Bulletin of the Technical Committee on

Data Engineering

March, 1994 Vol. 17 No. 1

IEEE Computer Society

Letters

Special Issue on TP Monitors and Distributed Transaction Management

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

Letter from the Editor-in-Chief

This marks the beginning of my third year as editor of the Bulletin. During this time, the Bulletin published issues that continue its role as a collection point for describing the state-of-the-art in some technical area of interest to the database community. I have been gratified by the positive response that the resumed Bulletin has received.

I am happy to report that Eliot Moss will be joining the Bulletin as an Associate Editor. Eliot has a well deserved reputation for his work in object-oriented databases, and is this year's program chair for OOPSLA. Eliot launched his career with the introduction of nested transactions and continues his work in transaction models and mechanisms. He is currently a professor at the University of Massachusetts, Amherst.

Establishing the publication procedures has proven to be very time-consuming. The TC membership list has been re-constituted, with most members now electronically enrolled and with both postal and email addresses. The email addresses are very important as we strive to produce the Bulletin in as inexpensive a way as possible. Email permits us to notify TC members promptly and without postage cost. In addition, the publication of the Bulletin electronically has been firmly established. This is essential as this is the least expensive way for us to distribute the Bulletin, and may eventually become the dominant way.

The postscript issues of the Bulletin produced several problems. The files are large and not all mailers can handle them. Postscript is not a consistent standard and particularly postscript figures can cause problems. I have tried to respond promptly to all reported problems. If you should have a problem, please report it to me at `lomet@crl.dec.com`. Not all problems are solvable (e.g. mailers with message size restrictions) but some may be and I would like to understand how successful this enterprise is.

We will continue to distribute hardcopy of the Bulletin to TC members so long as we have budget to cover the cost. We expect that at least three issues this year can be distributed. We will have to watch whether the fourth is also possible. Next year, hardcopy distribution may be more limited. We are still awaiting word on whether we can establish a paid subscription for hardcopy. This still seems to be the only long term solution for covering printing and mailing costs. I hope to have more news on this in the next issue.

Finally, about this issue. I am delighted that Mei Hsu convinced Ron Obermarck to edit the issue. Ron is uniquely qualified to assemble information about the commercial state of transaction monitors and distributed transaction managment, having been a leading industrial pioneer in this area. Like the December issue, the current issue furthers my goal of disseminating to our community information about current commercial products exploiting a particular technology. So I would like to thank Ron (and Mei) for a fine job on this. I am sure that there is much we can all learn from the contributions presented here.

David Lomet
Editor-in-Chief

Letter from the Special Issue Editor

The traditional TP Monitor serves important roles in industry where economy of scale dictate centralized operation. It has also evolved to support open distributed transaction processing (DTP) environments. There are vigorous new entries in the field. DTP is supported by TP monitors, and by client-server environments. Open systems, additional requirements imposed by workflow, real-time processing, and the explosion of casual connectedness, all show that the first quarter-century is just a beginning.

Susan Malaika takes us from 1969 to the present, and beyond with IBM's CICS in "A Tale of a Transaction Monitor". The product evolution is traced from its beginnings on the IBM/360 to its current use as client or server on a wide range of platforms, both IBM's and others.

Eric Newcomer presents Digital Equipment Corporation's ACMS and STDL, one of the pioneers in the control of distributed transaction management. Opening this work has resulted in the adoption of a standard for transactional RPC. STDL is accepted as a portable, high-level language to define the execution of transactions under the control of a TP monitor.

Dieter Gawlick discusses methods developed to handle exceptionally high transaction volumes and the availability requirements they imposed on the IMS/VS TP Monitor. The IMS Fast Path Feature and Extended Restart Facility (XRF) met those requirements. Emerging workflow management systems must exhibit comparable availability and performance.

C. Mohan and Dick Dievendorff describe recent work in distributed commit protocols and recoverable messaging and queuing. The commit protocols have been implemented in IBM's DB2 and the System Network Architecture. While DTP focuses on connected clients and servers, queuing and messaging support applications where the server and client may be disconnected.

Gary Bundell and Gene E. Trivett remind us that what we call "legacy systems" are the "work-horse systems" of today. They then move to transaction processing in real-time systems, and discuss aspects which require support at the operating system level.

Juan M. Andrade, Mark T. Carges and M. Randall MacBlane discuss the requirements driving open OLTP in today's market and an overview of the TUXEDO products designed to meet those requirements. With an emphasis on open DTP, TUXEDO's Enterprise Transaction Processing System supports open standards.

Hector Garcia-Molina and Kenneth Salem show a firm relationship between transaction processing and the low-level functions needed in a workflow management system. Without requiring extension of the transaction model, they postulate reliable low-level transactional services needed to support workflow management.

Patrick O'Neil, Mohsen Al-Ghosein, David Vaskevitch, Rick Vicik, and Laura Yedwab, present Microsoft's vision of the distributed database and DTP environment. Supporting transactional database access from the home and auto is an exciting prospect that poses an interesting array of problems.

I wish to thank the contributing authors for their support and patience. Special thanks to my colleague Mei Hsu. Without her long hours of effort in addition to her knowledge, this special issue would not have been.

Ron Obermarck
Digital Equipment Corporation
Palo Alto, CA
email: ronober@pa.dec.com

A Tale of a Transaction Monitor

Susan Malaika
Transaction Systems,
Mail Point 197, IBM Hursley Park,
Hampshire SO21 2JN, United Kingdom
malaika@vnet.ibm.com

1 Introduction

In order to tell the story of CICS, it is simpler to begin by explaining what CICS is and thus the next few sections describe the CICS products of 1994. For more general information on IBM's transaction monitors, see [IBM4]. For more general information on CICS see [Gera93, Yela85]. CICS runs on a number of operating systems which are listed at the end of this article. In addition, there are a number of CICS-like products that have been developed independently of IBM, which are an indication of CICS's success. An essential ingredient of a CICS-like product is usually the run-time environment to execute COBOL programs that include CICS commands. This article concentrates on some of the CICS products from IBM which are all developed at the IBM Hursley laboratory in the county of Hampshire in the south of the United Kingdom. (In IBM, the word "laboratory" denotes the place where software and hardware is designed).

2 The functions of CICS

CICS is software that provides a suitable environment to run and manage programs that are executed repetitively by many users with various device types, e.g. mobile computers, workstations and terminals. The programs access and update shared data and both the programs and the data may be distributed. CICS supports a number of functions which programmers use via the CICS commands, also known as the CICS application programming interface (API), and often the same CICS command can access local or remote resources without changing the application program.

The CICS API is made up of about a hundred commands with hundreds of keywords and it is summarized in [IBM5]. The CICS API and the way CICS systems communicate with each other is monitored closely at Hursley by the CICS Architecture Board, to ensure compatibility when extensions are made. Here is an example of the CICS API:

- EXEC CICS LINK PROGRAM(Hursley) COMMAREA(sun-shine)

This command causes control to be passed to a program called "Hursley" passing any parameters in the data area "sun-shine". Information can be returned in "sun-shine" to the calling program. The program Hursley can be on the same or different CICS system, e.g. the client system could be CICS OS/2 and the server system could be CICS/6000.

These are the CICS API functional areas, as listed in [IBM5]: abend handling (processing abnormal terminations), APPC (Advanced Program to Program Communication), authentication, BMS (Basic Mapping Support), diagnostic services (dump and trace), environment services, exception handling, file control, interval control, journal control, monitoring program control, storage control, syncpoint (commit, rollback), task control, temporary storage, terminal control and transient data.

CICS application programmers have many options for storing information as an alternative to a database, e.g. in a CICS managed file or queue which can be recoverable, or in memory which can be shared or specific to the application instance. The lifetimes of the memory can vary considerably. In general, CICS gives the application program the illusion that it is running alone, e.g. CICS copies any program variables that are needed for multiple concurrent instances of a program within a single CICS server system. This support makes it relatively straightforward to write server applications. Currently, the third generation programming languages that are considered most important to CICS are C and COBOL, and they are both supported wherever CICS is ported by IBM. C++ can also be used in some of the products.

CICS systems programmers or administrators define the attributes of resources such as files to the CICS system. This includes the location of the resources. In many cases there are interactive, off-line and programmable ways for supplying the definitions.

3 The characteristics of a CICS application

CICS supports the chained transaction model. Thus, when a CICS application starts running, it finds itself within a transaction (in CICS terminology within a "logical unit of work") and the application does not issue a "begin transaction" request. When an application terminates and returns control to CICS, CICS issues a commit request (a SYNCPOINT in CICS terminology) on behalf of the application. If the application itself issues a commit, then CICS automatically begins another transaction for the application after processing the commit. Most CICS applications do not contain commit requests and rely on CICS to take care of recovery considerations, which is an advantage of chained transactions.

A CICS application is typically made up of many programs, screens, layouts for work-areas, queues, files and databases. Associated with this may be one or more "transaction identifiers" or "transaction codes". A transaction code consists of four characters that when typed in, determines the first application program to be executed. Over the years, the CICS transaction code has accumulated many attributes, in addition to the name of the first program. Examples include security and execution priority. Thus it is often convenient to think of the transaction code as a grouping mechanism for a set of characteristics, that can be associated with a CICS execution unit at run-time. Selecting and changing transaction codes at the appropriate times can be vital to achieving desirable behavior in a CICS system.

Usually each user interaction causes a number of application programs to be executed within one CICS execution unit. A general recommendation is that CICS applications should be "pseudo-conversational". This means that the application should terminate instead of waiting for the human user to respond, in order to conserve computer resources, e.g. storage and execution units. The application can name the transaction code to be used next time that same device initiates work in that CICS system. The application can also save some context in a scratch pad which CICS presents to the program associated with the next transaction code. In CICS terminology, a program that waits for a human to respond is called "conversational".

On CICS/ESA and CICS/VSE, programmers have always been discouraged from using native operating system requests within their CICS programs. Indeed, they could only use the CICS API, and APIs that were specifically supported, e.g. SQL. The reason for the guideline was that one program could issue a request which would hold it up and all the applications waiting to run after it, whereas the CICS supported APIs would ensure that no waiting occurred. On the workstation platforms each CICS application runs within its own process with its own execution thread, so the impact of delaying one of the processes is less significant, although it could be noticeable. Thus, most APIs are available to CICS workstation applications.

Another piece of CICS terminology is "quasi-reentrancy", which means that a CICS application may give up control of the processor only when executing within a CICS command. Quasi-reentrancy does not apply on CICS OS/2 nor CICS/6000, as the application can be interrupted at any point during its execution by the underlying operating system which uses preemtive scheduling.

4 And eventually "A Tale of a Transaction Monitor"

4.1 The Des Plaines days

During the 1960s many public utilities (gas, water, electricity etc) in the US were independently writing applications to manage their customer records on IBM's System 360 operating system. It was observed that many elements of these systems were common, and a team of six people was set up at IBM in Des Plaines (near Chicago) to develop what became CICS [IBM92]. The product was initially announced as "Public Utility Customer Information Control System" (PUCICS) in 1968 [IBM68]. It was re-announced the following year at the time of availability as CICS [IBM69] and it included the following components: Program, storage and task management; Terminal management; File, temporary storage and transient data management; Signon/Signoff; Master and supervisory terminal function; System statistics and system shutdown.

One important theme of the early CICS systems was their support for various terminal types and the relative ease with which the application programmer accessed these terminals. The first CICS release was efficient in its use of the computer both in instruction path-length and storage. For example the basic CICS needed 15,000 bytes of memory to run excluding the memory needed to hold the CICS resource attributes, known as the CICS tables. (The size of the CICS client for PC DOS in 1994, is around 60,000 bytes to provide access into a server CICS system). The application programs had to be written in 360 Assembler, and access to CICS services such as shared files, was provided through assembler macros. Support for PL/I and COBOL was then introduced and CICS made a separate copy of the application's working storage for each instance, but used a shared copy of the program itself. The application programmers still used 360 Assembler macros to access CICS services.

4.2 A tale of two transaction monitors

Also in the 1960s, the Information Management System (IMS) transaction monitor and database manager was developed by IBM in collaboration with Rockwell [Graf83]. Rockwell was one of the contractors for the Apollo space program, and one of Rockwell's requirements was to manage the two million parts needed for a single spacecraft, which IMS helped to satisfy. CICS was designed to run on a System/360 with 64K of memory, whereas IMS was intended to run on a System/360 with 256K of memory. It is interesting to observe that the process structure of CICS on AIX and OS/2 resembles that of IMS more than mainframe CICS, e.g., the applications run in processes (address spaces) separate from one another, and from the CICS system software. Another difference between IMS and CICS is that in CICS the application program communicates with a terminal directly, whereas in IMS the application and terminal communicate through a queue. IMS also supports batch execution, e.g. by providing mechanisms to take checkpoints and to restart batch applications.

4.3 The Palo Alto days

In 1970, the CICS team moved to Palo Alto, California, and various extensions were incorporated into CICS. e.g.:

- Support for DL/I (IMS/DB): When the support first became available, only one user could access an individual database view (PSB) at a time. This was later improved to support multiple concurrent users. More flexibility was introduced through making the program able to select the view dynamically via API, whereas in IMS/TM (formerly IMS/DC) the view is dependent on the transaction code used.

- Basic Mapping Support (BMS): This function allowed programs to communicate with devices without having to understand the controls of the individual devices.

- Disk Operating System (System 360): CICS was modified to run on this operating system (which later became VSE). The support for CICS for DOS was moved to IBM in Germany, and later it moved to Hursley.

- Virtual storage (paging) operating systems: CICS's storage management algorithms were modified to operate better in a paging environment.

4.4 The Hursley days

In 1974 CICS product development was moved to the Hursley laboratory after a six month hand-over period. At about the same time, PL/I compiler development was moved from Hursley to the US. Thus, a number of the programmers working on CICS in the mid 1970s had a PL/I background.

4.5 EXEC CICS form of the API

Some ex-PL/I implementers thought it necessary to replace the 360 Assembler macros used to access CICS services by something better. Three small independent projects ran in parallel to define a replacement. There were various considerations and objectives, e.g.:

- A preprocessed API from which native language calls are generated which would be simpler for a COBOL or PL/I programmer. For example, keywords could be specified in any order in contrast with a call API.
- It was undesirable to reserve any special words in the enclosing programming language.
- Using macros meant that applications were placing and looking for data in CICS's internal structures.

Eventually the project that introduced the EXEC CICS API was selected. Each CICS request is prefixed by two consecutive words (EXEC CICS) that are easy for a preprocessor to detect. Also, in most programming languages, two consecutive variable names do not constitute valid syntax. The choice of the word "EXEC" appears to have been influenced by IBM 360 job control language. At the time it was also assumed that other similar APIs would be integrated with CICS, which later happened in the form of EXEC DLI to access IMS databases, and EXEC SQL to access relational databases. Many years later EXEC SQL became the standard way to prefix requests to many relational databases. In a subsequent release of CICS, a debugging utility was introduced to help programmers step through the CICS API at run-time and to allow them to modify the input and output parameters.

4.6 Distributed processing

During the late 1970s and early 1980s various forms of distributed processing were introduced into CICS, e.g.:

- Transaction routing: Routing whole CICS applications to a system different from the one where the application was initiated.
- Function shipping: Routing file and queue access and update requests to systems different from the ones where the application is executing. The way function shipping was implemented, using a "mirror program" to represent the client application in the server system was one of the first patents for CICS. It was applied for in 1978.
- Distributed Transaction Processing: Accessing remote CICS applications from other CICS or non-CICS programs.

At first there were just two protocols supported for distributed processing. One was for communication between multiple CICS systems in a single instance of the operating system, called Multi-Region Option. The other was for communication across SNA and known as Intersystems Communication. The application is not sensitive to the protocol. These distributed requests and others are supported across TCP/IP and NetBIOS on some platforms.

CICS's early implementation of distributed processing has proved to be very useful. For example, specialized CICS/ESA systems are often configured to manage the routing of applications, these are known as terminal owning regions. Other regions are configured to manage applications and files. These are known as application owning regions and file owning regions. Many years later, these configurations became the basis of mainframe CICS's operation on multi-processors and for the workload management support recently announced in CICS/ESA 4.1.

4.7 The Resource Manager Interface

As part of the work to make the SQL/DS relational database manager available to CICS/DOS/VS applications in the late 1970s, a general interface was introduced to allow external (non-CICS) resources to be coordinated with CICS resources, as part of the same transaction. This interface is also available in CICS/ESA and used by DB2, IMS/DB, MQSeries etc. The resource manager interface supports presumed abort, dynamic registration, single updater and read only optimisations. Thus, it is similar to the X/Open XA interface but it does have some additional facilities, e.g. the external resource manager can also participate in events other than commit processing such as application initiation and termination.

4.8 Restructuring CICS/ESA

In the early 1980s, some of the programmers working on CICS recommended that it be re-written, as it was becoming very hard to maintain. At about the same time, the CICS development manager heard Tony Hoare of Oxford University speak on the subject of the Z formal specification language. The eventual outcome was that important sections of CICS were identified, specified in Z, and re-written. The first release that included the restructured code was CICS/ESA 3.1 which was announced in 1989. This was a particularly difficult release to complete and ship, but eventually the work was completed and the release was very successful. Currently about one fifth of CICS/ESA has been specified in Z. These are the core sections which CICS spends 80% of its time executing. There are many Z experts at Hursley and at least two books on the subject have been written by Hursley authors [McMoPow93] and [Word92].

