if the state warehouse index includes only one date-type attribute—the `BeginDate`).

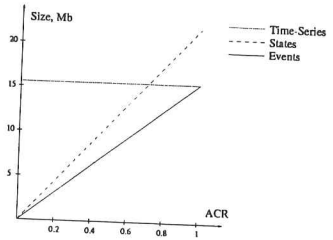

Figure 4: Data Warehouse Sizes as a Function of the Actual Change Rate

The inclusion of also the `EndDate` attribute in the index would increase the index size (consequently increasing the total size of the states warehouse about 14% and giving an actual change rate break-even point equal approximately to 0.62), but might also speed up queries. It is important to achieve an appropriate trade-off between space and speed; to do so, analyzing query performance is necessary.

6 Query Performance Experiments

In this section, we compare query performance when using each of the three data warehouses. We begin with general observations about the optimal query execution scenario. Then we give the types of queries used in the experiments, present the results, and draw conclusions.

6.1 General Observations

The special structure of a star schema, most notably the presence of one huge fact table and a number of relatively small dimension tables, greatly impacts the set of appropriate query execution strategies for data warehouse queries. Table size differences and distribution of values in tables have to be taken into account, and so cost-based optimization should be employed.

When dimensions are small, the optimal scenario for executing a query in a data warehouse begins with full scans of the dimension tables, identifying the dimension table rows that satisfy the constraints on dimension attributes given in the query. The next step is to

perform the Cartesian product of the keys of the qualifying rows. These composite keys are then used for lookup in the fact table using its index, thus identifying the qualifying fact table rows.

The main idea is to access the fact table as late as possible, using its primary index as much as possible. The index usage may range from performing unique key lookup to not using the index at all. This depends on the ordering of the key attributes of dimensions in the composite primary key on which the fact table is indexed and on the set of dimensional constraints given in the concrete query. For example, it is not beneficial to use the index at all if there are no constraints on the dimension having its key first in the composite key of the fact table index.

To fairly compare query execution speeds for the three warehouses, the first step is to ensure that the queries are executed optimally. The execution times provided in Section 6.3 were thus measured only after extensive and careful tuning of each query.

6.2 Query Types and Queries

The goal of the query performance experiments was to find out what affects the query performance in the regular warehouses and in the temporal warehouse. To accomplish this, we define several broad query types and then investigate representative queries of these different types. Since the warehouses differ in how they handle (temporal) change, the types of queries are defined based on the types of temporal predicates they employ and what they retrieve. The chosen types of queries are given in Figure 5.

Query Type	*Type of Temporal Predicate*	*Retrieval of*
TYPE1	specified time point	non-temporal value
TYPE2	specified time period	non-temporal value
TYPE3	specified time duration	non-temporal value
TYPE4	specified nontemporal-attribute value	temporal value

Figure 5: Query types

We find it reasonable to assume that differences in how queries involve time (constraints on time, operations on time-attribute values) lead to differences in query performance in the three warehouses. We thus believe that the chosen types of queries allow us to cover a broad range of aspects of query performance and to identify potential differences among the three warehouses. We emphasize that, of course, not all queries of a given query type have the same performance.

Next, we give sample queries for each of the four query types (many more queries were studied; here, we give the most illustrative ones).

TYPE 1

q.1.1: *What is the balance—for customers living in City3, with zip code 9086, and whose age is between 25 and 40—of savings accounts in Branch9 on September 10, 1997?*

q.1.2: *How much money was in all Branch10 savings accounts on September 10, 1997?*

q.1.3: *Which customer type had the biggest amount in savings accounts in Branch26 on May 1, 1997?*

q.1.4: *Compare the balances—for customers living in City3, with zip code 9086, and whose age is between 25 and 40—of savings accounts in Branch9 on September 10, 1997 and October 10, 1997.*

TYPE 2	
q.2.1:	*What were the balances—for customers living in City3, with zip code 9086, and whose age is between 25 and 40—of savings accounts in Branch9 from August 1, 1997 until November 1, 1997?*
q.2.2:	*Which customer types used two or more account types in Branch10 during 1997?*
q.2.3:	*What was the average April 1997 balance of savings accounts in Branch10 for customers younger than 18 that live in City5 and have zip code 5648?*

TYPE 3	
q.3.1:	*What customer types had the same balance in savings accounts in Branch2 for at least 12 weeks?*

TYPE 4	
q.4.1:	*When were the balances of savings accounts in Branch41 for customers younger than 18, living in City7, and with zip code 7700 bigger than 12000?*

6.3 Performance Results

We evaluated each of the queries given in the previous section a total of 10 times, measuring their actual execution times (see Figure 6). Execution times shown in the figure for queries on the temporal warehouse were measured using the index including both `BeginDate` and `EndDate` because this index yielded the best performance. The query execution times are normalized with respect to the longest execution time, which is given in seconds at the top of the appropriate column. Additional detail, including the actual tuned SQL queries and their execution plans, are provided elsewhere [2].

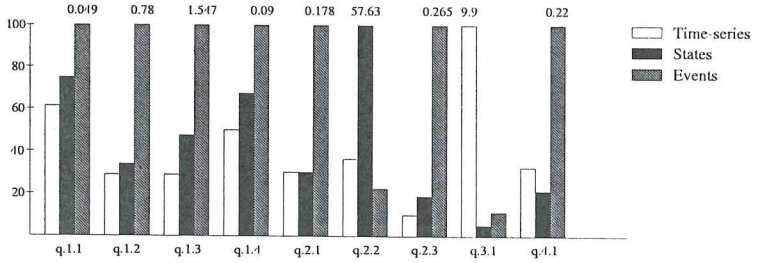

Figure 6: Query Execution Times in Percent of the Slowest Query

As we can see from Figure 6, queries on the temporal warehouse and the time-series warehouse are in most cases more efficient than queries on the events warehouse. The main reason is that in the events warehouse, it is difficult to find the time period when some state was valid. This is because one row in the fact table represents the beginning of the period; and in order to get an end of the period, which is located in another row, a subquery is needed. In addition to adversely affecting performance, subqueries reduce the readability and understandability of queries, especially if several subqueries are present and perhaps are nested.

In all queries (except Type 1) on the events warehouse, we also face the problems of finding the time period when the *current* balance state is valid. To accomplish this, either an additional subquery or a `UNION` with another `SELECT` statement is necessary to catch

the "terminating" event. However, there are also factors that reduce query performance in the other two warehouses.

The time-series warehouse fact table contains duplicate values and is bigger than the fact tables in the other two warehouses. Therefore more fact table rows satisfy the given query constraints and more blocks are often read from disk. For example, this is the main reason for the poor performance of query q.3.1 in the time-series warehouse.

On the other hand, the fact table rows and index entries are bigger in the temporal warehouse; thus, more disk block reads are required for the same number of rows than in the regular warehouses.

Another difference between the presented warehouses is the absence of the Time dimension in the temporal warehouse, thus eliminating any joins with such a table. The price for not having a time table is to have two date attributes in the fact table, which can cause lower performance for some queries.

Many queries (e.g., of Type 1) on the time-series and events warehouses access only those fact table rows that affect the answer and do not access fact table rows to test time constraints. The same holds for the temporal warehouse with both dates in the index. But if only one of the dates is present in the index, these queries on the temporal warehouse are forced to access far more fact-table rows. This results in more disk accesses and slows down query execution.

We have experienced that date comparisons are expensive on their own in Oracle. This becomes clear when a lot of dates must be checked. This holds for query q.2.2 (the worst performance in the temporal warehouse) where the index cannot be used, meaning that all fact table rows are accessed; for the temporal warehouse, there is a need to check two timestamps of each row to determine whether they overlap with a given period.

Joins with slowly changing dimensions in the temporal warehouse are more complicated than those in the regular warehouses—additional checks of fact and dimension row timestamps are made to ensure that the fact row timestamps overlap with the dimension row timestamps. This is an issue particularly for queries of Type 2, 3, and 4. Temporal query language [13] would make it significantly easier to formulate temporal join queries.

The above-mentioned problems with temporal warehouses explain why query performance for the time series warehouse is usually slightly better than for the states warehouse.

The systematic handling of slowly changing dimensions in the temporal star schema leads to complex temporal joins, which are not attractive. But the handling of slowly changing dimensions in the regular star schema using generalized keys for the Branch dimension also complicates queries because values of these keys do not identify real Branch entities (there can exist several rows, with different generalized key values, for the same branch). If a query on a regular warehouse contains a subquery and there is a constraint on Branch in the main query, a join with Branch is needed in the subquery. This makes such queries difficult to write and hard to comprehend.

Another important point is that for all three warehouses, virtually every query requires

manual tuning (using Oracle's hints) to achieve reasonable performance.

The general conclusion is that queries on the events warehouse always tend to become more complex and take more time to execute, while queries on the time-series and temporal warehouses stay simpler and run faster. The extended version of this paper [2] elaborates on the issues discussed in this section.

