Shel Finkelstein

# Sixth International

## High Performance Transaction Workshop (HPTS)

## Asilomar, California
## 17-20 September 1995

**General & Program Chair:** Don Haderle

**Program Committee:**

Ed Cobb
George Copeland
Sam DeFazio
Jeff Eppinger
Hector Garcia-Molina
Dieter Gawlick
Jim Gray
Pat Helland
Randall MacBlane
Susan Malaika
Andreas Reuter
David Vaskevitch

**Organization:** Nancy Owens
Diana Miller

**Draft:** Limited Distribution

# SIXTH INTERNATIONAL HIGH PERFORMANCE TRANSACTION WORKSHOP (HPTS)

ASILOMAR, CALIFORNIA
17-20th SEPTEMBER 1995

# Contents

**11. "Shared Air" - Exploiting Broadcast in Large-Scale Information Systems**
Michael J. Franklin, University of Maryland ..... 11-1

**12. THESIS: Queues are Databases**
Jim Gray, Microsoft, Corp ..... 12-1

**13. Drawing the Line between Consumers and Suppliers of Transactions in Object Oriented Application Development**
Geoff Hambrick, IBM ..... 13-1

**14. Transaction Processing on the Internet--Revolution or Evolution?**
James R. Hamilton, Susan Malaika, Patricia Selinger, Eugene Shekita, IBM ..... 14-1

**15. Netscape Secure Courier**
Michael Higgins, Netscape Communications Corp ..... 15-1

**16. Enterprise Transaction Processing on Windows NT**
Greg Hope, Paul Oeuvray, Paul Miniato, Prologic Corp ..... 16-1

**17. Charting the Seas of Information Technology**
Jim Johnson, The Standish Group ..... 17-1

**18. DBMS Bookmarks**
Ian Jose, Microsoft ..... 18-1

**19. Position Paper**
Johannes Klein, Tandem Computers, Inc ..... 19-1

**20. Transactional Objects with OpenStep**
Charly Kleissner, NeXT Computer, Inc ..... 20-1

# Contents


**21. Memory-Resident Database Systems: A Look Back to 1985**
Tobin Lehman, IBM ..... 21-1

**22. Workflows Make Objects Really Useful**
Frank Leymann, IBM ..... 22-1

**23. Shared-Everything versus Shared-Nothing: Why Hybrids Will Win in the Marketplace**
Charles Levine, Tandem Computers, Inc..... 23-1

**24. The Case for Log Structuring in Database Systems**
David B. Lomet, Microsoft Research..... 24-1

**25. Enterprise Client/Server Computing: At the Center of Merging Technologies**
M. Randall MacBlane, Juan M. Andrade, Novell, Inc..... 25-1

**26. Large-Scale SMPs: No End to the Parallelism Debates**
John McPherson, IBM ..... 26-1

**27. An Overview of the Exotica Research Project on Workflow Management Systems**
C. Mohan, G. Alonso, R. Gunthor, M. Kamath,
B. Reinwald, IBM..... 27-1

**28. DataLinks - Linkage of Database and FileSystems**
Inderpal Narang, Bob Rees, IBM ..... 28-1

**29. Temporal Data Manager**
Paul Oeuvray, Greg Hope, Paul Miniato, Prologic Corp..... 29-1

**30. DSS Performance is Now More Significant than OLTP Performance**
Pat O'Neil, University of Massachusetts..... 30-1

# Contents


**31. Towards Implementing Extended Transaction Models on Conventional Transaction Processing Monitors**
Calton Pu, Roger Barga, Oregon Graduate Institute..... 31-1

**32. 13 Million TP HA!**
T. K. Rengarajan, Rabah Mediouni, Sybase, Inc..... 32-1

**33. Indexing for Aggregation**
Andreas Reuter, Betty Salzberg, IPVR, University of Stuttgart..... 33-1

**34. Decision Support Services for Decision Support Systems**
Christiane Reuter, Tandem Computers, Inc..... 34-1

**35. Performance Prediction and Optimization in Workflow-based Applications**
Dieter H. Roller, IBM ..... 35-1

**36. On-Line Reorganization: A Position Paper**
Betty Salzberg, Chendog Zou, Rivka Ladin, Northeastern Univ..... 36-1

**37. APRICOTS - a workflow programming environment**
Friedemann Schwenkreis, IPVR, University of Stuttgart..... 37-1

**38. Transaction Processing for the Masses**
Eugene J. Shekita, IBM ..... 38-1

**39. The Market Perspective - Ease-of-use and Heterogeneity**
Alfred Z. Spector, Transarc Corp..... 39-1

# Contents

40. Why PC Databases are Important for High Performance Transaction Systems
Peter Spiro, Microsoft ..... 40-1

41. Coping with Lock Contention in HPTS
Alexander Thomasian, IBM ..... 41-1

42. Using Unix Workstations for a Low Latency, High Availability DBMS
Oystein Torbjornsen, Svein-Olaf Hvasshovd, Telenor Research..... 42-1

43. Flexible Business Processing with SAP Business Workflow 3.0
Helmut Wachter, SAP AG ..... 43-1

44. Workflow Monitoring: Queries on Logs or Temporal Databases?
Gerhard Weikum, University of Saarland..... 44-1

45. HPTS Agenda..... 45-1

46. HPTS Invitees..... 46-1

# The Rise and Fall of TP LiteLight
# The Fall and Rise of TP Heavy

Mohsen Al-Ghosein
(mohsena@microsoft.com)

## TP-lite won the battle

We thought the debate was settled. *TP-lite won!* Transaction processing was declared a database function, and the database was declared to be the new operating system. Customers chose hardware vendors based on how well the RDMBs worked on their platform. Oracle grew to a multi-billion dollar software giant and Microsoft started heavy investments in the database area.

Well, all of that is about to change. The attributes that made TP-lite so profoundly dominant, are about to become the same attributes that cause it to take a second seat to a rebirth of TP heavy, disguised in object systems.

## The reign of TP-lite

TP-lite was established around the idea that the database with its built in stored procedures, triggers, defaults, and rules, can simply be extended to efficiently manage and execute a large number of transactions supporting large numbers of online users.

The level of integration between the built-in TP monitor and the storage engine, allowed for many optimizations that would have otherwise been difficult (e.g. sharing of logs).

Indeed, TP-lite became the dominant paradigm for transaction processing in the late eighties and early nineties. The success of TP-lite was attributed to the following:

1. Simple programming of transactional applications based on procedural extensions to SQL.
2. Advanced capabilities in the database front-end to manage a large number of connected users. Thread, process, and network session pooling were all techniques implemented by the databases to accomplish this objective.
3. Integration of the database programming language with a subset of the system infrastructure for common tasks. Trigger integration with electronic mail is an example of this.
4. The emergence of a plethora of tools (Visual Basic, PowerBuilder, Oracle Objects,..) that allow ease of development of database oriented applications further accelerated the acceptance of such approach.

## Blind spots

While wildly successful, TP-lite is fundamentally limited by the following:

1. TP-lite offers little choice of a storage engine. To make TP-lite work, one must put all transaction related data in the database. Recent adoption of workflow and the associated requirements for document management, has led most to use the *relational* database as a storage engine for a medium number of unstructured objects, something relational databases were *not* designed to accommodate.
2. TP-lite offers little choice of a programming language. One must use a specific vendor's implementation of the procedural extensions to SQL for transaction logic. While attempts are being made in ANSI to establish certain procedural extensions as industry standards, proprietary languages continue to be the norm. Furthermore, these language extensions while being Turing-Complete, cannot compete with the large number of other well established languages such as BASIC, COBOL, C, C++, or Smalltalk. Indeed, it is often the case that the language used to develop business logic on the client (e.g. Visual Basic) is different (and largely incompatible) with the language used to build server resident business logic (e.g. Transact SQL).
3. The TP-lite model doesn't scale down. There are no provisions to have transaction programs developed to execute on a client machine and then migrated to run on a server with concurrent access.
4. The TP-lite model doesn't scale up. Those who have tried, know that TP-lite is not the right model for supporting thousands of online users.
5. While it wants to be, the database not really an operating system. Databases continue to be generally closed systems not specifically designed for other vendors to plug binary extensions (applications) on top of.
6. Databases and TP-lite are very vendor biased. Vendors view their databases as strategic centers of the known universe. As a result, inter-operability between $n$ vendor databases ends up being an $n^2$ problem. As an example two phase commit coordination between two database requires the complex integration of their client libraries with a transaction manager, and the purchase and installation of the TM on both databases.
7. Databases continue to optimized for output over input. Transactions in the database are not first order constructs. As a result, database client libraries do not take use transactions as boundaries to queue work on the client and reduce the number of network round trips required. The overall throughput of TP-heavy systems (which use transactions as fundamental scope constructs) is generally superior.

## New databases

We are witnessing increasing adoption and investment in a number of database and storage systems. Increased use of object oriented databases, the enormous success of Lotus

Notes, and Microsoft's investment in object file systems, are all indications that relational databases can't do it all.

Having the transaction processing system be so tightly coupled with the database implementation in a proprietary manner makes it very difficult for customers to build transaction processing applications which utilize a number of such databases simultaneously. It also makes it harder for smaller vendors (e.g. Illustra) to be viable competitors to commercial databases (e.g. Oracle) since they are expected to implement all of the TP-lite functionality into their products, a very expensive proposition.

### The rise of distributed object systems

An even more profound trend, is the rise of distributed object systems (DOBS) such as Microsoft's OLE / COM and OMG's CORBA.

DOBS introduce an object paradigm that is language neutral, allows for binary plug and play binding, and very advanced distribution, security, and management facilities.

### The return of TP monitors

While distributed object systems offer developers a choice of storage engines, programming languages, computing platforms, and excellent levels of vendor interoperability[1], they can't compete with TP-lite's ease of programming, tight integration with the database, and overall performance.

Historically, TP monitors have offered application developers tight integration with databases, massive scalability, and robustness while maintaining database and programming language neutrality.

TP monitors dominated and continue to dominate mainframes, but have failed to materialize as serious players in the open server industry (UNIX, Windows NT, OS/2,..etc).

The failure of the open TP monitor industry is probably a result of the following factors:

1. TP monitors are expensive, hard to install, and hard to administer.
2. TP monitors restrict application developers from accessing many of the native operating system services.
3. TP monitors did not facilitate levels of integration between the programming language and the database similar to those of TP light.
4. Perhaps the most important reason for the failure of TP monitors is that they introduced a programming model that was *different* from the native programming model of the operating system and programming languages. Certain TP constructs such as queues do not fit well with most object oriented or procedural languages.

### The new world order

So, who will win the next round for the position as the dominant model of infrastructure for OLTP applications ?

- Will it be TP-lite with a very simple programming model, moderate scalability, and database centricity ?
- Will it be distributed object systems with strong vendor support, language and database neutrality, and modern manageability ?
- Or will it be TP monitors reincarnated to provide superb scalability and robustness ?

The answer is all of the above.

We envision a world where the following trends:

1. Databases become objects, and get distributed and accessed as objects under the control of the object runtime system.
2. Languages mature to provide tighter integration with both databases and objects.
3. Object runtime systems merge with TP monitors and introduce transactional constructs as fundamental object properties, to become the next wave of TP systems
4. Databases shrink again to being storage, query, and access method focused.
5. Stored procedures and triggers become objects that can be written in any language while providing the performance, scalability, and simplicity of data access of TP-heavy and TP-lite.
6. An even larger plethora of development tools show up focused on allowing the development of data models, tie to object models, and dynamic selection of execution context (client or server).

The transition from the current world of three incompatible approaches to the new approach will not be easy. Indeed, use of all three flavors will continue for the next few years.

Ultimately, true downsizing of mainframe applications, the emergence of a new breed of transaction processing applications on the information superhighway, will all dictate and drive the move to the unified approach.

### Conclusion

In this paper, we argued that the wild success of database centered transaction processing will be limited by its proprietary nature. We predicted that transaction processing monitors will be integrated with object runtime systems and extended to better support databases. We predicted that the unification will become the new imperative in transaction processing.

---

[1] Object systems offer excellent interoperability between componentd developed by different vendors on a single object runtime environment. The industry has failed to establish a cross-vendor runtime environment standard.

# The Use of Production Quality Component Technology in Database and Transaction Systems

*Robert Atkinson, Microsoft Corporation, BobAtk@Microsoft.Com, 10 April 1995*

For many years, the software industry has sought to discover the paradigms and mechanisms that would allow it to deploy the artifacts of its craft as reusable building blocks for larger software assemblies, while at the same time allowing for the incremental and independent redeployment and evolution of these building blocks by their suppliers.

There are, of course, many benefits at many levels to be had from such technology, but a significant and important one is that of the creation of marketplaces of third party components that allow a customer to buy from an outside supplier specialized functionality that his main software vendors cannot or will not supply and which he thus would otherwise have to build himself, if indeed such extensibility is possible at all. Conversely, the opportunity for traditional software suppliers is that by architecting the integration of such components into their products, their products become significantly more flexible, extensible, and thus useful and valuable to their customers. However, such in-the-field integration of third party components into large and complex software systems necessitates careful attention in the component infrastructure to the issues of independent evolution, extensibility, and deployment.

Recently, significant progress has been made on these issues. In the past several years, we have gained much experience in designing, building, and deploying successful architectures involving component design and integration problems. While initially applied to the desktop application market, present and upcoming systems use the same underlying component technology to provide extensibility in high performance areas of, for example, the operating system which were traditionally thought of technical necessity to be the purview of closed, fixed designs. The question naturally arises as to the ways in which such technology can be applied to database and transaction processing systems. We believe this to be an area ripe for exploration and innovation.

For example, the problem of the integration of the resources involved in a distributed transaction together with a transaction coordinator is best thought of as a component problem: one wishes to mix and match resources of many different sorts (one or more databases built by different vendors, files in the file system, information in desktop applications, etc.) with one or more coordinator implementations with varying performance and robustness characteristics. Traditional approaches to this problem have on the whole not accounted for the issues of extensibility and evolution. Further, by using a special purpose solution to the problem in terms of their programming model, binding and linkage architecture, and so on, they have made it difficult to integrate distributed transaction management with other component and application integration domains.

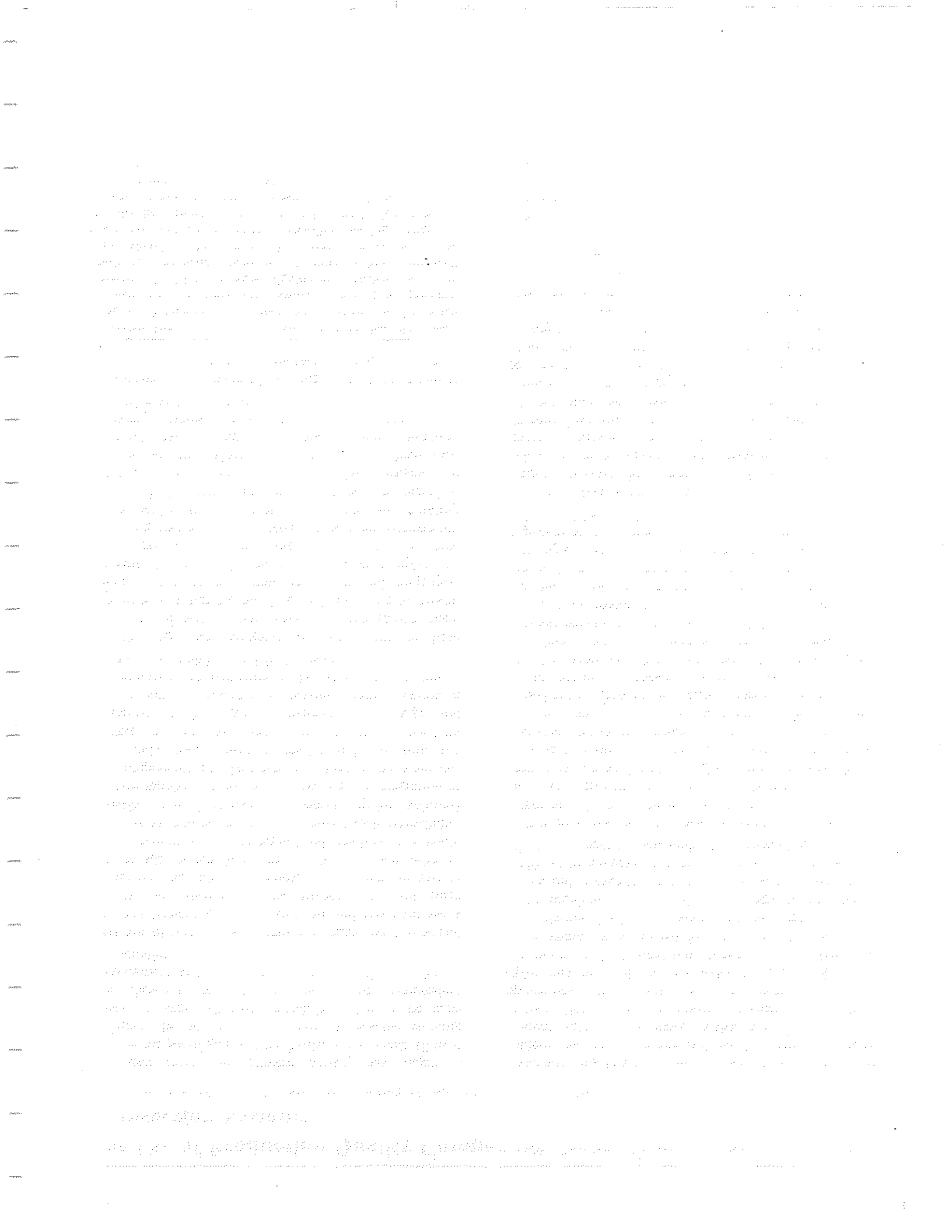
Another good example of a database problem in search of a component solution is the architecture by which application logic (as opposed to system or database logic) is constructed and deployed in high-volume high-availability server applications. By modeling this application logic as installable components inside the server / database system and by applying to the problem a generic and comprehensive production quality component architecture, one can decouple the construction of such applications from particular languages, from particular suites of tools, and even from particular databases. In this decoupling, customers benefit from being able to use their investment in training in the component architecture across a wide range of products, tools vendors benefit from being able to sell more easily into the breadth of the software industry, further benefiting customers, and mainstream vendors benefit from the growth of the overall marketplace that results from the flexibility of their products.

As a final motivating but perhaps less well developed example, consider the design of field calls and escrow locking. Existing schemes are designed to deal with arithmetic operations on fields. However, it is not the arithmetic operations *per se* which make these schemes work, but rather the properties of these operations (such as commutativity) and the properties of the underlying data types (such as total ordering of the real numbers). If the precise important properties of the underlying data and the transformations thereof can be abstracted, then it is plausible to consider that the architecture can be componentized, allowing for more flexible isolation designs that make use of higher level application semantics.

The use of componentization as an integral part of the design of software systems without compromise of performance is today becoming feasible, if not commonplace. In this light we believe it fruitful to reexamine many accepted architectural principles to assess the possible impact of this new capability.

---

The author has been with Microsoft for more than five years, during which time he has focused almost entirely on the design and deployment of component systems. As one of two principal architects of OLE2, he made significant contributions to the underlying component technology infrastructure (the Component Object Model, COM) and to other areas such as the storage architecture and the persistent, extensible naming design. In particular, he is directly responsible for the introduction of the use of transacted storage access in the mainstream desktop application market. At present, he works in Microsoft's transaction processing group, focused on marrying the worlds of traditional transaction processing and component software.



ORACLE® White Paper

# Oracle Mobile Agents

## Technical Product Summary

AUGUST 1995

Part A22547

ORACLE White Paper

# Contents

AUGUST 1995

Automating the Rest of the Corporation ..... 1
What is Oracle Mobile Agents ..... 1
The Mobile Setting ..... 2
System Architecture ..... 3
New Capabilities in Oracle Mobile Agents ..... 7
Summary ..... 9

ORACLE® White Paper

# Oracle Mobile Agents

## Technical Product Summary

AUGUST 1995

### Purpose

This paper introduces the Oracle Mobile Agents Version 1.1 product in terms of the customer and business problems it solves and its system architecture. The product implements a new computing paradigm for mobile workers: client-agent-server computing.

## Automating the Rest of the Corporation

The recent client-server revolution has enabled users to access critical business information from their desktop computers such as those running Microsoft Windows. This gives corporate users interactive, high-speed access to the data that keeps the corporation ticking. Data entry clerks have more efficient and reactive systems, support engineers can find customer solutions more quickly, financial analysts can evaluate data in new and exciting ways, managers have access to better, more current operations reports, etc.

To date, however, client-server has rarely extended beyond the walls of the corporation. Client-server applications are dependent on the reliable, high-speed networks found within the corporation. As a result most information tools for mobile workers are stand-alone, if they exist at all, and rarely connect the mobile worker with the information often most critical to his job.

Oracle Mobile Agents provides the power of the client-server paradigm designed and optimized for the mobile environment.

## What is Oracle Mobile Agents?

Oracle Mobile Agents™ is Oracle's application development environment for extending the client-server architecture into mobile systems. Oracle Mobile Agents introduces *client-agent-server* – an extension of client-server that provides the mobile worker with efficient, transparent access to corporate data sources.

Oracle Mobile Agents combines two key technologies:

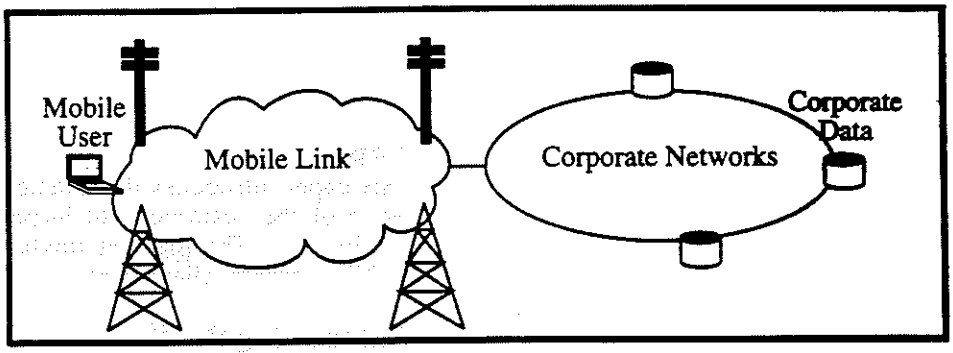
- an application messaging infrastructure for a variety of low-speed networks commonly available to mobile workers, and
- an application development interface allowing many of today's most popular Windows tools to easily access mission-critical corporate data services.

Together, these technologies allow applications that automate mobile workers to be built in a matter of days, not weeks or months, with efficient and flexible support for wireless networks, phone lines, and corporate LANs.

## The Mobile Setting

To understand the significance of the Oracle Mobile Agents product, you first must understand the demands of the mobile environment.

Mobile workers work outside the boundaries of high-speed corporate networks. The communications channels available to these workers today are a wireless network or a phone line while in the field and a LAN while in the office:



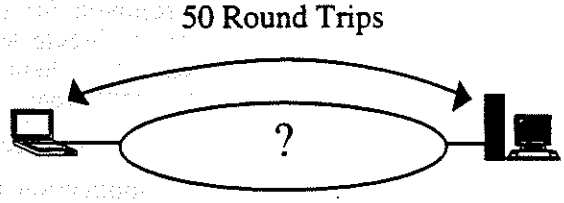

### Network Capacity

Both wireless networks and phone lines are orders of magnitude more constrained than traditional high-speed Local Area Networks (LANs). The following table shows the dramatic discrepancies between bandwidth (kilobytes per second) and network round trip times (latency) of the media:

| Networks | Bandwidth | Latency |
|---|---|---|
| LAN | 5,000-10,000 kbps | .0005 - .001 seconds |
| Modem | 2.4K - 28.8 kbps | .2 - .5 seconds |
| Wireless WAN | 2K - 9 kbps | 4 - 10 seconds |

### The Need for Client-Agent-Server

An example clarifies the impact of the different networks. A client-server order status application might exchange fifty criteria and the retrieval of multiple rows of data. The figures show how that application performs over a LAN, a phone line, and a wireless network.




| Network | Assumed Latency | Minimum Response Time |
|---|---|---|
| LAN | .002 seconds | .1 seconds |
| Modem | .3 seconds | 15 seconds |
| Wireless WAN | 4 seconds | 200 seconds (3 mins 20 secs) |

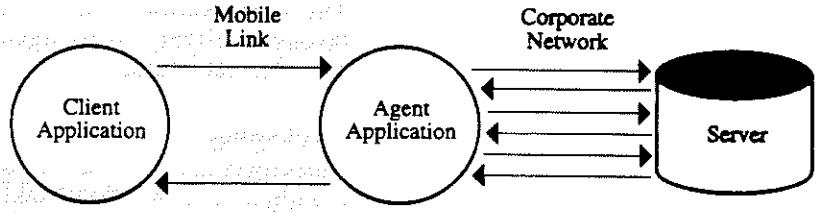
Compounding the difficulty, mobile networks disconnect or fade in and out of coverage regularly, neither of which is tolerated by the connection-oriented nature of client-server applications.

The simple truth is that today's client-server applications do not work well when applied to mobile communications. These demanding network characteristics were the inspiration for the Oracle Mobile Agents product.

## System Architecture

### Client-Agent-Server

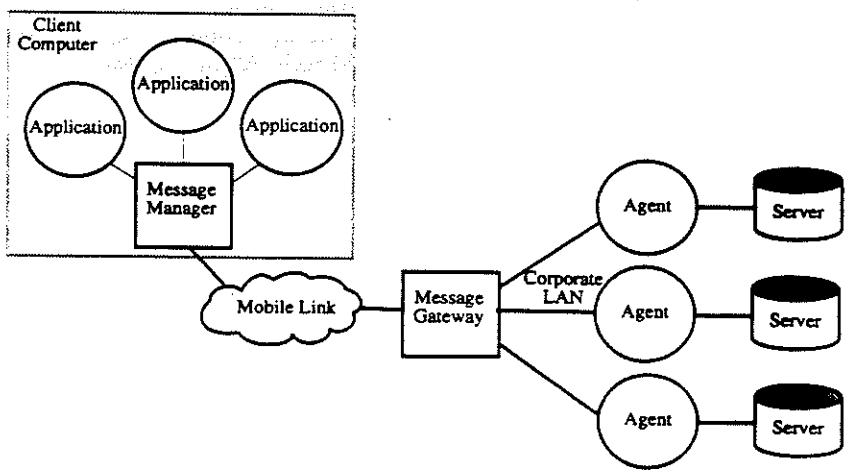
In a client-agent-server architecture, the client and agent combine to perform the work of today's "client" applications. Applications focus on user-interface and navigation, and agents implement data transactions and business logic. Clients communicate with agents via Oracle Mobile Agents *messages* causing the agent to execute work on behalf of the client. Similarly, results of the agent's work are bundled into a single message which is sent back to the client application where they can be displayed.




With client-agent-server, overall transaction performance is maximized by minimizing the use of the mobile client's communications link and taking advantage of the high-speed link between the agent and the server. The result is the fastest possible performance to the mobile client regardless of the complexity of the task.

### Oracle Mobile Agents System Components

Oracle Mobile Agents takes the client-agent-server architecture and delivers it as a system-level product. Unlike many products that exist on either the client or the server, the Oracle Mobile Agents product is a combination of components installed and used throughout the network. In the following component diagram all of the components shown are part of an Oracle Mobile Agents solution and all except the server are part of the product itself.



Each component plays a role in the system:

- Applications – the customer-written components on the client computer that allow the user to initiate Oracle Mobile Agents requests
- the Message Manager – the Oracle software on the client computer that manages communications for all applications
- the Message Gateway – the Oracle system administration software managing all mobile client computers and network services (agents)
- Agents – a combination of Oracle software and a customer-written application on the corporate network that responds to client application requests
- Servers – the existing corporate data sources, either Oracle-based or other data sources.

### Messages

The basic means of communication between the system components is the exchange of messages. Messages carry *structured* data between clients and agents and the reverse by way of the message gateway.

The next sections will follow a message from creation on the client through delivery to its agent to provide an understanding of how the components interact.

### Applications

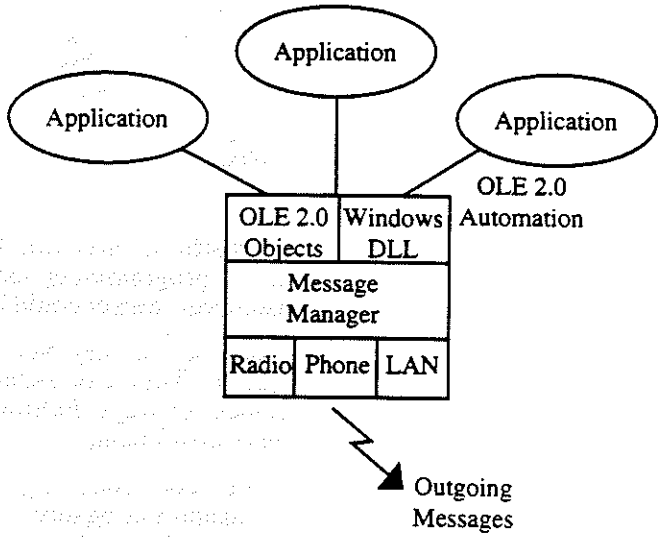
Customers interact with Oracle Mobile Agents using either a simple OLE 2.0 object or a Windows DLL. Any Windows tool that can act as an OLE 2.0 client or can call a Windows DLL can be used to write Oracle Mobile Agents applications. This includes tools such as:

- Oracle Developer 2000
- Oracle Power Objects
- Microsoft Visual Basic
- Powersoft PowerBuilder
- Microsoft Excel
- Gupta SQL Windows
- Microsoft's C/Visual C++
- Borland C/C++

Requests are initiated on the client by creating a message, identifying the message contents, and issuing a send command to pass it into the Oracle Mobile Agents system.

### Message Manager

The message manager is Oracle software that receives messages from all client applications and delivers them over the currently available communications link. It resides with the client applications on the laptop or mobile device. The message manager operates identically over wireless networks, phone lines, or local area networks.




### Message Gateway

Client messages are sent from the message manager to the message gateway to be forwarded to the appropriate agent. The message gateway has four primary roles:

- forwards messages from mobile users to their agents and the reverse
- queues messages intended for clients or agents that are out of coverage, turned off, or otherwise unavailable
- contains all system-level component configuration – all components are self-configuring minimizing the risk of installation or user errors.
- applies all system security through user password encryption and message authentication

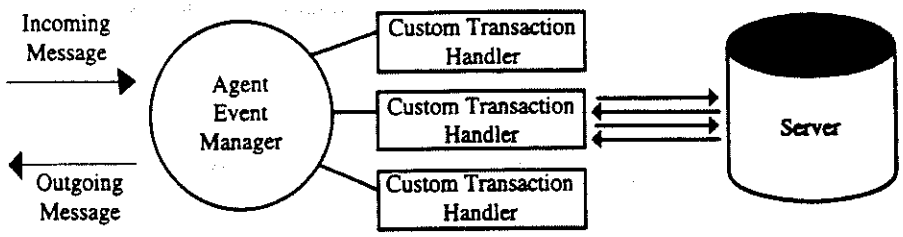
The message gateway and the message manager work together to guarantee that the data sent over the radio or phone line is sent successfully, securely, and with the integrity of the contents intact.

### Agent Applications

Messages are forwarded from the message gateway to application-specific agents. Agent programs consist of two components:

- the agent event manager – the Oracle software on a server, responsible for communicating with the message gateway to send and receive messages and determine what action a message should initiate and

- customer-written *handlers*, one for each transaction the client applications can initiate. A handler is the specific code required to interpret the specific message, perform some action based on the contents, and return a response.




Agents can be written using native application environments on each of the platforms they run. On UNIX servers, this will most commonly be the C programming language, while for Windows, any of the tools mentioned earlier could be used to write an agent.

Agents are the key data access components in the Oracle Mobile Agents system. They can access many different types of data in an infinite number of ways. Following are a few examples of agents and how they serve their clients:

- An order-status agent performs standard query, insert, and update capabilities against a database to manage sales orders. This agent also regularly checks all open orders against inventory levels to alert the field representative and inventory manager of increased demand or shortages. Even if the user has not accessed the order system in days, they can be notified of data changes impacting their orders.
- A support call tracking agent manages call entry and update for field support engineers. Additionally, the agent identifies all priority 1 customer problems as they occur and automatically forwards the details along an escalation path. Further, the agent can automatically send the customer contact internet e-mail updates with the call's status and the individuals working on the problem's resolution as data changes.
- A stock portfolio manager allows users to access real time stock quotes as well as the current value of their personal portfolios from their mobile computer. Additionally, for any stock, buy and sell limits can be placed to either alert the user to points of interest, or to automatically execute trades via on-line trading gateways.

### Servers/Data Sources

Servers are the sources of data requested by the user. While agents will often execute against database servers, such as Oracle7, they can also use files or file servers, or on-line services such as electronic mail or on-line information services as their data sources. The system is as open on the server-side as it is on the client-side.

Agents access servers using the best available native interface to that server. For example, Oracle7 is accessed using the Oracle PRO*C or OCI interface on UNIX, while on Windows these same interfaces as well as Oracle Glue or ODBC are available.

## Capabilities in Oracle Mobile Agents

Oracle Mobile Agents provides many product characteristics that are fundamentally different than the features found in today's client-server systems. The following sections summarize the highlights.

### Asynchronous Operations

Oracle Mobile Agents is an event-driven, store and forward system. This allows all applications to be written asynchronously. This capability unbinds the tight request-response model used in client-server allowing applications to send and receive information as the user needs it. Users can dispatch a number of jobs (i.e., message transactions) in the network and receive responses as they are completed. The client interface never has to wait while a transaction is processed by an agent.

### Changing Communications Links

With Oracle Mobile Agents, the user can choose the most appropriate communications available between the client and message gateway. A typical mobile user is not a "single transmission media" user, but will require a combination of technologies to ensure connectivity in any situation. For example, a sales representative may regularly use:

- packet radio technology – in the car, at the customers site, at the airport, etc.
- phone line connections – when radio is unavailable or out-of-coverage, when speed is important, when a phone line is convenient (hotel room, home), etc.
- LAN connections – in the office

As the user's needs change, they can change communications in Oracle Mobile Agents with the click of a button.

### Suspended/Disconnected Operations

Oracle Mobile Agents allows users to continue to work through marginal communications or even no communications at all. While disconnected, all user initiated messages are queued to be sent later. When communications are reinstated, any queued messages on the client and gateway are automatically sent. All messages are delivered as soon as possible.

This is particularly important when using packet radio networks or phone lines. If a user is temporarily out of range of the radio , and then comes back in, the messages automatically begin to flow. Similarly, if the phone line drops during a modem or cellular modem connection, the application can reconnect and pick up where it left off; all work is preserved.

### Unsolicited Data Alerts

Traditional client-server applications are dependent on the client connecting to a server then proactively requesting information. Data alerts allow the user to find out about changes as they occur, without having to actively search for them.

Further, when these messages arrive on the client, the message manager can automatically launch their applications and display the message.

With data alerts:

- A field service engineer can receive a data dispatch for a service outage requiring immediate attention.
- A sales rep can be notified of a customer escalation for a key account.
- A doctor can be forwarded the medical history of a patient coming into the emergency room.

Data alerts make the data and the user partners in navigating and managing their information.

**Auto-Configuration**

One of the most error-prone mechanisms in distributed systems is consistency across end-user configuration information. With Oracle Mobile Agents, all agents and clients register with the message gateway where all configuration information for the system is stored.

When an agent, user, or application requires system details, they can be looked up in the message gateway. This eliminates most of the user-errors many products encounter when configuration data is entered in many places. It additionally shields the mobile worker from any of the details of the system.

To the end-user, installation and configuration can be as simple as clicking on an installer.

**Open Agents**

The Mobile Agents system was designed from the beginning to be open and flexible on the client and agent interfaces. Application developers are free to write agents that can access many different sources of data, including:

- Oracle databases
- other sources of data
- electronic mail interfaces
- fax services
- other UNIX services such as file, print, or RPC services
- Windows services such as custom applications or workgroup servers

This flexibility makes Oracle Mobile Agents an attractive platform for both customers and third party vendors to develop integrated mobile solutions.

**Open Interfaces**

Oracle Mobile Agents promotes an open architecture through support of industry standards wherever possible.

On the mobile device, we use the most appropriate native operating system mechanisms. For example, the first platform, Microsoft Windows uses OLE 2.0 or Windows DLLs as its interfaces.

Each of the communications drivers supported are supported through their native interface:

- LAN support is through the international standard TCP/IP using the sockets interface.
- Phone line modems are supported through the open asynchronous standard Point-to-Point Protocol (PPP).
- Packet radio networks are supported through native published interfaces. These include the international standard Mobitex interface to the RAM Mobile Data network, and the Winsock interface to a Cellular Digital Packet Data (CDPD) network.

Agents can also be written using a variety of interfaces including:

- C/C++
- Oracle OCI and PRO*C
- Microsoft Visual Basic
- Powersoft PowerBuilder

As additional platforms and protocols are supported, Oracle Mobile Agents will continue to take advantage of native, open interfaces to maximize system flexibility.

### System Management

Oracle Mobile Agents can be easily integrated into existing management infrastructures. An SNMP agent can be used to manage and monitor the Message Gateway and Agent Event Manager components of a system. Dynamically configurable logging provides an audit of system activities.

Additionally, a remote administration tool for user, service, and message administration can be used from a LAN, dialup, or wireless connection.

### Structured Object Transfer

Most other messaging systems support the transfer of binary messages between points in a system. In a heterogeneous environment, this leaves the application programmer with the task of converting data structures and objects between different operating systems or hardware architectures.

Oracle Mobile Agents uses the native type systems of its clients to pass structured data directly from the objects and structures of the native programming environment. For example, in Microsoft Windows, a Visual Basic application can put and take data directly from screen fields into Oracle Mobile Agents messages. Underneath that programming model, Oracle Mobile Agents uses OLE 2.0 to determine the datatype of the field and convert it appropriately for use around the network.

The bottom line to the application developer is portable and worry-free use of structured data and objects.

## What's new in Version 1.1?

Oracle Mobile Agents Version 1.1 contains a number of features and enhancements derived directly from experience in the mobile setting.

### Usability improvements

The user panel has been redesigned to include more information to the power user without increasing the complexity to the non-technical user. Auto-configuration now senses mobile device addresses automatically, and can switch between new ones. The configuration tool has been redesigned to present all important information at a glance.

### Compression

Compression of data during transmission can increase system throughput, particularly over wireless links.

### System Management

An SNMP agent allows Mobile Agents to be managed by existing, standard management applications. Additional logging and statistics provide audit and trend capabilities.

### Integrated, traveling dialup support

Use of PPP vendors stacks is seamlessly integrated into the product, and provides dialup interaction and support to easily change dialing location while traveling.

### Multiple Network support

Mobile devices can now communicate simultaneously with multiple gateways, on completely separate systems. Two example uses of this feature include:

1. Information services may provide Oracle Mobile Agents applications that co-exist with corporate applications. For instance, an online travel service application can simultaneously receive updates and information from its gateway, while corporate applications receive data from HQ. Most other products and systems will only allow connection to one system at a time.
2. Within the same corporation, developers and testers can both use a test and development system, as well as a production system all at the same time.

### Developers Kit

The developers kit has been enhanced to allow developers instant productivity:

- A prototype mode in the Software Developers Kit (SDK) allows windows-windows client and agent development for prototype work. This saves configuration and setup time of the gateway.
- A debugging tool allows developers to see structured messages.
- An object exposing programmatic control of the panel is available.
- With the Windows NT version, one can develop agents and Windows 3.1 clients from a single machine, even over wireless, by attaching two modems and communicating modem-to-modem. The entire setup for this configuration takes less than one hour.

## Summary

With Oracle Mobile Agents, corporations have the means to automate the employees that work outside of the corporation. Its open mechanism provides transparent integration between various mobile networks, and is optimized for capacity constrained and intermittently available networks such as packet radio technology.

The ramifications of Oracle Mobile Agents as an enabling technology are broad. The product allows corporations and Value-Added-Resellers to focus their development efforts on their applications, not the communications infrastructure below. The Client-Agent-Server architecture preserves corporations' existing investments in their networked data infrastructures, while providing their mobile workers new horizons in data access. Oracle Mobile Agents enables an important next step to bringing information outside the walls of the corporation, creating a new kind of company, the Mobile Enterprise.

ORACLE®

**Oracle Corporation**
**World Headquarters**
**500 Oracle Parkway**
**Redwood Shores, CA 94065**
**415.506.7000**
**800.633.0676 ext.13**
**Fax 415.506.7200**

International Inquiries:
44.932.872.020
Telex 851.927444 (ORACLE G)
Fax 44.932.874.625


**Printed in the U.S.A.**
**Part A22547**

# The triumph of common sense:
## Farewell TP-Lite(weight)?
## Farewell TP-Heavy(weight)?
## Arise TP-Middle(weight)?

**Charles Brett**

**C3B Consulting Ltd.**
**&**
**Spectrum Reports/MIDDLEWARE*SPECTRA***

### Introduction

In the celebrated balloon-dominated debate at HPTS 1993, Jim Gray offered his credentials for supporting and sympathizing with TP-Heavy, elegantly argued the case for TP-Lite - and convincingly won the debate. Of all the arguments he marshaled, the economic-based ones were by far and away the most persuasive - as he summarized in his cumulative application of:

- "Ziph's Law: small is common
- "Schumacher's Law: small is beautiful, use appropriate technology
- "Gresham's law: bad money drives out good".

In his HPTS paper he also asserted that "TP-Heavy will be absorbed by TP-Lite (and that) TP-Lite outsells non-legacy TP-Heavy 100:1 in revenue and 1000:1 in units". Both statements remain demonstrably true.

Yet, in the two years since that HPTS, TP-Lite has increasingly found itself under pressure. Despite great economic success new pressures have appeared which:

- put the squeeze on TP-Lite (as previously understood)
- opened up whole new transaction opportunities
- could lead to a renaissance of TP-Heavy (if its vendors can 'get their acts together')
- seems most likely to see the rise of a hybrid - a so-called 'TP-Middle'.

This position paper suggests that this TP-Middle - exploiting a combination of queuing and workflow/process management - is already evolving and will pose a threat both to conventional TP-Lite as well as to TP-Heavy.

### [illegible]s want common sense

[illegible]94 some 45M+ PCs/workstations were sold. In each of 1992 and 1993, 35M+ PCs and workstations were sold - more than the number of automobiles sold worldwide in each year. In 1995 more home PCs will be sold in the US, we are told, than televisions. We live, therefore, in a de facto distributed processing world (at least at the physical if not yet at the processing level).

What these figures also indicate is the massive unbalancing of the conventional IT industry. Computing has, for ever, departed the exclusive shores of the glass house specialist and become common place. With 400M+ end users of personal computing (compared to less than 10M IT professionals), the traditional IT specialist has become an endangered species - however superior his or her specialization.

At the same time what relatively few have noted is that TP-Lite was, and is, largely the province of traditional IT professionals, albeit those wearing the new clothes called client/server. TP-Lite, combinations of relatively simple databases with clients and/or dumb terminals, are generally placed on operating platforms that demand professional support - UNIX or VMS or ...

In 1995 the pendulum is swinging away from TP-Lite. The reason is that the host platforms for TP-Lite are no longer appropriate for the common place computer user.

The emphasis has changed. Rather than look for 1000s of TpS on large hosts - or even 10s-100s of TpS per TP-Lite system, the fundamental computing issue is how to bring those 110M+ PCs (and their earlier brethren) into the transactional world, how to persuade them to:

- originate and undertake an average (say) 100TpD (Transactions Per Day) with other parties/systems
- thereby provide the source for additional volumes on which traditional transactional skills can depend for providing TP growth.

The reality is that the mass of today's users have neither heard of transactionality, nor care - because nobody has communicated with them as to why it matters. Therefore, if they do not know, it cannot matter.

Exacerbating matters, as every individual knows, is a passive 'acceptance' that computers can do anything (everything). Software remains the triumph of a flexibility that delivers all. Common sense - when combined with ignorance - tells us this truth (just as common sense, in ignorance, tells children that ostriches will fly: if it looks like a bird, ...).

## Would common sense choose TP-Lite or TP-Heavy?

Given this fundamental ignorance about what transaction processing and management offers, is it reasonable to expect that either TP-Heavy or TP-Lite in their current incarnations will become popular with the majority of users. In many discussions with clients and users (whether in the US, Europe or elsewhere) who know little or nothing of the disciplines of TP (never mind Distributed TP), what is abundantly clear is that the requirements for transactionality (at least as understood by IT) are invisible. The ordinary user has little or no appreciation of what is involved or needed and, at best, has minimal interest.

Indeed this extends into large IT shops, if for rather different reasons. For example, one major US insurance company CIO - with over 200 CICS partitions - bemoaned the fact that although he offered high transaction facilities, his IT staff had been so good at their jobs of protecting application developers from the intricacies and considerations of TP that they (the application developers) had lost most of their knowledge about such issues. What exacerbated his concern was that those selfsame application developers were all heading for TP-Lite like lemmings - even though they should have known better.

In this environment, the key question swiftly becomes - would common sense users (given their current state of knowledge) choose either TP-Heavy or TP-Lite? Jim Gray answered about the first of these in 1993 when he wrote "appreciate that TP monitors really are heavy. Besides a heavy list price, they carry a high price in specialized staff." TP-Heavy - however much computer theory may need it - is just incomprehensible to the average man (or woman) in his or her comfortable SoHo or enterprise office surroundings.

Interestingly TP-Lite turns out to be little different. Entrenched in the obscurities of database debates, lost in the technicalities of SMP vs. MPP as well as the rich acronyms of client/server, TP-Lite has turned out

to be as complex as TP-Heavy once it is forced to move out from beneath the protection of the simple single server (with a small number of clients) environment where it originally took root. Distributing processing is not simple. But the average user cannot understand why it should not be simple - after all distributing processing is "only a matter of only software". Distributing TP-Lite, unsurprisingly to IT and especially TP professionals, has proven to be at least as complex as TP-Heavy - and arguably more so.

The net result appears to be that common sense dictates that Distributed TP (whether of the Lite or Heavy variety) is simply not attracting the broad support base from the masses of today's users. TP-Lite and TP-Heavy are proving too clumsy, too technical and too inaccessible (if in different ways) for common acceptance. Their strengths will only be accepted at one or more removes from the user.

Yet those same hundreds of millions of users are demanding, or will demand, the processing of transactions. They will demand that their PCs robustly, securely and reliably allow them to initiate and complete transactions with other systems. The key question is, then, what might catch their interest and at the same time be capable of feeding existing transaction systems?

## Arise TP-Middle?

TP-Middle is an unlovely description for what initially might seem only to describe a compromise, a middle ground. Yet, in the absence of a more elegant term, it must suffice here. Initially we defined TP-Middle as a theoretical approach whose purpose must be to interest (and attract) millions of end users into using TP and yet also be relevant to (and usable by) traditional transaction systems. By definition it must have simplicity of appeal and use - at the distributed PC as well as other systems level - as its underlying foundation.

What, then, might offer such simplicity? Choosing technologies in advance always carries risk. However, from our researches, what is increasingly clear is that the queuing and work flow approaches (most probably in combination) offer real opportunities to:

- provide a simply understood, at the conceptual level, picture of a distributed process world ( which is readily modeled and can, therefore, be readily communicated)
- introduce invisible or transparent transactionality (especially if it is made available via automated queue-queue managers with transaction integrity - providing what we term Implicit Transaction Processing)
- permit independent (and parallel) application development, including retention of specialization at different levels
- separate process management (work flow management) from the various application components on the various different systems in the distributed world
- enable the bulk of processors (the 110M+ PCs/workstations) to be brought into the transactional world - but as peers to existing TP-Lite and TP-Heavy systems, not as 'thin subordinates'
- offer linkages between existing (legacy) investments and new investments
- introduce time and location transparency (at least where needed)
- generate the additional volumes of transactions which can drive traditional TP (100M workstations generating 100TpD each creates a massive flow of transactions to be processed by banks, retailers, insurers, etc., etc.).

The important factor here is not the 'technical' description of desirable attributes above but three other factors:

- queuing (whether in a distributed environment or not) is simple to describe and model in visual terms (try attempting the same with either TP-Lite or TP-Heavy); if the uninitiated think they can comprehend what is being offered, they can be sold to (q.v. Lotus Notes, SAP's R/3, etc.)

- work flow managers, as separate process managers, can assemble and disassemble components in multitudes of work streams in ways that are equally describable and recognizable

- decoupling of parts (followed by putting the parts together) almost always makes problems easier to understand than trying for perfection with one monolithic solution.

Combine this with, first, the acceptability of Implicit Transaction Processing before moving on to Explicit Transaction Processing and the acid tests of mass market acceptability can be met. This is the key which we repeatedly find is poorly understood by traditional TP. In essence, Schumacher's Law is being observed. Small is beautiful; queuing and work flow, as a combination of distinct elements, can provide the environment in which TP can continue to flourish (if communicated and made to work in coherent ways).

In effect it is the TpD volume sourced from the PC which, if established, will drive traditional TP. This volume could, however, displace traditional TP-Lite and TP-Heavy if either (or both) do not adjust.

## Conclusion

As Voltaire wrote "the best is the enemy of good". This applies as much to transaction processing and its future as elsewhere. If TP is not made accessible - as its has not been made by either the TP-Lite or TP-Heavy brigades - then it will not flourish in a market which sees no need for its existence.

Why then might we be welcoming back TP-Heavy? One answer is that so much is invested in TP in traditional systems that it would be an economically undesirable to discard this. Linking these (legacy) applications and environments into the 100 TpD environments will be what could save TP-Heavy - because so much proven transaction processing is already there.

In contrast, TP-Lite lacks the underpinnings for long term exploitation - with the one exception of data. But data can be more readily relocated than can processes. Yet there is hope for TP-Lite - if it reinvents itself and assumes the clothes of TP-Middle, namely that it makes distributed transaction processing accessible. The signs are already there that this will occur (Sybase's purchase of CAI is one example; Oracle In Motion is another).

For traditional TP to retain its raison d'être it must seek introduce common sense, make itself appealing. If it does this we can look forward to saying 'farewell TP-Lite', to welcoming TP-Middle (Implicit TP) and offering the opportunity for returning to the inclusion of TP-Heavy. If, however, this does not occur, expect the non-TP world to reinvent transactionality - based around TpD pressures, not TpS. Just as the automobile undercut the train and the truck undercut the canal, so TpD could undercut TpS as traditionally produced by TP-Heavy and TP-Lite.

Charles Brett
Winchester, August 1995

# The Revolution Came and We Missed It. Now What? (Position Paper)

Kurt Brown

Computer Sciences Department
University of Wisconsin, Madison
http://www.cs.wisc.edu/~brown/brown.html
brown@cs.wisc.edu

## 1 The Revolution

A revolution in distributed data and distributed transaction processing has occurred since the previous HPTS workshop in 1993. It wasn't the victory of TP-lite over TP-heavy. Nor was it the victory of shared-nothing over shared-everything, workflow and object frameworks over 4GLs, message queueing over RPC, Unix over MVS, or even Microsoft over IBM. It was the victory of HTML over SQL. And there wasn't a single paper or discussion about it at HPTS '93.[1]

During the week of HPTS '93, there were 18 items in the "NCSA What's New" page of WWW servers [2]. During the week of this year's workshop, there will be over 450 entries, along with advertisements from corporations like MasterCard International. In September 1993, WWW traffic over the NSF backbone measured 80 GB per month; in September 1995 it is projected to be 500 times that amount; in September 1997, 125,000 times that amount [4]. There's a tidal wave coming.

At the last HPTS workshop, I could surf through the WWW pages of a couple hundred physics, astronomy, and computer science research institutions. At this year's conference, I can make airline, hotel, and rental car reservations, lookup phone numbers, check on my FedEx packages, order pizza, read the NY Times, fetch income tax forms, apply for a job, register for conferences, search classified ads around the country, watch TV, and check the latest earthquake map for the Asilomar area. I can also see the first 500,000 digits of the square root of four [5]. In September 1993, OS/2 came bundled with a relational database. No longer. It now comes with a WWW browser and software to connect to IBM's Internet access service. Windows '95 will follow suit (connecting to the Microsoft Network, of course).

Tens of millions of people have access to this data. They can search it, browse it, and (sometimes) update it. It is text, audio, and pictures (still and moving). The "information society" has finally arrived. But this explosion of data doesn't live in a DBMS, nor is it controlled by a transaction monitor. The revolution happened without the participation of the HPTS community, and for that matter, without the participation of Compuserve, Prodigy, or America Online.

[1] Except, perhaps, for Susan Malaika's paper [1] – ironically from someone with a Big Company, Big Iron, TP-Heavy background.

## 2 Now What?

Is the HPTS community irrelevant to the new world data order? Certainly, database management systems and TP monitors are still very much needed, and will continue to be. If the HPTS community concentrates on issues related to DBMS and TP monitors, it will continue to be important to the DBMS and TP monitor vendors. But by the time a few more HPTS workshops roll around, I predict that there will only be three RDBMS vendors left.[2] It may be more important to pay attention to where Netscape Communications Corporation is headed with its Commerce Server [3] than it is to track the features of future DBMS releases.

Stepping back far enough, it might look as if many of the traditional HPTS topics for structured data and database management systems also apply to WWW data and WWW servers: security, standards, performance, and reliability. But the WWW is a different world, and underneath each of these categories are different problems and solutions. Do DBMS and TP monitor security mechanisms (e.g. SQL GRANT and CONNECT statements) have anything to do with the requirements for secure electronic commerce over the web? Traditional data and transactional interface standards (SQL, DRDA, ODBC, XA) will continue to be important, but the standards for WWW data (HTTP, HTML, CGI, URL, SSL) [7] are likely to become just as important.

A lot of the lessons learned in performance and reliability in the DBMS/TP world can certainly be applied to the WWW world, with one "minor" difference: every person connected to the Internet is a potential user of a WWW server. A common feeling among HPTS researchers over the last few years was that central site TP performance was a "solved problem" (e.g. [6]). Sure, American Airlines will still be looking for the latest and greatest in high TPS, but most of the market can just meet its TPS requirement with simple brute force solutions based on buying forever-cheapening hardware, right? Maybe not.

When Ticketron is configuring its web server to support electronic ticket purchases over the net, should they buy enough hardware to support the peak demand that might occur for a reunion concert of the surviving Beatles? Should any retailer that opens shop on the information superhighway buy enough hardware to support peak gift-buying periods? If a server vendor provides a way to dynamically offload demand during peak periods to other idle servers that the retailer doesn't need to purchase outright, this might be an attractive alternative to buying a lot of hardware. What are the mechanisms needed to accomplish this?

Data distribution becomes much more important (and interesting) with hundreds of millions of users scattered over the globe than it does for a single, unified organization with multiple sites and predictable access patterns. Some data are very perishable (stock data, daily news), some can go stale but remain somewhat interesting (the Sports Illustrated Swim Suit Issue), while some may have a very long shelf life (any historical data). Some data have only local appeal (seats for a symphony concert, orders for pizza delivery, real estate information, local news), some have regional or national interest (orders for clothing, flower delivery, new car prices, national news), while others have world-wide value (airline tickets, stock and currency markets, world health and population data). Perhaps the appropriate model for data distribution lies with existing techniques for distribution of food and manufactured goods.

We have a long way to go to make the distribution of electronic information as sophisticated as the distribution systems for food and manufactured goods. The current state of the art is somewhat abysmally summed up by messages like "server unavailable," "connection refused," and "too many queries, try again later." In the world of electronic commerce, these messages mean that potential customers are turned away and sales are lost. Innovative solutions in high performance transactions systems that can prevent these messages will make someone a lot of money. Will these solutions be embodied in database management systems or TP monitors? Looking at the past two years, it's a safe bet that they may not be.

[2] Identifying these three vendors is left as an exercise for the reader.

## 3 Summary

A tidal wave of data has already started crashing on shore, and it's not clear that very much of that data will be controlled by database and TP management systems as we think of them today. If the HPTS community concentrates its efforts on traditional DB and TP systems, we will certainly become less and less relevant. Instead of waiting for the data model issues to be resolved that might allow us to twist HTML data (kicking and screaming) into DB and TP systems that we understand today, we should widen our net to embrace the coming world of distributed data and transaction processing that does not include traditional DBMS or TP systems. There are exciting new problems in security, standards, reliability, and performance in the coming world of web servers that are going to be solved by *somebody*. The HPTS community has the skills to tackle these problems. Even though we missed the revolution, there's still time to join the party.

## References

[1] Maliaka, S. "Data Liberation," *Proc. 5th Int'l High Performance Transaction Processing Workshop,* Asilomar, CA, Sept 1993.

[2] NCSA, O'Reilly & Associates, Sept 1993 - Apr 1995,
`http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/old-whats-new/`

[3] Netscape Communications Corporation, Apr 1995,
`http://home.mcom.com/comprod/netscape_commerce.html`

[4] Rutkowski, A. M., The Internet Society, Aug 1994,
`ftp://ftp.isoc.org/isoc/charts/90s-www.txt`

[5] Sarstedt, S., Apr 1995,
`http://merlin.faw.uni-ulm.de/~sarstedt/sqrt4.html`

[6] Stonebraker, M. "DBMS Research at a Crossroads: The Vienna Update", *Proc. 19 Int'l VLDB,* Dublin, Ireland, Aug 1993.

[7] World Wide Web Consortium, Apr 1994,
`http://www.w3.org/`

# Position Paper: The Next Event for Client/Server TP Monitors

Mark Carges
Novell, Inc.
190 River Road, A-313
Summit, NJ 07901
fink@summit.novell.com

Most of today's client/server-based TP monitors offer several communication paradigms in their API repertoire for application development. The most common paradigm is some form of simple request/response. This style of communication manifests itself either in the form of a language-based remote procedure call (RPC) tool, or as a set of library-based function calls that invoke subroutines in server programs. In either case, two programs are communicating in real-time with one, the client, making a request of another, the server, with the hope that the server will do some processing on the client's behalf and possibly return some data.

Some monitors also offer a more general type of application-to-application communication in the form of a conversational or pseudo-conversational mechanism where a pair of programs can exchange several messages according to an application-defined protocol. Like request/response, this style is also characterized by two programs communicating in real-time, but is used when programs need to keep or build state information across several exchanges of messages.

Another paradigm found in many of today's monitors is reliable, disk-based queuing. Unlike the request/response or conversational tools, which assume programs are communicating in real-time, reliable queuing is used when programs communicate in a time-independent fashion. That is, a message placed in a disk-based queue might not be processed for quite some time after it is enqueued. The enqueuer and the dequeuer communicate through a disk-based queue but each does so at its own pace and without requiring the other to be present or available. Typically, work-flow applications and batch-style systems exploit reliable queuing. Queuing can also be used for mobile computing and high-availability applications.

As popular and as powerful as these paradigms are, I believe that another paradigm is equally important and useful: transactionally-aware event-based communication. Event-based communication fills a void in the space of building distributed TP applications that neither the real-time tools (ie, request/response or conversational), nor reliable queuing can adequately address.

In many mission-critical distributed business application, events occur that impact the state of the business and these must be handled efficiently and naturally by the application. Today, an application developer is usually faced with handling such either via expensive polling mechanisms, or by trying to build such functionality into the application which is cumbersome and most likely not very efficient or scalable. I believe that client/server-based TP monitors must incorporate some form of transactionally-aware event-based communication in order to more fully meet the needs of today's distributed business applications. What follows is a very brief overview of the semantics of this communication technique.

In the context of a TP monitor, event-based communication presumes the following roles: a subscriber, a poster and a broker. A subscriber is an application program that subscribes to an event or set of events, and declares what action should take place when an event is posted. The broker is a system-level entity (ie, provided by the TP monitor software) whose role is to maintain subscriptions and to cause subscribers' actions to occur when events are posted. The poster is an application program that posts events. Events can be posted with accompanying data such that the broker includes the data as part of performing the required actions.

For example, in an application that monitors cellular telephone calls, one program might scan call detail records looking for fraudulently used phone numbers. When such is detected, that program would post a "Fraudulent_Call" event. The event broker would dispatch the actions of programs that subscribed to that event. One subscription might request that a message be sent to the phone number's home cell, while another subscription might request that the call information be enqueued in the local cell for later processing. Note that the poster has no idea which programs subscribed to its event nor what they will do with its data.

The unique aspect of event-based communcation is that the broker sits between the subscribers and the posters, allowing these two applications to remain anonymous from each other. Thus, event subscribers have no knowledge of event posters, and vice versa. All communication is facilitated by an intemediary broker. It can be the case that subscribers have subscribed to events that may never occur. Likewise, posters may post events for which there are no subscribers. The benefit of this anonimity is that the list of subscribers can grow or shrink completely unknown to the programs responsible for monitoring and posting events (ie, the subscriber does not have to poll to see if an event has occurred; similarly, the poster does not have to maintain a list of subscribers and worry about keeping track of which subscribers are still alive, active, etc).

Another unique aspect of event-based communication is that while the subscriber states what action should take place when an event occurs, the poster is the one who provides the data to go along with the action. The types of actions supported by the event broker might be: notify the subscriber asynchronously of the event, issue a particular RPC, place the posted data in a reliable queue, write a particular message on the adminstrator's console, or run some arbitrary program. With each of these actions, the broker may have to "attach" the poster's data to each subscriber's requested action. Thus, application programs using this mechanism must have a priori agreement about the named events on which they will rendezvous as well as the nature of the data that may be posted along with each event. For example, if a program subscribes to event `X' and specifies that the action to perform is "put the posted data in reliable queue `Y'", then when the subscriber later reads its reliable queue, it must know enough about the data so that it can process it.

Since we are talking about event-based communication in the context of a TP monitor, this paradigm gets very interesting when events are posted on behalf of a program's transaction. As mentioned above, subscriptions can have actions such as "call this RPC" or "put posted data into that reliable queue". Because these actions are using transactional communication mechanisms, the event broker can leverage these to provide transactional event posting.

An application can start a global transaction and post its event along with any accompanying data. As part of fulfilling subscriptions that match the event, the event broker would then pass on the transaction context as part of performing any actions that support transaction propagation (eg, RPCs and reliable queuing). The actions, therefore, have become "infected" with the global transaction and can influence the outcome of the transaction in a consensus-building manner. That is, the poster has no idea how many actions will occur on behalf of its event but it wants to be sure that the effects of those actions become permanent if and only if all of them are performed successfully. If any of them fail at whatever they might be doing, then the poster will roll back its transaction undoing the effects that occurred as a result of the actions. Of course, the broker will have to give an indication to the poster as to whether the actions succeeded as a whole or if any failed so that the poster can decide to commit or roll back its transaction.

Adding transactional event-based communication to the set of communication paradigms offered by today's TP monitors will give application developers a powerful tool for modeling their businesses. Since events are location-transparent application-named entities, event-based communication complements other mechanisms offered in most TP monitors which are also based on high-level naming (ie, real-time request/response and conversations, and reliable queuing). Also, providing brokered events in the framework of a TP system provides for an important new transactional communication mechanism that leverages existing communication techniques but adds a new semantic for transactional consensus-building.

---

Mark Carges manages TUXEDO System Development at Novell where he has been a member of the TUXEDO Project for the last 10 years. He designed and developed early versions of the TUXEDO System and is one of the project's principal architects. He has participated on X/Open's Transaction Processing Committee since its inception and is the principal author of X/Open's XA, TX and XATMI interfaces. Recently, he has been working in the area of transactional event communication, transaction protocol interoperability and reliable queuing.

# Integrating TP Monitors and ORBs

**Edward E. Cobb**
**Senior Technical Staff Member**
**IBM Santa Teresa Laboratory**
**555 Bailey Ave.**
**San Jose, Cal. 95141**
**ed_cobb@vnet.ibm.com**

## 1.0 Introduction

Object Request Brokers (ORB) are touted as the next revolution in distributed computing, promising to simplify application development through the techniques of object technology: encapsulation, inheritance, substitution, and polymorphism while making the deployment of real distributed applications more feasible than earlier technologies like APPC and RPC. The current generation of ORBs rely primarily on static binding between clients and servers suggesting that they may not scale well as more applications and workload are deployed.

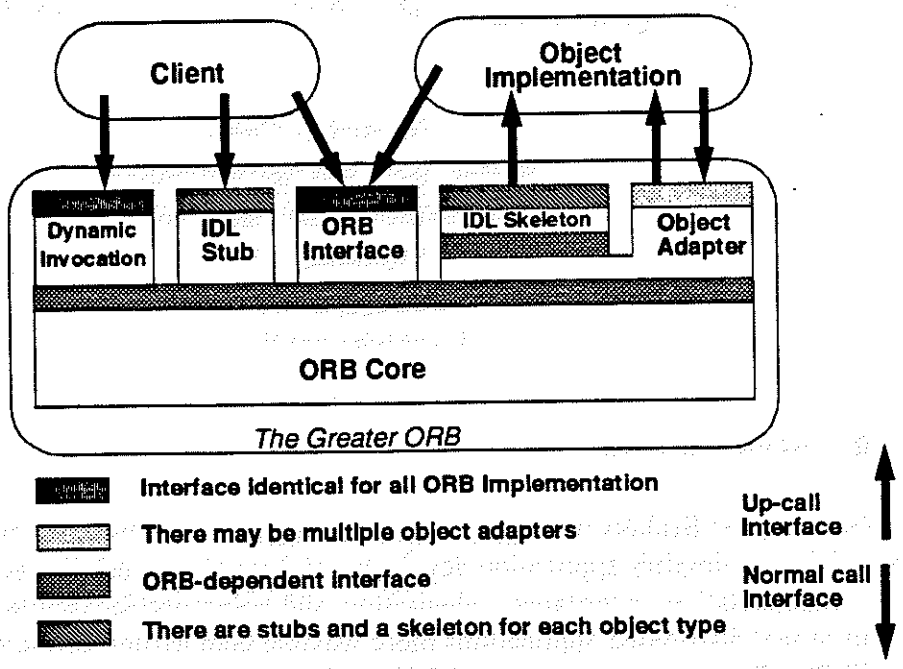
TP Monitors, on the other hand, have demonstrated good performance and scalability by providing dynamic binding between clients and servers with sophisticated algorithms for creation and destruction of server processes. These architectures have been commercially successful in the deployment of OLTP applications, a class of business application characterized by a large number of clients repetitively executing a relatively small number of short duration transactions.

It seems likely as ORBs mature and object technology is extended from the realm of the desktop to commercial transaction processing applications, that the two technologies should come together. The subsequent position paper explores some of the issues in such a convergence and suggests some possible ways it might come about.

## 2.0 What is an ORB?

An Object Request Broker (ORB) can be considered a software bus that provides the mechanisms for objects to communicate with each other, regardless of where in a network they are located. Functions of an ORB include *message dispatching. communications . and object activation*. One example of an ORB is the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA).Figure 1 below shows the major components of CORBA.

**FIGURE 1. Common Object Request Broker Architecture**




CORBA provides both a static and dynamic form of invoking object operations (methods):

- The Dynamic Invocation Interface (DII) interprets dynamically constructed requests.
- Interface Definition Language -- IDL (a syntax similar to C) describes interfaces provided by object implementations. IDL stubs are built by a compiler and bound with the client's executable code.
- IDL skeletons provide the linkages between the ORB and the object's implementation.
- An object adaptor is responsible for implementing specific object activation policies.