4.9 CICS OS/2

CICS OS/2 was introduced in 1988 and with it came support for some other CICS interfaces, e.g.:

- Distributed Program Link: The ability for a CICS program to invoke another CICS program across a network, without either program needing to know that the other program is remote. Data conversion, e.g. ASCII-EBCDIC, of the parameters is performed by CICS outside the application. Distributed Program Link is now available on all the CICS products from IBM.

- External Call Interface: This is a call request to enable a non-CICS program to invoke a CICS program synchronously or asynchronously. The server CICS program does not know that the invocation is any different from a local or distributed program link, nor does it know whether the request is synchronous or asynchronous. Multiple invocations can be part of the same transaction or run in separate transactions. An interface similar to the External Call Interface has recently been announced for CICS/ESA 4.1 enabling batch MVS, TSO or IMS programs to call CICS programs.

For more information on CICS OS/2, please see [Lamb92].

4.10 CICS/6000

CICS/6000 for the AIX operating system is built very differently from the other CICS products. It uses various components from Transarc, a Pittsburgh based company that specializes in OLTP, e.g. the Encina Server which acts as the transaction coordinator and supports the X/Open XA coordinator interface. Both the Transarc components and CICS/6000 itself use the Distributed Computing Environment (DCE). CICS/6000's internal use of DCE means that the components of a single CICS/6000 system can be placed on different servers. CICS/6000 (and CICS OS/2) support the External Presentation Interface enabling non-CICS programs to simulate a 3270 terminal to drive existing applications. A similar interface is available on CICS/ESA enabling CICS applications to simulate 3270 devices. See [StoKnut92] for more information about CICS/6000.

4.11 The CICS workstation products generally

These products tend to be smaller in terms of lines of code than CICS/ESA, reflecting the difference in underlying software. In addition, they come in two flavors, client and server. Thus, the client code has just enough support to enable a non-CICS program to access the CICS server system. On each server platform, CICS file control, temporary storage and transient data are implemented using local file systems, e.g. Structured File Server (SFS) from Transarc on AIX, and Btrieve on OS/2.

5 And finally

5.1 What next?

That's another story!

5.2 Thanks

Many thanks to Peter Alderson, Peter Collins, Steve O'Connell, George Czaykowski, Ken Davies, Phil Emrich, Mike Halperin, Gerry Hughes, Stuart Jones, Peter Lupton, Paul Mundy, Simon Nash, Bob Yelavich, Dennis Zimmer, who very kindly described their experiences with CICS. Thanks also to all the people who have worked on CICS.

5.3 Dates of first releases

These are the first shipment dates for some of the CICS products from IBM. The list is derived from information in [Gera93].

- CICS/OS: 1969, CICS/OS/VS: 1974, CICS/MVS: 1988, CICS/ESA: 1990.
- CICS/DOS: 1971, CICS/DOS/VS: 1974, CICS/VSE: 1990.
- CICS OS/2: 1989, CICS/6000: 1993, CICS/400: 1993.

CICS from IBM has been available for other operating systems, e.g. VM and DPPX. At the time of writing this article, CICS has been announced for HP and DEC. CICS client software has been announced (or is available) for DOS, Microsoft Windows, Apple Macintosh, AIX and OS/2.

The word "CICS" is pronounced differently across Europe and the US. Some examples are France and Germany: Sicks, Italy: Chicks, Northern Spain: Thicks, UK: Kicks, US: SEE - EYE - SEE - ESS.

References

[Gera93] Jim Geraghty, CICS Concepts and Uses - A Management Guide, McGraw Hill, 1993, ISBN 0-07-707751-2, 273pp.

[Graf83] William P. Grafton, IMS: Past, Present, Future, Datamation, September 1983, 158-171.

[IBM68] IBM program announcement: Generalized Information System, Information Management System, Public Utility Customer Information Control System, April 29, 1968.

[IBM69] IBM program announcement: The System/360 Customer Information Control System ready for shipment, July 8, 1969.

[IBM92] The Ben Riggins (and CICS) Story 40th anniversary, Reflections IBM Santa Teresa Laboratory, April 1992, 6-7.

[IBM4] GC33-0754 - Transaction Processing: Concepts and Products.

[IBM5] SC33-1007 - CICS Family: API Structure.

[Lamb92] Rob Lamb, Cooperative Processing using CICS, McGraw Hill, 1992, ISBN 0-07-014770-1.

[McMoPow93] Mike McMorran and Steve Powell, Z Guide for Beginners, Blackwell Scientific Publications, 1993, ISBN: 0-632-03117-4, 247pp.

[StoKnut92] Tony Storey and John Knutson, CICS/6000 Online Transaction Processing with AIX, AIXpert (IBM), November 1992.

[Word92] J.B. Wordsworth, Software Development with Z, Addison-Wesley, 1992, ISBN: 0-201-62757-4, 334pp.

[Yela85] B.M. Yelavich, Customer Information Control System: An Evolving System Facility, IBM Systems Journal, 24(3/4), 1985, pp. 264-278.

Pioneering Distributed Transaction Management

Eric Newcomer
Consulting Writer
Digital Equipment Corporation

1 Introduction

Digital Equipment Corporation is generally perceived as an engineering and scientific computing company rather than as a producer and seller of commercial application software. Digital, however, has been a pioneer in two significant areas of distributed transaction management: remote procedure call (RPC) communications and high-level transaction language. Technology based on Digital's distributed transaction management model is gaining wider acceptance in the industry.

Transaction management software guarantees that the state of a business, its operations or records, is reflected consistently on the computer without manual intervention. By bracketing a series of operations on business data within a transaction, users know that the operations will either succeed or be rolled back as a unit.

Digital's main contributions to distributed transaction management are twofold: one, to allow operations on data bracketed within a transaction to include procedure calls, local or remote using an RPC mechanism; and two, to specify transactional operations using a specialized, high-level programming language.

Digital was not the first to implement distributed transactional communication. IBM implemented distributed transactional communications within CICS as early as 1976. Nor was Digital the first to implement a transactional RPC. The Camelot project of Carnegie Melon pioneered the use of transactional RPC [Gray93]. But Digital was the first to create an open transactional RPC specification.

2 Transaction Processing Model

Digital's contributions are based on its unique transaction processing model. Digital's TP model has been called the "three-ball model" [Gray93]. Each "ball" is a group of procedures intended for a specific purpose, such as accessing a display, application flow and transaction control, and database and file access and general computation. Unlike mainframe TP monitors, Digital's ACMS monitor implemented the groups of procedures in separate processes. Communications among the processes was handled by an RPC mechanism. Digital thus pioneered the use of the RPC mechanism in transaction management, allowing parts of the application to be easily distributed among multiple processors in a network.

In the early 1980s, Digital created a TP monitor called the application control and management system (ACMS). It was integrated with the VAX Information Architecture (VIA) family of products built around the common data dictionary (CDD). The VIA products shared several characteristics. Among them were data definitions in the form of records stored in the CDD and an "English-like" high-level language. Those new to the VIA family of products confronted a learning curve to understand the architecture and its new languages. Often, however, after ascending this learning curve, programmers found they were very happy with the products and the productivity gains resulting from the use of a high-level language.

The ACMS "control" language, called the task definition language (TDL) [DEC91] puts into sequence a series of procedure calls to modules for display manipulation or for database and file access and general computation. Each "ball", or group of procedures is created and maintained separately. The interface to the procedures is

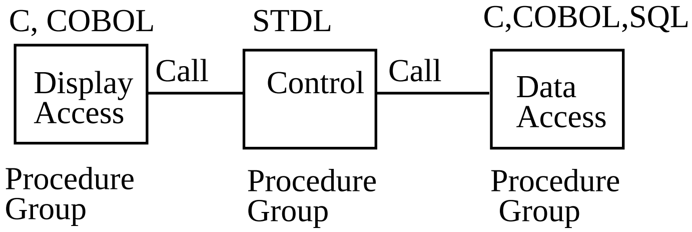

Figure 1: ACMS's "Three-Ball Model"

defined separately so that procedures in each of the groups can be changed independently. As long as the interface remains the same, the procedures within one group can be modified without requiring modification to any of the other groups of procedures.

Applications can be created in separate, specialized modules. One programmer can write the display manipulation procedures while another writes the database access procedures. A third can put it all together by writing the TDL control procedures. This model improves maintainability and eases upgrades to new technology by, for example, allowing the display manipulation procedures to be rewritten to use a new device without impacting any of the other procedures. Programmers can upgrade to GUI-based displays without having to change the control or database access procedures.

Furthermore, because the procedure grouping concept provides separation of procedure and interface, it is possible to transparently introduce an RPC in place of a local procedure call at any of the points of separation. Portions of the application can easily be distributed among multiple computers in a network. And by taking advantage of a name service such as DCE naming, the groups can be distributed and redistributed without modifying the procedure code.

The evolution of this approach lies in the VAX family line of computers and its phenomenal success in the 1980s. About the time ACMS was first released, small VAXes started to emerge in the market, such as the VAX stations I and II, the VAX 750 and the VAX 725. ACMS engineers realized that they could provide greater cost savings and increased performance by allowing the off-loading of the display processing (or forms control) onto one of the smaller VAXes, while the larger, or back-end VAXes were used for database processing. In ACMS terms, these were the "front end" and "back end" processors. In modern terms, these are the client and server processors. The high-level language of ACMS easily hid from the programmer whether the procedure calls were local or remote. ACMS engineers decided to implement this new functionality using an RPC mechanism that would be transparent to programmers.

3 An Open Architecture

During Digital's entry into the commercial applications market, two external events occurred which influenced the course of ACMS:

- Apple Computer worked with Digital to integrate the Macintosh with ACMS, so they could run their business applications using VAX servers and Macintosh clients.
- Nippon Telegraph and Telephone (NTT) adopted ACMS as the model for the multivendor TP standard in its Multivendor Integration Architecture (MIA) [MIA91].

MIA Consortium members included Digital, IBM, NEC, Hitachi, Fujitsu, and NTT Data (the systems integration subsidiary of NTT).

The result of the first was the ACMS Desktop product that integrates PCs with ACMS. The result of the second was the structured transaction definition language (STDL) and the remote task invocation (RTI) protocol, both of which are currently being implemented in the next generation of ACMS products. These two efforts are creating open technologies based on Digital's ACMS model.

Following its development by the MIA Consortium, the RTI protocol was adopted by X/Open as the basis of its TxRPC (or transactional RPC) standard [X/Open93]. STDL has been adopted by the Service Providers Integrated Requirements for Information Technology (SPIRIT) Consortium, a group of telecommunications service providers from the U.S.A., Europe, and Japan working to develop general-purpose computing platform specifications under the sponsorship of the Network Management Forum.

Although Digital was not the first to implement a transactional RPC, Digital was the first to put together OSF's DCE RPC with the ISO OSI TP and create a truly open standard for transactional RPC. Digital also developed STDL, a high-level procedural language specifically designed for transaction processing that unifies groups of procedures written using standard programming languages.

STDL is a portable application programming language for TP environments, based on the model of the original TDL of VAX ACMS. STDL is a block-structured language specifically designed for RPC-based distributed transaction management.

STDL provides application portability and TxRPC provides interoperability among TP monitor products. Portability and interoperability have been demonstrated on Digital, Hitachi, IBM, and NEC platforms. Both STDL and TxRPC are based on technology first implemented, tested, and proven in production applications using Digital's ACMS product.

In developing STDL, the original ACMS high-level language was modified and extended to make STDL portable, to ensure cost-effective implentability, and to meet the functional requirements of MIA. In developing the TxRPC, Digital drew on its experience with RPC-based transaction processing to integrate the DCE RPC with the OSI TP standard and move the technology into the open systems arena.

STDL is a persistent programming language especially suited to client/server computing:

- It allows programmers to structure computations as collections of tasks.
- These tasks can be local or remote.
- Task invocation in a heterogeneous client/server environment is supported by the TxRPC.

STDL's exception handling mechanisms make it easy for programmers to construct workable and manageable distributed heterogeneous applications that are portable among many vendors' TP monitors.

STDL demarcates transaction boundaries, performs transactional remote procedure calls, supports transactional queuing operations, transactional display management, and transactional exception handling. In this example, the STDL task called "example" represents the middle ball of the three-ball model, while the RECEIVE and SEND exchanges with the display represent the first ball. The CALL PROCEDURE represents the third ball. This separation of procedure according to the type of function being performed allows a process-based implementation to tune each type of process for the type of work being performed. Each process is active only long enough to complete its portion of the work. The calls are bracketed by a transaction that ensures the operations either all succeed or are rolled back. Any number of data operations, local or remote, can be included.

Most application logic, as well as operations on files and databases, is performed using a combination of standard C, COBOL, and embedded SQL. Transactional functionality and features additional to standard C and COBOL required to support distributed transaction management applications are implemented in STDL. The isolation of transactional functionality within STDL allows the use of standard C, COBOL, and SQL and permits the mapping of STDL to any commercial TP monitor.

By using the atomicity and isolation concepts from transaction processing, STDL insulates the application programmer from system calls, and permits transparent application distribution to any other STDL systems irrespective of hardware or operating system configurations.

```
TASK example
WORKSPACES ARE customer, control
BLOCK WITH TRANSACTION
EXCHANGE
  RECEIVE RECORD customer FROM input-form
PROCESSING
  CALL PROCEDURE update IN database-server
EXCHANGE
  SEND RECORD customer,control TO input-form
END BLOCK
EXCEPTION HANDLER IS
  GET MESSAGE failure INTO display-field
END EXCEPTION HANDLER
```

Figure 2: STDL Code Example

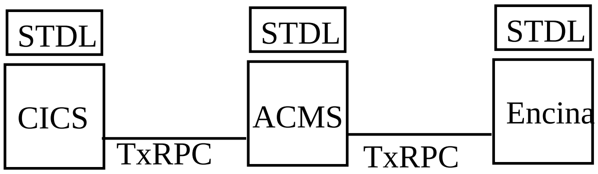

Figure 3: STDL for Portability and TxRPC for Interoperability

STDL separates execution flow control, user access, and database and file access. This modularity makes STDL application programs inherently distributable. Local procedure calls can be replaced by remote procedure calls without application recoding. Chained transactions and concurrent processing are supported [Bern93].

Tasks written in STDL replace the main program module of traditional TP applications. STDL eliminates the need for system service calls. This increases application programmer productivity by enabling application programmers to concentrate on business problems rather than dealing with the idiosyncrasies of operating systems.

The major advantage of STDL is portability. STDL can be implemented on top of any existing TP monitor on the market, and can also be implemented as the native language of a new TP monitor. In this respect, STDL is very similar to SQL, which also has been implemented on top of existing database management products as well as implemented as the native language for new database management products.

An application written using a combination of STDL, standard C or COBOL, and standard SQL, is portable at the source code level. This was demonstrated at NTT's labs in May, 1993 by porting a transaction processing application between the Digital and IBM implementations of STDL.

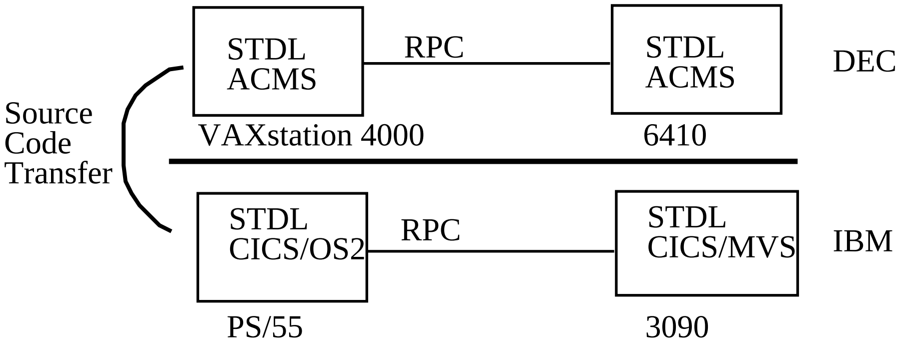

Figure 4: Portability Demonstration at NTT Labs, May 1993

IBM has sucessfully layered STDL on top of CICS/MVS and CICS/OS2. NEC has supplied NTT with an STDL product, as has Digital. HP, Fujitsu, and Hitachi have also committed to implement STDL and the TxRPC specification from X/Open that includes the MIA-developed RTI protocol [NTT93].

Through these efforts, the technology Digital originally developed in ACMS is becoming accepted more and more widely in the industry as the basis for modern, RPC-based transaction management. However, general acceptance of new, open technologies can be slow. The fate of these new, open technologies is essentially in the users' hands. If the benefit of standards is truly for the user, it is truly the user that will have to demand them and provide the force that will open the TP market.

The technologies derived from Digital's ACMS product can serve as the basis of a standard for opening the TP market because they are modern, thoroughly tested in production situations, and have been proven to work by multiple vendor implementations.

The separation of procedure from interface in STDL paves the way for future migration to object-oriented technology. Digital is investigating the future transformation of STDL into object-oriented STDL. The task group is essentially identical to a network object, paving the way toward future integration with OO technologies in distributed transaction management.

Because of STDL's origins in ACMS, the language is currently more like COBOL than C. As it evolves over time, however, more and more concepts from C are being incorporated.