7 Conclusions and Future Research

Dimensional data modeling using star schemas is a typical approach when designing a relationally-based data warehouse. The regular star schema treats all dimensions, one of them being the Time dimension, equally and assumes them to be independent. A data warehouse may capture events or states; and events or states are represented as either time-series or events using the star schema. Star schemas have fundamental difficulties with the handling of slowly changing dimensions and change in state oriented-data: the time-series representation records potentially large amounts of redundant data, and the events representation has difficulties with query performance. Both representations fall short in capturing the actual times when dimensions change.

A possible solution to these problems, temporal star schemas systematically capture changes to fact and dimension tables alike. In temporal star schemas, time is not a separate, independent dimension table, but is a dimension of all tables and is represented via one or more time-valued attributes in all tables. Event-oriented data is represented as events (with one time attribute in the fact table), and state-oriented data as states (with two time attributes) in the temporal star schemas.

In the temporal star schema, real-world changes are handled in a natural and systematic way, allowing for the storage of the full and correct history of time-varying data. After investigating data warehouse size and query performance in three warehouses representing state-oriented data, we conclude that queries are easier to write and usually run faster in the temporal states warehouse than in the regular events warehouse; and the temporal states warehouse size is smaller than that of the regular time-series warehouse. The temporal states warehouse does not keep redundant information and is convenient for computing periods when some state was valid, which is usually needed in state-oriented queries.

Additional studies may shed further light on the properties of temporal star schemas. First, experiments can be carried out using real data and real queries. Also, the issue of efficient bulkloading may be studied. Updates to existing fact-table rows will be necessary when bulkloading the temporal warehouse, while, in the regular warehouses, it is only necessary to insert new rows. It would also be of interest to analyze temporal star schemas that include a time table. In this case, it would be enough to keep compact integer-type timestamps for each row, making comparisons more efficient. However, joins with the Time dimension table will be required.

Another research direction is to investigate advanced join techniques such as the STAR-

join using the STARindex [10] or other techniques involving precomputed joins [14]. The performance gain resulting from the usage of precomputed joins would probably be more visible in the temporal warehouse than in the regular one, because temporal joins are more complex and expensive than regular joins.

Acknowledgements

This research was supported in part by the Danish Technical Research Council through grant 9700780 and by the CHOROCHRONOS project, funded by the European Commission DG XII Science, Research and Development, as a Networks Activity of the Training and Mobility of Researchers Programme, contract no. FMRX-CT96-0056.

References

[1] I. Ahn, R. T. Snodgrass. Partitioned storage for temporal databases. *Information Systems*, 13(4):369-391 (1988).

[2] R. Bliujute, S. Saltenis, G. Slivinskas, and C. S. Jensen. Systematic Change Management in Dimensional Data Warehousing (long version). URL: <http://www.cs.auc.dk/~simas/tss.html> (current as of 5 Feb 1998).

[3] W. K. Inmon. *Building The Data Warehouse*. John Wiley & Sons, 2nd ed. (1996).

[4] C. S. Jensen et al. A Consensus Test Suite of Temporal Database Queries. TR-R-93–2034. Dept. of Mathematics and Computer Science, Aalborg University (1993).

[5] C. S. Jensen et al., A Consensus Glossary of Temporal Database Concepts. *ACM SIGMOD Record*, 23(1):52-64 (1994).

[6] C. S. Jensen and R. Snodgrass. Semantics of Time-Varying Information. *Information Systems*, 21(4):311-352 (1996).

[7] R. Kimball. *The Data Warehouse Toolkit*. John Wiley & Sons (1996).

[8] T. Myrach, G. F. Knolmayer, R. Barnet. On Ensuring Keys and Referential Integrity in the Temporal Database Language TSQL2. *Proceedings of the Second International Baltic Workshop*, pp. 171–181 (1996).

[9] Leep Technologies, Inc. Supporting Temporal Data in a Warehouse. URL: <http://www.iftime.com/wpaper2.htm> (current as of 5 Feb 1998).

[10] Red Brick Systems, Inc. Star Schema Processing for Complex Queries. URL: <http://www.redbrick.com/rbs-g/whitepapers/starschema.html> (current as of 5 Feb 1998).

[11] R. T. Snodgrass, I. Ahn. A Taxonomy of Time in Databases. *Proceedings of ACM SIGMOD'85*, pp. 236–246 (1985).

[12] R. T. Snodgrass. Temporal Databases. *IEEE Computer*, 19(9):35-42 (1986).

[13] R. T. Snodgrass et al. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers (1995).

[14] P. Valduriez. Join Indices. *ACM TODS*, 12(2):218-246 (1997).

Views Defined by Dynamic Assertions

X. Delannoy A. Simonet M. Simonet

TIMC - IMAG - CNRS
Faculté de Médecine de Grenoble
38706 La Tronche Cedex - France
Xavier.Delannoy@imag.fr, {Ana,Michel}.Simonet@imag.fr

In this paper a framework is proposed to extend the Osiris object oriented database system with views defined by past temporal logic formulas. In particular, an original scheme to classify objects within these « dynamic » views is proposed. It relies on a translation of temporal logic formulas to regular expressions. It proves to be cost effective and to integrate harmoniously with the classification scheme used by the Osiris system for non dynamic views. [6] presents a very first draft of the proposed framework.

1. Background

Database object models such as ODMG [1] organise database schemas into graphs of subclasses and superclasses. A subclass inherits all the characteristics of its superclass(es) and can define and/or redefine characteristics. An instance of a subclass may be treated as an instance of each superclass.

These notions of classes and subclasses originate from the object oriented programming language field. The choice to remain in the framework of concepts grounded in the programming language field instead of the modelling field raises several objections, as mentioned in [2]. Examples are : object migration to a different class has to be done explicitly, the specification of views requires the building of additional subclasses, and all information inherited from the superclasses may not be relevant to the user of a subclass.

The Osiris database object model [3][4] derived from the abstract type theory [5], is a model centred on views and P-types which circumvents these objections. The central idea is that a same object of the real world can be seen from different *points of view* (hereafter referred to as *views*), possibly attributed to distinct categories of database users. The set of the views specified on the objects of a same family (Vehicles, Persons, ...) compose a P-type.

A view is defined by the view(s) it specialises and its own attributes and/or assertions. The root view of the so formed hierarchy is called the *minimal view*. Assertions are Horn clauses made of domain predicates, that is predicates of the form ATTR∈SUBDOMAIN where ATTR is an object attribute and SUBDOMAIN a subdomain of the domain of the attribute. An object belongs to the view(s) of which it satisfies the assertions.

Consider, for example, the set of the employees of a company. Some users of the database system may be concerned only with senior employees and some others by employees with children ; both groups of employees are defined by views on the set of employees. Assuming that no other views are defined, a P-type EMPLOYEE on the set of employees can be defined by :

```
P-type EMPLOYEE

  View MINIMAL
    Attributes
      Name : STRING ;
      Age : INTEGER ;
      Length_of_Service : INTEGER;
      Nb_Children : INTEGER ;
      Nb_Subordinates : INTEGER ;
    Assertions
      Age≤120 ; // Age∈[INF[1],120][2]
  End View

  View SENIOR : MINIMAL // Inherits from view MINIMAL
    Assertions
      Age≥60 ; // Age∈[60,120]
      Age≤65 ∧ Nb_Children=0 → Length_of_Service≥20 ;
               // Age∈[INF,65]∧Nb_Children∈[INF,0]→Length_of_Service∈[20,SUP]
  End View

  View EMPLOYEE_WITH_CHILDREN : MINIMAL
    Assertions
      Nb_Children≥1 ; // Nb_Children ∈[1,SUP]
  End View

End P-type
```

An employee of the set of employees of the company is an object (or instance) of the P-type EMPLOYEE and belongs to one or several views of the P-type. Updates on the attributes of an object may change the view(s) to which it belongs. An update is accepted if the object remains at least in the minimal view. For example, the update of the age of an employee to 150 will be rejected.

[1] INF and SUP are generic domain bounds for ordered and not explicitly bounded domains.
[2] We give in comments the assertions in terms of domain predicates.

Determining to which view(s) an object belongs is the very problem of instance classification (or recognition) in a Knowledge Base perspective [2]. In the Osiris system it is addressed by the building for each P-type, at definition, of an associated *Classification Space* from the assertions defining the views. For the P-type EMPLOYEE it is represented by the three dimensional diagram of Fig. 1 below. There are as many dimensions as classifying attributes, i.e., attributes which participate to view definitions. The building of the Classification Space is detailed in [6]. Objects which belong to the same « cubes » of the diagram belong to the same view(s) of the P-type.