## 2.1 Message dispatching

Objects communicate using messages. A client requests service from another object by constructing a message and passing it to the ORB. The format of the message is described in IDL and its definition is maintained in an Interface Repository, representing a durable contract between the object providing the service and all clients that will use the service. The ORB will locate the target object and deliver the client request, in the form of a message, to the object which implements that request. When both are collocated, message dispatching is similar to a dynamic link library (DLL) call:

- parameter values are passed without translation.
- the target object's method executes in the client process and on the client's thread.
- the method executes in the call stack of the client.

When they are in different execution environments:

- parameter values are marshalled and may be translated (e.g. ASCII to EBCDIC).
- communications services are utilized to move the resultant string.
- the method executes on a different thread within a different process, and a new call stack is created

## 2.2 Communications

When the target object is located in a different system than the client, the ORB utilizes the facilities of a communications transport to send the client's request to the target object. Until recently, each implementation of CORBA defined it own protocol to communicate with another instance of itself to transfer and receive the request. CORBA 2.0, adopted in December 1994, defines a TCP/IP-based communication protocol (UNO) and an architecture for utilizing additional protocols (e.g. DCE RPC). This permit different implementations of CORBA to communicate. Later this year, a specification for interoperability with Microsoft's OLE, will also be adopted, extending these capabilities to a very broad spectrum of personal computers, workstations, and mainframes.

When a request arrives at the target system, it is "forwarded" to the correct server process where local message dispatching takes place.

## 2.3 Object activation

To be able to dispatch a message to an object requires an object instance in the target environment. For collocated objects, this is simply memory management. For objects in different execution environments, the target must be located and an environment to activate the target object must be selected or created.

Within CORBA, an *object adapter* is the component which provides for the activation of objects in different execution environments. The Basic Object Adaptor (BOA) interface, "is intended to support a wide variety of common object implementations." CORBA further defines that, "more (but not many) object adaptors will be needed to support different kinds of object granularities, policies and implementation styles."[1]

# 3.0 TP Monitors

TP monitors are used today to support the execution of many commercial applications, especially those using transactions. Most TP monitors are packaged with a transaction manager and systems management facilities. The principal function of a TP Monitor is optimal connection of clients to servers, i.e. *scheduling*.

1. Cf. Chapter 9 of "The Common Object Request Broker: Architecture and Specification" available from the Object Management Group

## 3.1 Scheduling

The TP monitor provides a *light weight scheduler*, optimized for short duration applications typically found in OLTP systems. TP monitors perform a multiplexing function which allow a large number of clients to efficiently access a much smaller number of application servers.[2] The TP Monitor dynamically connects clients to available server processes, ensures that the proper server application code is available to the process selected, and may also dynamically create and destroy the server processes themselves.

TP monitors permit server application logic to be written for a single user instance. The TP monitor supports multiple users by scheduling *multiple instances* of the application -- ideal for languages like COBOL which do not natively support multi-threading. The typical server in a non-TP monitor environment must support multiple users in the same application simultaneously.

# 4.0 Integrating TP Monitors and ORBs

Today's generation of ORBs rely on a server model similar to the non-TP monitor environment.

- A finite number of server processes are started by some external mechanism (e.g. an operator command).
- A "listener" thread within the ORB monitors the communication transport for incoming request and routes them to the correct server process.
- Each server process must be able to handle multiple users so it will be forced to engage in some form of multi-threading within its implementation.

This approach has several disadvantages:

- Application programming is more complex.

  Each object implementation must cater to its simultaneous use by multiple users. This requires multi-threading, concurrency control (to access shared resources), and application level security. In other words the application starts to include many of the functions of today's TP Monitors.[3]

- Scalability is limited

  As new applications are added, new server processes are required. New clients can also require additional servers just to handle the increased workload. Since the client to server bindings are static, one quickly runs out of the ability to add additional servers, either through diminished processing power or memory or both.

2. Cf. chapter 5, Transaction Processing: Concepts and Techniques
3. SAP Release 3 is an example of exactly how these TP Monitor functions have migrated into the application program even without the use of an ORB.

## 4.1 An Alternative Design

In many respects, object invocation is analogous to the transactional RPC function of today's TP monitors. Operations are invoked as the **only way** to access data.

- ORBs invoke methods as the only way to access an object's data.
- TP monitors schedule the transaction program named as the target of the RPC. Data access occurs as a by-product of that program's execution.

Multiple clients may access the same object. This requires a mechanism for mediating access.

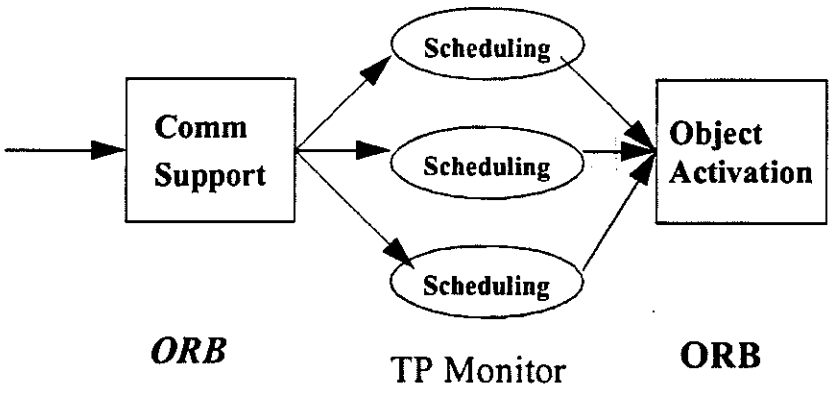
- In CORBA, this is provided by the *Object Transaction Service* (OTS)[4] and the *Object Concurrency Service (OCS).*
- In a TP monitor, this is provided by the transaction manager and a lock manager.

Since both the OTS and the OCS are based on the same core services (i.e. transactions and locking), the only difference is in the form of the interface (i.e. OO vs. procedural).

## 4.2 Combining ORBs with TP Monitors

ORBs and TP monitors can be integrated by separating the scheduling functions of the object adaptor from the activation functions and using the TP monitor to provide the scheduling. This is depicted in Figure 2.

**FIGURE 2. Integrating TP Monitors and ORBs**




There are potentially a very large number of scheduling algorithms. This approach allows them to be incorporated as required to meet the needs of different application environments. Some of the advantages are:

- efficient scheduling of the environment in which objects are to be activated,
  This is essential if objects are to be used to build traditional OLTP applications.

4. Cf. "Objects and Transactions: Together at Last," Object Magazine, January 1995

- the ability to define and use objects independent of the TP Monitor selected,
  An essential element for a component software market in the OLTP environment.
- and the ability to enhance the basic object environment or the TP Monitor without impacting the functions of the other.

## 5.0 Conclusion

TP Monitor and ORB technologies can be combined to build a new distributed computing infrastructure, exploiting the strengths of TP monitors, while providing a robust application development environment based on reusable objects. The marriage of the two can produce an integrated solution that optimally addresses the needs of the commercial transaction processing environment.

## 6.0 Bibliography

1. J. Gray, A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, 1993
2. Object Management Group, *The Common Object Request Broker: Architecture and Specification,* John Wiley, New York, 1992
3. Object Management Group, *Common Object Services Specification (COSS), Volume 2,* John Wiley, New York, 1995
4. Cobb, E., *TP Monitors and ORBs: A Superior Client/Server Alternative, Object Magazine*, February 1995

# System Services for SOM Objects

**George Copeland**
**IBM**
**Austin, Texas**
**copeland@austin.ibm.com**

***Abstract***

*The goal of IBM's System Object Model (SOM) is to provide the underlying technology to support large-scale code reuse. This involves addressing several problems, including binary-code reuse across compilers and languages, local-remote transparency, automatically inserting object services (e.g., persistence, concurrency, recoverability, security) into binary classes such as OpenDoc parts.*

*The result is that class and framework suppliers can write their code without the complexity of supporting object services or distribution in their code, and yet their binary code can then be employed in the broadest possible market.*

*This paper describes work in progress. Some of it has already been shipped in products, some has been implemented but not yet shipped, and some is a description of our direction but has not yet been implemented.*

## 1 The importance of binary code reuse

Source-code reuse has many problems. Some example problems are:

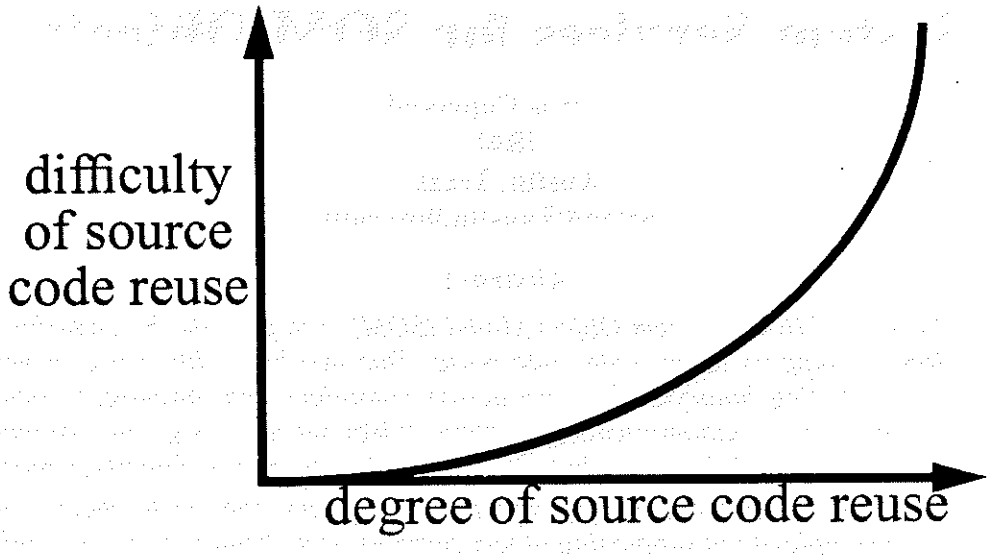
- The source code must be available.
- It is usually poorly tested and not with your compiler.
- It is difficult to tell whether the code has been changed.
- Software support is difficult.
- Compilation is difficult and requires long compile times.
- It is difficult to guard against software piracy.
- Etc.

As Figure 1 illustrates, these problems get worse with the degree of reuse, as more people and organizational boundaries are involved, representing separate economic or legal entities and with a broader range of cultures and development tools.

This means that the importance of binary over source code increases with the degree of reuse. The main purpose of object technology is code reuse. The above problems are just as true for OO as for procedural. We have only a limited form of object encapsulation if we have to give someone source code to either use or subclass the object. For object technology to achieve large-scale success, binary reuse is essential.

The software industry is gearing up toward selling application-specific binary classes (e.g., OpenDoc parts) that can be tailored and assembled together for a specific application. In this emerging component industry, binary code is a must.

**Figure 1: Source code reuse does not scale**




## 2 Compiler and language independence

To support binary reuse, SOM adds the following flexibility to object programming:

- SOM's compiler independence allows a class compiled by compiler A to be used or subclassed using compiler B. Note that this is not possible with the class libraries produced by C++ compilers that are not SOM-enabled because there is no standard library format.
- SOM's language independence allows a class written in language X to be used or subclassed in language Y. Note that this is not possible with OO languages today. Ironically, procedure calls across compilers and languages is supported today.
- SOM's method invocation mechanism allows a base class to change size without recompiling (solves the "fragile base-class problem").

SOM supports the first two features by building method resolution on top of existing technology and conventions for calling binary procedures across language and compiler boundaries. SOM supports the third feature by allowing adequate indirection for method resolution.

SOM classes can be defined in either of two ways:

- The Object Management Group (OMG) CORBA IDL (Interface Definition Language) can be used to define classes. SOM's IDL compiler then produces language-specific files (e.g., header files) to assist the class implementor in implementing the class in either non-OO languages (e.g., C, Cobol) or OO languages (e.g., C++, Smalltalk).
- Direct-to-SOM compilers allow classes to be defined in existing OO languages (e.g., C++, Smalltalk). These compilers can produce IDL for use with other languages as described above.

## 3 Local-remote transparency

SOM support for distribution provides local-remote transparency for binary objects. This allows a class or framework to be developed and compiled in a single address space without support for object distribution in their code, and then the binary code can be deployed with

distributed objects.

Figure 2 illustrates the local case where the object and its client are in the same address space (AS). In the local case, SOM uses a regular method dispatch, resulting in performance similar to C++.

**Figure 2: Local library method invocation**

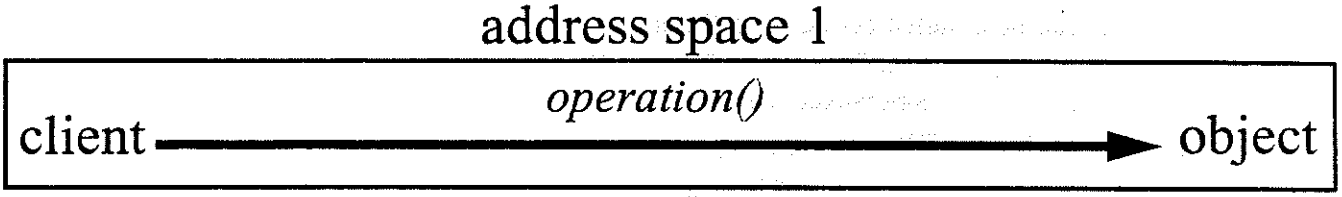
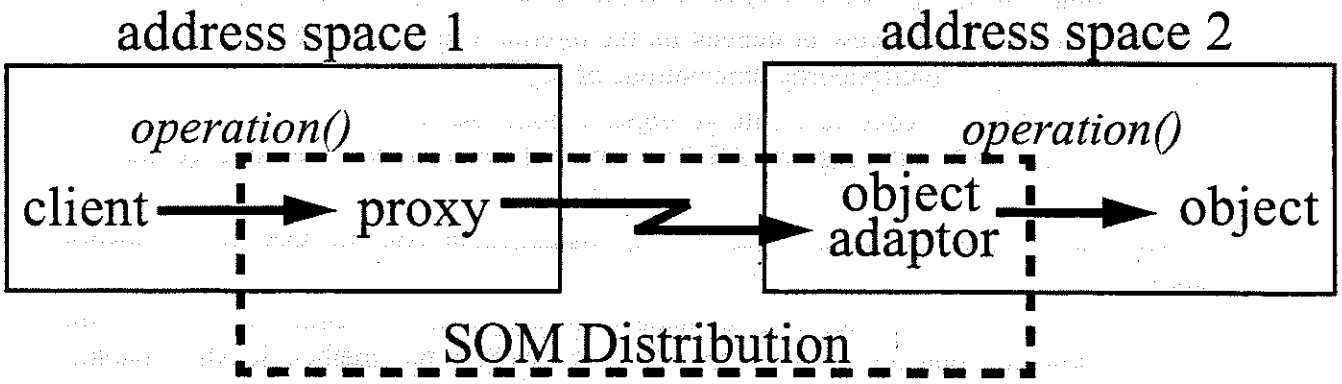



Figure 3 illustrates the remote case where the object and its client are in different ASs. In the remote case, both the object and its client are fooled into thinking that the call is local. When a client first gets a reference to a remote object, SOM produces a local proxy object. This proxy has the same interface as the remote object. Its implementation can either forward the method request to the remote object or cache the remote object. The remote object has an object adaptor (OA) in its AS that either forwards local calls to the target object or manages its caching in clients. The SOM Distribution framework can be tailored by subclassing to achieve specialized proxies and object adaptors, as well as employing various communications protocols.

**Figure 3: Remote method invocation**




## 4 Object Identity

Object identity includes both of the following:

- The ORB generates object references and ensures that operations on different references for the same object are invoked on the same object. For an incoming operation, the ORB either finds the already activated object or activates it.
- The IdentifiableObject operations allow a client to find out if two object references are to the same object.

### 4.1 Object references

An object client can have either a "live" or "stringified" reference to a CORBA object. SOM supports the OMG CORBA interface operations object_to_string and string_to_object to map between live and stringified references.

Methods can be sent directly to a live object reference. SOM's implementation of a live reference is a pointer to the object.

A stringified reference (SR) allows an object to be referenced from outside of its home AS. A SR is used for the following:

- The proxy of Section 3 that supports local-remote transparency.
- It can be emailed across the network.
- It is the persistent form of a reference. If object A references object B, then B's SR is stored in A's persistent state.

A SR includes the following identity information:

<ibm_tag, server_id, key>

- The ibm_tag is provided by OMG to distinguish our SRs from those of other vendors.
- The server_id identifies the object's home AS, including both the machine and the object's AS within that machine.
- The key identifies the object within the object's home AS.
- In general, anything in the SR becomes fragile (i.e., cannot be changed without invalidating outstanding SRs). To allow the object to be moved between servers without invalidating outstanding SRs, an indirection can be used. The server_id/ key pair could be replaced by a uuid which is mapped onto the server_id/key. This single mapping would be updated as part of the Lifecycle move implementation.

Object clients do not know or depend on the internal format of a SR or any part of it. This allows SOM to support multiple implementations of object identity.

A daemon AS is needed to create an object's home AS if it is not already active, and to connect the proxy's AS to the object's AS. The server id gets us to the correct target machine and AS.

Within an AS SOM separates object instance management from the OA. This is illustrated in Figure 4.

The SOM OA waits for messages, demarshals messages into a method call, asks the instance manager to find the object's pointer from the key, invokes the method locally, marshals the returned parameters into a return message, and sends the return message. For a returned object parameter, it asks the object to get its ket, so it can build the SR. The SOM OA can be subclassed to support different message formats.

The SOM instance manager (IM) is responsible for creating keys for objects, and finding or activating objects from a key. There are many techniques for instance management, each far superior to others for handling different cases. SOM provides an extensible mechanism for dynamically adding IMs to an AS, each of which manages its objects in its own way. Each AS has a root IM that the OA calls to translate the key to a pointer. An IM can be bound to the root IM or a nested IM with a component key, forming a tree. An object's key is a path of component keys. Resolving from a key to an object involves a recursive traversal down the tree. This tree is similar to a tree of name directories (the Name Service described in Section 5.5), except that it is used for identity instead of human-friendly names, it is a pure tree instead of a directed graph, and it is typically much shorter.

Some examples of nested IMs are:

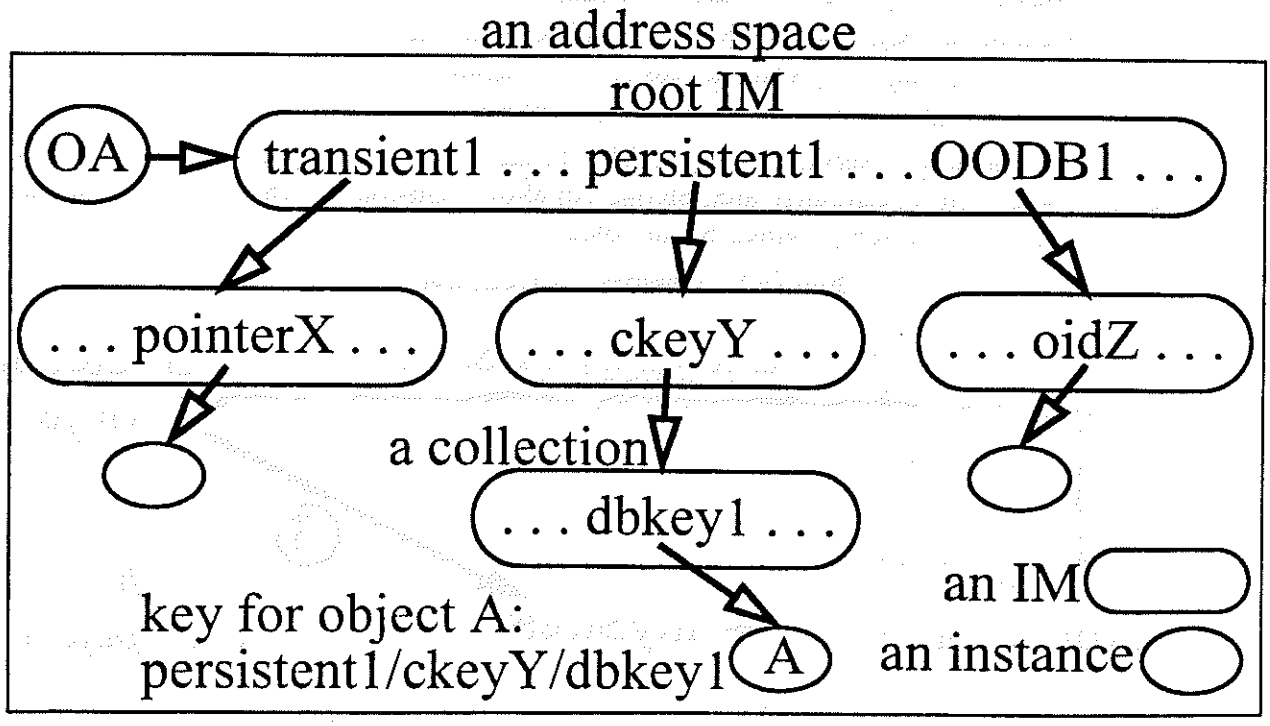
- An IM for transient references might use a pointer as the component key for its objects that are kept in heaps. The pointer might be either directly to the object (faster) or through an indirection (safer).
- An IM for a very large homogeneous collection of persistent objects might use a database record key as the component key for its element objects, manage a cache of recently used element objects, and manage the persistent state of its element objects.
- An IM for an OODB might simply use the OODB's object identifier (OID) as the component key for its objects, and call the OODB's runtime to find/activate its objects.

All objects (except the root IM) are assigned an IM at birth, and support the following operations:

- To find the object's IM: Each IM does this in its own way. One technique might be for the object to have a pointer to its IM. Another technique might be based on the location of the object in the IM's cache.
- To find the object's full key: This method is generic. It uses the above operation to find its IM, and then recursively asks its IM to find its key. The recursion ends at the root IM and the component keys are concatenated on each return down the tree to form the object's key.

An object reference can be either transient (lives as long as the object's AS) or persistent (lives beyond the object's AS). For persistent references, the IM must either include the persistent data identifier (PID) in the component key (cannot change without invalidating outstanding references), or persistently maintain a mapping between the component key and the PID (allows the PID to change).

**Figure 4: Object Adaptor and Instance Mangers**



### 4.2 IdentifiableObject

OMG provides an interface (IdentifiableObject) to compare the identity of two live object's to see if they reference the same object:

- The is_identical operation provides a boolean indicating whether the two input references are to the same object.
- The _get_constant_random_id operation always returns the same number for an object.

If two objects have different numbers, then they cannot be the same. If they have the same number, the is_identical operation must be used. This allows efficient testing of whether an object is in a large set, provided the number generation has reasonable randomness.

This programming model allows the semantics of identity to be determined by the object implementor. SOM provides a default implementation of the IdentifiableObject interface that uses the identity information in the SR, which is implemented within the client's AS (in the proxy if remote) and without activating the object. This default semantics can be overridden to allow more application specific notions of identity (e.g., replication).

## 5 OMG Common Object Services

OMG is defining a set of interfaces for system services for object called Common Object Services. SOM provides an implementation of these services.

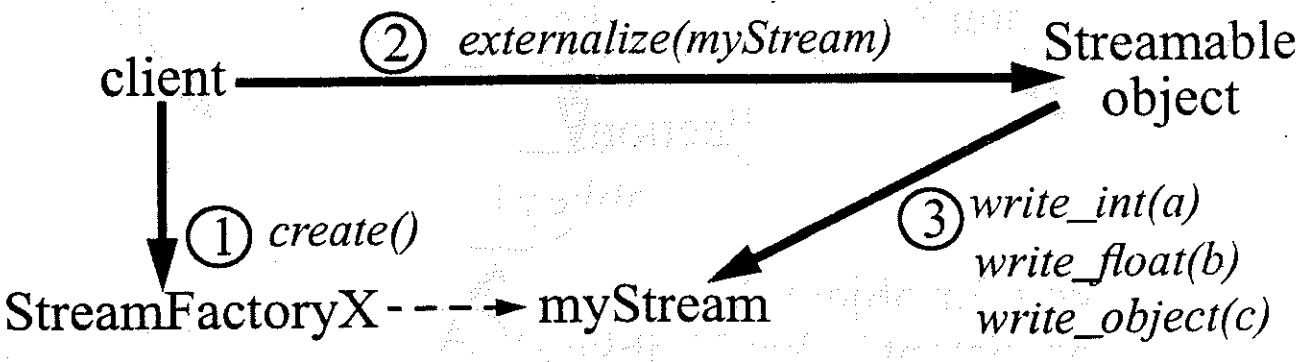
### 5.1 Externalization

The OMG Object Externalization Service is the OMG standard way of getting data into and out of an object for many usages:

- Copying an object across ASs or machines.
- Moving an object across ASs or machines.
- Caching an object in the client AS instead of forwarding operations.
- Passing an object by value across ASs or machines.
- Storing and restoring an object's persistent state.

The Externalization Service is inspired by Taligent Streams. Figure 5 illustrates how the Streamable operations (externalize and internalize) allows someone to ask an object to stream its state either into or out of the provided Stream object.

**Figure 5: Example of Externalization**




The implementation of these Streamable methods is extremely generic, so that they work for all of the above usages.

- It is independent of which of the above usages for which it might be used.
- It only deals with a shallow view of the object. Whenever a referenced object o1 is to be included, a simple write_object(o1) is called.
- The writer includes only non-derivable data.

The object implementor/supplier is responsible for implementing the Streamable methods, as well as describing the IDL for the data in the stream (the class stream schema).

The specialization for particular usages is handled by providing a specialized Stream object.

- It can be tailored to the particular usage and depth of traversal.
- It provides heterogeneous interoperability via a standard byte-stream format (e.g., little/big endian).
- SOM Distribution uses a specialized Stream proxy that does distributed blocking to reduce messages across ASs or machines.

The Providing the Stream implementation is the responsibility of whoever provides the corresponding usage. SOM provides the Stream for many usages, such as for copying, moving, caching, pass-by-value, and persistence where a schema mapping between the object and a database is not needed (e.g., a database BLOB, a Posix file, a Bento file). When thee is a database schema, the object consumer provides the mapping between the object's stream schema and the database schema.

## 5.2 Persistence

As described in Section 4, the ORB is responsible for persistent object references. If the persistent object has persistent state, then some other mechanism than an ORB must be used to maintain that state. The OMG Persistence Service provides a way to do this.

As illustrated in Figure 6, the OMG Persistent Service provides an architecture that supports two-level store (TLS) objects (i.e., an object that lives in regular transient memory and whose persistent state is stored in some kind of datastore mechanism). The architecture has a router mechanism called the PersistentObjectManager (POM) that finds the correct PDS to drive the datastore mechanism indicated by the object's persistent data identifier (PID). The PID describes how to find the object's persistent data. The PDS is responsible for getting the data in/out of the object and out/in to the datastore, and is the only entity that knows the datastore's native interface.

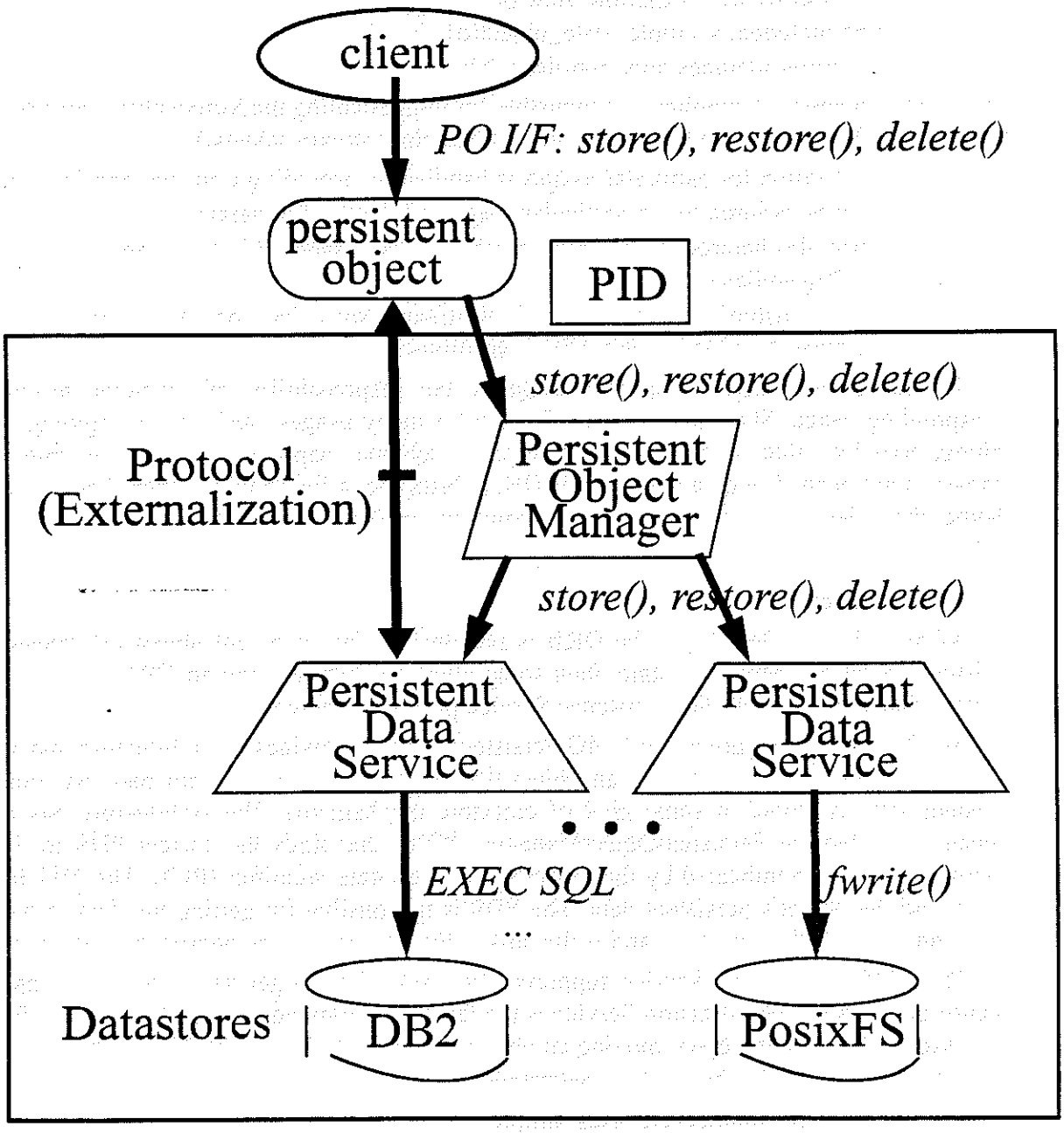
The OMG Persistence Service supports any mechanism to get data into/out of the object. Because the OMG Externalization Service is the OMG recommended way of doing this for many other usages (copying an object, moving an object, caching an object, passing an object by value), we recommend that it also be used for persistence.

This Persistence architecture uses simple operations that deal with object granularity to synchronize the object's state with its datastore:

- store(obj, PID) streams out the object's persistent state and stores it in the datastore indicated by the PID.
- restore(obj, PID) retrieves the object's persistent state from the datastore indicated by the PID and streams it into the object.
- delete(obj, PID) deletes the object's persistent state from the datastore indicated by the PID.

A persistent object can optionally support store, restore and delete directly (called the PO interface) for applications that need explicit external control of persistence.

**Figure 6: Persistence Service saves object's persistent state**




There are two ways to use persistence. To illustrate, let's say you and I want to cooperatively work on the same spreadsheet.

- Object sharing: If we both can use the same object interface and implementation, then we can share a single persistent spreadsheet object. I would create the object using a PID that describes where its persistent data is stored, and give you a reference to the object. We would remember the reference to get the object back or give a human-friendly name to the object in a object directory (Section 5.5).
- Data sharing: If we each want a different object interface or implementation, then we cannot share a single object. Instead, we would have two objects which share

the same data. We would each create our own object using the same PID, and would have to remember both the class and the PID to get the object back.

### 5.3 Transactions

The OMG Object Transaction Service provides the normal two-phase commit coordination among recoverable objects. A user creates a new transaction context relative to the current context. If the current context is not null, this will start a nested transaction. The Transaction Service also defines the interface that a recoverable object must support. These include prepare, commit, abort and forget. The Transaction Service is responsible for the following:

- Passing the transaction context between caller and callee.
- Reliably accepting registration of recoverable resources.
- Coordinating two-phase commit.

SOM Transactions implements the OMG Transaction Service and also supports inclusion in the two-phase commit protocol for procedural resource managers that support the X/Open DTP and other standard protocols.

### 5.4 Concurrency

The OMG Concurrency Service provides a lock manager that can get locks on behalf of either transactions or threads in the object's AS. When getting locks for transactions, it correctly handles lock inheritance for nested transactions. SOM Concurrency implements the OMG Concurrency Service and also supports an interface to the object to allow explicit locking and unlocking for applications where lock contention is a bottleneck. Unlocking early (before the end of the transaction) is probably the most important of these optimizations.

### 5.5 Naming

The OMG Object Name Service allows an object to be bound to a directory (a NamingContext) with a human-friendly name. Because a directory is an object, it can be bound to another directory, forming a directed graph much like a Unix file system. A bound object can be looked up using a path of names. A binding can also contain properties (in a PropertyStore object) which can be used in a search predicate to find qualifying objects. An iterator can then be used to further examine each object/properties in turn. Figure 7 illustrates an example.

### 5.6 Factories

A special case of this Naming Service directory is a FactoryFinder. A class description can be bound to one of these directories along with properties describing the class, including its functional features, the class library, its cost, its instruction set type (e.g., Intel, PowerPC, etc.), etc. A user can search for a Factory (basically a class object) using a predicate on the properties and/or its name. The predicate can include three types of information:

- The functional requirements of the class. Domain-specific property naming conventions facilitate writing these predicates.
- Location requirements (e.g., local AS or a particular server).
- Which of the Object Services are desired to be added to the factory if not in the original class (see section 5).

A chosen Factory is then dynamically created in the specified machine/AS, and one or more instances objects can then be created. Figure 8 illustrates an example class in a FactoryFinder NamingContext.

**Figure 7: Name Service example**

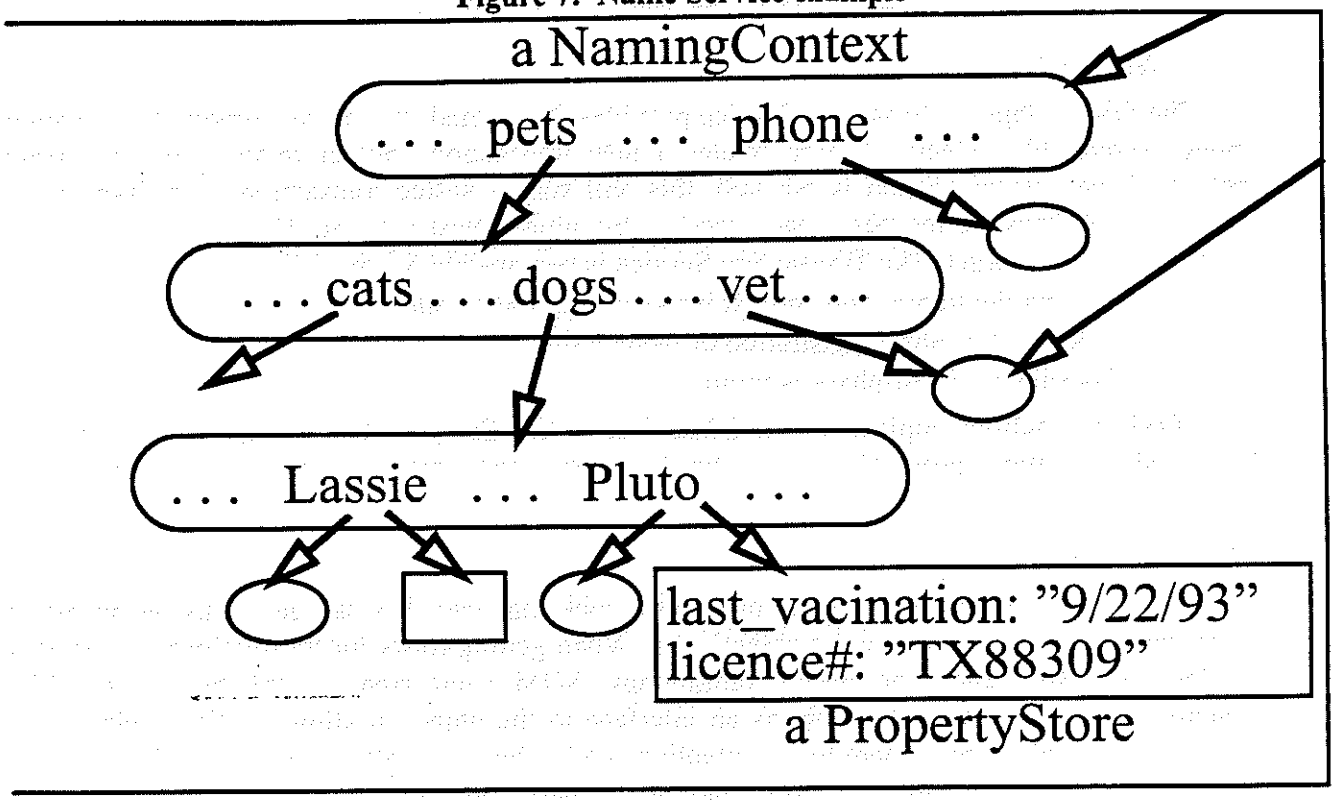
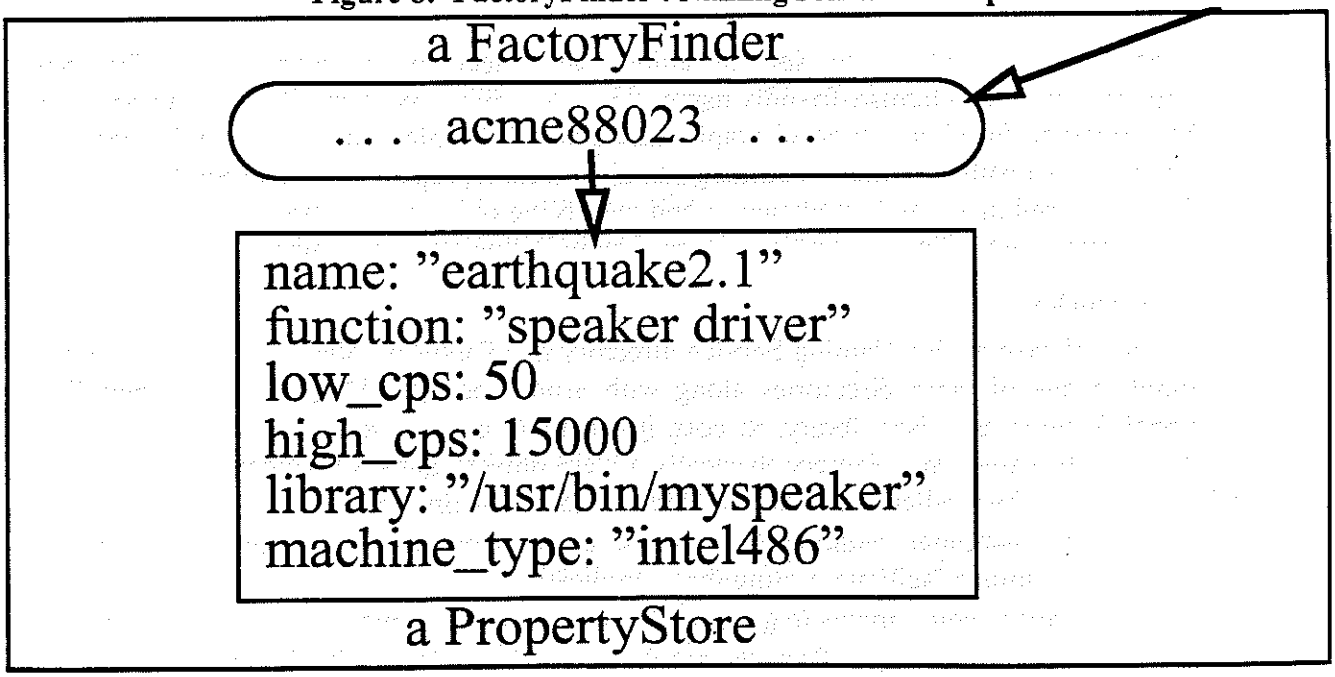



**Figure 8: FactoryFinder : NamingContext Example**




## 5.7 Other Services

Other OMG Object Services include:

- The Security Service provides user authentication and access control at object-method granularity.

- The Event Service provides asynchronous operations.
- The Relationship Service provides named links between objects and a traversal mechanism for the resulting graphs.
- The Property Service provides a way to associate descriptive information about an object.
- The Query Service provides a search mechanism for qualifying objects in a collection or directory.

## 6 Automatic Insertion of Object Services

We expect for the class provider to write their code without any concern for normal system-like services (e.g., persistence, concurrency, recoverability, security, distribution, etc.). When the customer wants an instance of a class, she specifies what combination of services are desired. We create another binary class from the original that adds in these services.

This is done using the following two SOM run-time mechanisms:

- Subclassing: A new class is created by mixing in the original class and a special class that provides the additional function.
- BeforeAfter metaclass: This inserts a method before and a method after the methods of the original class. The ordering of multiple BeforeAfter methods can be controlled. [see Forman, Danforth and Madduri, "Composition of Before/After Metaclasses in SOM, OOPSLA94]

As an example, say the user desires a concurrent, persistent and recoverable version of class A whose data is stored in a database that is also used by procedural applications or other objects. To each of A's original methods, we need to add a Before method that does the following:

- Lock the object on behalf of the current transaction.
- Registered the object with the transaction manager for the current transaction (this requests transaction manager to send a preprepare notification to the object prior to the commit phase).
- On the first method during a transaction, the objects state is out of date, so it must restore the object's state from the datastore (this also allows the datastore to get data locks on behalf of the same transaction and register with the transaction manager via its support of the XA interface).

We also need to add the following methods:

- preprepare: Store the object's persistent state to its datastore (this also allows the datastore to upgrade locks).
- commit and abort: Mark the object's state as out of date, so it will be restored again at the beginning of the next transaction.

If information is available that indicates whether the method is read-only, the above Before methods can be optimized accordingly.

There are several advantages of this automatic "just-in-time" insertion of object services over handcrafting:

- There is much less work to do. Handcrafting these services into objects takes a great deal of work.
- Source code is not required.

- The user's class diagram and library do not increase in complexity and size. With handcrafting, all of the possible combinations of these services could lead to an enormous explosion in the number of classes. Every new service could cause it to double (i.e., x 2^s where s is the number of services options).

The desired combination of object services are specified as part of the search predicate to a FactoryFinder. If the original class does not already provide some of these object services, automatic insertion is used to provide them.

## 7 Datastore independence

SOM Persistence, in conjunction with SOM Transactions, provide datastore independence. This allows the location of an object's persistent data to be transparent to the object's binary class and its clients.

Our goals are

- To provide full datastore independence across all datastores, including both single-level-store (SLS where the object lives directly in a persistent memory, e.g., AS/400, ODI's ObjectStore) and TLS implementations (e.g., SQL database, IMS database, VSAM file system, Posix file system).
- To have a simple programming model.

A datastore that can only store data can only provide a TLS implementation (the object lives in regular transient memory and its persistent data lives in a separate datastore). A datastore that can store either data or objects can provide either a TLS or SLS implementation.

We found that we could provide a unified programming model for both TLS and SLS implementations by using the more abstract SLS programming model. Fortunately, this also achieves our second goal of simplicity. The programming model for datastore independence is the following:

- At object creation, specify whatever is needed for persistence via the PID. For example, for a Posix file system (a TLS implementation), the PID might say
  datastore_type = "PosixFS"
  path = "/u/mickey/pets/pluto"
  For AS/400's SLS implementation, the PID might say
  datastore_type = "AS/400_SLS_Native"
  package = mickey_pets_heap
  Either the PID or the persistent object's reference is typically input to the application.
- After object creation, use
  - The Transaction Service interface to begin, end and abort transactions. This provides all synchronization, including between the object's state and the datastore (for TLS) and between the datastore's cache and disk.
  - The CORBA interface to manage references. This includes object_to_string to get a stringified reference (the object id) from a live reference, string_to_object to get a live reference from a stringified reference, and release to deallocate all resources for a reference, etc.
  - The object's normal operations. This includes the object's functional operations as well as the destroy operation which deallocates all resources for the object.

- The PO interface store and restore should be used only for import/export kinds of operations (e.g., initializing an object just after creation or making a backup of an object).

Support for this programming model can be automatically inserted into a class in the manner described in Section 6. At object creation time, if persistence is desired, then a PID is also provided.

If the PID indicates a TLS implementation, the factory does the following:

1) Create a new class from the original class that adds in the mechanisms required to make it persistent and recoverable. This includes the following:
   - Before methods to register with the SOM transaction manager (which includes a request for the transaction manager to send preprepare notification to the object), and a restoring the object's state from the datastore (which allows the datastore to get data locks).
   - Additional methods on the object to implement the preprepare operation (which stores the object's state to the datastore).
2) Create an instance of the new class in regular volatile memory.
3) The PID is assigned to the object.

If the PID indicates a SLS implementation, the factory does the following:

1) Ask the original class how large it is.
2) Allocate memory for the new instance in the indicated SLS heap.
3) Create an instance of the class in the provided memory.

Some have argued that an OODB should always be used for persistent objects. In some cases, this is the best approach. However, in many cases, this is not desirable:

- Often the data is already stored in a datastore and moving it is not possible, because it would be expensive or traumatic, or there are procedural applications that are still using the data.
- Even for new data, often the performance characteristics of an OODB do not fit the application workload. For example, SQL systems typically run OLTP applications with several times the throughput of OODBs. OO applications need the same freedom to choose among these alternatives as procedural applications.
- Often the security or integrity characteristics of an OODB are not adequate for the application. Again, OO applications need the same freedom to choose among these alternatives as procedural applications.
- Often only a small per cent of the object's state need be persistent (the rest is derived and cached). Only this part of the state is included in streams using SOM Externalization and Persistence, whereas most OODBs store all of an object's state.

We view the SOM environment as an OODB that is much more open than typical OODBs. It allows data to be stored in any datastore, any transaction manager to be used, objects to reference or be referenced by objects in other CORBA-complient ORBs, remotely referenced objects to be handled via caching or message forwarding, support transient and persistent objects, etc. Much of the SOM environment can be specialized by customers for many different purposes.

# Notes on Third Generation Database Access

Wayne Duquaine
Grandview DB/DC Systems
10777 Cherry Ridge Road
Sebastopol, CA 95472 707-829-9633

Independent Consultant, Client/Server Interoperability

**HPTS 1995**

*Abstract: This paper discusses the recent changes occurring in the heterogeneous client/server database interoperability environment. In particular, it describes the growing movement away from proprietary protocols toward standardized DBMS interoperability protocols, and away from 3GL embedded SQL and native client APIs toward 4GL objected-oriented tools. It is based on "lessons learned" over the last 7 years in the client/server arena.*

# 1. Introduction

The database industry is in the midst of a set of very sweeping changes in terms of storing and accessing "mission-critical" data. Gone are the days of centralized (mainframe-based) data servers storing all of the mission-critical data. Instead, a plethora of database servers running on mainframes, Unix, and PCs, all containing some form of mission-critical data, is becoming the norm. Increasingly, these data servers are being used in client/server plug-and-play environments, accessed via GUI front-ends. This set of developments is forcing a fundamental re-thinking of how applications are built, and how networks of heterogeneous DBMS servers (and supporting transaction managers) interoperate.

This paper explores the major areas where these changes are taking place, and makes several projections based on historical trends, as to where the changes are likely to lead. This paper takes the "politically incorrect" view that the current proprietary Database Gateway "middleware" and associated protocols, which are the primary means of heterogeneous database access today, will ultimately go away, and be replaced by standardized protocols, which will almost certainly be IBM's DRDA (Distributed Relational Database Architecture). It further takes the view that DRDA has a better "fit" with the new crop of fourth generation (4GL) object-oriented client GUI tools, than the existing proprietary protocols.

## 1.1. Heterogeneous DBMS Access Evolution

Historically, access to heterogeneous DBMS servers has gone through the following stages:

- First Generation (1982-1988) - Primarily slow DSS-oriented systems, that were completely closed. These required special code on the mainframe, and typically did not work with the standard (DB2) database tools. These products used proprietary protocols and proprietary gateways to communicate with the mainframe.
- Second Generation (1989-1995) - OLTP oriented systems, featuring fast response times and Open APIs, allowing customers to write specialized code to access relational or non-relational databases. The APIs were vendor proprietary, and required special code on the mainframe. They used proprietary protocols and proprietary gateways. Different vendor's gateways could not cross-communicate to other vendor's gateways. Client/server computing is heavily embraced by all the "early adopters".
- Emerging Third Generation (1995-...) - OLTP oriented systems featuring fast response times and Open APIs. Industry standard APIs (such as X/Open SAG CLI or Microsoft ODBC) become the primary customer programmable interface, and (de facto) industry standard protocols that allow plug-and-play between different vendors clients and servers begin to emerge. Client/server computing becomes mainstream, and the "late majority" begin building and deploying client/server based systems.

## 1.2. Critical Infrastructure Pieces for Mission-Critical Client/Server

Client/server is now becoming mainstream, but it still has a significant number of rough spots that are in the process of being worked out. For client/server to succeed in displacing the currently ubiquitous 3270/VT-100 dumb-terminal/smart-host production application paradigm, the following key pieces need to coalesce:

- A common, set of interoperability standards needs to emerge, which will allow plug and play between different vendor's clients and servers. Currently, each vendor has proprietary interoperability protocols. This requires gateways (and in some cases, two tiers of gateways) to allow cross vendor DBMS access. What is needed is some form of industry "Esperanto" that will allow plug-and-play access between different vendors databases, running on different platforms and in different departments of a corporation.
- The above Esperanto must allow fairly "tight" integration with the back-end DBMSes, in order to ensure OLTP performance when processing "mission-critical" data. It must also provide robust 2PC support, when multiple DBMS servers are involved in a transaction.
- Good integration with the new 4GL GUI object oriented client access tools is required, since virtually all new client development will be done using 4GL tools.
- Logical high-speed bi-directional "pipes" between mainframes and LAN-based servers or departmental servers is needed, that will enable the mainframes to retrieve and update summary or detail data on those lower-tier servers.
- Mature Workstation Operating Systems, that are capable of supporting departmental "mission-critical" data on LAN-based DBMS servers.

# 2. Building the Esperanto - DRDA

The reality is that the existing "de jure" standards for DBMS interoperability, RDA and OSI have failed. RDA has never seriously taken off, and has no significant marketshare anywhere. OSI is, was, and continues to be a niche product, that is increasingly being clobbered by TCP/IP. At this point, looking for the sanctioned de jure standards to solve the interoperability problem is like waiting for Godot.

One of the key weaknesses that sunk the de jure standards was their emphasis on a "least common denominator" (LCD) approach. An LCD approach effectively cripples customers from using their 4GL front-ends to tap into all the various features that each of the different DBMS servers offer, and in many cases can severely reduce performance on the back-ends as well. LCD approaches fail, because customers want to be able to use the vendor specific features of a back-end.[1] They buy MVS DB2 because they want high-end robustness with thousands of users and terabytes of data, they buy Oracle because they like its rich functionality, they buy Microsoft SQL Server because it is cheap and they like Stored Procedures, they buy Informix because they like its parallel scalability, and so on. Nobody buys a database because they are in love with how well it conforms to the 1992 SQL standards. Customers *like* standards, but they *buy* product differentiation.

To avoid the least common denominator trap, DRDA allows server-dependent SQL enhancements to be utilized by the client on the target DBMS. DRDA "passes through" such requests, and lets the back-end either accept or reject the request. On occasion, this has caused much angst and wringing of hands among SQL purists about the lack of 100% SQL transparency in a DRDA environment. But different market niches dictate different requirements and feature sets, and users do want to take advantage of those features. Were that not true, everyone would buy the same database from desktop to mainframe.

[1] On a pragmatic note, the majority of the vendor differences manifest themselves in the DDL area (e.g the ability to tune the underlying databases by playing games with STOGROUPs, SEGMENTs, index clustering, etc). Most of this work is done by DBAs, not end-users or application programmers. SELECTs tend to be the other major area of weirdness. Most real world apps (probably 80%) tend to be fairly simple, standard SELECT/UPDATE/DELETE type operations. The areas where vendors' featuretttes are heavily used tends to be in a small number of very critical transactions, where the customer wants to use the vendor extensions to super-tune the application. LCD approaches get in the way of allowing these kinds of workload optimizations.

In reality, a fully transparent heterogeneous SQL dialect probably won't happen within our lifetimes. Hence, DRDA's "let the target process it" escape mechanism, is the most reasonable pragmatic solution. DRDA will win in the "real world" because it effectively bridges the above mentioned customer dichotomy.

One of the underlying forces that are driving the need for some form of Esperanto is the fact that virtually all F1000 corporations have at least 3 different vendor's databases deployed throughout their enterprise, and in many cases have five, six or more. Studies by DBMS market research firms show the following "average" F1000 profile in terms of deployed DBMSes:

1. Nearly every F1000 has IBM's MVS DB2 database deployed at the corporate level
2. Nearly every F1000 has Oracle installed at the departmental or regional level
3. Nearly every F1000 has a third vendor's DBMS installed - a two-way tie between Informix Turbo or Sybase's SQL Server DBMS.

In addition to the above, follow-up research by Forrester indicates that Microsoft's SQL Server has "slipped in the back door" at the majority of F1000 corporations (typically in a workgroup or departmental setting). As corporations continue to "down-size" and push decision making and P&L responsibility into lower levels of the corporation, the need to interconnect the various mainframe, Unix, and PC based DBMS servers will increase.

Current market projections indicate that a veritable "explosion" will occur over the next 5 years in workgroup and departmental based LAN servers. This will further exacerbate the need for plug and play interoperability, not only between departments and mainframes, but between and across departments as well. While Microsoft's ODBC offers a potential help, it can also degenerate into a gazillion different protocol stacks running off to different vendor's servers, with the inevitable excessive storage, bugs, maintenance currency issues, etc. The logistics for a F1000 organization with hundreds of LAN-based servers and thousands of desktops, with each desktop having to support multiple different protocols stacks (Oracle, Sybase, Informix, Gupta, DB2, IBI, ...) to communicate with different departmental servers is staggering. The maintenance headaches and costs of keeping such an environment operating reliably would be a huge expense. To cure this maintenance nightmare, corporations will ultimately start winnowing down the number of disparate DBMS client/server protocols. Just as the number of proprietary communications protocols have significantly dropped over the last few years[2], ultimately, the number of proprietary DBMS protocol stacks will be reduced as well, with one or two protocols ultimately dominating as de facto standards.

## 2.1. So what is DRDA

In a nutshell, DRDA is a fully self-describing datastream, designed on an object oriented architecture. It features a layered architecture, where the database requests and replies are cleanly separated from the data contents[3]. Overall, the wire flows closely match an embedded SQL API (e.g. an EXEC SQL OPEN Cursor causes an DRDA Open Query request to flow, a Dynamic EXEC SQL PREPARE causes a DRDA Prepare Statement request to flow, status replies are sent back as wire-ized versions of SQLCAs and SQLDAs, and so forth). DRDA utilizes an efficient length-type-value (LTV) encoding scheme, where every command request, parameter, data object, and reply is encoded in a self-describing LTV format. Result sets are efficiently encoded such that column descriptor information is sent back only once, at the very front of the result set. A wide range of data types (integer, float, char, varchar, decimal, long varchar, binary, blobs, etc) can be sent as input parameters, output parameters, or row results. Recent enhancements have included support for distributed two-phase commit, invocation of stored procedures, and support for multiple result sets.

---

[2] In the Unix and mainframe markets, TCP/IP and SNA dominate, owning 90% market-share between the two of them. In the still young PC market, IPX/SPX and NetBIOS/NetBEUI currently dominate, but TCP/IP is making extremely rapid inroads. With Microsoft's latest bundling of TCP/IP into NT and Win95, NetBIOS is effectively dead, and IPX/SPX will probably be very seriously wounded. By 1996 it is reasonable to expect that TCP/IP will be the protocol-of-choice for LAN-based servers.

[3] For example, the DRDA database command request is always sent first (such as a Execute Immediate request), then a specifically tagged object containing the actual data or parameter contents (such as SQL statement text) is appended behind it. In the future, the tagged object's data contents could be a spoken voice command, rather than an SQL text statement. The DRDA command requests would be unaffected by the change in data contents. In OOB-ese, DRDA database requests and replies are polymorphic.

DRDA utilizes a "receiver-makes-right" philosophy, such that clients send the commands and data in their native format (e.g. Intel x86 little-endian ASCII), and the receiving server converts it to its internal format. Conversely, the server sends replies and result data back in its native format (for example 370 big-endian EBCDIC), and the receiving client converts it to its internal format. Part of the initialization handshake when a DRDA conversation between a client and server starts up exchanges what type of receiver and sender each side is.

In practice, DRDA and its receiver makes right approach has proven to be quite robust, and transparent to applications utilizing its services. The author has built DRDA server software that can transparently take mainframe SQL requests originating from SPUFI and COBOL applications, and have those requests access or update LAN-based Oracle and SQL Server database servers[4]. From the SPUFI or COBOL applications' standpoint, they cannot tell the difference as to whether the data was obtained from the local DB2 database, or from a different vendor's database operating as a remote DBMS server. DRDA is real, it works very well, its fast, and it's plug and play. The rival de jure standards have failed, while many of the existing proprietary solutions are slow, and cannot do multi-vendor plug-and-play.

The cost to build a DRDA client and server implementation is approximately 40,000 executable lines of code. This is roughly on par with the lines of code needed for Sybase's TDS 5.0 proprietary protocol, probably slightly more than Oracle's SQL-Net, and noticeably less than the size of building a full RDA implementation. The biggest knock and stumbling block against building a DRDA implementation is the documentation (in particular, the DDM documentation)[5]. It often requires a number of line flow traces and laborious tests to completely get the implementation right. The other DRDA knock is the lack of "native" TCP/IP support (IBM currrently encapsulates SNA-based DRDA requests over TCP/IP). This is a political issue, not a technical issue.[6] In spite of those irritations, over 13 non-IBM vendors have built DRDA support (mostly client-side access to date).

Enhancements to the DRDA architecture are performed through two avenues: an IBM internal architecture board that is used by IBM products, and an external review board (the "DRDA ISV Council"), composed of all the non-IBM ISVs that have implemented DRDA products and/or licensed DRDA technology. Licensing is non-discriminatory (e.g. IBM has licensed DRDA to competitors such as Oracle), at reasonable fees and conditions for all involved.

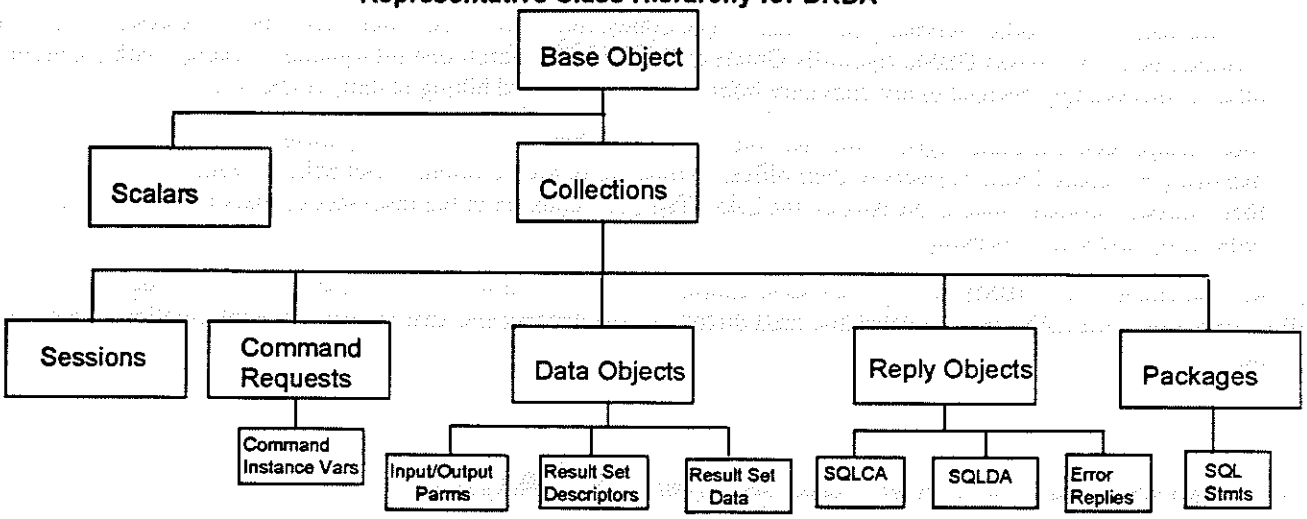
As previously mentioned, DRDA is based upon an object-oriented architecture. To date, DRDA is the only client/server database protocol that has been designed from the ground up to be object-oriented. All of the current proprietary client/server protocols (Oracle SQL-Net, Sybase TDS, IBI FocNet, ...) are all based on older 3GL oriented technology. A diagram depicting the major object classes in DRDA is shown below..

[4] DB2 3.1 communicating to AIX 3.2 and OS/2 2.1 based versions of Oracle and SQL Server. (An NT based version was also available but not used during the testing).

[5] It has been rumored that the DDM documentation writers were ex-archeologists who worked on translating the Rosetta Stone's hieroglyphics into ancient greek. Given the significant amounts of barely decipherable greek that exists in the DDM documentation, this would appear to be an accurate rumor.

[6] Specifically, the mainframe-based DRDA products are reluctant to step up to native TCP/IP support due to a lack of resources. The mainframe world continues to be dominated by SNA, while the LAN world is increasingly dominated by TCP/IP. Currently DRDA only runs over SNA protocols. Using "AnyNet" technology, the SNA packets can be transported over TCP by encapsulation. But it still requires an SNA protocol stack on each client. The IBM SAA strategy was predicated on a similar "SNA on every desktop", and it failed miserably. The complexity of installing SNA on every desktop is not feasible in the bulk of most departments. While there has been recent hand-waving by the mainframers that "we'll make SNA installation a no-brainer and invisisble to those users", in 18 years of working with SNA I have yet to see it happen. Apparently IBM has still not learned from the SAA fiascos.

**Representative Class Hierarchy for DRDA**




# 3. Moving TM functionality out to the LAN

Like it or not, mission-critical data now lives out on the LAN. And as time goes on, even more mission-critical data is going to live out on the LAN. The upshot of this, is that ultimately Transaction Managers (TMs) will have to live out on the LANs, in order to ensure correct ACID support in a distributed client/server environment.

## *3.1. Down-sizing: Forward into the Past*

Looking from a historical perspective, it can be argued that the centralization of data into a single spot (e.g. a mainframe)[7], was somewhat of a historical aberration. Through-out the 1800s and the first half of this century, nearly all corporate "mission-critical" data was local. Accounting, payroll, inventory, etc data was kept via manual systems at the local branch office, local branch bank, or local warehouse. Summary data was then prepared daily, weekly, or monthly, and sent to the "head office", where it was manually entered into the corporate books.

With the widespread introduction of corporate computerization in the late 1950s and early 1960s, the departmental and branch office manual book-keeping systems became replaced by computerized book-keeping systems. Unfortunately, given the technology at the time, it was too expensive to place computers in every branch or department. So inevitably, the users had to reluctantly relinquish control of their "local" book-keeping data to a centralized computer complex. Centralizing things in mainframes at the time was a necessary compromise, because today's infrastructures (cheap micros, LANs, etc) did not exist. Yet the movement of "local" data to a centralized spot was not without its pains and complaints. The irony is that studies by both IBM and Xerox in the 1970s and again in the 1980s showed over and over again that over 80 % of the critical data needed to run a business on a day-to-day business is ideally located within 1200 feet of the department or people needing the information.

The emergence of low-cost MIPs, robust PC and Unix based DBMS servers, cheap large (1GB+) disks, and fast LANs has allowed data to be moved back out to the departments that use the data for day to day operations. Trends indicate that as corporations "down-size" and "right-size", more and more data will eventually go out on the LAN. The new client/server technologies allow the local users to "get their data back", and many of them are doing it with a vengeance. For example:

- Most Wall Street traders now use "real-time" position keeping of their portfolios, with their current stock/bond positions being recorded and tracked on their local SUN (or AIX) workstation running a Sybase or Oracle DBMS server. Summary information is periodically sent up to DB2 on MVS. From the "big bucks" trader's position, their mission-critical data lives on the LAN-based Oracle or SQL Server. MVS DB2 is nothing more than a back-end trade summary engine.

[7] The reality is, the data was never truly centralized into a mainframe either. The data was put into a small number of regional or local mainframes, which were then lashed together, using either tape dumps or message switching between the systems to exchange data.

- Several of the major brokerage houses are now starting to move their customer account data off the mainframe and back into the local branch office servicing the account. The branch office and local broker now "own" the customer data, and it resides on a LAN-based DBMS (typically Oracle or Sybase). The mainframe information exchange with the branch office is increasingly devoted to just summary information (roll-up and billing of daily trades, etc).
- Most independent insurance agents (who sell and service over half of the insurance policies in the U.S.) are now deploying PC-based DBMS servers in their offices to track all of their customers and policies. From their perspective, their "mission-critical" data is the data on the LAN. The 3270 hook-up to the insurance carriers is only used for initial order entry and status checking.

As LAN and client/server DBMS management tools continue to improve, it is inevitable that even more mission-critical data will move out onto the LAN, to be updated and used directly by the departments that are using the information on a day-to-day basis.

## 3.2. Server PCs: It's Not Your Mother's PC Anymore

PC based servers are encroaching on Unix's and mainframe's turf. Indeed, in some areas, the PCs are pulling ahead, particularly vis-a-vis Unix. Compaq Proliant Servers provide "hot-pluggable" disks (something which not all Unix servers can boast), duplex-ed controllers, multiple busses, "hot standby" processors, multi-bit error detection and correction for main-memory, and sophisticated environmental monitoring. Advanced PC software operating systems are starting to incorporate sophisticated, MVS-like recovery schemes - for example Microsoft's NT Advanced Served can successfully recover from most swap device failures. (An area where most Unix boxes still crash and burn with the infamous "panic" message.) Hence, the concept of putting mission critical data on LAN-based servers is no longer a "high-risk" bet. The reliability of those machines exceeds the reliability of 308x class mainframes of a decade ago.

It is not an exaggeration to say that today's 100 Mhz Pentium-based servers, with RAID "hot-pluggable" disks, large 32-256 MB memories[8], and robust multi-tasking operating systems (NT Advanced Server and OS/2 SMP) are effectively 308x-class mainframes, yearning to break free of the PC stigma.

Finally, the highest RDBMS growth rates over the next 5 years are projected to be in PC-based servers. The hottest RDBMS market is for "Workgroup" and departmental PC-based servers, which is growing in excess of 70% per year (in contrast to Unix-based servers which are growing at approximately 35%). With Oracle and Microsoft aggressively pursuing this market, there will be an explosion in the number of PC-based RDBMS servers. In the F1000 world, this will entail a strong need to interoperate these new LAN-based servers into the existing mainframe and Unix based server networks.

## 3.3. Mainframes: Accessing LAN-based Servers

As data continues to move down, and new "mission-critical" data is created on departmental and workgroup systems, the need for existing corporate mainframe applications and tools to access that data on the LAN will be overwhelming. Thus, it will increasingly become common for today's mainframe-based applications to not only access, but also update data on databases residing on LANs, outside of the glass house.