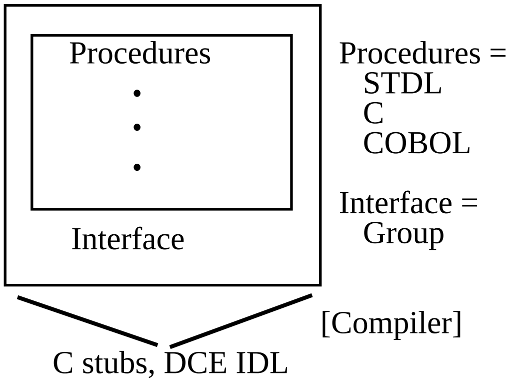

Figure 5: Separation of Procedure and Interface

4 Summary

Despite Digital's reputation for unfamiliarity with the commercial applications market, Digital has leveraged its strength in networking and high-level languages into pioneering efforts within the distributed transaction management field.

References

[Bern93] "STDL – A Portable Language for Transaction Processing", Philip A. Bernstein, Per O. Gyllstrom, Tom Wimberg, Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993.

[DEC91] VAX ACMS V3.2 Transaction Processing ADU Reference Manual, Digital Equipment Corporation, 1991

[Gray93] "Transaction Processing Concepts and Techniques", Jim Gray and Andreas Reuter, 1993 Morgan Kaufmann, Pub.

[MIA91] "Multivendor Integration Architecture, Division 1, Overview, Technical Requirements" TR550001, Nippon Telegraph and Telephone Corporation, 1991

[NTT93] Report by NTT at the X/Open User's Requirements Group meeting, September 1993

[X/Open93] Distributed TP: The TxRPC Specification, X/Open Preliminary Specification, 1993.

High Performance TP Monitors - Do We Still Need to Develop Them?

Dieter Gawlick
Activity Management Group
Digital Equipment Corporation
529 Bryant Street, Palo Alto, CA 94301
email: Gawlick@HPTS.enet.dec.com

1 Introduction

Since the 1970's IMS/VS is IBM's preferred TP monitor for installations requiring large, high performance data bases, automatic scheduling, reliable end to end computing and fault tolerance. By the mid 1970's the IMS/VS infra-structure was in place. However, the existing system neither delivered the required performance nor the level of fault tolerance for banking applications. Two extensions, IMS/VS Fast Path and XRF (Extended Restart Facility), were created to address these issues.

The reminder of the paper is as follows: Sections 2 explains the challenges of high end business applications using the classical Debit/Credit transaction, the ancestor of the TPC (Transaction Processing Council) work, as guideline [Anon89], [tpca92]. Sections 3 and 4 describe, how IMS/VS Fast Path and XRF solve the majority of these previously described problems. Section 5 and 6 will explain the support that needs to be offered by operating and database systems to solve the Debit/Credit challenges. In section 7, I will argue why these solutions are a subset of the requirements for evolving intra and inter company "groupware" and/or workflow systems.

2 The Challenge of the Debit/Credit Transaction

IMS/VS Fast Path and XRF were created to handle a system that can process large volumes of data with high availability and high performance. Since Debit/Credit was and still is one of the most frequently used transactions in these environments, I will use it to explain these issues. The following is a definition of the Debit/Credit transaction:

```
Get the next work item
Identify customer account number
Retrieve customer account
If there are enough funds
Update the account
Update the ATM information
Update the general ledger
Append a journal record
Send OK and new balance to ATM
Else
Deny request
End
```

Although this transaction is very simple, it creates the following set of challenges:

1. The ATM information is a hot spot. Using disk based databases leads to expensive I/O operations.
2. The general ledger is a hot hot spot., requiring potentially hundreds of updates per seconds for some records.
3. The application journal requires a database with excellent performance to append data while being able to retrieve these data entries efficiently for individual accounts as well as for groups of accounts.
4. The system can deal with very large amounts of data, especially for the application journal.
5. The system requires no service interrupts for reorganization of data bases, i.e. the reorganization has to be on-line and impact the performance as little as possible.
6. The system requires no service interrupt for data base checkpoints, i.e., data base backups have to be done on-line.
7. Databases can be moved to different I/O devices while the system is operating close to full speed.
8. Data have to be replicated to avoid lengthy recovery. Furthermore, local errors can be repaired while the system is operating close to full speed.
9. The response time of the server - database plus application program, is well below one second, preferably below .1 second.
10. The response time of the system is flat up to 95load.
11. The connection to the ATM has to be such that each request will be processed exactly once and each response will be deliv- ered exactly once.
12. A hot stand by system can deliver the full service of a running system within a short period of time (2 minutes), without intervention of end users, system operators and without any effort by the programmer.

The next two sections describe how IMS/VS Fast Path and XRF allowed IMS/VS to meet these challenges. I will always point out which problem is solved by a specific technology.

3 IMS/VS Fast Path Functionality

Descriptions of the IMS/VS Fast Path technology can be found in [gawl8(2)], [gray91], [IMSGIM] and [oneil86]. Here are some highlights:

Fast Path supports two database type, MSDBs (Main Storage Databases) and DEDBs (Data Entry Databases).

MSDBs deal with small amount of heavily used data. Normally, MSDBs are kept in main memory, giving good response time and preventing unnecessary I/O operations. Extreme hot spots are supported through a combination of Optimistic Escrow, Fast Commit and Group Commit technology. MSDBs allow the classical read and update approach. Furthermore, they allow the user the specify the desired change of a numerical field. At request time the systems checks, whether the request could be granted under the current conditions. No locks will be held. The request will be re-checked at commit time. If a request can still be granted the transaction commits, otherwise it aborts. Fast Commit prevents the need for I/O during commit time. However, Fast Commit allows the completion of all internal processing. All external notifications to databases and terminals are held back until the required journal records are written to non volatile storage. In case of a failure, the system can not re- cover all the committed transactions. However, it can recover all those transactions for which the completion may be externally known. The system requires no further action, there is no recovery gap through this technology. It is indeed not necessary to make a transaction recoverable before it can be committed. Additional enhancements for optimization of the journal I/O are achieved through Group Commit. (Problems 1 and 2)

DEDBs support a hierarchical database technology in the spirit of IMS/VS. DEDBs can be configured into up to 240 Areas, each of which can be placed and managed independently. The size of an area can be up to 4 Giga bytes. DEDBs support a record type called Sequential Dependent. Data of this type are appended to database records in LIFO order. For each Area these records are stored at the tail of an Area in commit sequence order. This creates again an hot hot spot. A process local work-to-do-list and the fast commit technology are used to solve this problem. (Problem 4)

Access to records is either by key or sequentially. The Sequential Dependent segments can either be retrieved for each database record in LIFO or sequentially in an Area by defining a starting or ending record. (Problem 3)

Utilities such as reorganization and database checkpoint work on Area level. They can be executed while the Area is used for on-line services. Through a skillful usage of the underlying file system, the utilities perform well as batch processes without interfering in any significant way with the on-line system. (Problems 5 and 6)

DEDBs can be replicated up to seven times. Copies of Areas can be added and deleted while the system is running. This concept allows not only data replication but also the relocation of Areas while the system is running, enabling the user to balance the load of the I/O system, to move Areas to new devices or device types, and to eliminate faulty devices. (Problems 7 and 8)

In 1987 the system achieved a transaction rate well above 1ktps with response time well below 1 second. The system showed in- teresting performance characteristics. Under increased load the required instructions per transaction decreased, the speed of the hardware increased and the response time of the journal decreased. In effect the response time of a system running Debit/Credit transactions was flat close to the saturation point. (Problems 9 and 10)

Each transaction is guaranteed to run exactly once and each response to the front-end is delivered exactly once. This is simply done through the use of the network support (Set and Test sequence numbers) and journaling (Problem 11)

An IMS/VS system can support and manage tens of thousands of terminals. IMS/VS Fast Path needs about 35k instructions per transaction. On an IBM mainframe each instruction requires on the average 3.5 CPU cycles.

4 IMS/VS XRF Functionality

While IMS/VS Fast Path was able to solve the functional, performance and reliability requirements of the Debit/Credit transaction, a more general approach was needed to solve the hot standby problem. This was done through a project called IMS/VS XRF. The hot standby technology was started through the requirements on IMS/VS. However, it influenced, challenged and improved the underlying operating system, file system and network support.

IMS-XRF assumes that there are two local processors which are connected through a high speed link and which share I/O devices. While one system is active, the other system, the backup system, is in hot standby mode. The backup systems monitors closely the operations of the active system. It knows which terminals are connected and how they can be reached, and which databases are opened and where they are located. It keeps closely track of the database locks and monitors whether the active system is still alive. The backup system starts frequently used programs for im- mediate activation. The backup system is really ready to go.

If a failure occurs, the journal of the active system is disabled and is read to the end, which takes only parts of a second. The operating system blocks all access of the active system to the databases. Furthermore, it improves the service to the standby system by changing it's priority. The database parts that need recovery are locked, the transaction queues are synchronized. The backup system is now ready to run. While processing new transactions, the databases and the terminals are synchronized and restarted, without intervention from operations or the end users. (Problem 12)

For additional descriptions of IMS/VS XRF see [gray91] and [IMSXRF].

5 Operating System Support for High Performance and High Reliable Systems

The necessary operating system support for systems that respond to these requirements is well understood:

1. Fast switch from privileged to non privileged code and vice versa (well below 100 instructions).

2. Good support of shared memory with very fast locking mechanism (below 100 instructions).

3. Very Light Process structure with limited functionality and access to shared memory (process start and termination must be below thousand instructions).

4. The ability to write efficient I/O programs that use the semantic knowledge of the database systems for optimization.

5. Fast Workload Acceptance: The ability to switch the system resource allocation for a set of processes almost instantly (In the range of a few seconds).

6. Fast I/O prevention: The ability to stop I/O to pre-defined data sets instantly (well below one second).

7. The ability to connect multiple programs to the same data set in update mode, only the active system will actually perform updates.

8. The ability to connect multiple programs to the same ’terminal.’ Only one, the active program will communicate, the other, the backup program, needs the help of the network support to synchronize backup system and terminals very fast.

These requirements represent well understood and implemented technology. There is no reason why it can not be added to server oriented operating and file systems. Operating systems also need to offer a scheduler that could be used for transaction scheduling.

As noted in [Klein93], 2PC needs to be a fundamental extension of operating systems. Local 2PCs can be very efficient by using shared memory and shared journals, which should be offered by database systems. Distributed 2PCs should run on low level standard transports and support standard protocols for interoperability. Although there seems to be no technical obstacle to implement the last two requirements, I am not aware of any product that includes either a scheduler or an efficient 2PC technology within the just mentioned constraints .

While a significant part of the support for high performance, high reliable systems must be imbedded in the underlying system infra-structure, some parts of the required support is best provided by database systems.

6 Data Base Support for High Performance and High Reliable Systems

These are some of the most important requirements for database systems:

1. Applications require the storage of an increasing amount of history information. It is time that databases offer good support for temporal data.

2. To simplify the programming and support increased parallelism, support for escrow technology is required.

3. High reliability requires data replication as well as on- line utilities for database checkpoints, relocation and reorganization.

4. Hot standby support demands the ability to accept the workload of an active system and to resume full operations within a very short time interval ($<$ two minutes, which seems to be equivalent to a “very bad” response time).

While the support for the first three requirements seems to be straight forward and well understood, the support for hot standby will be the most demanding task. Database systems will have to deal with varying level of support by the different underlying operating systems, file systems and networks.

7 Groupware/Workflow Systems

The modern business environment becomes increasingly business process oriented. New systems are evolving to support this trend, [gawl94], [leym93]. While transaction systems are focused on the support of isolated activities, steps in the language of workflow, workflow systems are focused on the connection of steps. These two approaches need to be integrated. We should use the power of transaction systems for high volume transactions. However, the workflow technology should integrate these steps into the business context. For some time to come, TP/DB systems and workflow system can cooperate through the evolving CORBA technology. However, at some time in the future we need a tight integration. Such an integration will only be possible if these system are based on a common infra-structure such as the one described in the previous sections. Furthermore, workflow technology will only be mature enough for the use as backbone for business environments, if it supports all the performance, reliability and fault tolerance characteristics of mature TP/DB systems.

Workflow system requirements exceed the TP/DB technology in respect to scalability and system wide reliability. It will be an important challenge to use the experience of the Mail technology to achieve these system additional attributes.

8 Summary

For many years, the technology to develop high performance, high reliable systems has been well known. While the technology was developed for the special use of transaction monitors, it is applicable to other areas. However, the economics of today's software development prohibits specialized development efforts such as IMS/VS Fast Path and XRF [gray94]. The time has come to leverage the knowledge, that has been acquired by the development of these systems and incorporate it into the operating system and database technology of today's high end servers.

I challenge the developers of modern operating systems and database systems to create systems that execute a Debit/Credit transaction with at most 100k CPU instructions, and still achieve all the operational, performance and reliability characteristics of today's top of the line specialized TP/DB systems.

References

[Anon89] Anon Et AL., "A measure of Transaction Processing Power." Datamation 31(7) 1 April 1989, pp. 112-118.

[gawl8(2)] Gawlick, D., and D. Kinkade, "Varieties of Concurrency Control in IMS/VS FastPath." IEEE Database Engineering, 8(2): pp.3-10

[gawl94] Gawlick, D., M. Hsu, R. Obermarck, "Strategic Issues in Workflow Systems." CompCon 94, March 1994, San Francisco, CA, IEEE 1994

[gray91] Gray, J., and A. Reuter, "Transaction Processing: Concepts and Techniques." Morgan Kaufmann, San Mateo, CA, 1991

[gray94] Gray,, J., C. Neyberg, "Desktop Batch Processing." SFSC Technical Report 94.1, January 1994, San Francisco, CA: DEC

[IMSGIM] IBM-IMS, Information Management System: General Information SC26-4275 and Application Programming SC26-4279. San Jose, CA: IBM

[IMSXRF] IBM-IMS-XRF, IMS Extended Recovery Facility (XRF) Planning, GC24-3151. San Jose, CA: IBM

[Klein93] Klein, J., Upton, Open DECdtm, Constraint Based Transaction Managment Proceedings of the ACM SIGMOD, May 1993

[leym93] Leymann, F., W. Altenhuber, “Managing Business Processes as Information Resource.” ITL IRM Conference, Thornwood, USA October 1993

[oneil86] O’Neil, P. E. “The Escrow Transactional Method.” ACM TODS. 11(4) 1986 pp. 405-430

[tpca92] “TPC Benchmark A,” Chapter 2 of “The Benchmark Handbook for Database and Transaction Processing Systems,” 2nd ed. Morgan Kaufmann, San Mateo, CA, 1992

Recent Work on Distributed Commit Protocols, and Recoverable Messaging and Queuing

C. Mohan, and Dick Dievendorff
IBM Almaden Research Center
San Jose, CA 95120, USA
{mohan, dieven}@almaden.ibm.com

Abstract

This paper briefly summarizes some recent IBM work in the areas of distributed commit protocols, and recoverable messaging and queuing. We discuss the original presumed nothing commit protocol of SNA LU 6.2 and the current industry standard presumed abort (PA) protocol which we originally developed in IBM's R project. We also discuss generalized presumed abort (GPA) which resulted from the integration of PA into LU 6.2. GPA has been implemented in DB2 V3. We provide a brief introduction to the Message Queue Interface (MQI), an architected application programming interface, and Message Queue Manager (MQM) MVS/ESA, one of the IBM MQSeries products that implements MQI. Some internal design features of MQM are also described.*

1 Introduction

Several paradigms have been adopted for inter-program communication in a distributed computing environment. The most important of them are conversations, messaging and remote procedure calls (RPCs). In this paper, we will discuss some of our recent work relating to the first two topics.

2 Distributed Commit Protocols

In [MoLi83, MoLO86], we introduced two efficient distributed transaction commit protocols called Presumed Abort (**PA**) and Presumed Commit (**PC**). PA and PC were first developed and implemented at the IBM Almaden Research Center in the context of the distributed database management system R*. PA and PC are extensions of the well-known, classical two-phase commit protocol. PA is optimized for read-only transactions and a class of multisite update transactions, and PC is optimized for other classes of multisite update transactions. The optimizations result in reduced inter-site message traffic and log writes, and, consequently, a better response time for such transactions. Both PA and PC were designed to make mainline (i.e., no-failure) performance be very good, at the possible expense of some performance degradation under certain failure conditions.

Of late, PA has become very popular. It is now part of the ISO-OSI and X/Open standards for distributed transaction processing (DTP). It has been implemented in IBM Almaden Research Center's R* and QuickSilver, Tandem's TMF, DEC's VMS, Transarc's Encina, CMU's Camelot and Unix System Laboratories' TUXEDO.

IBM's communications architecture SNA LU 6.2 supports inter-program communication using half-duplex conversations [SJFF89]. Originally, LU 6.2 included a commit protocol (call it *Presumed Nothing (PN)*) which was inefficient in terms of logging and message overheads. PN was implemented in CICS/MVS and VM/ESA. Honeywell Bull implemented it in GCOS8. The distributed computation model that LU 6.2 supports (peer-peer) is very general compared to the one that we had to deal with in R* (client-server). Recently, the LU 6.2 architecture has been enhanced to include PA as an option [IBM93d]. We call this version of PA *generalized PA (GPA)* since it accommodates LU 6.2's peer-peer computation model [MBCS93]. It coexists with PN in the sense that even during the execution of a single transaction some of the nodes executing the transaction may be using PN while others are using PA. Some flaws in the original architecture with regard to the read-only optimization (i.e., avoiding the second phase of message exchanges with read-only process subtrees) and handling of terminations of conversations which result in two or more islands of the original distributed computation continuing to exist were also fixed. In those situations, problems arose due to the manner in which transaction identifiers were generated and used.