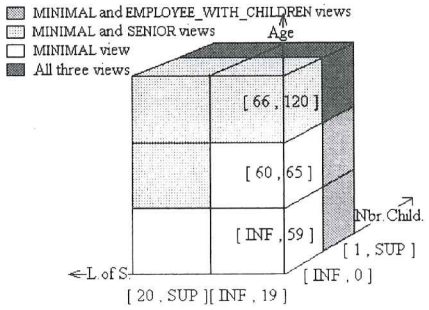

Fig. 1: Classification Space for EMPLOYEE

2. Dynamic Assertions

In Osiris, views are defined by assertions on the current state of the objects. However, in areas such as office automation, financial trading, judicial records, workforce management, and more generally whenever history-related characteristics are concerned, there is a need to consider views defined by assertions involving several past[3] states of the objects. We denote them *dynamic assertions*. For example, it may be relevant to create a view MANAGER on the set of the employees of a company on the criteria of being more than 40 (years old) and since then having more than 10 subordinates.

Taking these assertions into account raises several questions. Firstly it requires to define what the history of an object is. Secondly, a formalism to express dynamic assertions is required. Finally, the question of evaluating efficiently those assertions has to be addressed. The proposed answers to these three questions organise the remainder of the paper.

[3] Future states are not considered since they are undetermined.

3. History of an Object

Each update on an object generates a new state for this object. The successive updates on an object O generate a *finite sequence*[4] *of states* denoted S(O) and called *the history* of the object O. It contains at least the first state of the object, i.e., the state at insert in the database. S(O) is isomorphic to an interval [1,n] of the set of integers. In this interval 1 is the *initial state* and n the *current state* of the object. These two states mark the boundaries of the history of the object.

4. Expression of Dynamic Assertions

A usual means for expressing properties on sequences is temporal logic. We use this formalism to express dynamic assertions. As history has only to do with the past, we consider only *past* temporal logic. Therefore dynamic assertions will be expressed by Past Propositional Temporal Logic (PPTL) formulas. On a technical point of view, temporal logic is not different from modal logic.

- *Syntax of PPTL*

Informally speaking the syntax of PPTL is that of propositional logic extended with two « temporal » operators : the « $\Box$ » operator and the « $\Diamond$ » operator. More formally, consider a set $Vp = \{p,q,...\}$ of propositional symbols, the set $\{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$ of binary operator symbols, the set $\{\neg, (), \Diamond, \Box\}$ of unary operator symbols and the two boolean values **true** and **false**. Formulas of PPTL on Vp are defined as follows :

(*i*) **true** and **false** are formulas ;
(*ii*) any element of Vp is a formula ;
(*iii*) if F and G are formulas, then so are $\neg F$, $F \wedge G$, $F \vee G$, $F \Rightarrow G$, $F \Leftrightarrow G$, $\Box F$, $\Diamond F$, (F) ;
(*iv*) nothing else is a formula.

- *Semantics of PPTL*

Modal logic is classically interpreted on the so called Kripke frames and so are PPTL formulas. Consequently, they are interpreted on models made of a 3-uple $M = <W,R,m>$. In this 3-uple, W is a set of dates, R is a binary relation on W named relation of temporal precedence and $m : W \rightarrow P(Vp)$ is a function from W to the powerset of Vp. This function associates to each date the set of propositional symbols which evaluate to **true** at this date.

Given a model M, a date $w \in W$ and a formula F, one defines the relation « F is true at w in M » (notation $M,w \models F$) by the following rules :

$M,w \models p$ iff $p \in m(w)$;
$M,w \models \neg A$ iff not ($M,w \models A$) ;
$M,w \models A \wedge B$ iff $M,w \models A$ and $M,w \models B$;

[4] By construction, states do not repeat in this sequence.

$M,w \models \Box A$ iff $\forall\, w' \in W$, $w'Rw$ implies $M,w' \models A$;
$M,w \models \Diamond A$ iff there exists $w' \in W$ such that $w'Rw$ and $M,w' \models A$.

Less formally the meaning of $\Box$ is « always in the past » and the meaning of $\Diamond$ is « once in the past ». The universal nature of and the existential nature of $\Diamond$ make these two operators dual by the relation $\Box A \equiv \neg\Diamond\neg A$. Therefore, the PPTL syntax only requires one of these two operators. It is common practice however to consider them both. As in proposional logic the semantics of $\vee$, $\Rightarrow$ et $\Leftrightarrow$ is defined from that of $\neg$ and $\wedge$.

Within the formalism of PPTL, the dynamic assertion which defines the view MANAGER is expressed by the following formula,
(Φ) $\Box(\Diamond$Age>40 $\Rightarrow$ Nb_Subordinates>10) $\wedge\ \Diamond$Age>40.

5. Evaluation of Dynamic Assertions

A naïve evaluation of dynamic assertions raises two problems : (*i*) it requires access to the whole history of the objects and (*ii*) the « longer » the history the higher the cost of evaluation. On the contrary, the method we propose does not require access to object histories (*history-less* method). Therefore its cost is limited and constant. It is based on the translation of each dynamic assertion into a regular expression.

5.1 A Model of Interpretation for Dynamic Assertions

Let H be a dynamic assertion made of n elementary propositions p_i. H_Σ is the cartesian product $\overset{n}{\underset{i=1}{X}}$ (**true**,**false**) or more simply $\overset{n}{\underset{i=1}{X}}(+,-)$. Each element of H_Σ is called a *truth-vector*. For example, if H is made of two elementary propositions p_1 et p_2, $H_\Sigma = \{(+,+), (+,-), (-,+), (-,-)\}$.

Any state of an object O is associated with a unique truth-vector of H_Σ. For example, if a state makes p_1 true but not p_2, then this state is associated with the truth-vector $(+,-)$. Therefore one can canonically associate to S(O) a unique sequence of truth-vectors noted $S_\Sigma(O)$. Now, the truth value of H in the current state only depends on the truth value of its elementary propositions on S(O) and therefore only depends on $S_\Sigma(O)$ which constitutes the model of interpretation of H. Taking the notations of § 4, one have :

W : the set of states of object O ;
R : the order relation on W which describes the sequence S(O) ;
m : the function which associates to each state of the object O the corresponding truthvector.

5.2 Regular Expressions

Let Σ and $\Sigma' = \{\bullet, \cup, \cap, -, *, +, \varnothing, ()\}$ be two disjoints alphabets. A word P on $\Sigma \cup \Sigma'$ is a regular expression on Σ iff P is either :

(*i*) a character of Σ or the character $\varnothing$;
(*ii*) of the form $(Q \bullet R)$, $(Q \cup R)$, $(Q \cap R)$, $(Q - R)$, $(Q)^*$, or $(Q)^+$, where Q and R are themselves regular expressions on Σ. In the remainder, parenthesis are omitted when it does not raise ambiguity.

The following rules associate to each regular expression P a language L(P) on Σ :

(*i*) the language described by $\varnothing$ is the empty language ;
(*ii*) the language described by $a \in \Sigma$ is the word « a » ;
(*iii*) for the regular expressions Q and R on Σ,

(*concatenation*) $L(Q \bullet R) = L(Q) \bullet L(R)$
(*Kleene closure*) $L(Q^*) = L(Q)^*$
(*union*) $L(Q \cup R) = L(Q) \cup L(R)$ (*difference*) $L(Q - R) = L(Q) - L(R)$
(*intersection*) $L(Q \cap R) = L(Q) \cap L(R) = L(Q - (Q - R))$
$L(Q^+) = L(Q) \bullet L(Q^*)$.

A language $\mathcal{L}$ on Σ is *regular* iff there exists a regular expression P such that $\mathcal{L} = L(P)$.

5.3 Dynamic Assertions and Regular Expressions

Let H be a dynamic assertion. L(H) is the language made of the sequences of H_Σ for which H evaluates to **true**.

Theorem – For any dynamic assertion H, L(H) is regular.

The proof of this theorem is given in [7], chapter 10. A consequence of this theorem is that for any dynamic assertion H there exists a regular expression R(H) which describes the language L(H). Again in [7], chapter 10, we show that the following rules define R(H) inductively[5] :

(*i*) $R(p_i) = [\pm]^* \bullet [+_i]$;
(*ii*) $R(\neg F) = [\pm]^+ - R(F)$;
(*iii*) $R(F \vee G) = R(F) \cup R(G)$;
(*iv*) $R(\Diamond F) = R(F) \bullet [\pm]^+$.

In particular, $R(\Box F) = R(\neg\Diamond\neg F) = [\pm]^+ - (([\pm]^+ - R(F)) \bullet [\pm]^+)$ and $R(F \wedge G) = R(F) \cap R(G)$.