Over the next few years, we will begin to see an inversion of the traditional master/slave host/terminal hierarchy, into a peer-to-peer client/server orientation. The "central host" will gradually just become another server (albeit a very large one). But corporate applications will still need access to the various data which has now been down-sized to (or new data created on) the lower tier systems. Weekly, monthly, yearly summary reporting and collection of data will still be required at the corporate level, and mainframes are still the most cost effective at coordinating and generating such information. Rather than the mainframe being the "big cuhuna" that all the lower tier systems slavishly request data from, it will instead see its role

[8] By way of contrast, the largest mainframe, circa 1984, was a "4-way" 3084 mainframe, with a maximum main memory size of 64 MB with simple ECC and single-bit error correction. The one area the 3084 still has over Server PCs is that it had built-in "power partitioning" (half the machine could be powered down and repaired while the other half kept running). In all other respects, the Server PCs have higher reliability MTBF numbers (2,000 hours for 3084 versus 20,000 hours for Server PCs), larger main memories, and equal or better online DBMS throughput numbers over a 3084.

evolve into a top level coordinator and summarizer. The mainframe will have to go out as a client and request data (or update data) on the LAN-based servers.

With technologies such as DRDA, it is trivial to take existing mainframe-based relational applications, and re-route them to request data off other DRDA servers (e.g. LAN-based servers). It just requires entering the name of the remote LAN-based server into the SYSLOCATIONS table, and setting up the LAN server's node name in the SYSLUNAMES table. None of the existing proprietary "middleware" solutions provide such an easy migration path.

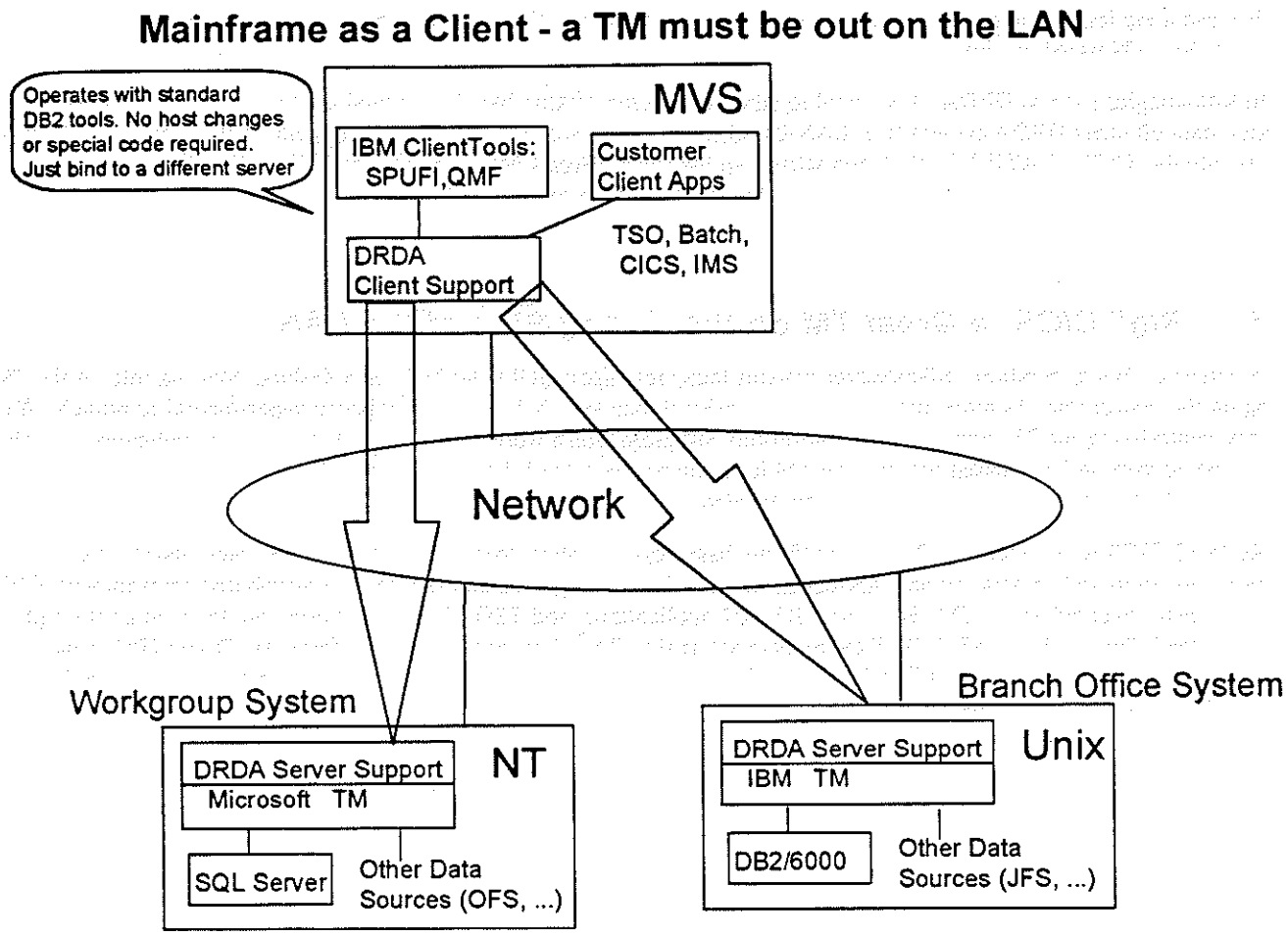
## 3.4. "Big" CICS: a Great TM on the Wrong Side of the LAN

Most existing "first generation" client/server systems today rely upon GUI front-ends on a desktop hooking into on the TM being on the mainframe. As more and more mission-critical data moves down to LAN-based departmental or branch-office servers, centralizing the TP monitor in the mainframe just plain won't work. In the case of mainframe applications reaching out to update data on LAN-based servers, the TM is on the wrong side of the wire ! The real-world rule of thumb is that you want the TM near the database(s) that are being updated.

While MVS CICS is an excellent TM for mainframe-based apps, it often doesn't buy much value when mainframe applications try to update data out on LAN-based servers. When using DB2 and DRDA in a distributed environment, CICS is often not even involved as the TM. Batch or TSO DB2 applications, and TSO SPUFI, et al users do not even go through CICS. Instead, DB2 and its DRDA DDF component act as the TM.[9] This means that to enforce ACID and 2PC semantics in a host-to-LAN server environment, a TM function will need to reside on the LAN, and the host components will need to interoperate with the TM on the LAN[10].

[9] This is true regardless of whether the remote database is another DB2 system or a LAN-based DBMS system. CICS is only involved as a TM if an active CICS application is making client calls to a local DB2, or to a remote DBMS through the CICS-DB2 interface.

[10] For example, the new DB2/6000 Version 2 support requires either CICS/6000 or Encina/6000 to be present on the system, in order to allow two-phase commit to be performed between a local DB2/6000 and other DB2 databases.




# 4. API Changes: 3GLs Replaced by Client GUI Objects Communicating to Server DBMS Objects

The rules for writing client/server apps have significantly changed over the last six years. The first generation client/server DBMS LAN-based apps almost always used the "native" programmable third generation language (3GL) based APIs for communicating with the DBMS. Example of these include Oracle's OCI, Sybase's DB-Lib, or 3GL embedded SQLpre-compilers. Results from informal surveys in 1990, showed that approximately 90 % of the LAN-based apps were using 3GL techniques, and only about 10 % were using any form of 4GL GUI tool. By mid-1994, this situation had almost completely reversed - over 80 % of client development was being done using 4GL or related "visual" design tools, most typically PowerBuilder, SQL Windows, and Visual Basic. Less than 20 % of client development was still being done with 3GL tools ("native" API calls or embedded SQL). This was due to two major reasons:

(1) the 4GL tools were providing measurable 10-to-1 productivity improvements over 3GL methods, and

(2) the PC client hardware was moving off the old slow 286/386 based platforms, and onto 486-based platforms which could adequately handle the overhead associated with the first generation 4GL tools.

With the emerging "second" generation 4GL GUI tools, such as Borland's Delphi, Oracle's Power Objects and Oracle Objects for OLE, Gupta's SQL Windows 5.0, etc, the remaining performance gap between 3GL tools and 4GL tools will effectively disappear, since most of these newer products can produce compiled code, rather than interpreted code. Ultimately this means that the future of "native" APIs such as OCI and DB-Lib will eventually wither into a niche market, used only by ISVs and 4GL tools vendors "under the covers". Client GUI development will ultimately rest upon object-oriented GUI 4GL paradigms and tools.

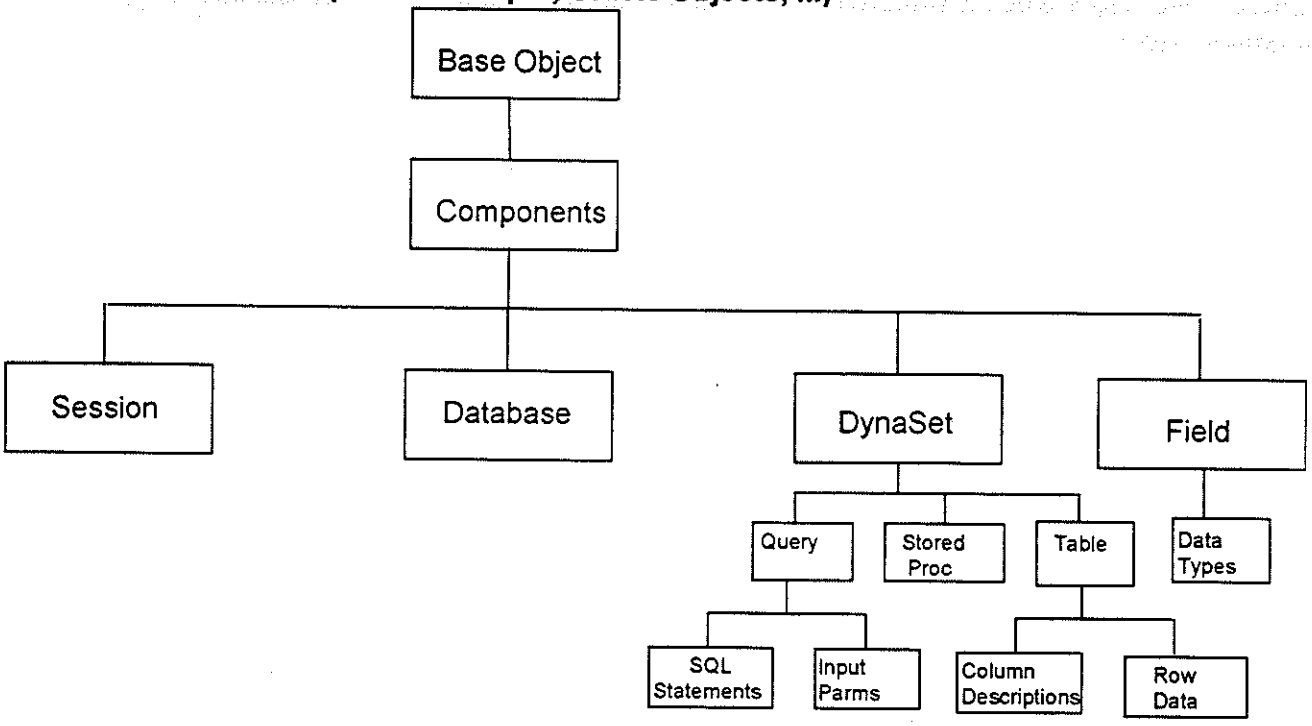
That's the good news. The bad news is that a lot of the "back-end" server DBMS stuff (particularly mainframe related) is still dominated by 3GL techniques (usually embedded SQL in COBOL programs). As the 4GL tools add new functionality,

such as support for persistent SQL objects, this backlog could quickly be alleviated. DRDA allows "static" SQL to be compiled and stored on the host, without requiring any host COBOL programs to be written. In DRDA-ese, these are referred to as packages. If 4GL tools could merge persistent SQL object support with DRDA's package support for static SQL, the need for hand-written COBOL programs on the mainframe for OLTP environments would be drastically reduced, if not eliminated.

## 4.1. Typical 4GL GUI Object-Oriented Class Hierarchy

Most of the new 4GL tools utilize a fairly common set of classes and hierarchies for database access. While the various products differ in some of the details, the majority of them support a class hierarchy similar to what is shown below.

**Representative Class Hierarchy for 4GL Client DBMS Access (Borland Delphi, Oracle Objects, ...)**




It is instructive to compare this class hierarchy diagram for GUI tools, with the class hierarchy diagram for DRDA. While some of the boxes have been moved around to different spots in the hierarchy, the basic objects (sessions, parms, SQL statements, descriptors, fields/scalars) are consistent between the two models.

# 5. Summary

Nature (and customers) abhor a vacuum. Some form of standardized heterogeneous plug-and-play protocol is going to emerge to unite the disparate mainframe, Unix, and PC worlds. "De jure" standards such as RDA have failed. The de facto DRDA standard is the odds on favorite to win the interoperability wars, assuming IBM doesn't do something stupid.[11]

3GL proprietary vendor DBMS APIs are entering a period of de-volution. 4GL object-oriented GUI and visual tools will rule the market for end-users and most corporate developers. The new "second generation" of ODBC and X/Open CLI drivers have dramatically improved in performance. Most drivers now have no more than a 5-10% performance difference between

[11] Such as severely restricting licensing or raising the cost of licensing DRDA, not publishing new architecture changes or enhancements on a timely basis, or not aggressively pursuing "native" TCP/IP support for DRDA on the LAN.

the ODBC driver and the DBMS vendor's "native" API drivers. This will cause ISVs and tools vendors to increasingly rely upon industry standard APIs such as X/Open SAG CLI and ODBC to build their "low-level" client/server calls, rather than upon the proprietary database vendor APIs.

Over the next few years, the traditional departmental desktop-to-host data access model will become inverted. Mainframes will need to reach down to departmental/LAN-based DBMS systems to access and update mission-critical data on those systems. The mainframe will begin to take on the role of a client. The major challenges to be solved as more mission critical data resides on the LAN-based servers, is how to coordinate and interoperate the mainframe applications and TM support with the TM support that resides on the LAN-based systems.

More effort is needed to better understand how to map object-oriented client facilities onto some of the newer DBMS server features, to yield better integration and productivity. In particular, mapping "stored procedure" concepts into object-oriented front-ends today is still klutzy. The lack of support for "persistent" pre-compiled SQL objects (e.g. DRDA static "packages", or SQL 3 "modules") is another area of weakness. Finally, most of the current GUI tools are often restricted to only one database at a time, and only a few tables at a time. Some form of multi-database oriented browsers that would allow simultaneous browsing of different databases on different servers would be highly desirable for both DSS and Systems Management support.

# Queuing Systems: a Floor Wax *and* a Dessert Topping?

Jeffrey L. Eppinger
Director of Product Management
Transarc Corporation
Gulf Tower
Pittsburgh, PA 15219

September 1995


Listening to the debate about the purpose of queuing systems in transaction processing reminds me of the Saturday Night Live commercial about the fictitious product called Shimmer. In the commercial a young married couple is squabbling. The wife demonstrates that Shimmer is a great floor wax, while the husband shows that Shimmer is a delicious dessert topping. The announcer proclaims that they are both right: Shimmer is a floor wax *and* a dessert topping!

How analogous is the debate about queuing systems? Some claim that queuing systems are databases while others claim that queuing systems are messaging systems. The problem with Shimmer is that I don't think I would like to use it on my floors and I *know* I don't want to eat it. The issue for queuing systems:

1. Would you use a queuing system in place of a relational database?
2. Would you use a queuing system in place of a request-response communications system?

I argue that (1) queues make a fine databases for certain applications, such as workflow, but not (2). Queues should not be used in places of request-response communication systems, such as remote procedure calls.

## Queues as Databases

For workflow applications, such as customer service requests, printing packing lists, and funds transfers, a queuing system is a great paradigm. Data can be stored in queues for later, asynchronous processing. Two-phase commit and the ACID properties guarantee that queued data will not be lost due to system failure. Data can be dequeued with transactional integrity so that if the subsequent processing fails, the transaction aborts and the data remains in the queue.

Queuing Systems: a Floor Wax *and* a Dessert Topping?

Queues have several advantages over a relational database:

1. **Queues provide a structuring mechanism for workflow applications.** In a relational database, queues could be implemented using stored procedures, but most customers would rather purchase a queuing system. Most ISVs find it simpler to either build their own queuing system from scratch so that they are not tied to any particular database or to bundle their queuing system with a comprehensive turn-key application running on the database. So, there is a market for turn-key queuing systems not built on a relational database.

2. **Queues provide an enhanced security model.** The implementation of the queuing system guarantees the integrity of the queue data structures plus provides access control on which users can access which queues. This is not easy to do with stored procedures, today. In the future, stronger security models may become easier to implement in a relational database as stored procedures incorporate object models that will allow the instantiation of queue objects, implemented with common code, but with different access control lists.

3. **Queuing systems provide better performance.** With custom data structures and no overhead for using SQL or stored procedures, queuing systems can out performance relational databases. Although, there may be additional cost for two-phase commit if the work were otherwise to be done completely local to the relational database.

4. **Queuing systems provide simplier administration.** Administration commands in queuing systems permit the administrator to inspect and manipulate queue objects. Such operations may make much more sense to the administrator than manipulating relational database tables that use many parts to represent the queues. (Of course, one must keep in mind that the customer must have a significant application because one does not want to take on the administration of an additional type of system lightly.)

## Queuing Systems as Messaging Systems

Queuing systems are very good for asynchronous transmission of data, however queues are not so good as a basic communications paradigm for synchronous communications in transaction processing systems. Queues are either volatile or recoverable:

1. **Volatile queues are not a good paradigm for synchronous communication.** The application must write additional code: code to send the request, code to receive the reply, and code to cope with all of the errors. Compare this with a remote procedure call. With an RPC, the application only makes a call to send and receive the reply and must cope with fewer errors. In case of system failure, there is little difference in the outcome of the communication: you get no response indicating whether the request executed before the failure. After restart, the request will not execute (if it hadn't already executed before the failure).

2. Recoverable queues provide an additional point of comparison. Recoverable queues still have the same additional code requirements, but in the case of system failure the request will execute and the response will be provided after restart. There are two additional concerns about recoverable queues as a communications paradigm:

   (a) **Performance concerns.** The recovery guarantee is given at the expense having to store the requests and the responses on disk. This can be very expensive, although through the use of batching, many requests and/or responses can be written to disk at one time.

   (b) **Requester looses interest.** What is the benefit of recovery guarantee? The vast, vast majority of the requests execute correctly. In cases of failure, the likelihood of the requesting application still being available is small. The PC will have been rebooted, or the customer will have hung up the phone.

## Conclusions: a Floor Wax, but Not a Dessert Topping

In conclusion, queuing systems make a fine database. The structuring queues provide is essential for many types of applications and can provide enhanced security, performance, and simplier administration.

Queuing systems do not make a good messaging system. Remote procedure calls allow simplier coding and better performance. Some applications do use queuing systems as a messaging paradigm for interoperability across heterogeneous systems. However, if a common remote procedure call mechanism were provided everywhere, we would be a better off.

# "Shared Air" - Exploiting Broadcast in Large-Scale Information Systems*

(Position Statement for HPTS '95)

Michael J. Franklin

Department of Computer Science
A.V. Williams Building
University of Maryland
College Park, MD 20742
*franklin@cs.umd.edu*

## Introduction

The perennial HPTS debate among advocates of the Shared Disk, Shared Memory, and Shared Nothing architectures, while still unresolved, has grown somewhat stale. More importantly, many emerging Information Systems application domains present new constraints that are not addressed by these traditional architectures. One such constraint is that of *Communications Asymmetry* — the communications bandwidth available for transmitting information can differ widely between the nodes of a distributed system. A common form of asymmetry occurs when the *downstream* communication bandwidth available from servers to clients greatly exceeds the *upstream* bandwidth available from clients back to servers. This type of asymmetry arises in a wide range of environments, including:

- Wireless networks with mobile clients.
- Cable and direct broadcast satellite TV systems.
- Advanced traffic information systems.
- Information dispersal applications.
- Information retrieval systems.

In such asymmetric environments, the advantage in bandwidth from servers to clients can be exploited by broadcasting data to the clients rather than (or in addition to) serving data to clients in response to specific requests. In systems that exploit server-to-client broadcast, the broadcast medium is a key shared resource. Therefore, we refer to broadcast-based systems as "Shared Air"

*Partially supported by NSF grant IRI-9501353 and a research grant from Intel Corporation.

architectures. It is important to note, however, that while data broadcast over airwaves is likely in many scenarios, broadcast will also be used over more conventional, wired media.

## Communications Asymmetry

Communications asymmetry can arise in two ways: The first is from the bandwidth limitations of the physical communications medium. An example of physical asymmetry is a wireless environment in which stationary servers utilize a high bandwidth satellite broadcast while (possibly mobile) clients cannot transmit at all, or can do so only over a lower bandwidth (e.g., cellular) link. Perhaps less obviously, communications asymmetry can also arise from the patterns of information flow in the application. For example, an information retrieval system in which the number of clients is far greater than the number of servers can be asymmetric if there is insufficient capacity (either in the network or at the servers) to handle the simultaneous requests generated by the multiple clients. Also, it is important to note that asymmetry can arise either statically, through invariant properties of the physical devices and/or workload, or dynamically. Dynamic asymmetry can arise, for example, in wireless network if mobile clients become temporarily incapable of transmitting due to environmental considerations (e.g. interference), or are temporarily prohibited from transmitting (e.g., on a commercial airplane flight).

## Broadcast Disks

In an asymmetric communication environment, data broadcast can be used to exploit the servers' advantage in transmission bandwidth in order to provide responsive data access to clients. Data broadcast has been explored previously, in the Boston Community Information System of Gifford [Giff90], in which news and other items were broadcast over FM to PC's around the Boston area, and in the Datacycle project at Bellcore [Herm87], which continually broadcast data over an optical ring and used special filtering hardware to extract required data from the broadcast. In recent work with Stan Zdonik, Swarup Acharya, and Rafael Alonso, I have been investigating a technique called "Broadcast Disks" [Zdon94] that goes beyond this earlier work. As in Datacycle, Broadcast Disks uses a broadcast stream consisting of data that are repeatedly and cyclicly transmitted as a type of storage device. The essence of the Broadcast Disks technique is to superimpose multiple broadcast programs (or "disks") spinning at different speeds on a single broadcast channel. This in effect creates an arbitrarily fine-grained memory hierarchy. The advantages of this approach are twofold. First, it provides improved performance for non-uniformly accessed data by allowing the broadcast program to be tailored in a way that reduces the delay for "important" data (at the expense of less important data). Secondly, the availability of critical data can be improved by placing that data on the faster (and thus, more frequently repeated) disks. In addition, any number of clients

can listen to the broadcast without impacting performance, and thus, broadcasting can have great scalability advantages for applications in which clients primarily read (rather than update) shared data.

The Broadcast Disk approach also encompasses the use of client storage resources for caching and prefetching data that is delivered over the broadcast. Caching improves performance for frequently accessed data and allows clients to continue to access such data even if network connectivity is reduced or lost. Prefetching is a more aggressive use of local resources that opportunistically absorbs important data from the broadcast in order to anticipate future data requests and protect against future connectivity lapses.

## Architectural Issues and Design Challenges

The central problems that arise in the development of the Broadcast Disk paradigm can be divided into server-side and client-side issues. On the server side, the issues involve: determining a good broadcast program for a given client population, coping with changes in the priority of data items and the needs of mobile clients, and providing timely dissemination of modified and new data items. The client-side issues involve: devising cache management policies that improve responsiveness for a given broadcast program, developing simple and effective prefetching strategies, and determining the best use of an upstream channel, if and when one is available. Furthermore, although it is conceivable that many systems will use data broadcast as the primary method of data delivery to clients, it is also interesting to look at hybrid architectures in which data broadcast is used in conjunction with request-based data access. In this manner, the Shared Air approach can be integrated with the more traditional systems architectures.

Our initial work on broadcast disks has focused on a restricted environment in which data is accessed in a read-only fashion by a static client population that has does not use an upstream communications capability. Even in this highly restricted setting, we have found that the inversion of the traditional relationship between clients and servers that arises in the broadcast environment has significant implications for the management of client storage resources.

In [Acha95a] we show that traditional LRU-based cache replacement policies do not work well in a broadcast disk environment. The problem is that because the broadcast program is a shared resource, it must be designed taking into account the needs of the entire client population. Such a broadcast is unlikely to be ideal for any individual client — a client may find that some important (to it) data items are being broadcast infrequently (i.e., on a slow disk), while relatively unimportant data items are broadcast on faster disks. To address this mismatch, the client cache replacement policy must be *cost-based.* That is, the cost of re-accessing an item must be factored into replacement decisions. We have developed a policy called $\mathcal{LIX}$ that addresses this need and have shown that it approaches the performance of an idealized cost-based policy.

In [Acha95b] we examine the unique opportunity for prefetching that exists when using data broadcast. The dissemination-based nature of the broadcast environment makes it particularly conducive to prefetching. In traditional disk-based environments, prefetching is a risky business because it places additional load on shared resources (i.e., the disks) in anticipation of possible future requests. Performance for all users can suffer if this gamble does not pay off. In contrast, in a broadcast environment data pages continually flow past the clients, so prefetching can occur without placing additional load on shared resources. Only a client's local cache is affected. We describe a simple heuristic for prefetching and examine implementable approximations to it. We show that in contrast to traditional systems, prefetching from a Broadcast Disk improves performance not by increasing the cache hit rate, but rather by reducing the penalty that is paid on a cache miss.

## Conclusions

The development of systems architectures that can meet the requirements of emerging application domains and environmental considerations is crucial if HPTS systems are to remain a relevant technology. The Shared Air approach has the potential to provide performance, availability, and scalability improvements for a large class of new applications — in particular, those that exhibit communications asymmetry. The inversion of the traditional client-server relationship that results from broadcast-based information delivery changes many of the fundamental tradeoffs for client resource management. As an additional benefit, the discussion of such architectures can add some spark to the inevitable debate on Shared "X".

## References

[**Acha95a**] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments", Proc. ACM SIGMOD Conference, San Jose, CA, May, 1995.

[**Acha95b**] S. Acharya, M. Franklin, S. Zdonik, "Prefetching From a Broadcast Disk", *Technical Report CS-TR-3511*, University of Maryland, College Park, August, 1995.

[**Giff90**] D. Gifford, "Polychannel Systems for Mass Digital Communication", Communications of the ACM, 33,(2), February, 1990.

[**Herm87**] G. Herman, G. Gopal, K. Lee, A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems", Proc. ACM SIGMOD Conference, San Francisco, CA, May, 1987.

[**Zdon94**] S. Zdonik, M. Franklin, R. Alonso, S. Acharya, "Are 'Disks in the Air' Just Pie in the Sky?", IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, December, 1994.

Position Paper For HPTS95
Jim Gray
Computer Science, UC Berkeley
310 Filbert St., S.F., CA. 94133
Gray@crl.com

# THESIS: Queues are Databases.

**Abstract:** Message-Oriented-Middleware has become a popular buzzword. It represents an industry that offers queued transaction processing as an advance over pure client-server transaction processing. This brief note makes three points:

1. Queued transaction processing is less general than direct. One can build a queued system on top of a direct system. One cannot build a direct system atop a queued system. Conversational and distributed transactions are very difficult in a queued system.
2. Queues are "interesting" databases with interesting concurrency control. It is best to put these mechanisms into a standard database so other applications can use these interesting features.
3. Queue managers are simple TP-monitors managing server pools driven by queues.

**Acknowledgments:** These ideas derive from discussions with Andrea Borr, Richard Carr, Dieter Gawlick, Franco Putzolu, and Andreas Reuter.

## 1. Queues Are Best Built Atop Direct TP Systems.

Traditionally TP systems offer both queued and direct transaction processing. They offer both client-server and peer-to peer direct processing. Gray & Reuter [pp. 246] offer the following common taxonomy:

- Process-to Process
  - Direct
    - Peer-To-Peer
    - Client-Server
  - Queued

In queued processing, requester processes place request messages in a queue. A pool of server processes, managed by a TP-monitor, service these request queues, perhaps placing results in other queues. Requestors can poll these output queues to see the status or outcome of their transaction requests.

Queued processing is the basic mechanism of IMS, so we have 30 years experience with its pros and cons. Advocates of queued processing point out that, at saturation, a direct system is really a queued system: servers are dispatched via a queuing mechanism. When clients saturate a server pool, the queues become visible. Indeed, it is optimal to schedule new requests to a server pool via a single global queue.

The difficulty is that queued transaction processing of a request-response is three ACID units:

1. Requester places request in queue.
2. Server dequeues request, performs task, enqueues response.
3. Requester dequeues response from output queue.

This tri-ACID unit model has the virtue of decoupling the requester from the server, but has the flaw that it makes multi-request transactions impossible. Implementing distributed transactions, conversational transactions, or multi-step transactions on top of a queued system requires building a lot of applicaton-level machinery.

By contrast, direct transaction processing systems can easily add a queuing mechanism by implementing a direct transaction that places requests in queues, and by having pools of servers that poll these queues. Indeed, this is the course that CICS, ACMS, and Encina have taken. At last count, CICS had over six distinct queue managers as part of the regular product -- each with slightly different performance-functionality tradeoffs.

We are not arguing that queued processing is bad -- quite the contrary. Queued processing has been a common transaction processing style and will continue to be very important in the future. Queued

processing is increasingly important for workflow, disconnected operation, and replication applications. It has always been the mainstay of batch and spool operations.

The controversial opinion here is that I believe queues are best built as a naive resource manager atop an object-relational database system. That system must have good concurrency control, recovery, triggers, and utilities -- indeed it must be a good TP-lite system.

## 2. Queues Are "Interesting" Databases

Storing queues in a conventional database has considerable appeal. The idea is that queues are encapsulated as a database class with create(), enqueue(), dequeue(), poll(), and destroy() methods. By using the database, the queue manager becomes a naive resource manger with no special code for startup, shutdown, checkpoint, or even commit. Rather it is just a simple application. In addition, it benefits from all the database utilities to query, backup, restore, reorganize, and replicate the data.

Queues pose difficult problems when implemented atop a database. An enqueue request transaction is an insert followed by a commit. This places extreme performance demands on the concurrency control and recovery components of a database -- it exposes hotspots and high-overhead code.

The dequeue transaction typically involves deleting a record from the queue, processing the request, enqueuing results in other queues, and then committing. Serializable isolation requires that there can be at most one dequeue executing at a time. This suggests that queues need lower, indeed specialized, isolation levels.

In Gray and Reuter [ibid. pp. 402] we outlined the concurrency control mechanisms needed to implement queues within a database:

1. READ-PAST locks allow a program to skip over dirty (uncommitted records) to find the first committed record. This is what a dequeue operation wants.
2. READ-THROUGH locks allow a program to examine records that have not yet been committed. This is useful in polling the status of a queued request that is currently being processed.
3. NOTIFY LOCKS (events) allow a program to wait for a state change in a lock. This allows a dequeue operation to wait for the queue to become non-empty.

Non-transactional queues are sometimes needed for performance reasons. The same reasons encourage us to support non-transactional tables in an SQL database. These tables and queues are not durable (do not survive system restart or media failure), but have low overhead.

The paradox is that queues are just an application data structure. Their concurrency control and recovery needs appear in many other contexts. An auction application looking for a set of sellers to match a buyer needs exactly these lock modes. One sees similar needs when looking workflow, CASE, and parallel programming models like Linda where an application wants any free tuple (read past locks and notify locks help here).

There is a pattern here. Each new requirement for a queuing system seems to reflect a corresponding requirement for user-application data. This recurs when one considers query interfaces to queues, queue performance monitoring, queue backup/restore/recovery utilities, queue security, and so on. Indeed, Richard Carr reports that when a queueing mechanisms was added to Tandem's database servers, several applications became simpler and faster.

## 3. Queue Managers Are Simple TP-Monitors

So far, the discussion has ignored the question of server pool management. Some queues have a server pool attached to them. TP-monitors configure, manage, and load-balance these pools.

Typically the pool is allocated with a minimum and maximum number of servers. At startup, the pool is configured at its minimum size. As traffic on the queue grows, the pool grows. As traffic shrinks, the pool shrinks. If a server fails, a new server is allocated. If too many servers fail in a time window, the TP-

monitor declares the queue broken and human intervention is required. Operator and programmatic interfaces are defined to create, configure, query, and control (start/stop/redefine) queues.

Queued processing has many variants:

- Periodic: Servers are created at certain times,
- Event: Servers are created on demand when a request first arrives in a queue.
- Batch: Servers are created when the queued grows to a certain size.

Queues have a FIFO scheduling policy by default, but it is often desirable to have a priority scheme whereby some queue elements are processed before others if the pool is saturated.

Gee! This sounds like a lot of stuff you do not find in a database system.

But... what about triggers? Modern database systems allow users to associate procedures with data records. These trigger procedures fire when records are inserted, deleted, or updated. Currently, the triggers fire at the time of the operation (immediate), or at the time of commit (deferred) -- and they execute within the ACID transaction of the operation that fired the trigger.

Trigger procedures are out-calls from the DBMS. Typically they are written in C, COBOL, FORTRAN or Visual Basic. In general they have to be executed in a protection domain separate from the requester and separate from the DBMS. Consequently, they are typically executed in a separate process (address space). Sybase's OpenServer design is typical of this idea -- although it uses a single multi-threaded process rather than having a separate protection domain per trigger. Oracle's Rdb uses a separate process to process outcalls.

Managing the trigger processes is a chore. For performance, they must be pre-allocated. There must be a load-control facility to prevent saturation. The pools must grow and shrink with demand. Gradually, the trigger-execution mechanism of the DBMS merges with the DBMS's TP-lite front-end dispatcher to make a fairly general TP-lite monitor. Indeed, the Sybase OpenServer started as a front-end, then became a side-end (trigger) and back-end (gateway) mechanism.

So DBMS systems are growing a server pool management system. This is part of the evolution of TP-lite to TP-heavy.

Not much is needed to add queued processing to a TP-lite DBMS. First one must implement the queues as an encapsulated type atop the Object-Relational system. Then one must recognize that triggers may be fired as part of a transaction, or fired asynchronously as a new ACID unit (either immediately, or if and when the transaction commits). This small conceptual leap and a simple matter of programming gives a simple queued transaction processing system. It should be as scaleable and robust as the underlying DBMS.

## 4. Summary

Many people are building queue managers from bare metal as a transactional resource manager and a TP-lite monitor. An alternative approach is to evolve Object-Relational systems to support the basic mechanisms needed to build a queuing system:

- reduced isolation levels and fine granularity locking
- efficient support for simple transactions.
- asynchronous trigger invocation executed by server pools
- management of server pools.

With these basic facilities, enable the implementation of queue managers but also make the DBMS more useful by other applications.

## 5. References

[ Bernstein, Hsu, & Mann] P.A. Bernstein , M. Hsu and B. Mann. "Implementing Recoverable Requests Using Queues." Proc. Proc. ACM SIGMOD. Atlantic City, NJ. 1990

[Gray & Reuter] J. Gray and A. Retuer.*Tranasaction Processing Concepts and Techniques*. Morgan Kaufmann, San Francisco, 1993.

# Drawing the Line between Consumers and Suppliers of Transactions in Object Oriented Application Development

August 25, 1995

Geoff Hambrick

IBM Object Services Technology Center
MS 9372
11400 Burnet Road
Austin, TX 78758

email: geoff@austin.ibm.com

## Problems with Source Code Reuse

Object technology has been touted as the means by which large scale reuse of maintainable fine-grained objects will become a reality. Unfortunately, reuse of this sort has yet to happen because most practitioners are still looking to inheritance and subclassing rather than exploiting encapsulation and separation of concerns.

For applications requiring object services such as transactions, separation of concerns means drawing an appropriate line between the business logic of the application developed by a domain expert and the platform specific logic of the transaction system developed by an expert in high performance transactions. Given that both experts employ object technology, the question then becomes when and how these two separate object models are integrated together.

Assuming a typical development cycle with analysis, design and code phases, the obvious answer seems to be at design time (the "when") using the source code (the "how"). That is, the transaction model can be viewed as a general pattern applied to each object in the business domain, resulting in a design model that gets coded and built in the programming language of choice.

This source code based approach has the advantage of faster execution time, since the transaction code is directly intergrated into the methods of the application objects. The direct integration also makes it relatively easy to "tweak" performance by modifying transaction boundaries, choosing read transactions over read-write where possible, etc. However, there are some serious disadvantages:

1. True encapsulation is broken, since the internals of the business objects and their behaviors are exposed to those who would add transactions at a later step.
2. Code complexity increases as each method must include the appropriate transactional boundary calls (begin, commit, rollback). This makes the resulting binary larger as well as making maintenance much harder (even given good tools).
3. Dependency on one particular transaction model is hard coded into the application, making it nearly impossible to reconfigure later if a new and/or more efficient implementation becomes available.

These problems are compounded when one takes into account other services like concurrency, persistence, replication, security, etc.

## Problems with Binary Code Reuse

Since the "obvious" answer is out, what about the other extreme? That is, assume that the two experts have shipped only binary objects and no source code. In this scenario, integration can occur anywhere from install to run time, which poses some interesting questions of its own:

- how to make appropriate boundary calls in objects that were not coded with transactions (or other services) in mind;
- how to get enough information about the essential state of the object or its behavior such that the most efficient commits and rollbacks can be made.

This paper proposes a programming model that answers both questions in a consistent, object-oriented manner.

## Making Appropriate Boundary Calls through a "Usage" Object

One approach to overcoming the first obstacle is to generalize the notion of boundary calls across services to "before/after" behaviors. That is, for each method invoked on an object, a given service may have code that must be run before the method, and may also have some that must run after it. For example, a persistence service may need to restore the internal state of the object prior to executing a method; it may store it afterward. A transaction service may issue a begin transaction before the method, and a commit or rollback afterwards (depending on the success of the method executed).

There may be "state dependent" aspects to executing the service efficiently as well. For example, a persistence service may only restore an object once and only store it at release, if and only if it has been changed during the methods run since the restore. As another example, most transaction services only register an object with the transaction context once. These examples not only require the ability to extend an object with arbitrary attributes (such as "dirty bits" and/or "current context"), but also the need to determine the type of method executed in terms of an abstract domain independent lifecycle on the associated object (e.g., "create", "read", "update", and "delete").

Sometimes there may be more than one approach to using the service that trades one resource for another, or provides for a higher or lower "quality of service". For example, a "flat" transaction service may begin a transactional boundary with some outermost call and only register other objects within this context, trading speed at the expense of partial rollbacks. A "nested" service could be provided by the same vendor as well, starting a new transaction with every method, allowing for partial rollbacks at the expense of speed.

The proposal is for the service provider to ship one or more "usage" objects that encapsulate this before/after behavior and associated state variables that are integrated into the business objects at install or run time (through either inheritance or aggregation). These usage objects can take the place of lengthy tutorials on how to code to the service interfaces. Shipping separate objects makes the service easier to use by the customer; they also allow the service provider to change any and all interfaces (except for the before/after calls), since services are transparent to the domain objects.

Well, almost transparent, since there are times when more than just domain independent information is needed about an object. For example, transaction services need to restore/save the state of the object (or save an "inverse" operation) prior to executing a method in order to handle a rollback. Enabling these kinds of services requires some work from the domain expert to expose the essential state and/or behavior of the business objects. The following details standard approaches for each.

## Exposing Essential State via Externalization

An approach to exposing external state borrows from OMG, OpenDoc and Taligent, which make heavy use of the notion of "externalization" behaviors that are associated with each object. In this model, an object has methods that either write its essential state to a "stream" or read state from it. Strictly speaking, this does not break encapsulation since the internal form of the object can be encrypted, rearranged, or indirectly accessed. The "stream schema" (order of data written/read) represents the contract between the business object provider and any arbitrary service provider. The service provider supplies the stream implementation tailored to the service.

Thus, a transaction service needing to save an image of the object for recovery would simply create a stream (either in memory or on disk depending on the quality of service), and use it as a parameter to an "externalize_to_stream" method. In case of a rollback, it would use the same stream and execute an "internalize_from_stream" method on the object. Other services, like persistence would work similarly: "internalizing" during a restore and "externalizing" during a store. In this manner, the domain expert supplies two methods to support any number of services that need access to state.

## Exposing Essential Behavior via Commands

The approach to supporting those services that rely on behavior rather than state, is to borrow again from Taligent and OpenDoc and use a lightweight "command" object that supports "do", "undo" and "redo" interfaces. The idea is for the domain expert to provide one for each method of a domain object (paying particular attention to the undo method since the do/redo can be generated automatically). Any service can create and use commands during the before/after methods described above for various purposes. For example, a transaction service might save a list of command objects representing the methods executed during the course of the transaction that are "undone" if a rollback occurs. A "disconnected" client service might save commands to be batched and executed on the server when connection is made.

Since a command is an object, it can support externalization methods that enable it to have services transparently attached using the same mechanisms that we are discussing here for arbitrary domain objects. Taken together, streams and commands provide a consistent mechanism for the domain expert to enable complete configurability without breaking encapsulation.

## A Concrete Example

To see the above proposed solutions in action, walking through a concrete example is in order. The domain chosen is a Travel Reservation System, with the following objects of interest:

- **Travel Service.** Represents a company that provides a given travel service. For example, Hertz rental cars or Hilton hotels. Basically serves as a "factory" for Reservations.
- **Travel Agency Member.** Represents the customer of the travel agency. Basically serves as a "collector" for Profiles and Trips.
- **Profile.** Represents the "preferences" that a member has for a given Travel Service. For example, non-smoking rooms, mid size cars, aisle seats, etc. This object is used by the Travel Service for default values when making Reservations.
- **Trip.** Represents a given trip planned or actually taken by the Member. Basically serves as a "collector" for Reservations, with extra attributes to record the "purpose", primary destination, etc.
- **Reservation.** Represents the need for a specific travel service, like a hotel, car, train, airline flight, etc. In general, these objects are "pinned" to the type of travel service, but serve to abstract out the common attributes needed for the travel agency to print itineraries, etc.

All the objects above should have externalization routines that read/write the essential attributes according to a published stream schema (IDL that describes the attribute types and names). Given that the nonderivable attributes of a Travel Agency Member were name, address, phone, fax, department, preferred credit card, list of profiles and list of trips, the domain expert can write/read them in any order and/or format desired (as long as she is consistent). For example, the credit card number might be written/read as a string, the lists as sequences of references, etc.

It is important for the domain expert to design the stream schemas for stability across various internal implementations, as once they are published, the format becomes a contract with users. Thus, strings and sequences of references were chosen as more generic than hash tables or linked lists (which may in fact be the internal form of the Profile/Trip lists described above for Travel Agency Members). Each of these forms is easily converted to/from a sequence (or sequences) during the externalization methods.

The Reservation class deserves some extra attention since it represents a "domain boundary crossing" from the travel agency's system to that of the travel service company (for example American Airlines' Sabre system). This crossing makes it likely that the functions associated with reservations (such as "confirm" and "cancel") will affect two disconnected systems that cannot work within the same transactional context (or any

other service for that matter). Therefore, the domain expert should at least provide command objects for those methods and supply an undo method for each to enable changes to be "backed out" consistently. Of course for the maximum configurability, the domain expert should provide command objects for every method.

In any event, undo methods on commands can be written in terms of other methods on the domain objects that in fact invoke compensating transactions on the underlying systems. For example, undo for the "Reservation_confirm" command could simply execute the "cancel" method on the associated reservation (stored as part of the command during the do method), which would issue a cancel reservation on the appropriate Travel Service company system. The undo for the "Reservation_cancel" command could simply execute the confirm method again.

Other than providing the externalization methods and command objects, the domain expert should strive to ignore any code associated with object services such as transactions, concurrency, persistence, security, etc. Code associated with these services is the concern of another expert.

## The Role of the Transaction Expert

So, how could a transaction expert make use of these domain objects with externalization methods, stream schemas, and commands to enable an efficient implementation? The answer depends on the assumptions made about the end user's configuration in terms of how tightly integrated transactions are with other services like persistence. For simplicity these configurations can be divided into four categories:

1. ***Objects with persistent data from procedural databases.*** For many years to come, it is probable that objects will "wrapper" data from existing procedural databases that are still accessed via external query engines, etc.. For this reason, any associated transactional usage will need to restore the persistent data during the before method from a stream that wrappers the database, and register the object to the transactional context. During a pre-prepare for commit, the data is then stored to the database, again via the database stream wrapper. A rollback need not do anything, since the object will be restored again anyway.
2. ***Objects with persistent data from object oriented databases.*** As objects become pervasive over time, object databases will come to implement a single-level-store model of both object and data as a single entity. The the transactional usage can then take advantage of the fact that no other applications can update the associated data and only restored the object during the first call (not on every transaction as above). The commit behavior is not affected; however, a rollback must at least reset a flag indicating that the object needs restoring on the next call (it could restore automatically if desired).
3. ***Object oriented transaction monitors.*** These are object "wrappers" around purely transactional interfaces (like the American Airlines Travel Service object does for the Sabre system in the example above). As such, there is no persistent data to store/restore, so commands with undo methods that execute the compensating transactions must be saved in case of a rollback. Nothing actually needs to be done in the case of a commit except reclaim the space taken by the saved commands (possibly in a stream).
4. ***Transient objects.*** This category does not necessarily imply that the domain objects lack persistent data within the system configuration, but that any persistence must be treated orthogonally to the transaction service. For example, a high availability server with transparent backing store could benefit from a usage object that creates a stream (either in memory or a file depending on the quality of service) to save the image of the object at the start of the transaction. If a rollback occurs, the object is internalized from the stream to restore its state. As is the case with the transaction monitor, there is no need to do anything during a commit operation (except reclaim the beforeimage stream).

The common theme of each of these approaches is that each one provides a specialized usage and stream optimized for the documented assumptions. Other usage/streams could be provided that mix and match these assumptions where they make sense (such as a "transient transaction monitor", where both a

beforeimage and compensating commands are saved). The fundamental advantage to this approach based on separation of concerns should be clear: the transaction service can be completely transparent to the domain objects without limiting the choice of implementation.

## Summary

Given that binary code is the only proven means to get large scale reuse, and that objects are the way to get fine-grained reuse with maintainability through encapsulation, we must enable separation of concerns between the domain expert and the service provider in order to get both.

The simple programming model outlined above has the domain expert encapsulate essential state and behavior of business objects through externalization methods (including a stream schema) and command objects; it has the implementation expert encapsulate the essential state and behavior needed by services through usage objects supporting before/after methods, and stream objects used by externalization methods on business objects. Integration tailored to specific performance and legacy constraints can then be achieved at install, start-up, or run-time as is desired by the application assembler with little or no coding through inheritance and/or aggregation.

Drawing the line between consumers and suppliers of transaction services, i.e., separation of concerns, allows the "right expert" to do the work in developing a complete commercial quality application.

# Transaction Processing on the Internet -- Revolution or Evolution?

James R. Hamilton, IBM Toronto <jrh@vnet.ibm.com>
Susan Malaika, IBM Hursley <malaika@vnet.ibm.com>
Patricia G. Selinger, IBM Almaden Research Center <pat@almaden.ibm.com>
Eugene Shekita, IBM Almaden Research Center <shekita@almaden.ibm.com>

## Abstract:

Commercial data processing over the Internet is rapidly evolving along a path similar to the one taken by intra-enterprise transaction processing systems in decades past. The two fundamental issues that still must be addressed are the statelessness of the current Web protocols today and issues related to security and authentication. This paper argues possible approaches to be taken and speculates on their success.

## Introduction:

The revolution turning the Internet into a mechanism for global commerce is in some ways similar to what occurred in commercial intra-enterprise data processing several decades ago. We anticipate, however, that the timescale of change will be significantly more rapid driven by what is quickly approaching universal access. The Internet in the 1970's linked computer science research sites funded by ARPA, and was used primarily for electronic mail and file transfer. During the 1980's, this usage expanded to remote server access (ordering research reports automatically, for example), and participation broadened to the general scientific community. Today the Internet has entered the home, with consumers and businesses as well as the scientific community communicating and participating actively. Usenet news group distribution and mail transfer fueled early growth. However, the emergence and widespread availability of World Wide Web [WWW] browsers and servers has led to explosive growth and has been one of the driving forces behind the spread of Internet usage into the home computing market. Web browsers are well on the way to becoming the Internet "universal client" in that they not only provide access to HyperText Markup Language (HTML) documents, but also RFC822 [RFC] mail, USENET news, Wide Area Information Server (WAIS), File Transfer Protocol (FTP), Gopher, and telnet. And, more importantly, Web browsers are typically installed or available for most client computing platforms. An emerging area of Web usage is electronic commerce with a few vendors such as Netscape (http://home.mcom.com) providing software supporting server authentication, encryption, a reliable channel, and optional client authentication.

It is illuminating to examine in more detail the evolution of intra-enterprise data processing. In the late 1950's, computer access was almost entirely local, with jobs submitted serially via punched cards or paper tapes. Then the concept of spooling was invented to improve batch throughput by setting up a background task to read off a slow input device while processing another job concurrently. The input from the background task was queued for the next job. A good example of early use is the SABRE system of American Airlines, which became operational in 1964, supporting electronic airline reservations from 1100 agent terminals submitting requests to a single complex with duplexed processors [JS81]. In August 1967, IBM announced Remote Job Entry (RJE) for the OS/360 operating system, allowing more sophisticated scheduling, queuing, and job control from remote terminals.

Having evolved from a single job to multiple jobs being queued, and then to remote job entry, early systems provided telecommunication access methods to manage this work. These were followed in the 1970's by transaction monitors, built using protocols such as SNA. In 1974, SNA [SUND87] was introduced for the DOS/VS operating system supporting communication networks of one computer connected to multiple terminals. Remote terminal support was added the following year, and, in 1977, SNA added the capability of reaching multiple hosts from a given terminal, followed by network management services, DES encryption, and parallel session support in 1979. During the 1980's SNA added option sets for transaction program security, for authenticating session partners, automatic reestablishment of sessions and resynchronization of user sessions upon link failures, classes of service, and two-phase commit. Despite the functional richness of SNA and many predictions that it would "emerge as [the] standard" [GRAY89] it faces strong competition due to the widespread use and availability of TCP/IP.

## Web protocols and current status:

Web browsers (the user agent) communicate over TCP/IP connections with Web servers using HyperText Transfer Protocol [HTTP]. For backwards compatibility and ease of use, most browsers are able to also support additional protocols: FTP, telnet, Wide Area Information Service (WAIS), Gopher in addition to being able to send RFC822 mail [RFC]. However, the native protocol for Web browsers is HTTP, which is used for accessing hypertext documents at the Web server. HTTP is a simple protocol supporting 7 main methods: get, head, text search, link, unlink, post, and put.

The servers, and the documents on these servers, are accessed via Uniform Resources Locator (URL). The URL "http://home.mcom.com/newsref/index.html" for example, refers to the Hypertext Markup language (HTML) document "index.html" on the server "home.mcom.com" in directory "/newsref" to be accessed via the HTTP protcol. Similarly the URL "ftp://ftp2.netscape.com/1.1b3" refers to the file netscape1.1b3 accessed via ftp from the ftp server ftp2.netscape.com to be found in the default (root) ftp directory. Web browsers primarily access HyperText Markup Language documents (HTML). HTML is an application of the SGML ISO Standard (SGML ISO Standard 8879:1986 Information Processing Text and Office Systems, Standard Generalized Markup Language).

The HyperText Transfer Protocol is currently a draft of the Internet Engineering Task Force [IETF]. HTTP/1.0 was made widely available in November of 1994. HTTP uses Multipurpose Internet Mail Extensions (MIME---Internet RFC 1342) headers to describe the format of the data being sent. Among the defined MIME standard headers are descriptions for HTML documents, GIF images, PostScript documents, JPEG images, MPEG movies, and many others [GRAH95]. Most Web browsers support some of these formats directly and also allow the user to register other viewers and players to extend the document format support.
HTML describe the logical organization of a document (a Web browser chooses the presentation format supported by the client system for the display of this document) and HTML supports hyperlinks. These hyperlinks are embedded URLs, essentially pointers, that can be followed by the Web browser to the next HTML document.

Web servers support the Common Gateway Interface [CGI] specification written by Rob McCool at the NCSA. The CGI allows an external program written in any language to be queried by a Web Browser. The CGI specifies the mechanism by which the query is communicated to the external program. The expected output is an HTML document to be displayed at the Web client. This interface allows existing database and transaction processing systems to be easily integrated into a Web environment. CGI programs are generally available supporting Web queries against Oracle, DB2, and Sybase amongst others.

## Premise: the revolution of commercial data processing on the Internet:

Given where it is today, what will it take for the Web to support commercial data and transaction processing? Historical evidence from the intra-enterprise transaction processing world strongly suggests that to be successful, some of the Web paradigms and protocols must change fundamentally. Our premise is that two fundamental problems must be addressed before the Web can evolve to the point where it can be used for real commercial transaction processing: 1) the current stateless HTTP must evolve to retain state, and 2) there must be mechanisms for authentication and security. Whether these changes take place and, if they do, how successful they will be at addressing the problems, remains controversial.

## Context management and remote agent surrogate support at server:

To make our case on these two issues, it's useful to contrast the capabilities of a transaction monitor, specifically CICS [CICS95], with Internet interactions over the World Wide Web [WWW]. One difference is that Web servers retain no state, and have no continuing connection to a client across invocations, while conversational use of CICS provides both. The concept of "state" is fundamental, as it establishes context for an active, already authenticated agent to hold resources at a remote server. If we disregard the concept of "state" for a moment, Web servers and CICS have many similarities. Both CICS and Web servers schedule and process short repetitive pieces of work on behalf of many users accessing shared data. And both CICS and Web interactions have a similar application model: when the application is running on the server, the client is synchronously dedicated to waiting for the response in order to present it to the user. Web programs can provide the

illusion of state at the server by transferring context information (user identification, the page the user was last accessing, etc.) via a hidden element in the HTML form passed with every server interaction.

While these programming tricks can provide the illusion of state, they do not provide the reality. This approach would prevent servers from holding resources between interactions, such as locks or authentication information, memory, or transaction records in the prepared state of two phase commit. It would also make it impossible to establish and maintain a connection while interacting with a given server. Either Web protocols must step up to supporting the concept of state, with notification and cleanup at servers upon failure, or data processing capabilities will be forever limited.

Without state, a fundamental problem remains: it is impossible to know whether a transaction will ever be completed or was just left unfinished (A user may choose never to finish the interaction). This is fundamentally incompatible with any concurrency and recovery system that holds transaction duration locks. The solution may be optimistic concurrency control systems (OCC) [KUNG81]. Even though they have not been extensively used in existing commercial systems, these protocols are ideally suited to this environment where holding locks is impractical.

Another alternative might be simply to place on the user the burden of verifying the completion state of a transaction. For example, a user would tacitly commit to a transaction by clicking the "send" button on an order screen. To avoid holding locks for unpredictable time periods, the order would be processed at the server after receipt of the user's "send". Under normal circumstances, the server would respond with an acknowledgement and probably a digital sales slip of some sort. If the network times out, or if there is a crash at either end of the connection, however, then it would be up to the user to verify whether that order was actually processed. This could be accomplished by having the user call up a "status" screen on the server after being reconnected. The status screen would allow the user to check on the order status. While this scheme is hardly user-friendly and probably not an attractive long-term solution, it has the advantage of simplicity. Further complications arise if there are multiple servers, the possibility of legitimate duplicate orders; in those cases unique application-specific tags retained at the user system will be required.

It's also interesting to draw parallels between the WWW evolution we anticipate and the one that has already taken place in shared file systems. For example, the Sun NFS protocol is stateless, fairly easy to program but a relatively poor performer. Although the argument in favour of statelessness is based on increased server performance, it is exactly this feature that prevents many very effective performance optimizations (e.g. client side cache and one time authentication check). The AFS system, is a much better performer, in that it is stateful and supports the maintenance of a client side cache. We speculate that Web servers will need to maintain state (and connections) to get the performance and robustness required by high volume commercial OLTP systems. In the interim, OCC may be a partial answer.

## Authentication and Security:

The other significant problem area for the evolution of Web protocols to perform commercial data processing transactions is the requirement to have secure communications with a known, authenticated client. Today, and for the foreseeable future, the Web client is an insecure environment, and Web communications are subject to malicious modification and monitoring by an adversary.

To support commercial transaction processing, the following security issues must be addressed: 1) support client authentication, 2) support server authentication, 3) avoid monitoring and theft of information, 4) support replay protection, 5) prevent modification, and 6) implement once-only semantics.

The Pretty Good Privacy mail system [PGP95] is a good model in that, as a mail system operating over insecure communications links, it must address many of these same issues. PGP uses the IDEA [PGP95] private key encryption system to protect against monitoring and information theft, it uses the RSA [PGP95] for both sender (client) and recipient (server) authentication, and the MD5 message digest system [PGP95] to prevent modification.

Netscape Communications Corp. has implemented a secure web protocol called HTTPS (HypterText Transfer Protocol--Secure) in their Netscape Commerce Server (http://home.mcom.com/comprod/netscape_commerce.html). HTTPS is implemented on top of the Secure Sockets Layer (http://home.mcom.com/ newsref/std/SSL.html) which, like PGP, is implemented using the RSA public key encryption system and MD5. Rather than using only the IDEA private key

encryption system as PGP does, SSL supports RC2, RC4, DES (Data Encryption Standard), in addition to IDEA. Both 40 bit and 128 bit RC4 keys are supported (128 bit is far more secure but RC4 with 40 bit keys are legal for US export whereas most of the other algorithms still carry export restrictions).

Netscape Communications Corp. has also proposed an electronic payment protocol for Internet credit-card applications (http://home.netscape.com/newsref/std/credit.html). The proposal is for a three-party protocol, involving a Customer, Merchant, and Payment Gateway (representing the acquiring bank). The protocol is layered on top of Netscape's Secure Sockets Layer. The Secure Sockets Layer provides connection security, while the electronic payment protocol provides signature, non-repudiation, and secondary encryption. Secondary encryption is based on public keys and is used to conceal the customer's credit-card number from the merchant, generate digital signatures, and so on.

IBM has also proposed a secure electronic payment protocol for the Internet called iKP (http://w3.zurich.ibm.com/g/kk/www/extern/ecommerce). iKP is a family of protocols that implements credit-card-based transactions between the customer and merchant while using the existing financial network for approval and clearing. Like the Netscape proposal, iKP is a three-party protocol, involving a Customer, Merchant, and Payment Gateway. Public-key encryption is used to protect confidential information such as credit-card numbers and for signing authorization messages. iKP has been designed to provide strong encryption while still satisfying import/export restrictions.

Other approaches to Web security can be found on http://www-ns.rutgers.edu/ www-security/drafts.html. Some examples are "Shen: A Security Scheme for the World Wide Web" from CERN, and "Secure HyperText Transfer Protocol (SHTTP) proposal" from Enterprise Integration Technologies.

We believe that the technology exists to properly secure Web communications and, in many cases, implementations already exist. What is missing is an agreed upon, widely implemented standard (ideally, one free of US export restrictions) and a widely used, reliable encryption key distribution system (public keys must be verified as valid, then published and made broadly available).

## Other issues:

If you accept our premise that the Internet is rapidly evolving to support the same model of commercial transaction processing as we have today within the enterprise, and given that the two significant issues needed to complete that evolution are resolved, there remain some concerns:

1) session management and naming. Communications between web clients and servers have no notion of session state on the target system, and names of servers are associated with their addresses, preventing transparency for modular growth or failure redundancy.

2) guaranteed invocation, with restart on failure. CICS provides hardened messages, which guarantees delivery of messages and execution of transactions.

3) ACID enforcement.

4) mapping services from server back to clients. We anticipate that there will be extensions to help with commercial Web applications, e.g, to ease the integration of business data held on relational databases with multimedia on the Web. There are extensions already available to enable programming on the client, e.g. HotJava from Sun at http://webrunner.neato.org

5) load balancing. In many cases, a single server will not be able to handle the demand placed on a popular WWW site. A potential solution to this problem is described in http://www.ncsa.uiuc.edu/InformationServers/. The basic idea is to rotate through a pool of servers, each of which is alternately mapped to the hostname alias of the WWW server.

## Conclusion:

We predict that commercial data processing on the Web will follow the evolutionary path of traditional, intra-enterprise technology. However, this evolution will be much compressed, nearly a revolution, driven by near universal access, and will require innovative solutions to traditional problems.

## References:


[CICS95] GC33-0155 CICS General Information Manual.

[CGI] Common Gateway Interface. For more information see: http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[GARF95] Simson Garfinkel, "PGP: Pretty Good Privacy", O'Reilly & Associates, 1995.

[GRAH95] Graham, Ian S., "HTML Source Book: A Complete Guide to HTML", John Wiley & Son, Inc., 1995.

[GRAY89] Gray, Jim., "Transparency in its Place -- The Case Against Transparent Access to Geographically Distributed Data", Tandem Tech. Report TR 89.1.

[HTTP] More information on HTTP is available from http://info.cern.ch/hypertext/WWW/Protocols/Overview.html.

[IBM95] CICS Family: Interproduct Communication, IBM SC33-0824 [IETF] More information on the Internet Engineering Task Force (IETF) can be obtained from http://www.ietf.cnri.reston.va.us. [KUNG81] H. Kung and J. Robinson, "On Optimistic Methods for Concurrency Control", ACM Transactions on Database Systems, Association of Computing Machinery, 1981.

[RFC] All Internet Engineering Task Force Request for Comments (RFC's) can be viewed or obtained from http://ds.internic.net/ds/dspg1intdoc.html.

[SUND87] Sundstrom, R.J. et al. "SNA: Current requirements and directions," IBM Systems Journal. Vol 26, No.1, 1987. [JS81] Jarema, D.R., and Sussenguth, E.H. "IBM Data Communications: A Quarter Century of Evolution and Progress," IBM Journal of Research and Development. Vol. 25, No. 5, September, 1981.

[WWW] http://www.w3.org/hypertext/www/TheProject

# Netscape Secure Courier

## A Presentation to HPTS

Michael Higgins

Overview and Goals:

The goal of this presentation is the increase the awareness and understudying of the listeners on the topics of: *Commerce on the Internet*, and *Secure Financial Transaction Processing on the Internet,* as the apply to the HTPS framework.

Secure Courier is a Specification and a Product offered by Netscape Communications Corp. in conjunction with its partners, which include: Intuit and MasterCard.

Secure Courier is a Transaction based protocol for doing very secure financial transactions over a public TCP/IP network, specifically the Internet. The protocol involves moving data between at least three players in a financial transaction: the consumer, the merchant, and the credit card processor (sometimes called the acquirer).

The outline below contains the slide topics to be covered in the presentation.

# Netscape Secure Courier

## A Presentation to HPTS

- Internet Overview
  - Brief History
  - Size: Users, Computers, Nations, etc.
  - Typical Usage Models
- Netscape Communications Corp.
  - History
  - Charter
  - Product Overview
- Commerce on the Internet
  - Revenue Today
  - Advantages
  - Problems
  - Security Issues
- Browsing vs. Transaction
  - Session-less Model
  - HTTP vs. HTML
  - Client (Browser) State
- Secure Courier
  - Problem Definition
  - Architecture
  - Cryptography
  - Transactions
  - Gateway Model
- Summary
  - Performance
  - Partnerships
  - Q & A

# Enterprise Transaction Processing on Windows NT

Greg Hope, Chief Architect, Paul Oeuvray, VP Systems Design,
Paul Miniato, Senior Consultant
Prologic Corporation
100-3851 Shell Road
Richmond, B.C. V6X 2W2 Canada
Phone: (604) 278-6470 Fax: (604) 278-5206
Internet: hope@prologic.ca

*"We sit here and look at the world's most demanding transaction-processor-based systems, and we ask ourselves 'What does it take to make PCs or networks of PCs suitable for those tasks?'"*
—Bill Gates, *Fortune*, January 16/95.

## Migrating to Client/Server Solutions

For over 10 years Prologic has been replacing mainframe and mini-computer based enterprise-wide transaction processing systems with distributed networks of PCs.

## Road to Success

Prologic's customer base includes over 100 installations of line-of-business systems in banking, insurance and inventory control. With the development of PROBE® for Windows NT™, our new partnerships include high profile transaction-processing environments at EDS, Dallas, Texas and Nomura Research Institute, Yokohama, Japan.

Back in 1987, Prologic's first customer, Richmond Savings of Vancouver, B.C., used Prologic's PROBE for DOS-based technology in production. This award-winning application (Best in MicroBanking, 1988) is a complete retail banking system, including a consolidated customer information file, deposits, loans, GL, and ATM, EFT/POS, VRU.

## Re-Engineering Pay Off

Richmond Savings has over 400 users in 12 locations performing approximately 150,000 online transactions per day; month-end batch transaction volumes exceed 300,000; their 20 million row database is located on their enterprise server: a 486/66 with 16MB of RAM and 3GB of disk using MS-DOS and Netware. End users experience an average 1.5 second response time over low bandwidth 9600 BPS WAN circuits with 99.98% availability. Richmond Savings' new business system has paid off—since 1987 it has had an industry leading 16% compound growth rate.

## Key Design Goals

Prologic established some key criteria early on for PROBE. These goals include: 1) simplified application development and maintenance through a *unified*, homogenous platform and tools for the entire solution including client, server, gateway, and batch components; 2) high *performance*, scalable transaction-throughput utilizing the lowest available life-time operating cost platform and efficient communications; and 3) increased developer *productivity* to improve responsiveness to business change, and reduce cost and risk of new application development.

## Development Infrastructure

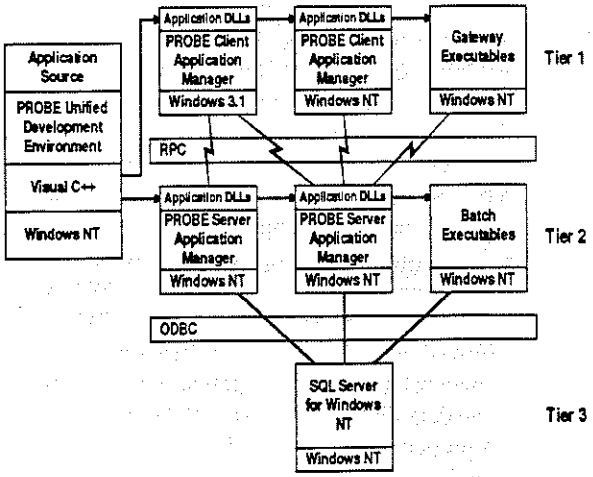
Applications are developed using PROBE's Unified Development Environment (UDE) and deployed using PROBE's Unified Application Framework (UAF). The application is composed of the data dictionary where all database objects and properties are specified using SQL DLL extended to include features such as pre-defined relationships, time relationships, and visual properties of data objects. Compiling the data dictionary automatically generates the complete Windows user interface and SQL database catalog. The Windows interface includes a complete MFC MDI application including forms, menus, and data entry and retrieval functions. The SQL database catalog is automatically synchronized via ODBC including conversion of existing data, and

generation of all indexes and stored procedures required for high performance transaction processing data access.

Application business rules are encapsulated in event procedures corresponding to the location, object, and event that fires the business rule using a full-function procedural logic language based on extended C and SQL. The application function is automatically compiled into high-performance 32-bit DLLs using Microsoft Visual C++. With this development environment, developers can see dramatic increases in productivity. In one case, a California financial firm replaced a CICS/COBOL application of more than 2 million lines of code with a PROBE application of less than 100 thousand lines.

Once written, the application is deployed with PROBE's UAF to deliver a three-tiered, distributed-function, message-shipping, enterprise-wide transaction processing architecture on local and wide area networks. PROBE operates efficiently over 9600-BPS WAN bandwidth, reducing communications costs, improving response time, and providing greater scalability.