In LU 6.2, during the commit of a transaction, all the processes of the then-existing process tree of the distributed computation were involved in the commit protocol message exchanges whether or not some of those processes did any work as part of the committing transaction. In R*, all work originated from only the root process of the tree since that was the only process where the user application ran. The other processes performed database accesses based on requests from the root process. So, in R*, any process that was not involved in the current transaction was excluded from the commit protocol, thereby gaining significant performance advantages. In LU 6.2, since the user application may be executing in more than one process, work within a transaction can be self generated by the processes of the distributed computation. As a result, determining which processes performed work as part of a transaction is not easy. GPA supports R*'s optimization as an option by taking advantage of the half-duplex nature of LU 6.2 conversations [BCMS93].

PA, as originally designed and implemented in R*, did not support *coordinator migration* (i.e., making a process other than the one that started the transaction be the coordinator of the commit protocol). Also, in R*, the *initiator* of the commit protocol was always the same process that began the transaction (unilateral rollbacks could be initiated by any process). Both initiator and coordinator migrations are supported by PN and GPA [CiMo91, MBCS93]. While they give rise to some race conditions which complicate the protocols, they provide some performance and availability advantages [SBCM93].

Integrating PA into the LU 6.2 commit protocol architecture was not a simple task given the complex and flexible nature of PN. Since much of the functionality and mechanism of the OSI DTP standard is based on LU 6.2, our work should be of value for future enhancements of the standard. Many of the advanced features of LU 6.2 are not currently part of the OSI DTP standard. Unfortunately, much of the research community has been unexposed to PN. It is important that the strengths and weaknesses of this widely-in-use protocol be understood to enhance the new DTP standards. OSI DTP has a few features that LU 6.2 currently does not have. Recently, GPA has been implemented in DB2 V3 [Moha93a]. Since IBM's Distributed Relational Database Architecture (DRDA) [IBM93a] incorporates LU 6.2, now DRDA-compliant products can exploit and benefit from the efficiency-enhancing features of presumed abort.

3 Messaging and Queuing

Messaging supports asynchronous communication between programs. Communication takes place via recoverable message queues. Transaction semantics is supported with respect to the interactions between the programs and the message queues. IBM has defined an application programming interface (API) standard for messaging called Message Queue Interface (MQI) [IBM93b]. IBM has released a family of products called MQSeries that supports MQI. One of those products is Message Queue Manager (MQM) MVS/ESA [IBM93c].

Messaging products are referred to as middleware. They interface between applications and communication networks. They shield the complexities of the underlying multivendor, multiprotocol networks, allowing a more user-friendly programming environment in an open, distributed world where interoperability is essential. They allow integration of multiple transaction and database management systems. E-mail messaging (e.g., using X.400 and X.500) provides non-realtime communication between people, whereas the MQSeries products also provide online/realtime, asynchronous message exchanges at computer speeds between programs. Typically, MQSeries messages reside only for a few seconds in message queues before they are processed.

MQSeries products operate on IBM and non-IBM platforms and they support the architected MQI. Named queues are used by the applications for message exchanges in a location-transparent manner. Because of the asynchronous nature of the supported communication paradigm, the participating programs need not all be available (i.e., up and running) at the same time. MQI is not sensitive to network transport protocol differences. It lets the application designer concentrate on the business logic alone rather than having to deal with communications logic also. Conversational communication (e.g., as in LU 6.2) and RPC are connection based. As messaging is connectionless, it shields the application program from communication failures and recovery. The underlying messaging and queuing infrastructure deals with those problems.

3.1 Message Queue Interface

Queues are named and each queue is owned by a queue manager. Multiple queues may be owned by the same manager. Put (add a message) and Get (retrieve and delete a message) are the main calls in MQI. Each message has a *MsgId* and a timestamp which marks the time when the message was Put. A message consists of some application data and some control information. The latter is examined by a queue manager to determine, for example, what the message's priority is and whether the message is persistent or not. Within a priority class, messages are appended to the end of the subqueue representing that priority class. To identify problem messages, for each message, a count is maintained of the number of times the message was retrieved and then the retrieval was rolled back. A message can be typed to indicate if it is a one way message, is a request which needs to be replied to or is a reply to an earlier request. In the case of a request, the identity of the message queue to be used for the reply is given. A reply identifies (using the *correlation identifier (CorrelId)* field) the original request message.

On a Get call, if no message satisfying the request is currently available for retrieval, then the *wait* option can be used to indicate that the call should wait for a new message to be Put and committed or for an existing message's Put to be committed or for an uncommitted Get to be rolled back. A time limit for the wait can be specified. A Get can be specified to retrieve the next available message in a queue or a message which has some specific values in the MsgId and/or CorrelId fields. The *browse* option of Get can be used to retrieve a message without deleting it from the queue. With this option, *first* and *next* calls can be used to look at a sequence of messages. To enable this, internally, a browse cursor which holds a position in the queue of messages is maintained. Get can be used to retrieve and delete the message on which the browse cursor is currently positioned. Puts and Gets can be issued within or outside the scope of the current transaction. When a Get/Put is issued with the outside option, that call's effect alone is committed immediately without the current transaction's other actions being committed.

When a message is Put, it can be tagged as being persistent or non-persistent. Once the Put of a *persistent* message is committed, even if there are system failures, the message will continue to stay in the queue until it is deleted via a Get. On the other hand, all *non-persistent* messages disappear if there is a system failure. In the absence of system failures, both persistent and non-persistent messages obey transaction semantics (i.e., commit and rollback).

A *local queue* is one which is managed by the queue manager to which the application is connected. A *remote queue* is managed by some other queue manager. An application can issue Put and Get calls to a local queue, but can issue only Put calls to a remote queue. When a Put involving a remote queue is issued, first the message is inserted into a *transmission* queue in the local queue manager. Some time after the Put commits, the message is moved *asynchronously* to the remote queue by the queuing system software.

Certain events that relate to a queue (e.g., the queue becoming non-empty) can be made to trigger some activity. When triggering is enabled for a particular queue, the occurrence of a triggering event at that queue causes a trigger message to be inserted into an *initiation* queue. The latter can be monitored by a program which then schedules, based on the contents of the initiation queue message, another program to deal with the event-triggering message in the original queue. Even if the transaction adding the trigger message rolls back, the trigger message's Put is always committed. This is necessary to deal with situations where one transaction may decide not to trigger a Put based on the so-far-uncommitted earlier triggering action of another transaction. Later on the former transaction may commit while the latter transaction rolls back. For this case, triggering is still needed even though the triggering transaction rolled back. To improve performance, trigger messages are always treated as non-persistent messages.

3.2 Message Queue Manager MVS/ESA

Message Queue Manager (MQM) MVS/ESA is the implementation of MQI on IBM's MVS/ESA operating system. MQM can be invoked from IMS, CICS, TSO and Batch applications. It participates in two-phase commit processing with IMS and CICS. It uses CICS Inter-System Communication (ISC) for moving a message from a transmission queue to a remote queue.

Internally, MQM is designed like a DBMS (e.g., DB2 [Moha93a]) in that it does its own logging, recovery, space management, buffer management and locking. It supports recovery from not only system failures but also media failures. MQM uses the write-ahead logging based ARIES recovery method [MHLPS92] which has been implemented in a number of products (DB2, DB2/2, DB2/6000, Encina, WDSF and ADSM) and prototypes (Starburst, QuickSilver, Exodus and Gamma). It supports fuzzy checkpoints, fuzzy backups and dual logging. It allows a message's length to be as high as 4MB. It locks individual messages with non-intention locks (e.g., S and X). Since the lock hierarchy includes queues and pages also, typically, intention (e.g., IX) locks are acquired at those levels. It uses the Commit_LSN technique of [Moha90, Moha93b] to avoid/reduce locking overheads under certain conditions. MQM's buffer manager follows *no force* and *steal* buffer management policies, and it does batched (i.e., multipage) I/Os and asynchronous writing of dirty pages to disk. MQM uses the *nested top actions (NTA)* feature of ARIES to efficiently support commit of some actions of a transaction irrespective of whether the transaction as a whole commits or rolls back. An NTA is used to support, for example, a Put/Get that is issued outside the scope of the current transaction [DiMo93] or the Put of a trigger message on an initiation queue. The writing of log records (*compensation log records (CLRs)*) during updates performed as part of rolling back a transaction permits the easy support of features like recoverably tracking the number of times a message had been retrieved and then the retrievals were rolled back.

Mixing of persistent and non-persistent messages on the same queue is supported efficiently. In order to avoid scanning every message in every queue during restart after a failure to purge non-persistent messages, storage allocation within a file for persistent and non-persistent messages is handled differently. On a given page, persistent and non-persistent messages are not intermingled. Space map pages (SMPs), as in DB2 [MoHa94], keep track of which pages have been allocated to non-persistent messages. Consequently, it is enough to examine the SMPs to deallocate the space associated with non-persistent messages during restart or on first access to a given SMP after restart had occurred. The pages storing the non-persistent messages themselves do not have to be accessed for purging such messages. To guarantee that across persistent and non-persistent messages the messages are retrieved in the same order in which they are appended, each queue header has 2 subqueue pointers: one for each type of message on the queue. All the pages containing persistent (respectively, non-persistent) messages of that queue are chained together starting from the persistent (non-persistent) subqueue header. Actually, this is done separately for each priority class (i.e., on a given page messages of different priority classes are not mixed together). Within a page, messages are stored contiguously without chaining. During a Get operation, the timestamps on the first available message in each of the subqueues is examined to determine whether the next message to be retrieved should come from the persistent subqueue or the non-persistent subqueue.

To provide transaction semantics to Gets/Puts involving non-persistent messages, a virtual memory log (*soft log*) private to each transaction is used [DiMo93]. This strategy improves performance compared to the one which uses the usual disk-based log for them also. No redo logging is needed for Puts/Gets of non-persistent messages. Only undo logging is needed. Whenever an NTA involving a non-persistent message completes, the soft log records relating to that NTA are purged since they are no longer needed.

MQM performs logical deletion, as opposed to physical deletion, of messages during Get operations to avoid having to log message contents during Gets. Once a Get is committed, the space occupied by the deleted message could be garbage collected. An asynchronous garbage collector is used for this purpose. To determine efficiently that a page full of deleted messages is in the committed state, the Commit_LSN technique is used. Only when the former condition is true is such a page removed from the linked list of pages representing the queue contents. Performing logical deletions also avoids the need ever for a *logical undo* when the rollback of a Get is performed, unlike in the case of B^+-trees with ARIES/IM [Moha94, MoLe92]. To avoid the need for logical undos, the undo of a Put also is implemented as a logical deletion of the corresponding message. Logical deletions also avoid physical deletions' need for a space-reservation method like the one described in [MoHa94]. Since the logically deleted messages continue to be in the queue until garbage collection happens, to improve performance for Gets, a pointer is maintained in the queue header to the first page containing a non-deleted message. When a Get is rolled back this pointer is reset to the first page of the queue, instead of using complicated logic to determine the position of the affected page in the chain of pages. To make a Put efficient, for each priority class and for each message type (persistent or non-persistent), a pointer is maintained to the last page (tail) of the subqueue. For efficiency reasons, queue header information is retrieved from the database and cached in a hash table in virtual storage.

It is possible that a transaction, after doing some updates (which includes doing some Gets and, possibly, some Puts), encounters an error condition which relates to one or more of the messages which were retrieved. Under these conditions, if the transaction were to merely rollback, then the problem-causing messages would be put back on their queues due to the rollback of the Gets. Those messages would then become available for subsequent Gets to retrieve them. If the application were to determine that the problem-causing messages should not be put back on the queues since they would continue to give problems to future transactions that would Get them, then the application might desire the following alternate sequence of events: the transaction undoes all its updates *except* the Gets relating to the problem-causing messages, Puts one or more error messages and then *commits*.

Supporting the above semantics requires some extensions to the usual logging and recovery logic since it involves *selective* undo under certain conditions (e.g., when there is a rollback due to an application-detected error condition but not when the rollback is due to a process or system failure). MQM supports this functionality by handling application-initiated rollbacks (AIRs) differently and by letting transactions indicate (*mark*) during Get operations which of those operations should not be rolled back in case of an AIR. Internally, in MQM, this is implemented as follows: A list of the marked Gets of a transaction is maintained in virtual memory. When an AIR occurs, longer-than-commit-duration (i.e., *allocation* duration) locks are obtained on the marked messages, the transaction is rolled back normally (which includes rolling back the marked Gets), all locks except allocation-duration locks are released, a new transaction is begun, the marked Gets are redone, allocation-duration locks are released and control is given back to the application. Now the application can issue its error messages and commit. The latter will commit the marked Gets also. If a rollback other than AIR were to occur, then the normal undo logic will cause the marked Gets also to be rolled back without any subsequent automatic redo of those Gets. Acquiring the allocation-duration locks ensures that the redo of the marked Gets can be performed without interference from other transactions. In the current implementation, only one marked Get can be issued in a transaction.

A Get with the browse option retrieves only a committed message, but since no lock is retained on the retrieved message that message may be deleted by another transaction. A browse cursor's position is retained across a commit. Holding an allocation duration page lock on the current page ensures that garbage collection does not unchain that page from the queue.

Acknowledgements We would like to acknowledge the contributions of our Almaden, Hursley and Raleigh colleagues to the work described here.

References

[BCMS93] Britton, K., Citron, A., Mohan, C., Samaras, G. *Method of Excluding Inactive Nodes from Two-Phase Commit Operations in a Distributed Transaction Processing System*, **U.S. Patent 5,258,982**, IBM, November 1993.

[CiMo91] Citron, A., Mohan, C. *Combining Presumed Abort Two-Phase Commit Protocols with SNA's Last Agent Optimization*, **IBM Technical Disclosure Bulletin**, Vol. 34, No. 7B, December 1991.

[DiMo93] Dievendorff, D., Mohan, C. *Selective Participation in Unit-of-Work Protocols*, **IBM Technical Disclosure Bulletin**, Vol. 36, No. 11, November 1993.

[IBM93a] *Distributed Relational Database Architecture Reference*, **Document No. SC26-4651-01**, IBM, March 1993.

[IBM93b] *Message Queue Interface: Technical Reference*, **Document No. SC33-0850-01**, IBM, April 1993.

[IBM93c] *Introduction to Using the Message Queuing Interface via Message Queue Manager/ESA*, **Document No. GG24-4062**, IBM, June 1993.

[IBM93d] *Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2*, **Document No. GC30-3084-5**, IBM, July 1993.

[MBCS93] Mohan, C., Britton, K., Citron, A., Samaras, G. *Generalized Presumed Abort: Marrying Presumed Abort and SNA's LU 6.2 Commit Protocols*, **Proc. 5th International Workshop on High Performance Transaction Systems**, September 1993. Also available as **IBM Research Report RJ8684**, March 1992.

[MHLPS92] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P. *ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging*, **ACM Transactions on Database Systems**, Vol. 17, No. 1, March 1992.

[Moha90] Mohan, C. *Commit_LSN: A Novel and Simple Method for Reducing Locking and Latching in Transaction Processing Systems*, **Proc. 16th International Conference on Very Large Data Bases**, August 1990.

[Moha93a] Mohan, C. *IBM's Relational DBMS Products: Features and Technologies*, **Proc. ACM SIGMOD International Conference on Management of Data**, May 1993.

[Moha93b] Mohan, C. *Transaction Processing System and Method With Reduced Locking*, **U.S. Patent 5,247,672**, IBM, September 1993.

[Moha94a] Mohan, C. *Concurrency Control and Recovery Methods for B^+-Tree Indexes: ARIES/KVL and ARIES/IM*, To appear in **Performance of Concurrency Control Mechanisms in Centralized Database Systems**, V. Kumar (Ed.), Prentice Hall, 1994. Also available as **IBM Research Report RJ9715**, March 1994.

[Moha94b] Mohan, C., Haderle, D. *Algorithms for Flexible Space Management in Transaction Systems Supporting Fine-Granularity Locking*, **Proc. 4th International Conference on Extending Database Technology**, March 1994.

[MoLe92] Mohan, C., Levine, F. *ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging*, **Proc. ACM SIGMOD International Conference on Management of Data**, June 1992.

[MoLi83] Mohan, C., Lindsay, B. *Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions*, **Proc. 2nd Annual ACM Symposium on Principles of Distributed Computing**, August 1983.

[MoLO86] Mohan, C., Lindsay, B., Obermarck, R. *Transaction Management in the R* Distributed Data Base Management System*, **ACM Transactions on Database Systems**, Vol. 11, No. 4, December 1986.

[SBCM93] Samaras, G., Britton, K., Citron, A., Mohan, C. *Two Phase Commit (2PC) Optimizations and Trade-Offs in the Commercial Environment*, **Proc. 9th International Conference on Data Engineering**, April 1993.

[SJFF89] Sanders, J., Jones, M., Fetvedt, J., Ferree, M. *A Communications Interface for Systems Applications Architecture*, **IEEE Journal on Selected Areas in Communication**, Vol. 7, No. 7, September 1989.