Example – For the dynamic assertion $\Box(\Diamond p_1 \vee p_1)$, one gets,

[5] For the purpose of simplicity the following notations are introduced : $[\pm]$ designates H_Σ and $[+_i]$ designates the set of all truth-vectors where p_i is true.

$$
\begin{aligned}
&R(\Box(\Diamond p_1 \vee p_1)) \\
&= [\pm]^+ - (([\pm]^+ - R(\Diamond p_1 \vee p_1)) \bullet [\pm]^+) \\
&= [\pm]^+ - (([\pm]^+ (R(\Diamond p_1) \cup R(p_1))) \bullet [\pm]^+) \\
&= [\pm]^+ - ([\pm]^+ ([\pm]^* \bullet [+_1] \bullet [\pm]^+) \cup ([\pm]^* \bullet [+_1]))) \bullet [\pm]^+)
\end{aligned}
$$

According to the definition of L(H), H evaluates to **true** for an object O iff $S_\Sigma(O) \in L(H)$. According to the preceding theorem, it is equivalent to check that $S_\Sigma(O) \in R(H)$. Evaluating a dynamic assertion is therefore equivalent to checking that a word belongs to the language described by a regular expression. It is quite usual to use finite state automata to perform this task.

5.4 Finite State Automata

Indeed, it is a well known result (see e.g, [8]) that for any regular expression there is an automaton[6] which recognises the same language. Below is the automaton for R(Φ). It was generated with the Rank Xerox finite state compiler[7]. This automaton is made of three nodes. One of them, fs_1, is a final node.

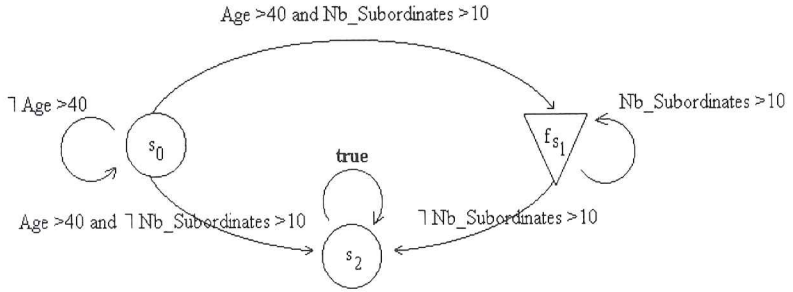

Fig. 2 : Finite-state automaton for the dynamic assertion Φ

Finally, the proposed method of evaluation for dynamic assertions has two parts :

(*i*) *building of the finite state automata* (one per dynamic assertion). This is performed by first translating the dynamic assertions to regular expressions and next to automata. A pointer to each of these automata is then added to each object. The pointers initially point to the initial node of the corresponding automata.

(*ii*) *monitoring*. After each update on an object, its pointers are moved according to the truth-vector associated with the new state of the object (since automata are deterministic only one new state is possible). If the new node reached is a final node, then it means that the corresponding dynamic assertion evaluates to true.

[6] Since a non-deterministic automaton can always be translated to a deterministic one, we only consider here deterministic automata.

[7] The Xerox Finite-State Compiler is at http://www.rxrc.xerox.com/research/mltt/fst/fsinput.html.

6. Classification Space with Dynamic Assertions

In view definitions, dynamic assertions act as boolean attributes, therefore extending the Classification Space by one dimension. For example, adding the view MANAGER to the P-type EMPLOYEE would result in adding an axis for the dynamic assertion Φ in the Classification Space of Fig. 1. This axis would only be divided into two parts : **true** and **false**. This harmonious embedding of dynamic assertions within the Classification Space allows to consider views defined both by static and dynamic assertions at no additional cost.

7. Conclusion and Future Work

This paper presents a framework for taking time-related properties into account in the specification and computation of views in an object oriented database centred on the notion of view. Properties are defined in PPTL, a subset of temporal logic. Evaluation of the properties is efficient and constant since it does not require access to the history of the objects. Only two temporal operators are concerned, but the method can be extended to other temporal operators provided that one can give their corresponding regular expression.

The method for evaluating temporal logic properties is general enough to be used for other purposes than the classification of objects into views. For example we used it to implement the typing *a posteriori* of objects of the NewtonScript [9] prototype-based language, i.e., a language where objects are not instances of classes but are fully defined by themselves. This implementation is presented in [7], chapter 11.

Other areas of application are possible such as workflow rule processing or the checking of dynamic integrity constraints[8] in ODMG compliant object databases. For the future we are working towards providing the regular expressions for some other « classical » temporal operators. Indeed, for some of them, e.g. the SINCE operator, it proves to be a rather difficult task.

Acknowledgements

The authors wish to mention the attentive contribution of C. Del Vigna to the formal aspects of this work and sincerely thank him accordingly.

References

[1] Cattell, R.G.G, *The Object Oriented Database Standard : ODMG93*, Morgan Kaufmann, 1994.

[8] See, e.g., [10] or [11].

[2] Simonet, A., Simonet, M., *OSIRIS : an Object-Oriented System unifying Databases and Knowledge Bases*, KBKS 95 : Toward very Large Knowledge Bases, IOS Press, 1995.

[3] Simonet-Sales, A., *Types Abstraits et Bases de Données : Formalisation du Concept de Partage et Analyse Statique de Contraintes d'Intégrité*, Doctoral Thesis, University of Grenoble, France, Avril 1984.

[4] Simonet, A., Simonet, M., *Objects with Views and Constraints : from Databases to Knowledge bases*, Int. Conf. on Object Oriented Information Systems (OOIS), London, 1994.

[5] Guttag, J., Horning, J., *The Algebraic Specification of Abstract Data Types*, Acta Informatica, 1978.

[6] Delannoy, X., Simonet A., Simonet, M., *Database Views with Dynamic Assertions*, Third International Conference on Object Oriented Information Systems (OOIS), Springer Verlag Ed., London, December 1996. *(http://curie.imag.fr/~delannoy/perso/publications.html)*

[7] Delannoy, X., *Contributions to the study (i) of the tension between of consistency and confidentiality and (ii) to the classification of objets according to their history, in databases* , Doctoral Thesis, University of Grenoble, France, 1997. *(http://curie.imag.fr/~delannoy/perso/publications.html)*

[8] Aho, A., Ullman, J., *The Theory of Parsing, Translation and Compiling, Volume 1 : Parsing*, Prentice Hall series in Automatic Computation, 1972.

[9] Apple Corp. , *The NewtonScript Programming Language (version 2)*, Newton System Group, 1996.

[10] Chomicki, J., *Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding*, ACM Trans. on Database Systems, Vol.20, No.2, June, 1995.

[11] Gertz, M., Lipeck, U., *Deriving Optimised Integrity Monitoring Triggers from Dynamic Integrity Constraints*, Data & Knowledge Engineering, 1996.

Optimistic Concurrency Control Algorithms with Partial Abort for Firm Deadline Real-Time Database Systems

Piotr Krzyżagórski

Institute of Computing Science, Poznań University of Technology

Poznań, Poland

E-mail: piotr_k@cs.put.poznan.pl

Abstract

Optimistic concurrency control algorithms are widely proposed for firm deadline real-time database systems. The main drawback of the optimistic approach concerns the degradation of the system performance due to the restart based conflict resolution policy. We propose a new method - called partial abort - that is aimed to minimize the costs of restarts by saving the part of previously performed work, that is independent of conflicting data. The modifications of OPT-BC, OPT-WAIT and WAIT-50 algorithms are presented. Through simulation experiments we evaluate the performance of the modified versions and show that the improvement is significant.

1 Introduction

Modern computer systems that are involved in gathering large data volumes from the environment, providing durable storage and efficient access to these data are usually implemented using database technology. Some of them also process temporal data and have to deliver results in time. A database system designed to process transactions having timing constraints associated with them, and accessing data

items that values and validity change in time is called a real-time database system (RTDBS). These timing constraints are usually expressed as deadlines. If the deadlines are missed the information provided by the system could be of little value, hence the effects of controlling activities performed by the RTDBS may be wrong or even disastrous [11, 10].

The increasing number of real-time applications maintaining large volumes of data results in the growing importance of RTDBSs. The applications concern Computer Integrated Manufacturing (CIM), factory automation and robotics, workflow systems, military command and control, telecommunication and computer network management, telephone directory service systems, medical monitoring, aircraft control, traffic control, stock arbitrage systems, multimedia systems [2, 6].

There are two significantly distinct application environments: hard deadline RTDBSs and soft/firm deadline RTDBSs. Hard deadline RTDBSs are typical for life-critical applications, where all transactions have to be scheduled before their deadlines - the violation of the timing constraints is equivalent to a catastrophe. Transactions have soft deadlines if they are still valuable after missing their deadlines (although the value is reduced) and may run to completion. Firm deadline transactions have no value if the deadline is missed. The performance goals for soft deadline systems are to minimize total tardiness, mean tardiness and mean weighted tardiness of transactions, and for firm deadline RTDBSs to minimize the percentage of transactions that do not meet their deadlines.