### The Three-Tiered Answer




The three-tier client/server architecture consists of:

- Tier 1, the Client Application Manager (CAM) coordinates the WIN32s presentation functions, data access, update operations, and client application dynamic-link libraries (DLLs).
- Tier 2, the Server Application Manager (SAM), a Win32 multiprocessing, multithreaded program that manages transactions, data, and server application DLLs, processes messages from the CAM received via NetBIOS, NetBEUI, IPX, or TPC/IP.
- Tier 3, the SQL database performs database management system functions requested by the SAM via a 32-bit ODBC driver.

### Client Services

CAM provides a complete user interface including a visual service and a non-visual service. The visual service supports object and task user interface models, workflow, views, security, navigation, forms, field input, printing, menu and toolbar management, keyboard and mouse management, status bar, context-sensitive help, clipboard functions, resource management, and OLE functions, including linking, embedding, drag and drop, and in-place activation. The non-visual service provides network session management, data entry and retrieval, field binding, event notification, error handling, and interfaces to 3rd party visual services via OLE automation and APIs.

### Server Services

SAM includes the Communications Manager (CM), the Transaction Manager (TM), the Temporal Data Manager (TDM), and the ODBC Optimizer (ODBC-O). The CM provides scalable transaction throughput with session management, context free operation, connection multiplexing, heuristic load balancing, transaction time-out, and priority queuing. The TM provides robust and scalable processing with features such as transaction bracketing, logging and recovery, nested transactions, lock initiation, deadlock/livelock resolution, commit, savepoints, transaction logging, exception handling, and event logging. The ODBC-O provides high performance through optimized set and ISAM access, catalogue and database synchronization, data conversion, automatic usage of optimized stored procedures, and metadata suppression. The TDM provides high quality and functional applications through automatic management of time ordered relationships between tables, including: automatic audit trail of master changes, temporal integrity which guarantees the "sum" of transactions equals the master, "asof" past and future projections,

automatic recalculation of master values on non-destructive error correction.

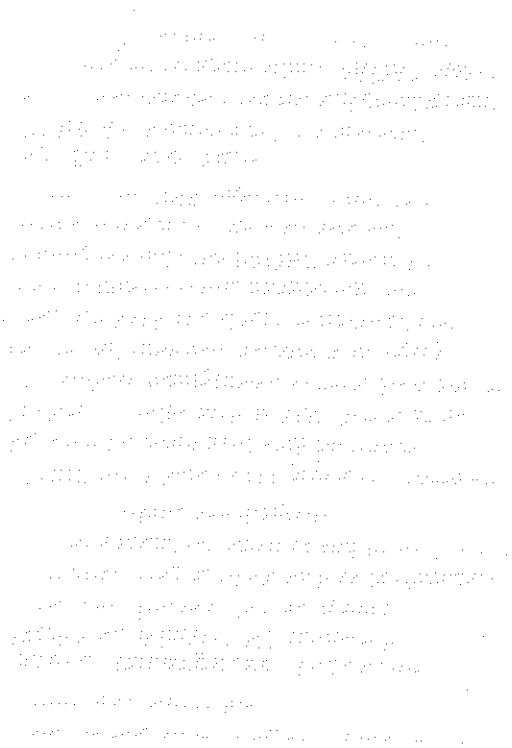
**Complete Enterprise Solution**

PROBE for Windows NT completely integrates with Windows NT for system administration using services such as Performance Monitor for transaction statistics and Event Viewer for problem tracking and diagnosis.

PROBE for Windows NT provides a complete, unified solution when used with Microsoft BackOffice facilities such as SQL Server as the ODBC database management system, SNA Server to provide real-time data transfer with legacy systems, and SMS as a single, centralized control point for troubleshooting, monitoring, and distributing not only the PROBE transaction processing application, but also personal productivity and DSS applications and data.

**Ready for the Task**

PROBE for Windows NT is a powerful transaction management system and development tool for client/server applications. PROBE makes networks of PCs suitable for the world's most demanding transaction-processing systems.



# Charting The Seas Of Information Technology

# CHAOS

Jim Johnson

THE STANDISH GROUP

*The Roman bridges of antiquity were very inefficient structures. By modern standards, they used too much stone and, as a result, far too much labor to build. Over the years we have learned to build bridges more efficiently, using fewer materials and less labor to perform the same task.*

Tom Clancy: *The Sum of All Fears*

In 1986, Alfred Spector, president of Transarc Corporation, co-authored a paper comparing bridge building to software development. The premise: Bridges are normally built on-time, on-budget, and do not fall down. On the other hand, software never comes in on-time or on-budget. In addition, it always breaks down. (Nevertheless, bridge building did not always have such a stellar record. Many bridge building projects overshot their estimates, time frames, and some even fell down.)

One of the biggest reasons bridges come in on-time, on-budget and do not fall down is because of the extreme detail of design. The design is frozen and the contractor has little flexibility in changing the specifications. However, in today's fast moving business environment, a frozen design does not accommodate changes in the business practices. Therefore a more flexible model must be used. This could be and has been used as a rationale for development failure.

But there is another difference between software failures and bridge failures, beside 3,000 years of experience. When a bridge falls down, it is investigated and a report is written on the cause of the failure. This is not so in the computer industry where failures are covered up, ignored, and/or rationalized. As a result, we keep making the same mistakes over and over again.

Consequently the focus of this latest research project at The Standish Group has been to identify:

- The scope of software project failures
- The major factors that cause software projects to fail
- The key ingredients that can reduce project failures

## FAILURE RECORD

In the United States, we spend more than $250 billion each year on IT application development of approximately 175,000 projects. The average cost of a development project for a large company is $2,322,000; for a medium company, it is $1,331,000; and for a small company, it is $434,000. A great many of these projects will fail. Software development projects are in chaos, and we can no longer imitate the three monkeys -- hear no failures, see no failures, speak no failures.

The Standish Group research shows a staggering 31.1% of projects will be cancelled before they ever get completed. Further results indicate 52.7% of projects will cost 189% of their original estimates. The cost of these failures and overruns are just the tip of the proverbial iceberg. The lost opportunity costs are not measurable, but could easily be in the trillions of dollars. One just has to look to the City of Denver to realize the extent of this problem. The failure to produce reliable software to handle baggage at the new Denver airport is costing the city $1.1 million per day.

Based on this research, The Standish Group estimates that in 1995 American companies and government agencies will spend $81 billion for cancelled software projects. These same organizations will pay an additional $59 billion for software projects that will be completed, but will exceed their original time estimates. Risk is always a factor when pushing the technology envelope, but many of these projects were as mundane as a drivers license database, a new accounting package, or an order entry system.

On the success side, the average is only 16.2% for software projects that are completed on-time and on-budget. In the larger companies, the news is even worse: only 9% of their projects come in on-time and on-budget. And, even when these projects are completed, many are no more than a mere shadow of their original specification requirements. Projects completed by the largest American companies have only approximately 42% of the originally-proposed features and functions. Smaller companies do much better. A total of 78.4% of their software projects will get deployed with at least 74.2% of their original features and functions.

This data may seem disheartening, and in fact, 48% of the IT executives in our research sample feel that there are more failures currently than just five years ago. The good news is that over 50% feel there are fewer or the same number of failures today than there were five and ten years ago.

## METHODOLOGY

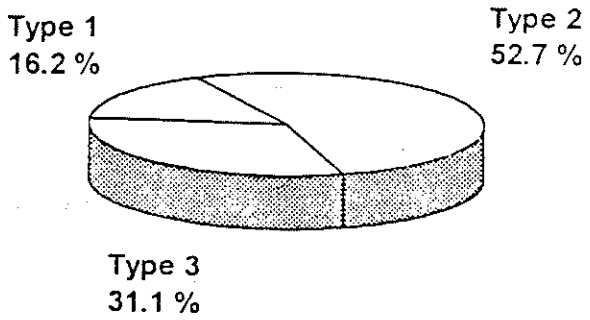
The survey made by The Standish Group was as thorough as possible, short of the unreachable goal of surveying every company with MIS in the country. The results are based on what we at The Standish Group define as "key findings" from our research surveys and several personal interviews. The respondents were IT executive managers. The sample included large, medium, and small companies across major industry segments, e.g., banking, securities, manufacturing, retail, wholesale, heath care, insurance, services, and local, state, and federal organizations. The total sample size was 365 respondents and represented 8,380 applications. In addition, The Standish Group conducted four focus groups and numerous personal interviews to provide qualitative context for the survey results.

For purposes of the study, projects were classified into three resolution types:

- Resolution Type 1, or project success: The project is completed on-time and on-budget, with all features and functions as initially specified.
- Resolution Type 2, or project challenged: The project is completed and operational but over-budget, over the time estimate, and offers fewer features and functions than originally specified.
- Resolution Type 3, or project impaired: The project is cancelled at some point during the development cycle.

Overall, the success rate was only 16.2%, while challenged projects accounted for 52.7%, and impaired (cancelled) for 31.1%.

## PROJECT RESOLUTION BY TYPE



FAILURE STATISTICS

The Standish Group further segmented these results by large, medium and small companies. A large company is any company with greater than $500 million dollars in revenue per year, a medium company is defined as having $200 million to $500 million in yearly revenue, and a small company is from $100 million to $200 million.

The figures for failure were equally disheartening in companies of all sizes. Only 9% of projects in large companies were successful. At 16.2% and 28% respectively, medium and small companies were somewhat more successful. A whopping 61.5% of all large company projects were challenged (Resolution Type 2) compared to 46.7% for medium companies and 28% for small companies. The most projects, 37.1%, were impaired and subsequently cancelled (Resolution Type 3) in medium companies, compared to 29.5% in large companies and 21.6% in small companies.

Restarts

One of the major causes of both cost and time overruns is restarts. For every 100 projects that start, there are 94 restarts. This does not mean that 94 of 100 will have one restart, some projects can have several restarts. For example, the California Department of Motor Vehicles project, a failure scenario summarized later in this article, had many restarts.

Cost Overruns

Equally telling were the results for cost overruns, time overruns, and failure of the applications to provide expected features. For combined Type 2 and Type 3 projects, almost a third experienced cost overruns of 150 to 200%. The average across all companies is 189% of the original cost estimate. The average cost overrun is 178% for large companies, 182% for medium companies, and 214% for small companies.

| COST OVERRUNS | % OF RESPONSES |
|---|---|
| Under 20% | 15.5% |
| 21 - 50% | 31.5% |
| 51 - 100% | 29.6% |
| 101 - 200% | 10.2% |
| 201 - 400% | 8.8% |
| Over 400% | 4.4% |

Time Overruns

For the same combined challenged and impaired projects, over one-third also experienced time overruns of 200 to 300%. The average overrun is 222% of the original time estimate. For large companies,

the average is 230%; for medium companies, the average is 202%; and for small companies, the average is 239%.

| TIME OVERRUNS | % OF RESPONSES |
|---|---|
| Under 20% | 13.9% |
| 21 - 50% | 18.3% |
| 51 - 100% | 20.0% |
| 101 - 200% | 35.5% |
| 201 - 400% | 11.2% |
| Over 400% | 1.1% |

Content Deficiencies

For challenged projects, more than a quarter were completed with only 25% to 49% of originally-specified features and functions. On average, only 61% of originally specified features and functions were available on these projects. Large companies have the worst record with only 42% of the features and functions in the end product. For medium companies, the percentage is 65%. And for small companies, the percentage is 74%.

| % OF FEATURES/FUNCTIONS | % OF RESPONSES |
|---|---|
| Less Than 25% | 4.6% |
| 25 - 49% | 27.2% |
| 50 - 74% | 21.8% |
| 75 - 99% | 39.1% |
| 100% | 7.3% |

Currently, the 365 companies have a combined 3,682 applications under development. Only 431 or 12% of these projects are on-time and on-budget.

SUCCESS/FAILURE PROFILES

The most important aspect of the research is discovering why projects fail. To do this, The Standish Group surveyed IT executive managers for their opinions about why projects succeed. The three major reasons that a project will *succeed* are user involvement, executive management support, and a clear statement of requirements. There are other success criteria, but with these three elements in place, the chances of success are much greater. Without them, chance of failure increases dramatically.

| PROJECT SUCCESS FACTORS | % OF RESPONSES |
|---|---|
| 1. User Involvement | 15.9% |
| 2. Executive Management Support | 13.9% |
| 3. Clear Statement of Requirements | 13.0% |
| 4. Proper Planning | 9.6% |
| 5. Realistic Expectations | 8.2% |
| 6. Smaller Project Milestones | 7.7% |
| 7. Competent Staff | 7.2% |
| 8. Ownership | 5.3% |
| 9. Clear Vision & Objectives | 2.9% |
| 10. Hard-Working, Focused Staff | 2.4% |
| Other | 13.9% |

**The survey participants were also asked about the factors that cause projects to be challenged.**

| PROJECT CHALLENGED FACTORS | % OF RESPONSES |
|---|---|
| 1. Lack of User Input | 12.8% |
| 2. Incomplete Requirements & Specifications | 12.3% |
| 3. Changing Requirements & Specifications | 11.8% |
| 4. Lack of Executive Support | 7.5% |
| 5. Technology Incompetence | 7.0% |
| 6. Lack of Resources | 6.4% |
| 7. Unrealistic Expectations | 5.9% |
| 8. Unclear Objectives | 5.3% |
| 9. Unrealistic Time Frames | 4.3% |
| 10. New Technology | 3.7% |
| Other | 23.0% |

**Opinions about why projects are impaired and ultimately cancelled ranked incomplete requirements and lack of user involvement at the top of the list.**

| PROJECT IMPAIRED FACTORS | % OF RESPONSES |
|---|---|
| 1. Incomplete Requirements | 13.1% |
| 2. Lack of User Involvement | 12.4% |
| 3. Lack of Resources | 10.6% |
| 4. Unrealistic Expectations | 9.9% |
| 5. Lack of Executive Support | 9.3% |
| 6. Changing Requirements & Specifications | 8.7% |
| 7. Lack of Planning | 8.1% |
| 8. Didn't Need It Any Longer | 7.5% |
| 9. Lack of IT Management | 6.2% |
| 10. Technology Illiteracy | 4.3% |
| Other | 9.9% |

**Another key finding of the survey is that a high percentage of executive managers believe that there are more project failures now than five years ago and ten years ago. This despite the fact that technology has had time to mature.**

| | Than 5 Years Ago | Than 10 Years Ago |
|---|---|---|
| Significantly More Failures | 27% | 17% |
| Somewhat More Failures | 21% | 29% |
| No Change | 11% | 23% |
| Somewhat Fewer Failures | 19% | 23% |
| Significantly Fewer Failures | 22% | 8% |

FOCUS GROUPS

**To augment the survey results, The Standish Group conducted four focus groups with IT executives of major companies. The attendees were from a cross section of industries, including insurance, state**

and federal government, retail, banking, securities, manufacturing and service. Two of the focus groups were in Boston. The other two, in San Francisco. Each focus group had an average of ten participants with an overall total of forty-one IT executives. The purpose of these particular focus groups was to solicit opinions on why projects fail. In addition, The Standish Group conducted interviews with various IT managers. Some of their comments are enlightening about the variety of problems besetting project development.

Many of the comments echoed the findings of The Standish Group survey. "We have 500 projects. None are on-time and on-budget. This year, 40% will get cancelled," said Edward, Vice President of MIS at a pharmaceutical company.

Other comments went directly to the reasons for failure. Jim, the Director of IT at a major medical equipment manufacturer, said: "Being that it's a mindset, it's very difficult to get all of the management -- it's even on the local level, not even on a worldwide level -- to get all of the management to agree on a set of rules.... That's a challenge in itself because you have to, in some cases, convince them that this is best for the company, not necessarily best for them, but best for the company. And you have to have that buy-in. If you don't have that buy-in, you're going to fail. I don't care how big or how small the project is."

John, Director of MIS at a government agency added: "Probably 90% of application project failure is due to politics!" And Kathy, a programmer at a telecommunication company, offered an even more scathing comment on politics: "Sometimes you have to make a decision you don't like. Even against your own nature. You say well, it's wrong, but you make that decision anyway. It's like taking a hammer to your toe. It hurts."

Bob, the Director of MIS at a hospital, commented on external factors contributing to project failure. "Our biggest problem is competing priorities," he said. "We just had a reorganization today. So now that's going to sap all the resources. And explaining to senior management that, 'Well, it's really taking us the time we said it was going to take. But because you've reorganized the company, I'm going to take another six months on this other project, because I'm doing something else for you.' That's the biggest issue I have." Bill, the Director of MIS at a securities firm, added: "Changes, changes, changes; they're the real killers."

Some of the comments were darkly humorous. "Brain-dead users, just plain brain-dead users," said Peter, an application analyst at a bank. "When the projected started to fail," said Paul, a programmer at a personal products manufacturer, "the management got behind it -- way behind."

The comment most indicative of the chaos in project development came from Sid, a project manager at an insurance company. "The project was two years late and three years in development," he said. "We had thirty people on the project. We delivered an application the user didn't need. They had stopped selling the product over a year before."

## CASE STUDIES

For further insight into failure and success, The Standish Group looked carefully at two famous Resolution Type 3 (cancelled) projects and two Resolution Type 1 (successful) projects. For purposes of comparison, the project success criteria from the survey of IT executive managers was used to create a "success potential" chart. The success criteria were then weighted, based on the input from the surveyed IT managers. The most important criterion, "user involvement," was given 19 "success points". The least important -- "hard-working, focused staff" -- was given three points. Two very important success criteria -- "realistic expectations" and "smaller project milestones" -- were weighted at ten and nine points respectively. Finally, as presented later in this report, each of the case studies was graded.

### California DMV

In 1987, the California Department of Motor Vehicles (DMV) embarked on a major project to revitalize their drivers license and registration application process. By 1993, after $45 million dollars had already been spent, the project was cancelled.

According to a special report issued by DMV, the primary reason for redeveloping this application was the adoption new technology. They publicly stated: "The specific objective of the 1987 project was to use modern technology to support the DMV mission and sustain its growth by strategically positioning the DMV data processing environment to rapidly respond to change." Also, according to the DMV special report "The phasing was changed several times, but the DMV technical community was never truly confident in its viability...."

The project had no monetary payback, was not supported by executive management, had no user involvement, had poor planning,

poor design specifications and unclear objectives. It also did not have the support of the state's information management staff.

The DMV project was not rocket science. There are much harder applications than driver licenses and registrations. But because of internal state politics, unclear objectives, and poor planning, the project was doomed from the start.

American Airlines

Early in 1994, American Airlines settled their lawsuit with Budget Rent-A-Car, Marriott Corp. and Hilton Hotels after the $165 million CONFIRM car rental and hotel reservation system project collapsed into chaos.

This project failed because there were too many cooks and the soup spoiled. Executive management not only supported the project, they were active project managers. Of course, for a project this size to fail, it must have had many flaws. Other major causes included an incomplete statement of requirements, lack of user involvement, and constant changing of requirements and specifications.

Hyatt Hotels

While Marriott and Hilton Hotels were checking out of their failed reservation system, Hyatt was checking in. Today, you can dial from a cellular airplane telephone at 35,000 feet, check into your Hyatt hotel room, schedule the courtesy bus to pick you up, and have your keys waiting for you at the express desk. This new reservation system was ahead of schedule, under budget, with extra features -- for a mere $15 million of cold cash. They used modern, open systems software with an Informix database and the TUXEDO transaction monitor, on Unix-based hardware.

Hyatt had all the right ingredients for success: user involvement, executive management support, a clear statement of requirements, proper planning, and small project milestones.

Banco Itamarati

A year after a strategic redirection, Banco Itamarati, a privately-held Brazilian bank, produced an annual net profit growth of 51% and moved from 47th to 15th place in the Brazilian banking industry. Three fundamental reasons account for Banco Itamarati's success. First, they had a clear vision with documented specific objectives. Second, their top-down level of involvement allowed Banco Itamarati to stay on course. And finally, the bank produced incremental, measurable results throughout the planning/implementation period.

Banco Itamarati's clear business goal is to be one of Brazil's top five privately-held banks by the year 2000. Their objectives include maintaining a close relationship with their customers to improve and maintain an understanding of their needs, offering competitive financial solutions, guaranteeing customer satisfaction, and finally producing balanced results for the Itamarati Group. Banco Itamarati's objectives were incorporated into a strategic plan that clearly identified measurable results and individual ownership.

Their strategic plan made technology a key component of the business strategy. Itamarati used Itautec's GRIP OLTP monitor as a basic tool for integrating software components. According to Henrigue Costabile, Director of Organization Development, "We are one of the first banks to implement a client-server architecture that maximizes the potential of this architecture." Executive leadership, a well-communicated plan, and a skilled diverse team provided the foundation for Banco Itamarati to achieve their long-term goal, potentially ahead of schedule.

## CASE STUDY CONCLUSIONS

The study of each project included adding up success points on the "success potential" chart.

| SUCCESS CRITERIA | POINTS | DMV | CONFIRM | HYATT | ITAMARATI |
|---|---|---|---|---|---|
| 1. User Involvement | 19 | NO ( 0) | NO ( 0) | YES (19) | YES (19) |
| 2. Executive Management Support | 16 | NO ( 0) | YES (16) | YES (16) | YES (16) |
| 3. Clear Statement of Requirements | 15 | NO ( 0) | NO ( 0) | YES (15) | NO ( 0) |
| 4. Proper Planning | 11 | NO ( 0) | NO ( 0) | YES (11) | YES (11) |
| 5. Realistic Expectations | 10 | YES (10) | YES (10) | YES (10) | YES (10) |
| 6. Smaller Project Milestones | 9 | NO ( 0) | NO ( 0) | YES ( 9) | YES ( 9) |
| 7. Competent Staff | 8 | NO ( 0) | NO ( 0) | YES ( 8) | YES ( 8) |
| 8. Ownership | 6 | NO ( 0) | NO ( 0) | YES ( 6) | YES ( 6) |
| 9. Clear Vision & Objectives | 3 | NO ( 0) | NO ( 0) | YES ( 3) | YES ( 3) |
| 10. Hard-Working, Focused Staff | 3 | NO ( 0) | YES ( 3) | YES ( 3) | YES ( 3) |
| TOTAL | 100 | 10 | 29 | 100 | 85 |

With only 10 success points, the DMV project had virtually no chance of success. With 100 success points, Hyatt's reservation project had all the right ingredients for success. With only 29

success points, the CONFIRM project had little chance of success. With 85, Itamarati, while not as assured as Hyatt, started with a high success probability.

## THE BRIDGE TO SUCCESS

Notwithstanding, this study is hardly in-depth enough to provide a real solution to such a daunting problem as the current project failure rates. Application software projects are truly in troubled waters. In order to make order out of the chaos, we need to examine why projects fail. Just like bridges, each major software failure must be investigated, studied, reported and shared.

Because it is the product of the ideas of IT managers, the "Success Potential" chart can be a useful tool for either forecasting the potential success of a project or evaluating project failure.

Research at The Standish Group also indicates that smaller time frames, with delivery of software components early and often, will increase the success rate. Shorter time frames result in an iterative process of design, prototype, develop, test, and deploy small elements. This process is known as "growing" software, as opposed to the old concept of "developing" software. Growing software engages the user earlier, each component has an owner or a small set of owners, and expectations are realistically set. In addition, each software component has a clear and precise statement and set of objectives. Software components and small projects tend to be less complex. Making the projects simpler is a worthwhile endeavor because complexity causes only confusion and increased cost.

There is one final aspect to be considered in any degree of project failure. All success is rooted in either luck or failure. If you begin with luck, you learn nothing but arrogance. However, if you begin with failure and learn to evaluate it, you also learn to succeed. Failure begets knowledge. Out of knowledge you gain wisdom, and it is with wisdom that you can become truly successful.

# DBMS Bookmarks

( position paper )

March 31, 1995

Ian Jose

Microsoft Corporation

## Introduction

The position described in this paper was developed through work with Microsoft Access, a wildly popular PC DBMS, and with Microsoft Exchange, an unreleased mail product, with a server database. Bookmarks are record identifiers. They are currently a controversial topic in Microsoft as existing DBMSs adopt new features, including clustered indexing and on-line compaction, which invalidate previous bookmark designs.

Bookmarks have a variety of uses in both client and server components of DBMS products. Server component uses include **non-clustered indexing**, **query execution**, cursors, record lock management, and record multi-version management. Client component uses include updatable query results and cursors. This paper focuses on bookmark designs for server component non-clustered indexing and query execution uses. In non-clustered indexing, bookmarks are the record pointers. In query execution, bookmarks are used as record data surrogates, avoiding costs of data retrieval from records not in the final query result. They are also used to combine subquery results into a final query result.

Bookmark designs must support minimal requirements and, in addition, may meet design goals to greater and lesser extents. Forward reference and primary key bookmark designs are criticized in the context of the requirements and goals defined in this paper.

## Bookmark Design Requirements

Bookmarks must support the following requirements.

### *unique identification of a record*

For the uses described in this paper, unique identification of a record is only required in the context of a table. It may be possible for an inserted record to have the same bookmark as a previously deleted record. In order to avoid aliasing between the two distinct records, a duration of bookmark validity must be established. For example, a bookmark may be valid for the duration of a transaction in which it is retrieved, or for the duration the cursor with which it is retrieved is open.

### *navigation to a record*

A bookmark must support navigation to the record it points to during the period the bookmark is valid. If the record has been deleted, then an appropriate error code should be returned from the navigation operation.

### *consistency*

Query engines must be able to decompose queries into subqueries on different table indexes, and logically combine the resultant sets of bookmarks. In order to logically combine sets of bookmarks, i.e., intersect, union, etc., comparable[1] bookmark values must be returned for the same record regardless of the index from which the bookmark is retrieved, and also for the duration of bookmark validity.

### *retrieval from non-clustered index without record page access*

Retrieval of bookmarks from non-clustered indexes must not access record pages. Query execution will retrieve large numbers of bookmarks from non-clustered indexes and combine these intermediate results with set operations, greatly reducing the number of bookmarks in the final query result. Accessing record pages in non-clustered index order typically has a poor locality and will result in one or more buffer cache misses for each bookmark retrieved. Avoiding page accesses during non-clustered index bookmark retrieval so dramatically improves query performance that it is a requirement of a bookmark design.

## Bookmark Design Goals

In addition to design requirements, bookmark designs should endeavor to meet the following goals.

### *navigation to a record with minimal page accesses*

Bookmarks are used to navigate to records when accessing record data from non-clustered indexes and when data is retrieved from records in query results. Page accesses, which lead to buffer cache misses and limit server through put, should be minimized.

### *record reorganization with minimal page accesses*

Records may be reorganized during page splits and page merges. Page splits occur in clustered indexes when a record is updated or inserted on a page and insufficient page resources are available to support the operation. Page merges occur in clustered indexes when record updates or deletions allow two or more adjacent pages to be merged into fewer pages. Page merges may also occur when no clustered index is present, when record deletions or reductions allow two or more adjacent pages in a sequential ordering to be merged into fewer pages. Page accesses should be minimized.

### *orderable by record page*

If bookmarks can be ordered by record page, bulk bookmark navigation can be optimized to access those records from the same page together[2]. This can substantially reduce the number of page accesses when multiple bookmarks are from the same page. Accessing records from different pages in page order may also be more efficient via read-ahead.

### *compact*

Non-clustered indexes will have one bookmark in every index entry. More compact bookmarks will allow more index entries per page and hence reduce page accessed during non-clustered index range scans.

---

[1] Bookmarks which represent the same record but are not identical may be compared to be equal according to rules or a provided comparison function.

[2] Bulk bookmark navigation includes non-clustered indexes with duplicate key values. Ordering non-clustered indexes by both key and record page number minimizes record page access during non-clustered index range scans.

### *unrestricted record reorganization*

Retrieved bookmarks should not prevent records from being reorganized, due to page splits, or page merges. Preventing page splits reduces concurrency. Preventing page merges can cause additional page accesses during subsequent DBMS operations.

## Bookmark Design Alternatives

### *Forward Reference*

Forward reference (FR) bookmarks are the record page number/slot number address. When records are reorganized to a new page during split or merged, a forward reference to the new page number/slot number is stored in the original address and the original bookmark is stored in the reorganized record [1]. The bookmark uniquely describes the record until such time as the record address is available for reuse. This can occur if the record is deleted thereby freeing the page number/slot number for reuse, or if the forward reference for a reorganized record is deleted, again freeing the page number/slot number for reuse. As a result, deleted record addresses are freed by a back ground cleaning thread instead of being freed immediately. Forward references are also deleted and freed by the back ground cleaning thread. The cleaning thread only operates on bookmarks that are no longer valid and hence bookmarks uniquely describe a record for the duration of validity.

Navigating to a record is as simple as accessing the page number/slot number and checking for a forward reference. If one exists, the record is accessed with the forward reference page number/slot number.

When a bookmark is retrieved from a record, the page number/slot number is returned, or if the record has been reorganized, the stored original bookmark is returned. As a result, a consistent bookmark is returned until a back ground cleaning thread deletes the forward reference and stored original bookmark. Note that the back ground thread must only operate on bookmarks that are no longer valid. This may mean that overlapping usage of bookmarks prevents the back ground cleaning thread from ever operating on a table.

Non-clustered indexes contain FR bookmarks as record pointers, and hence bookmarks can be retrieved from non-clustered indexes without accessing the record page. Also, non-clustered indexes are not updated with new record bookmarks during record reorganization. Instead, they are updated by a back ground cleaning thread. The background cleaning thread operates atomically on both the record and all the record non-clustered index entries. As a result, bookmarks retrieved from non-clustered indexes are consistent in the same way with bookmarks retrieved from the clustered index.

Navigation to a record is in typically one page access, or two if the record has been reorganized since the last back ground clean up could operate on the record.

Record reorganization is required to access pages for all previously moved records. In the worst case, one additional page access is required for each record moved! As the number of records on a page increases, more page accesses are required for record reorganization.

FR bookmarks can be ordered by original record page number since this is contained within the bookmark. Provided that records have not been reorganized too many times since the backup ground clean was last able to operation on them, this is as good as ordering by actual page number. Non-clustered indexes can be ordered by both key and bookmark for reduced record pages accesses during non-clustered index range scans of duplicate keys. Note that the back ground cleaning thread must also coordinate operation with queries performing non-clustered index range scans to avoid the scan finding the same record twice, when a bookmark of low value is updated to a bookmark of high value, or conversely, not finding a record in the index when a bookmark of high value is updated to a bookmark of low value.

FR bookmarks are small, and can stored in non-clustered indexes as big-endian page number/slot number for prefix compression along with non-clustered index keys.

FR bookmarks do not restrict record reorganization. Both page splits and pages merge can proceed since neither operation changes a record bookmark or prevents navigation to a record from its bookmark.

### *Primary Key*

Primary key (PK) bookmarks are primary keys. All tables must have and be clustered on a primary key. When records are reorganized to a new page, the primary key is unchanged and can still be used to navigate to the record via the clustered index [1]. It is possible for a record to be deleted and another record to be inserted with the same primary key. As a result, deleted keys must be retained for the duration of bookmark validity. The duration can be for the current transaction which would then allow existing isolation mechanisms, i.e., locks or multi-versioning, to be used to guarantee bookmark validity.

Navigating to a record is a simple as a clustered index seek.

When retrieving a bookmark, the primary key is returned. Since the primary key does not change, a consistent bookmark is returned.

Non-clustered indexes contain primary keys and, hence, PK bookmarks can be retrieved from non-clustered indexes without accessing record pages.

Navigation to a record is typically three to four page accesses, depending on the size of the relation. However, considering locality, two to three of these pages should already be buffered and actual page faults should be one to two [1].

Record reorganization can occur with minimal page access. Non-clustered index pages do not have to be accessed as the record pointers are unaffected by the record reorganization.

PK bookmarks are orderable by record page to the extent that the clustered index is ordered by record page. Frequent update activity, with associated page splits and page merges, may leave adjacent clustered index leaf pages poorly clustered. However, ordering PK bookmarks will ensure that each record page is accessed only once for all bookmarks for records on that page.

Primary keys may not be compact but typically are compact. For example, purchase order number can be just as compact as record page number/slot number. Further, primary keys may be prefix compressed when non-clustered index duplicate keys are present.

PK bookmarks do not restrict record reorganization.

## Conclusion

Both forward reference and primary key bookmark designs satisfy the minimal design requirements described in this paper. Forward reference bookmarks are a natural choice at Microsoft, since previous bookmark designs, in DBMSs which did not support clustered indexing or on-line compaction, used record page number/slot number as bookmarks. Many query engine data structures and algorithms have been designed specifically for page number/slot number bookmarks. However, primary key bookmarks have substantial benefits over forward reference bookmarks.

Primary key bookmarks require no additional page accesses during page splits and merges, over that normally found in index page splits and merges. Forward reference bookmarks require additional page accesses to update the forward references for those records which did not originally reside on the page being split or merged. Further, these accesses have poor locality and are likely to be buffer cache misses. Also, deleting forward references, to reduce page accesses during page splits and merges, requires non-clustered index updates. Again, poor cache locality is likely to result in two buffer cache misses per

forward reference deleted per non-clustered index[3]. Lastly, the forward reference bookmark design described in this paper cannot delete forward references in the presence of bookmark usages which have overlapping durations of validity. This load must be supported by server DBMSs.

Primary key bookmarks may involve typically two page faults per bookmark navigation, but this is no worse than forward reference bookmarks with a substantial percentage of record reorganizations. Further, primary key bookmark record reorganizations do not require non-clustered index updates. As a result, the primary key bookmark should be employed to the exclusion of the forward reference bookmark design in server DBMSs.

## References

[1] J. Gray and A. Reuter. Transaction Processing Concepts and Techniques. Morgan Kaufmann Publishers, San Mateo, California.

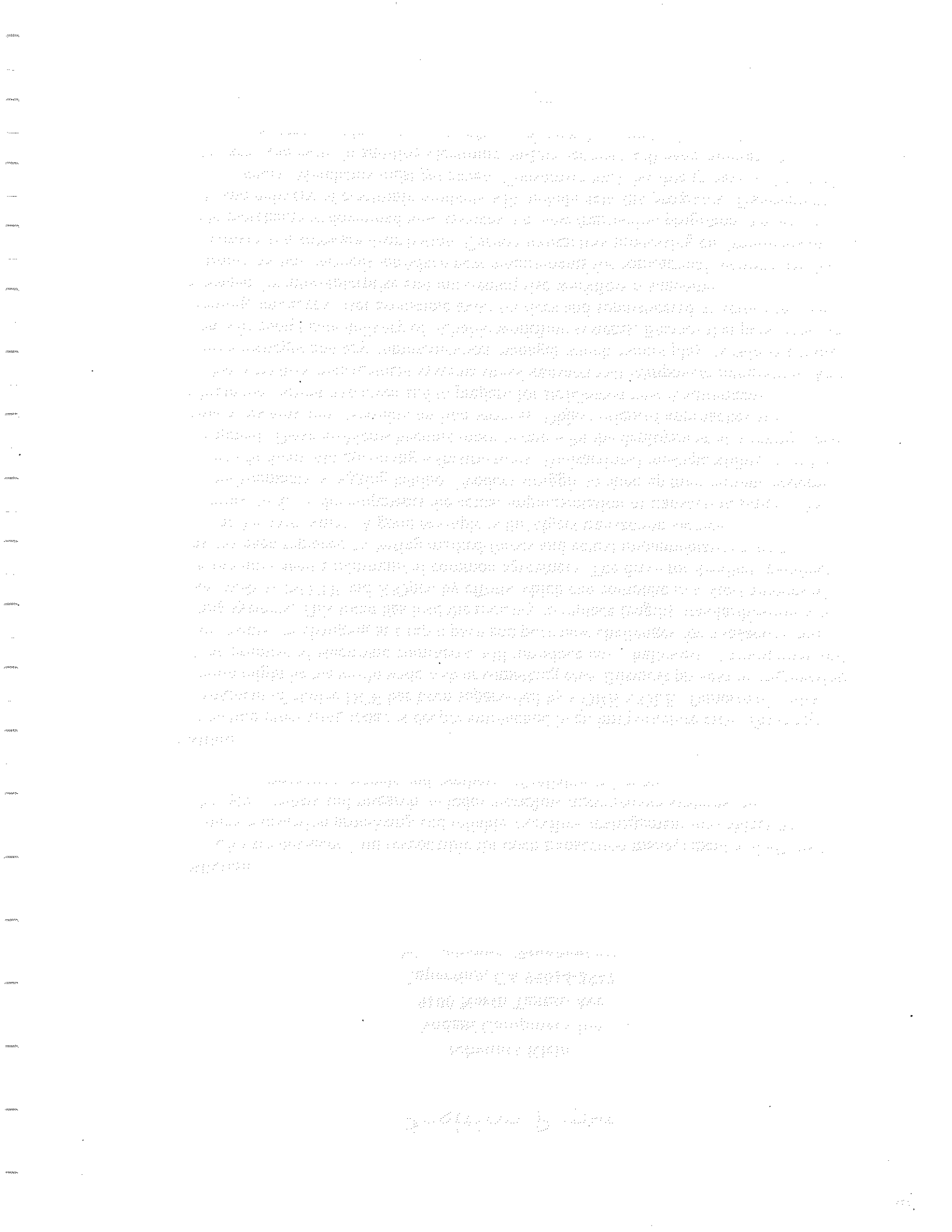
[3] Note that as many as four page accesses may be needed to update each non-clustered index bookmark when a non-clustered index has many duplicates, since the index is ordered by both key and bookmark.

# Position Paper

**Johannes Klein**
**Tandem Computers Inc.**
**10100 North Tantau Ave.**
**Cupertino, CA 95014-2542**
**klein_johannes@tandem.com**

**Affiliation.**
As a software designer I am responsible for open transaction management and efforts to introduce transaction processing and reliable workflow management into object frameworks. My interests and research includes workflow management systems, electronic commerce, transaction models and synthesis of reliable protocols.

**Position.**
Transaction processing today is deeply entrenched in an infrastructure race. Yesterday's environment of choice DCE has been superseded by CORBA/OLE. Tomorrow's infrastructure might be the world wide web or something else. Business process re-engineering and the promise of electronic commerce will introduce new challenges. Assumptions and requirements are changing at a rapid pace and pose new challenges for transaction processing systems. This trend has lead the industry to replace lengthy standardization processes such as ISO/TP and X/Open by efforts which can complete in a short amount of time and only need a minimum of common agreement. The drive for detailed specifications has been replaced by letting marked forces and actual implementations work out many of the hard issues. A good example is the object transaction service.
In contrast to those developments the actual implementation of transaction support for new environments is lagging behind. Vendors struggle to open up their current systems and retro-fit them into upcoming infrastructures. Transactional interoperability is still not wide spread. These problems become more severe with the deployment of workflow management systems and electronic on-line services. Object oriented approaches are used to facilitate the current transition and to prepare for anticipated new requirements.
Reliable workflow management systems stress transactional component integration. Any resource manager and any communication manager which cannot play its role in a transaction will complicate delivery of reliable workflow systems. Besides that persistent programming languages, user accessible audit services and sophisticated directory services are needed for inter-enterprise and intra-enterprise workflow management.
Electronic on-line services introduce new requirements for commercial security, service deployment and software distribution. Current transaction processing environments must enable merchants to download new services, i.e. new transaction programs. Advertisements and delivery of electronic products will include data and programs. Transactions must be secure. Merchants must get payed. Consumers must be able to have a choice of credit cards and cash. In addition electronic on-line services will need support for workflow management in order to process orders and track the actual delivery of goods.



# Transactional Objects with OpenStep

Charly Kleissner
*Director of OpenStep*
NeXT Computer, Inc.
charly_kleissner@next.com

## 1. Introduction

Today's information systems need to be flexible to adapt to ever changing business conditions. Management of change is a key computing concept of this decade. Information systems must also be robust, scalable, and reliable in order to guarantee correct execution of mission critical custom applications.

Object-oriented tools and development frameworks are starting to deliver the benefits of increased productivity and flexibility, code re-use, easier code maintenance, and better modeling of the real world. In order to scale to the enterprise level, object technologies need to leverage transaction technologies and seamlessly integrate with existing *TP lite* and *TP heavy* system architectures. This marriage of technologies results in the development and deployment of *transactional objects*. These objects fully exploit the object-oriented development environment without being burdened with legacy-style application development as in stored procedures or SQL. At the same time transactional objects take advantage of proven data and process management capabilities of database systems and transaction processing monitors.

The OpenStep multi-vendor initiative has the goal of creating an open, high-volume portable standard for object-oriented computing. The primary benefit of this standard is to enable rapid development of business applications, in particular distributed applications which fully exploit the power of client/server computing through the use of object technology. The OpenStep standard has been published and can be copied from the net free of charge. NeXT Computer, Inc. provides a reference implementation and a test suite that certifies compliance with the reference implementation.

In this extended abstract we summarize the products, architectures, and technologies that allow OpenStep objects to scale up to the enterprise level by taking advantage of existing transaction management technologies. These transactional objects leverage object-oriented technologies for application development and transaction processing technologies for production deployment.

We first review object-oriented client/server architectures and then describe the Enterprise Objects Framework, a product that is used to automate the process of storing complex objects in relational database systems. We then outline the usage of Enterprise Objects Framework to implement TP lite objects and show how this can be extended to TP heavy objects. We conclude with an assessment of major industry trends in the area of object-oriented transaction processing.

## 2. Object-Oriented Client/Server Architectures

Object-oriented client/server architectures maintain a clear separation between the user interface, business objects, and the database server. Business policies are implemented as methods on business objects in order to take full advantage of customization, inheritance, and re-use. The user interfaces of business objects may be changed without having to change the business policies. The reverse is true as well: The developer may choose to change certain business policies while maintaining the same user interface.

In non object-oriented client/server architectures, some of the business policies might be implemented in screen definition code of 4th generation language products or stored away as stored procedure code in database products. These non object-oriented approaches to distributed application design do not yield the desired results with respect to code re-use and therefore are not covered in this abstract.

Business objects transform the traditional two-tiered relational database model into a three-tiered model. The user interface usually executes on a client machine, the database server on a server machine. The business objects may execute either on the client, on the database server, or on a separate application server.

This flexible, object-oriented, three-tiered client/server architecture provides the architectural framework for transactional objects. Scalability requirements with respect to networking traffic, session management, and server management dictate where the objects actually execute and which part of the system architecture manages the communication between various tiers, i.e., the database management system, the distributed object infrastructure, or the transaction processing monitor.

This architectural framework can be used to implement object warehousing as well as workflow applications. In object warehousing, the main data resides on a database server on the third tier. The online transaction processing applications which have the most stringent throughput and response time requirements are executed on the second tier. The second tier also features some object store (e.g., an object oriented database) which is used as a mostly read-only decision support database, i.e., an object warehouse. In workflow applications, the network topology usually includes multiple different types of object servers. Enterprise objects which execute on these servers decide at execution time that some other type of enterprise object needs to be invoked or created in order to achieve a business objective. The dynamic nature of the Openstep object model allows this type of flexibility. Utilizing the transactional technologies extends this flexibility to include the required level of robustness and scalability for enterprise deployment.

OpenStep products like Portable Distributed Objects (see *[NeXT-OpenStep 95]*) and Enterprise Objects Framework provide the core object infrastructure for flexible two-tiered or three-tiered object-oriented client/server system architectures. They can be used in conjunction with existing relational databases and TP monitors to provide the desired robustness for enterprise production systems.

# 3. Enterprise Objects Framework

The most significant problem that developers face when using object-oriented programming environments with relational databases is the difficulty of matching static, two-dimensional data structures with the extensive flexibility afforded by objects. The features of object-oriented programming concepts such as encapsulation, inheritance, and polymorphism and their associated benefits like fewer lines of code and greater code re-usability are often negated by the programming restrictions that come with accessing relational databases within an object-oriented application.

In his recently introduced "Third Manifesto" (see*[DD95]*), Date faults the relational database vendors for not correctly implementing the relational concept of "domain" which he falsely equates to "object class" (also see *[Date 95]*), thereby mixing up user-defined data types with object classes. There is, of course, no one-to-one mapping between an object class and any single relational concept (like domain or relations) since objects contain not only data but also methods and adhere to an object model which defines, among other things, the semantics of inheritance and polymorphism. It does, however, make sense to define the mapping between the data portions of the object model and the relational model.

The Enterprise Objects Framework product provides a conceptual bridge between a mature object model as described and implemented in OpenStep (see *[NeXT-OpenStep 95]*) and existing relational database products by

taking the more practical approach of allowing data of multiple base relations to be mapped to an enterprise object. Enterprise objects, like any other object, also contain logic, algorithms, and methods. These methods represent business policies and procedures. Only part of an enterprise object's data may be stored in a relational database management system, other parts may not require the same level of persistence or may be stored in different storage engines.

Object technology improves developer productivity and simplifies maintenance by enabling applications to be built out of re-usable, pre-built components. Relational database systems have the reliability and performance needed to build large-scale, multi-user decision-support and production on-line transaction processing applications - capabilities still unavailable from most object database systems. Enterprise Objects Framework provides a flexible mapping between the two models.

# 4. TP Lite Objects

The Enterprise Objects Framework system architecture provides the framework for TP lite objects. It closely follows the three-tiered object-oriented client/server architecture introduced earlier. It is divided into two major layers: the interface layer and the access layer. The interface layer provides a mechanism for displaying data, while the access layer provides data access methods to storage engines. The layers are separated by a data source protocol which provides a generic object interface to data. The information which describes the mapping between the relational world and the objects is stored in model files. In this section we briefly describe the data source, the two layers, and the transactional semantics of TP lite objects before we summarize two-tiered versus three-tiered system architecture options and trade-offs.

## *4.1. Data Source*

A data source is a protocol that has the ability to fetch, insert, update, and delete enterprise objects. It is the means by which the interface layer accesses stored data; from the perspective of the interface layer, how data is stored (whether in a relational database or a flat-file system, for example) is of no consequence. The interface layer interacts with all data sources in the same way.

## *4.2. Interface Layer*

The interface layer contains a controller and the user interface objects themselves. The controller coordinates the values displayed in the user interface with its enterprise objects. The user interface objects display data from enterprise objects.

## *4.3. Access Layer*

The Framework currently provides an implementation of an access layer which conforms to the data source protocol and which provides access to relational storage engines. Access layers to other storage backends (e.g., flat file systems, object-oriented database management systems, etc.) may be plugged into the infrastructure provided by the Framework. Note that the classes in every component in every layer have public interfaces and may be subclassed, adapted, and extended by the customers.

The access layer itself is comprised of two components: the access component proper and the adaptor component. Adaptors provide data access to a particular relational database server. Adaptors for Sybase and Oracle are bundled with the product. Adaptors for other database products (e.g., DB2, Informix, InterBase, etc.) are provided by third parties. The access component provides the translation from relational data into an object graph. It implements transaction semantics on the object level.

### 4.4. Transactional Semantics

The Enterprise Objects Framework uses *snapshots* to implement flexible and robust concurrency control mechanisms. An update strategy determines how updates should be made in the face of concurrent changes by others. The default update methodology is optimistic locking which assumes that the data won't be changed by others, but checks this assumption before writing changes back to the database. The access component performs this check by comparing the data in the snapshot with the data in the database. An exception is raised if differences for the relevant attributes are detected. The Framework also provides for pessimistic locking and no locking (i.e., write back dirty data independent of conflicts). The user may also declare that there are no updates; in that case the system does not take any snapshot at all.

The Framework also provides a flexible buffering mechanism that determines when changes in the user interface are actually applied to the enterprise objects. The system keeps a stack of undo operations which may be applied before the changes are actually committed to the database. The buffering mechanism coupled with the concurrency control options provide flexible and powerful transaction capabilities. *[NeXT-EO 94]* contains a more thorough and comprehensive description of some of these concepts.

### 4.5. Two-Tiered

Customers may choose to execute their enterprise objects on the client side and use the database client library to obtain access to the database (i.e., fat client architecture). This system architecture is similar to most 4GL or OO4GL tools and scales with the database. There is no customer or object code running on the server and subsequently scalability becomes an issue. The appropriate SQL code is generated by the Enterprise Objects Framework's adaptor component.

Customers may choose to execute their enterprise objects on the server side and use OpenStep native distributed object capabilities to communicate between client and server (i.e., fat server architecture). This system architecture leverages the CPU power of the server by executing the business policies on the server side. The robustness (e.g., transactional transport, number of sessions, etc.) of the distributed object component has a big impact on the scalability of this system architecture.

Customers may choose to combine the above two models and can thereby leverage the distributed objects' capability of migrating objects between clients and servers without changing any application code. Two-tiered architectures are simple and well suited for simple distributed applications that do not require scalability beyond the database scalability.

### 4.6. Three-Tiered

In three-tiered architectures, customers decide to execute some of their business policies on dedicated or specialized servers. TP lite objects leverage database technology and do not integrate with TP monitors. Depending on the requirements, some of the enterprise objects can also execute on the client side or the database server. Three-tiered architectures provide the most flexibility, but are harder to manage from an operational standpoint (e.g., security, dynamic load balancing, etc.).

## 5. TP Heavy Objects

In order to provide high-end scalability for OpenStep objects, an integration with TP monitors needs to occur. The goals of this integration are threefold: (1) to maintain all benefits of an object oriented development environment like OpenStep, (2) to maintain all benefits of Enterprise Objects Framework's seamless integration of the relational world into the object world, and (3) to leverage the scalability, distributed transactions and reliable queuing of a TP

monitor. In this paper we will show how to extend the current Enterprise Object Framework's capabilities to achieve this. Following is a very short synopsis of the architectural approach.

The following three pieces of code need to be plugged into the existing architecture to achieve these goals: (1) a *TP Monitor Adaptor* (for each TP Monitor), (2) a *TP/DB Adaptor* (for each database within each TP Monitor), and (3) a *DB Resource Service* (for each database within each TP Monitor).

The TP Monitor Adaptor and TP/DB Adaptor link in with the TP Monitor client library either on the client side or on the second tier. The TP Monitor Adaptor creates tx_calls based on the transaction policies specified within the transactional objects. The TP/DB Adaptor creates dynamic SQL. The Resource Service knows how to evaluate SQL and how to fetch results for a particular database.

This level of integration works with dynamic SQL. Performance gains can be achieved by creating stored procedures and executing stored procedures instead of dynamic SQL. However, it is not advisable to bury business policies into these stored procedures as one will loose the object-oriented advantages of application code.

# 6. Conclusion

We see a couple of trends in the industry that will bring object orientation and transaction processing closer together. Relational database vendors are extending their products with object capabilities as well as process management capabilities like a TP monitor. TP Monitor vendors will also extend their product offerings with object capabilities in the future. Finally, object oriented vendors are putting some of the reliability and scalability infrastructure attributes into their distributed object infrastructure and are learning how to effectively integrate with legacy databases and transaction monitors.

The OpenStep product suite currently offers three-tiered object-oriented client/server computing. Transactional objects can take advantage of TP lite as well as TP heavy based architectures in an integrated way.

# 7. Acknowledgments

The ideas expressed in this paper reflect the work of many people. The author wishes to thank all members of the Enterprise Objects Framework project team, and particularly the members of the design and implementation team for pioneering and developing the concepts and implementing the Framework. The main designers and implementors are Craig Federighi, Linus Upson, Dan Willhite, and Richard Williamson. Jim Mynatt was instrumental in expanding the implementation from TP Lite to TP Heavy.

# References

*[Date 95]* Date C.J., "An Introduction to Database Systems", Sixth Edition, Addison-Wesley, 1995.
*[DD 95]* Darween H., Date C.J., "Introducing The Third Manifesto", Database Programming & Design, January 95.
*[NeXT-EO 94]* "Enterprise Objects Framework: Building Reusable Business Objects", White Paper, NeXT Computer, Inc., 1994.
*[NeXT-DO 94]* "Interoperability Through Distributed Objects", White Paper, NeXT Computer, Inc., 1994.
*[NeXT-OpenStep 95]* "Portable Distributed Objects (PDO): The NEXTSTEP/OpenStep Object Model", White Paper, NeXT Computer, Inc., 1995.

# Memory-Resident Database Systems: A Look Back to 1985

Tobin Lehman

IBM Almaden Research Center
650 Harry Road (K55/801), San Jose, CA-95120

- HPTS Position Paper -

At the 1985 HPTS I presented some new index algorithms for memory-resident database systems. The tone of the talk was futuristic—memory-resident DBMSs were not yet quite ready for prime time, but their time was coming. Given the 1985 memory sizes (PC sizes 256k to 4 megabytes, medium and high end machines 4 to 64 megabytes ) and memory prices (around 150 $/mb), the trend of memory sizes quadrupling every 3 years (at a seemingly constant price per chip) suggested that memory prices would be at 1 $/mb by 1996.

The following table shows the (flawed) thinking. Starting with the 256k by 1 bit chips at $ 4 each, using 9 of them to get 256k bytes of memory (with parity), it seemed clear that as long as the individual chip prices remained relatively constant (let's say we multiply by 50% for 3 years of inflation and other factors), we'd be near 1 $/mb before the year 2000.[1]

| year | Chip Size | Chip Price | Total Cost per mb |
|---|---|---|---|
| 1985 | 256k bit | $ 4.00 | $ 142.00 |
| 1988 | 1m bit | $ 6.00 | $ 54.00 |
| 1991 | 4m bit | $ 9.00 | $ 20.25 |
| 1993 | 16m bit | $ 13.50 | $ 7.60 |
| 1996 | 64m bit | $ 20.25 | $ 2.85 |
| 1999 | 256m bit | $ 30.38 | $ 1.07 |

At 1 $/mb, even the most basic business PC would have a minimum of 1 gigabyte or DRAM—making memory-resident database systems not only possible but common-place since most PC applications could store all needed data in memory.

There was another interesting trend. It seemed that the disk drive folks were getting worried—they weren't sure if they could increase densities much more. Certainly the high-end drives, such as the 14 inch IBM 3380's, couldn't spin any faster without breaking apart. At one time the price per megabyte differed by two orders of magnitude, but the difference grew smaller every day. There was talk about the memory price/performance curves and the disk price/performance curves crossing somewhere in the near future.

Such talk is not heard anymore. Today, the disk drive folks are happily planning for another order of

[1] Granted, chips no longer work this way—1 million bits by 1—but you get the idea. Today's chips would be configured more like 4m by 3 or 4 bits.

magnitude increase in density—giant magneto-resistive fields show great promise for incredible disk densities. In contrast, the chip folks are getting worried. The current high-end chips are operating at line widths of .3 microns, and there is great concern over whether or not we will be able to break the .1 micron barrier. And, interestingly, a snapshot of the current 1995 disk and memory prices:

| |
|---|
| $ 0.34 per disk megabyte |
| $34.00 per memory megabyte (sims). |

shows that the old two orders of magnitude difference is alive and well.

In 1985 we didn't consider the explosion of the PC and laptop markets. The need for 3 1/4 inch hard drives pushed disk drive technology to the limits, without doing the same for DRAM. As a result, a gigabyte of *disk space* is a common minimum configuration on many business PC's, whereas 8 to 16 megabytes of main memory is still a common minimum configuration for business PC's.

Now, in 1995, what does the immediate future hold for memory-resident database systems? I claim that PC-based memory-resident database management systems are just around the corner for two reasons.

First, consider the popular *Rapid Resume* feature on PC's. It maintains the current (volatile) state of applications by saving all of the applications' memory contents just before power-off, and then restoring them on power-on. As users' application and interface contexts grow more complicated, keeping the on-going application sessions will become increasingly important. It will change from being a convenience to a necessity—making data saving a regular on-line activity rather than something that is done immediately before power-off. Also, as the application working space grows to hundreds of megabytes, the data saving component will need to start tracking *incremental* changes to the application data, since a complete working space dump to disk would not only take a bite out of the available disk space (since two full copies would probably be needed) but it would also be time (and disk bandwidth) consuming to continually save the entire memory contents.

Second, the need is growing for a common persistent information repository that can manage most of a user's (and user's applications') assets. And, given the recent trends in application development, it is likely that there will be a large amount of object data that will be stored in this repository, along with some amount of traditional (administrative-like) relational data.

The more advanced (and more expensive) forms of the repository would also be able to act as a cache for data from even larger server repositories. The client object-relational DBMS (or CORD) could offer full buffer pool-like services for both Relation and Object-Oriented data when attached to the server. However, the CORD would also be able to "dock" with the server to checkout some amount of data, and then run stand-alone with that data.

Eventually, the CORD will become the new persistent storage interface for all client data. Applications will be able to do one-stop-shopping for any kind of persistent data, be it files, objects or tables.

How long will it take to get there? Unknown. Although memory is still quite expensive, that doesn't stop people from loading their PC's with 512 megabytes. Some server machines have as much as 36 gigabytes of real memory (even though any one process can address only 2 gb of it). As the customer demand pushes for larger memories to run the upcoming spectacular multi-media applications, the gigabyte memories will become common and affordable. Although memory will probably never be cheaper than disk, soon it will be cheap enough.

# Workflows Make Objects Really Useful

**Frank Leymann**
**IBM Software Solutions Division**
**German Software Development Lab (GSDL)**
**Hanns-Klemm-Str. 45**
**D-71034 Böblingen**
**Germany**

e-mail: frank_ley at vnet.ibm.com

***Abstract:*** *One possible perception of workflow management systems is that of a means to realize advanced, distributed application systems. An application developed according to the workflow paradigm consists of a process model and a collection of program granules. As a result, the application becomes flow independent, and its encompassed program granules become subject of reuse. Since object technology strives towards reusability too, the relation between both technologies is often asked for: We show how workflow technology can be beneficially used from the very beginning during object oriented analysis and design, how transaction handling is eased, and how workflow management systems can even enhance the reusability of objects.*

**Keywords**: Business objects, business processes, object technology, reuse, transaction models, workflow technology.

## 1 Introduction

The WfMC (Workflow Management Coalition) is striving towards standardized interfaces to enable both, interoperability of workflow management systems, and to allow programs to be written is such a way that they can be used within different workflow management systems [26]. The expectation is that the latter will stimulate vendors to provide software components being reusable across the variety of implementations of workflow management systems. This in turn will allow to compose applications out of these components via models of business processes.

Recent efforts of the OMG within its BOMSIG (Business Object Management Special Interest Group) and its vertical industry SIGs show the increasing impor-

In: Proc. 6th Intl. Workshop on High Performance Transaction Systems (HPTS) (Asilomar, CA, September 17 - 20, 1995)

tance of business objects as granules of reuse. A joint cooperation between OMG and WfMC is targeted towards a standard allowing the exploitation of business objects in workflows. The result will facilitate a straightforward composition of applications out of business objects.

This meeting of the workflow paradigm and the object paradigm is very likely just the initial step towards a mutual large-scale exploitation of both technologies, especially within the area of component based software construction: Together, the advent of industry standards for models of business processes and those ready to use "off-the-shelf" components will accelerate the exploitation of workflow technology. The graphical features for editing process models typically provided by workflow management systems [10] will allow for an easy customization of the resulting applications. Inherent features of the object paradigm like polymorphic business objects will make customization even more flexible.

We show in this contribution that workflow technology is a powerful vehicle to enable the widespread exploitation and reuse of objects (or components, respectively): For this purpose we first remind the fundamental benefit of workflow technology, i.e. the enablement of flow-independence and its consequences on application structures is emphasized. Next, we discuss the dual role of a WFMS in an object request broker environment namely as an application object exploiting object services as activity implementations, and as an object service itself providing a framework which allows the exploitation of objects in business processes. Various techniques used in object oriented analysis and design to depict dynamic behavior are abstracted and it is outlined how the modelling features of WFMSs can be beneficially used in these phases. We outline the different roles a WFMS can play in terms of X/Open's reference model of distributed transaction processing and the benefits originating from the provision of the resulting transactional services. Finally, we sketch a transaction model which provides compensation based partial backward recovery in WFMSs to support "business transactions", and the benefits of business transactions on objects.

## 2 Removing Flow Dependency

Nowadays, it is general knowledge within the information processing community that the insight into the so-called *data dependency* of applications and its negative consequences [4] was the primary impetus for the wide-spread exploitation of database management systems that can be realized today. Applications are much more flexible and less vulnerable against changes at the data organization level when leaving data handling with a database management system and exploiting its accompanying features.

### 2.1 The Notion of Flow Dependency

Today, a similar observation can be made: It becomes more and more clear that contemporary applications are obstructions to change the business processes in which enterprises serve their customers. Large application systems contain the code that directly represents these business processes themselves. Consequently, changing a business process (e.g. to become more competitive) requires changes of the affected application systems.

While the algorithmic knowledge of how to perform a particular step of a business process reflects a proper application functions, it is the knowledge of sequencing these steps (i.e. the flow of control) and distributing them to the responsible organizational units (i.e. the flow through the organization) together with the appropriate information (i.e. the flow of data) what reflects the proper business processes ([13], [14]). It is exactly the latter which makes the application vulnerable with respect to changes of the underlying business processes: Applications are *flow dependent*.

### 2.2 Workflow Based Applications

The exploitation of workflow technology allows to separate algorithmic knowledge from knowledge about business processes. The knowledge about the potential sequencing of algorithmic parts and the technical details of their execution, distribution, and data exchange (the so-called *process model*) is left with the workflow management system, allowing to extract the corresponding code from an application, consequently removing its flow dependency. What results is called a *workflow based application* consisting of a process model and a set of programs. These programs are implementations of the proper algorithmic components of the underlying business process and directly support its basic execution steps (the so-called *activities*).

Within a workflow based application, required changes of the business processes are easily achieved by adapting the associated process model. Most frequently, no changes of the encompassed activity implementations (i.e. the basic execution steps) are necessary, revealing that no code has to be modified to reflect the change of business processes. The above mentioned lack of flexibility is removed, i.e. a workflow based application is flow independent and thus more robust.

As a remark only we mention that one can expect beneficial impacts on the productivity of a corporation's application development staff from building workflow based applications: Once the process model is defined, the interfaces between the programs implementing the activities of the business process are defined. In principle, implicit assumptions about data routing, program sequencing and states are removed. Thus, parallel development becomes more effective.

### 2.3 Flexible Implementation Structure

The workflow based application paradigm immediately allows for a very flexible implementation structure: Since the programs may be dispersed on different machines, running in heterogeneous environments it becomes a *networked* application. Moreover, the binding of each activity to its supporting program is dynamic, i.e. it occurs at run time based on the definitions of the workflow [15]; in this sense the locations where the various pieces of the applications are executed are transparent and can be changed at run time. This is another aspect of increased flexibility of applications exploiting workflow technology ("topology independence"). Because the workflow manager does not assume any inherent structure of a program associated with an activity such a program may even be implemented as a client/server structure; in this sense, the client/server paradigm and the networked application paradigm are independent from each other.

## 3 Object Technology

Enterprises are investing today in object technology to improve the productivity of their programmers and to enable even non-data-processing professionals to build applications. Application building will become more and more component based. We describe now the granules that can be used (i.e. ideally *re*-used) for application construction, and how and when workflows can be beneficially exploited.

### 3.1 Granules of Reuse

Based on object technology the most common granules of reuse are class libraries, frameworks, parts, and design patterns. From a reuse perspective they differ as follows (e.g. [8], [22]):

- Class libraries provide for reuse at the finest level exploiting especially subclassing, overriding, and overloading. The emphasize of reuse within class libraries is on implementation, i.e. on code. Reusing class libraries typically requires coding within a programming language.
- Frameworks allow for reuse at a coarser level by ignoring many details of the underlying class library, externalizing only the aspects relevant to the subject application area. Often, classes of the frameworks are used as implemented, sometimes they will be subclassed, or frameworks encompassing abstract classes have to be completed. What is reused within frameworks is a mixture of design and implementation.
- Parts are used exactly as they are provided, especially, they are already completely implemented. The reuse of parts emphasizes a component based application construction. Typically, this construction is performed via a *scripting* language, i.e. a "usage language" (in contrast to the "implementation language" of the parts themselves).

- Design patterns emphasize the reuse of design. A design pattern provides the documentation of a solution for a small architectural problem and explains the intend, trade-offs, and consequences of the chosen design. Since its documentation is independent from a particular programming language, a design pattern must be implemented each time it is used. Thus, it cannot be embodied in code and is only of help for programmers.

As described in (2.2), activity implementations for process models are typically flow independent and free of assumptions about their usage (with the consequence that an external mechanism to establish transaction boundaries is required; see (4.1)): Thus, one activity implementation can be used in many different process models. If both, the activity implementation and the workflow manager comply to the WfMC standard for "invoked applications" the activity implementation can be used in many different WFMSs. As a result, the exploitation of workflow technology also stimulates reuse of code, having activity implementations (i.e. proper application logic) as granules of code reuse. While the reuse of class libraries, frameworks, parts, and design patterns is coupled with object technology reuse based on workflow technology is independent from it.

Beside the fact that activity implementations become subjects of reuse, the strong demand for reusing process models themselves origins from users of workflow technology. For this reason, many WFMSs allow for activity implementations which are again process models (*subprocesses*), thus enabling top/down and bottom/up modelling of processes especially stimulating the reuse of process models as subprocesses. The meaning of this kind of reuse will be amplified by defining industry standards for models of business processes typically performed in particular application domains; these process models can especially be reused as subprocesses in enterprise specific processes.

Also, the application of object technology can be very beneficial in defining reusable process models. For example, subtyping and polymorphism allow to define abstract process models (e.g. a "disbursement" process) by defining activities as method-invocations on abstract classes (e.g. a WITHDRAW method of an ACCOUNT class). The abstract process models are transformed into "concrete" process models (e.g. disbursement for savings accounts or current accounts, respectively) by subclassing the ACCOUNT class with a SAVINGS ACCOUNT class and a CURRENT ACCOUNT class with (obviously) different implementations of the WITHDRAW method.

### *3.2 Removing Flow Dependencies From Objects*

Some of the components built for application construction based on object technology will be *business objects*, i.e. objects which are immediate subjects of business processes (e.g. person, account, contract, business card, etc.). Especially, object technology will be exploited to build such components. Now, one of the underpinnings of

object technology is the insight that robustnesss of a system is normally achieved by encapsulating things which might become subject to changes. So, if the order in which operations are to be performed can change, or if operations can be added or removed, for example, the guidelines of object technology consequently recommend a dedicated control object encapsulating operation scheduling. Thus, to achieve robustness via encapsulation not only behavior and data must be regarded (this is what is usually done) but also "ordering".

If the last proposition is ignored, then following the encapsulation paradigm tends to hide fragments of the proper business processes in the implementations of the components. This means that not only the components themselves become flow dependent, but transitively each component based application as a whole too. In addition, the business processes (having an important value by themselves [14], [24]) are only partially explicitly described and externalized to a broader community. As a consequence, implementing components in a way such that they become flow independent will result in much more flexible component based applications.