Real “Real Time Transactions”

Dr. Gary Bundell
University of Western Australia

Gene E. Trivett
University of Western Australia

1 Introduction

Having been a part of the development of the current “work horse” systems that are doing the major work in transaction processing today, we believe they should be recognized for what they are and not called legacy systems that really misrepresent what they are. That being said, the “work horse” systems do have some significant limitations, Among these is their ability to handle “real time” transaction processing. We refer to most of the transaction processing work being done today as event driven or dat a driven; here, many of the results of transaction are the changing of data from one state to another. Transferring money from one account to another, the issuance of an insurance policy, or the booking of a reservation are examples of event driven transactions. However, real time actions are a significant extension to the current focus of event driven or protected actions. It is our objective to discuss the additional Transaction processing services that must be made avail! able to perform transaction proce

To establish some common terminology, Gray and Reuter in their Transaction Processing [gray91] text outline three classes of action type :

1. UNPROTECTED, which do not have ACID properties.
2. PROTECTED, which have ACID properties.
3. REAL, which may or may not have ACID properties. They “need special treatment”.

It is the purpose of this paper to outline the “special treatment” that is needed for transactions that have REAL actions associated with them.

One major architecture and design point for the “work horse” systems was whether the transaction processing functions were “in” or “on” the operating system. Whether they were “open” or “closed” was mostly a marketing issue not a technical issue! I would note that in the early days of the “workhorse” systems, the source code was sent to the customer as part of the product, and the maintenance was done via source updates - it is hard to get more OPEN than that! In real time transaction systems, the de cision is clear - some transaction processing functions are “in” the operating system as an integral part. This is largely driven by performance. This forces an approach that addresses both the operating system and transaction management as a united effort. The time requirements of the transaction must be the same as the time management services of the operating system.

2 An Approach to Real-Time Processing

In some systems, notably those that at some point must interface with the physical world, there are hard time constraints that must be met in the responsiveness of certain sequences of transactions. No matter how that transaction processing system is configured to be responsive to these time-constraints they will not be met if the underlying operating system is unable to support a hard-responsiveness to external time-constraints. For this reason we briefly identify the available real-time operating system approaches, and then see how one of these operating systems can be used to support real-time transaction processing.

2.1 Real-time Operating Systems

There are four broad classes of real-time operating system [Levi90]:

- Priority-driven
- Priority-driven with Enhanced time services
- Time-driven Scheduling
- Deadline-Guaranteeing

The first two classes represent the vast majority of currently implemented systems, including those supporting commercial transaction processing systems. The third class attempts to address the problem in priority driven systems of missing deadlines (which is not permissible for hard real-time systems). The time-driven approach does this by allocating various value functions to measure the criticality of a task. Task scheduling is done on the basis of attempting to maximize the value functions of all tasks and the workload of all tasks The problem with this approach is that no guarantee can be given to meet particular deadlines although the extent of deadline misses would be significantly lower than for priority-driven schedulers. The four class of real-time operating system is the one we are most interested in since it provides an environment where transaction agents will guarantee to complete within a specific time-constraint.

2.2 Deadline-Guaranteed Transaction Processing

Using a Deadline-guaranteeing approach (an implemented example of which is the Spring kernel), we can assume that our operating system supports the setting and reading of the criticality for transaction agents and the imposition of time constraints on transaction agents. An important point is that transaction agents are only created by the scheduler per transaction so that their time criticality is directly related to the transaction time criticality. In implementation terms, the transaction agents would b e lightweight processes or tasks so that their creation overhead is minimal. Although the term 'lightweight' may have various interpretations depending on the context, here we are talking about POSIX-like multi-threaded operation.

A significant issue is that most deadline-guaranteeing operating systems are usually very restrictive on memory allocation policies and that pre-execution segmentation is used in preference to virtual memory systems. This means that transaction agents have fixed limits on their capabilities when invoked (although a range of pre-assigned capabilities can be provided). In most applications not all transactions will necessarily require hard-responsiveness, so that the capability to specify non real-time transactions and consequently obtain a more relaxed resource environment should be supported. Traditionally this has been achieved by a physical separation between systems, i.e.: a real-time 'front-end' with some loss in flexibility. However with the compute power available today, this separation should become a logical rather than a physical one.

The transaction scheduling function can be separated into three components: resource allocation, schedule feasibility verification and run-time scheduling. The function of the resource allocator is to determine the resource set needed to satisfy a specified transaction request, i.e. the transaction agents and their location and then obtain the resource and time constraint information from the participating distributed nodes. This information is used by the scheduler to verify feasibility of the modified op erating-system schedule and to determine the required time criticality of all transaction agents to meet it. If favourable, all participating nodes have transaction agents allocated for the specific time constraint window, which is propagated to these nodes. in the final phase of run-time scheduling, the transaction agents reach their defined constraint window and then execute, subject to minor local node scheduling variations. Because the issue of time-constraint propagati! on is an important one in this co

3 Transaction time-constraint propagation

Transactions are primary objects or justificands from the perspective of an object-orientated real-time system, And the transaction agents serve as joints between sources and destination nodes for the transaction object. As shown in the diagram below the transaction agents maintain a real-time calendar which holds the scheduled time-constraint windows for access to and from the transaction. This is illustrated by Figure 1:

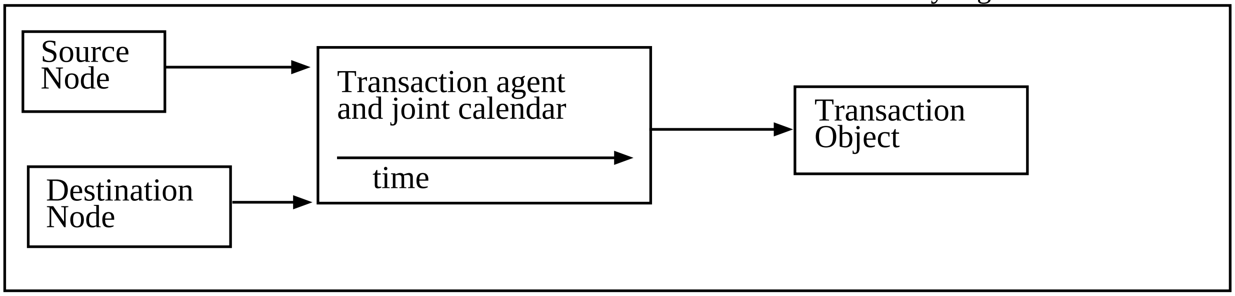

Figure 1: Transaction agent and Joint Calendar

The purpose of the calendar in the transaction agent is to allow the planning of transport for the transaction, i.e. to support the reservation of future time-constraint windows which in turn enforces deadline compliance. The time-constraints themselves are referenced to a global-time which is mapped to local time by each node responsible for the transport of transactions as shown in Figure 2:

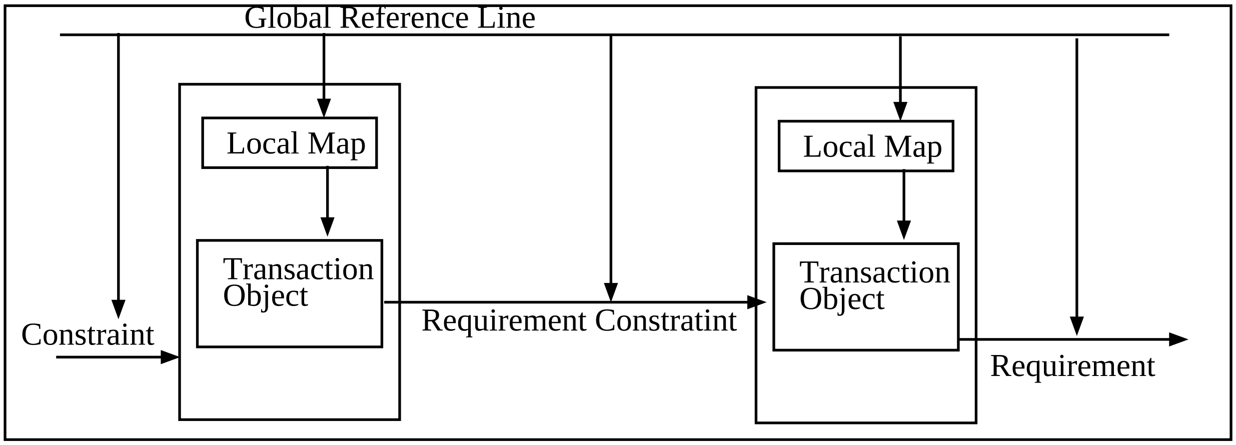

Figure 2: Time constraint relationships global to local

An important issue in distributed systems is fault tolerance in either the two forms of fault containment (graceful degradation) and fault recovery [Mullen89]. In real-time transaction systems the penalty for failing to recover a transaction is just as severe as missing deadlines. Thus transaction recovery agents must be scheduled with sufficiently high time criticality to achieve this - in fact node-based local schedulers must reserve a transient transaction recovery procedures. Transaction agents are deliberately not terminated until a transaction has been two-phase committed and no further transport operations for that transaction are currently scheduled within the time horizon. This non-termination of agents is the basis for the recovery process. There are other issues associated with recovery that relate to the dynamic nature of real-time applications which also need to be considered.

3.1 RECOVERY CONSIDERATIONS

When one characterizes the effect of having a major change in the environment when recovery occurs, then the approach to recovery is very different. Transactions that exhibit REAL actions are probably idempotent and the time to complete recovery probably determines the extent of the environmental change. For example in mining operations the conveyor belts are moving and continue to move during recovery; or a liquid flowing through a major pipeline will continue to flow even during recovery. Hence real time transaction processing requires some different recovery models. The major recovery models used in "work horse" systems and even newer transaction systems are as seen in Figure 3:

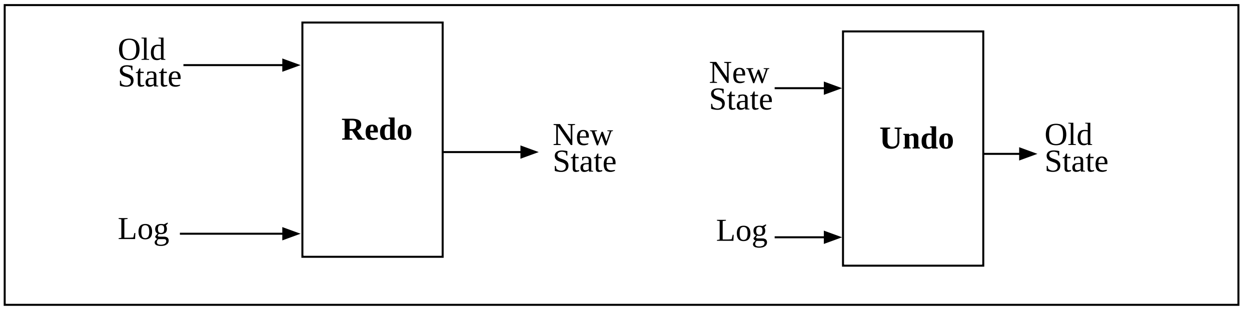

Figure 3: Standard (Work-horse) recovery model

However, a new model for real time transaction processing systems that is more useful is as shown in Figure 4:

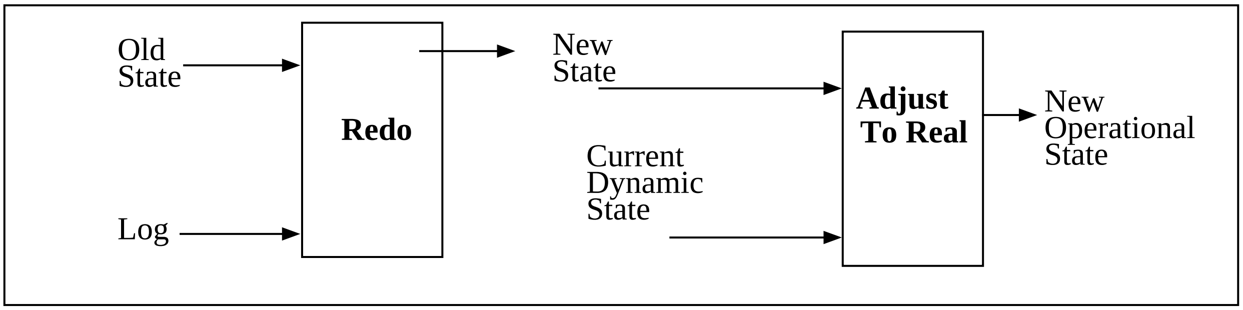

Figure 4: Real-Time recovery model

It is the addition of CURRENT DYNAMIC STATE and ADJUST TO REAL that are needed in real time transaction systems. In effect, the addition of CURRENT DYNAMIC STATE forces the transaction system recovery (ADJUST TO REAL) to take into account the state of the reality that exists in the real world; this is in addition to any compensating transactions. The objective is that once the real world state is known (CURRENT DYNAMIC STATE), then the transaction manager can again process transactions that affect the oper ation of the REAL process. Now, a key point here is that CURRENT DYNAMIC STATE is changing even during recovery (for example a conveyor belt is still moving or liquid is still flowing). The longer to recover, the more the ADJUST TO REAL is required to do.

4 Conclusions

This paper has addressed some of the salient issues for real-time transaction processing. In particular, the important issue of the connection between transactions transaction agents and the underlying real-time operating system has been covered. The related issue of time constraint propagation in a real-time transaction architecture has also been considered. The approach taken to recovery management is another issue. A mechanism was proposed whereby the dynamic state of a real-time process (held by the s ystem) that is involved in transaction exchange, can be adjusted.

5 Short Biography

Gary A. Bundell received the B.E. and M.Eng.Sc. degrees in electrical engineering from The University of Western Australia in 1978 and 1981, respectively, and the Ph.D. degree in control engineering from Cambridge University, U.K. in 1985. From 1986 to 1990 he was progressively s Senior Research Engineer, Research and Development Manager (Electrical), and Research and Development Manager with ACET Ltd. In 1991 he joined the faculty of The University of Western Australia where he is currently a Senio r Lecturer. His research interests include real-time distributed information systems design and analysis and his professional affiliations are MIEEE, MIEAust, and AMIEE.

Gene E. Trivett worked at IBM Corp. for thirty years as a senior developer involved in the research and development of a large number of systems including ESA, IMS, AI products, and communication products. He worked for HaL Computers for one year prior to his taking his current position of Senior Lecturer at The University of Western Australia. He is a graduate of UCLA and has a number of patents and publications.

References

[gray91] Gray, J., and A. Reuter, “Transaction Processing: Concepts and Techniques.” Morgan Kaufmann, San Mateo, CA, 1991

[Levi90] Levi, S.T. and Agrawala, A.K., “Real-Time System Design”, McGraw-Hill, 1990.

[Mullen89] Mullender,S.J., “Distributed Systems”, Addison-Wesley, 1989.

The TUXEDO^{TM} System: An Open On-line Transaction Processing Environment

Juan M. Andrade, Mark T. Carges and M. Randall MacBlane
Novell Inc., 190 River Road, Summit, NJ USA 07901

Abstract

On-line transaction processing (OLTP) users are moving from monolithic proprietary solutions (e.g., CICS) to open distributed solutions. Open distributed TP environments blend together standards-based interfaces, processing models and protocols into a flexible solution for OLTP users. The TUXEDO^{TM} Enterprise Transaction Processing (ETP) System [TUXEDO] provides a UNIX^{TM} System-based transaction processing monitor (TPM), System/T, designed to enable the integration of many different components of OLTP in an open systems environment. This paper discusses the requirements driving open OLTP in today's market and an overview of the TUXEDO products designed to meet those requirements.[1]

1 Open OLTP Requirements

As OLTP systems and applications are being deployed on open platforms, the list of requirements for OLTP system providers is growing. Not only are many of the traditional requirements, such as supporting hundreds of users performing short interactive tasks, still valid but there are also new requirements specific to open systems environments. This section explores many of these requirements.

Open OLTP systems have been driven by industry needs to support decentralized applications in order to decrease high computing costs. These systems must be able to handle distributed and heterogeneous configurations of software, hardware and networks. They must provide mechanisms for the integration of multiple tiers of processing, from PCs to Mainframes. Such integration requires a consistent set of application programming interfaces (APIs) across a broad span of computing environments, and the coordination of transactions across multiple heterogeneous resource managers (RMs).

The X/Open DTP Model [XOPEN] addresses the issues of consistent APIs and some level of component interchangeability, but does not completely address the total OLTP environment. Open OLTP systems still require a great deal of flexibility to integrate other required software provided by different vendors. What follows is a list of requirements that OLTP system providers must meet for open systems platforms:

Freedom of choice - vendor independence. The main thrust behind open OLTP systems is that customers have the freedom to choose the parts of their computing environment based on their needs. This includes the hardware they buy, the way in which it is networked together and the different software components. A TPM must therefore support standard interfaces like X/Open's XA interface [XA]. The XA interface allows applications to easily integrate different RMs (e.g., DBMSs) while maintaining the transactional data integrity they currently have in a single vendor environment.