The performance of any computer system depends on several factors such as the system architecture, the speed of its components (processors, disks, memory buffers), etc. However, for a given system configuration, its performance is mainly determined by the policy scheduling access to the system resources [6]. The scheduling techniques developed for hardware resources are not suitable for data objects access scheduling - usually called concurrency control. The policy significantly influence the overall system performance. Initially, real-time concurrency control algorithms were based on two-phase locking and its variants [1]. However, the possibility of deadlock, and long and unpredictable blocking time make meeting transaction timing constraints difficult. Alternative method is based on the optimistic approach that is deadlock-free and nonblocking. The main drawback of the optimistic approach concerns the degradation of the system performance due to the restart based conflict resolution policy.

In the paper, we propose a new approach that is aimed to minimize the costs of transaction restarting, by saving the part of previously performed work that is independent of the conflicting data. The approach is applied here to several widely used optimistic algorithms.

The organization of the remaining parts of the paper is as follows. In Section 2 special characteristics of real-time concurrency control is described. The details of the algorithms with partial abort and the discussion of their properties are presented in Section 3. Section 4 describes our RTDBS simulation model and presents experimental results. Finally, in Section 5 we summarize the main conclusions of the study.

2 Real-Time Concurrency Control

The activity of coordinating concurrent access of transactions to shared data in DBSs is referred to as concurrency control. Concurrency control protocols resolve data access conflicts in a manner that induces a serialization order among the conflicting transactions. If a conflict between transactions cannot be reconciled some transactions have to be aborted and restarted, and - as a consequence - the service already received from hardware resources is lost. It shows that data resource scheduling significantly influence the overall system performance.

So far, most of the concurrency control algorithms have been developed for firm deadline RTDBSs, that may provide a foundation for more complex soft deadline systems [7]. These algorithms are in general some extensions or combinations of traditional concurrency control techniques, i.e. two-phase locking (2PL), optimistic concurrency control (OCC) or timestamp ordering (TO), that guarantee a serialization order among conflicting transactions. They are generally based on the serializability criterion that is easy implementable and obviously correct. However, in RTDBSs the correctness of database operations depends also on the time the results are delivered.

Initially, most of real-time concurrency control algorithms were based on two-phase locking as the algorithm has been well studied and widely used in traditional

DBSs. Later on, optimistic concurrency control algorithms as alternatives to 2PL have been proposed for RTDBSs.

In the classic optimistic algorithm presented by Kung and Robinson [8], the execution of the transaction consists of three phases: read, validation, and write. During the read phase the transaction reads from the database and writes (prewrites) to its local workspace without any concurrency control restrictions. During the validation phase, a test is performed to check whether updates prepared by the transaction preserve the database consistency. After the test, the validated transaction either enters the write phase (its updates become permanent) and commits, or is aborted and restarted. The conflicts are detected relatively late (after the read phase) that degrades the system performance. In a variant of the above mechanism called OPT-BC a broadcast commit mechanism [9, 12] was incorporated. The validation test searches for currently executing (active) transactions that conflict with the validated transaction. In the case of conflicts the conflicting transactions are immediately restarted.

The optimistic algorithms described above use restart-based conflict resolution policy, that significantly wastes the system resources. After the restart all the results of previously performed work are lost, and the work has to be redone.

OPT-SACRIFICE algorithm [7] is a modification of the basic OPT-BC algorithm by incorporating a priority sacrifice mechanism. The validated transaction is restarted if it detects a conflict with any currently executing higher priority transactions. Wasted sacrifices, where transactions are restarted on behalf of other transactions that are later discarded, degrade the system performance.

OPT-WAIT algorithm [7] uses a priority wait mechanism. The validation of the transaction is delayed if there is a conflict with higher priority transaction(s). Wasted sacrifices are eliminated because the waiting transaction cannot be restarted by transactions that miss their deadlines and are later discarded, but the process of waiting increases the probability of new conflicts.

WAIT-50 algorithm [7] was proposed to diminish the disadvantages of OPT-WAIT algorithm by incorporating a wait control mechanism. Each transaction waits as long as half or more of the transactions that conflict with it have higher priorities, otherwise it is allowed to commit, while the conflicting transactions are

aborted.

The above considerations show that the significant drawback of optimistic algorithms is restart-based conflict resolution policy and - as a result - high cost of restarting transactions. We would like to propose a solution that is aimed to reduce this cost. Moreover, the approach can easily be incorporated into existing optimistic concurrency control algorithms.

3 Optimistic Concurrency Control Algorithms with Partial Abort

3.1 Basic Concepts

In the approach proposed here, an attempt is made to save the part of previously performed work that is independent of the conflicting data, and thus minimize the cost of restarting transactions. Each transaction T is an ordered sequence of read and write operations followed by the commit or abort operation. We denote these operations by $op_1(T)...op_n(T)$. Let's consider two conflicting transactions T_a and T_b. The conflict concerns operations $op_k(T_a)$ and $op_j(T_b)$. Let's assume that according to the conflict resolution policy T_a is restarted and the ordered sequence of already performed operations of T_a, i.e. $op_1(T_a)...op_{k-1}(T_a)op_k(T_a)op_{k+1}(T_a)...op_t(T_a)$, is lost. However, in the above sequence a subsequence of operations, i.e. $op_1(T_a)...op_{k-1}(T_a)$, is independent of the conflicting operation $op_k(T_a)$. In a particular case some of the operations $op_{k+1}(T_a)...op_t(T_a)$ are also independent of the conflicting operation $op_k(T_a)$. However, a detailed knowledge concerning semantics of these operations is necessary to prove this. Here, we assume that such information is unavailable so we restrict the set of conflict-independent operations to the sequence $op_1(T_a)...op_{k-1}(T_a)$.

The idea of the approach proposed here is, that instead of detecting a conflict, and - as a consequence - aborting T_a, the information specifying the conflicting operation $op_k(T_a)$ is sent to T_a. Transaction T_a after obtaining the information resumes execution starting from the operation $op_k(T_a)$ instead of $op_1(T_a)$. It means that the previously performed sequence of operations $op_1(T_a)...op_{k-1}(T_a)$ is not lost.

The approach may be applied to different concurrency control algorithms - both

optimistic as well as locking based. Of course, if (according to the concurrency control algorithm) T_a has set some locks, it now releases all the locks set by operations $op_k(T_a)...op_t(T_a)$. In the paper, we restrict our attention to modifications of optimistic concurrency control algorithms.

3.2 Algorithms with Partial Abort

Now we would like to present modifications of three optimistic algorithms, i.e. OPT-BC, OPT-WAIT and WAIT-50. The modifications concern including partial abort method into conflict resolution policy. The execution of each transaction consists of three phases: read, validation and write. During the read phase the transaction reads from the database, and prewrites to its local workspace. During the validation phase a test is performed that searches for currently executing transactions that conflict with the validating transaction. According to the conflict resolution policy used in the algorithms conflicting transactions have to be restarted. In this case we propose to inform these transactions not only about the existing conflict, but also about conflicting data element(s). Using the information, each conflicting transaction identifies the conflict-independent subset of previously performed operations, and resumes execution saving this subset. Then, the validating transaction enters the write phase - makes all its updates permanent and commits.

The modified versions of optimistic algorithms are shown below. In the algorithms, the term conflict set is used to denote the set of currently executing transactions that conflict with the validating one. There are two kinds of transactions in the conflict set: *CHP* (Conflicting Higher Priority) transactions that have higher priorities than the validating transaction, and *CLP* (Conflicting Lower Priority) transactions that have lower priorities than the validating one. The index *HPpercent* is the percentage of *CHP* transactions in the whole conflicting set [6].

3.3 OPT-BC/PaA - OPT-BC Algorithm with Partial Abort

IF transactions in conflict set **THEN**
    partially abort transactions in conflict set;
ENDIF
commit the validating transaction;

3.4 OPT-WAIT/PaA - OPT-WAIT Algorithm with Partial Abort

WHILE CHP transactions in conflict set **DO**
    wait;
partially abort transactions in conflict set;
commit the validating transaction;

3.5 WAIT-50/PaA - WAIT-50 Algorithm with Partial Abort

WHILE CHP transactions in conflict set **AND** HPpercent $\geq$ 50 **DO**
    wait;
partially abort transactions in conflict set;
commit the validating transaction;

3.6 Example

To illustrate the idea of partial abort let us consider a simple example in which the behavior of two algorithms, i.e. original OPT-BC algorithm and its modified version with partial abort - OPT-BC/PaA, are compared. For simplicity, we assume a single-processor memory-resident database system. The priority assignment policy is *earliest deadline first*. The processor scheduling procedure always selects the highest priority transaction for execution.

Given are two transactions A and B with the arrival time a, deadline d and data requests rd (read) and wr (written) shown in Table 1.

Table 1.

Transaction	a	d	rd	wr
A	0	150	x, y, z	y
B	30	100	y	y

The schedules produced by the algorithms are shown in Figure 1.