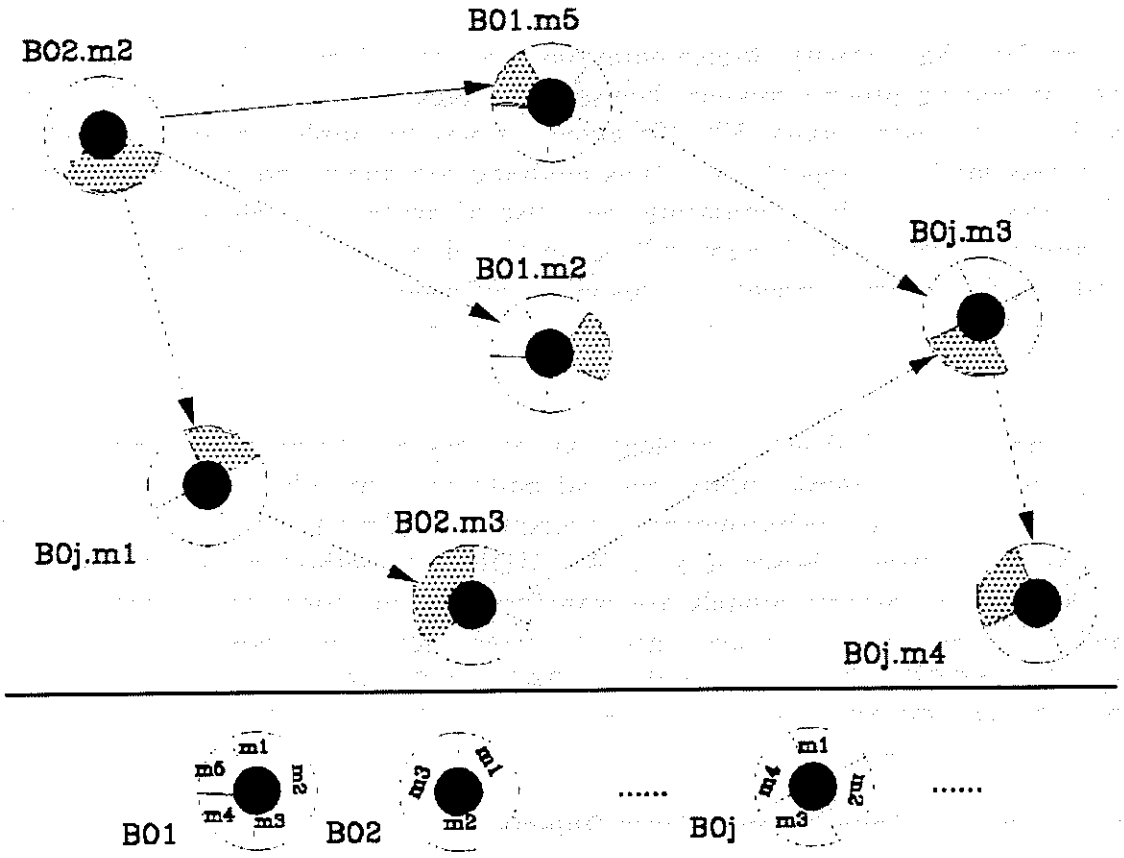



**Figure 1.** *Workflow As Conductor Of Objects*

### 3.3 Scripting

Building flow independent business objects will enforce a clear separation of the

more stable behavior of the business objects from the more dynamic behavior of the business processes. A business process explicitly describes the rules of how, when and by whom the services provided by the various business objects are exploited. An activity within a business process may directly correspond to a method of a business object (shaded areas in figure 1).

When the statics of a business (i.e. its business objects in our current scenario) is split from its dynamics (i.e. its process model) the interaction between business objects is defined by the process model. The process model may be perceived as a *script* prescribing the employment of business objects to reach particular business goals. At run time the workflow management system will manage the flow of control and data between the business objects (figure 1). Even more, it will enforce that the proper organizational units of the enterprise will become responsible for utilizing the services provided by the various business objects.

In order to allow for a direct exploitation of business objects as activity implementations within business processes the workflow management system has to support the invocation of methods of the business objects. When the execution of an activity is requested the workflow management system will directly invoke the respective method. Note, that restricted to control flow C-- follows a similar philosophy: Programs written in C-- in general consist of objects and procedural elements explicitly describing the control flow between method invocations of various objects.

Because of the BOMSIG efforts of the OMG mentioned in the introduction it can be expected that many business objects will become standardized and defined via IDL (Interface Definition Language [20]). These IDL defined business objects can be implemented by using SOM (System Object Model) technology [9]. Then, by natively supporting the invocation of methods of SOM objects the workflow management system facilitates the exploitation of services provided by the corresponding business objects as activity implementations. By further exploiting DSOM technology (Distributed SOM [9]), which is IBM's CORBA [20] implementation) even the location of these objects is irrelevant to the workflow manager and objects accessible via any CORBA compliant implementation can be used as activity implementations. In CORBA terminology the workflow management system (as a scripting feature) would be an application object exploiting object services or common facilities provided by business objects (figure 2).

Also, OpenDoc as well as OLE allow the composition of applications out of components, more precisely out of parts (see 3.1). Especially, the joint presentation of the interfaces of such applications as an integrated, composed document is supported. By adding the mechanism of invoking OpenDoc parts to the workflow manager it will be possible to exploit such parts as implementations of activities too. Vice versa, OpenDoc parts representing particular workflow constructs may be provided. For example, a part handler for activities may allow for including work requests in docu-

ments and to start the execution of work directly from the document. It is the former that reveals that process models may also be perceived as scripting language (in the sense of 3.1) for using parts.

**Workflow as Script**

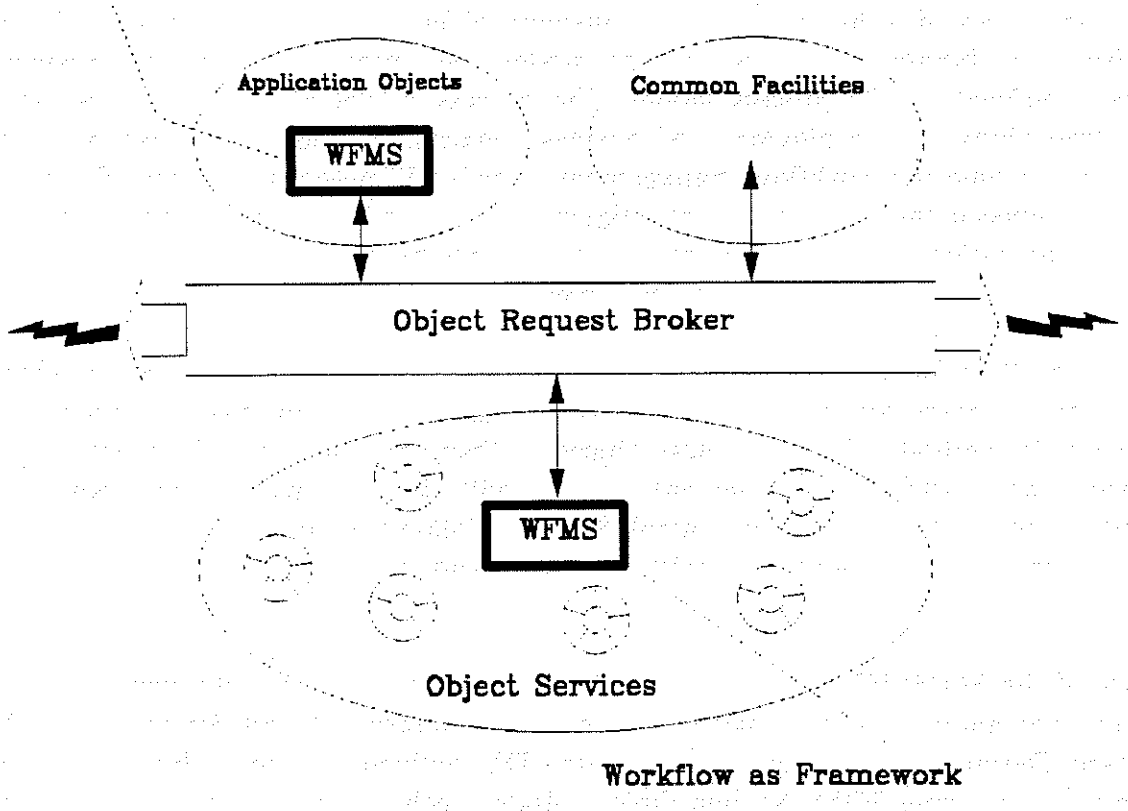



Figure 2. *Workflow Management System And Object Request Broker*

Note, that we do not argue that workflows are a substitute for scripting languages like REXX or VisualBasic. Instead, we would like to introduce the notion of "lightweight scripting" and "heavyweight scripting": *Lightweight scripting* is performed whenever a desktop application has to be composed for a single enduser perhaps by the enduser himself; a "traditional" scripting language is used for this. *Heavyweight scripting* is performed whenever an application has to be composed which requires the collaboration of multiple people distributed in an enterprise; a process model executed by a workflow management system is used for this. Heavyweight scripting adds features like parallelism, heterogeneity, distribution, context dependent (i.e. dynamic) resource allocation (e.g. staff resolution) to the notion of scripting. The implementation overhead inherent to these features is the reason why we call this kind of scripting "heavyweight" and why two different categories of scripting have their own right of existence.

### 3.4 Object Oriented Analysis and Design

Object technology provide many methodologies to capture an application domain during the analysis and/or design phase. The derived object models represent the static representation of an application domain. In order to allow for an analysis of the dynamic behavior of an application various techniques are proposed: Collaboration graphs [25], event flows [21], timing diagrams [1], interaction diagrams [11] all show at the abstract level a structure as depicted in figure 3 which can be called most appropriately *message flow diagram*. Basically, such a diagram may be perceived as a description of the control flow between method invocations. Some authors even associate additional conditions with the "arrows" describing the "control flow" (e.g. [11]) revealing the workflow aspects in the sense of [13] of such message flow diagrams.

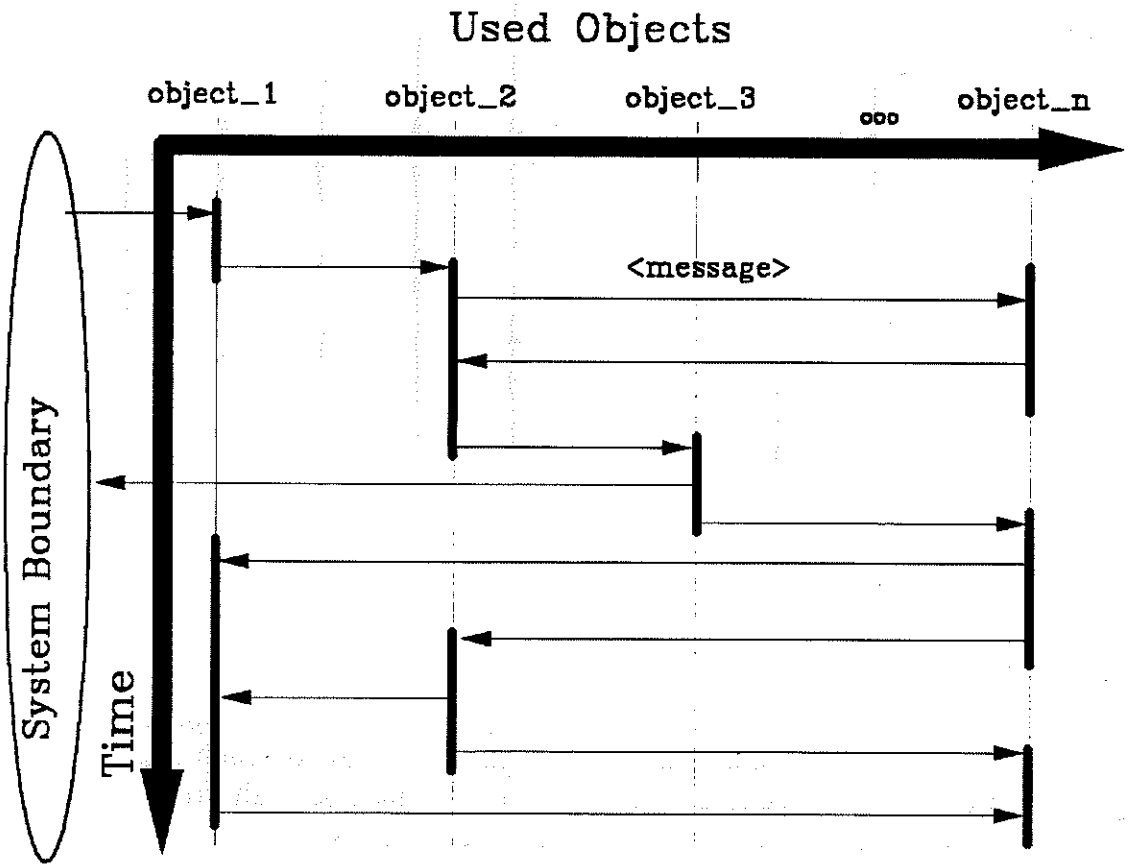



**Figure 3.** *Message Flow Diagram*

In [12] *use cases* represent a business from an external point of view, i.e. by "actors" representing the environment and who want to use the business. Consequently, structures not seen by actors are excluded from the use case but the interactions of the environment with the business is represented. A use case describes the flow of events necessary to yield a certain result. Also, a coarse grained object model (as a result of the analysis phase) accompanies the use case introducing the objects (of one

of the category "interface", "control" and "entity" [11]) necessary to run the business. At a more detailed level (i.e. during the design phase) *interaction diagrams* show how a use case is realized by communicating objects [12]. An interaction diagram shows the stimuli that are transmitted between the objects, as well as parameters that follow the stimuli. Here, the corresponding message flow diagram contains even data flow aspects emphasizing its workflow relation [14].

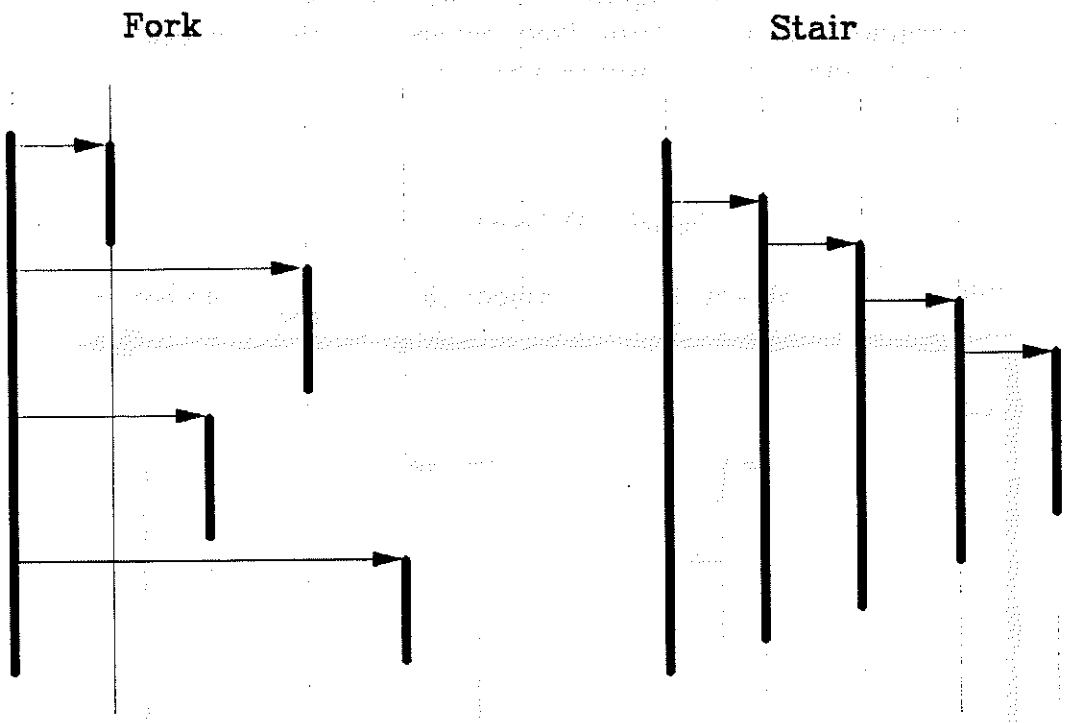



**Figure 4.** *Fork And Stair Diagrams*

Based on the fact that message flow diagrams represent workflow aspects a fundamental observation has been made in [11] on the structure of such diagrams: A message flow diagram is in general composed of two basically different structures namely either a *fork* structure or a *stair* structure (see figure 4).

- Fork structures are implied by centralizing responsibilities, i.e. when the global control and data flow is placed in one object. The remaining objects are used for enquiries, utilities, interfacing users etc.. Using such a kind of "control object" is an approach workflow purists will prefer.
- Stair structures are implied by delegating responsibilities, i.e. when each object only knows a few other objects and how to exploit them. In such a kind of decentralized structure the responsibility for the local control and data flow is with

each object itself; note especially that these objects are then flow dependent. The delegation of responsibility is the approach object purists will prefer.

The proposition derived in 3.2 (namely that robustness achieved via encapsulation must not only regard behavior and data but also ordering) is made manifest in forks: A fork typically encapsulates ordering. In contrast to this, stairs express assumed stability of ordering (or collections of strongly connected operations [11]). In general, fork and stair structures will be used in combination to yield a stable and robust structure.

It is one of the strengths of workflow technology to facilitate in an easy manner especially modifications of orders of operations. Thus, it is only natural to exploit workflow technology for the implementation of fork structures, i.e. for encapsulating ordering of operations: Simply, the controlling object itself becomes an instance of a process model which in turn describes the control flow between the affected objects.

In [23] a methodology has been sketched that uses process models as representations of control objects of use cases; additionally, a protocol is outlined that tights such kinds of control objects together with the (appropriately behaving) other kinds of objects (i.e. interface objects and entity objects) to result in a workflow application. Since that approach requires the exploitation of workflow technology after the analysis phase we sense obstructions from object purists against this. Because of this, our approach goes one step further: We allow modelers to stay in the object paradigm also during the design phase and we support them to map their design to workflow based applications afterwards.

The enabling factor for this is our fundamental observation that (subdiagrams of) message flow diagrams can be mapped to templates of process models (in terms of [13]). Before applying the corresponding transformation we analyze a message flow diagram and subdivide it into a collection of subdiagrams each of which is either a fork structure or a stair structure. Fork structures typically contain control objects which govern the behavior of the affected objects and are thus represented as process models in a straightforward manner. Stair structures characterize stable delegation of responsibilities and are natural candidates for modularization (as programs or as subprocesses).

The following sketches our proceeding in transforming a message flow diagram into the control flow aspects of a process model. Since our intend is to show the basic philosophy behind our transformation of message flow diagrams into templates of process models we omit to show how data flows can be derived from message flow diagrams.

1. *Identify all control objects*:

Forks have a single control object. A control object is defined as one sending many messages out and receiving only "a few" messages back, thus the difference of outgoing messages and incoming messages must be "high". The real crux is to tune this difference appropriately to identify control objects.

2. *Build all fork structures*:

   All objects which receive a message from a single control object are included in the associated fork. Objects receiving during their life-time messages from more than one object need a special treatment (not covered in this sketch).

3. *Build all stair structures*

   Each connected component of the diagram which results when all outgoing messages of all control objects are removed is considered to be a stair structure. Especially, a control object receiving a message out of a stair is (virtually) added to the stair.

4. *Transform forks into process models*:

   The target of a message sent by a control object becomes an activity the associated implementation of which corresponds to the invocation of the method specified in the subject message of the target object. If the target is a fork structure its associated diagram is transformed into a subprocess. If the target is a stair structure and it has been transformed into a process model this process model becomes a subprocess; otherwise, the stair is considered to be a single activity with exactly the module as implementation which has been associated with the stair.

5. *Refinement*:

   Message flow diagrams in general hide parallelism. Thus, the resulting control flow structures should be reworked to enhance the degree of parallelism.

6. *Transform stairs*:

   A stair can be transformed into a process model too; from that source code can be generated (see below and [18], [19]). Otherwise, the diagram is transformed into a program with whatever is available.

In 3.3 we made the observation that C-- programs may be perceived in general as specifications of the control flow between method invocations of servicing objects. This in turn shows the desirability of a *workflow compiler* or source code generator which will transform a process model having method invocations as activity implementations into C-- programs (or DSOM programs, respectively: see [18], [19]). This is especially useful if the subject process model shows a particular behavioral pattern like for example being consecutively executed by a single person.

### 3.5 Coexistence

Object technology and workflow technology are two "orthogonal" technologies: One can exploit the other in various ways resulting in synergies valuable for both paradigms, but they can also coexist in the sense that each technology has its own area of applicability without the other.

Coexistence can simply mean that the workflow manager executes programs written in an object oriented programming language. The workflow manager's mechanism of starting programs bound with activities supports natively various invocation mechanisms, for example starting programs provided as EXE files, CMD files, or entry points in DLLs, CICS or IMS transactions, message queue applications, and so on. Since the way a program is constructed is irrelevant to these invocation mechanisms the workflow manager can start the corresponding programs regardless whether they are written in an object oriented programming language or not.

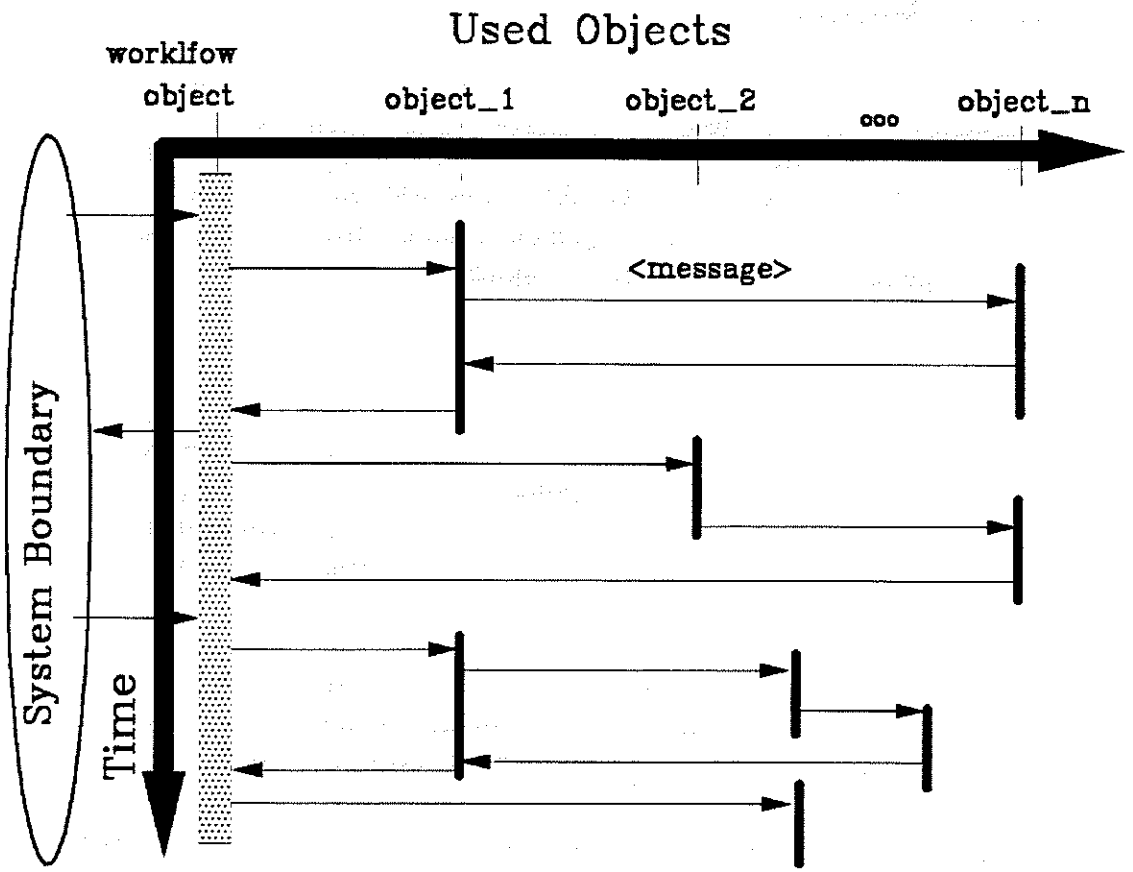



**Figure 5.** ***Workflow Framework***

Another form of coexistence results when treating the workflow management system as a collection of object services accessible via an object request broker (*workflow framework*, see figure 2 and figure 5). This will allow to let application objects to participate in workflows, especially those who will not adapt the workflow paradigm.

For example, an application object which has been included into a process model may use an ACTIVITY object of the workflow framework to communicate a particular state change which might be control flow relevant for the encompassing business process. Such kind of a workflow framework is the expected outcome of the current activities performed in the cooperation between OMG and WfMC.

Business objects may be implemented in such a way that they are not externalizing state information which is relevant for determining the flow within a business process. Such a business object may wish to communicate in an object oriented manner with the workflow manager in order to influence the navigation through the business process the business object is tight in. A workflow framework providing the corresponding services as objects will allow for that. Again, exploiting SOM technology for this may be beneficial because of resulting in language neutrality and binary compatibility.

# 4 Transaction Support

With the increase of exploitation of workflow technology for application construction workflow management systems need to provide transactional services. Both, the capability to bind activities into atomic processing units is required (e.g. [6], [17]), as well as the support of a transaction model for real-world business transactions [16]. This strategically positions workflow management systems from an application point of view as the next generation of transaction management systems.

## *4.1 Atomicity*

When components (e.g. business objects) are build for reuse it is a known proposition from software engineering that their coupling should be weak, i.e. the number and complexity of interconnections between components is to be minimized [5]. As a consequence, when dealing with recoverable resources each single component must be allowed to manage transaction boundaries as appropriate from its own local perspective.

For example, a business object ACCOUNT has a WITHDRAW and a DEPOSIT method. Since a customer may sometimes wish to pay in money to his account, or sometimes wish to withdraw money from his account, both, the DEPOSIT as well as the WITHDRAW method will establish its own transaction boundary. Now, the transfer of money from one account to another will reuse both, the DEPOSIT method and the WITHDRAW method by invoking WITHDRAW on the first account and DEPOSIT on the second. This reveals the necessity of a separate transaction boundary surrounding both method invocations [7]: In case the WITHDRAW or the DEPOSIT method aborts as a subtransaction the whole transfer transaction (as a superior transaction of the former transactions) must be aborted.

Thus, reusing components with separate transaction boundaries does require some flavor of a nested or multi-level transaction model. This will allow for writing components without any assumption on their use in other transactions. It has been shown in [3] how this can be achieve by exploiting OMG's Object Transaction framework [2] which of course nicely matches with business objects. For the integration of applications in workflow management systems [6] proposes enhancements of X/Open's DTP [27] and the introduction of dependency rules to WFMSs which will allow for that.

In [17] a more conservative approach is taken: It is suggested to exploit the existing X/Open DTP protocol and use dependencies implied by a simple and pragmatic modelling construct called "atomic sphere" to combine components into a unit which provides for atomicity of the executions of the collected components.

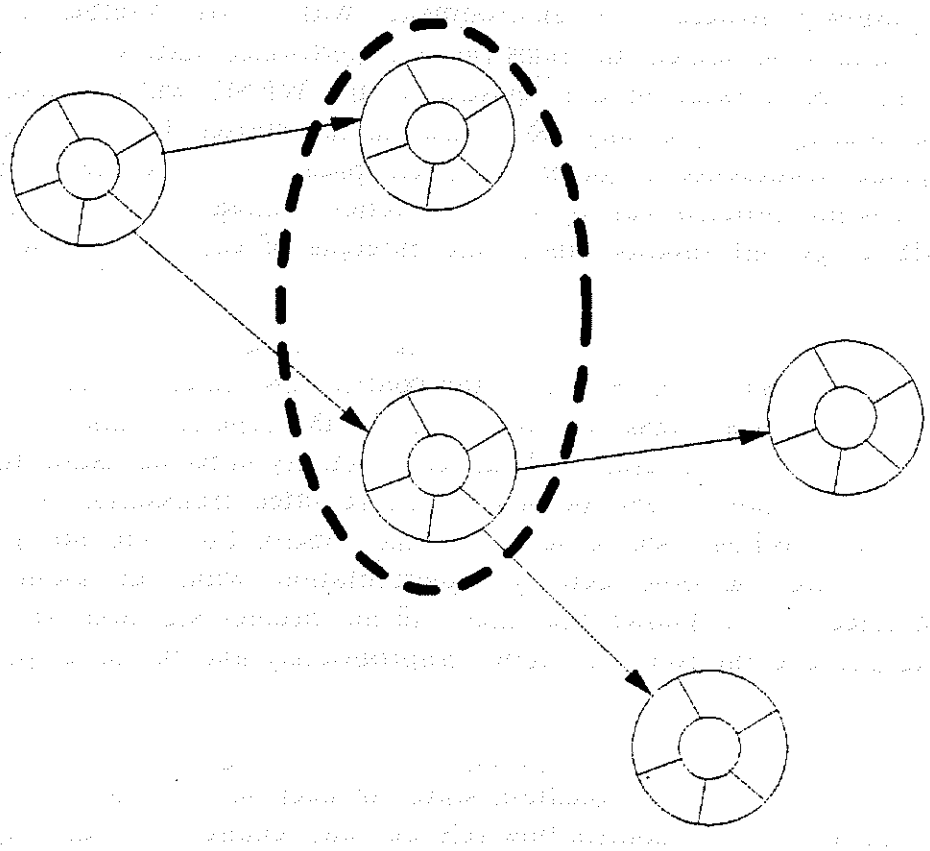


**Figure 6.** ***Atomic Sphere***

At the syntactic level an *atomic sphere* is a set of activities of a process model where each activity itself ensures ACIDicity. Moreover, consider the subgraph induced by these "transactional activities" when representing a process model as a directed graph defining the control flow between activities: All paths between two nodes of this subgraph must already be paths within the subgraph, i.e. it is enforced that the control cannot leave the atomic sphere and enter it again (e.g. the activities collected via the

dashed line in figure 6). The semantics of an atomic sphere requests that either all activities that have run within the atomic sphere committed or all aborted; note that due to different heuristic decisions of two participants this semantics cannot be enforced. At the operational level it is required that each implementation of an activity within an atomic sphere exploits only resource managers in the sense of [27].

As a consequence, the workflow management system appears in different roles within the structure defined in the X/Open DTP reference model:

- As a resource manager participating in the termination protocol of the coordinating transaction manager: The WFMS registers itself as a resource manager whenever it detects that the control flow enters an atomic sphere. This registration is performed dynamically (i.e. via *ax_reg( )* with the TMREGISTER flag set in its **xa_switch_t** structure) in environments with a low number of atomic spheres; in other environment the registration is performed statically (i.e. via configuration file). As a result of this registration the WFMS will be asked by the transaction manager to join any global transaction started from that time on. When a global transaction is joined the participation of the WFMS within the two-phase-commit protocol run by the transaction manager will basically allow the WFMS to get information about the outcome of each activity within the atomic sphere.
- As an application controlling the root transaction which represents the atomic sphere: When the WFMS detects that the control flow enters an atomic sphere the first time it starts a transaction which will be the superior transaction for all activities within the atomic sphere. Thus, each activity implementation that runs within this atomic sphere will become a subordinated transaction to the root transaction. The WFMS will ensure that the control flow will not leave the atomic sphere before all other activity implementations within the atomic sphere have terminated or it is known that they will not become startable. In this case the WFMS will end the root transaction appropriately and the usual processing continues.
- As an application exploiting participating resource managers: The WFMS stores persistent information of the execution status of each instance of every process model in a database. Each control flow relevant state change of an activity implementation and the associated recording of that fact in the WFMS's database system must be performed in an atomic unit. For this purpose, the transaction of the WFMS recording such a state change in its database runs as a subordinated transaction to the activity implementation's transaction (which thus is treated as a superior transaction).

### *4.2 Business Transactions*

Due to the nature of business processes activities are in general long running (especially tolerating system shutdowns), must be thus interruptible, and often externalize

intermediate results. Obviously, the same is true for business processes themselves. Furthermore, a business process contains in general collections of activities which are semantically coupled in the sense that either all coupled activities must be performed successfully or the work associated with the activities must be backed out to allow the business process to continue correctly. In this context the usual transaction models (in general realized via mechanisms like locking etc.) do obviously not apply.

A transaction model that supports such "business transactions" is proposed in [16]: Since the above mentioned semantical coupling of activities is in general not expressed via control flow dependencies or data dependencies we allow arbitrary sets of activities to be coupled into so-called *spheres of joint compensation* (or *compensation spheres* for short). In case a compensation sphere has to be aborted compensation activities are scheduled by the WFMS automatically in an order which is reverse to the order in which the proper activities within the compensation sphere have been executed. For this purpose, activities must be specified as pairs consisting of the implementation of the proper activity and the implementation of the compensation activity.

Many different parameters effect the actual behavior when backing out a compensation sphere (refer to [16] for all the details): For example, it can be specified whether after compensation the work within affected process branch(es) should continue at the entry points of the compensation spheres to be backed out, or whether some administrative actions have to take place, or whether the control flow simply has to continue at the entry points of the compensation spheres without performing any compensation. Furthermore, compensation spheres can become a target of cascading backouts, and backout cannot only be performed "discrete" by running the compensation activities associated with the proper activities but also in an "integral" manner by simply running a compensation activity (which can again be defined as a process model) which is directly associated with the affected compensation sphere itself.

Interesting features result when atomic spheres and compensation spheres are combined. For example, an atomic sphere can also be defined to be a compensation sphere too. When it is detected (e.g. by specifying a 'true' **report_heuristics** parameter of the *commit*-operation of the TERMINATOR interface [2]) that a heuristic decision (which differ from the consensus outcome) has been taken by a participant represented by an activity within the atomic sphere the abortion of the associated compensation sphere can be initiated. Thus, appropriate actions can be immediately and automatically scheduled, for example, the checking for integrity violations, or the cleansing of inconsistent data.

### *4.3 WFMS-Based Transaction Management*

A workflow manager can be perceived as a means to couple applications together

even if they are developed without having the other applications in mind. The common context often needed for such a coupling may be provided by the dataflow facility of the workflow manager especially providing for persistence of the required context information. What results is an application system providing added value to its user when compared with its encompassed discrete applications. For example, the correct sequencing of discrete applications in order to achieve a business goal paying regard of each particular business situation is automatically provided, idiosyncrasies of the various invocation mechanisms are hidden, information is automatically passed from one application to the other etc.

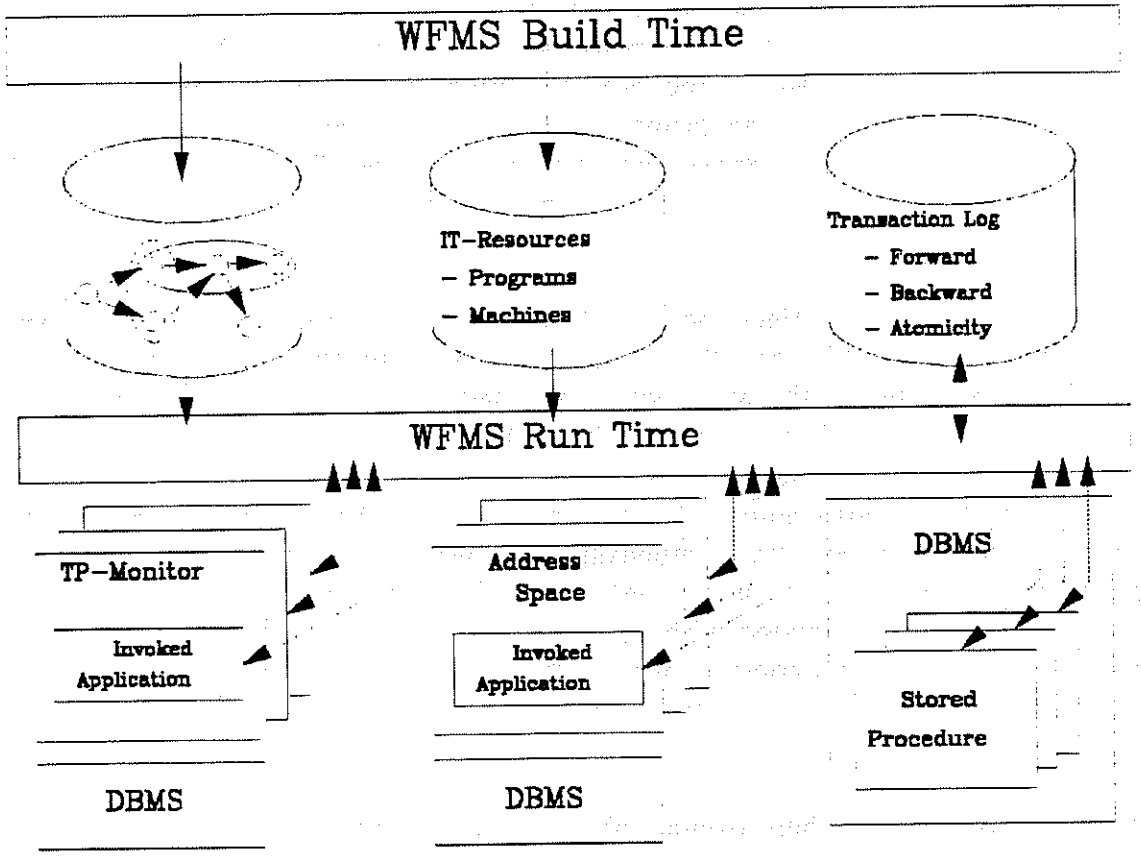



Figure 7. *WFMS-Based Transaction Management*

In addition to the context and state of such an integrated application itself the history of the executed discrete applications, their order, and their local context can be tracked and made persistent. In case erroneous situations occur which require to compensate already finished applications the system can -based on this persistent data- automatically schedule formerly defined "compensation steps" to correct the situation. What results is a feature that supports partial backward recovery introducing a compensation based semantic transaction paradigm for applications integrated via a workflow manager (see 4.2).

Also, a workflow manager can provide a common transactional context for applications with transactional behavior, and can manage and control the atomic outcome of collections of such transactional applications (see 4.1). By this, a workflow manager facilitates a certain flavor of multi-level transactions.

Figure 7 depicts that the coupled applications may run under the control of various transaction monitors, as native operating system programs, or as stored procedures, they may be transactions or non-recoverable units, they may run locally or distributed in heterogeneous environments: The workflow management system ties them together in the sense sketched above revealing the middleware aspects of workflow management. With atomic spheres and compensation spheres workflow management systems can be perceived as the base for future transaction management.

# 5 Summary

We have shown that workflow technology is a powerful vehicle to enable the widespread exploitation and reuse of components: The inherent potentials of workflow technology like the enablement of flow-independence, forward and backward recoverability of workflows, transactional workflows, etc. in conjunction with its embedding in object request broker environments and its exploitation in object oriented analysis and design allow for both, a seamless use of prefabricated components, as well as a seamless build for reuse of components.

**Acknowledgements:** I am grateful to Dieter Roller and Jürgen Uhl for discussing with me many of the subjects covered in this paper.

# 6 References

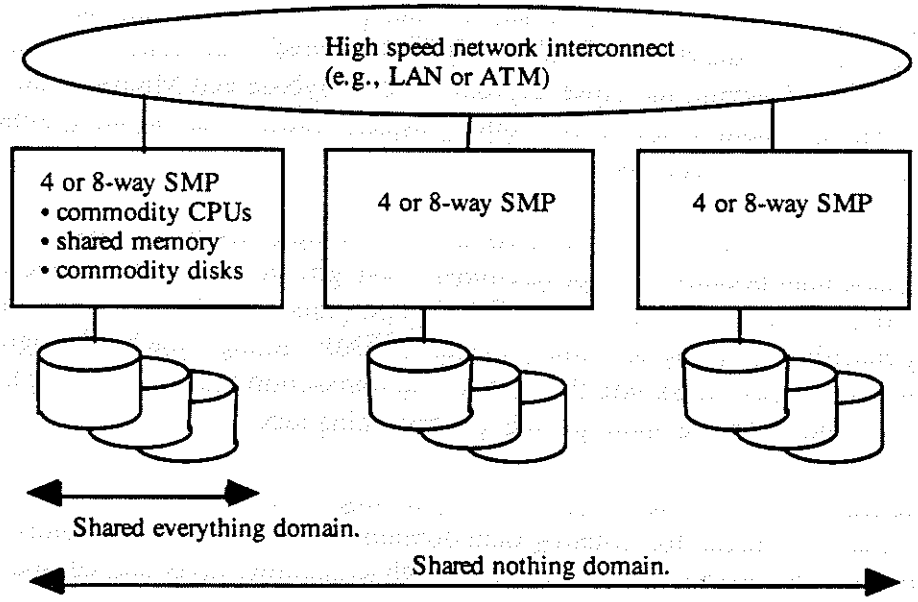
[1] G. Booch, *Object oriented design with applications* (Benjamin/Cummings, 1991).
[2] E.E. Cobb et. al., *Object transaction service,* OMG Document TC 94.8.4 (1994).
[3] E.E. Cobb, *The impact of object technology on commercial transaction processing,* Technical Report, IBM Santa Teresa Lab (1995); submitted for publication.
[4] R. Elmasri, S.B. Navathe, *Fundamentals of database systems* (Benjamin/Cummings, 1989).
[5] R.E. Fairley, *Software engineering concepts* (McGraw-Hill Book Company, 1985).
[6] R. Günthör, S. Jablonski, *Transaction-based application integration in workflow management systems,* Technical Report, University Stuttgart (1994); submitted for publication.
[7] J. Gray, A. Reuter, *Transaction processing: Concepts and techniques* (Morgan Kaufmann Publishers, Inc., 1993).

[8] E. Gamma, R. Helm, R. Johnson, R. Vlissides, *Design patterns: Elements of reusable object-oriented software* (Addison-Wesley Publishing Company, 1995).
[9] *SOMobjects: A practical introduction to SOM and DSOM,* Document number GG24-4357-00 (IBM Corporation, July 1994).
[10] *IBM FlowMark for OS/2,* Document number GH19-8215-01 (IBM Corporation, March 1994).
[11] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard, *Object-oriented software engineering: A use case driven approach* (Addison-Wesley Publishing Company, 1992).
[12] I. Jacobson, M. Ericsson, A. Jacobson, *The object advantage: Business process reengineering with object technology* (Addison-Wesley Publishing Company, 1995).
[13] F. Leymann, *A meta model to support the modelling and execution of processes,* Proc. 11th European Meeting on Cybernetics and Systems Research EMCR92 (Vienna, Austria, April 21-24, 1992), World Scientific 1992, 287 - 294.
[14] F. Leymann, W. Altenhuber, *Managing business processes as information resources,* IBM Systems Journal 33(2) (1994) 326 - 348.
[15] F. Leymann, D. Roller, *Business process management with FlowMark,* Proc. COMPCON Spring 94 (San Francisco, CA, February 28 - March 4, 1994) IEEE Computer Society Press 1994, 230 - 234.
[16] F. Leymann, *Supporting business transactions via partial backward recovery in workflow management systems,* Proc. BTW'95 Databases in Office, Engineering and Science, (Dresden, Germany, March 22-24, 1995), Springer 1995.
[17] F. Leymann, *Transaction concepts for workflow management systems* (in German), in: J.Becker, G.Vossen, Geschäftsprozeßmodellierung und Workflows (Thomson Publishing, 1995).
[18] D. Roller, F. Leymann, *Method and computer system for generating process management computer programs from process models* (Patent Application, January 1995).
[19] D. Roller, F. Leymann, *Methodology to generate C++ programs from Process Models with SOM Objects methods as activities,* (Patent Application, May 1995).
[20] Object Management Group, *The Common Object Request Broker: Architecture and Specification* (OMG, Framingham, MA, 1992).
[21] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorenson, *Object oriented modelling and design* (Prentice Hall, Englewood Cliffs, 1991).
[22] D. Tkach, R. Puttick, *Object technology in application development* (Benjamin/Cummings, 1994).
[23] K. Walk, *Object oriented development of workflow management applications,* Technical report (IBM Austria, 1994).
[24] T.E. White, L. Fischer (ed.), *New tools for new times: The workflow paradigm* (Future Strategies Inc., Book Division, 1994).
[25] R. Wirfs-Brock, B. Wilkerson, L. Wiener, *Designing object oriented software* (Prentice-Hall, 1990).
[26] Workflow Management Coalition, *The workflow reference model,* Document Number TC00-1003, 1994.
[27] X/Open Guide, *Distributed Transaction Processing Reference Model (Version 2),* X/Open Company Ltd., U.K., 1993.

# Shared-Everything versus Shared-Nothing: Why Hybrids Will Win in the Marketplace

**Charles Levine**
Tandem Computers Inc.
10555 Ridgeview Ct., LOC 252-10
Cupertino, CA 95014
levine_charles@tandem.com

While the debate over shared-everything versus shared-nothing goes on, market forces are driving the industry to the realization that hybrids are the answer. Hybrids consisting of shared-nothing collections of commodity SMPs are going to win. These will look like:




The figure above depicts a shared-nothing cluster of SMPs. I refer to this as *course-grained shared-nothing*, in that the shared-nothing components each have substantial CPU, memory, and I/O capacity. By contrast, in *fine-grained shared-nothing* all CPUs are loosely-coupled.

The following factors strongly favor shared-nothing clusters of SMPs:

• **Economic** - SMPs with 4 or 8 CPUs are the sweet spot on the price/performance curve. Ever more of the hardware support necessary to tightly couple CPUs is being integrated directly into commodity microprocessors. In the near future, chip manufacturers will sell SMP modules just like they sell microprocessors today. The economic advantage of SMPs will be overwhelming versus fine-grained shared-nothing.

• **Hardware** - Only off-the-shelf hardware is needed today to do course-grained shared-nothing. Previously, the need for specialized networking hardware increased the entry

cost to build shared-nothing systems. Today, software is the only fundamental differentiator.

• **Scalability** - Shared-nothing scales the best, particularly for BIG systems. Clearly, the ability of SMPs to scale well is improving. The largest SMPs today typically have 8-12 CPUs compared to 4-6 CPUs a few years ago. It appears that this trend toward larger SMPs will continue. Nevertheless, SMPs have inherent bottlenecks which limit their scalability. Today SMPs provide one order of magnitude scalability, whereas shared-nothing has proven two orders of magnitude scalability with the potential to grow well beyond that. When the largest SMP is too small, shared nothing can provide two to three orders of magnitude of further scalability by using SMPs as nodes in a shared nothing cluster.

• **Architecture** - The merchant databases are being redesigned for shared-nothing, usually under the banner of distributed database. Informix, Oracle, and DB2 offer this today; NonStop SQL (Tandem) has since day-one (1987); Sybase and Microsoft are not far behind. The key point is that share-nothing capable DBMSs are running today on your choice of commodity hardware.

• **Programmer convenience** - The SMP model is simpler for the programmer, if for no other reason than because most programmers have grown up with the shared memory model. In practice, it is easier for Joe Cobol to program an SMP than a fine-grained share-nothing machine. In a shared-nothing cluster of SMPs, though, the hard programming problems are pushed down into the DMBS and transaction monitor, leaving the familiar SMP environment for the more prosaic programming tasks.

• **Availability** - Large systems typically require high availability and/or fault tolerance. Shared-nothing is better for isolating fault domains and increasing availability. Consequently, the market requirements for high availability favor shared-nothing.

• **Management domain** - Pure shared-nothing is harder to manage than equivalent shared everything environments because there are more logical components and, consequently, more management domains. The management overhead is proportional to the number of nodes. The overhead is relatively minor in a large system with a few nodes, which favors shared-nothing clusters.

• **Load balancing** - Load balancing is easy in shared everything and hard in fine-grained shared-nothing. The difficulty of load balancing in shared-nothing is proportional to the number of components. In a shared-nothing cluster of SMPs, the number of shared-nothing components is small compared to an equivalent number of fine-grained shared-nothing CPUs. So load balancing is easier using SMPs as the building blocks in a shared-nothing cluster.

# The Case for Log Structuring in Database Systems

David B. Lomet
Microsoft Research
One Microsoft Way, Bldg. 9
Redmond, WA 98055

## 1 Introduction and Background

The notion of a log structured file system (LFS) [6, 9] evolved from earlier efforts using similar techniques [8, 2] as a means to improve write performance of file systems. Other benefits include faster metadata operations, e.g. file create and delete. But there is controversy about the the utility of LFS for database systems, especially in light of the critique in [10]. This position paper argues that LFS has wonderful potential as the underpinning of a database system, solving a number of problems that are known to be quite vexing, and providing some additional important benefits. These include atomic writes, system management and scalability, storage efficiency, and recovery system performance.

There are three inter-related ideas in LFS.

- LFS virtualizes the placement of files on disk. Every write to a file dynamically relocates the data being written. Thus, a write must also update the data structures involved in this relocation mapping.
- Because data is dynamically relocated, it is possible to combine the writes of (logically) widely separated data into a large batch, hence replacing a number of separate smaller writes with a single larger write. Large contiguous file writes can be written contiguously.
- LFS must garbage collects old versions of the data that has been re-written. LFS partitions the physical disk space into a number of large (e.g. 256KB to 2MB) "segments" into which the batched data is written. A relocating garbage collector (CLEANER) reclaims complete segments, hence permitting file activity to continue indefinitely.

In this short position paper, I first list the advantages of LFS system. Then I examine the negatives and discuss how they can either be minimized or avoided. The final section contains some not unexpected conclusions based on this discussion.

## 2 Advantages of LFS

### 2.1 Write Performance

LFS can greatly reduce the I/O accesses needed to support a database. This reduction has a beneficial impact on a number of system performance parameters.

- it reduces the amount of time that a disk spends seeking, and hence improves effective disk bandwidth.
- it reduces the number of executed instructions per page written. LFS batches the writes and hence the number I/O interrupts equals the number of batches (segments) written, not the number of pages written.
- it reduces arm contention on the disk by consolidating the writes.

This performance gain is the traditional and obviously important reason for being interested in LFS. The advantage is not uniquely to database systems.

LFS works particularly well when combined with RAID disks [7]. RAID5 disk systems have a problem supporting OLTP applications that involve a large number of small writes. These single page writes each result in four I/O's, (i) read the old data page, (ii) read the old parity page, (iii) write the new data page, and (vi) write the new parity page. LFS can turn a RAID3 disk array into a RAID5 disk array at no increase in write cost over RAID3. One sets the segment size to be some multiple of the RAID stripe size (could be set equal). When this is done, small writes are batched so that an entire stripe is written (with parity). The small write penalty for RAID5 then disappears.

### 2.1.1 Storage Efficiency

The fact that LFS writes into very large containers(segments) readily permits variable size pages to be stored on disk. By that I mean, the system caches a fixed size page (e.g. of 8KB) for ease of cache mangement and system simplicity, but this can be represented on the disk with a variable size unit within a much larger segment that can contain many such units. The mapping table tells LFS where each page starts, which could be a block boundary, but doesn't have to be.

Variable page sizes on the disk permits storage optimization of two sorts.

**Compression** Compressing data written to disk and uncompressing data read from disk with LFS was suggested in [1]. Compression of 2:1 is commonly achieved in this way, effectively doubling the disk capacity.

**Selective Writing** One need not write an entire page if only part of it is used. For example, B-tree pages are typically only 70% utilized. Writing only the part in use makes the utilization effectively 100%. This might be regarded as a special case of compression, but is especially useful and inexpensive.

## 2.2 Managability and Scalability

Two ways of handling disk storage allocation are extent-based and inode-based. Extent-based file systems usually perform better because of disk page contiguity. Inode-based systems are more flexible in that pre-planning is not usually required, but contiguity is lost. LFS allocation is a hybrid. It is dynamic like inode-based systems. but the granularity, i.e. size of a segment, means that contiguity is frequently preserved. On large contiguous writes, it is indeed preserved. And data written with large granularity can be read with large granularity. So managability of disk storage is very easy, like inode-based systems but with better performance.

In addition. LFS scales uniquely well. The addition of more disk storage immediately means that $I/O$ performance improves because the CLEANER runs less often when the disks are lightly utilized. And, of course, data "flows" to the new disks as the segments are exploited by LFS, without any explicit direction.

## 2.3 Atomic Writes

Database systems need to have high confidence that certain disk writes are either atomic, or have a high probability of being atomic. An atomic write failure in most dbms's requires media recovery which is a very expensive. It requires restoring from a backup at least part of the database, and then rolling that part forward using the media log until it is current. Finding the backup for the invalid part of the database can require operator intervention. Also, the media log is typically MUCH longer than the crash recovery log. The expense of such a failure causes some dbms's to write pages twice, first to a dedicated work area on the disk, then back in-place. That avoids atomicity failures, at the expense of double writing during normal operation.

LFS does not update in place. Rather, when it writes, until after the write is complete, two versions of the data being written are available (essentially, there is a "shadow" version until the write is complete). Should the write fail in the middle, the old version of the data is still ok. Thus, a page(which could be multi-block) or group of pages is either written or not- it is not corrupted by a write failure. This eliminates atomicity failure as a source of media failures.

## 2.4 Recovery and Availability

But there are additional gains from larger units of atomicity. Modern recovery techniques employ physiological logging [3, 5]. These techniques permit logical operations but the operations are restricted to affect exactly

one disk page so that atomic installation of the operation's updates can be assumed. LFS permits multiple discontiguous pages to be installed atomically via its low cost shadowing. This frees recovery systems to log more encompassing operations, and hence to move toward logical operations [4]. This can lead to reduced log size, i.e. one logical operation instead of a number of physiological operations that require the logging of substantial state. It may also lead to greater concurrency because the logical operations have weaker conflict orders compared to read/write operations.

In LFS, all recently written data is present in the recently written segments. This makes rebuilding the actively updated part of the cache very fast as entire segments can be read sequentially into the cache. In addition, pages with high read frequency can also be recovered quickly by writing them from time to time, hence relocating them near the hot update pages. This gives a database system layered on LFS many of the attributes of a main-memory database system with respect to how stability is achieved, and what performance and availability can be expected.

### 2.5 Backup and Archiving

LFS provides at least three advantages when moving data from on-line to near-line storage as is normally done for backup and archiving.

- while the cleaner traverses the file contents reclaiming space, it can also move data to near-line storage. The better cleaners segregate hot from cold data, and could migrate cold data to archival storage.
- recently written data is separated from previous data. This optimizes the performance of incremental backup as only the recent changes need be backed up.
- the mapping index, already present in LFS, permits newly migrated data to be integrated easily with previously migrated data. Hence an existing LFS mechanism provides a framework for both near-line storage and for on-line storage. This mapping index could provide integrated access across the storage media, thus supporting incremental archiving.

## 3 Disadvantages of LFS and what to do about them

### 3.1 Reading Cost

In extent-based systems, contiguous reads are usually done in a single $I/O$ access. Likewise for LFS when read granularity is smaller than or equal to write granularity. This covers most reads, in fact, even ones that appear in "scans", e.g. B-tree scans. However, large granule reads following small granularity writes will not be contigous but will require a number of separate $I/O$ accesses. This may be a problem for decision support and batch applications. Let us look at two usage scenarios and to determine an effective response.

- There are a large number of writes to data between its reads. In this case, one has substantial savings in those writes, which have all been batched. Some of the write performance gain is given back with poor read performance.
- Few writes occur between reads. These writes are sufficient to destroy contiguity but LFS only produces modest savings from these writes. This is the essence of the LFS problem.

One possible response is to simply write back data that has been read when the data request results in a number of discontiguous reads. This restores the contiguity of the data at the granularity of the read. And the cost of the read has already been paid- no separate re-organization cost is incurred for the read, only for the write. Such a strategy will not have much of an impact on the high write-frequency case, but should significantly improve read performance for the low write frequency case.

An additional complication arises when systems plan parallel execution of queries. Careful placement of data reduces I/O access interference among parallel tasks. But careful placement requires planning and can force particular parallel decompositions. Random placement of data does not perform as well as well-tuned careful placement, but can be more robust with respect to dealing with a large number of possible parallel execution strategies. LFS writes approximate random placement of the data. This is frequently a good approach to load balancing of the disks as well.

### 3.2 Cleaning and Mapping

Cleaning and mapping are intrinsic parts of an LFS. These mechanisms are more complex and costly in performance than their counterparts in more traditional file systems. However LFS gains are sufficient to justify these costs.

Analysis confirms that the cost of cleaning is sufficiently modest that it only slightly reduces the gain in write performance unless disk utilization becomes dangerously high (greater than 90%). Only the impact of segment writing and cleaning on read response times might be of some concern. But I/O throughput is clearly improved. Read response time needs to be watched, but in an era when caching becomes increasingly effective because of increased cache size, this should not be a major concern.

The size of the mapping table, which is basically an index to all data on all pages of the file system, can be quite large. The critical parameter is the ratio of mapping table size to file size. For large pages, e.g. 8KB. with a mapping entry of 8 bytes, the ratio of index to file is 1000 to one. This should be tolerable in a well balanced system, given todays prices of 30-50 to one between disk and main memory.

A final issue is recovery of the mapping table when the system crashes. LFS's data placement approximates a sequential log and permits us to identify recent changes in data placement. Using page writes themselves as log records means that the pages of the mapping table can be written relatively infrequently. One treats the mapping table as a database whose transactions are the atomic writes to LFS. That is quite straightforward and very effective. (This is important as a naive implementation would require re-writing the entire path in the mapping index.)

### 3.3 Margo Seltzer's Thesis

Seltzer's dissertation [10] threw some cold water on LFS, suggesting only minor gains for LFS and only under some circumstances. This critique was particularly oriented toward transaction systems, including databases. However, I do not trust her results for a number of reasons, including-

- Her comparison of embedded transactional LFS to user-level transactional system layered on LFS (conventional DBMS style) did not factor in cleaning(the cleaner was not on). The conventional system writes much less data and so cleans much less frequently. She concluded that the embedded system performed better, which I believe is false.
- She then used the embedded LFS for subsequent comparisons with conventional file systems which had been extensively tuned. But, as noted above, the embedded LFS was a poor choice compared to the layering of a conventional DBMS on top of LFS.

## 4 Conclusions

My bottom line is pretty clear. LFS offers not only write performance improvement, but also managability, scalability, and storage efficiency. Perhaps most exciting is its support of larger atomic writes, which can be a boon for improving database recovery- and may permit the merging of log based recovery with an integral-shadowing based scheme.

Finally, for those applications where careful placement of data might have an advantage, it is not impossible to envision an LFS in which some segments are managed using in-place updating. But I would argue that one should first explore whether small optimizations to LFS might reduce those cases to no more than marginal importance.

## References

[1] Burrows, M., Jerian, C., Lampson, B., and Mann, T. On-line data compressin in a log structured file system. Proc. 5th Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems, Boston (1992)

[2] Gait, J. The Optical File Cabinet: A Random-Access File System for Write-Once Optical Disks. IEEE Computer 21,6 (June 1988) 11-22.

[3] Gray, J. and Reuter, A. *Transaction Processing: Conccpts and Techniques* Morgan-Kaufman (1993)

[4] Lomet, D. and Tuttle, M. Redo Recovery after System Crashes. (Feb. 1995) (submitted for publication.)

[5] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., and Schwarz, P. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. on Database Systems 17,1 (March 1992) 94-162.

[6] Ousterhout, J. and Douglis, F. Beating the I/O Bottleneck: A Case for Log-structured File Systems. Technical Report UCB/CSD 88/467 (Oct. 1988).

[7] Patterson, D., Gibson, G., and Katz, R. A Case for Redundant Arrays of Inexpensive Disks (RAID). Proc. of ACM SIGMOD Conf. Chicago, IL (June, 1988) 109-116.

[8] Reed, D. and Svobodova, L. SWALLOW: A Distributed Data Storage System for a Local Network. *Local Networks for Computer Communications*, North Holland (1981) 355-373.

[9] Rosenblum, M. and Ousterhout, J. The Design and Implementation of a Log-Structured File System. ACM TOCS (Feb. 1992)

[10] Seltzer, M. File system Performance and Transaction Support. Ph.D. Dissertation, University of California, Berkeley (1992)

# Contents

**1 Introduction and Background** 1

**2 Advantages of LFS** 1
- 2.1 Write Performance . . . 1
  - 2.1.1 Storage Efficiency . . . 2
- 2.2 Managability and Scalability . . . 2
- 2.3 Atomic Writes . . . 2
- 2.4 Recovery and Availability . . . 2
- 2.5 Backup and Archiving . . . 3

**3 Disadvantages of LFS and what to do about them** 3
- 3.1 Reading Cost . . . 3
- 3.2 Cleaning and Mapping . . . 4
- 3.3 Margo Seltzer's Thesis . . . 4

**4 Conclusions** 4

# Enterprise Client/Server Computing: At the Center of Merging Technologies.

M. Randall MacBlane
Juan M. Andrade
Novell, Inc.
190 River Road
Summit, NJ, 07901

Enterprise computing is dramatically changing in a world where distributed computing is becoming more pervasive. The office is becoming mobile as the network infrastructure becomes more widespread and gets integrated with wireless communications technologies. Desktop, groupware, object, client/server, and TP technologies are converging to create the dynamic environment required for enterprise-wide computing and electronic commerce. Some people may argue that distributed object technology already provides the infrastructure required for this environment but the reality is that this technology is still evolving and far from providing the foundation required for mission-critical applications.

Within an enterprise, client/server computing spans a wide variety of application models from well-defined mission critical applications to groupware style messaging and task scheduling to loosely defined networks of cooperating entities. Distributed object computing (DOC), in particular CORBA and compound documents, is just one component of a computing solution that also includes transaction processing monitor and groupware messaging/task scheduling technologies.

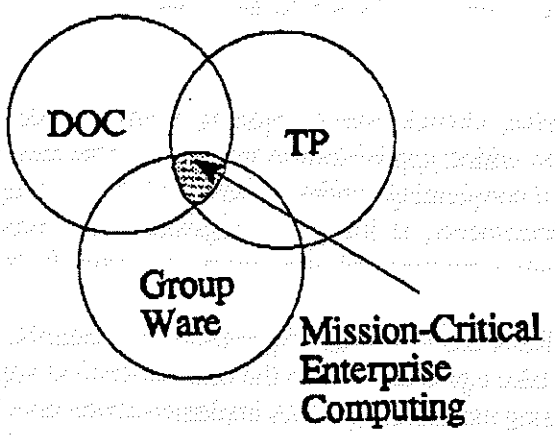
Although the CORBA 2.0 shows good progress in standards, there is still a long road to be paved before ORBs can take a primary role in the mission-critical applications commonly associated with transaction processing monitors. CORBA implementations have been lacking of key TP elements like dynamic load balancing, prioritization, replication of services, fault tolerance and high performance, as well the administrative tools and services designed to provide the control necessary for the support of mission critical applications.

The Object Transaction Service approved as part of the CORBA 2.0 specification is a good step toward the integration of TP concepts with object-oriented concepts. The Object Transaction Service allows applications to be designed using transactions to insure robustness and failure control. However, there is much more than just transactions. ORBs should be able to scale up and deal with thousands of clients and objects to provide a performance of transactions per second. They should be able to provide an administrative environment that controls the deployment of the services provided by objects, prioritize requests, insure high availability by allowing replication, improve in routing techniques, and provide fault tolerance.

However, neither TP monitors nor distributed object technologies are the complete answers to enterprise computing. Groupware technology is also a key component. This technology includes store-and-forward messaging, scheduling, task management and loosely coupled non-performance critical client/server capabilities. Integration of compound document technology with Groupware will create "live" documents that evolve as they migrate from desktop to desktop. The integration of Groupware workflow technologies with the transactional queuing and event management provided by TP monitors can create the environment required for reliable delivery in electronic commerce.

Workflow is likely to become the basic model for mobile enterprise computing.

In dealing with company boundaries, electronic mail still plays an important role in the daily business of an enterprise. Reliable and secure business transactions must be possible via this communications technology. Communications technologies coupled with visualization techniques and compound document technology will play a fundamental role in electronic commerce over the Internet. The technologies provided by Mosaic and Netscape should evolve to become more integrated with the enterprise computing. Catalog browsing, or orders initiated from desktops may feed the TP monitor or workflow subsystem in charge of automatically processing these requests.




There are components of enterprise client/server computing that are supported by each of the three application models discussed; distributed object computing, transaction processing monitors and groupware. The intersections of these technologies provide the reliable and flexible environment required by enterprise and inter-enterprise computing. The move toward "business objects" will be fully possible when products offering this integrated environment appear in the marketplace.

*References:*

1. Object Management Group, The Common Object Request Broker: Architecture and Specification, version 1.2, 1993.

2. Object Management Group, CORBA Services: Common Object Services, Draft, March 1995.

# Large-Scale SMPs: No End to the Parallelism Debates

## *A Position Paper*

John McPherson
Almaden Research Center
650 Harry Road
San Jose, CA 95120
johnm@almaden.ibm.com

## *Introduction*

In recent years, many RISC hardware vendors have introduced symmetric multiprocessor (SMP) systems. Today, 4-way to 12-way systems are common either in real products, or in announced product plans, with larger systems also available from some vendors. There is much talk, however, of much larger systems with 100's of processors from the hardware architecture community. These systems, offering cache coherence access to any location in common address spaces across all processors, will have non-uniform memory access (NUMA) characteristics such that access to data physically closer to the processor doing the accessing is faster, by an order of magnitude or more, than to data that is stored "near" other processors.

The promise of the hardware architects is that these NUMA machines provide the same programming model as much smaller SMPs, so will be much easier to develop applications on than either the shared disk, or shared nothing machines currently available. Many people are starting to argue that in the long run, SMP hardware architectures will dominate the industry. However, the non-uniformity in memory access means that database developers cannot treat large scale NUMA machines the same way they treat SMPs. I believe we cannot simply have a single shared buffer pool that is shared by all processors, but must consider multiple buffer pools to maintain locality. This means that shared everything database architecture may not be appropriate and either a shared disk or shared nothing database architecture may be more appropriate. SMPs may end the hardware debate (thought I doubt it), but I do not believe they will end the database architecture debate - they will only start a new round of debate about the "best" parallel database architecture.

## *The Database Architectures*

The database community has settled on a taxonomy of parallel database architectures consisting of share everything (SE), shared disk (SD), and shared nothing (SN) configurations. SE databases have a single buffer pool, lock table structures, and uniform access to all disks in the system by all database processes. SD databases have multiple buffer pools, usually one for each processor, but each process can share data from all the disks. Thus a disk page can reside in any one of the buffer pools, and there may be multiple copies. SD architectures must deal with cache consistency, and locks must be obtained at a global level (with optimizations for local acquisition of locks in many cases). The SN architecture also has multiple buffer pools, again, usually one per processor, but the SN architecture also partitions the disks so that each disk page can only appear in a single buffer pool. This avoids the need for cache consistency protocols, and it also means that most locks can

be obtained at a single processor. One must be careful to separate parallel database architectures from machine architectures, since, for example we can simulate shared disk architectures on shared nothing hardware.

Mapping an SE parallel database to a NUMA machine with a single buffer pool, single lock table, single set of buffer pool control bocks, etc. will certainly produce a functionally correct database solution. However, performance will be disasterous for every shared structure if the hardware is busy maintaining consistency of shared processor cache lines. The first step in tuning such a system would be to logically partition the data and database processes, so that database data and processes execute together on the same processor. One has to make the immediate decision about what to do when a process needs data in a non-local partition. Does one do function shipping or data shipping? Also, this may alleviate contention for buffer pool pages, but one must redesign buffer pool control blocks, for example, so that an operations such as fixing a page which may require searching buffer pool control block hash chains, does not inadvertantly result in walking across control blocks that are not local to the processor. This will happen if the function used to partition the data is not the same as used by the buffer pool to hash buffers containing disk pages to buffer pool hash chains (its unlikely the mappings would be the same.).

Tuning a SE parallel database architecture for a NUMA machine will require considering every shared control block in the system, and invention of many new techniques to partition these control blocks. Modeling, simulation, and prototyping will be needed to determine the correct tradeoff between function shipping and data shipping, and many other aspects of the design. I believe it is likely that a conclusion that will be drawn is that it will be easier to map a SD or SN parallel database architecture to NUMA machines to get good scaleup. It is likely that NUMA machines will be implemented with local clusters of 2 - 6 processors that behave like today's SMPs with uniform memory access. In this case, the preferred database architecture will be a hybrid solution with SE parallel database technology used for these clusters and either SD or SN technology used between the clusters.

Regardless of the parallel database design decision that is ultimately made, I believe NUMA machines will not live up to the promise of providing a simple programming model to the database system implementor. Performance is too important to ignore orders of magnitude differences in access time to various memory locations. The debates about shared everything, versus shared disk, versus shared nothing will not end, they will just take on a new dimension.