Application portability. Open OLTP application developers need the same application development environment across different hardware and software platforms. This requirement calls for a standard set of APIs that

[1] TUXEDO is a registered trademark of UNIX System Laboratories, Inc., a wholly-owned subsidiary of Novell Inc, 190 River Road, Summit, NJ USA 07901. UNIX is a registered trademark of the X/Open Company Limited. Other brand and product names referenced in this document are trademarks or registered trademarks of their respective holders.

span different computing bases (that is, from the PC, to the UNIX server, to the proprietary mainframe). These APIs allow application developers to take advantage, for example, of different graphical user interfaces (GUIs) across dissimilar platforms while preserving the same OLTP transaction and communication semantics. X/Open has defined three such interfaces to date. The first, XATMI [XATMI], provides application buffer management and both request/response and conversational style communication mechanisms. The second, TxRPC [TxRPC], provides a Remote Procedure Call, or RPC, interface. The third interface, TX [TX], provides transaction control verbs for transaction demarcation. Together, these interfaces define a powerful set of tools that allow portable applications access to the power of transaction processing in an open distributed environment.

Distribution transparency. Communications paradigms in open OLTP systems must present location transparent communication services to application programmers. This transparency must allow application programs to be completely independent of the environment with which they communicate. This includes independence from the hardware on which the other program is running, the language in which it is written, the data format it expects and the TPM system by which it is administered. As systems become more distributed, the need for location transparency becomes greater.

Performance, modular growth, and scalability. OLTP applications require high throughput and short response times. Distributed open OLTP must also allow application growth through the orderly expansion of the hardware, system software and application software on different computing bases.

Reliability, robustness, and reconfigurability. In a distributed environment, an open OLTP system must be resilient to failures and it must provide mechanisms to prevent application data from becoming inconsistent. OLTP systems provide the concept of a distributed transaction to ensure that data at all sites remains consistent.

Monitoring and administration. OLTP systems must allow systems administrators to tune the load distribution of their application, and to schedule the priority of work. In distributed environments, there is also an interest in automating the start-up and shutdown of an application (or parts of it). Such automated support should be controlled from a central place. That is, OLTP systems administrators need to have a centralized view of a distributed application.

To illustrate how the requirements of open OLTP can be met, the TUXEDO ETP System will be described. First, a high level description of the TUXEDO ETP System distributed architecture is given to show the various computing bases that an open OLTP system must span. Next, to motivate how the diverse requirements of open OLTP are met with a simple but powerful software architecture, the TUXEDO ETP System computational model is described. Lastly, the individual components of the TUXEDO ETP System architecture are briefly described.

2 Overview of the TUXEDO ETP System

The TUXEDO ETP System distributed architecture allows a set of heterogeneous computer nodes to cooperate towards the efficient and reliable execution of OLTP applications. Figure 1 depicts the different computing bases on which the TUXEDO ETP System runs (e.g., workstations and UNIX-based server nodes). It also illustrates that it interoperates with other OLTP environments, perhaps dissimilar from its own (e.g., IBM MVS/CICS).

The TUXEDO ETP System distributed architecture has the following main components:

- *System/T nodes*. System/T is the TPM of the TUXEDO ETP System. System/T nodes are the fundamental nodes in a TUXEDO ETP System application. Because the TUXEDO ETP System runs on a variety of open systems platforms, the computing base that a System/T node is defined on is provided by many platform vendors.

- *Workstation nodes*. Applications can offload CPU cycles used for forms processing (or other input activities) from System/T nodes to workstation nodes, thus improving the overall performance of an OLTP appli-

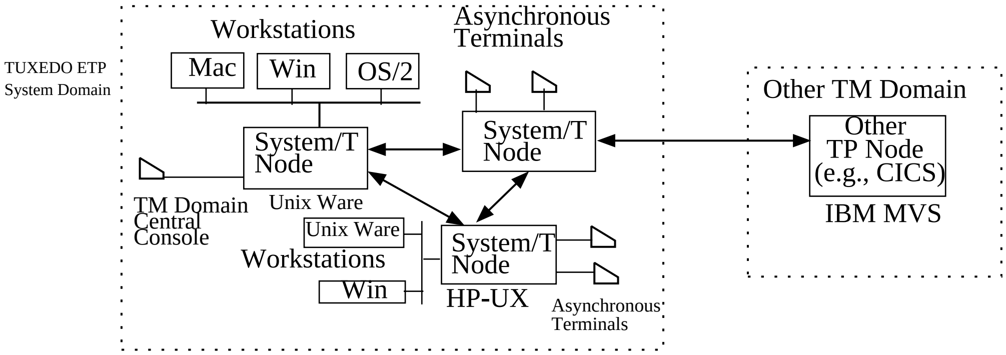

Figure 1: TUXEDO ETP System Distributed Architecture

cation. Applications running on workstations nodes are requesters of OLTP services offered on other System/T nodes. The TUXEDO ETP System supports PCs running MS-DOS, Windows, Macintosh System 7, OS/2, Windows NT and workstations running various versions of the UNIX operating system. Workstations nodes can be connected to System/T nodes using many types of networks (e.g., TCP/IP, OSI, IPX/SPX).

- *TM Domains*. A TM Domain represents an independently administered set of TP nodes, whether they be System/T nodes or other TP nodes. System/T nodes can be connected to other System/T nodes using many types of networks and transport interfaces. Interaction between heterogeneous TM domains, i.e. domains controlled by different TPMs, is performed using standardized protocols. Interaction between homogeneous TM domains, e.g. two TUXEDO ETP System Domains, is often performed using more efficient customized protocols.

- *Other TP nodes*. These nodes are part of a cooperating TM domain that is interacting with the local TUXEDO ETP System domain to form a larger application. As was mentioned above, these other domains may be either homogeneous (e.g., other TUXEDO ETP System domains) or heterogeneous (e.g., IBM MVS/CICS).

In a distributed, heterogeneous environment, as described above, a computational model that simplifies the development of an open OLTP application and hides the complexity associated with distribution application processing is required.

3 Client/Server: A Computational Model for Open OLTP Applications

A client/server model is ideal for building OLTP applications in distributed environments. This approach consists of splitting the application into two types of software components, clients and servers. Clients gather input from terminals (or special devices) and construct service requests. These service requests refer to operations implemented within a server. Hence, a server's main function is to process clients' service requests and send them replies indicating the outcome of their requests.

The client/server model has important properties for OLTP applications:

- *Modularity*. It provides a modular approach to building distributed applications. Such modularity allows for extensible and scalable applications that can grow with an enterprise's needs.

- *Performance*. By conserving system resources (e.g., many clients can be served by a few servers), the client/server model permits applications to maintain short response times and high transaction throughput.

- *Data independence*. Clients have to know very little about the structure of servers. They need to know only the name of the service they want performed and the parameters that service expects. Because the service routines hide the actual data access methods, clients have no idea what kind of DBMS is being used. In fact, the DBMS accessed within a server can be changed with little or no impact on the clients.

- *Location transparency*. Because of the separation between client and server processes, clients communicate with servers using abstract, location independent names. This allows servers to migrate across computing bases depending on resource needs with no affect on clients.

The TUXEDO ETP System exploits the client/server computational model towards the goal of enabling high performance OLTP applications. Because of the special needs of OLTP applications, the TUXEDO ETP System recognizes the following programming paradigms: request/response, conversational, store-and-forward queuing, event notification and remote procedure call (RPC). The first four of these paradigms are provided through an API, called ATMI (a superset of XATMI), that was designed to meet the requirements of open OLTP. The last paradigm is provided through X/Open's TxRPC interface which adds transactional syntax and semantics to DCE's RPC and IDL.

ATMI's request/response verbs provide a dynamic, run-time resolution of service names. These verbs also provide support for synchronous and asynchronous service requests. Asynchronous service requests allow several requests to be executing in parallel and are particularly well suited to distributed environments. In conjunction with the transaction management verbs of the TX interface, several requests can be grouped together within a single transaction. Application programmers also have control over the priority at which a service request is handled by a server. Because of the high level of abstraction in ATMI, TUXEDO ETP System/T is able to handle data presentation services automatically. These data presentation services include transparently encoding/decoding application data as it passes between nodes having different processor types. Services defined using ATMI can return replies to clients, or forward work to other servers that take over the responsibility of sending the originating client its reply. This ”bucket-brigade” style of communication is well suited to distributed environments as it keeps a high percentage of servers busy, and working in parallel, during peak periods.

While the request/response paradigm is simple and powerful, it is not ideal for applications that require either bulk data transfer or incremental results. These two styles of communication are best performed by a conversational paradigm that allows a client and a server to communicate more than just a single request and response. Part of ATMI, therefore, is an interface allowing clients and servers to communicate using a simple, half-duplex conversational paradigm while maintaining the premise of location transparency.

At times, applications require time independent disk-based queuing so that requests may be queued for parts of the application that may not yet be available. Queuing of request/response requests must be controlled by the same transactional boundaries that bracket the services being requested. These capabilities are provided by the store-and-forward features of the ATMI interface.

While ATMI provides a complete and powerful communication interface, many users desire the simplicity that comes with using a remote procedure call interface. X/Open has defined the TxRPC interface to support this type of user in a transaction processing environment. The TUXEDO ETP System's TxRPC interface allows application programmers the freedom to program to function call interfaces while leveraging the functionality of a complete transaction processing environment.

4 Architecture of the TUXEDO ETP System

Figure 1 shows the components of a TUXEDO ETP System application. These components provide the fundamental services required by open OLTP applications shown in the figure below. These services provide an open systems solution to the Open OLTP requirements specified earlier.

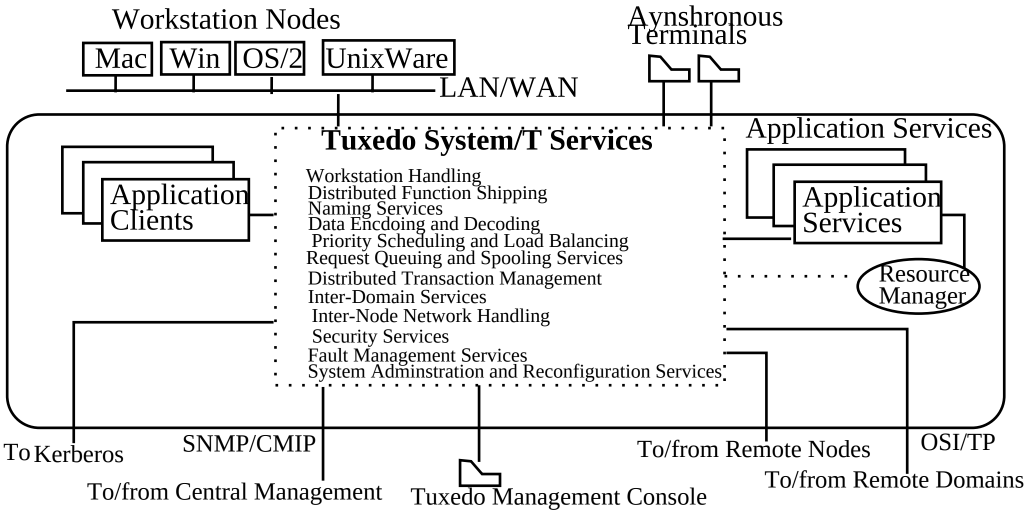
Workstation Nodes
Mac
Win
OS/2
UnixWare
LAN/WAN
Aynshronous
Terminals
Tuxedo System/T Services
Workstation Handling
Distributed Function Shipping
Naming Services
Data Encdoing and Decoding
Priority Scheduling and Load Balancing
Request Queuing and Spooling Services
Distributed Transaction Management
Inter-Domain Services
Inter-Node Network Handling
Security Services
Fault Management Services
System Adminstration and Reconfiguration Services
Application Services
Application Clients
Application Services
Resource Manager
To Kerberos
SNMP/CMIP
To/from Central Management
Tuxedo Management Console
To/from Remote Nodes
OSI/TP
To/from Remote Domains

Figure 2: TUXEDO ETP System/T Services

Freedom of choice (vendor independence) is provided through support for X/Open's XA interface. This interface allows an application to use the TUXEDO ETP System with heterogeneous RMs (e.g., DBMSs) in a single application without sacrificing data integrity for transactions that span RMs. Vendor independence is also provided at the hardware level by the availability of the TUXEDO ETP System on more than 20 different platforms.

Application portability is provided at the hardware level by the availability of the TUXEDO ETP System on a wide range of open platforms, at the language level by the availability of the programming interfaces in C, C++ and COBOL, and at the API level through support of the communication and transaction APIs defined by X/Open. **Workstation handling** allows applications to migrate client code based on these APIs to workstations. This in turn allows application builders to utilize the graphical user interfaces on the platforms of their choice.

Distribution transparency. The TUXEDO ETP System **naming services** provide location transparency for service requests. Clients use abstract names, such as "DEBIT," to call on a service. System/T's naming services are responsible for resolving these abstract names and routing the request to the appropriate server. System/T's naming services contain information about application services located within a TUXEDO ETP System domain as well as about those known to be located in other TM Domains. **Priority scheduling, context-sensitive routing, and load balancing** allow an application to specify additional criteria to be used in choosing the best service for a particular request. Automatic **data encoding/decoding services** provide hardware independence to application programmers so that they do not need to know when a particular request might be sent to a different type of machine for processing.

Performance, modular growth, and scalability are supported by the client/server model used within the TUXEDO ETP System. This model allows a few servers to support many clients while allowing additional servers to be added to meet increasing demand. The hardware and vendor independence mentioned earlier allow applications to expand by adding machines and RMs of different types as needed. The client/server model is based on **distributed function shipping** which saves data transfer and processing time by implementing services near the data rather than retrieving large amounts of data for processing by a front-end client. Additionally, **inter-domain services** allow applications to share information between cooperating application domains without sacrificing the administrative integrity of either domain. Inter-domain services are available to IBM MVS/CICS via LU6.2 protocols, to other TUXEDO ETP System domains via proprietary protocols and through the OSI-TP protocol to any other TPM meeting this protocol.

Reliability, robustness, and reconfigurability. **Fault management services** increase the availability of an application by allowing servers and services to be replicated across several System/T nodes. Also, System/T monitors the viability of the different components in a System/T node, such as machines, networks and processes.

When a failure is detected, automatic recovery or automatic server re-start is performed. **System administration, monitoring, and reconfiguration services** provide capabilities for transaction monitoring and on-line reconfiguration. **Distributed transaction management services** control the two-phase commitment mechanism required for interoperation with X/Open XA-compliant resource managers. the use of a two-phase commitment protocol and logging service. **Request queuing and scheduling services** allow applications to store requests in a stable queue and to control their delivery to application servers at a later time.

Monitoring and administration. **System administration, monitoring, and reconfiguration services**, as mentioned above, provide monitoring and reconfiguration capabilities. An administrative API is also provided that is used to create customized GUI administrative interfaces and SNMP or CMIS administrative gateways for example.

5 Summary

Open On-Line Transaction processing is a rapidly evolving area that requires support for distributed and heterogeneous configurations of software, hardware and communications networks. The architecture of the TUXEDO ETP System presented in this paper shows that Open OLTP is viable. The paper illustrates how the different components of the TUXEDO ETP System and the client/server model provide an Open OLTP environment.

References

[TUXEDO] Novell, “TUXEDO^{TM} ETP System Release 4.2 Product Overview”.

[TX] ‘Distributed TP: The TX Specification”, X/Open Company Ltd., Preliminary Spec., November 1992.

[TxRPC] ‘Distributed TP: The TxRPC Specification”, X/Open Company Ltd., Preliminary Spec., July 1993.

[XA] ‘Distributed TP: The XA Specification”, X/Open Company Ltd., CAE Spec., February 1992.

[XATMI] ‘Distributed TP: The XATMI Specification”, X/Open Company Ltd., Preliminary Spec., July 1993.

[XOPEN] ‘Distributed TP: Reference Model, Version 2”, X/Open Company Ltd., November 1993.

Services for a Workflow Management System

Hector Garcia-Molina [†]
Stanford University

Kenneth Salem [‡]
University of Maryland

Abstract

This paper represents an effort to understand the problem of workflow management. Our goal is not to propose yet another model or system. Instead, we give examples of the types of services that a workflow management system may provide.

1 Introduction

Traditionally, database management systems (DBMSs) have had a very "data centered" view of the world. An application program is simply an entity that issues a transaction. It starts interacting with the DBMS by issuing a "begin work" command, then emitting a series requests to access and modify data, and finally ending with either an "abort work" or a "commit work" command. After that, if the application issues other transactions, it is considered to be a different program. It is of no concern to the DBMS what happens to the program if its work is aborted or if a system crash occurs, or how one program is related to another.

There has been recent interest in providing more comprehensive support for data intensive applications. This general research area has been referred as either workflow management, new transaction models, activity management, or third generation transaction monitors [Elma93, Daya93].

To illustrate some of the issues, let us consider a typical application or *activity* that accesses one or more database systems. This activity processes a purchase order at some company, and can be represented as a collection of *steps*. The activity starts when a phone call is received to place a new order. In this "phone call" step a clerk enters the required information into an electronic form. The next step, "enter order," is to run a transaction to record the new order in the database. After the order is entered, two parallel steps are executed. The "billing" step charges the customer and the "inventory" step updates the inventory. Each of these steps may involve a transaction that reads and updates the database. After these transactions complete, a final "shipping" step is executed to generate the appropriate shipping orders.