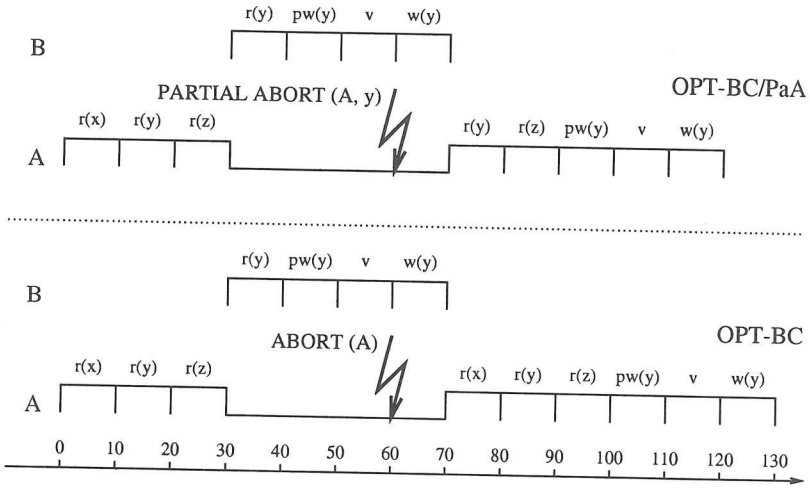

Figure 1: Example transaction schedules

Transaction A is the only job in the system at time 0. It gains the processor and reads data objects x, y and z. B arrives at time 30. Since B has an earlier deadline than A, B preempts A and begins to execute. It reads and prewrites data object y that is in conflict with the former read operation of A. During the validation phase, according to OPT-BC/PaA algorithm, B informs A about the conflict and specifies the conflicting data element - y. Then B makes the update permanent and finishes at time 70. A resumes at time 70. It identifies the conflict-independent subset of operations - $r(x)$, and begins execution starting from rereading data element y. It finishes at time 120. In OPT-BC transaction A is aborted and restarted at time 60. It finishes later - at time 130.

4 Simulation Experiments and Results

The performance of the concurrency control algorithms was tested in a model of a disk-resident multiprocessor real-time database system. Table 2 summarizes the workload and system parameters. Their values are the same as in [6] just to make

the results of the experiments easy comparable with the previous experiments. The load changes from 10 to 100 transactions per second.

Table 2.

Workload Parameter	Value	System Parameter	Value
minTransactionSize	8 pages	Database Size	1000 pages
maxTransactionSize	24 pages	numCPUs	10
writeProbability	0.25	numDisks	20
Deadline Assignment	DA1	pageCPU	10ms
		pageDisk	20ms

The basic performance metric in the experiment is $MissPercent$ that is computed as: $MissPercent = \frac{number\ of\ tardy\ transactions}{number\ of\ transactions\ arrived} \cdot 100$.

We also measure the average progress made by transactions before they are aborted or partially aborted due to data conflicts, and the average part that is saved according to the partial abort policy. These results are measured as the fraction of the total execution time (including disk write operation after validation). In the experiments, three original algorithms are compared with their modified versions. The results of the experiments are similar, so here we present only the results for one pair, i.e. OPT-BC and OPT-BC/PaA.

Figure 2a indicates that OPT-BC/PaA always outperforms OPT-BC - specially when the load is not very heavy. In this range of workload the part of previously performed work, that is saved by partial abort method, is significant. Figure 2b confirms this supposition. Under OPT-BC/PaA transactions are aborted a bit later. It shows that the system resources are better utilized, and the system throughput is higher. When the transaction is aborted it is possible to save about half of the work already performed. Of course, when the load is very high, transactions make little progress and the number of aborts increases. Most of the transactions miss their deadlines and are discarded from the system. The results were obtained assuming that the concurrency control overhead (including finding the conflict-independent subset of operations) was negligible compared to the CPU time for page processing

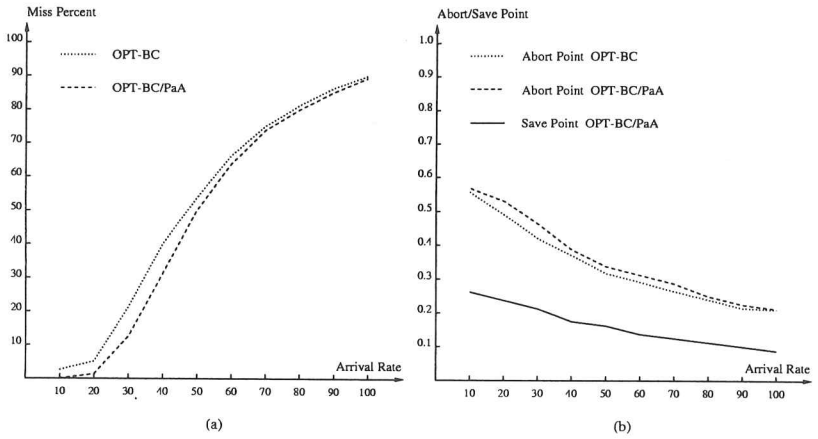

Figure 2: Sensitivity of a) Miss Percent , and b) Abort/Save Point to mean Arrival Rate

and therefore set to 0.0.

5 Conclusions

In the paper, a new approach is presented that is aimed to decrease the costs of restarting transactions. It seems specially suitable for real-time optimistic concurrency control algorithms that use restart-based conflict resolution policy. If restart occurs the method tries to reduce the costs saving the part of the previously performed work. The simulation results presented in the paper show significant improvement of the performance offered by the optimistic algorithms using partial abort method.

We must admit that the implementation costs are not considered here. The effectiveness of the approach is closely related to its physical implementation. In the description of the method presented above, we have assumed that after the partial abort each transaction can resume execution starting from the operation just preceding the conflicting one. This is possible if before each database operation all the data necessary for resuming the operation are stored. The cost of storing the auxiliary data may be significant. A tradeoff have to be found between this cost and possible effects. In a compromise solution the data would be stored periodically. The experiences with checkpoints technique used for recovery might be helpful [5].

The performance can be further improved using knowledge concerning data semantics. There are some experiences with synchronization algorithms for the abstract data types in distributed database systems that use a transaction model extended to include operations that are specific to a particular application [3, 13]. Moreover, some initial attempts have been made to adopt the object-oriented data model for real-time databases. This data model has rich data semantics that can be utilized in real-time transaction processing [4].

We hope that further investigations help precisely evaluate the advantages of the method.

References

[1] Abbott, R., and Garcia-Molina, H., Scheduling Real-Time Transactions: a Performance Evaluation, *ACM Transactions on Database Systems*, vol. 17, no. 3, pp. 513-560, September 1992.

[2] Buchmann, A.P., McCarthy, D.R., Hsu, M., and Dayal, U., Time-Critical Database Scheduling: A Framework For Integrating Real-Time Scheduling and Concurrency Control, Data Engineering Conference, February 1989.

[3] Cellary, W., Gelenbe, E., and Morzy, T., Concurrency Control in Distributed Database Systems, North Holland Pub. Co., 1988.

[4] Di Pippo, L.B.C., and Wolfe, V.F., Object-Based Semantic Real-Time Concurrency Control, Proc. 14th Real-Time Systems Symp., 1993.

[5] Gray, R., and Reuter, A., Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993.

[6] Haritsa, J.R., Transaction Scheduling in Firm Real-Time Database Systems, Computer Sciences Technical Report #1036, University of Wisconsin-Madison, 1991.

[7] Haritsa, J.R., Carey, M.J., and Livny, M., Data Access Scheduling in Firm Real-Time Database Systems, *The Journal of Real-Time Systems*, no. 4, pp. 203-241, 1992.

[8] Kung, H., and Robinson, J., On Optimistic Methods for Concurrency Control, *ACM Transactions on Database Systems*, vol. 6, no. 2, June 1981.

[9] Menasce, D., and Nakanishi, T., Optimistic Versus Pessimistic Concurrency Control Mechanisms in Database Management Systems, *Information Systems*, vol. 7, no. 1, 1982.

[10] Özsoyoğlu, G., and Sondgrass, R.T., Temporal and Real-Time Databases: A Survey, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, no. 4, 1995, pp. 513-532.

[11] Ramamritham, K., Real-Time Databases, *Distributed and Parallel Databases*, Vol. 1, no. 2, 1993, pp. 199-226.

[12] Robinson, J., Design of Concurrency Controls for Transaction Processing Systems, Ph.D. Thesis, Computer Science Department, Carnegie Mellon University, 1982.

[13] Weihl, W.F., Data Dependent Concurrency Control and Recovery, Proc. 2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing, Cambridge, 1983.