## *About the Author*

John McPherson is the Manager of the Exploratory Database Systems Department at the IBM Almaden Research Center. The department is engaged in a variety of exploratory and applied research activities. We work very closely with IBM's Software Solutions Division on the delivery of Version 2 of DB2 for AIX, OS/2, and other platforms. We are developing data mining technology to discover patterns in huge masses of data, we are exploring new ways to provide innovative query and browse capabilities to diverse, heterogeneous repositories in an enterprise or across the internet, we are exploring new transaction paradigms for use in workflow products, and we are investigating many other exploratory database problems. Before managing the Exploratory Database Systems Department, John worked on a parallel database prototype, he designed and implemented the buffer pool manager, data manager, and query evaluation system in the Starburst extensible database system, and he worked on a high performance communications hardware prototype. John obtained a Ph.D from the University of Wisconsin - Madison.

# An Overview of the Exotica Research Project on Workflow Management Systems*

C. Mohan G. Alonso R. Günthör M. Kamath B. Reinwald

IBM Almaden Research Center, San Jose, CA 95120, USA
*http://www.almaden.ibm.com/cs/reports/workflow/exotica_home_page.html*
*mohan@almaden.ibm.com, alonso@inf.ethz.ch*
*Roger.Guenthoer@informatik.uni-stuttgart.de, kamath@freya.cs.umass.edu*
*reinwald@almaden.ibm.com*

## 1 Introduction

Workflow is probably one of the most exciting areas of research that has emerged in the past few years. Workflow concepts and ideas have been around in one form or another for a long time: *computer supported cooperative work*, *forms processing*, *cooperative systems*, *office automation*, etc. However, only recently the technology and know-how required to implement commercial systems have become available. In general, workflow management systems (WFMSs) are used to coordinate and streamline *business processes*. These business processes are represented as *workflows*, i.e., computerized models of the business processes [13], which specify the individual *activity* steps, the order and the conditions in which the activities must be executed, the flow of data between activities, the users responsible for the execution of the activities, the tools to use with each activity, etc. A WFMS is the set of tools that allow the design and definition of workflows, their instantiation and controlled execution, and the coordination and integration of heterogeneous applications within the same workflow [13]. Users interact with a WFMS by accessing their individual *worklists*, where they can find the activities for which they are responsible without necessarily being aware of the higher level processes to which the activities belong. A crucial point to understand workflow management systems is their dependency on a variety of technologies, from databases to distributed processing. When designing a WFMS, the challenge is to build a feasible and common working environment in which all these technologies are integrated in a flexible and easy to use fashion. This integration aspect is precisely what has made WFMSs so elusive to date and explains the limited success of earlier attempts to provide support for cooperative work. With the first generation of commercial WFMSs [12], there are still many integration issues that remain to be solved, but there is no doubt that in the future WFMSs will be pervasive in large corporations.

In this paper, we present the Exotica research project currently in progress at the IBM Almaden Research Center. One of the goals of the project is to bring together industrial trends and research issues in the workflow area. It is for this reason that we have focused on a particular commercial product, *FlowMark*, IBM's workflow product [15, 16, 19, 20]. However, our results are easily generalized to other WFMSs since FlowMark's model is similar to that proposed by the Workflow Management Coalition [13]. In particular, the rest of this paper contains a high-level overview of our research in six specific areas that are not product specific. The list of these areas is not, by any means, exhaustive. There are still many issues that remain open. We also discuss the relationship between WFMSs and transaction processing monitors.

*This work is partially supported by funds from IBM Hursley (Networking Software Division), and IBM Boeblingen and Vienna (Software Solutions Division). Even though we refer to specific IBM products in this paper, no conclusions should be drawn about future IBM product plans based on this paper's contents. The opinions expressed here are our own.

# 2 Large Scale Workflow Systems

Before discussing workflow management, it is necessary to put in perspective the applications addressed by such systems. Some of the common goals of a WFMS are to achieve better performance of business processes, better quality, enhanced effectiveness, enterprise wide coordination and monitoring, etc. As we believe it is the case with most WFMSs, FlowMark addresses the coordination of *large* scale business processes. With respect to this, there are certain similarities with DBMSs. Small DBMSs for personal computers have an undeniable value. However, interesting research and industrial strength products relate to issues such as query optimization, performance, data caching, data mining, schema evolution, or triggers, which only become interesting in large DBMS scenarios. For small applications, it is possible to obtain similar or even more benefits by using a less sophisticated cooperative tool rather than a full-fledged WFMS. However, when the size of the application grows, and the number of users, sites, and processes increases, only a WFMS is suitable for the task. It remains to be determined how "large" is a large application. To give an idea of the magnitude of the problem, consider some of the studies that have been done with FlowMark. In one instance, the number of business processes to be executed concurrently during the course of a month is 300,000. In another case, the problem is to link together more than 4,000 sites geographically distributed over an entire country and coordinate the work taking place over heterogeneous platforms at those sites. A third example involves more than 100,000 users working concurrently on the completion of processes. With these figures, issues that were not previously considered as related to workflow become key aspects to the success of a commercial system. Failure handling, continuous availability, navigational flexibility, and replication, to name a few, are no longer nice features but crucial components of the overall system. It is within this framework that our research is inscribed.

# 3 Research Issues in Workflow Management

The Exotica project started out focussing on six major research areas. Each of them reflects a distinctive need for WFMSs but, in most cases, they are related to each other and successful solutions will have to integrate aspects from all of them. The list of these areas is by no means exhaustive, there are many other issues that remain open.

## 3.1 Failure Resilience in Distributed WFMSs

Given its goals, the architecture of a WFMS is necessarily distributed in the sense that its components will reside in different and heterogeneous machines. Furthermore, given its relevance in the control of the business processes of a corporation, the system must be continuously available. Hence, each component must be able to deal with local and communication failures, and the design must be robust enough to avoid stopping the execution of processes even in the event of failures. Such requirements define a wide range of failure handling capabilities that the system must support to be commercially viable. Part of our research has been to analyze the possible failure scenarios and design methods to handle them [1, 2, 18].

In the first place, the execution of an activity within a process involves several components: the database server where the workflow-process-related data is stored, the workflow server that determines where the activity is going to be executed, the client interacting with the user, and the client interacting with the actual application performing the activity. All of these components interchange information before, during and after the execution of the activity. This information should not be lost, since this may imply that the results of the execution are not recorded, or that the command to start executing the activity never reaches its destination. Hence, some form of *coordination protocol* must be used amongst all these components to guarantee that all agree on the status of a particular activity. In general, the DBMS in which the data is stored is transaction based. In FlowMark, for instance, the data is stored in ObjectStore, an object-oriented DBMS. ObjectStore does not provide either *hot standby* support or a *prepare-to-commit* interface, which makes it complicated to reach a point in the execution where data is guaranteed not to be lost. Moreover, the applications invoked by the WFMS may not be aware of the WFMS's existence, thereby becoming uncooperative and committing work regardless of the state of the WFMS. We are studying several approaches to these problems: using a persistent message mechanism, providing stable storage in each of the components, and designing a handshake protocol that ensures that information is almost never lost.

A second aspect of failure resilience is the possibility that a component fails while it is executing part of a process. If the failed component is the DBMS, replication can be used to minimize the impact of the failure, as discussed below. If the failed component is part of the WFMS, means must be devised to allow other components to reroute their operations to components that are still available. This implies using multiple connections among components. We have designed a new architecture for WFMSs in which the notion of clusters of servers is used to provide enhanced availability of the system [2, 18]. By using several clusters, each of them attached to a separate database, the impact of failures on overall performance and availability can be reduced. Since each cluster's database contains the same schema information such as process templates, role and staff assignments, and organization definitions, process instances can be run at any cluster. Thus, failures will affect only a subset of the process instances and will not prevent the users from starting new instances. The problem that still remains is how to handle the instances that were running in a failed cluster. This particular issue is addressed below when discussing replication in WFMSs.

## 3.2 Compensation and Navigation in Workflow Networks

A workflow network, or a process, is the representation of a business process. Its instantiation for execution is a process instance. Any process instance usually encompasses a wide range of activities and those activities involve several participants in the form of computer programs and individuals. At any given moment in time, there may be several instances of the same process being executed simultaneously, with possibly each one being at a different stage of progress. Since processes are defined in advance, it is not possible to evaluate all possible exceptions and error conditions beforehand. A WFMS must provide means to cope with such situations. Similar ideas are found in advanced transaction models [9] such as Sagas [10, 11] for backward recovery, i.e., how to undo the effects of certain operations, or in Flex transactions [7, 22, 31] for forward recovery, i.e., how to select different paths of execution. However, in WFMSs, there are certain errors that may force the user to abort the execution of a process instance, and therefore to lose work that has to be performed again when the process is restarted.

Forward recovery is guaranteed in FlowMark in the sense that if there are failures, the process is guaranteed to make progress. However, semantic failures are more difficult to address, unless the designer of the process was able to foresee them and introduce the appropriate checks in the flow of control. For backward recovery, FlowMark is being enhanced to provide a form of compensation that allows the user to specify *spheres of compensation* [21] to determine the scope and extent of compensation in case of failures.

Our current research is focused on how to provide a flexible mechanism to navigate through the flow of control of a process in either direction, i.e., forward progress or compensation, and allowing to switch directions several times as necessary during the execution of a process.

## 3.3 High Availability through Replication

High availability is a key requirement of a WFMS. Failures should be transparent to the users and should have minimal impact on the normal functioning of the organization. This can only be achieved through replication of the process instance information. As in DBMSs [25], we use a primary/backup architecture in which process instances run on a primary server, while all actions are also recorded at the backup so that if the primary were to fail, then the backup server can take over the execution of those same instances. As in DBMSs, replication has a high cost in terms of synchronization and lower throughput. Given the large number of processes involved in the application, replication of every single process instance may not be cost effective. Hence, we define three priority levels with respect to replication. A *hot stand-by* process is fully replicated, i.e., all changes to its data are performed both in the primary and in the backup databases. A *cold stand-by* process is also replicated, but the backup contains only the messages with the information regarding the different steps taken in the execution of the process. When the backup needs to take over the execution of the process, it must first replay those messages to bring the process state up-to-date. This will delay resuming the execution but it reduces the overhead of maintaining the replicated copies. Finally, a *normal* process is not replicated at all. The only guarantee FlowMark provides for these processes is forward recovery, i.e., when the server comes up again, execution will be resumed from where it was left off.

As in DBMSs, besides the replication scheme, it is also necessary to deal with additional problems such as dynamic configuration of the system, incorporation and exclusion of new servers without interrupting the system,

message duplication, and so forth. We have addressed all these issues and proposed a replication mechanism for WFMSs that greatly improves the availability of the system while avoiding excessive overhead [18].

## 3.4 Mobile Computing

WFMSs must coordinate users distributed over a wide geographic area. The basic idea being that the processes to be executed are defined and controlled in a centralized server, while the users can execute parts of these processes at remote clients. Each step of a process can be executed anywhere in the system, but after its completion, the results are communicated to the server which in turns prepares the next steps for execution. This is the most common approach in existing systems, and simplifies many problems such as synchronization, concurrency and monitoring, but forces the users to remain *connected to the server* while performing multiple tasks. An increasingly large set of users may not be satisfied with this mode of operation, given the widespread use of portable computers and home computers for office work.

For these users, the common way of operation is to load data in their laptops or desktops by briefly connecting with a server in the office. Then they disconnect from the server, work on that data and, after a few hours or few days, reconnect to the server to transfer the results of their work. This has been identified as one of the most common modes of computing that will occur in the future [17]. Note that this includes not only mobile terminals but also desktops or any other computer type connected to the server only occasionally via a modem. Mobile or nomadic computing greatly expands the scope of an organization's distributed computing infrastructure. However, from the workflow management point of view, it becomes more difficult to coordinate the work of many users. Also, note that while WFMSs are tools for cooperation, portable computers are generally viewed as tools for individual work.

To address this problem, we have proposed a design, called Exotica/FMDC, for disconnected client operation based on FlowMark [3]. Among other constraints, the design had to provide support for disconnected clients with minimal changes to the semantics of a business process and its implementation, effectively allowing users distributed over a wide geographic area and working with heterogeneous resources to cooperate, while preserving their mobility and independence. In particular, we define the semantics of loading work to a mobile, disconnectable computer by introducing the notion of *locked activities* and the *user's commitment* to eventually execute them. Locked activities stay within the user's machine unless they are unlocked. The feasibility of our approach was initially proven by outlining the implementation issues for FlowMark. We have more recently completed prototyping Exotica/FMDC.

## 3.5 Distributed Coordination

To further enhance the availability and failure resiliency of a WFMS, it is possible to design it entirely as a distributed system. This implies that processes are transferred from site to site as their execution progresses, without a centralized server keeping all the relevant information. There are two possible approaches. One is to make a large package that contains all the information pertaining to the process and its execution and circulate this package across the different sites. This approach has several problems, mainly the size of the message and the need to maintain multiple copies of it if the system were to allow concurrent activities to take place. The other solution is to precompile the process definition to determine at which sites the different activities are to be executed. Once this has been done, the only information that gets transferred from site to site are the results of previous computations and the process state. Using this approach, failures can be made transparent by rerouting a process to different nodes. It also eliminates the potential bottleneck created by a centralized database and server. Exotica/FMQM [1] proposes a similar architecture in which persistent storage is replaced by persistent messaging. For persistent messaging, IBM has defined an application programming interface (API) standard called Message Queue Interface (MQI) [14], and a family of products called MQSeries that supports MQI [23]. MQSeries products operate on IBM and non-IBM platforms and they support the architected MQI. Communication takes place through named queues that do not require all participating programs to be available, i.e., up and running, simultaneously. Moreover, MQI is not sensitive to network transport protocol differences. The system we propose, *Exotica/FMQM*, FlowMark on Message Queue Manager, is a distributed WFMS in which a set of autonomous nodes cooperate to complete the execution of a process. Each site functions independently of the rest of the system, the only interactions between nodes being conducted through persistent messages which inform about activity completions. This approach, while enhancing the availability and resilience of the system,

has the problem of needing more complex mechanisms to monitor and audit the overall execution since there is no centralized server where this information resides. It also requires enhancements to the MQI to be able to assign an activity to more than one user but let only one of the users actually execute it.

### 3.6 Advanced Transaction Models

The goal of most advanced transaction models is to eliminate the constraints imposed by conventional DBMSs on new applications. The requirements of such new applications are completely different from the traditional data processing applications targeted by standard DBMSs. Hence, conventional DBMSs are unsuitable in these new environments. Several advanced transaction models have been proposed [9] but, to this date, most remain as theoretical constructs which have not been implemented. A reason for this state of affairs could be that advanced transaction models are ahead of the available technology and their time is yet to come. We believe, however, that the reason why these models are not being implemented has to do more with their inadequacy to operate in real working environments than with the available technology or application demands. Advanced transaction models are too database-centric, which provides a nice theoretical framework to work with but limits the possibilities and flexibility of the models. Since they tend to remain theoretical models with no implementation, there are a number of important design issues that are generally not discussed in the literature [24].

Paradoxically, WFMSs are tools to support distributed, heterogeneous environments very similar to those targeted by transaction models. However, the models being used are *workflow* models instead of transaction models. Compare the more than 70 vendors who claim to have WFMSs [12] with the almost absolute lack of commercial products supporting advanced transaction models. Workflow systems bear a strong resemblance to advanced transaction models both in their goals and modeling approach, yet they are quite different in that they address a much richer set of requirements than existing transaction models. As part of our research, we have analyzed the characteristics of workflow models and the notion of business processes by comparing them with existing transaction models [2, 4]. It is possible to show that the semantics of workflow models are, in general, richer than those of advanced transaction models and more apt to be used in commercial products. Hence, workflow models can be used to implement advanced transaction models. For instance, we have used FlowMark to implement Sagas [10] and Flexible Transactions [31]. Our basic goal was to provide synergy between advanced transaction models and workflow models, an interaction from which both sides may benefit. In doing so, we have developed a better understanding of the inherent limitations of the so-called "advanced models" and identified many points for improvement of WFMSs. Our approach is not another attempt to merge different transaction models into one or to provide a general framework to program advanced transactions [5]. Transactions are used as the accepted currency in the database community, but our goal is to show the expressibility and power of workflow models compared with what is currently available in the research literature [8].

From a purely advanced transactions point of view, our proposal differs from previous ones in that we use an existing commercial product that runs across different platforms to implement different transaction models. This allows us to draw conclusions from direct experience implementing such models in real, working environments. We see workflow models as the the next step after advanced transaction models. They complement many aspects of the latter and address an entirely new range of issues (role and staff assignment and resolution, worklist management, interaction with manual activities, etc.) that make them more suitable for building applications. As has been widely recognized, there is a lack of tools to scale from individual transactions to complex applications [30]. One reason being that advanced transaction models will never reach their technological maturity until they are interpreted in a much broader context. As we show in our research, business processes constitute such a context and workflow models are the tools required to support complex applications.

## 4 TP Monitors and Workflow Systems

A WFMS is a tool to control the execution of business processes based on representations of the organizational, informational, and functional aspects of an enterprise. In its origins, closely tied to *office automation*, workflow management was geared towards the system-enforced organization, processing, and automation of well structured cooperative work. However, automation efforts were, at the time, limited by the lack of appropriate technology. Nowadays, advances in the state of the art, such as new desktop operating systems or more sophisticated DBMSs and networks, have allowed the incorporation of many new technologies into the workplace. The result has been the partition of the enterprise into *islands of automation* that do not necessarily communicate with each other.

In this situation, WFMSs are seen as the key element to bring together all these technologies and provide an integrated cooperative environment in large organizations. This would explain the interest aroused and successes enjoyed by the first generation of commercial WFMSs.

Transaction processing monitors, TPMs, on the other hand, are a well established technology that has been around for almost 20 years [6]. The first generation of TPMs, e.g., IMS/DC and CICS, were single monolithic systems used in proprietary mainframe environments to achieve high transaction rates in applications such as airline reservations or finance handling. The second generation of monitors, e.g., ACMS, Encina, and CICS/6000, are more open and able to talk to heterogeneous resource managers under certain restrictions (such as support for standard *commit protocols*, for instance). In general terms, TPMs are very mature with regard to concurrency control and recovery and have solid foundations in the notion of *ACID transactions* as well as in high-end load balancing and failure handling.

In spite of addressing similar problems, the characteristics of WFMSs and those of TPMs are almost complementary. Workflow management lacks a clear transactional concept, and has problems with scalability, reliability, and failure handling. It provides, however, modelling of inter-application relationships (i.e., business process modelling), and persistent forward execution. WFMSs permit the invocation of applications operating in different execution environments with parameter passing between them.

TPMs are essentially specialized operating systems. Applications execute in the context provided by such environments. This makes it very difficult, if not impossible, to let applications of one TPM type invoke applications of another TPM type. WFMSs, on the other hand, are not like operating systems. Notions well known in the transaction processing area like backward recovery, atomicity, or compensation are non-existent in current WFMSs. TPMs provide some of these capabilities since they are especially designed for reliability, but they lack the notion of persistent forward execution, and the ability to coordinate the execution of related but independent activities. While the Workflow Management Coalition is proposing standard APIs for interoperability of WFMSs, no such standard APIs exist for TPMs.

In recent years, the deficiencies of the ACID transaction model have lead to the proposal of numerous *advanced transaction models* that extend or relax the ACID properties of classical transactions [9]. Common to most transaction models is the idea of supporting additional capabilities to control the interactions of groups of activities and provide relaxed ACID properties to groups of transactions instead of to individual transactions.

It is our belief that the existing ideas in the field of advanced transaction models will be implemented in WFMSs rather than in TPMs. The goals for the next generation of both WFMSs and TPMs are very similar. Since TPMs already support ACID transaction, the question arises whether TPMs are not better suited to meet the customers' requirements for advanced transaction models. However, it is more likely that, if advanced transaction models are finally going to make it as a viable commercial technology, this will happen in the workflow arena for, among others, the following reasons:

- WFMSs are designed to operate in modern information environments: distributed, client/server, heterogeneous, and with multiple applications.
- New and existing applications can be easily incorporated into a workflow environment without major modifications to either the workflow system or the application itself.
- WFMSs provide multiple key features that do not exist in other systems: coordination facilities, monitoring and auditing capabilities, abilities to reflect the organizational structure, and high level programming languages for the definition of processes.
- WFMSs incorporate users into the system. This is a first step towards a cooperative tool. Moreover, WFMSs provide support for defining individual users, grouping them into logical entities (roles) and assigning activities to particular users or roles.
- WFMSs reflect the organizational model and work environment in which they are used. They can be easily tailored for different environments.

However, with regard to advanced transaction models, the implementation of such models in WFMSs is not straightforward. There are many areas open to research. Some of the issues that need to be addressed are the following:

- Current WFMSs lack facilities to extend the ACID properties to groups of activities. The semantics of many operations such as recovery, rollback, compensation and undo are still undefined in WFMSs. There are many possibilities, but it is not trivial to evaluate their advantages and drawbacks.
- Legacy applications are here to stay. Better means must be devised to incorporate them in a workflow environment. As these applications need to be treated as blackbox applications, specialized *wrappers* are required to interoperate with existing applications.
- Although it is much used, the notion of *transactional workflows* [29] is still unclear. There is a wide gap between research advocating transactional workflow and what is being implemented in commercial systems. It is not clear, for instance, what are the requirements of WFMSs in terms of concurrency control and recovery. The *Workflow Management Coalition*, on the other hand, seems to be reluctant to incorporate the notion of transactional processing in its standards. This might be a problem of finding a common language.
- There is no single advanced transaction model that meets the requirements of all customers and applications. WFMSs will incorporate concepts rather than particular implementations: savepoints, nesting, coordination, compensation, invariants, dependencies, etc. and combine them into more flexible constructs. However, the exact semantics of such constructs are yet to be defined.
- There are many options to define the failure semantics for complex workflow processes. What should happen to a parallel branch if one branch fails? What should happen to the results of a sequence of activities which need to be reexecuted? Are they compensated for and in which order? How can a specific order of compensation steps be specified? All these questions need to be solved before solutions can be proposed.
- The workflow definition language is likely to end up providing a large set of primitives to specify transactional behavior. This set of primitives will be much richer, more complicated and clumsier than the ACID transaction primitives.

# 5 Conclusions and Future Work

The importance of WFMSs in large corporations cannot be stressed sufficiently. There is a high demand for such systems, but successful solutions will have to integrate many technologies and approaches. The challenge lies in providing systems capable of dealing with very large, heterogeneous, distributed and legacy applications, while providing an acceptable degree of reliability and availability. The research areas briefly discussed in this paper point towards some of the issues that need special attention in the design of WFMSs. These issues apply to the vast majority of existing systems, since they are not constraints imposed by a particular architecture but generic demands of the working environment targeted by these systems. In our case, we have focused on FlowMark since this is an IBM product and we have access to its code and developers. However, the solutions we proposed can be easily extended to other systems given that FlowMark's model (and architecture) closely resembles the generic model proposed as part of the Workflow Management Coalition standarization effort [13].

With the recent merger of IBM and Lotus, we have started exploring how to establish more synergy between Lotus Notes and FlowMark, taking advantage of the strengths of the two products. Some of the issues under consideration are: document-centric versus process-oriented workflow, structured versus ad hoc workflows, object orientation, external event handling, agent technologies, etc. Since the *send* model of workflow in Lotus Notes is similar in many ways to distributed workflows as we have discussed here, we have recognized its limitations based on our Exotica/FMQM work. We are in the process of doing further analyses.

**Acknowledgements** Part of this work has been done in collaboration with Amr El Abbadi and Divyakant Agrawal of the University of California at Santa Barbara while they were visiting IBM Almaden Research Center.

# References

[1] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, R. Günthör, M. Kamath, *Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management*, **Proc. IFIP WG8.1 Working**

**Conference on Information System Development for Decentralised Organizations**, Trondheim, August 1995. Also available as **IBM Research Report RJ9912**, IBM Almaden Research Center, November 1994.

[2] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör, C. Mohan, *Failure Handling in Large Scale Workflow Management Systems*, **IBM Research Report RJ9913**, IBM Almaden Research Center, November 1994. See also http://www.almaden.ibm.com/cs/reports/workflow/exotica_papers.html

[3] G. Alonso, R. Günthör, M. Kamath, D. Agrawal, A. El Abbadi, C. Mohan, *Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System*, **Proc. 3rd International Conference on Cooperative Information Systems**, Vienna, May 1995.

[4] Gustavo Alonso, Divy Agrawal, Amr El Abbadi, Mohan Kamath, Roger Günthör, C. Mohan, *Advanced Transaction Models in Workflow Contexts*, **IBM Research Report RJ9970**, IBM Almaden Research Center, July 1995. See also http://www.almaden.ibm.com/cs/reports/workflow/exotica_home_page.html

[5] A. Biliris, S. Dar, N. Gehani, H.V. Jagadish, K. Ramamritham, *ASSET: A System for Supporting Extended Transactions*, **Proc. ACM SIGMOD International Conference on Management of Data**, Minneapolis, May 1994, pp. 44-54.

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

[7] A.K. Elmagarmid, Y. Leu, W. Litwin, M.E. Rusinkiewicz, *A Multidatabase Transaction Model for Interbase*, **Proc. 16th International Conference on Very Large Data Bases**, August 1990.

[8] D. Georgakopoulos, M. Hornick, A. Sheth, *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*. **Distributed and Parallel Databases**, Vol. 3, No. 2, pages 119–153, April 1995.

[9] A.K. Elmagarmid (Ed.), *Transaction Models for Advanced Database Applications*, Morgan-Kaufmann, 1992.

[10] H. García-Molina, K. Salem, *Sagas*, **Proc. SIGMOD International Conference on Management of Data**, San Francisco, May 1987.

[11] H. García-Molina, D. Gawlick, J. Klein, K. Kleissner, K. Salem, *Coordinating Multi-transaction Activities*, **Proc. IEEE Spring Compcon**, San Francisco, 1991.

[12] C. Frye, *Move to Workflow Provokes Business Process Scrutiny*, **Software Magazine**, April 1994, pp. 77-89.

[13] D. Hollinsworth, *The Workflow Reference Model*, Workflow Management Coalition, TC00-1003, December 1994. Accessible via: http://www.aiai.ed.ac.uk/WfMC/

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

[15] *FlowMark: Managing Your Workflow*, IBM Document No. SH19-8176-01, September 1994.

[16] *FlowMark: Programming Guide*, IBM Document No. SH19-8177-01, September 1994.

[17] T. Imielinski, B.R. Badrinath, *Mobile Wireless Computing: Solutions and Challenges in Data Management*, **Communications of the ACM**, Vol. 37, No. 10, October 1994.

[18] M. Kamath, G. Alonso, R. Günthör, C. Mohan, *Providing High Availability in Very Large Workflow Management Systems*, **Research Report RJ9967**, IBM Almaden Research Center, July 1995.

[19] F. Leymann, W. Altenhuber, *Managing Business Processes as an Information Resource*, **IBM Systems Journal**, Vol. 33, No. 2, pp. 326-348, 1994.

[20] F. Leymann, D. Roller, *Business Processes Management with FlowMark*, **Proc. 39th IEEE Computer Society International Conference (CompCon)**, February 1994, pp. 230-233.

[21] F. Leymann, Supporting Business Transactions Via Partial Backward Recovery in Workflow Management Systems, **Proc. GI-Fachtagung Datenbanken in Büro Technik und Wissenschaft - BTW'95**, Dresden, Germany, March 1995, Springer Verlag.

[22] S. Mehrotra, R. Rastogi, A. Silberschatz, H.F. Korth, *A Transaction Model for Multidatabase Systems*, **Proc. International Conference on Distributed Computing Systems**, June 1992, pp. 56-63.

[23] C. Mohan, R. Dievendorff, *Recent Work on Distributed Commit Protocols, and Recoverable Messaging and Queuing*, **Bulletin of the IEEE Technical Committee on Data Engineering**, Vol. 17, No. 1, March 1994, pp. 22-28.

[24] C. Mohan, *Advanced Transaction Models - Survey and Critique*, Tutorial presented at the **ACM SIGMOD International Conference on Management of Data**, May 1994. Available at http://www.almaden.ibm.com/cs/reports/workflow/tran_models_tutorial_sigmod94.ps.Z

[25] C. Mohan, K. Treiber, R. Obermarck, *Algorithms for the Management of Remote Backup Data Bases for Disaster Recovery*, **Proc. 9th International Conference on Data Engineering**, Vienna, April 1993.

[26] R. Obermack (Ed.), *Special Issue on TP Monitors and Distributed Transaction Management*, **Bulletin of the Technical Committee on Data Engineering**, Vol. 17, No. 1, March 1994.

[27] M. Hsu (Ed.), *Special Issue on Workflow Systems*, **Bulletin of the Technical Committee on Data Engineering**, Vol. 18, No. 1, March 1995.

[28] B. Reinwald, *Workflow Management*, Tutorial handout, **13th IFIP World Computer Congress**, August 1994.

[29] A. Sheth, Rusinkiewicz, *On Transactional Workflows*, **Bulletin of the Technical Committee on Data Engineering**, Vol. 16, No. 2, June 1993.

[30] B. Salzberg, D. Tombroff, *A Programming Tool to Support Long-Running Activities*, **Technical Report NU-CCS-94-10**, College of Computer Science, Northeastern University, Boston, 1994.

[31] A. Zhang, M. Nodine, B. Bhargava, O. Bukhres, *Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems*, **Proc. SIGMOD International Conference on Management of Data**, May 1994.

# DataLinks - Linkage of Database and FileSystems

Inderpal Narang, Bob Rees
Almaden Research Center
650 Harry Road, San Jose, CA 95120, USA

## Abstract

DataLinks is a research prototype being developed in the IBM Almaden Research Center.

The goal of the DataLinks project is to provide linkage between data stored in the database and the external filesystems. There is a class of applications where large objects, such as, video, image, graphics, etc. would be stored in the external filesystems. The reason is that they are captured in the filesystem, and the development tools work with the file paradigm. There is *no* need for physically bringing such data in the database since their raw content is not used for the database query[1] . However, the file data is related to the database tuples. A relational database system provides referential integrity for the data which it stores. So the question arises, "Can one provide referential integrity for the data which is linked with the DB data in the external filesystems?" The answer is yes, by using the DataLinks technology. DataLinks makes the file linkage of the database tuple a *first class citizen*. That is, the link would be robust from integrity, access control, and recovery standpoint, as if the file is stored in the database (but actually it is not).

In order to accomplish the robust linkage it is proposed that we add a new datatype in the relational database system. This datatype is referred to as the external file reference (efr).
The SQL call interface for the efr datatype involves an efr-data-structure. The application provides the server-name and the filename of the file in the efr-data-structure which is associated with the efr column in the SQL call. Likewise, the database system would return the server-name and the filename in the efr-data-structure in the SQL SELECT call.

The behavior of the efr datatype is that the DBMS would issue *link-file* (*unlink-file*) operation to the appropriate fileserver for the specific file when the application issues SQL insert/update (delete/update) calls[2]. The link-file operation results in, for example, making DB as the owner of the file and marking the file as read only. Note that all this happens in the transactional scope. The rationale for changing the owner is that the file can no longer be deleted by normal users of the filesystem; the rationale for marking the file as read-only is that indexes may be created on the file which are stored in the database for search.

DataLinks would prevent deletion or rename of a file by a normal filesystem user if such a file is linked with the DB data. It would also provide access control to the file as an option. Note that all this requires *no* modification to the filesystem where the file is actually stored. The added functions are implemented in a

1. Typically, a user extracts features of an image or a video and stores them in the database for performing search on the extracted features. An example of the features which can be extracted of an image are, color, shape, and texture. IBM's Query By Image Content (QBIC) supports extraction and serach on such features..

2. This is in contrast with the middleware approach where the middleware is cognizant of the database and the filesystems and provides the linkage. We believe that time has come to formalize the database linkage with the filesystems so that middleware software may not be needed.

layered approach. For example, using the DataLinks technology, the files stored in the video server can be linked with the database data without modifying it.

With DataLinks, an application uses standard APIs for the database access and the file access. The application uses SQL for the database access and standard filesystem calls (e.g., Open, Read, Close) to access files. We believe that this is one of the strengths of the DataLinks technology, i.e., no new API[1]. An application scenario goes as follows. An application issues SQL SELECT to search on the business data and the extracted features of the image stored in the database. The query returns its results which includes filename/server-name as normal column data in the efr data-structure if an efr column is selected in the query. The application can then use the standard filesystem protocols (such as, open, read, close) to access the relevant portion of the file.

DataLinks does not come in the data-path of the file access. It only interposes itself in the file open, rename, delete type of calls. Therefore, DataLinks can be used with the stream servers (e.g., video servers) and provide the value add of robust linkage with DB data.

DataLinks imposes no data model of its own for applications[2]. The data model is whatever can be supported by the relational model (or other database management systems in future).

DataLinks technology interoperates with the IBM's ADSM space management technology to provide storage hierarchy transparently to the filesystem as an option. That is, the filesystem is not modified to support storage hierarchy.

The system configuration as shown in Figure 1 is possible when large objects are stored in a filesystem but are linked by database tuples. The database can be a centralized index for searching across the enterprise-wide data which has both business data and extracted features of the non-coded data, and the large objects can be distributed among several fileservers. Such a configuration can save network costs since the large objects can be stored close to the end-users and hence can be delivered over shorter distances. Note that such a configuration is not possible if the large objects were stored in the database.

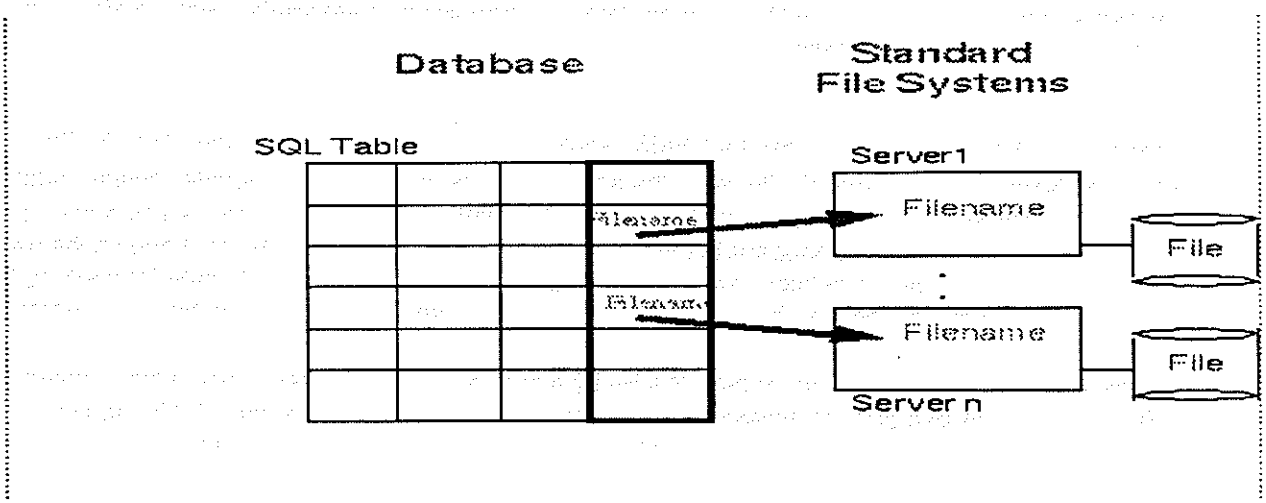



Figure 1. Cenralized database with several distributed fileservers. The SQL table has a column of datatype called external-file-reference (efr). The efr column contains server-name/filename which provides the location of the file.

1. This is in contrast with the middleware approach which may have its own API.
2. This is in contrast with the middleware approach which may impose a data model of its own.

Currently, DataLinks is being developed as a research prototype in the IBM Almaden Research Center. However, in order to develop applications *today* where a customer may want to store large objects in filesystems and link it with DB data, it is possible to make such applications "DataLinks Ready". We would provide a few macros which if incorporated in the application would make these applications "DataLinks Ready". This would not have any impact on the performance of the application. When DataLinks becomes available in the product, such applications only need to be re-compiled with new macros and they would have the full functionality of DataLinks, i.e., referential integrity, access control, and storage hierarchy support as options.

# Temporal Data Manager

Greg Hope, Chief Architect, Paul Oeuvray, VP Systems Design,
Paul Miniato, Senior Consultant
Prologic Corporation
100-3851 Shell Road
Richmond, B.C. V6X 2W2 Canada
Phone: (604) 278-6470 Fax: (604) 278-5206
Internet: hope@prologic.ca

*A transaction processing application design often contains master tables and associated transaction tables. Each row in a master table represents the current state of some object, and each row in a transaction table represents an event that affected the object in the master table. An application of this type often has complex requirements relating to the treatment of time.*

The automated implementation of these crucial application requirements is provided in a consistent application-wide manner by the Temporal Data Manager (TDM), a component of PROBE's Unified Application Framework (UAF).

The TDM includes the following functionality:

- recording and making available all user updates for review by auditors
- using time-dependent business rules to specify the effect a transaction has on its master
- using time-dependent business rules to specify the effect the passage of time has on the master
- recreating data as of some time in the past or predicting data into the future
- applying transactions out of sequence (backdated transactions) if they arrived later than their effective time.

Human error is inevitable and the application must facilitate detection and easy correction:

- users of a transaction processing system will make mistakes in transaction entry
- information systems staff (developers and operators) will also make mistakes in programming business rules or database maintenance.

Ultimately, while meeting the complex requirements of the application and managing unavoidable human error, the application must guarantee the integrity of the master and transaction tables containing the data that represents the heart of the business.

Implementation of these features in an application normally requires enormous effort, both as a result of the volume of programming required, and of the very complex problems that can arise from maintaining the integrity of time-sensitive data and business rules. Typically, the complexity forces compromises in application requirements (for example, manually recalculated backdates). Errors in this type of data are not uncommon and require time-consuming diagnosis and correction; they could even remain unnoticed until a customer complains.

The TDM anticipates human error and delivers complete integrity for this critical data without additional programming effort.

*Temporal integrity* is PROBE's unique patent-pending feature within the Temporal Data Manager that guarantees the result of the transactions will equal the current master row.

The TDM delivers to transaction processing applications a dramatic increase in complex functionality while reducing programming effort and guaranteeing data integrity.

## Time relationships

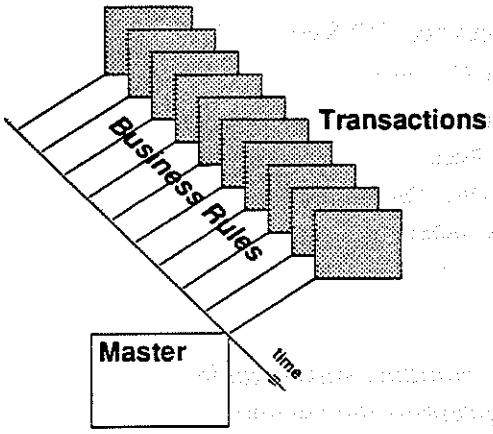



*Figure 1: Time relationship concepts.*

A *time relationship* is a one-to-many relationship between a master table and a transaction table in which each row of the master table is related to a set of rows in the transaction table.

Applications usually support two ways of describing the time the transaction happened. The *entered* time is used for audit purposes and records the actual time the transaction was entered into the database. The *effective* time records when the transaction is to take effect for the purpose of applying the application's business rules.

### Encapsulation of time-related business rules

To complete the time relationship, the application supplies two sets of business rules defined by the *activity* and *periodic* triggers. The first rule describes the effect a new transaction or "activity" has upon the master (such as adding a transaction amount to an account balance), and the second describes the effect the "periodic" passage of time has upon the master (such as the accrual of interest on the balance). These two business rules always work on the premise of moving forward in time, regardless of whether a transaction was input in sequence or inserted into the past.

Encapsulation of business rules into these well-defined triggers provides two main benefits. The business rules are guaranteed to be consistently executed regardless of whether a new transaction is initiated by user input, batch utility, or batch application program. Specifying business rules in one place forces a reduction in programming effort by eliminating the possibility of duplicated or scattered business rules across the application.

### Automatic calculation of a past or future master row "as of" a given time

An application often requires the ability to inquire into past or future states of the database. The TDM delivers this without additional programming.

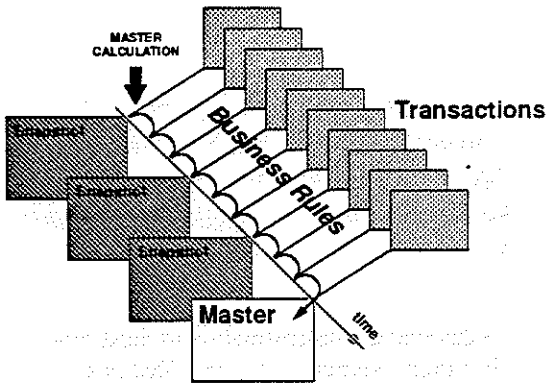



*Figure 2: Creating snapshots and updating the master as time passes.*

The TDM keeps a copy of each initial master, and as time passes and transactions are inserted, additional copies of the master are made periodically. These copies of the master through time are called *snapshots*. Using these snapshots and the application's business rules, the TDM is able to efficiently recalculate the master for any exact point in the past. As illustrated in *Figure 3*, the periodic and the activity triggers (the business rules) are invoked one after another for each transaction between the prior snapshot and the desired "as of" time. A final invocation of the periodic trigger calculates the master from the effective time of this last transaction up to the "as of" time.

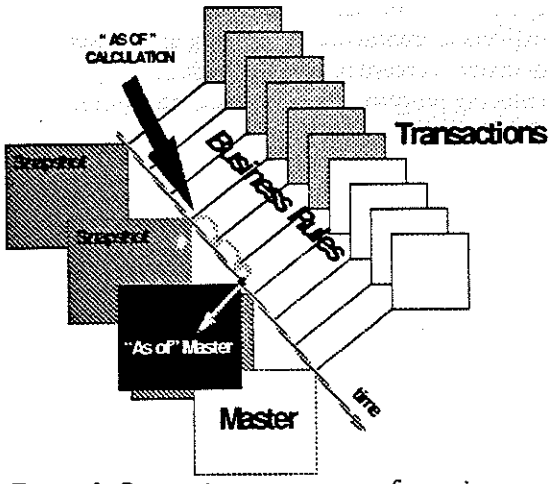



*Figure 3: Recreating a master as of any given time.*

The TDM can project a future master starting from the current master and using only the periodic trigger.

### Automatic audit trail of master changes

Any change a user is allowed to make directly to a master is automatically recorded as a transaction. Because this manual change is recorded as a transaction, it is considered when recreating a master in the past. The TDM supports this feature even for tables other than time relationship master tables, enabling a user to see a consistent application-wide audit trail of all changes to the database—all without any programming.

### Automatic error correction with backdated transactions

Because users of an application will make mistakes, the ability to make corrections is required. The correction could involve a change to any column(s) on the transaction row, but usually a PROBE application supporting non-destructive updates only allows changes to the built-in column called *reversed* (and its associated audit trail columns). When a value is entered into the reversed column, it indicates that the transaction row has been canceled and it will now be ignored by the TDM. This leaves a record of all canceled transactions for audit purposes. If required, a replacement transaction then is inserted.

In the case of errors of omission or transactions received late from other systems, backdating allows transactions to be inserted out of sequence. These are two examples of a *backdated* transaction. Supporting backdates often becomes a large and complex part of an application particularly when the time or sequence of the input has an effect on the business rules.

The TDM automatically provides the ability to allow backdates, delivering an application-wide consistent ability to make corrections or insert transactions that arrive late.

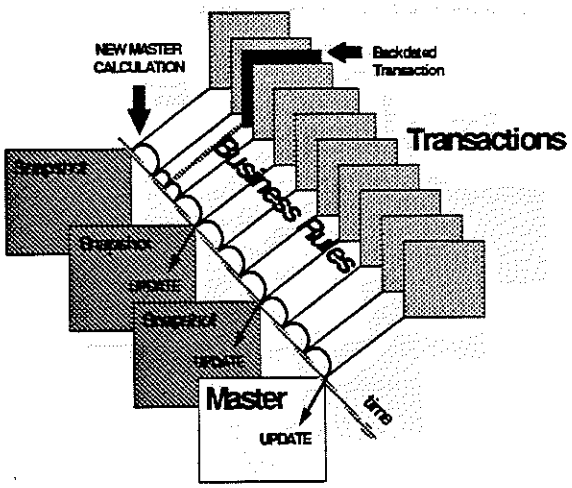



*Figure 4: Updating the master after inserting a transaction into the past.*

To apply a backdated transaction, the TDM recreates the master row just prior to the backdate using the same technique as in the "as of" example. The TDM applies the application's business rules to the backdated transaction. Then in ascending effective time order, the TDM re-executes the business rules for each transaction after the backdate. The resulting new master incorporates the backdated transaction.

A manual change to a master (recorded as a transaction by the TDM) automatically is allowed to be a backdate.

### *Temporal integrity (patent pending)*

Traditional systems tend to avoid storing calculated columns because there is no way to guarantee the integrity of these columns over time. Without storing these columns, it is necessary to calculate them repeatedly through time, which significantly reduces performance. If calculated columns are stored, and performance is increased, the system is difficult to build and maintain because a significant amount of

program logic is devoted to ensuring integrity. In many cases backdates simply are not supported.

The most common cause of an integrity violation is a change to a business rule intended only for transactions in the future, but applied to all transactions. Now the existing transactions do not add up to the current master row.

The net result of errors due to developers specifying business rules incorrectly or operators performing incorrect database maintenance operations is a loss of data integrity.

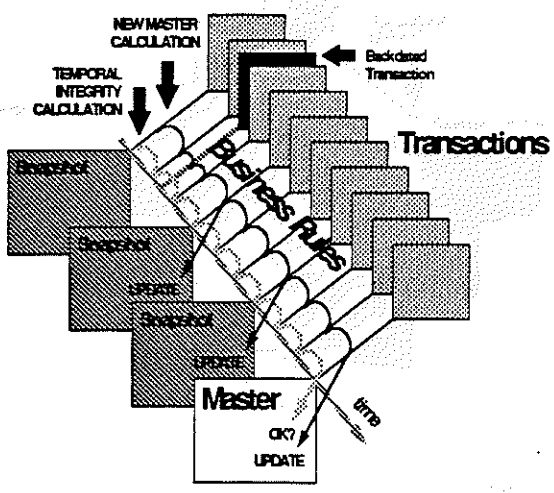



*Figure 5: Temporal integrity checking during the new master calculation*

To maintain integrity, the TDM ensures that business rules are always properly time-sensitive. The TDM similarly protects against a loss of integrity due to corruption of data or direct update of the master or transaction table. With the TDM, master rows and snapshots (copies of past master rows) contain calculated information derived from the initial master row plus the total effect of each transaction according to the business rules. Temporal integrity is PROBE's unique patent-pending feature which guarantees that this calculated information always correctly corresponds to the transactions.

The TDM checks for temporal integrity in two situations. While performing a new master calculation, the TDM checks the existing temporal integrity of the current master and its transactions. As shown in *Figure 5*, the temporal integrity calculation ignores the new backdated transaction and verifies existing temporal integrity only. A batch utility program also can be used to test all masters and their transactions for integrity.

In summary, PROBE's Temporal Data Manager delivers to transaction processing applications a dramatic increase in complex functionality while reducing programming effort and guaranteeing data integrity.

# DSS Performance is Now More Significant than OLTP Performance

Position Paper for HPTS-95
Pat O'Neil
UMass/Boston
poneil@cs.umb.edu

The thesis of this paper is that update transactional performance of OLTP systems has been a solved problem for some time now, with most database system vendors implementing appropriate algorithms to avoid well-understood bottlenecks, while data mining queries of the kind used in Decision Support Systems are still poorly understood, with tremendous variation in performance between commercial products. Although the amount of money spent on OLTP is probably 90% compared to 10% for DSS, the large variation in DSS query performance implies that greater savings are probably possible for DSS, and it is argued that the industry should now concentrate its efforts on this area.

## OLTP Benchmarking

From an historical standpoint, the DebitCredit benchmark [ANON] focused the attention of the Database industry on a number of OLTP bottleneck problems and their standard solutions: writing logs to disk at a high rate (group commit), inserting new rows at a high rate to a sequential History file (row locking rather than page locking), I/O of different size tables (good memory buffering of disk pages), and contention on a small number of Branch records (row locking). Because of the early publicity the benchmark received and the customer demand for objective measures, vendors began to feel they needed to publish Debit Credit results even when the results were relatively embarrassing. (See [FTSN-69], May 1988: "Cullinet evidently also tried to emulate the Sybase test . . . but only got 14 TPS [running IDMS/SQL Release 1.0 on a VAX 8850] for its trouble . . . there were 3 direct I/Os per transaction, strongly suggesting that Cullinet's memory buffering scheme (if any) is entirely ineffective.") This led to an immediate focus in many vendor shops to improve performance and remove the embarrassment (FTSN-74, in October 1988, reported new Cullinet IDMS/SQL results of 27 TPS on a VAX 8850 with their newest Release 1.2).

The Transaction Processing Performance Council (TPC) vendor consortium was founded by Omri Serlin to create standard benchmarks for the industry, starting with the DebitCredit replacement (TPC-A) and the TP1 replacement (TPC-B, the batch version of TPC-A with no terminal costing). Their specification and a large number of validated results appear in The Benchmark Handbook [HANDBOOK]. It is noteworthy that since the TPC was formed, no other benchmarks created outside the process have reached prominence requiring recognition by the industry. At the same time, a number of vendors have included clauses in their licenses that make it an actionable breach of contract to measure their product and report the results without permission.

When DebitCredit was modified to create TPC-A, two major changes incorporated were these: (1) the elapsed time allowed for the 90th percentile of response was increased from 1 to 2 seconds; (2) the think time between transactions on a terminal was reduced from 100 seconds to only 10 seconds. The increase in response time seems reasonable, an attempt to allow less expensive machines to satisfy criteria for inclusion. Probably very few customers would refuse this minor increase in response time if promised some cost savings. However the drop in think time is more questionable, since ten seconds seems an unreasonably short time between order entry commits. The shortened think time stemmed from the belief that terminal and network costs in Debit-Credit were too large a percentage of the total cost for a transactional system. Such large extraneous costs seemed to trivialize the importance of the transactional

system software which the TPC was formed to measure. This consideration gained significance as transactional measurements improved. As mentioned on the inside front cover of the [HANDBOOK] where TPC-A results from 1991 and 1993 are compared, the average $/TPS numbers from 1993 were well below the best results of 1991.

Possibly because of this improvement, the extremely lightweight property of the TPC-A transaction logic became controversial. In 1995, TPC-A and TPC-B were decommissioned, and the council will no longer validate results. To replace them, the TPC-C benchmark was created to be a better representation of what really happens in a medium-sized transactional application system. In TPC-C, several different kinds of work were performed on multiple tables, and the average transaction was much more heavyweight than it had been in TPC-A. (See [HANDBOOK] for a complete description.) One of the aims in creating this benchmark was to make it clear that for real OLTP work transaction system software was an important cost factor compared to terminal and network costs. However, the TPC-C benchmark assumed a somewhat more sophisticated terminal than TPC-A (not unreasonable in real practice), and as a result the cost of terminals and communication networks remained high, about 2/3 of total costs. Most recently, the TPC has dropped from the TPC-C specification the requirement to cost terminals, making the significance of these costs less visible.

From this history, the lesson seems to be that for several state-of-the-art database systems available today, the cost of CPU and I/O in OLTP processing is so low as to be relatively insignificant compared to other costs of building such a system. (None of the benchmarks mentioned ever attempted to assign a cost to developing and supporting applications, for example.) What this probably means is that system acquisition decision should give large weight to other factors than TPC-C performance results — factors such as a helpful application development environment and effective sets of utilities. To a great extent, the lessened importance of transaction performance is a vindication of the original DebitCredit benchmark, which focused vendor attention on transactional bottlenecks, and brought performance to a point where OLTP transactions can be viewed as a commodity.

## Query Performance and Comparisons to OLTP

There are three benchmarks that deal with query performance included in the [HANDBOOK], the *Wisconsin Benchmark*, the *AS3AP* benchmark, and the *Set Query* benchmark. More recently, the TPC has developed its DSS benchmark, *TPC-D* [TPC-D], to a point where it is near release. All of these benchmarks have certain deficiencies from a standpoint of properly measuring Decision Support System performance, but even so we can learn a number of lessons from considering results that have been obtained. We outline two important ones.

(1) Individual queries within the DSS query sets have much larger response time on the average than typical update transactions.

The TPC-D benchmark can be thought of as executing queries to produce reports on the kind of data produced by TPC-C. In the introduction to the TPC-D benchmark, Decision Support systems are said to execute queries that, "Are far more complex than most OLTP transactions". If we consider transactions profiles in TPC-C, the data accessed (including index) is usually quite limited, only what is needed to locate rows for update, and usually only a few rows. Jim Gray (pg. 10 of the [HANDBOOK]) characterizes the weighted average of the five transaction profiles from the TPC-C benchmark as being about ten times "heavier" than TPC-A, and the units reported are in transactions per minute (tpm-C) rather than transactions per second (tps-A). But many of the sixteen DSS queries of TPC-D potentially reference all the data collected over a long span in several large tables: for example Q6 considers all the lineitems shipped in a given year, and lists the amount by which total revenue would have increased if certain discounts had been eliminated. The extent to which a DSS can avoid looking at all data involved often depends solely on the sophistication of the indexing capability of the underlying system. While no per-

formance results from TPC-D have been made public at the time of this writing, it should be noted that the units reported for TPC-D are based on a queries per hour rate. As a separate indication of the relatively long response time for queries, recent measured results from the Set Query benchmark provide an average measure of about 1 query per minute on a platform that provides about 50 TPS on TPC-A, implying a ratio of about 3000:1 in resource use for queries, compared to TPC-A transactions.

Notice that the tremendous resource requirement of an average DSS query means that it takes only a moderate number of workstations to keep even a very powerful server busy compared to a typical OLTP system. This observation matches well with experience, since a small number of analysts typically drive large Decision Support system.

(2) Performance of individual queries within the DSS query sets depend on a large set of detailed performance features in the underlying database products; with the current state of sophistication, this implies extremely disparate performance between different products.

The Set Query benchmark can be used to distinguish underlying performance features of database systems by examining measurements of a number of the distinct single-user queries. In the [HANDBOOK] Set Query article, two database products were compared on a single platform with identical hardware, giving ultimate Query Per Minute (QPM) ratings of .6267 (for DB2 V2.2) and 2.697 (for M204, i.e. MODEL 204, V2.1), a ratio of 4.3. The $/QPM rating was 3.735. Measurements on more recent releases of both products (DB2 V2.3 and M204 V2.2) showed a new QPM ratio of about 3.3 on a somewhat faster CPU platform, illustrating a number of performance improvements by DB2. To compare OLTP measures, one can examine the TPC-A benchmark results from Appendix A of [HANDBOOK], tabulated January 4, 1993; these show a total of 6 different pairs of measurements for distinct database systems on identical computer models (three different database system products were involved: INFORMIX V5.0, SYBASE V4.9.1, and ORACLE7). The maximum ratio between any two products in $/TPS ratings was 1.173. The fact that these ratings are so close serves to confirm the earlier conclusion that transactions have become a commodity, while the much larger ratios for query performance illustrates point (2) above, the disparate query performance of different database products.

But even this ratio is misleadingly small from a standpoint of an ideal amalgamation of the features of the two products. DB2 and M204 actually have quite different strengths. M204 is superbly efficient at combining indexed filter factors of multiple predicates in a where clause, so as to achieve a small RID list of rows to retrieve. This capability is based on efficient bit map use and is not properly accounted by traditional query optimization theory, in that it is 50 or so times as efficient as the method used by DB2. But DB2 has extremely sophisticated ability to perform efficient multi-page I/O, using sequential prefetch and list prefetch to greatly improve extremely time-consuming queries. This extremely efficient implementation also seems not to be properly accounted in query optimization theory. There is evidence too that DB2 leads all other products in multi-page I/O capability, as well as in efficient joins using RID lists. A benchmark performed in 1992 for a large insurance company, with a preponderance of report queries retrieving all rows in long clustered ranges, showed DB2 to be equal in price-performance to leading UNIX based database systems. This equality arose from relative efficiency of DB2 I/O and CPU processing, and held in spite of the relatively high IBM hardware prices of that time. The much better utility performance available on DB2 (loads were much faster) clinched the acquisition decision in their favor. Since that time, multi-page reads have improved on many products, but not to a sufficient extent, while IBM prices have come down a great deal, so a benchmark of today would continue to be of interest.

## Summary

Decision Support System query performance needs are relatively poorly understood and not well implemented in most database systems of today. There are a large number of detailed features

that seem to define the state of the art in query performance, many of them not yet properly reflected in commonly understood theory of query optimization. As a result, no product seems to have made correct architectural decisions to implement all these features, and indeed different product implementations seem extremely scattered in this regard. Because of this, there is a wide disparity in measured query performance between database systems. Since queries take large amounts of hardware resources, this disparity results in surprising differences in costs between different DSS platforms. Because OLTP price-performance seems to be converging on the different platforms, and DSS purchases are probably rising, we can expect query performance to be an important distinguishing feature between database systems of the near future. Note that the ability to provide intra-query parallelism will not address this need, since parallelism can only hope for linear speed-up, and will not actually *improve* query cost-performance.

## References

[ANON] Anon et al., *A Measure of Transaction Processing Power*, originally published in Datamation, February 1985, appears in *Readings in Database Systems* (both in the First and Second Editions), M. Stonebraker (Ed.), Morgan Kaufmann.

[FTSN-69] *FTSN (Fault Tolerant Systems Newsletter) 69*, 13 May 1988, pg. 10, ITOM International, Los Altos, CA.

[HANDBOOK] The Benchmark Handbook For Database and Transaction Processing Systems, Second Edition, Jim Gray (Ed.), Morgan Kaufmann, 1993. Contains numerous benchmark descriptions as cited here, together with TPC-A, TPC-B, and TPC-C results in Appendix B.

[TPC-D] Available from TPC, contact Shanley Public Relations, San Jose, CA, tpc@cup.portal.com.

# Towards Implementing Extended Transaction Models on Conventional Transaction Processing Monitors

Roger Barga Calton Pu

Department of Computer Science and Engineering
Oregon Graduate Institute of Science & Technology
P.O. Box 91000
Portland, OR 97291-1000
email: {barga,calton}@cse.ogi.edu

## 1 Introduction

The last five years have witnessed the introduction of numerous extended transaction models [Elm93]. Despite their popularity, there are no commercial transaction processing (TP) products or prototypes that incorporate extended transactions, and relatively little has appeared in the literature on implementing extended transaction models. We present the *Reflective Transaction Framework*, a practical and modular framework that can be used to implement a wide range of extended transaction models on conventional TP monitors [BP95]. We achieve modularity by applying the Open Implementation approach [Kic92], also known as meta-object protocol [KdRB91], to design the reflective transaction framework. We achieve practicality by basing the implementation of the reflective transaction framework on the well documented ideas of the TP Monitor architecture [GR93], which is widely applicable to many modern transaction processing systems.

Our proposed implementation introduces *transaction adapters*, add-on modules built on top of existing commercial TP components that extend their functionality to support extended transaction features and semantics. We further demonstrate practicality by incorporating transaction adapters within Encina, a commercial TP facility. This modular and practical design enables us to implement a wide range of extended transaction models on a commercial TP monitor, and we demonstrate this with the implementation of two independently proposed extended transaction models for collaborative work [BP95] (split/join model [PKH88] and cooperative groups [MP92, RC92]).

## 2 The Reflective Transaction Framework

Classic transactions are bracketed by the control operations `Begin-Transaction`, `Commit-Transaction` and `Abort-Transaction`, while extended transactions can invoke additional operations to control their execution, such as `Split-Transaction`, `Join-Transaction` or `Join-Group`. A particular transaction model defines both the control operations available to transactions that adhere to that model and the semantics of these operations. For example, whereas the `Commit-Transaction` operation of the standard transaction model implies the transaction is terminating successfully and that its effects on data objects should be made permanent in the database, the `Commit-Transaction` operation of a member transaction in a cooperative transaction group implies only that its effects on data objects be made persistent and visible to other member transactions. To capture this distinction, we first separate the programming interface of the transaction facility in order to keep the basic function of a transaction independent of the advanced operations required for extended transactions, and to control implementation level concerns.

To design the reflective transaction framework, we first apply the Open Implementation approach [Kic92] to separate the programming interfaces to extended transactions, and identify modular functional com-

ponents required to realize this separation. Next, we proceed to extend the underlying transaction processing facilities to support these modular functional components via transaction adapters. We outline these steps below, and their complete description is available in the full version of this paper [BP95].

## 2.1 A Separation of Interfaces

The Reflective Transaction Framework separates the programming interface to transactions into distinct levels, where each level presents a different view of transaction functionality. This separation follows the Open Implementation approach [Kic92], in which the *functional interface* is separated from the *meta interface*, and the purpose of the meta interface is to modify the behavior of the functional interface. In our separation of interfaces, presented below, Level 1 and Level 2 are functional, subdivided for clarity only. Level 3 is the meta interface that modifies the semantics of the transaction functional interface (Levels 1 and 2).

Level 1 The transaction demarcation interface: `begin-E-transaction`, `commit-E-transaction`, and `abort-E-transaction`. The addition of letter `E` in front of `transaction` indicates that these operations extend transaction semantics beyond ACID.

Level 2 The extended transaction interface (operations defined by each extended transaction model):

- For the split/join transaction model, it is `Split-Transaction` and `Join-Transaction`.
- For the cooperative group transaction model, it is `Begin-Group`, `Join-Group`, `Commit-Group`, and `Abort-Group`.

Level 3 The meta-transaction interface: extends the implementation of the TP monitor to support the extended transaction interface (Level 2). For the extended transaction models considered in this paper, the operations needed are: `instantiate`, `reflect`, `delegateOp`, `delegateLock`, `formDependency`, and `noConflict`.

The *transaction demarcation interface* (Level 1) exports the basic transaction interface. When used alone (Level 2 and Level 3 not involved) it provides classic ACID transaction semantics. The *extended transaction interface* (Level 2) exports a model-specific transaction interface when extended transaction functionality and semantics are required. Finally, the *meta-transaction interface* (Level 3) exports a modifiable interface to the underlying transaction processing facility for implementing extended transaction models.

Realization of extended transaction models is facilitated through the careful design of the meta-transaction interface and its implementation. The design of the meta-transaction interface was inspired by the ACTA framework and readers familiar with ACTA will recognize that it supports many of the ACTA basic building blocks for describing extended transaction models. On the design side, the meta-transaction interface is close enough to ACTA to obtain modularity and applicability to a wide range of extended transaction models. On the implementation side, it is close enough to standard TP monitor architectures to support a practical implementation on top of commercial software. This design and implementation of the meta-transaction interface is captured in transaction adapters.

## 2.2 Implementation Through Transaction Adapters

Transaction adapters are modules built on top of an existing transaction processing facility that extend the underlying functionality. Each transaction adapter provides a representation (model) of the underlying transaction processing component for use by the meta-transaction interface, mechanisms for reasoning about and with such a representation, and a set of commands for controlling both the representation and the underlying transaction facility. This set of commands is referred to as **TRACS**, for **TR**ansaction **A**dapter **C**ommand **S**et. TRACS expose features such as operation and lock delegation, dependency tracking between transactions, and relaxed definitions of conflict, as explicit commands by which extended transaction models can be implemented. Thus, instead of applying operations in the meta-transaction interface directly to the underlying transaction system, we base them on an abstract and enhanced description of the underlying transaction system provided by transaction adapters.

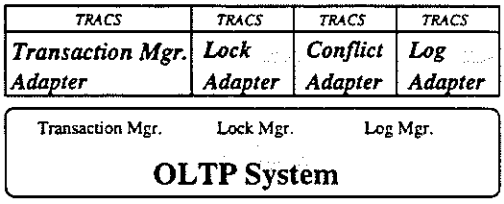



Figure 1: Transaction Adapters in the Reflective Transaction Framework.

Transaction adapters and their associated TRACS, as illustrated in Figure 1, are built on top of and use existing transaction processing services. For practicality we base their design on the TP monitor architecture, and are currently exploring their implementation via Transarc's Encina OLTP system. This way, TRACS provide similar reliability to mature OLTP systems software and minimize overheads often associated with increased flexibility. In fact, overhead associated with extended transactions is incurred only when the extended facilities provided by transaction adapters are used, and since an ACID transaction uses only the transaction demarcation interface (Level 1), it is executed without additional overhead.

## 2.3 Realizing Extended Transactions

In the Reflective Transaction Framework, a transaction is simply a partially ordered set operations on data objects, with calls to operations in any of the three levels of interfaces. Users are not limited to a predefined extended transaction model, which may or may not be appropriate for their application, nor is a transaction restricted to a single set of extended semantics and properties. Instead, a transaction declares its intention to use extended transaction properties and semantics via the `instantiate` command prior to beginning execution. The effect of this command is the creation of entries in the transaction adapters to represent this transaction, effectively creating a *metatransaction*. The meta-transaction command `reflect` assigns extended semantics to the metatransaction and supports the further refinement and extension of transaction properties during execution. For example, the sequence of transaction control operations presented below first creates a metatransaction for transaction $T_i$, assign it the semantics of the Split/Join transaction model [PKH88], and begins the execution the execution of transaction $T_i$.

```
instantiate(Ti);              // create metatransaction
reflect(Ti,Split-Join);       // assign split-join transaction semantics
begin-transaction(Ti);        // begin execution of extended transaction
```

A transaction with semantics beyond ACID properties is referred to as an *E-transaction*. When an E-transaction invokes a transaction control operation, the metatransaction is responsible for determining which function is actually executed based on the extended semantics of the transaction. Figure 2 illustrates the processing when the E-transaction $T_i$ invokes the `Split-Transaction` control operation. Thus, from the system point of view, we consider the *metatransaction* as an object that supplies the extended transaction model semantics, separate from the application programmer's view of transactions.

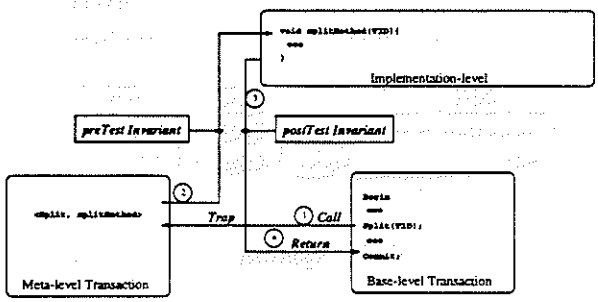



Figure 2: Transaction management method execution redirection.

The transaction systems programmer can define and specify the implementation of E-transaction operation by defining another metatransaction and then substituting it. In this sense, a suitable grouping of E-transaction operations form an extended transaction model. The sequence of metatransaction commands presented below specify the implementation of the `Split-Transaction` control operation. Metatransactions are thus realized through both the metatransaction interface and the transaction adapters and their associated TRACS.

```
E_splitMethod{
   instantiate(T2);                  // instantiate new transaction.
   reflect(T2, sj_model);            // add transaction semantics through reflection.
   delegate_lock(T2, DelegateSet); // delegate locks related to objects in the DelegateSet.
   delegate_op(T2, DelegateSet);   // delegate operations related to objects in the DelegateSet.
   begin(T2);                        // begin execution of the new transaction.
   return;                           // return control to invoking transaction
}
```

## 3 An Encina Implementation

One of the salient features of design of the reflective transaction framework is its compatibility with the TP monitor architecture, which is widely applicable to many modern transaction processing systems. One major advantage of this compatibility is the ease for implementation of the reflective transaction framework. Instead of starting from scratch, we can extend an existing OLTP system through the definition and implementation of the transaction adapters. Concretely, we are currently implementing the Transaction Manager Adapter, Lock Adapter, and Conflict Adapter (Figure 1) on Encina, a commercial OLTP system distributed by Transarc. The full version of this paper [BP95] contains a complete description of the implementation details.