Activities sometimes fail. For example, if the "billing" step discovers that the customer's credit is not good, then we may want undo some of the work done and perhaps perform some alternative. For instance, we may run a "delete order" step to record the fact that the order has been canceled. Note that this is not simply an "undo" of the original "enter order" transaction. That transaction already committed; furthermore, we want a record of the canceled order. In addition, we may want to initiate other steps, like generating a letter to the customer.

Each step of the activity may run as a separate process (or thread or whatever unit of processing the operating system supports). To implement this activity, the application programmer must write code for:

- Starting up and initializing the processes;

[†] Department of Computer Science; Stanford, CA 94305-2140; email: hector@cs.stanford.edu

[‡] Department of Computer Science; Collage Park, MD 20742; email: salem@cs.umd.edu

- The internal logic of each step (e.g., how to bill a customer). This code may call on a DBMS to access data, and to start or terminate transactions;
- The overall flow of the activity, e.g., when can the "shipping" step start up,
- Inter-process communication (e.g., the part number ordered must be sent from the "enter order" step to the "inventory" step);
- Detecting and coping with failures of the processes.

The goal of a *Workflow Management System (WFMS)* is to support activity programming, that is, to make it easier to program activities like the purchase order one. There has been a great flurry proposed "new models" that a WFMS should support, like Contract Nets, Sagas, Split-Transactions, Flex Transactions, and S-Transactions, among others. There have also been proposals for "took kits" for allowing the definition of new models, for formalisms for describing transaction models, for new languages for describing activities, and for execution systems that can support a variety of "transaction models" [Elma93, GaSa87, GrRe93, Hsu93].

To understand what all these "models" are contributing, we think it is important to separate the two main components of a WFMS: the *services* and the *programming environment*. The services provide the underlying, operational functionality for activity programming while the environment provides the language and tools for the programmer to express the logic of the activity. This is analogous to the role operating systems (OS) and programming environments (PE) play in general purpose computing: The OS provides processes, interprocess communication, scheduling, and so on, while the programming environment provides a way to write programs and represent variables.

The environment and the services are of course closely related, but in a way are also orthogonal. For example, consider a trigger service that initiates processes when certain conditions occur (e.g., when a transaction aborts). This service may be made available in the language through an "on abort:" statement. On the other hand, it could be made available in an implicit way, e.g., because steps like "billing" and "enter order" are defined to be part of the same activity, then an abort of "billing" may automatically imply that the compensation for "enter order" is run. Similarly, an environment concept such as shared step variables can be implemented in several ways, e.g., by storing the variables in a shared database, or by exchanging messages between steps when the variables are updated. The models suggested in the literature usually combine service and environment ideas. We believe that by separating the components, one can gain a better understanding of what each model contributes.

In this paper we briefly illustrate some of the basic services a WFMS may provide, independent of the programming environment. We start by illustrating, through an example, how hard it is to program activities if the WFMS provides no support beyond what the OS provides (Section 2). Then we suggest some simple but powerful services that can make the programming task simpler (Sections 3 and 4).

2 Programming an Activity

As a running example for this paper, let us consider an even simpler activity that the one in Section 1. The activity consists of two "forward" steps. The first step A increments a variable X stored in a database and the second step B modifies a database variable Y. The activity has a constraint that if X and Y cannot both be modified properly, then any changes made to either value should be compensated for.

Suppose that a programmer has available some common services, such as those provided by a simple operating system, a transaction manager, and a DBMS. What additional services must be provided by a WFMS to make it possible to program this (or any other) activity? The is answer is: none! Our activity is just a program, so it should be implementable on any computer.

The problem is that the program would be quite messy because it has to recover from failures. In particular, while the steps executed, they would have to write detailed state information to disk, to allow recovery. For example, before the database transaction for step A commits, we need to write to disk any parameters that may be needed by the compensation step. We also need to record the transaction id for this transaction, so that at recovery time we may determine its fate (by querying the DBMS).

At recovery time, an application specific program would have to be run to determine the actions to take. In particular, since database transactions and processes that execute them are independent, there are many states to consider. That is, the processes that were executing the activity may or may not be running (perhaps they failed, perhaps they were never created). Similarly, the transactions may have or may have not committed. For each scenario, we take different actions, re-starting processes, aborting or committing open transactions, runing compensations if necessary, and so on.

3 Binding Processes and Transactions into Steps

The WFMS can simplify matters tremendously if it provides the concept of a *transactional step*. The idea is that a process and a database transaction, called its *underlying transaction*, are bound together as an atomic unit. Any actions, e.g., creation of other steps, do not take effect until the underlying transaction commits. The process may request that its current underlying transaction be committed or aborted, to be replaced by a new one.

The WFMS provides "create-step()", "commit-step()", and "abort-step()" services. The latter two services cause the current underlying transaction and the process to terminate. In addition, "commit-and-chain()" and "abort-and-chain()" are provided to allow a step to terminate its underlying transaction and start a new one, without terminating the process.

Using these new facilities, our example could be coded as follows.

```
program-A:
x := read-db(X);  /* get value of X from the database */
x := x + 1;
write-db(X,x);  /* write the new value of X */
create-step(program-B, parameters);
commit-step();

program-B:
y := read-db(Y);  y := f(y);  write-db(Y,y);
res := commit-and-chain();  /* returns result of commit request */
/* new underlying transaction begins here */
if (res = aborted) then create-step(program-C, parameters);
commit-step();

program-C:
x := read-db(X);  x := x - 1;  write-db(X,x);
commit-step();
```

In our example, we assume that the WFMS maintains the mapping from processes to underlying transactions. Thus, when step A issues the "read-db" call, the transaction id of the underlying transaction is automatically passed on. The same holds for other calls.

Recovery is simplified because all of a program's actions are part of an underlying transaction. In fact, a step that has only a single underlying transaction can simply be rolled back and restarted. For example, if there is a system failure before step A commits, then it can be redone. However, recovery is still not completely application independent. One reason is that a failure may occur part of the way through a step that uses more than one underlying transaction. For example, program-B's execution may fail after the "commit-and-chain()" but before "create-step()". The proper recovery action (creating a step to run program-C) in that case is a matter of application semantics.

The WFMS may provide additional services that allow an application to define how recovery should be performed in such cases. For example, the WFMS could support *persistent triggers* to be executed in case a transaction enters a specified state. Program-A could be re-written to use triggers as follows.

```
program-A:
x := read-db(X);  /* get value of X from the database */
x := x + 1;
write-db(X,x);  /* write the new value of X */
s := create-step(program-B, parameters);
on-abort(s,program-C,parameters); /* trigger program-C in case program-B fails */
commit-step();
```

The *on-abort* command triggers a step to run Program-C in case the step running program-B aborts. (Of course, Program-B would also have to be modified so that it did not call create-step().) Like the other WFMS services, the trigger commands are transactional. This means that both the program-B step and the trigger will exist if and only if the program-A step commits.

4 Activities

In our previous example, the fact that a series of steps comprises an activity is known only to the application. Compared to processes, or transactions, or steps, activities are clearly second-class entities.

A WFMS can alleviate this problem by maintaining a activity status, or context, in a database. The WFMS may permit an activity's steps to access and modify the context.

Our second example illustrates how our running example might look if the WFMS implemented activity contexts. We have assumed that the new activity services are transactional, like other services. We have also assumed that the WFMS maintains the mapping of steps to activities, so that explicit activity identifiers are not needed in the programs.

```
program-A:
create-activity-context();
x := read-db(X);   x := x + 1;   write-db(X,x);
s := create-step(program-B, parameters); /* new step, part of same activity */
commit-step();

program-B:
y := read-db(Y);  y := f(y);  write-db(Y,y);
res := commit-and-chain();
/* new underlying transaction starts here */
if (res = aborted) then  s := create-step(program-C, parameters);
    else  set-activity-context("success");
commit-step();

program-C:
x := read-db(X);  x := x - 1;  write-db(X,x);
set-activity-context("failure");
commit-step();
```

The "create-activity-context" call in step A tells the WFMS that this step is the beginning of a multi-step activity. If A commits, the context is created. If other steps inquire about the state of A after it has committed, they are *not* told of the commit; instead they are returned the state of the activity context, in this case still active. Only after B successfully commits, is the context set to "success" and other steps would be then told that A has successfully committed. If the compensation step C is run, the activity has logically failed, and an "unsuccessful" status would be returned.

The basic idea is that to the step that invoked A the execution of multiple steps is transparent. The invoker sees the entire activity as a single step. This makes it easy to compose activities out of activities in a recursive fashion, a key to modular systems.

5 Conclusions

Given our space limitations, we have only illustrated two types of WFMS services that can simplify the activity programming effort. These examples omitted many details, and of course, these are not the only ways to define steps and activities. Furthermore, there are several other types of useful WFMS services that we did not cover.

The key point we are trying to make is that a good WFMS must provide some low level but activity-specific services, beyond the traditional operating and database services. These services must be based on clean abstractions such as transactional steps, triggers, and activities.

References

[Daya93] U. Dayal, H. Garcia-Molina, M. Hsu, B. Kao, M. Shan, Third Generation TP Monitors: A Database Challenge, Proc. of the ACM-SIGMOD International Conference on Management of Data, May, 1993, pp. 393-397.

[Elma93] A. K. Elmagarmid (editor), Database Transaction Models for Advanced Applications, Morgan Kaufmann, 1992.

[GaSa87] Garcia-Molina, H., K. Salem, "Sagas," Proc. of the ACM-SIGMOD International Conference on Management of Data, May, 1987, pp. 249-259.

[GrRe93] Gray, J., Reuter, A., Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993.

[Hsu93] M. Hsu (editor), Bulletin of the Technical Committee on Data Engineering, Special Issue on Workflow and Extended Transaction Systems, 16, 2, June, 1993.

Transaction Processing at Microsoft: Present and Future

Patrick O'Neil, Mohsen Al-Ghosein, David Vaskevitch, Rick Vicik,
Laura Yedwab:
poneil@cs.umb.edu
{v-poneil, mohsena, davidv, rickv, lauraye}@microsoft.com

1 Introduction

The Microsoft vision of transactional database systems over the next few decades is for an increasingly pervasive market for packaged products. Microsoft today has a strong presence in software that empowers individuals, software that can be taken out of the box, plugged in, and used immediately, often without even the need to read the documentation. The Microsoft Database and Development Tools (DDT) Division is currently gearing up for a major effort to extend this strength. The plan is to provide an integrated database system that will work on a large NT server with high concurrency in one configuration, or a small laptop with reliable deferred workflow scheduling in another. In both environments this will be an industrial strength, high reliability system, with sophisticated transaction coordination and package ease of use. We try here to give some idea of the design concepts that will bear on distributed transactions.

As Jim Gray and Chris Nyberg point out in their Compcon-94 paper [GRAYNY], the Microsoft Access database product costs about $100 and has several million licenses, whereas IBM's DB2 costs over $100,000 for an initial license fee and has about 10,000 licenses. Both systems generate $300M in annual revenue, and therefore can sustain comparable engineering organizations. Now the Access product has only been available for a few years and is still in an immature phase of its life cycle, so to consider a more mature market we look to computer processor hardware. In [VASK], it is noted that there are about 30,000 mainframes in existence (characterized as costing $1,000,000 by [GRAYNY]) and 300,000 big minis (or departmental servers, costing $100,000), while there are more than 100M PCs and laptops, costing well over $1000 each. In this market the center of gravity of weighted investments ($30B for mainframes, $30B for departmental servers, and $100B++ for PCs) has already moved well into the PC realm. Taking into account the continuing thrust for downsizing in business and the growing market for home applications (many of which have not yet been invented), this trend can be expected to continue into the foreseeable future.

Looking perhaps a decade into the future, we can easily imagine a world in which there are 100M database servers world-wide (see [VASK] for an extended treatment of this scenario). These will be servers in the sense of being continuously available on the net as agents for a home or business, to transact needed interchanges; home servers will normally not have multiple concurrent request threads active, although concurrent thread execution will continue to be an important feature. As we see with telephone use today, much of the home server net traffic will interact with businesses to make appointments, order and pay for services and goods, and so on. But more personal uses will also exist: to receive and screen email messages, make personal appointments, etc. Dis- tributed database transaction capabilities will be crucial to making these services a reality. We can also expect to see several hundred million notebook PCs that people carry with them, many of these not continuously connected to the net because of volume limitations in cellular phone traffic. Notebook users will make at least daily contact with their office and personal servers in order to back up work and continue long-running transactional workflow that has made progress while disconnected from net communication, for example dealing with details of a computer conference. If this scenario seems too visionary, the reader should know that many of these capabilities are already being supplied by large companies, such as accounting firms, for knowledge workers who spend a good deal of time traveling. Just as the sophisticated word processing machines in 1980 offices became the ubiquitous PC word processing software of 1990, we believe that the transactional systems are about to enter a shrink-wrapped phase: the scene is set for another fundamental shift to home use.

At the same time that there is an explosion in home use of database systems, office use will expand as well. Every dentist and take-out restaurant will have an inexpensive office server to deal with customer's personal servers to make appointments and place orders. It is possible that some applications such as airline reservations will retain a presence on large mainframe servers because of a need to centralize flight reservation data, although large centralized parallel databases of the Teradata model make this uncertain. In any event, cost-performance considerations will probably force much of the work currently performed centrally on behalf of about a hundred thousand reservation terminals to become local on the office server. As with grocery stores of today, an inexpensive local server can collect information (grocery reorder, commissions for reservation agents) that can then be cheaply centralized.

All of these observations have a number of important implications for the design of transactional database systems of the future.

- Systems must be easy to install and maintain. This is an aspect of a shrink-wrapped package product: people will want to be able to use their database server with no training. We shouldnUt imagine having SEs around to install 100M copies of CICS.

- Systems must be easy to query. Most people will not write programs, or even use SQL. The Access and MS Mail browsers provide a good initial model for what is needed.

- Easy program development for office servers. We should picture a large number of vertical markets where entrepreneurial programmers provide customized extensions to standard frameworks for small businesses. The Visual Basic interface to databases gives a good first cut picture of this.

- Systems must provide a sophisticated paradigm for distributed transactions. Standard 2PC protocols are only the beginning. Replication management with weak consistency will be needed, and a Transaction Manager (TM) to deal with appropriate Workflow models such as given in [GHO], Sagas [GARSAL, GGKKS], ConTracts [WÄCHTREU], ACTA [CHRYKRI], etc.

As we will try to illustrate in this paper, there are a number of aspects of database theory that must be rethought and applied in completely new ways to properly address some of the challenges arising in these new database systems. In Section 2, we discuss Microsoft SQL cursors available today and a few details of the transactional programming interface provided. In Section 3, we discuss a number of distributed transaction concepts needed in future products.

2 Cursors and the Transactional Programming Interface

To provide an easy to use browse interface to data, programs often need to concurrently access a database server and display results on a client screen for interactive scrolling and update. The standard cursor of today must be enriched in a number of ways to provide a programming model that makes this easy to accomplish. We list a number of features to address this, contained in the ODBC standard and implemented in Microsoft SQL Server, and known as the Microsoft Scrollable Cursor API (or MSC-API, see [VICIK]).

- The lack of backward scrolling in SQL-89 has long been a handicap for applications that require the ability to scroll backwards at the request of interactive users with scrollbars. The MSC-API provides essentially all the capability in SQL-92, the ability to fetch: NEXT, PREV (SQL-92 ”PRIOR”), FIRST, LAST, RANDOM (SQL-92 ”ABSOLUTE”), or RELATIVE.

- In SQL-89 and continuing in SQL-92, a transaction commit automatically causes all open cursors to close, so cursor position is lost. The behavior of these cursor standards and many product implementations is clearly a significant liability in common interactive applications that wish to scroll through a result set making occasional updates. The problem is solved in MSC-API, where cursors remain open and positioned after commit.

- To minimize communication overhead and latency in client-server retrievals, the MSC-API takes advantage of a sensible opportunity to batch requests by fetching a (parametric) number of rows at once from the server to the client, usually the number needed to fill a screen display. This is known as a ”fat cursor” capability, and seems to be unique.

- It is unclear how classical Isolation Levels should be used with a user scrollable cursor. Even Cursor Stability (SQL-92 ”Read Committed”) seems to cause too much contention delay if locks are held awaiting user response. The solution is to provide an ”optimistic” form of concurrency, where rows on the user screen are not kept locked on the server. Updates to rows on the screen are identified on the server during a short-term transaction, and the updates are performed if the rows involved have not been modified by other users. Otherwise, row changes are reflected back to the interactive user to try again.

To expand a bit on this ”optimistic” form of isolation we explain the OPTCC option, set by the programmer when opening a cursor or else as a default for all cursor opens. When a fat cursor fetch is made under this option, a set of rows in the server will be retrieved to the client with a unique identifier and a ”Timestamp” for each row, if a column in the table has been defined to contain these. (These ”Timestamps” are used in SQL-Server as version numbers; they are really use-driven counters, incremented when row updates occur.) When updates to rows on the client screen are to be executed, a transaction is started and the affected rows on the server are identified. If the affected rows have not changed their version numbers then the update will be successful. Otherwise, a transactional abort occurs and the client displays any new values for these rows on the client screen. Note that if no version number exists for the rows retrieved, an analogous approach is taken except that changes to affected rows are detected by testing values of columns retrieved for those rows and seeing if changes have occurred. These values, after all, are the only things the user has seen on which the decision to update was based. In the case where only a few values are retrieved to the screen, the value based approach seems to have an advantage over the Timestamp approach, because the method allows concurrent updates to different columns by different clients. Because of this, another option, OPTCCVAL, requires value based row update testing even when Timestamps exist.