A PREWRITE TRANSACTION MODEL

Sanjay Kumar Madria
School of Computer Science
University Sains Malaysia
11800 Minden, Penang, Malaysia
skm@cs.usm.my

Abstract

In this paper, we introduce a transaction model where a prewrite operation occurs before an actual write operation is performed on data objects. A prewrite operation announces the value that the data object will have after the commit of the write operation. A prewrite operation does not change the state of the data object but only makes available the value that the data object will have after its associated final write is performed. Once all the prewrites of a transaction are announced, the transaction executes a pre-commit operation. After the pre-commit, a read transaction is permitted to read the announced prewrite values even before the other transaction has finally updated these data objects and committed. Therefore, our algorithm allows non-strict executions with no cascade aborts and at the same time increases the potential concurrency as compared to the algorithms that permit only read and write operations on the database objects.

1. Introduction

In a database system, users access database by executing read and write operations. A read operation on a database object does not conflict with another read operation since neither of them modifies the data object. A write operation on the otherhand conflicts with both read and write operations. A concurrency control algorithm [1,2,3,4] is needed to control interleaving of conflicting operations which otherwise violates the consistency of the database.

Many concurrency control protocols are based on the notion of locks [1,2,5,6,7] where a data object in the database can be accessed only after a lock on that object has been acquired for the duration of the read or writes. That is, a read (write) operation can be executed only after a read(write)-lock is obtained. After acquiring the lock, the transaction executes its operation and then may release the locks. Since read operations

do not conflict, many read transactions may share the read-lock on an object but sharing is not permitted if one of the lock is a write-lock. The above design is more suitable for database management systems that support short-duration transactions that read and write data objects for a short period of time. However, in case of long-duration transactions [8,9,10,11] which access large volume of data objects, the concurrency control algorithms based on above protocols suffer from performance degradation. These types of transactions usually occur in engineering design or software development process. Due to isolation requirements, the data items held by long transactions can not be released until the transaction commits. Therefore, once the transaction acquires a write-lock, the other transactions have to wait for very long before they get the lock. Thus, if short-duration transactions want to read access some data items held by a long-lived transaction, it will end up waiting for the long-lived transaction to commit. Also, if a transaction is considered as a basic unit of work, a significant amount of work may be lost in case of a failure. Also, in a real time system, it is desirable that a read transaction should not wait for very long in order to access the data objects. Therefore, it is desirable to make the response of a system fast particularly for read-only transactions. Also, the system should not delay short-duration transactions due to the presence of long transactions.

The transaction models developed for early database applications [1,2,4] fall short of meeting the requirements of these new transactions has been shown in [12,13]. Also, some of the proposed algorithms [14,15] to manage long-duration transactions do not rely on serializability [1,2] and therefore, use different correctness criterion.

In this paper, we incorporate a prewrite operation [16,17] before an actual write operation is executed on database objects to increase the potential concurrency. A prewrite operation announces the value the data object will have after its associated write is performed. It actually does not update the state of the data object but only makes available the value a transaction intends to write in future. Prewrites are kept in a private work space of each data object. Once all the prewrites of a transaction are performed, the associated transaction executes a pre-commit operation. After the transaction has finally written all the data objects (for which the prewrite values have been announced), it can finally commit. Therefore, the introduction of prewrite values helps transactions in

reading the values of the data objects before the other transaction has actually updated the state of those data objects. The prewrite values are made available for reads after the associated transaction has executed a pre-commit but before it has been actually committed. Hence, prewrites allow non-strict executions without the fear of cascading aborts. Thus, prewrites increase concurrency as compared to the environment where only read and write operations are allowed.

Once a transaction has announced a pre-commit, it is not allowed to abort. This is due to the following two reasons. First, it will help in avoiding cascading aborts since the prewrites have been made available before the transaction has finally committed. Second, it is desirable for a long-duration transaction that it does not lose all its work at finishing stage in case there is an abort or a system failure. To accomplish this, a pre-commit operation is to be executed only after all the prewrite log records are stored on stable storage. Once write operations start, each write log is also stored on stable storage. Thus, in case of a failure after pre-commit, there is no need of executing rollback (undo) operations. The recovery algorithm has to, at the most redo, those operations (using prewrite and write logs) whose effects are not there on stable storage. The failed transaction then can restart from the failure point onwards. If a transaction aborts before executing all prewrite operations, it is rolled back by discarding the prewrites. The detailed recovery algorithm appears in [17].

By introducing prewrites, short-duration transactions can read the value of a data item held by a long transaction before its commit. Therefore, using prewrites, one can manage a system consisting of short and long transactions without causing delay for short-duration transactions. Prewrites may be useful in engineering design applications [8,18] where transactions are usually very long. Prewrites may also increase the throughput of the system by making the response of the system faster. This can be established only after implementation of the algorithm as prewrites might introduce some overheads in the system due to an extra lock. However, we think that the increase in overheads will not compensate the increased concurrency and efficient management of long and short transactions within the same system.

In our algorithm, the existence of a prewrite is only visible to the scheduler and data manager (DM), not to user transaction. The user transaction consists of read and write operations for the data objects it wants to access. Once a transaction is submitted to the system, the system analyzes the received transaction. If the transaction has some write operations, the system will announce prewrite operations for those data objects. If the transaction has a read operation, the system will return the corresponding prewrite value (if any) from the prewrite-buffer otherwise it returns the value from the write-buffer.

We have used two phase locking [1,2] to control concurrent operations.

2. Prewrite Transaction Model

The main features of our transaction model are :

- Each transaction has a prewrite operation before a write operation. A prewrite operation makes visible the value the data object will have after the commit of the transaction.
- Once all the prewrites have been processed, the transaction pre-commits. A pre-committed transaction's results are visible before the final commit. This minimizes the blocking of other transactions and increases concurrency.
- A pre-committed transaction is guaranteed to commit. This feature of our model avoids an undo or compensating transaction, which is costly.
- A pre-read returns a prewrite value whereas a read returns a write value.
- The transactions are serialized based on their pre-commit order.

The formal definitions of a prewrite operation and a pre-committed transaction are as follows.

Definition 1: A prewrite operation announces the value that the data object will have after the commit of the corresponding write operation.

Definition 2: A transaction is called pre-committed if it has announced all the prewrites values and read all the required data objects, but the transaction has not been finally committed.

Definition 3: A pre-read returns a prewrite value of the data object whereas read returns a result of the write operation. A pre-read becomes a weak-read (in case the data objects involved are not simple) if it returns a prewrite value even though the transaction who

announced the last prewrite has been finally committed. However, this weak-read should not to be aborted.

The transaction model using prewrite operations has the following features:

- Each prewrite operation makes visible the value the transaction will eventually write. Prewrites have different semantics in different environments. For simple data objects, the prewrite and write values match exactly. For database files, the prewrites may only contain primary-key values and the new values of the fields of records. In case of design objects, prewrites may represent a model of the design. However, the final design released for manufacturing may differ from prewrite design of the model. For example, the final design may have a different colour shade than the prewrite design model. In case of a document object, the prewrite may represent an abstract of the detail document.
- Once the required data objects are read or pre-read and prewrites are computed, the transaction pre-commits. A transaction is required to read or pre-read all the required data objects before pre-commit because after a transaction releases a lock for a prewrite operation, it can not get a lock for read operation due to the condition of two phase locking [1]. After a pre-commit, the prewrites are made visible to other transactions for processing. Initially, prewrites in our model are kept in the transaction's private workspace. Once the transaction pre-commits, they are posted in the prewrite-buffer. The data objects are physically updated on the disk the pre-commit operations. The prewrites are handled at the transaction manager level and physical writes are handled at the data manager level.
- The transactions commit order in the serializable transactions execution history is decided at the order of pre-commit action.
- Our concurrency control algorithm is to be executed in two servers: For controlling pre-read (i.e., read of prewrite value) and prewrite operations at TM server and read (i.e., read of write value) and write operations at DM server. Since prewrite values are made publicly visible after pre-commit, the lock-type held by the prewrite operation is converted to the lock-type for write operation after pre-commit. The lock acquired for a prewrite operation is not released after pre-commit because the two phase locking

[1] does not allow a transaction to acquire a lock after the transaction has released some locks.

- A transaction is not allowed to abort after pre-commit. The prewrite provides non-strict execution without cascading aborts. In figure 1, T1 and T2 are two subtransactions where pw(x), w(x), pr(x), and r(x) are the prewrite value, write value, pre-read value and read-value respectively, for the data object x. The transaction T2 commits before T1. In case T1 aborts after T2 commits, there will not be a cascading abort. Since our model does not need undo-recovery from transaction aborts [17], no compensating transaction is to be executed. In case the pre-committed transaction is forced to abort due to system dependent reasons such as system crash, the transaction restarts on system revival.
- Our model relaxes the isolation property as the prewrites are made visible to others after pre-commit but before the final commit of the transaction. Also, durability of prewrites is guaranteed at the pre-commit point.