The goals of our implementation effort are to: (1) demonstrate the practicality of the reflective transaction framework and transaction adapters, (2) evaluate extended transaction models in a real environment, (3) determine how easy it is to implement a wide range of extended transaction models, and (4) facilitate eventual technology transfer to real users.

# 4 Summary

We have presented the reflective transaction framework as a practical and modular framework to implement extended transaction models on conventional TP monitor software. Using the framework, we outlined an implementation method for extended transaction models based on *transaction adapters*, add-on modules built on top of existing commercial TP components, such as Encina, that extend their functionality to support extended transaction features and semantics. Our early experience [BP95] shows that the framework is general enough for a wide range of extended transaction models. Although the implementation details were product specific (Transarc's Encina), our framework was designed in the context of the TP Monitor Architecture, so it is applicable to many modern commercial TP monitors.

While the importance of extended transaction models has been known for many years, their use in real-world applications has been hampered by the lack of practical implementations. Furthermore, since most extended transaction models have been merely theoretical constructs, there are a number of important design issues that have generally not been discussed in the literature [?]. Our hope is that the reflective transaction framework will remedy this situation, providing a clear migration path to incorporate research advances in extended transaction models into commercial TP monitors. This will enable us to draw conclusions from direct experience in applying extended models in real, working environments.

# References


[BP95] Roger S. Barga and Calton Pu. A practical and modular method to implement extended transaction models. In *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich, Switzerland, September 1995.

[Elm93] Ahmed K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann, 1993.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers, 1993.

[KdRB91] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol.* MIT Press, 1991.

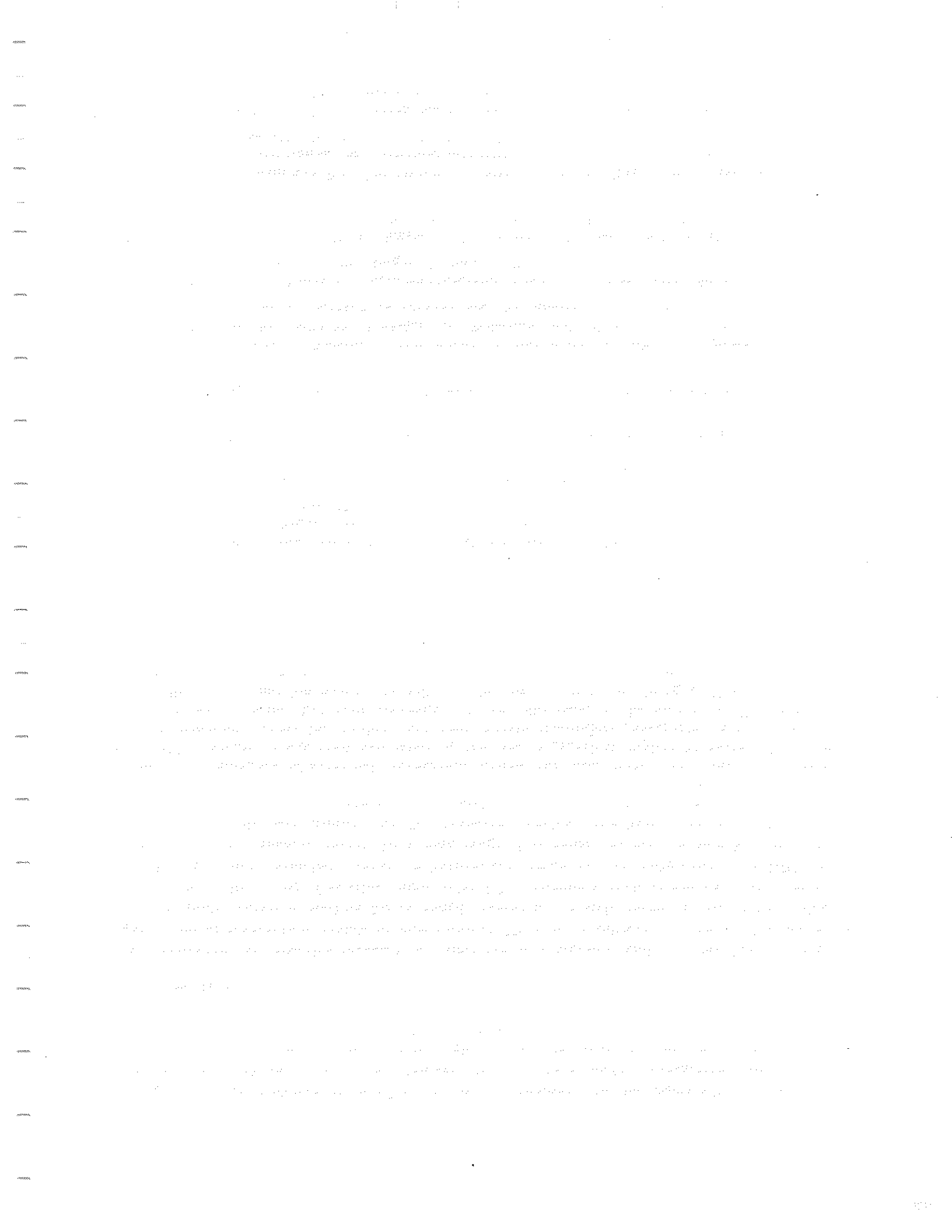
[Kic92] Gregor Kiczales. Towards a new model of abstraction in software engineering. In *Proceedings of the IMSA'92 Workshop on Reflection and Meta-level Architectures*, 1992. See `http://www.xerox.com/PARC/spl/eca/oi.html` for updates.

[MP92] B. Martin and C. Pederson. Long-lived concurrent activities. In Amar Gupta, editor, *Distributed Object Management*, pages 188–206. Morgan Kaufmann, 1992.

[PKH88] C. Pu, G.E. Kaiser, and N. Hutchinson. Split-transactions for open-ended activities. In *Proceedings of the Fourteenth International Conference on Very Large Data Bases*, pages 27–36, Los Angeles, August 1988.

[RC92] K. Ramamritham and P.K. Chrysanthis. In search of acceptability criteria: Database consistency requirements and transaction correctness properties. In Amar Gupta, editor, *Distributed Object Management*, pages 212–230. Morgan Kaufmann, 1992.

[RP91] K. Ramamrithan and C. Pu. A formal characterization of epsilon serializability. Technical Report CUCS-044-91, Department of Computer Science, Columbia University, 1991. To appear in IEEE Transaction on Knowledge and Data Engineering, June 1996.



# 13Million TP HA!

**T.K.Rengarajan[1] & Rabah Mediouni[2]**

**Abstract :** On April 12, 1994, Oracle Rdb 6.1 set the world record for TPC-A performance at 3692 tpsA at $4873/tpsA running on a 4 node 7000-650 AXP VMScluster using ACMS 3.3 transaction monitor and OpenVMS 6.1 operating system. This is about 13 million TPC-A transactions per hour. In this paper we describe the technical problems encountered and solutions used during the benchmark.

## 1.0 Introduction

On April 12, 1994, Oracle Rdb 6.1 set the world record for TPC-A performance at 3692 tpsA at $4873/tpsA running on a 4 node 7000-650 AXP VMScluster and OpenVMS 6.1 operating system.

In this paper we describe the technical problems encountered and solutions used during the benchmark. Included are tuning considerations, hardware configuration, Rdb enhancements incoporated based on expectations as well as actual problems found during the benchmark analysis.

## 2.0 System setup

The intial goal of the benchmarking effort was to reach 3000 tpsA in an effort to far outdo the tpcA record at the time for relational databases, which was at 1079 tpsA.

The back-end consisted of a VMScluster(ref?) of 4 nodes. Each of these 4 nodes was a multi-processor with 5 Alpha AXP 21064 processors (ref?). The processors had a clock rate of 200 MHz. Each of the nodes had 512MB of memory. The 4 nodes were connected to each other and to a set of 9 HSJ40 disk controllers by two Cluster Interconnects (CIs). About 220 RZ28 and RZ74s were hooked up to these disk controllers and load balanced. The controllers and disks were physically placed in StorageWorks cabinets. The front ends were VAX machines. They were connected on ethernet segments which were then connected to the backend via an FDDI connection.

1. Sybase Inc, ranga@sybase.com
2. Oracle Inc, mediouni@nova.enet.dec.com

# 3.0 Benchmark issues

## 3.1 Partitioned Lock Trees

With Rdb, each tpcA transaction did 2 lock operations, one to acquire and the other to release a transactional page lock on the account page. Since we had configured 2 CI interconnects for the cluster, we did not expect 6000 lock operations per second to be a bandwidth problem. However, we had a solution called Partitioned Lock Trees (PLT) implemented in Rdb 6.1.

All the lock resources for any Rdb database are arganized in one logical tree. This organization provides naming convenience as well as necessary hooks into OpenVMS to provide a simple recovery scheme on node-failure. PLT is a database-wide option. It splits the lock tree for the database into many trees, one per database area. OpenVMS migrates lock trees dynamically in the VMScluster based on load. With PLT, any application partitioned by areas is expected to perform only local lock requests. In the case of TPC-A, each branch, teller, and history tables was accessed only from one node. The account table also had this property except for the 15% remote requests required by the benchmark. Such partitioned access was ensured by routing of transacions by the ACMS transaction monitor.

PLT had the effect of eliminating most of the lock traffic from the CI. It also eliminated some of the code path in the VMS lock manager bo avoiding remote lock requests as well as saved some context switches for remote locks. We could not afford the time to precisely measure the impact of PLT.

## 3.2 Reduced size of db pages

One CI has a maximum bandwidth of 10Mbits/s(?). Digital folklore put the realistic limit on CI bandwidth at 5 Mbits/s. Each account page was 1KB. All disk IO in the cluster has to go via the CI. Since the CI bandwidth may become a bottleneck at 3000 tpsA, we chose to configure the cluster with two CI interconnects. Higher number of CIs were known to work with OpenVMS, but were not certified and hence was out of question for the benchmark.

In addition to configuring two CIs, we also wanted to reduce the size of the account page to 0.5K, since this was a tunable parameter in Rdb. The account table for range partitioned into multiple areas and then hashed with each area for 1 IO access. This had a bad effect on space utilization for account pages. To work around this, we used the "modulo hash function" feature of Rdb. This hash function simply places the account rows in round robin order from the first to the last page of the areas. This function eliminated all randomness and gave us precise control over sizing the account areas to fill each database page to the maximum. In addition to reducing the CI bandwidth requirements, this also reduced the disk arm movement for accounts.

## 3.3 Database load performance

We realized early on that the time to load the database of 50GB was going to be significant. The load time determined the extent to which we could try different physical organizations. Data was generated in a program and stored into the database via insert statements. The account rows were generated such that all rows that were placed on page 1 were generated first, followed by all rows that were placed on page 2 and so on. The use of the modulo hash function enabled us to precisely determine the target page for any account row with a specific account_id. This algorithm guaranteed sequential loading of data into each partition of the account table. We wrote a number of parallel load programs that loaded all the partitions in parallel. When we started all the load programs in parallel, we ran out of cpu. Therefore we scaled back the number of concurrent loaders, while keeping the 20 cpus completely busy. The load proceeded in big batches: each batch loaded a number of areas. The account areas were all sized the same and there was no trouble with load balancing. Incidentally, this is the load that was used as a benchmark of generalized parallel load in [Gray94].

## 3.4 Memory shortage!

In spite of having large amount of memory 512 MB, we ran into paging problems when we ran the final benchmark. We encountered this even though we had sized the Rdb buffer pool to be fairly small, since TPC-A does not benefit from large buffer pools. We attributed this to the large page size (8K) of OpenVMS on Alpha AXP. This can be explained by an example. Let usa say Rdb touches 1000 clusters of memory locations during a TPC-A run. On a VAX with a page size of 0.5KB, Rdb required 500 KB per process to eliminate paging. However, on Alpha AXP, Rdb required 8MB (8K * 1000) per process. This eight-fold increase in memory requirements going from VAX to AXP probably caused the memory problems.

## 3.5 VMS buffer objects

OpenVMS 6.1 included an innovative mechanism to share memory management responsibilities with its applications. It is called the buffer object. An application is allowed to declare a certain range of virtual memory as a buffered object. This makes the range of virtual memory non-pageable. Thereafter, every IO in the buffer object was executed via a faster code path that did not have to check for pinning pages to physical memory for the IO duration. VMS buffer objects was a perfect match for Rdb buffer pool. Rdb implemented this feature that allowed customers in turn to declare Rdb buffer pool as a VMS buffer object.

## 3.6 Rdb code path improvements

Many many areas, code path expansion due to # of areas, 10% code path cut due to new algorithms.

The database design for 3k tpsA required about 1000 (?) areas. One of our early experiments was to determine if Rdb could handle the large number of areas. This was determined not to be a problem, but a different issue surfaced. The large number of database areas changed the cpu profile of the tpcA transaction considerably. We used DEC Performance and Coverage Analyzer (PCA) for our cpu profiling. In fact, Rdb was running significantly (?) slower on one processor against the larger database sized for 3K tpsA, compared to our previous results on a database sized for 350 tpsA.

We completely changed our algorithms to make the code path independent of the number of areas. By this time in our work on Rdb performance, quite a number of unexpected places in the code started to hit the top of the cpu profile list. The code fragment that initialized a 2KB log buffer to zeroes showed about 5% cpu usage. We explained this as memory latency for the 300MHz processor. So we changed the code to avoid needless initialization! After all the talk about memory latency here was a concrete example where it hurt performance. In another instance, looping through all the buffers in a process's allocate set to reset a flag (to indicate undo logging was no longer necessary) was found to be expensive. We changed this algorithm too to limit the looping based on LRU buffer management.

All in all, Rdb tpcA code path was improved by 10% over previous releases, in addition to handling the slow down due to the large # of areas.

## 3.7 HSJ firmware limit

In the Storageworks disk configuration, we had configured one HSJ40 controller for ? disks. However, we ran into IO response problems for IOs to the account disks. This resulted in greater stall time per transaction and required more processes to fully utilize the cpu. The additional overhead of more Rdb processes decreased the tps. We performed a number of experiments with an IO exerciser for VMS called IOX and also used the tool called VTDPY and determined that we were saturating the IO controller. Fortunately, a new version of the IO controller firmware was being developed that promised savings in controller cpu cycles. This indeed got us over the hump with this IO bottleneck.

## 3.8 Client network problem

During the benchmark analysis phase, we tested to make sure each node with 6 cpus could execute more than 1000 tpsA. Once we started the test on all the nodes, we could not even reach 3000 tps. The transactions were not reaching the back end. The network turned out to be the problem. The ethernet segments were utilized to their practical maximum, about 53%. This forced us to understand about the configured network topology, various network components and the LAN analyzer.

## 3.9 Solid State Disks & ACE

ACE feature. IO to ACE took too long; last thing we expected; One 30K IO to SSD took longer than mag disk. Turns out to be 2MB/s throughput limit to the SSD. All SSD perf #s

done with 512 byte IOs! Mag disk stripe set? Stripe the SSDs? Log IOs larger than 32K. Small stripe size, VMS stack oveflow!

Rdb has a special feature called AIJ Cache on Electronic disk (ACE) to exploit solid state disks for transaction processing. We fully intended to use this feature for the big benchmark since it provided significant improvement in the results. However, when we tried the ACE feature on, the performance did not improve at about 3000 tpsA. The IO to the solid state disk took too long, about 10 ms. This was due to the size of log IOs at high tps. The log IOs kept increasing in size with tps per node such that we had to increase our limit to 64K (from 32K) for the size of one log write. We were running into a 2MB/s throghput bottleneck for the solid state disk. It turns out that all response time ratings were based on 0.5K IO sizes. Striping the SSDs three-ways yielded some improvement in response time, but increased the cost. Thus, our novel algorithm to exploit SSDs could not be used.

## 3.10 New goal : beat mainframe (3503 tpsA)

We had gone through a lot of troubleshooting and Rdb changes when we reached our goal of 3000 tpsA. We reckoned it is hard to get another opportunity to assemble such hardware in one spot to be put to test. Therefore we went after the 3500 tpsA number set by the TPF system on the IBM mainframe. It was not only the world's best current TPC-A number, but also represented the ultimate airline transaction processing system.

Since we were able to reach 3000 tps with only 16 processors, we were not short of cpus. However, we were short of disks and controllers. Since we could not assemble the necessary additional disks and controllers, we decided to double up on the existing disks for the account areas. We created a stripe set of 3 disks each using hardware striping from the HSJ40 controller and used it to hold 4 account areas. This worked well and helped us reach our new goal.

## 3.11 The Audit

The actual audit was itself an experience to remember. The auditor was friendly and had very flexible hours, worked around the clock. Our Isolation test was coded in Cobol and could not be run since a Cobol compiler was not installed on the benchmark machines. This required us to show isolation tests by hand using interactive SQL scripts and provoked a lively debate. The system failure test took almost all night. The recovery took a long time in spite of parallel recovery processes recovering independent account disks. Counting the history records after every performance run was made even possible by the asynchronous pre-fetch (APF) feature in Rdb. Otherwise we could not have completed the audit in time for the announcement. One of the performance runs showed too high tps numbers and required creative selection of the steady state window to bring it in line with the other results.

## 4.0 Conclusion

TPC-A is a deceptively simple test. However, it has taken almost a decade for all commercial RDBMSs to do well in this benchmark. The benchmark experience forced us to understand all components of the system, i.e., Alpha AXP processors, OpenVMS operating system, ACMS transaction monitor, StorageWorks disk subsystems, Digital's magnetic disks, Digital's network components. It provided us with a whetstone against which we could test Rdb and extract lessons that could be generally useful to many applications. It also provided us with an example of a system where the shared-disk hardware and partitioned application can work well together. Besides, this was about the only way to get roses for our wives from an authority on transaction processing who shall remain anonymous!

## 5.0 References

1. Original DATAMATION article
2. [GRAY94] Jim's db load paper
3. Our DTJ paper on ALS, ACE etc
4. Our old paper on Rdb commit sharing
5. Rdb doc set
6. Alpha AXP reference

## 6.0 Acknowledgements

This world record was a very big team effort that stretched over many years. A number of people have contributed over the years that culminated in setting this world record. These include Rick Anderson, Wael Bahaa-el-din, Mike Brey, Jay Banerjee, Arun Gopalan, Jim Gray, Steve Hagan, Paul Houlihan, Ashok Joshi, Steve Klein, Walt Kohler, Richie Lary, Gigi Lirot, David Lomet, Drew Mason Ray Pfau, Peter Spiro, David Walrath. We apologize for the omissions of other names.

# Indexing for Aggregation

Betty Salzberg *
College of Computer Science
Northeastern University
Boston MA. 02115

Andreas Reuter
Institute of Parallel and Distributed
High-Performance Systems (IPVR)
Breitwiesenstr. 20-22
D-70565 Stuttgart, Germany

## 1 Introduction

In decision support applications, very often aggregate values over several variables (using GROUP BY in SQL) are of greatest interest. These can be calculated each time they are needed using a standard SQL query. However, if the same values are used often, it may be wise to store them rather than recalculate them. In this case, they should be clustered so that likely subqueries involving ranges of the variables can be answered rapidly. This would imply that a multivariable index such as the R-tree [Gut84] or the hB$^{\Pi}$-tree [ELS95, ES93] be used to store the aggregate values.

In addition, the aggregate values being calculated may come from a relation other than the one(s) used for grouping. We have in mind a SQL query such as the following:

```
Select type-of-business, type-of-part, state, sum(amount)
from Customer, Parts, Sales-Offices, Sales
where Customer.CID = Sales.CID and Parts.PID = Sales.PID
and Sales-Office.OID =Sales.OID
and (state = Massachusetts or state = California)
group by type-of-business, type-of-part, state
```

In this example, the Sales relation contains information about each sale: an identifier of the customer, the object sold (the part), the office where the sale takes place, the dollar amount of the sale and other information. Each customer record lists the type of business the customer is in. Each sales office record contains the location of the office, in particular the state. Each part lists a type identifier as well as the part identifier. Many

*This work was partially supported by NSF grant IRI-93-03403 and by the IPVR.

parts will have the same type. The grouping is done by type-of-business, type-of-part and state, none of which are in the sales record. Without further support, this query would require three joins.

For discussion, we shall use the following more abstract formulation of such a query:

```
Select D1, D2, D3, sum(Q)
from R1, R2, R3, S
where R1.K1 = S.K1 and R2.K2 = S.K2 and R3.K3 = S.K3
and D1 > 43 and D2 < 56
group by D1, D2, D3
```

Here, as pictured in Figure 1, the relation S contains the value Q which is used for the aggregate, just as the Sales relation contained the *amount* attribute. The grouping is done by D1, D2 and D3, which are attributes which are not stored in S. Instead, foreign keys in S enable one to find in relations R1, R2 and R3 the corresponding D1, D2 and D3. There should be many different Ki corresponding to a given Di in general. That is, Di to Ki is a one-to-many relationship for each i. (So, for example, a *state* will have many sales offices.) The Ki's are keys in the relations Ri for each i. (So, for example, each Sales-Office record is uniquely identified by an OID—an Office Identifier.)

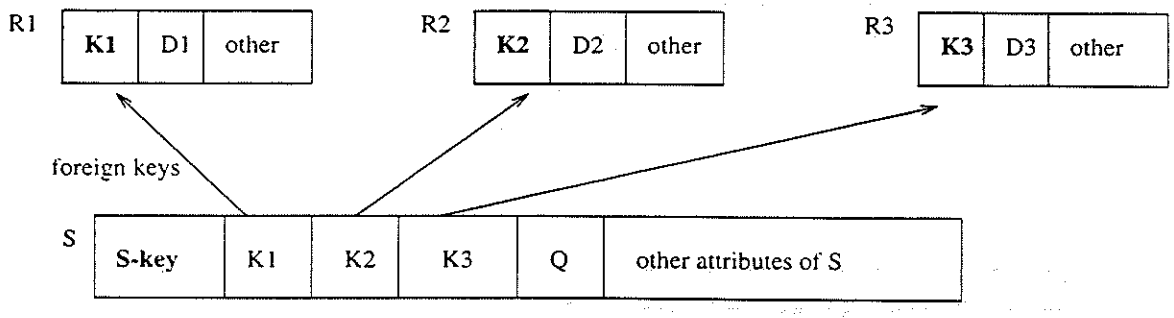



Figure 1: Relations R1, R2, R3 and S

In Figure 2, we show a multiattribute indexing structure $I$ which can be used to produce answers to the query above. The leaf level contains records with D1, D2 and D3 attributes, and for each triplet, the aggregate value(s) (sum, count, max, etc) wanted. We expect ranges of D1, D2 and D3 to be used in the queries so that the clustering in pages is in terms of the D-values.

## 2 Use of the aggregate index for the "cube"

As another example of a possible application of such a clustering index is the "cube" construct suggested in [GBLP95]. In the cube, group aggregation is done over $n$ attributes

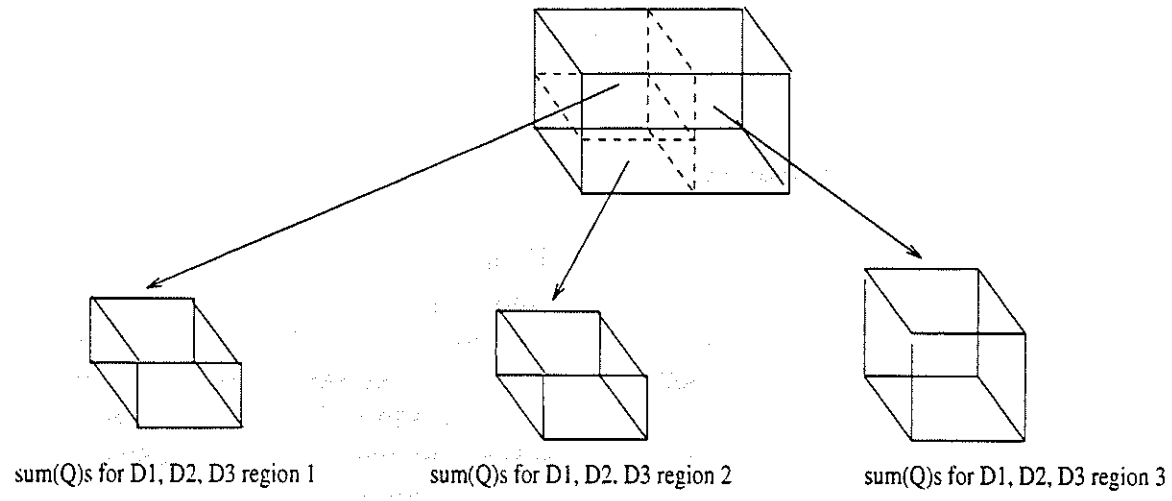



Figure 2: The three dimensional space for attributes D1, D2 and D3 is indexed by a multiattribute index $I$. Stored in the data pages at the leaves of the structure $I$ are the aggregate sums of Q for each tuple D1,D2,D3 belonging in the space described by the upper levels of the index for that page.

by grouping by any $k$ attributes from the $n$ attributes when $0 <= k <= n$. Thus, for example, the answers to the following SQL queries would be among those included in the response:

```
Select D1, D2, D3, sum(Q)
from R1, R2, R3, S
where R1.K1 = S.K1 and R2.K2 = S.K2 and R3.K3 = S.K3
group by D1, D2, D3

Select D1, D3, sum(Q)
from R1, R3, S
where R1.K1 = S.K1 and R2.K3 = S.K3
group by D1, D3
```

## 2.1 Calculation of Super-Aggregates from $I$

The index shown in Figure 2 is fine for the cube because clustering of data in leaf pages is fairly distributed among all attributes. Here, to calculate the result for two attributes (such as D1 and D3), the sum of all the stored values for a given D1 and D3 tuple would be made. That is, for less than $n$ attributes, calculating aggregates of stored values would be necessary. (This requires a little extra information for some aggregates such as average, which can be calculated from the count and sum or from the count and

individual averages, for example and it cannot be done for some aggregate values such as median, or middle value [GBLP95].)

### 2.2 Storing Super-Aggregates

An alternative is to store as well all the super-aggregates—the sums for each two attributes and each one attribute and so forth. If there are $n$ attributes, this has the property that $2^n$ updates must be made when a new tuple is inserted in the database. If $n$ is relatively small, this may be all right. For example, if $n$ is 3, as in our example, 8 changes must be made to stored aggregates and super-aggregates when a record is inserted. This may be acceptable if calculating super-aggregates from simple aggregates requires many disk accesses from the leaves of $I$ (so that calculation of super-aggregates is expensive) and/or if the super-aggregates can be kept in main memory (so that updating them is cheap). Then the savings on querying the cube would make the extra cost of updating worthwhile.

## 3 Data Structure for Updates

The use of a multiattribute index $I$ is straightforward if queries only are made and updates are never made. But when updates are made, the problem becomes more complicated.

When updates are made, by adding a tuple to S, the values of D1, D2 and D3 which correspond to K1, K2 and K3 are not known. Without adding another data structure, one could look up these values in the relations R1, R2 and R3 and then use them to update the index $I$.

However, in the case that such a tuple already exists in the database and only the value of the aggregate need be changed, it would be faster if there was an additional index $I'$, which would find the correct data page in $I$ and the correct record in that data page for a given tuple K1, K2, and K3. Such an index $I'$ would not cluster the data. $I'$ would contain individual references to records in leaf pages of $I$. The index $I'$ would probably be implemented efficiently as a hash table. This is pictured in Figure 3.

If a tuple was inserted in S which had values K1, K2 and K3 which did not already exist in the database, $I'$ would only be able to indicate that this was the case. Then it becomes necessary to do a join with R1, R2 and R3 to find the corresponding Ds. If a new (D1,D2,D3) tuple corresponds to the new (K1,K2,K3) tuple, a new record in $I$ is created. If not, an update is made in $I$. In both cases, a new entry is made in $I'$.

### 3.1 Splits in $I$

If $I'$ contains disk page addresses for the leaves of $I$, and record slot numbers, there is a problem when the leaves of $I$ split. Then all of the moved records are incorrectly addressed in $I'$ and a very expensive update is required to change this. Normally, there will be many references in $I'$ to one record in the leaf page in $I$. This is because for

each value of the attribute Di for each i, there may be many values of the attribute Ki corresponding to it.

On the other hand, if the key for $I$, (D1, D2, D3), is used as a reference in $I'$ instead of the disk page address, $I$ must be traversed to find the correct aggregate record in the leaf page to be updated. In addition, the size of $I'$ may be large (because the tuples (D1, D2, D3) may be long).

In addition, at a certain point in time, there is some reason to believe that almost all tuples (D1,D2,D3) which will ever exist are already represented in $I$. In the example we used to begin this paper, (D1,D2,D3) was (type-of-business, type-of-part, state). Once most of the business types had been sold most of the part types in most of the states, this would be the case. At this point, very few insertions in $I$ would be made, and consequently very few $I$-leaves would be split.

## 3.2 Our Proposal

We propose therefore, that $I'$ is not constructed until we are reasonably sure that most $I$ entries are present, and that disk page addresses and record slots be used in $I'$. We show how updates to $I'$ resulting from $I$-leaf splits can be done lazily by detecting in an $I'$ search that a split has been made.

First suppose the records in the hash table buckets of $I'$ contain only the (K1,K2,K3) tuples and the corresponding page address and record slot number for a record in $I'$. Without more information, there is no way to tell without doing a join, which (D1,D2,D3) tuple corresponds to a given (K1,K2,K3) tuple. So a moved tuple in an $I$-leaf could not be detected by looking at the contents of the $I$-leaf.

To solve this problem, we shall assign to each (D1,D2,D3) tuple a unique identifier and store it both in $I$ and also in each reference to this tuple in $I'$. When a search from $I'$ is made and the identifiers do not match, that indicates that a split has been made. But then where is the correct (D1,D2,D3) tuple?

We also store in each page that is split the number of times it has been split and a pointer to the page which most recently split from it. A new sibling copies the pointer from the page it split from and has split count zero. With this information, we will know how far to go to find the new position of a record which has been moved.

Split counts are actually only needed if we allow deletions of (D1,D2,D3) tuples when their count attribute is zero. If such deletions are allowed, they are distinguished from moves when the tuple cannot be found by the end of the search.

Since this proposed structure follows linked lists, it only makes sense if it is used infrequently. So the condition of knowing that there are unlikely to be many splits in $I$-leaves is important. There are two other alternatives: (1) have no $I'$ and update using joins and (2) use the (D1,D2,D3) tuples as references in $I'$ and follow $I$ to find the tuple to update. These are both slower in the anticipated usual case when there are few splits of $I$-leaves.

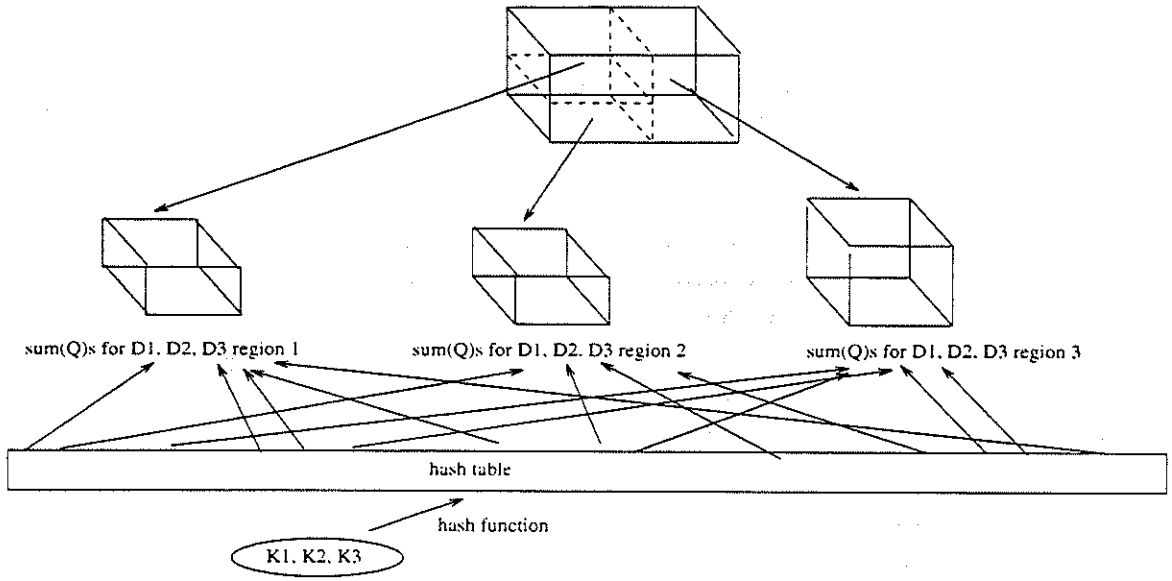



Figure 3: Hash table access for updates. Addresses in the hash table give the page address, slot number and unique identifier for a (D1,D2,D3) tuple in an $I$-leaf.

# 4 Discussion

Maintaining indexes always takes system resources. An index for group-by queries takes space. Every separate (D1,D2.D3) tuple requires an entry in $I$. Every separate (K1,K2,K3) tuple requires an entry in $I'$. Insertion requires updating at least $I$ and, when the tuple is a new one, both $I$ and $I'$. But this buys time in query processing. In the example we have looked at, if no index is kept, the group-by query would require accessing four relations and making three joins. Using the index $I$ should be much faster.

Even for the cube query, only the index $I$ and not the relations needs to be accessed if super-aggregates are calculated each time from simple aggregates. (Super aggregates can also be stored, with the penalty that an insertion (or delete or update) of a data record will require $2^n$ updates where $n$ is the cube dimension.)

The separate update index, $I'$, saves time only when tuples are inserted (or deleted) and only when the grouping tuple already exists in $I$. The alternative to keeping $I'$ is finding the values of (D1, D2, D3) using the relations R1, R2 and R3. We have suggested a variant of $I'$ using page addresses, slot numbers and unique identifiers and involving lazy updates of $I'$ when there are splits in $I$-leaves.

We think there will be many circumstances where keeping both the index $I$ and the update index $I'$ is worthwhile. We hope to have performance results soon to support this conjecture.

# References

[ELS95] G. Evangelidis, David Lomet, and B. Salzberg. The hB-$\Pi$-tree: A Modified hB-tree Supporting Concurrency, Recovery and Node Consolidation. *International Conference on Very Large Data Bases*, September 1995.

[ES93] G. Evangelidis and B. Salzberg. Using the Holey Brick Tree for Spatial Data in General Purpose DBMSs. *IEEE Database Engineering Bulletin*, 16(3):34–39, September 1993.

[GBLP95] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Draft*, February 1995.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of ACM/SIGMOD Annual Conference on Management of Data*, pages 47–57, Boston, MA, June 1984.

# $(DSS)^2$

## Decision Support Services for Decision Support Systems

Christiane Reuter, Tandem Computers Inc., HPRC

**Summary**

Decision Support Systems require a different set of services than the ones provided by today's relational data base systems. Continuous decline of hardware prices, makes customers think about huge DSS data bases, which keep a growing history of facts and a number of descriptive data around it. In general, all the programming budgeted for decision support systems goes into loading the data base only. Business users are expected to use one of the query, reporting or analysis tools, which are available on the market today. These tools more or less provide SQL as the only language to access the data base. They typically sit directly on top of the SQL catalog.

In order to be able to effectively work with the DSS data base, the business user needs a combination of the two features:

- a description of the data in such a way, that he/she recognizes the business entities
- a query language, which is closely related to the business problem

In addition, a decision support service should map the business data model into the logical model of the relational data base. It also should translate the business query into the SQL query to be executed by the underlying relational data base. These services should be provided by the data base system, which hosts the DSS data. In the long run, this is the only way to minimize data model maintenance. Also, the more knowledge is kept in the business data model, the better the chances for the data base system, to do things like materialized joins automatically.

### Current Situation

Customers, confronted with the need for building a new decision support system, regularly come up with nicely structured, but complex data models. When asked to write a "medium" complex SQL query against this model, in most cases they are not able to do this correctly. An SQL expert will usually come up with a 2 to 3 pass query, each having at least half a page of a where clauses, to solve the business problem. Any other person than this guru will give up reading this query after the first page, because he/she does not understand what it means in business terms. Unfortunately, customers think they are building a system to be used by their business users and not by SQL gurus.

To illustrate this situation, figure 1 shows the main entities of a 52 table logical DSS model. The size of the data base is about 250 GB for the pilot, with PDF_2 being the main contributor of more than 50% of the size of the data base.

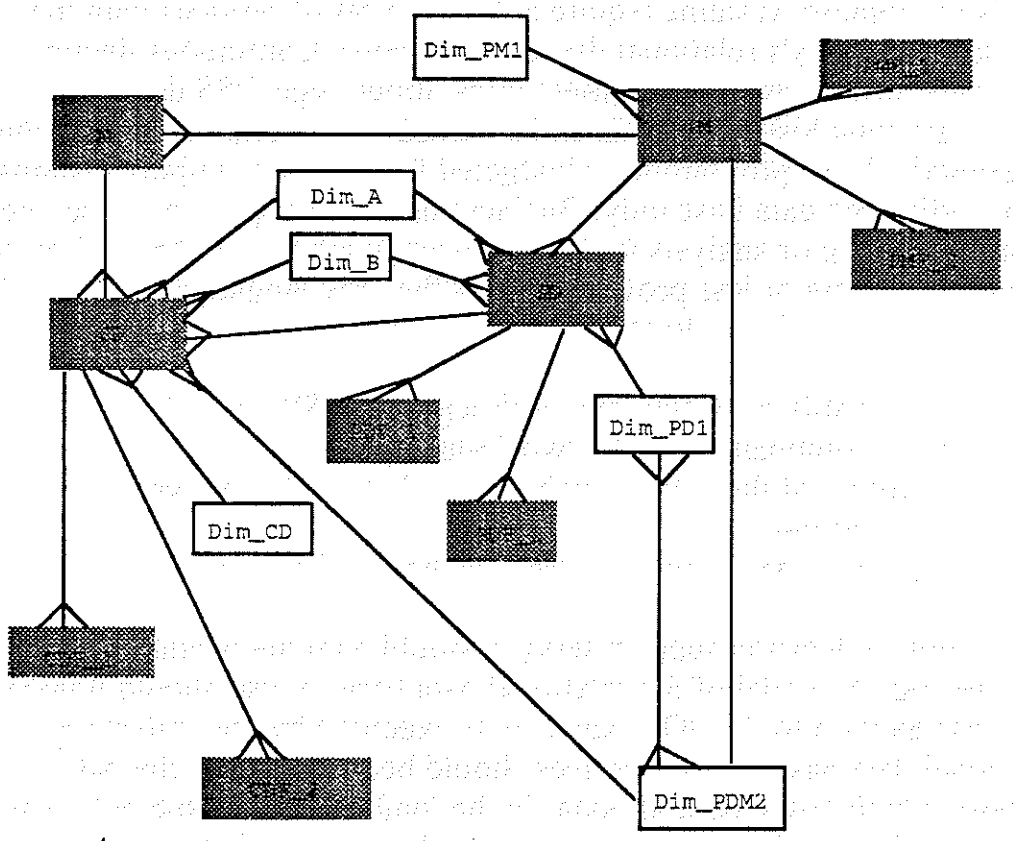



Figure 1: Main entities of 52 table DSS model

Figure 2 shows a graphical presentation of an arbitrary query against the above data base. This picture does not cover the additional complexity, which comes from a combination of inner and outer joins required to return the full list of an attribute combination. Figure 2 does also not cover the additional complexity coming from multi-dimensional queries to be coded in ANSI SQL.

Looking at figure 2, there is no doubt about that, that an average business user will not be able to correctly construct a query like this from the business terms. He/she will at most be able to pick the set of attributes, he/she is interested in, from the full attribute list of this join.

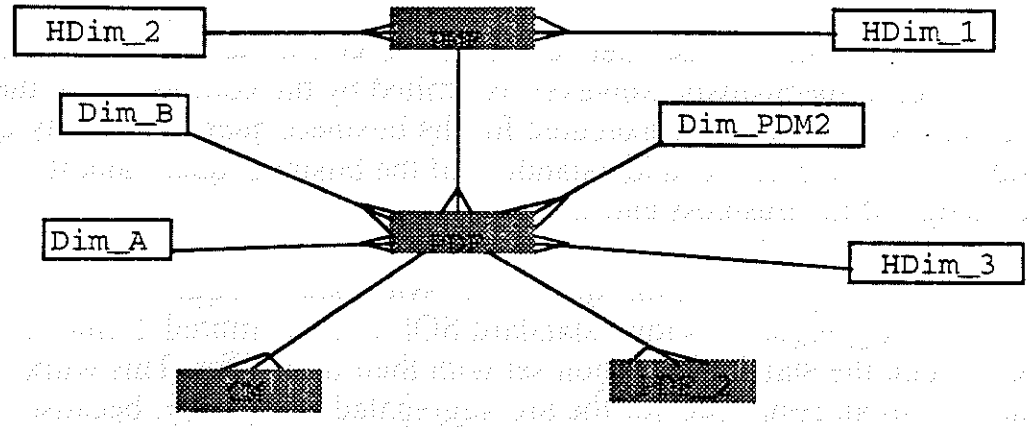



Figure 2: Graphical representation of an example query

De-normalization is a common technique to simplify SQL queries and to provide reasonable host performance at the same time. In some cases, de-normalization might be the only way to execute a query at all. Customers often express a strong resistance against this de-normalization plan, because the de-normalization process wipes out the nice data structures, they are so familiar with. This issue is usually raised by the customer's data base representatives. Presumably, the business user's view will be different. But even he/she will need some kind of structured information on top of the logical data model, which is closer to the business view of things.

To illustrate this, lets have another look at the above example data model. From the known query patterns, CM and CD as well as PM and PD are candidates for a major de-normalization. Each pair could be collapsed into one table named CMD and PMD respectively. Table 1 shows the number of attributes for each of CM, CD, PM and PD. Taking this into account, the customer's initial resistance against the de-normalization does not seem to be too far fetched. Small PC screens usually do not do a good job in displaying long lists of table attributes. And the only "structuring" technique, SQL offers today, is a naming convention for attribute names.

| CM | CD | PM | PD |
|---|---|---|---|
| 22 | 20 | 39 | 48 |
| total CMD: | 42 | total PMD: | 87 |

Table 1: Number of attributes for CM, CD, PM and PD

The trend among query, reporting and analysis tools is to provide some dictionary extensions. Mostly, they supply their own names dictionary to allow for business area dependent attribute names. Some go beyond this and

offer a mapping mechanism between business entities and data base entities. This mapping mechanism, however, is limited by the complexity of the SQL statement, which has to be executed for the business query. It usually gets build by the tool from its understanding of the business query and its knowledge of the business entities.

The two major tasks of a DSS query is to select and to aggregate. The support for aggregation within standard SQL is very limited. Some client tools extend the standard function set with their own offer. This works for relatively small result sets for the pre-aggregated query only, because usually, the client's aggregation is executed in main memory. Often, the aggregates only exist for that one query and cannot be re-used. In general, it does not seem like a good idea to make the data travel. The function should come to the data not the other way round. A data base engine, doing all required aggregation itself, could help both the business users and the overall system performance. This requires to extend the DSS data base system by user defined abstract data types. The SQL compiler should check the correct usage of these data types within an SQL query the same way as programming languages handle strong typing. This feature will help the business user to write correct queries from his/her business point of view, because an invalid operation will get rejected by the system.

**Proposal**

To overcome today's deficiencies as described above, a DSS data base management system should offer additional services on top of a relational data base system, which is proven to handle big data bases. This means both, good support for loading and querying a data base of greater 100 GB. Figure 3 shows a proposal for components to be added to the relational database system for better support of DSS.

- The DSS DBMS should hold both, the business data model and the logical data model, as well as a map between these two. DSS client tools should be able to query the content of the business data model and present it to the user in any suitable form. This presentation might be different, depending on
  - the user's task, e.g. spreadsheet work or multi-dimensional analysis
  - the user's role: data model specialist, business data analyst

  In general, data attributes of the business data model as well as of the logical data model shall be of user defined abstract data types.

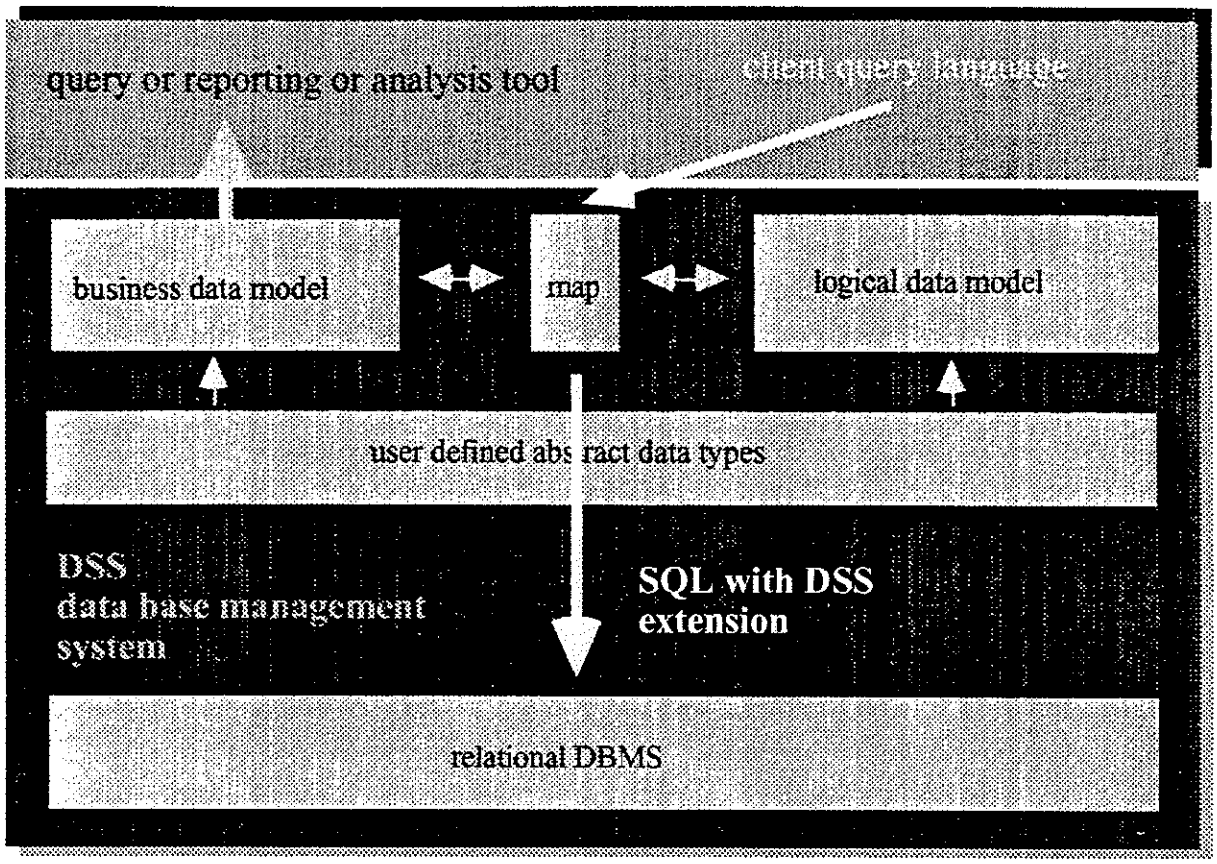



Figure 3: Proposal of a DSS data base management system

- The DSS client tool might pick any query language, suitable to help the business user write his/her queries. This can be structured natural language, full graphical or you name it. The client will pass the query in a standardized format to the data model map. The communication protocol between the client and the map requires further investigation still.

- The map is the hard of the query engine. It makes sure, that the user's query gets mapped correctly onto the logical data model, e.g.: it constructs all required joins, not the business user. It keeps statistics of common business query patterns. This information might be used later on to propose materialized views. To simplify the generated SQL queries, e.g.: proposals like Jim Gray et al.'s cube by operator should be added to the SQL language.

- User defined abstract data types will introduce that much programming into the DSS DBMS, as is required to insure that
  - data is used correctly from a business point of view
  - aggregation is done as close to the source data as possible

# Performance Prediction and Optimization in Workflow-based Applications

Dieter H. Roller
German Software Development Laboratory
IBM Germany Development Inc.
Hanns-Klemm-Straße 45
D 71034 Böblingen
droller @ vnet.ibm.com

## 1 Introduction

Two key factors, among others, determine whether an application can live up to the expectations: the system on which the application runs must be able to handle the system load generated by the application programs and database access activities, and the people must be able to handle their assigned workload.

System performance is determined in terms of CPU utilization, transaction rates, and database accesses [?, ?], people performance in number of people required to perform the activities in a given time frame. Typically the estimates for system performance and people throughput are quite inaccurate. Several reasons contribute to this: (1) the transaction rates and number of data accesses are usually guessed by a system administrator, as business analysts know nothing about transactions and SQL calls; they deal with processes, activities, business objects, and critical success factors, (2) the number of required people is quite often estimated without considering that the same person performs activities of different processes, and (3) the system and people performance are not correlated as is the case when an activity can not be completed in the estimated time due to the system not being able to process the user interactions fast enough.

This paper shows how the above situation can be enhanced in workflow-based applications. It proposes a consistent approach of building and designing business processes, where metric information is supplied during the various design phases, process simulation and database tuning is performed based on the collected information, the results of the actual execution are measured, and the results are compared with the supplied metrical information and simulation results.

A prototype has been implemented for IBM FlowMark for OS/2[?]. FlowMark's process model is that of a directed, acyclic, weighted, colored graph, where the nodes of the graph represent the activities to be performed and the edges the control flow between the activities.

## 2 Development Approach

The implementation of workflow-based applications may be performed as shown in figure ??. The whole process is iterative by its nature; if the results of a particular phase are not satisfactory, the process is restarted from a previous phase with modified assumptions, for example, when the average process execution time exceeds the goals set, one may restart by re-analyzing the designed business process.

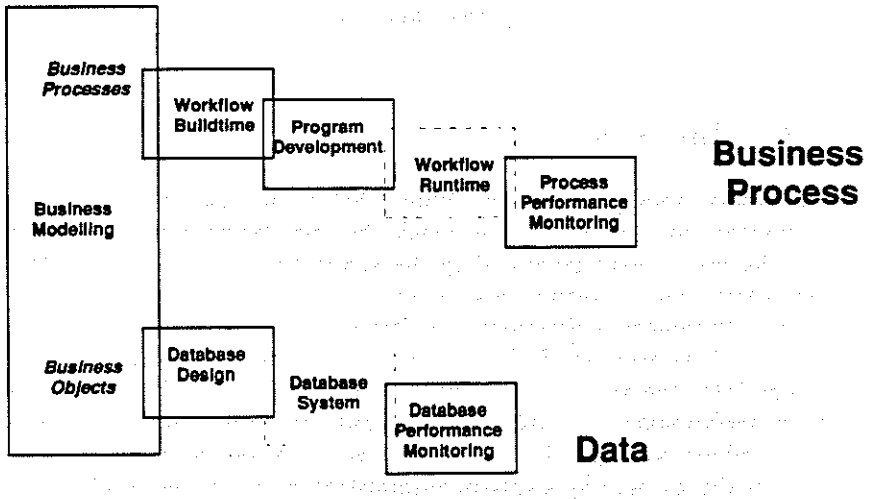



Figure 1: Development Phases

During *business modelling* the essence of the business process is captured: the process goals describing the envisioned properties of the process such as process duration or customer satisfaction, the top-level structure of the process with the most important business actions, and the business objects upon which the business actions operate.

The top-level process structure is refined via the *Workflow Buildtime* by breaking the business actions into sub-processes until the business actions are refined to a stage where they are implemented as programs. As business processes are exhibiting the behaviour of business transactions, spheres of joint compensation [?] are defined during this phase.

During *Program Development* the activity implementations are created or located, depending on whether they are new or existing programs. These pro-

grams can be, for example, OS/2 programs and CICS or IMS transactions. The implementation may be as a single program or as a client-server application. It may be an ACID transaction or no transaction. The workflow manager invokes these programs when the process is executed. The process thus constitutes a *federated networked application.*

The business objects are input to the *Database Designer* as local conceptual schemes, which integrates these local schemes into a global conceptual schema, transforms the global schema into a logical schema, and turnes this into a physical schema.

*Workflow runtime* manages the creation of process instances, the routing of the process instance through the network, the dispatching of tasks to users, the allocation of resources, and the generation of audit trail entries.

The *Process Performance Monitor*, through analyzing the audit trail, helps to obtain process performance relevant information, such as the average process duration, idle time for activities, or excessive notifications as work is not performed timely.

The *Database Performance Monitor* component can be used to determine, for example buffer pool usage, number of SQL calls, and I/O activity.

# 3 Metrical Information Collection

The base for performance predictions and database optimization is metrical information, which is determined for most parts of the process model. The information is collected for the enterprise level, the process level, and the activity level.

The enterprise level information is determined from enterprise information sources, for example payroll, system management information, and database management systems for things like types of computing resources, people's salary, and database sizes.

The process level information is usually provided by the business analyst : the number of processes started, the probability that a certain branch is taken in the process, the probabilty that an activity is repeated, and the size of the process input and output containers.

For the activity, the business analyst provides the process-related information such as the average time required to perform the activity including idle and wait time, and the averages size of the input and output containers. The program implementor provides the program related information which includes the SQL statements executed and the number of instructions executed.

# 4 Analysis

Analysis is performed in two categories : Analytical simulation and discrete simulation.

## 4.1 Analytical Simulation

Analytical simulation is used to calculate the required people and computer resources. If this turns out to be insufficient, any further analysis is superflous. The sufficiency of computer resources is evaluated by determing the CPU load on servers and clients, the network traffic caused by server-to-server communication and data passed from one activity to the next, and the transaction load on the database and the TP monitors. People resources are calculated by determining the amount of time required to perform the activities. The information derived for a process is then combined with the resource information derived from other processes.

## 4.2 Discrete Simulation

Discrete simulation is used to determine the impacts of multiple process instances competing for the same resources. Input to the simulation are scenarios describing which process models should be used in the simulation. The simulation component uses this information to drive the navigation engine of the workflow manager with the proper requests, such as process start and activity completion. The results are written to a file, which serves as input to create the simulation results. Typical simulation results are the probability distribution of process execution time and transaction rates.

## 4.3 Database Tuning

The results of analytical and discrete simulation can be used to tune the accessed databases. Using the transaction rates, one can determine the number and types of SQL calls against each database. This information and the current or estimated size of the databases provides sufficient input to the physical database designer to determine the proper database physics. Furthermore, it allows to determine how the size of the database changes over time. Collecting additional information, such as the distribution of keys, allows for example to detect hot-spots in tables.

## 4.4 Process Optimization

The results of the simulation help to optimize processes by calculating the costs associated with each process and activity. This does not only allow to compare different process models and select the most cost-efficient one, but it also helps to eliminate those activities where the costs for the activity exceed the estimated benefit. An insurance company may eliminate in a claims process the dispatching of an inspector if the costs associated with the inspection exceed the average amount in the claim.

# 5 Implementation

A first prototyp has been implemented to test the approach. The process is instrumented by using the description field of the process, activities, and control connector notebooks to provide metrical information such as the activity processing time (with distribution) and the probabilities associated with a certain control connector. This allows to use FlowMark's buildtime unmodified to populate FlowMark's database. A simple navigation engine has been constructed which reads this information from the database. This engine is fed with scenarious.

# 6 Summary

This paper outlined a methodology for performance predictions and database tuning for workflow-based applications by following a particular development approach, collecting metrical information during all phases of the development process, and using this information for analytical and discrete simulation.

A first prototype shows the validity of the approach and is the base for implementing this in FlowMark.

# 7 Acknowledgement

I would like to thank my collegues Jens Grotrian, Frank Leymann, Marc-Thomas Schmidt, and Alfons Steinhoff for the discussions we had on the subject.

# References

[1] H. Kobayashi, *Modeling and Analysis : An Introduction to System Performance Evaluation Methodology*, Addison-Wesley, 1978.

[2] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kauffmann Publisher Inc, San Mateo, CA 1993.

[3] *IBM FlowMark for OS/2.* Document Number GH19-8215-01 IBM Corporation 1994. Available through IBM branch offices.

[4] F. Leymann, *Supporting Business Transactions Via Partial Backward Recovery in Workflow Management Systems*, Proc. BTW '95, Springer 1995.

[5] A. Law and W. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, New York, 1982.

[6] J. Banks and J. Carson, *Discrete-event system simulation*, Prentice-Hall, Englewood Cliffs, 1984.

# On-Line Reorganization: A Position Paper

Betty Salzberg and Chendong Zou *
College of Computer Science
Northeastern University
Boston, MA. 02115

Rivka Ladin
Cambridge Research Lab
Digital Equipment Corporation
One Kendall Sq. Bldg. 700
Cambridge, MA 02139

## 1 Introduction

On-line reorganization is and will be a major problem for transaction systems of the 1990s and the 2000s. Mainframes are rapidly being replaced by workstation farms. Most large software systems must eventually be ported to these new cheaper hardware architectures. The new hardware architectures lend themselves to a client-server software architecture. Presentation services are off-loaded to client workstations. Even servers are no longer on mainframes, but instead on collections of workstations, each perhaps responsible for only a part of the database. Application software will have to be rewritten to conform to the new hardware and software architectures.

In addition, some companies may want to take advantage of newer object-oriented software and object-relational systems and new indexing and presentation options. They must then reorganize their data to make the best use of these new software systems.

But at the same time, large companies such as airlines and banks can not afford to be off-line for any significant amount of time. Companies want to take their complex applications and rewrite or modify them to work with new hardware and software without having to interrupt service. This means that either reorganization must be incremental or that massive duplicate systems must be constructed with a fast switch to the duplicate system when it is in place. It is not clear that anyone really knows how to accomplish either of these options.

We propose to look at this problem and formulate some reasonable and general principles for on-line reorganization of massive database systems. The goal is to (1) provide some basic reorganization utilities which could be used in more than one system and (2) give an analysis of the general techniques involved so that implementors have some guidelines for writing their own reorganization tools.

*This work was partially supported by NSF grant IRI-93-03403.

# 2 Basic On-Line Reorganization Methods

There are a few basic ways to do reorganization on-line. In this section we outline a few of the known techniques and point out their strengths and weaknesses.

## 2.1 New Copies with Change Logs

One approach is to make two copies of the whole database and use the old copy until the new one is ready. We must then keep some kind of log of changes being made to the new database so we can apply the new changes. Thus, there is a kind of "catch-up" done at the end of the reorganization, where database updates (modifications, insertions and deletions of database records) made during the reorganization are reapplied to the new structure if needed. Database searches must be directed to updated records during the reorganization process.

An example of this technique in the context of the creation of new secondary indexes can be found in [MN92] and [SC91]. In one option in [MN92], key inserts and deletes relating to an index being constructed are maintained separately in a side-file. The index-building runs sequentially through the data file and maintains a counter saying where it now is. If another transaction sees that the counter is ahead of where it wants to insert or delete a record, it uses the side file. After the whole file is scanned once by the index builder, it uses its list to create the new index. Then it processes the side file. When the side file is processed, new transactions use the new index.

## 2.2 Partial Indexes and Small Transactions

Another approach is to look at one moved record at a time and treat it somewhat like a deletion and insertion within one transaction, locking the record until all changes have been made as was done in [SD92]. This is however very slow as only one moved record is treated in each transaction. Every reference to the moved record in the database must be changed before the lock can be removed, and every record is handled separately.

In contrast, [Sto89] suggests reorganizing indexes in chunks. For example, a key range could be used to lock a collection of records and they would be unavailable during this time. Then perhaps some savings could be made in batching the records. For example, if the records are stored in key order, finding those records does not have to be done individually. Unfortunately, the references to a collection of records in a given key range are usually not stored together. For example, if the records are stored by *last-name* and there is a secondary index on *first-name*, there is no reason for any locality of reference in the secondary index. First names of people whose last names start with "A" are probably not clustered in any way.

## 2.3 Forwarding Lists

A third method keeps a forwarding list as in [OLS92] and [OLS94]. Instead of changing all references to a moved record in the database as is done in [SD92] before unlocking the record, the record is moved and a forwarding address is placed for the record in a list. After the address is in the list, the record is unlocked. When any transaction finds the address of any record in a secondary index or in a foreign key, for example, that transaction must consult the forwarding list to see whether the record has been moved, and if so, what is its new address.

Then, in a background process, updates to references to moved records can be made lazily. When all updates to references to a moved record have been made, the address of the record can be removed from the forwarding list.

The main drawback to this method is the fact that every transaction must always consult the forwarding list in a search until there is no forwarding list. This will be especially inefficient if the forwarding list does not reside in main memory. The advantage is that not all references to the moved record have to be changed immediately when the record is moved. (An assumption here is that there is no primary index for records which translates logical references to physical locations.)

## 2.4 Logical Names

A fourth method is to keep logical names for all records and refer to the records everywhere in the database using only logical names. Then only one index, the one that translates logical names to physical addresses, needs be reorganized when the data is moved. This is used in Tandem's Non-Stop SQL for example, where data is stored by primary $B^+$-tree key [Gro87]. When the records are moved, when a $B^+$-tree leaf splits, no other changes need to be made to the database. Of course, if it is decided to change from one attribute to another to be the basis of the primary $B^+$-tree, massive reorganization must be considered again. This was partially treated in [SD92], but here again only one record at a time was moved to the new primary organization.

More recently, we have been looking at dynamic hierarchical clustering, which is an extension of the use of the primary $B^+$-tree to be able to cluster records from separate relations according to an ancestor-descendent relationship [ZSL95]. This also enables records to be moved to retain clustering without having to change references to the records in other places in the database. It also has the problem that if one decides to reorganize and cluster according to another relationship, an entire new organization must be constructed and references to old primary keys must be changed to references to new primary keys.

# 3 What Next?

The main objective of incremental on-line reorganization is to change a part of the database without affecting on-going operations for very long. The portion of the database locked for reorganization should be small. But reorganizing only one record at a time, in a background process, may take too long. Ways to batch groups of records to be reorganized efficiently have to be found.

As a vehicle for examining reorganization, we shall first look at one very specific problem. This is the problem of taking data which is in one primary $B^+$-tree based on one particular attribute of a relation and reorganizing it into another $B^+$-tree based on a different attribute. We shall assume that there are many references to the primary keys elsewhere in the database—in secondary indexes and in foreign keys.

As a variation of this basic problem, we will also look at moving data referred to by page and slot number (RID) into a primary $B^+$-tree. A further variation involves moving collections of relations stored separately into our combined hierarchical structure, [ZSL95].

We hope by focusing on the specific but very practical problem of constructing new primary $B^+$-trees to be able to bring to light some of the general principles of massive on-line reorganization.

# References

[Gro87] Tandem Database Group. Non-Stop SQL: A distributed high performance high availability implementation of SQL. In *Workshop on High Performance Transaction Systems*. Springer-Verlag, 1987.

[MN92] C. Mohan and Inderpal Narang. Algorithms for creating indexes for very large tables without quiescing updates. In *Proceedings of ACM/SIGMOD Annual Conference on Management of Data*, pages 361–370, May 1992.

[OLS92] E. Omiecinski, L. Lee, and P. Scheuermann. Concurrent file reorganization for record clustering: A performance study. In *International Conference on Data Engineering*, pages 265–272, 1992.

[OLS94] E. Omiecinski, L. Lee, and P. Scheuermann. Performance analysis of a concurrent file reorganization algorithm for record clustering. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):248–257, April 1994.

[SC91] V. Srinivasan and Mike Carey. On-line index reconstruction algorithms. In *Workshop on High Performance Transaction Systems*, 1991.

[SD92] B. Salzberg and A. Dimock. Principles of transaction-based on-line reorganization. In *International Conference on Very Large Data Bases*, pages 511–520, 1992.

[Sto89] M. Stonebraker. The case for partial indexes. In *Sigmod Record*, Dec. 1989.

[ZSL95] C. Zou, B. Salzberg, and R. Ladin. Back to the future: Dynamic hierarchical clustering. Technical Report NU-CCS-95-09, College of Computer Science, Northeastern University, Boston, MA, 1995.

# APRICOTS - a workflow programming environment

Friedemann Schwenkreis
Institute of Parallel and Distributed High-Performance Systems
Department of Applications of Parallel and Distributed Systems
University of Stuttgart
Breitwiesenstr. 20-22
D-70565 Stuttgart, Germany
email: schwenk@informatik.uni-stuttgart.de

## 1 Introduction

The ConTract model [6], originally introduced as an extended transaction model, covers almost every aspect of workflow systems. In particular, the prototypical implementation of a ConTract system (APRICOTS [5]) shows the applicability of the concepts in the workflow area.

Since the programming of workflow processes is one of the most critical tasks of workflow systems, the programming environment of APRICOTS will be presented. Furthermore, the representation of the low-level description, which is directly used by the ConTract-Manager to execute a workflow process, is introduced. It will be shown that a so-called "predicate-transition-net" is a powerful means to express control flows. However, because it is difficult to read and maintain, this predicate-transition-net is not used directly by the programmer. Instead, APRICOTS provides the programmer with a graphical representation of the control flow.

We will follow a bottom-up approach to describe the programming environment. In section 2 the predicate-transition-net (PTN) is introduced. Section 3 introduces the high-level programming constructs to program a ConTract. In section 4 the depiction of the high-level constructs on a PTN will be described. In the last section we will conclude with a summary and we will give an outlook on our future work.

## 2 The predicate-transition-net (PTN)

A predicate-transition-net or *PTN* is a mechanism to define processes or control flows very similar to *petri nets* [2]. With the combination of the functionality of petri nets and the expressiveness of predicates a very powerful means is available to express processes in the workflow area.

There are three basic elements on which PTNs are built:

- Spots (S)
- Predicates (P)
- Transitions (T) with an associated attribute expected_tokens

Based on these elements, a directed graph can be built with the following constructs:

- Tuples (T, S) which express the connection of a transition T with a spot S.
- Triples (S, P, T) which express the connection of a spot S with a transition T, controlled by a predicate P.

The definition of a PTN can be animated by an execution mechanism which realizes the following rules:

- The execution of a PTN can be started by placing a token on a transition or a spot.
- If a transition T receives a token it *fires* if the number of received tokens is greater than or equal to the number of expected tokens. To *fire* means to pass a token to every spot $S_i$ for which a tuple $(T, S_i)$ exists.
- If a spot S receives a token every predicate $P_i$ is evaluated for which a triple $(S,P_i, T_j)$ exists. If the predicate $P_i$ evaluates true a token is passed to the transition $T_j$ of the triple.
- The execution of a PTN ends if no more recipients of tokens can be found when a transition wants to fire.

It is easy to recognize that PTNs are a very powerful mechanism to express the control flow of a process. They are very similar to the so-called Event-Condition-Action-rules [3] and have the same drawbacks on the programming level. Even so, PTNs are suitable for usage as a low-level representation of processes executed by workflow engines.

## 3 Programming in APRICOTS

As it has turned out, the original approach of ConTracts to define the *Script* by a (procedural) language is not useful. Since the users of a workflow system want to observe the progress of their processes, a mechanism to *visualize* the progress is needed. Furthermore, it is a requirement of users to have (almost) the same view of a *Script* in both the programming phase and the execution phase.