The ideas behind OPTCC and OPTCCVAL are not new in theory. The method is mentioned in [GRAYREUT], Section 7.12.3, but seems to be new in this client-server form. It is likely that a number of sophisticated application programmers have used schemes like this for equivalent scenarios, but such schemes are at a disadvantage without system support. One of the major aims of Microsoft database transaction design will be to make hard things easy, for programmers as well as users.

3 Concepts for the Future

As we mentioned at the end of Section 1, there seem to be a number of aspects of database theory that must be rethought and applied in entirely new ways to properly address some of the challenges of future database systems. We are not talking here about major research breakthroughs, but rather reevaluation of a number of fundamental concepts, many of which have been published but not adopted by existing transactional systems. As the Microsoft transactional strategy develops, some of these ideas will become central to design.

3.1 Replication, Weak Consistency, and Allied Concepts

Weak consistency replication is an approach used in distributed document handling as an alternative to Two-Phase commit [KBHOG]. The idea is that text data is replicated (copied) from one processor to several others and then updates are permitted on one version of the text without immediate update of the others. The Rweak consistencyS guarantee is that all changes on one version will eventually propagate to all the versions. When conflicting updates arrive at a node, a resolution scheme must be available to RcombineS or Rchoose betweenS the effects of two updates; the scheme used is dependent on the type of application. A common use for this form of replication is for laptops out of touch with the net: documents for which several people share responsibility can be shared out to a number of authors/reviewers, and changes collected later to be integrated. We see this capability now in products such as Lotus Notes.

The weak consistency conflict resolution step is left rather vague in most products today, and it is an important goal to provide more structure for specific application areas. A classical form arises in CASE applications, where designers check out code modules for update projects, and weeks later check the changed modules back in. Many CASE venders provide a sophisticated versioning method to test for code change overlap, with editors to help users combine the effects of overlapping changes. On reflection, it seems that this is exactly the type of consistency appropriate for this situation. When numerous programmers are working at once on different projects that touch the same code modules, a concurrency scheme that takes UPDATE locks and refuses access by others to modules locked is not helpful. Often, different projects changing a module will not overlap at a statement level where conflict problems actually arise, and we probably would not want to detect and deal with such detailed conflicts at the earliest possible time even if this were supported, but would rather wait until the entire set of updates has been made to get the benefit of performing the resolution all at once.

Weak consistency seems to generalize to a number of other applications as well, where atomic distributed transactions are difficult or costly. Even banks, which seem to require strong consistency in dollar transfers, have handled ATM withdrawals of limited cash amounts without a two-phase commit to the account balance; the money saved on less expensive hardware dominates possible losses from fraud. Many banks also allow withdrawal of funds deposited by check before the check can clear a foreign bank, a much more serious risk. Formal generalizations of this type of approach seem to exist with Epsilon Serializability [PULEFF] and a number of solutions based on Escrow locking [HÄRDER, BARGAR, KRIBER].

The weak consistency approach can also be thought of as providing an important element of fault tolerance, since by avoiding two-phase commit we allow failures to occur without blocking progress in distributed situations. Sometimes this kind of fault-tolerance must be provided by a flexible business rule model and a workflow infrastructure. For example, when the normal course of accepting an order requires access to a server that handles warehousing, we need a rule deciding whether the order should be accepted even when that server is out of contact. If so, then the order will be accepted with a number of special follow-on actions: later access to availability might find that a back-order must be prepared for delayed delivery, notification then might be sent to the customer to advise of the delay, etc., a classical workflow scenario. It is a challenge to provide an easy to use system that will make it easy for an entrepreneurial programmer to determine the rules a small business owner wants to impose and then write the workflow to execute these scenarios.

3.2 Some Future Work: Object Orientation and Transaction Coordination

Microsoft is currently looking into a new approach to data access interoperability based on cursors. The approach would make it possible to interchange data naturally between different data providers: spreadsheets and word processors as well as database tables. The approach is object based, integrated with the OLE model, and places an emphasis on efficient access to remote data. A few basic underpinnings for object extended relational capabilities are also envisioned. In this model rows will be treated as lightweight objects with object pointers that can be RswizzledS. In the further future, complex structured attributes and user defined functions are being considered.

A start is being made to define a Transaction Manager (TM) model that will perform distributed transactional coordination. One of the goals is to support interoperability with legacy transactional systems. Interoperability with existing standards will be achieved by treating a TM as something that can be written as an application on top of an extremely basic set of verbs to be exposed by Microsoft Resource Managers (RMs) and special components that provide Logging services and Durable Store and Forward services. Clearly, these capabilities are intended to support the workflow capabilities we have been discussing, although the first phase of implementation will likely support only a basic subset of these capabilities.

One of the interesting aspects of these underlying exposed capabilities is that it should be possible to write applications that can handle specific forms of data with specialized treatment. These applications would use a normal database resource manager for storage, but would become resource managers in their own right, with complete control over their data. As an example, we could create an RM to provide Escrow locking on records for which it is responsible. The Escrow RM would handle all types of accesses to its specialized data by normal application logic: reads, updates, and logical locking for such accesses, as well as prepares, commits and, in particular, recovery. To perform prepare and commit, the Escrow RM must be able to access the transaction coordination verbs provided by the underlying database RM that serves as a data store. To provide recovery, the underlying database RM must signal the Escrow RM when its own recovery is complete; the Escrow RM can then perform its own recovery based on logs written to the Logging services. With this kind of flexibility, it is hoped that new exotic approaches to locking and concurrency can be easily implemented as if they were applications by sophisticated Independent Software Vendors, without the long delays that occur when new fundamental capabilities must be added to the database system.

References

[BARGAR] D. Barbara and H. Garcia-Molina, ”The Demarcation Protocol: A Technique for Maintaining Arithmetic Constraints in Distributed Database Systems,” Technical Report CS-TR-320-91, Princeton University, Apr. 1991.

[CHRYKRI] Panos K. Chrysanthis and Krithi Ramamritham, ”ACTA: The Saga Continues,” Database Transaction Models for Advanced Applications, Ahmed K. Elmagarmid, Editor, Morgan Kaufmann, 1990.

[GARSAL] H. Garcia-Molina and K. Salem, ”Sagas,” ACM SIGMOD Proceedings, 1987.

[GGKKS] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem, ”Coordinating Activities Through Extended Sagas,” Proceedings IEEE Spring Compcon, 1991.

[GHO] Dieter Gawlick, Mei Hsu, and Ron Obermarck, ”Strategic Issues in Workflow Systems,” Proceedings IEEE Spring CompCon, 1994.

[GRAYNY] Jim Gray and Chris Nyberg, ”Desktop Batch Processing,” Proceedings IEEE Spring CompCon, 1994.

[GRAYREUT] Gray, J., and A. Reuter, ”Transaction Processing: Concepts and Techniques.” Morgan Kaufmann, San Mateo, CA, 1991

[HÄRDER] Theo Härder, ”Handling Hot Spot Data in DB-Sharing Systems,” Inform. Systems, Vol. 13, No. 2, pp. 155-166, 1988.

[KBHOG] Leonard Kawell Jr., Steven Beckhardt, Timothy Halvorsen, Raymond Ozzie, and Irene Greif, ”Replicated Document Management in a Group Communication System,” Proceedings Second Conference on Computer- Supported Cooperative Work.

[KRIBER] Narayanan Krishnakumar and Arthur J. Bernstein, ”High Throughput Escrow Algorithms for Replicated Databases,” Proceedings of the VLDB 1992, pp. 175-186

[PULEFF] Calton Pu and Abraham Leff, ”Replica Control in Distributed Systems: An Asynchronous Approach,” Proceedings of the ACM SIGMOD 1991, pp. 377-386

[VASK] David Vaskevitch, ”Microsoft’s Vision for the Transaction Environment,” OTM Spectrum Reports, v. 8, no. 1, February 1994. Spectrum Reports Ltd., MCI Mail: 313 8708

[VICIK] Rick Vicik, ”Microsoft Scrollable Cursor API,” Microsoft White Paper, August 1993.

[WÄCHTREU] Helmut Wächter and Andreas Reuter, ”The ConTract Model,” Database Transaction Models for Advanced Applications, Ahmed K. Elmagarmid, Editor, Morgan Kaufmann, 1990.

CALL FOR PAPERS

RIDE-DOM'95

Fifth International Workshop on Research Issues on Data Engineering:
DISTRIBUTED OBJECT MANAGEMENT
Taipei, Taiwan, March 6-7, 1995

Sponsored by the IEEE Computer Society (pending

RIDE-DOM'95 is the fifth of a series of annual workshops on Research Issues in Data Engineering (RIDE). RIDE workshops are held in conjunction with the IEEE CS International Conferences on Data Engineering. Past successful RIDE conferences include RIDE-IMS'91 (Kyoto, Japan), RIDE-TQP'92 (Pheonix, USA), RIDE-IMS'93 (Vienna), and RIDE-ADB'94 (Houston, USA). The next RIDE workshop will also focus on distributed object management systems.

The objective of the workshop is to provide a forum for the discussion and disseminatin of original and fundamental advances in all aspects of distributed object management. Original research papers are sought in all areas related this objective. The following is partial list of research areas of interest:

- Distibuted Objectbase Design
- Object Migration
- Managing large distributed object stores
- Language Support for Persistent Objects
- Operating System Support
- Object Views
- Distributed object system architecture
- Transactions in Object-Oriented Systems
- Distributed garbage collection
- Queries and Optimization
- Applications (e.g., GIS, CSCW)
- Supporting Interoperability

The workshop encourages papers from industrial and user communities that will promote debate among researchers and practitioners. The workshop will include panel sessions that will investigate the emerging products, standards and application platforms. The proceedings consisting of the accepted papers will be published by IEEE Computer Society and will be widely available.

INSTRUCTIONS: Authors are invited to submit six copies of manuscripts (up to 25 pages, double-spaced) by July 22, 1994 (hard deadline!), to the RIDE'95 Secretariat:

RIDE'95 Secretariat
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1

CONFERENCE ORGANIZATION:

HONORARY CHAIRMAN:	GENERAL CHAIRMAN:	PROGRAM COMMITTEE CO-CHAIRS:	
Yun Kuo	**Ahmed Elmagarmid**	**M. Tamer Özsu**	**Ming-Chien Shan**
Institute for Information Industry	Purdue University	University of Alberta	HP Laboratories
	ake@cs.purdue.edu	ozsu@cs.ualberta.ca	shan@hplmcs.hpl.hp.com

PROGRAM COMMITTEE:

Gul Agha (USA)	Peter Apers (Netherlands)	Malcolm Atkinson (Scotland)	Francois Bancilhon (France)
Elisa Bertino (Italy)	Jose Blakeley (USA)	Alex Buchmann (Germany)	Michael Franklin (USA)
Olivier Gruber (France)	Mei Hsu (USA)	Yahiko Kambayashi (Japan)	Wolfgang Klas (Germany)
Paul Leach (USA)	Chiang Lee (Taiwan)	Frank Manola (USA)	Eliot Moss (USA)
Jack Orenstein (USA)	Hans Schek (Switzerland)	Marc Shapiro (France)	Henry Tirri (Finland)
Gerhart Weikum (Germany)	Akinori Yonezawa (Japan)	Stan Zdonik (USA)	

Steering Committee:	Ahmed Elmagarmid (Chair) Joseph Urban (Chair) Yahiko Kambayashi Marek Rusinkiewicz
Local arrangements:	C.J. Cherng (Taiwan); cjcherng@iiidns.iii.org.tw
Publicity co-chairs:	Ken Barker (Canada); barker@cs.umanitoba.ca Gary Gong (Taiwan); gary@iiidns.iii.org.tw
Proceedings chair:	Omran Bukhres (USA); bukhres@cs.purdue.edu

IMPORTANT DATES:
Deadline for submission:
July 22, 1994
Notification of acceptance:
October 30, 1994
Final camera-ready due:
December 05, 1994

CALL FOR PAPERS

DATA ENGINEERING

11th International Conference on
Data Engineering

March 6-10, 1995
The Grand Hotel, Taipei, Taiwan
Sponsored by the IEEE Computer Society

IEEE

SCOPE

Data Engineering deals with the modeling and structuring of data in the development and use of information systems, as well as with relevant aspects of computer systems and architecture. The 11th Data Engineering Conference will provide a forum for the sharing of original research results and engineering experiences among researchers and practitioners interested in automated data and knowledge management. The purpose of the conference is to examine problems facing the developers of future information systems, the applicability of existing research solutions and the directions for new research.

TOPICS OF INTEREST

The topics of interest include but are not limited to:

- AI and Knowledge-Based Systems
- Data Consistency, Integrity and Security
- Data Modeling and Database Design
- Data Structures and Access Methods
- Engineering/Scientific Databases & Applications
- Extensible and Active Databases
- Incomplete, Imprecise or Uncertain Information
- Heterogeneous Systems and Interoperability
- Knowledge Discovery
- Mobile and Personal Computing
- Multimedia Databases
- Object-Oriented Databases
- Query Languages and Optimization
- Parallel and Distributed Databases
- Real-Time Databases
- Temporal and Spatial Databases
- Transaction and Workflow Management
- Tuning and Performance Evaluation

PAPER SUBMISSION

Six copies of original papers not exceeding 6000 words (25 double spaced pages) should be submitted by May 20, 1994 to:

America and Europe:

Philip S. Yu, ICDE
IBM T. J. Watson Research Center
30 SawMill River Road
Hawthorne, NY 10532
E-mail: icde95@watson.ibm.com
Tel. (914) 784-7574, FAX: (914) 784-7455

Far East, Australia and other areas:

Arbee L.P. Chen, ICDE
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300
E-mail: icde95@cs.nthu.edu.tw
Tel. (35) 731065, FAX: (35) 723694

Authors should indicate one or two of the conference areas most relevant to the submission.

ORGANIZING COMMITTEE

General Chairs: R.C.T. Lee, National Tsing Hua U. and C. V. Ramamoorthy, U. C. Berkeley
Program Chairs: Philip S. Yu, IBM Watson Research and Arbee L.P. Chen, National Tsing Hua U.
Steering Committee Chair: Benjamin Wah, U. of Illinois at Urbana-Champaign
Industrial Program: Ahmed Elmagarmid, Purdue U.
Panel Program: Maria Zemankova, MITRE
Tutorial Program: Y. Lien, ITRI/CCL

PROGRAM VICE CHAIRS

Object-Oriented Databases
Kyu-Young Whang, KAIST

Engineering & Scientific Databases
Margaret Eich, Southern Methodist Univ.

Database Design and Modeling
Gunter Schlageter, Univ. of Hagen

Heterogeneous Systems & Interoperability
Calton Pu, OGI

Performance Evaluation
Gerhard Weikum, ETH Zurich

High-Performance & Parallel Systems
Masaru Kitsuregawa, Univ. of Tokyo

Transaction Management & Real-Time Databases
Krithi Ramamritham, Univ. of Massachusetts

AI, Knowledge-based Systems & Deductive Databases
JiaWei Han, Simon Fraser Univ.

Access Methods & Query Optimization
David Lomet, DEC Cambridge Research Lab

Extensible, Temporal, Spatial & Active Databases
H. V. Jagadish, AT&T Bell Labs

Data Consistency, Integrity and Security
Chin-Chen Chang, National Chung-Cheng Univ.

There will be a series of sessions focused on issues relevant to practitioners of DBMS technology. Each *panel, industrial* and *tutorial proposal* should include a one-page description of the subject matter and the name of the organizer, and for panels, a list of proposed panelists. Submissions should be addressed to the Program Chairs, who will forward them to the appropriate conference organizer.

PUBLICATIONS & AWARDS

All accepted papers will appear in the Proceedings published by IEEE Computer Society. The authors of selected papers will be invited to submit an extended version for possible publication in the *IEEE Transactions on Knowledge and Data Engineering* and in the *Journal of Distributed and Parallel Databases*. An award will be given to the best paper. A separate award honoring K.S. Fu will be given to the best student paper (authored solely by students).

IMPORTANT DATES

- Paper, Panel, Industrial and Tutorial submissions: May 20, 1994
- Notification of acceptance: Sept. 10, 1994
- Tutorials: March 6-7, 1995
- Conference: March 8-10, 1995

EUROPEAN COORDINATOR

- Elisa Bertino, Univ. of Genova

FAR EAST COORDINATORS

- Makoto Takizawa, Tokyo Denki Univ.
- Mike Papazoglou, Queensland U. of Tech.

EXHIBITS PROGRAM

- Jie-Yong Juang, National Taiwan Univ.

PUBLICATION CHAIR

- Kun-Lung Wu, IBM Watson Research

PUBLICITY CHAIRS

- Abdelsalam Helal, Univ. of Texas at Arlington
- Chiang Lee, National Cheng-Kung Univ.

FINANCIAL CHAIRS

- Steve Y.L. Lin, National Tsing Hua Univ.
- Jeffrey Tsai, Univ. of Illinois at Chicago

REGISTRATION

- Chuan-Yi Tang, National Tsing Hua Univ.

LOCAL ARRANGEMENTS

- Allen Wu, National Tsing Hua Univ.
- Chen-Pang Lin, III

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903