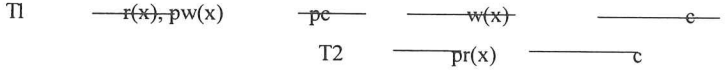

Figure 1. Two Concurrent Transactions

3. Concurrent Operations

In our transaction model, a pre-read operation and a write operation can take place concurrently. Similarly, a read can be executed concurrently with a prewrite. A prewrite operation can also be executed concurrently with another write operation since prewrites are managed at the transaction manager level whereas the writes are performed at the data manager level. However, there are some interesting cases:

Case 1: Suppose a pre-read is currently being executed and at the same time, another transaction who has announced the prewrite values before finally commits (final updates are performed) (see figure 2(a)).

In this case, the pre-read will return a prewrite value which might be different than the last write value. For example, if the data object x is the simple data object, the read transaction T2 can commit as the prewrite and write values will be same. In case x is a

design object, the system designate the read T2 as a weak- read. The transaction T2 can resubmit its read request later if it needs the latest complete model of the design.

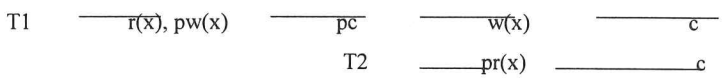

Figure 2 (a). The Situation in case 1

Case 2: In table 1, observe that a read operation is compatible with a prewrite. Consider a case where a read transaction commits after the transaction who announced the prewrite operation has been pre-committed.

The read in this situation will return an old value. However, this is not a significant problem because the transaction can still be serialized. For example (see figure 2(b)), transaction T2 returns a write value, however it commits after T1 has pre-committed. The transaction T2 can be serialized before T1.

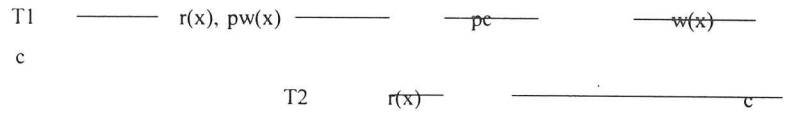

Figure 2(b). The Situation in case 2

4. Locking

We develop a concurrency control algorithm to control the conflicting operations in our mobile transaction model. The operation-compatibility matrix of the various operations is given in table 1. The concurrency control algorithm is executed in two phases at two places. In the first phase, the concurrency control for controlling prewrite and pre-read operations is performed at the transaction manager level. In the second phase, the concurrency control for controlling write and read operations (to access write values) is performed at the data manager (DM) level mainly since the data managers are accessed only while performing updates on the databases.

	Pre-read	Read	Pre-write	Write
Pre-read	Yes	Yes	No	Yes
Read	Yes	Yes	Yes	No
Pre-write	No	Yes	No	Yes
Write	Yes	No	Yes	No

Table 1. Operation Compatibility Matrix

We use read-lock for read, pre-read-lock for pre-read, prewrite-lock for prewrite, and write-lock for write operations, respectively. A prewrite-lock conflicts with other pre-read- and prewrite-locks, however it does not conflict with read- and write-locks. A prewrite-lock can not be released after pre-commit as the transaction has to still get a write-lock for final updates. A prewrite-lock acquired by a pre-committed transaction is converted to a write-lock provided no other transaction holds the conflicting locks. Once a prewrite-lock is updated to a write-lock, the same transaction can not acquire any other lock. However, pre-read-locks can be acquired by others to access prewrite values.

There will be no a deadlock involving the transactions which are pre-committed. This is due to the fact that prewrite- and write-locks are acquired in an ordered fashion so deadlocks will occur only at the time of acquiring prewrite- or read-locks. Thus, a pre-committed transaction will not be aborted due to the deadlocks.

In case of replicated data, the number of locks to be acquired in our algorithm depends on the particular replication algorithm used. The two main replication algorithms used are majority consensus and read-one-write all (ROWA) [1]. In majority consensus algorithm, locks are acquired on majority of sites whereas read-one-write-all (ROWA) requires lock on all the sites. In case read/write ratio is less, ROWA is preferred otherwise majority consensus will be preferred.

5. Conclusion

In this paper, we have introduced a prewrite transaction model. In our transaction model, a prewrite operation precedes each write operation in the transaction. A prewrite operation declares in advance the value the data object will supposed to have after the write operation is executed. A read operation can read the prewrite value before the value was actually written. Our algorithm increases the concurrency as compared to other algorithms permitting only read and write operations. The concurrent operations are controlled by the two phase locking algorithm. We have used this model for mobile computing. We are working on the workflow transaction model using prewrites.

References

[1] P.A. Bernstein, V. Hadzilacose and N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, Reading, MA,1987.

[2] C.H. Papadimitriou, The Theory of Database Concurrency Control, Computer Science Press, Rockville, MD, 1986.

[3] W.E. Weihl, Commutativity-based Concurrency Control for Abstract Data Types, IEEE Transactions on Computers, 37, No.12, pp. 1488-1505, Dec.1988.

[4] H.T. Kung and J.T. Robinson, On Optimistic Methods for Concurrency Control, ACM Transactions on Database Systems, 6, No.2, pp. 213-226, 1981.

[5] M. Yannakakis, Serializability by Locking, Journal of ACM, 31, No.2, pp. 227-244, 1984.

[6] K.P. Eswaran, J.N.Gray, R.A. Lorie and I.L. Traiger, The Notion of Consistency and Predicate Locks in Database Systems, Communication of ACM, 19, No.11, pp. 624-633, 1976.

[7] H.F. Korth, Locking Primitives in Database Systems, Journal of ACM, 30, No.11, pp. 55-79, 1983.

[8] F. Bancilhon, W. Kim, and H.F. Korth, A Model of CAD Transactions, in Proceedings of the 11th International conference on Very Large Databases, VLDB Endowment, pp. 25-33, 1985.

[9] U. Dayal, M. Hsu, and R. Ladin, Organizing Long-running Activities with Triggers and Transactions, in Proceedings of the ACM SIGMOD international Conference on Management of Data, ACM, New York, pp. 204-214, 1990.

[10] H.F. Korth and G. Speegle, Long Duration Transactions in Software Design Projects, in 6th IEEE International Conference on Data Engineering, New York, pp. 568-674, 1990.

[11] H.F. Korth, W. Kim, and F. Bancilhon, On Long-Duration CAD Transactions, Information Science, 46, pp. 73-107, Oct.1990..

[12] C. Beeri, P.A. Bernstein, and N. Goodman, A Model for Concurrency in Nested Transaction System, Journal of ACM, Vol.36, No.2, pp.230-269, 1989.

[13] G. Weikum, Principles and Realization Strategies of Multi-level Transaction Management, ACM Transaction on Database Systems, Vol.16, No.1, pp.132-180, 1991.

[14] H.F. Korth, and G. Speegle, Formal Model of Correctness without Serializability, in ACM SIGMOD International Conference on Management of Data, NewYork, pp.379-386,1988.

[15] H. F., Korth, G. Speegle, Formal Aspects of Concurrency Control in Long-duration Transaction Systems using NT/PV Model, ACM Transactions on Database Systems, Vol.19, No.3, pp.492-535, Sept.1994.

[16] S.K. Madria, Concurrency Control and Recovery Algorithms in Nested Transaction Environment, Ph.D. Thesis, Indian Institute of Technology, Delhi, India, 1995.

[17] Madria, S.K., Maheshwari, S.N, Chandra, B., Bhargava, B., Crash Recovery Algorithm in an Open and Safe Nested Transaction Model, 8th International Conference on Database and Expert System Applications (DEXA'97), France, Sept.97, Lecture Notes in Computer Science, Springer Verlag, Vol. 1308, 1997.

[18] W. Kim, R. Lorie, D. Mcnabb, and W. Plouffe, A Transaction Mechanism for Engineering Design Databases, in Proceedings of the 10th International Conference on Very Large Databases, VLDB Endowment, pp. 355-362, 1984.

An Efficient Conflict Detection Scheme for Concurrent Temporal Transactions

Bong-Ok Ha and Yoo-Sung Kim

Department of Computer Science & Engineering
INHA University
Inchon 402-751, KOREA
yskim@dragon.inha.ac.kr

Abstract

An efficient conflict detection scheme for Temporal DataBase Systems(TDBSs) is proposed. In TDBSs, a temporal transaction may access more data records than a transaction in traditional database systems because a TDBS usually manages both the historical versions and the current version of a data item. Hence, a concurrency control subsystem should be able to correctly and efficiently detect actual conflicts among concurrent temporal transactions while the cost of detecting conflicts is maintained in low level without detecting false conflicts which cause severe degradation of system throughput.

1. Introduction

Traditional database systems may be not appropriate to some applications such as hospital patient information systems since hospital patient information systems should manage not only current information but also historical information of patients. Instead of traditional database systems, temporal database systems(TDBSs) are appropriate to such applications, since TDBS allows users to access not only the current snapshot information but also the historical information of temporal database. Hence, TDBS has to manage the historical database in addition to the snapshot database of real world modelled into temporal database.

Since a TDBS may have several versions of a data item, the number of data items accessed by a temporal transaction must be larger than that in traditional database sys-