To fullfil these requirements a graphical programming environment for ConTracts has been developed in APRICOTS. This environment allows the programming and controlling of ConTracts on the basis of the same graphical representation of the control flow. The control flow is represented as a directed graph in which *Steps* are represented by ellipses, predicates by small circles and "flow paths" by lines. This approach follows the "drawing bubbles and arcs" principle but was enhanced by powerful editing functions.

The graphical editor for the definition of a ConTract is totally object-oriented and offers the following programming primitives (objects) to build up a process definition:

- *Steps*
- sequences
- loop blocks (for, while do, repeat until)
- branch blocks (if then else, n paths from m paths)
- parallel blocks
- transaction boarders
- blocks (similar to macro definitions)

All additional information needed to define the objects properly can be entered with specialized dialog boxes. For example, if the user opens the property dialog box of a *Step,* he

or she can enter the special information associated with *Steps*:

- Parameters needed by the *Step* and the corresponding *Context* variables
- Invariants which are needed for a proper isolation
- The definition of the conflict resolution policy to resolve concurrency conflicts
- *Steps* which should be started if the transaction of the current *Step* is aborted or a conflict resolution is finished
- The logical name of the associated *Step-Server* which has to be called to execute the *Step.*

The main advantages of the newly developed editor are the object-orientation and the direct usage of graphical elements for programming. Through the object-orientation the programmer of a ConTract is able to use and manipulate well-known constructs of high-level programming languages instead of "drawing a net". Furthermore the editor can easily be enhanced by new, not yet supported, constructs like e.g. the *parforeach* statement.

On the other hand the representation generated or used by the editor cannot be used directly for the execution. Since there is too much additional information integrated to support a proper visualization and manipulation, the direct interpretation of the editor output would cause a big overhead. To solve this problem, a translator has been developed to generate a PTN out of the editor's output. Once again, this is similar to an object-oriented approach: the editor is used to define a *template* of a ConTract which is used by the translator to generate an executable *instance* of a ConTract.

From an interoperability point of view the distinction between the two forms of the definition of a ConTract has another advantage: the people who define a workflow (e.g. workflow consultants) are very interested in the "migratability" of their processes, i.e. that a defined process should be executable on several platforms with different workflow engines. On the other hand, they want to avoid that the original business process (which is modelled in the programming environment) can be reconstructed from the information used by workflow engines. By using a very low-level mechanism on the execution level, enough information from the original definition can be hidden to avoid the unauthorized reconstruction of business processes (this is similar to the differentiation between source code and binaries of usual programming).

## 4 From the programming to the executable representation

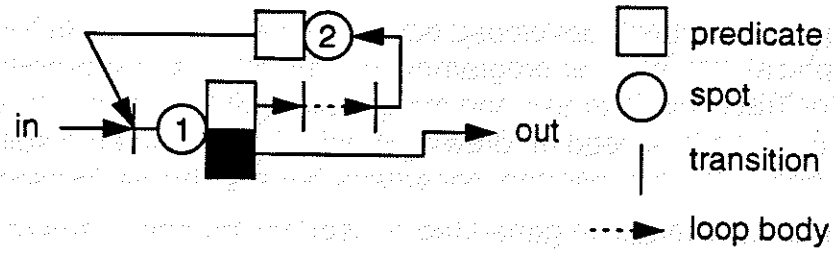
Several design decisions have been made to realize a proper representation of a ConTract's control flow as a PTN. Since users want to observe the progress of their processes with the same graphical representation as they have used during the programming phase, some information from the original definition must be integrated with the predicate transition net. Furthermore, it is very helpful for the implementation of advanced features of the executing engine to have some additional information beyond the "raw" PTN.

The first decision in the design was to have *Steps* of a *Script* represented by spots in the PTN. Since we want to handle the result of a *Step*, we can express the different paths for different results easily with predicate controlled connections of spots and transitions.

The second decision concerned the definition of the parts of a PTN which represent the high-level constructs. For each high-level construct, we had to make a separate design.

In this extended abstract we present one example: the *for-loop*.

A for-loop is defined similar to its definition in classical programming languages. There is a variable which represents a loop-counter (currently only integer variables are supported). A predicate is needed to define the end condition of the loop. Last but not least the "increment" of the loop has to be defined, i.e. in the current state of the programming environment a number has to be defined which is added to the loop-counter every time the loop is executed. The following figure represents the corresponding part of a PTN.




In this figure, the spot marked with 1 is only needed to pass a token to the two predicates and does not represent any action. The spot marked with 2 stands for the incrementation of the loop counter. It has turned out that for almost every high-level construct such "send-token" spots like spot 1 are needed which are called "pseudo-steps". Likewise the predicate associated with spot 2 is always true and only needed to fulfill the requirements of PTNs. The predicate represented by the black square after spot 1 is the negation of the other predicate of spot1 (end condition) which is again needed due to the requirements of PTNs.

As a matter of fact, the PTN generated from a high-level representation of a control flow creates redundant information. Since the visualization during the execution of processes should be done on the basis of the high-level representation, this can be a problem because some elements of the PTN are should not be seen on the user's level. These problems are solved in APRICOTS by assigning types to spots and predicates to be able to differentiate in elements which are relevant for the visualization and which are not.

Due to the fact that APRICOTS supports the user with a transactional environment - on the programming level as well as on the execution level - we decided to use a relational DBMS as a stable storage of the whole system. That means that templates and instances of ConTracts are stored in a RDBMS. The data needed (loop counters, number of expected tokens, etc.) during run-time is read from the RDBMS and the changes to this data are recorded in the RDBMS. This approach has the advantage of high portability and, what is more important, of allowing state changes of an instance to be done under the protection of transactions as they are defined on the *Script* level.

## 5 Summary and future work

The distinction of two levels of representation of a workflow definition has several advantages and is requested by users. First, it provides programmers with a simple graphical representation which is easy to read and maintain. Second, it avoids the overhead of directly interpreting the input of the programmer by translating that input to a PTN. Last, the PTN hides some of the proprietary information needed to reconstruct the business

process. With a simple and powerful mechanism on the execution level it is possible to migrate the execution of workflow processes as well as to support every "structure" needed on the definition level. Since there is currently no established standard for the low-level description of the control flow of workflow processes, we think that PTNs have a good chance to fill this gap.

We are currently working on the depiction of further high-level constructs as e.g. the *par-foreach* statement on a PTN. Furthermore we are investigating the structure of so-called "groupware working modes" and their depictability on both high-level programming constructs and PTNs.

Another working line in the development of APRICOTS is the connection of the ConTract Manager with a transactional environment like Encina from Transarc. This work aims at the definition of a "handbook" to enhance transaction monitors by mechanisms to run workflows directly concerning advanced transaction models like e.g. the ConTract model. Likewise we are looking at the upcoming standard of the Workflow Management Coalition to extend their approach with transactional semantics.

Since the external manipulation commands of a running process are currently limited to *start, stop, suspend* and *resume,* we are also working on the development of concepts to handle arbitrary (external) events. That means we want to integrate mechanisms in our engine to catch events from running ConTracts which will cause activities in other running ConTracts as e.g. the negotiation about resources. Furthermore the handling of time and events caused by a clock is another important point of our future work.

Last but not least we will have a look at the management of changes in the process definition while instances of this process are running. That means concepts will be developed on how and when a process instance can be changed. This work will lead us from the current view of statically predefined processes to the dynamic approach which allows the evolution of a process a run time like e.g. in INCAS [1].

## Acknowledgment

I would like to thank Prof. Betty Salzberg (Northeastern Univ.) for her great help during the preparation of this paper.

## References

[1] D. Barbara, S. Mehrotra, M. Rusinkiewicz: INCAS: A Computation Model for Dynamic Workflows in Autonomous Distributed Environments, Technical Report, University of Houston (TX), 1994

[2] Wilfried Brauer "Petri nets: central models and their properties; advances in Petri nets", Springer Verlag (Lecture notes in computer science). 1987

[3] U. Dayal, M. Hsu, R. Ladin "Organizing Long-Running Activities with Triggers and Transactions, ACM SIGMOD Record, pp. 204-214, 1990

[4] Ahmed K. Elmagarmid ed. "Database Transaction Models for Advanced Applications", Morgan Kaufmann Publishers, Inc., 1992

[5] Friedemann Schwenkreis "APRICOTS - A Prototype Implementation of a ConTract System: Management of the Control Flow and the Communication System" Proc. of the 12th Symposium on Reliable Distributed Systems, pp.12-21, 1993

[6] Helmut Waechter, Andreas Reuter "The ConTract Model" chapter 7 in [4]

# Transaction Processing for the Masses

(position paper)

Eugene J. Shekita
IBM Almaden Research Center
San Jose, CA, USA
shekita@almaden.ibm.com

## 1 Introduction

I've heard some people say that transaction processing (TP) systems are no longer interesting. It's not hard to see where this sentiment is coming from. Right now, you can squeeze close to 1500 TPM-C out of a uniprocessor. Over the course of an 8-hour work day, that translates into 720,000 transactions. How many companies can process that many transactions per day? Probably not too many. Moreover, that 1500 TMP-C will probably quadruple to 6000 TMP-C or 2,880,000 transactions per day by the end of the decade as processor clocks approach 500 Mhz. And if that's not enough horsepower, you can always move up to a symmetric multiprocessor, or a loosely-coupled cluster, or even to massively parallel system.

The bottom line is that 20 years of hard work has paid off. Through a combination of clever software algorithms and astounding improvements in hardware, high performance TP systems are now capabable of supporting even the largest business workload. Moreover, they can do it reliably using a mix of logging and hardware redundancy.

So what's left for a bunch of TP hardliners like the attendees at this workshop? Are TP systems really becoming a dead area as far as interesting problems go?

## 2 Transaction Processing for the Masses

Despite claims to the contrary, I think that a lot of interesting TP problems remain. I didn't always feel this way, of course. In fact, my outlook was pretty gloomy just a few years ago. What turned my thinking around was the World Wide Web.

During any given hour in the day, I can walk down my hallway at work, and out of the 50 or so people on my floor, I'm almost guaranteed to catch at least one person poking around the Web with a Netscape browser. One of the most popular Web sites at work, the Security APL stock-quote server, claims to get 400,000 queries per day [1]. And another even more popular Web site, the Yahoo Index, claims to get 2,000,000 queries per day [2]. The fact that the Web has managed to capture the imagination of so many people despite its relatively crude state and low-bandwidth links bodes well for the future.

How do the Web and transaction processing come together? In a nutshell, digital commerce. The Web will bring TP to the masses. Right now, the Web is mostly used as a read-only medium, and the companies that have set up shop on the Web haven't seen much demand for their services. The main problem, I believe, is a lack of trust. For now, most users simply feel uncomfortable sending their credit card number over the wire. Over time, though, I think that this will change. Network security will improve and encryption standards will emerge [3,4]. Digital money may even eliminate the need to send credit card numbers [5,6]. Little by little, users will grow more comfortable conducting business over the Web, and then digital commerce will take off.

But will digital commerce just be more of the same old stuff from the standpoint of transaction processing? I don't think so. Issues like security and reliability will probably need to be re-thought. For example, many databases today transmit passwords unencrypted from client to server. This is obviously inadequate for the Web. It also isn't clear that conventional transaction models are well-suited for digital commerce [7]. Some combination of transactional queues and replication may be more appropriate to deal with the latency and unreliability inherent in the Web. Finally, from the end user's perspective, all of this needs to be bundled together in an easy-to-use, shrink-wrap package that works right out of the box. I predict that ultimately Web browsers will include their own TP monitor of sorts.

# 3 Conclusions

We've solved the problem of how to build a system that's capable of supporting thousands of transactions per second. But interesting problems in tranasaction processing still remain. Digital commerce over the Web is about to bring transaction processing to the masses, along with a whole new set of problems to solve.

# 4 References

[1] K. Rodriguez, "Netscape Upgrades Web Servers," InfoWorld, pg 12, March 27, 1995.

[2] D. Plotnikoff, "Net Worth,", San Jose Mercury, page 1C, March 30, 1995.

[3] P. Fahn, "Frequently Asked Questions about Today's Cryptograpy," RSA Laboratories Tech Report, Sept 1993, http://www.rsa.com/faq/faq_toc.html

[4] K. Hickman, "The Secure Sockets Layer (SSL) Protocol," Netscape Communications Corp, Nov 1994, http://home.mcom.com/newsref/std/SSL.html

[5] CyberCash Inc., http://www.cybercash.com/

[6] DigiCash Inc., http://WWW.DigiCash.com/

[7] M. Sirbu and J. Tygar, "NetBill: An Internet Commerce System Optimized for Network Delivered Services," IEEE CompCon, March 1995.

# The Market Perspective - Ease-of-use and Heterogeneity

Alfred Z. Spector
President & CEO
Transarc Corporation
Pittsburgh, PA 15219

September 1995


As many of you know, my close friend and colleague, the most illustrious Dr. Gray, succeeded in obfuscating many important OLTP architectural issues in our 1993 HPTS debate. However, if we look beyond his grand-standing stunts, there was one kernel of truth amidst the rotten earful of arguments. That is, that OLTP strategies perceived to be "light" or "easy" will benefit in the marketplace. I believe achieving phenomenally great ease-of-use will prove to be the major challenge for this community over the next decade or two. The other challenge that we and our customers will have to understand are the trade-offs between unlimited choice within a system and cost/simplicity. I will get to both of these issues in this brief note.

*Function Shipping is Breaking Out All Over*

The TP Marketplace is burgeoning. If one considers the growth in home-brew solutions, the Web, database-centric TP Light, and targeted OLTP middleware, the number of network-based transaction processing solutions is growing very rapidly.

New markets are driving this. Electronic mass consumer marketing will grow with the increasing ability of desktop computers and networks to deliver more diverse services. Underlying this all is a vastly increasing amount of function shipping: from the desktop to the server and between servers. Things will keep humming nicely in the OLTP world as standard processing steps such as tracking customers, tracking merchandise, booking orders, and billing users are handled. But there will even be more uses, with transactions and multi-media combining forces.

And, traditional markets are driving this. Having made major productivity advances in the manufacturing sectors, businesses are reengineering their service sectors looking for equivalent efficiency gains. And in the service sectors particularly, information technology that is based significantly on function shipping techniques will underlie most systems. Whether it is down-sizing, rightsizing, networking, or integrating diverse applications, all result in the use of more transaction processing technology of one form or the other.

The Market Perspective - Ease-of-use and Heterogeneity

Objective metrics show the growth: Whether the use of stored procedures in databases, the growth of internet-based, request-response protocols, or the geometric sales of the new crop of OLTP environments, it is clear that transaction processing is growing, and will continue to grow very rapidly.

*So, Function Shipping is a Winner*

The technical debate is over. Nearly everyone agrees that the functional request, embodied as either a request/response message pair, a remote procedure call, a queued request, or a method invocation is clearly required in larger systems. The ability to encapsulate an object, an application server, or a legacy application across well-defined interface points and to invoke those interfaces has been recognized by all as an essential concept. Pure, unabstracted data-oriented systems plainly scale poorly in many ways. If any group were to fight function shipping, it would have been the relational database companies but they have embraced, and are even standardizing, a type of remote procedure call for accessing their servers.

*What else we understand*

Geometric growth in processing speed, primary, secondary, and tertiary memory and network bandwidth will continue for at least another decade. 1,000 MIPS platforms with a gigabyte of primary memory are not far from being the commodity platform. Local-area and wide-area networking will both continue to grow similarly giving us enormous bandwidth locally and a continuous set of trade-offs between bandwidth and cost in global communication. I thus believe that performance issues for the traditional processing required in nearly every OLTP environment will soon be last generation's problem. While this argument has traditionally been risky, Transarc customers are already telling us that Encina/DCE middleware performance is rarely a constraint for them.

We also understand the critical decomposition of systems into the many components that will form the middleware layers surrounding OLTP: Security, naming, the various types of resource managers, etc. I discussed these in my 1993 HPTS paper, and I think that paper is on-target.

Since 1993, however, we have learned that the consumer market may drive many more protocol and service choices than we anticipated. The verdict is still out on whether the companies serving the internet and consumer marketplaces will be able to invent new services rapidly or whether there will be adoption of the more traditional OLTP products into the consumer marketplace. So, we do not know if it's the productivity tools from the Netscapes or the Oracles that will have the biggest impact. But architecturally, things will be about the same.

In summary, I would say that we in this community now understand (mostly) how to build OLTP middleware that can provide the requisite functionality. With, for example, somewhere between ten and one hundred two-phase commit engines out there, many security servers, etc., we really can make things work. And, a word of congratulations to ourselves is in order: we have come a long way since the late 80s. We can now glue most any systems together.

The Market Perspective - Ease-of-use and Heterogeneity

*Two topics we do not understand*

I would like to turn the attention of our community to two topics we do not really have our arms around: These are of ease-of-use and the appropriate limits to heterogeneity.

*Ease-of-use*

Today, we ask customers developing new OLTP applications to choose the overall architecture of their application and which of hundreds of components they will use. (I note that as soon as they make a decision, a salesman for another competing product or technology makes it as difficult as possible for the customer to maintain his nerve.) For example, we make our customers choose the number of application severs, and decide whether they should utilize a cluster multiprocessor, a shared-memory multiprocessor, or a LAN-based solution. This is but one of hundreds of complex decisions required. These decisions are hard for our community, and we live and breath transaction processing.

And once these decisions are made, the programming, debugging, and systems integration remain difficult, as does the system administration and long-term management of an applications' evolution. While most complex, distributed transaction processing applications are today feasible (and we have come a long way), we haven't made them easy.

Therefore, I think the number one challenge in the marketplace has moved from the architecture and implementation of working middleware to our absolute, primary focus on ease-of-use: both from an application development and system administration perspective. I believe we need to turn the very considerable intellectual capacity of our community away from the things we love so much to real-user related problems.

Regarding the high-level architectural problems our customers face, very few are trying to solve these in a methodical, let alone automated, fashion. An unusual project is the Advisor project within IBM's client-server marketing group. It is attempting to codify client-server experiences and to develop working templates that are reusable by others. I believe that eventually, our systems software will have aspects of expert systems technology in them to enable customers to make rational decisions on applications and systems architecture. And, I do believe this is feasible.

If we are successful in this, customers will get automated advice on the structure of their environment and then just deposit business functions into a container. After the deposit occurs, the system will then ensure the right distribution, installation, replication, operation, and management of the new function. The result will be even faster growth of our marketplace.

*Limits to Heterogeneity*

I think we do not understand the appropriate limits to heterogeneity. As computer scientists, we believe in isolation and abstraction, thereby permitting the development of systems in a methodical and parallel fashion. Additionally, we believe that innovation is best served if the development of differing components can proceed along parallel lines. These are the basis for our faith in object technologies.

The Market Perspective - Ease-of-use and Heterogeneity

However, heterogeneity is expensive from both a vendor and customer perspective. Today, as an example, Transarc tests DCE and Encina with 8 resource managers, nearly 10 operating systems, many front-end tools, various TCP/IP and LU6.2 stacks, etc. And our products must be a little different on various platforms. For example, we emulate threads on HPUX, and use slightly different native threads on Solaris 2, AIX 4, and NT. Encina Console uses the JAM product on AIX for screen support, and Visual C++ on NT.

Some heterogeneity is a clear advantage for customers and for vendors. Customers have much more flexibility and the much greater price leverage with many –if not all– of their suppliers. For vendors, the diversity contributes to products that are probably more thoroughly tested, more versatile, and more long-lived.

But there are limits, too much heterogeneity leads to procurement and management nightmares for customers. For vendors, it leads to rapidly increasing testing and release costs. And, the diversity that does creep into a system (e.g., slightly different systems management screens) makes the system more difficult for the both vendor and customer alike.

There's a reasonable balance to provide the benefits of open systems, without the enormous expense of excess diversity. I think there is work and understanding to occur here, both in a technical and economic sense.

*Conclusion*

The OLTP research and development community has made enormous progress in the last decade. Distributed transaction processing in an open systems environment is not only feasible: it's getting more commonly used every day. How the products will play out given the growth of the web and consumer market place is not clear; on the other hand, regardless of packaging, we understand the low-level architecture of distributed transaction processing environments pretty well.

We must now recognize that the core components on which we have been focusing for years may not be the area requiring the greatest innovation. In this paper, I've talked about the challenges of understanding ease-of-use and the appropriate role of heterogeneity. There are many other challenges beyond the core components as well. As an industry, meeting these new challenges will yield the greatest benefits.

# Why PC databases are important for High Performance Transaction Systems

*Peter Spiro*
*Microsoft Corporation*
*petersp@microsoft.com*

There is a natural and inevitable trend in database systems software. The trend is that customers always pull products into more high-performing, more functionally-rich. more demanding spaces. For example, if a database system can comfortably handle 50 GB databases, and 100 users, it's virtually guaranteed that customers will load up the database to its usable limits, and then demand more. They want 100 GB and 500 users. If a system can run 1000 tps, users will configure it for 2000 tps. It's not really an evil plot designed to make life difficult for software engineers, it's simply that customers very often undersize the problems/solutions; or they fail to accurately predict new and important uses of their database. There is a corollary to this first principle, and that is, it's incredible how customers can be so successful using products that are either ill-conceived for the particular problem space or just simply not meant to do a particular job. In fact, this dictum applies across the board to almost every invention known to man. Examples are: using a rolling desk chair as a moving cart, a butter knife as a screwdriver, a plastic garbage bag as rain gear, a spreadsheet as a multi-user database system etc., etc. The key point here is than man is very creative and resourceful; he will find a way to make the thing work.

Now where have these principles brought us to in the realm of transaction processing and database systems?

At one snapshot in time, say around 1990. We had a nice three-tiered structure for database products. In the low-end there was Fox, Access, Paradox, Dbase and others. In the mid-range, there were the traditional relational database vendors: Oracle, Sybase, Ingres, Informix, Rdb, others. At the high-end, Tandem, Teradata, IBM's various products, others.

The mid-range players were doing a pretty nice job handling 5-50 GB databases. That's where the bulk of their 'big' databases were. These systems worked pretty well, they supplied a portion of the 7x24 features that customers wanted. They could supply 100s of transactions per second depending on the hardware. Then they fell prey to the principles described above. Many of their customers began pulling them upwards towards database sizes in the 100s of GB, 100s to 1000s of users, 1000s of tps, complete 7x24 features. Parallelism was now needed to query these large databases. Parallelism, reaching almost mythical proportions, became the mantra of all these companies. High-priced consultants could actually make a living just talking about which vendors had which capabilities!

These mid-ranged systems on steroids have become pretty successful in the high-end, Sure they had some problems with the very large bids, but over time they will succeed in the

high-end because they'll build market-share and they'll slowly but surely add the appropriate features/performance. And they've brought their niceties from the mid-range into the high-end. That is, these systems are somewhat easier to use, they're cheaper, and they have many more customers and programmers that are familiar with the product's features and quirks. The bottom line is they can pretty much do the job and customers can deploy solutions pretty quickly. Much quicker than system solutions of ten years ago. In effect, the traditional high-end players were tp-heavy and db-heavy; the new players in this space are tp-lite and db-lite.

Anyway because of the exodus from the mid-range to the high-end we now have a vacuum in the mid-range. It's not really a vacuum, it's more of an opportunity. That is, the traditional mid-range players had an option: they could have noticed what was going on in the low-end space and adopted some of the low-end conventions/tactics/features, or they could have pulled/pushed into the high-end. There's no question, they all looked upwards. They focused on down-sizing opportunities instead of up-sizing.

Now just as the mid-range players were inevitably pulled upwards, the low-end pc database systems are going to encroach into the traditional mid-range space. Often times these systems won't even initially be recognizable as a tp/database system as we know them. A good example is Lotus Notes. Although it doesn't have the same capabilities that we traditionally associate with a relational database system, it certainly is solving many data management problems; it's even threatening to become a complete application development environment. Another good example is Microsoft Access. There are probably more multi-user Access solutions than any other traditional relational database. Both these products, instead of focusing on robust, high-performing engines, concentrated on allowing developers/users to easily and quickly build solutions that solved real customer problems. Remember the plastic garbage bag.

Transactions? Sure they like those sometimes. Locking? Certainly not degree 3. SQL? No way, too complicated, user-friendly query tools will generate the SQL. Time for deployment of the application? Now! In fact, in the realm of quick application deployment, the PC systems are as far ahead of the mid-range players, as the mid-range was to the high-end. Whereas a traditional high-end deployment might take one to two years, and the mid-range systems brought that number down to say six months, the low-end database systems allow deployment in weeks or even days!

Compared to the previous mid-range players (tp-lite/db-lite), these systems are no-tp and db-very-lite. So why might they become important to the high-end space? First, as described above, they WILL be pulled into the mid-range. It's simply a law of nature. Then the question becomes, how successful will they be when they attempt to cover both ranges?

Whenever a product attempts to cover multiple usage paradigms it becomes very tricky to design features appropriately. For example, the PC databases use a file sharing model. Backup consists of simply copying the database file; often there is no log that will be used

for media-failure. A mid-range system usually has a standard backup mechanism and a log, but not too many bells and whistles. And a high-end system will have online incremental backups, a sophisticated automated log management scheme that rolls over to multiple disks and includes operator notification, and the recovery procedures should work against a small granule such as a page (not a table, file, or database).

Ok, so the task before the PC database systems is how to design their product to cover multiple target spaces, in this case, the low-end and mid-range. They need to move from db-very-lite to db-lite. Interestingly this is the same problem the mid-range players faced. Some of the vendors using 'band-aids', simply hacked on more features to their existing architectures. Big mistake. Using this process you end up with some short term success but over time it becomes very difficult to add/modify functionality. Another model is to use some sort of add-on product which leverages your traditional product so it will scale to multiple machines. This is architecturally cleaner than the first option but it's not a very efficient model. It's not really addressing the core problems. The third approach is to rearchitect and rebuild the product. This is the best approach.

But even so, the 'redo' model has to be done correctly. For most features, it's very difficult to make the right design tradeoffs so that the feature works correctly under a wide range of user requirements. The solution is **knobs**. The system has to be designed so that a feature can function in one of N ways. For example, a commit sequence might be able to flush data pages at commit (no redo log), or it might use WAL and only flush after-image records to the log. The proliferation of knobs is both a curse and a blessing. It does indeed allow a system to be tuned for different purposes. For example, Rdb/Oracle which holds the world record for TPC-A tps at 3692, can also be configured as a single-file database system with no after-image log. Such a configuration would limit performance to tens of tps. The curse is that usually there are only three people in the whole world who can tune such a system appropriately.

Hence the next step beyond knobs is automatic feature configuration and tuning. This will be especially important in the low to mid-range. These systems can't afford to and don't want to have sophisticated database administrators. The automatic capabilities must cover physical design optimizations (data placement); they must cover which 'flavors' of the algorithms to use (flush at commit/WAL, local buffers/global buffers, versioning/no versioning, locking/no locking); they must cover logical designs (ER design, access paths, views); and they must cover dynamic resource management (memory mgmt contention between query execution and buffer pool, thread mgmt).

Ok, so why should the high-end systems be concerned with these activities?

If a low or mid-range player redesigns and rebuilds their core engine, and creates a scaleable system which can auto-configure itself to run appropriately for either domain, they'll have a very attractive product. It's very likely that in the redesign process the system will also be built to scale to the highest end. And the auto-configure/auto-tune mechanism will pay great dividends on the high-end. Certainly many of the utilities and

other services in the high-end won't initially be present but these can be added just as the mid-range players added them to their products previously.

And finally there's one last piece of the puzzle that's needed to allow a rebuilt PC product to scale to the high end: an integrated TP monitor. But a TP monitor playing with the PC product will have to have some additional functionality. The products and users in this space aren't going to be forced into only using database technology. They want to merge spreadsheets, textual documents, Notes-like data conferences, database tabular data, project management information, etc., into one seamless integrated data system. The new TP monitor would have to be able to coordinate data from all these sources. This new TP monitor, while it might not look like it, is really TP-heavy, just used slightly differently.

Then it's only a matter of time before this low-end/mid-range product is pulled into the high-end to square off against the other high-end players. But along the way the new product may have changed the rules of the game, and it will have market share, price, and simplicity as its advantages, just as the previous systems did in their earlier migration. So it's inevitable, just as mini-computers replaced mainfames, and PCs replaced minis, and tp-lite replaced tp-heavy, and the mid-range database products replaced the high-end products, the natural cycle will continue: some simple existing or even unbuilt product will evolve to dominate the database industry.

# *Coping with Lock Contention in HPTS*

## *(Position paper for HPTS'95)*

**Alexander Thomasian**
**IBM T. J. Watson Research Center**
**30 Saw Mill River Road**
**Hawthorne, NY 10532, USA**
**athomas at watson.ibm.com**

### 1. Introduction

It has been ten years since the presentation of the "transaction (txn) pipeline processor" (TPP) as a technique to reduce the level of lock contention in high performance OLTP systems [Reut85]. In the first execution phase txns are pre-executed concurrently to prefetch the data required for their execution, while taking advantage of the multi-programming effect. No Concurrency Control (CC) is applied in the first phase, which we call *virtual execution* (VE) [FrRT90],[FrRT92], while as a substitute for CC txns are executed serially in the second phase. This already short phase, since no disk I/O is required, can be made even shorter in a specially tuned system. A prototype implementing TPP is reported in [LiNa88].

Specialized techniques to reduce the level of lock contention in DBMSs for high performance OLTP, such as record rather than page level locking and short-term locks for indexes [MHL+92],[MoLe92] have kept the level of lock contention at a low enough level that TPP is not required for the moment. We contend that while this is so for mature DBMS implementations, the cost of implementing sophisticated locking methods and the run-time overhead may make TPP desirable again, especially for new (object-oriented) DBMSs.

Two-phase processing (2PH) methods described in [FrRT91],[FrRT92] generalize and extend TPP. In what follows we describe 2PH methods and contrast them with TPP. We then point out some problem areas and techniques to cope with them. In this note we will not concern ourselves with the data contention between short online txns and long read-only queries, which can be handled using specialized methods such as versioning [MoPL92].

The paper is organized as follows. 2PH methods are described in Section 2. This Section touches upon another set of methods based on limiting the wait depth of blocked txns in locking [FrRT92], which is a viable alternative to 2PH. In Section 3 we discuss methods for coping with the use of uncommitted data, which may lead to an incorrect path in txn execution. Prefetching as part of txn execution is discussed in Section 4.

### 2. Two-Phase Processing Methods

Two-phase processing (2PH) methods differ from TPP in that: (i) The first execution phase of a txn may be in VE mode or involve CC. In the latter case some txns may complete their execution at the end of the first execution phase and commit, resulting in significant savings by eliminating the second execution phase. (ii) Multiple txns may be executed concurrently in the second phase, which eliminates the limit on the maximum throughput (the inverse of the mean serialized re-execution time) set by 2PH. In fact it may be possible to categorize txns into multiple non-interfering classes from the lock contention viewpoint, in which case txns in different classes can proceed concurrently. 2PH involves CC at least in the second execution phase and hence introduces more overhead.

The simplest form of 2PH based on optimistic CC (OCC) executes txns according to the optimistic *die* policy in the first phase, i.e., a txn known to be conflicted is allowed to continue its execution to the end. At the end of their first execution phase txns either validate successfully and commit, obviating the need for re-execution, or they are re-executed according to the optimistic *kill* policy, since there is no advantage in executing conflicted txns to the end when objects required for re-executing a txn are retained in a sufficiently large database buffer.

The optimistic die/kill policy, i.e., optimistic die in the first phase and kill in all following phases, outperforms an optimistic kill/kill policy if the following conditions hold: (i) the system is *data contention* rather than *hardware resource contention bound,* i.e., running the system in optimistic die mode does not result in saturating processors (presumed to be the bottleneck hardware resource); (i) *access invariance* prevails, i.e., txns access the same set of database blocks or more generally database objects when they are re-executed. We distinguish between logical and physical access invariance, where the latter implies an access to a data item collocated with the previously accessed item. It is contended in [FrRT92] that the changes to the database state are not severe enough to modify the outcome and hence execution path followed by restarted txns, which is a justification for *serialization* as the correctness criterion for CC. Serialization implies that any execution (commit) sequence is acceptable, as long as it corresponds to some serial execution of txns, but this is not necessarily so in all applications, e.g., stock trading applications [PeRS88]. Provided access invariance prevails, the second-phase of txn execution will be very fast since no disk I/O is required, which implies that the possibility that a second phase txn is conflicted by a first or second-phase txn is rather small. The variability of txn response time can be reduced by ensuring that the second execution phase will always be successful, provided that access invariance prevails. This is accomplished by using a *hybrid CC method,* i.e., the optimistic die policy in the first phase and lock preclaiming or static locking in the second phase [ThRa90],[FrRT92]. Given that the number of txns executing in the second phase is much smaller than the number executing in the first phase, txns in the second phase tend to introduce less lock contention. The multi-programming level for executing second phase txns can be limited to reduce the level of data contention in this phase. It follows from the simulation results in [FrRT92] that under such circumstances the CC method for the second phase has little effect on performance.

OCC implies a *private workspace* paradigm, i.e., data items accessed by a txn are copied into txn's woskspace to which txn updates and further accesses are made. A successfully validated txn externalizes its updates after logging them, but this is fraught with numerous difficulties [Moha92]. It has been argued that OCC is inefficient in dealing with aggregate variables constituting hot-spots [GrRe92], but this is similarly a problem in locking which is handled by the "field calls" technique of IMS Fastpath [GrRe92]. Other techniques to reduce the level of lock contention in this manner are beyond the scope of this discussion.

Optimistic methods result in a significant degree of wasted processing due to txn restarts as compared to other CC methods. In fact according to the *quadratic effect* the probability of txn restart increases quadratically with its size (the number of objects accessed by the txn) provided that database objects are accessed uniformly [FrRT92]. Txn level checkpointing using (volatile) savepoints or syncpoints [GrRe92] shows a rather limited effect on reducing wasted processing due to txn restarts, except when the access to hot-spots at the end of txn execution is preceded by a single checkpoint [Thom95].

Because of the excessive wasteful processing and implementation difficulties associated with OCC, 2PH methods based on locking have been devised [FrRT90],[FrRT92]. Since deadlocks associated with standard locking are rare [GrRe92] and the performance degradation in this case is due to txn blocking, an appropriate method is required to *induce* txn aborts to reduce the lock contention level in the system. The running priority (RP) method [FrRo85] is one such method, which aborts a txn holding a lock requested by an active txn. This action is expected to improve performance since it partially fulfills the *essential blocking property* [FrRo85], that blocking is acceptable only when the requested lock *is held by an active txn doing useful work,* i.e., provided that the txn will not be aborted at a later time. Thus first-phase txns are run according to the RP method, except instead of aborting txns they continue their execution in VE or in *optimistic die* mode. In both cases a txn releases all of its locks. Undo log records need to be maintained in main storage to ensure inexpensive txn rollbacks, since they are considerably more frequent than in the standard locking case.

VE is similar to the optimistic die method, except that in the presence of locking methods, which generally do not follow a private workspace paradigm, a txn may be allowed to read

uncommitted data (a latch can be used to ensure that data block is not being currently modified). In fact latching is adequate in TPP and the release of latches will coincide with txn commit in this case (see [GMSa92] for locking in main storage databases). In the case of the hybrid OCC method txns in optimistic mode may defer their access to currently locked objects [ThRa90]. A private workspace paradigm is used in VE mode, but a txn does not externalize its updates upon completion. The optimistic die method has the advantage over VE that a previously conflicted txn may discover at the end of its execution that it can validate successfully. This is possible when the (first-phase) txn that obtained the lock released by the txn is itself aborted releasing its lock (the validation of the completed txn can be deferred if it has not encountered any conflicts and the other txn is still in progress).

Standard locking or lock preclaiming can be used for the second-phase of txns running with RP in the first phase. In fact for realistic parameters for disk access times and database buffer hit ratios the number of txns in the second phase tends to be an order of magnitude smaller than those in the first phase. Prioritizing the execution of second phase with respect to the first phase txns at the CPU can be used to further reduce the execution time of second phase txns. In effect there is very little lock contention among txns in the second phase. Txns requesting locks held by second phase txns are blocked (even though the second phase txn may be blocked itself), while a second phase txns lock request will result in the preemption of the lock held by a first phase txn (although it may be active). Deadlocks are still possible in this phase if it uses standard locking.

It was verified by simulation studies [FrRT90],[FrRT92] that using the RP method in the first phase of txn execution results in an improvement in performance with respect to an optimistic method when the system is hardware resource bound (in systems with "infinite resources" optimistic methods outperforms all others). In fact an adaptive method is desirable when the system initially runs with OCC, but when the processors become saturated it switches from OCC to RP. The system may switch to more conservative versions of RP based on the number of txns blocked at a wait depth of one or the wait depth exceeding one [Thom94]. Alternatively, the (symmetric) RP method [FrRT92] can be modified to take into account the progress made by txns in deciding which txn to abort. The wait-depth limited (WDL) method with the *restart waiting* paradigm of delaying the restart of aborted txns until conflicting txns have left the system outperform all others for a wide range of parameters [FrRT92] (except OCC and 2PH in high contention systems with infinite resources). Trace-driven simulation studies in [WHMZ94] also show that this method performs quite well compared with other methods proposed for coping with thrashing in standard locking.

*Txn spawning* is a method to speed up the execution of blocked txns [FrRT92]. The spawned sub-txn runs in VE mode, prefetching data required for the main txn's execution, while a private workspace paradigm is used from this point on. It might be advantageous to run the spawned sub-txn to the end, even when the main txn is unblocked before the spawn completes execution, since no spawning will be required if the main txn is blocked again. The main txn switches from the first to the second phase when the spawned sub-txn completes, regardless of whether it is active or still blocked at that point.

2PH paradigms have also been considered for real-time TP applications [O'RP92].

### 3. Techniques to Cope with Access Variance

Prefetching and speculative execution of two paths of a branch are techniques used in high-performance processors to speed up execution. One method to cope with access variance is to allow multiple instances of txn execution based on the possible values of variables, as in "polyvalues" [Mont78] and "branching txns [BuTh94]. Polyvalues were intended to cope with failures in a distributed database environment through a bookkeeping tool described in [Mont78], which keeps track of several potential values for a data item depending on the commit or abort of precommitted txns. For example, when two pending txns $T_1$ and $T_2$ have accessed a bank balance, there are three polyvalues specified as $(Balance_{init} - Withdraw_1 - Withdraw_2, T_1, T_2)$, . $(Balance_{init} - Withdraw_1, T_1, \neg T_2)$, and $(Balance_{init} - Withdraw_2, \neg T_1, T_2)$, where $T_i$ (resp. $\neg T_i$) indicates the commit (resp. abort) of txn

$T_i$, respectively. Thus items remaining locked due to the failure of a commit coordinator can be accessed conditionally by other txns. The escrow paradigm [O'Ne86] assures that a txn can follow a certain path regardless of pending operations on an aggregate variable, while the polyvalues paradigm considers the possibility that one or more txns may fail.

*Branching txns* paradigm [BuTh94] allows a txn to pursue "multiple" paths, based on the fact that multiple versions of an object may be available (all versions reside in main storage for efficiency reasons). A txn may introduces additional branches when it encounters additional objects with multiple versions. Note that only one execution instance out of many can succeed, i.e., most of the processing in the system is wasted, which is the reason why this scheme is proposed in the context of a highly parallel system. A completed txn cannot commit until all txns affecting it are committed. Load control methods to cope with this wasted processing are obviously required as mentioned but not elaborated in the paper. Note that branching is not required when different versions of an object result in the same path.

This discussion is also relevant to optimistic methods, where a txn accesses a committed value of a desired data item, while there may be multiple updated instances of it in the private workspace of other txns. In effect a branching txn paradigm can be followed in this case, such that one of the instances of txn execution may be successful.

In fact in both locking with multiple versions and optimistic CC with the branching paradigm, new branches need to be created as new instances of data values accessed by txns are created. This is especially inefficient when the same original txn accesses a data value again and again. This is why some CC methods make the assumption that an object is updated once after its X-lock is obtained and never again [AEAL94]. Of course this programming paradigm would help improve the efficiency of the branching txn paradigm as well.

### 4. Prefetching Inline with Transaction Execution

Prefetching as part of txn execution can also be useful in reducing txn response time and hence the lock holding time and the level of lock contention. As noted in [Reut86] TPP is a "brute force" method to predict the reference pattern of a txn. Given the class of a txn and its input parameters, it is possible to predict the data objects to be accessed by a txn, i.e., the execution of a txn starts only after all objects required by it have been prefetched. A txn oriented prepaging method is described in [WeZo86], which minimizes the time txns spend in the system. One may distinguish between a *strategy-based* and a *training-based* predictors, where an explicit programmed strategy is provided in the former case, while the latter requires extensive monitoring [GeKe94]. In fact a combination of the two methods may be required for more complex txns, which are affected by data dependencies. *Sequence prediction* techniques are applicable to the problem . More specifically, the TDAG algorithm [LeSa94] limits the storage by retaining the most likely prediction contexts and forgetting less likely ones. This remains an area of further investigation.

### References


**[AEAL94]** D. Agrawal, A. El Abbadi, and A. E. Lang. "The performance of protocols based on locks with ordered sharing," *IEEE TKDE 6*,5 (Oct. 1994), 805-818.

**[BuTh94]** A. Burger and P. Thanisch. "Branching transactions: A transaction model for parallel database systems," *Directions in Databases: Proc. 12th Nat' Conf. Databases, BNCOD 12*, D. S. Powers (Ed.), Guildford, UK, July 1994, pp. 121-136.

**[FrRo85]** P. Franaszek and J. T. Robinson. "Limitations of concurrency in transaction processing." *ACM TODS 10*,1 (March 1985), 1-28.

**[FrRT90]** P. Franaszek, J. T. Robinson, and A. Thomasian. "Access invariance and its use in high contention environments," *Proc. 6th ICDE*, Los Angeles, CA, Feb. 1990, pp. 47-55.

**[FrRT92]** P. Franaszek, J. T. Robinson, and A. Thomasian. "Concurrency control for high contention environments," *ACM TODS 17*,2 (June 1992), 304-345.

**[GeKe94]** C. A. Gerlhof and A. Kemper. "A multi-threaded architecture for prefetching in object bases," *Advances in Database Technology-EDBT'94* Cambridge, UK, March 1994, pp. 351-364.

**[GMSa92]** H. Garcia-Molina and K. Salem. "Main memory database systems: An overview," *IEEE TKDE 4*,6 (Dec. 1992), 509-516.

**[LaSa94]** P. Laird and R. Saul. "Discrtete sequence prediction and its applications," *Machine Learning 15*, (1994), 43-68.

**[LiNa88]** K. Li and J. F. Naughton. "Multiprocessor main memory transaction processing," *Proc. Int'l Symp. Databases in Parallel and Distributed Systems*, Austin, TX, Dec. 1988, pp. 177-187.

**[Moha92]** C. Mohan. "Less optimism about optimistic concurrency control," *Proc. 2nd Int'l Workshop on Research Issues on Data Eng.*, Tempe, Az, Feb. 1992, pp. 199-204.

**[MHL+92]** C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwartz. "ARIES: A transaction recovery method supporting fine granularity locking and partial rollbacks using write-ahead locking," *ACM TODS 17*,1 (March 1992), 94-162.

**[MoLe92]** C. Mohan and F. Levine. "ARIES/IM: An efficient and high concurrency index management method using write-ahead logging," *Proc. 1992 ACM SIGMOD*, San Diego, CA, June 1992, pp. 371-380.

**[MoPL92]** C. Mohan, H. Pirahesh, and R. Lorie. "Efficient and flexible methods for transient versioning of records to avoid locking by read-only transactions," *Proc. 1992 ACM SIGMOD*, San Diego, CA, June 1992, pp. 124-133.

**[Mont78]** W. A. Montgomery. "Robust concurrency control for a distributed information system," *MIT-LCS-TR-207*, MIT, CS Lab., Cambridge, MA, Dec. 1978.

**[O'Ne86]** P. E. O'Neil. "The escrow transaction method," *ACM TODS 11*,4 (Dec. 1986), 405-430.

**[O'RP92]** P. E. O'Neil, K. Ramamithram, and C. Pu. "Towards predictable transaction executions in real-time database systems," Tech. Report 92-35, Univ. of Mass. 1992.

**[PeRS88]** P. Peinl, A. Reuter, and H. Sammer. "High contention in a stock trading database: A case study," *Proc. 1988 ACM SIGMOD*, Chicago, IL, June 1988, pp. 260-268.

**[Reut85]** A. Reuter. "The transaction pipeline processor," *Int'l Workshop on High Performance Transaction Systems*, Pacific Grove, CA, Sept. 1985.

**[Reut86]** A. Reuter. "Load control and load balancing in a shared database management system," *Proc. 2nd ICDE*, Feb. 1986, Los Angeles, CA, pp. 188-187.

**[ThRa90]** A. Thomasian and E. Rahm. "A new distributed optimistic concurrency control method and a comparison of its performance with two-phase locking," *Proc. 1990 ICDCS*, Paris, France, May 1990), pp. 294-301.

**[Thom93]** A. Thomasian. "Two-phase locking and its thrashing behavior," *ACM TODS 18*,4 (Dec. 1993), 579-625.

**[Thom94]** A. Thomasian. "Concurrency control: Methods, performance, and analysis," *IBM Research Report RC 19964*, Dec. 1994.

**[Thom95]** A. Thomasian. "Checkpointing for optimistic concurrency control methods." *IEEE Trans. Knowledge and Data Eng. 7*,2 (April 1995).

**[WeZo86]** H. Wedekind and G. Zoerntlein. "Prefetching in realtime database applications," *Proc. 1986 ACM SIGMOD Conf.* Washington, D.C., May 1986, pp. 215-226.

**[WHMZ94]** G. Weikum, C. Hasse, A. Moenkeberg, and P. Zabback. "The COMFORT automatic tuning project, *Information Systems 19*,5 (1994), 381-432.

# Using Unix Workstations for a Low Latency, High Availability DBMS

## Extended abstract

Øystein Torbjørnsen and Svein-Olaf Hvasshovd
Telenor Research, N-7005 Trondheim, Norway
Email: {Oystein.Torbjornsen,Svein-Olaf.Hvasshovd}@tf.telenor.no

Submitted to HPTS '95, Pacific Grove, California, September 17–20

## 1 Introduction

Databases have over the last years been facing new applications in the telecom industry. One such application is universal personal telephone (UPT). These applications are characterized by requirements which in combination are not met by current database systems [2, 6]. These requirements are: 1) low, predictable response times (15 to 30 $ms$); 2) high throughput (more than 1000 transactions per second); and 3) very high availability (maximum 2 minutes downtime per year). The transaction types in these applications are very light, ie. touching one to four tuples in the database, and with up to 90% read-only transactions.

Multiple projects address these applications, but none so far has aimed for reaching all three requirements. Smallbase [4] and Dalí [7] are two of these systems. Both have excellent response times and throughput for light transactions, but have not focused on the high availability aspect.

ClustRa is a parallel and distributed DBMS engine for telecom applications which is designed to meet the combined three requirements. It is using standard Unix workstations interconnected with a high-performance ATM network. A Unix workstation constitutes a node, which is the smallest unit of failure. A site is a collection of several nodes and a ATM switch, and is the unit of failure for catastrophic events like fire and avalanche. Some of the nodes on each site are dedicated spare nodes.

Database tables are horizontally fragmented based on hashing. Fragments are replicated with one primary and one hot stand-by replica. The replicas are distributed so that one site only stores one replica of a fragment. Currently we are supporting mirrored declustering.

Data is stored on disk with a traditional page-oriented layout. To avoid disk accesses for real-time transactions data used by them are cached in main memory. A B-tree access method with real-time extensions is used for direct tuple access.

Transactions can enter the system on any node which control its execution. Based on the fragmentation key value and the fully replicated fragmentation dictionary, tuple operations are forwarded to the nodes carrying the primary replicas the tuples are stored in. Independent operations inside one transaction are shipped and executed in parallel. Multiple operations targeted to the same node are sent as one message. Tuple updates cause log records to be shipped from the primary replica to the hot stand-by. A prepare-to-commit request is piggybacked on the last independent message to each involved primary node. A primary responds ready when it has executed all tuple operations and shipped the log-records to the hot stand-bys [5].

When the transaction controller ships the updates to the primaries, it also ships requests to the hot stand-bys to participate in the two-phase-commit (2PC) of the transaction. When a hot stand-by has received all the log records for the transaction, it responds ready to the controller. Note that neither the primaries, nor the hot stand-bys wait for log disk dips.

When the transaction controller has received ready from all primary and hot stand-by participants, it ships the commit decision to a hot stand-by transaction controller on an independent site. After receiving an

acknowledge, it responds the results to the transaction requester. Finally, all participants are informed of the outcome. This execution model gives a total number of messages for a TPC-B transaction varying from 16 to 40 depending on tuple clustering. The number of messages sent before responding to the requester varies between 10 and 22. The time critical path from the reception of the request until the response consists of a sequence of 5 messages. These numbers do not include the communication with the requester.

Scheduling based on ordinary two-phase locking is performed to the primary data. Tuple granularity locks are kept on both primary (read and write locks) and hot stand-by data (write locks only). Tuple log records are fully location transparent both with respect to redo and undo operations. Node internal log records are kept within the originating node.

Node failures are detected through a heart-beat protocol. A new availability set is computed based on the virtual-partition algorithm [1]. The hot stand-bys for the unavailable primary replicas take over as primaries. In flight transactions involving the failed node are aborted. The replicated fragmentation dictionary is automatically modified to reflect the current system state. If the failed node is unable to immediately recover, data and log with only one replica left, is automatically rereplicated onto one of the spare nodes. We consider securing log records in primary memory on two different sites to provide sufficient safety when it is combined with an immediate rereplication capability. This reduces the time-window when the system is vulnerable to double failures.

Except where stated the hardware used is Sun Sparcstation 10/40 workstations with 256 MB primary memory and 1 GB disk running SunOS 4.1.3. They are interconnected with a Fore Systems ASX-200 ATM switch using SBA-200 100 Mb/s TAXI S-bus adapter cards.

# 2 UNIX deficiencies

Unix was designed as a time-sharing operating system for interactive use. Although it is not the perfect system for all applications it has gained acceptance for its ubiquitousness. The portability of the operating system itself and applications on top of it has resulted in a large user base. The development environment is excellent with many tools available and an excellent integrated network support.

Real-time systems and transaction processing demand other aspects than those originally addressed by Unix. Over the years this has somewhat improved with real-time extensions, multi-threading support, and efficient access to shared memory and disk resources.

## 2.1 Process scheduling

At each node transaction processing activates three processes: the transaction controller (TCON), the database kernel (KERN), and the update channel (UCHN). This subdivision was motivated by increased protection between independent modules in the system. This increases the probability of early detection of software errors and reduces the chance of corrupting the database content. Transaction concurrency is achieved by using multithreading inside each of these processes. The thread package implements non-preemptive scheduling between threads to avoid use of expensive semaphore mechanisms. Especially those supported by Unix are too expensive for our application.

A transaction is executed in parallel both over the processes internally on a node and over the nodes in the system. The interaction between the processes is through messages. Some of the processes also share memory regions but only one process is allowed to write into it. Our design assumed that Unix did not reschedule processes when sending messages, but rather after spending its relatively long (20 $ms$) time slot or having to wait for a resource. Therefore, a process could send multiple messages before deliberately giving up its control by waiting for a message response without being interrupted by Unix. For instance the TCON could send all its messages uninterrupted to the participating KERNs before waiting for response.

Despite this careful design, Unix caused problems. The three processes schedule very unpredictable and occasionally compete with system processes. In a study of the scheduling of the processes [3], we found

that we could get a rescheduling at any message transfer operation, also when sending messages. In effect, this caused a varying number of context switches within the execution of a transaction which could not be predicted. Also the scheduling causes messages in the time critical path to be delayed. We have measured a variation in the transaction response times of the exact same transaction from 11.7 $ms$ to 16.6 $ms$ without any other load on a two node system. Given the 15 $ms$ response time requirement this was disappointing.

The fast transaction execution resulted in 7 context switches on the node where the transaction controller resides before responding to the user, while the slow had 10. Our optimistic scheduling prediction was only 5 context switches. We have measured the context switching time on the workstations used to 57 $\mu s$. In total this results in negligible CPU time for context switching itself. Most of the added time can be explained by a suboptimal scheduling of the parallel transaction execution plan.

Given this new insight we did some modifications to our design. The change that gave the most significant effect with respect to response time was merging the three processes into a single one. This was fairly simple since we already had a multithreaded design. The threads that originally were distributed over three processes are now running in one and are still communicating by messages. Obviously this compromises the intended protection between the modules. We find this acceptable since the main defense against software and hardware errors is the logical and physical isolation between nodes. If the extra protection is required, the system can optionally be recompiled and built into multiple processes. The overall improvement of this change alone was a speedup in the range of 20-25% depending on target platform.

The most recent version of the DBMS executes 99.5% of the transactions timely. On the other hand, a few transactions end up with with a response time more than 10 times longer than the average at 9.77 $ms$. These are caused by Unix activating system processes periodically. These run system management tasks, like paging, swapping, and file-system cache flushing. For our application this is not necessary since we require that there should be enough primary memory to avoid this. These redundant processes run for periods that are longer than our required response time, causing transaction processing to be blocked. Fortunately this occurs so infrequently that it does not threaten our soft real-time requirement.

Modern Unices have support for real-time processes. This allows processes to have a fixed priority which is higher than ordinary Unix processes. Testing this on HP-UX 9.0.5 showed that we got much less variance than using traditional Unix scheduling. Unfortunately, the average response time increased slightly. Since calls to the Unix kernel are non-preemptive there still existed a few long lasting transactions. We solved this by killing the system process responsible for the trouble, without knowing whether this will be feasible in a production environment.

## 2.2 Message passing

Interaction between processes, both inside a node and between nodes is done using a lightweight RPC (remote procedure call) mechanism supporting reliable delivery. This simplifies the implementation of the DBMS layer code without adding too much processing overhead. The RPC paradigm corresponds well to what is needed by distributed transaction processing, included the 2PC protocol. The RPC mechanisms makes few assumptions about the underlaying message passing mechanism provided by Unix.

The need for fast communication between nodes are given by the soft real-time requirement. High communication bandwidth is required to support high volume transaction processing and fast rereplication of fragments after a node crash. Normal 10 Mb/s Ethernet would soon be saturated for high transaction volumes and would only be able to handle small databases in case of failures. Close to saturated Ethernets can not guarantee response times due to high collision rates. Of the alternative high bandwidth solutions we have chosen ATM, because it is an established product, scalable, and has a low hardware latency. The literature reports measured bandwidths larger than 10 MB/s and RPC latencies lower than 100 $\mu s$ [8].

UDP/IP is the basic communication protocol used in the implementation. It is a datagram protocol with unreliable delivery which perfectly matches our RPC paradigm. It gives high portability over all platforms and can be used over both ATM and Ethernet.

It was evaluated against three other alternatives: TCP/IP, ATM API, and Unix SysV messages. TCP/IP's

byte stream paradigm represents a mismatch with message passing applications. Normally it delays sending non-full messages in case more data are sent soon after, causing unpredictable message latency. On the receiving site messages have to be refragmented causing extra overhead. Since TCP/IP provides reliable delivery, it is sending acknowledgement messages adding to the total number of messages generated. Due to the semantics of RPC, the acknowledgements must anyway be implemented on a higher level. A proprietary API was provided with the ATM product. This alternative was ruled out because it is not portable to other platforms, it has poor functionality, and it gives little or no performance improvements over UDP/IP. Unix SysV messages have excellent performance, but can only be used inside a node.

Our system design was based on the assumption that messages were relatively cheap (less than 2000 instructions sending and receiving a message) and had a low latency (less than 100 $\mu s$). These assumptions were founded on an overall knowledge of the ATM architecture and published reports telling that commercial ATM hardware was able to do an RPC between the Unix kernels on two different workstations in 55 $\mu s$ [8].

When we started to experiment with the hardware the process to process UDP/IP RPC latency was measured to 1.7 $ms$. The same figure for Ethernet was 1.5 $ms$! By instrumenting the driver software for ATM we have found that the Unix protocol stacks are eating close to all of this time. This disappointing observation lead us to a careful reconsideration of the architecture since rewriting of communication drivers and protocol stacks was considered outside the scope of the project. The architecture was from the beginning optimized with respect to message passing. Despite this fact, we made an effort to further eliminate the total number of messages for a transaction and moving messages out of the time critical transaction execution path.

A consequence of the Unix scheduling in the original design log records were frequently shipped as individual messages from the primary to the hot stand-by. Immediately when a log record was produced, this was reported to the UCHN. This frequently resulted in its activation and sending of one message. To avoid this, the UCHN was not informed before prepare-to-commit causing log records generated by one transaction to be clustered into one message. This modification both reduced the number of context switches and the number of log messages.

In the original design we informed the hot stand-by TCON when a transaction was initiated. This was done to guarantee that two TCONs knew about the existence of the transaction. This lead to a RPC round-trip delay in the time critical transaction execution path. The round-trip was eliminated and is now only involved at commit time. This could be done by letting all participants know which node the hot stand-by TCON eventually would be located on. If the the primary TCON fails, all participants contact the hot stand-by TCON which completes the transaction. If no knowledge about the transaction exists at the TCON when it is contacted by a participant, it can assume an abort. If the decision was commit, the TCON would know about the transaction, because the commit decision is sent to it before being sent to the participants. This change resulted in the number of messages before replying to the requester to be reduced from 12 to 10.

The merging of the three processes into a single process also had an effect on the number of messages sent using the operating system. Messages sent from one thread to another inside the same process are short-circuited saving CPU cycles spent by Unix. This removed from 2 to 10 messages before replying to the requester, and removed from 2 to 12 messages in the total execution plan for a TPC-B transaction. The number of messages depends on the clustering of data relative to the TCONs.

# 3 Performance results

Table 1 shows the performance evolution of the system for TPC-B like transactions on a two node system.

The first row in the table shows the performance of the original design. These results clearly did not reach the latency requirement with 95% completing within 15 $ms$.

The second row shows the system when the message removal indicated in section 2.2 have been incorporated. Compared to the original design, the average response time is reduced with 21% and the throughput increased with 40%. On a system with low load, the response time requirement is clearly satisfied.

The third row shows the results for the single process version of the system in row two. The improvement

| | Minimum response time | Average response time | Percentage of transactions $< 15\ ms$ | Transactions per second |
|---|---|---|---|---|
| Original multi process version | 10.9 $ms$ | 14.4 $ms$ | 81.0% | 50 |
| New multi process version | 10.0 $ms$ | 11.4 $ms$ | 99.0% | 70 |
| Single process version | 8.0 $ms$ | 8.8 $ms$ | 99.8% | 86 |

Table 1: Response time and transaction throughput evolution.

by going from three to one process is 23% for both response time and throughput. The overall effect of the optimizations presented in this paper is a 39% response time reduction and a 72% throughput increase.

# 4 Conclusions and further work

So far, our focus has been on response time optimizations. In a number of telecom applications single tuple transactions are dominating. We are now working on response time optimizations for single tuple read and write transactions.

Throughput can be further optimized without compromising response time. One direction we are considering is grouping multiple operations into one messages when they occur after completion of the time critical phase of the transactions.

The project has reached its goals with respect to response time using commodity workstations with Unix. Onwards our focus will primarily be on achieving the required availability.

# References

[1] A. E. Abbadi, D. Skeen, and F. Cristian. An Efficient, Fault-Tolerant Protocol for Replicated Data Management. In M. Stonebraker, editor, *Readings in Database Systems*, pages 259–273. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.

[2] I. Ahn. Database issues in telecommunications network management. In *Proceedings of ACM/SIGMOD (Management of Data)*, May 1994.

[3] C. A. Galindo-Legaria and S.-O. Hvasshovd. A performance analysis of the ClustRa DBMS. Technical Report TF R 46/94, Norwegian Telecom Research, Kjeller, Norway, 1994.

[4] M. Heytens, S. Listgarten, M. Neimat, and K. Wilkinson. Smallbase: A main-memory dbms for high-performance applications (release 3.1), Mar. 1994. Hewlett-Packard Laboratories, Palo Alto, CA, USA.

[5] S. Hvasshovd, T. Sæter, Ø. Torbjørnsen, P. Moe, and O. Risnes. A Continously Available and Highly Scalable Transaction Server: Design Experience from the HypRa Project. In *Proceedings of the 4th International Workshop on High Performance Transaction Systems*, September 1991.

[6] S.-O. Hvasshovd, Ø. Torbjørnsen, S. E. Bratsberg, and P. Holager. The ClustRa telecom database: High availability, high throughput, and real-time response, 1995. Submitted to VLDB-95.

[7] H. V. Jagadish, D. Lieuwen, R. Rastogi, and A. Silberschatz. Dalí: A high performance main memory storage manager. In *Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile (VLDB '94)*, pages 48–59, Sept. 1994.

[8] C. A. Thekkath, H. M. Levy, and E. D. Lazowska. Efficient support for multicomputing on ATM networks. Technical Report 93-04-03, Department of Computer Science and Engineering FR.35, University of Washington, 1993.

# Flexible Business Processing with SAP Business Workflow 3.0

Extended Abstract

Helmut Wächter, Architect and Developer of the SAP Workflow Manager
SAP AG, Business Process Technologies, Postbox.1461, D - 69185 Walldorf
email: helmut.waechter@sap-ag.de

## 1. Motivation

Most companies operate in an environment with a very dynamic nature: rapidly changing customer demands, market situation, legal or political regulations, technology, etc. result in frequently changing organizational structures and business procedures. This means that a re-engineered and "optimized" business process [HaCha93] is no longer optimal after a while and has to be improved again and again. Thus the ability to change a process easily and quickly is at least as important as its optimization, e.g. with respect to time or money. Critical users argue (and partly they are right) that available workflow management systems (WFMS for short) will "cement" a business process once it is implemented. When using a WFMS one has to be careful not to produce "legacy workflows" which hinder or even prevent ordinary alterations, such as replacing hardware and software components which implement a business process; creating, dividing or merging organizational units, jobs and workplaces, or adjusting assigned tasks and authorities; employees join the company, are temporarily absent, or move to another department and so on. In order to support these various changes efficiently, a WFMS has to provide a high degree of flexibility in modelling, maintaining as well as executing business processes.

Release 3.0 of the SAP R/3 System contains an integrated set of workflow tools. This paper presents these tools with a special focus on the concepts and mechanisms for implementing flexible business processes. After the general architecture (section 2) we describe the basic design and modelling principles and give some examples to illustrate the resulting flexibility (sections 3-5).

## 2. SAP Business Workflow Architecture

The SAP R/3 System is a well known standard software package for business applications [BuGa94]. Based on a TP monitor it implements a three-level client/server architecture [GrRe92] and runs on various hardware and software platforms. Its rich set of integrated functions covers nearly all business tasks which need to be carried out in medium to large-sized and even multinational companies. It also comes with an enterprise data model and a process repository, which describe the architecture and functionality of its business objects and procedures. Release 3.0 contains a workflow workbench, which can be used either to customize the pre-defined workflow process templates or to build individual business processes based on the various building blocks delivered with the workflow repository (business objects, events, tasks, roles, etc.).

Besides the existing interfaces to the SAP System, the workflow tools offer an additional interface at the process level: workflows within the R/3 System can call external workflows and vice versa. Since SAP is an active member of the Workflow Management Coalition [WfMC94], it plans to support the standardized Workflow API (WAPI) when it is published.

Fundamental for the flexibility which can be achieved with SAP Business Workflow is the three-level architecture sketched in figure 1. It distinguishes various aspects of a business process and maps each of them to a separate level. These perceptional views are the *organizational level*, the *process level* and the *object level*:

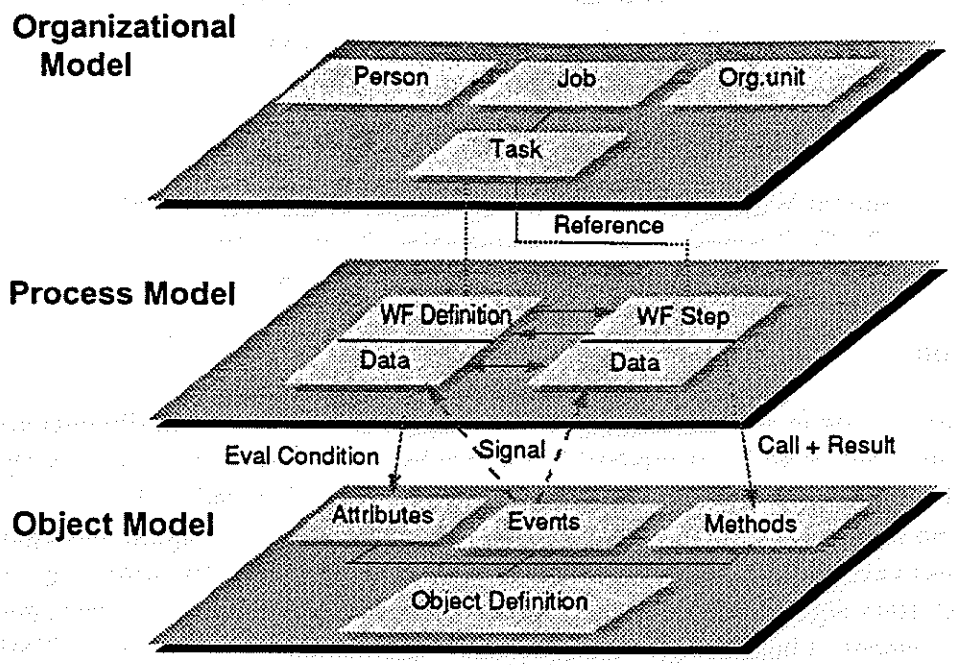



*Figure 1: Architecture of SAP Business Workflow 3.0*

- **Organizational model**
  The organizational model describes all organizational elements participating in a business process: jobs and positions; their grouping into organizational units like workgroups, departments etc.; job descriptions with the tasks and responsibilities of a staff member; reporting and management relationships etc.
- **Process Model**
  A business process consists of several sequential or parallel steps. Defining the flow of data and control between the steps through a graphics editor [SAP-WF] is based on the ConTract Model [WäRe92]. The steps of a workflow are not defined by specifying a *program* (transaction, report etc.), but by referencing a *task* of the organizational model, which thereby takes the leading role in business process modelling.
- **Object Model**
  Tasks in SAP Business Workflow are implemented by calling methods on objects. The object level provides a unified interface to the existing application data and programs inside and outside the SAP System, which are involved in a business process (like database tables, PC documents; transaction programs, 4GL function modules, etc.). Remote data or object servers can be accessed by OLE2 [OLE94].

The following sections sketch the three levels and their interrelations in more detail.

## 3. Organizational Model

Integrated in the SAP Business Workflow Tools is a powerful component for organizational modelling and management [SchDr95]. As mentioned above, **tasks** play the central role in linking the organizational and operational structure: On the one hand, they are part of the operational structure and describe a procedure which manipulates one or more business objects, for example creating an order, approving an invoice and so on. On the other hand, tasks are linked to agents (usually to jobs, not to persons directly) which are part of the organizational structure. This link is the foundation for addressing workflow steps to agents ("routing"): When a workflow step is to be carried out, it is clear from the organizational model which jobs contain the corresponding task and which persons fill these jobs. Thus, when logging on to their personal work list, the users find only those workitems which are part of their job description, or which are addressed to the organizational unit they belong to. Addressing tasks to jobs or organizational units, and not to persons directly, has the advantage that acting as a substitute, moving job or department, and other organizational changes

become effective immediately in the workflow system. Especially in large organizations it is not acceptable to re-route active workitems manually in order to see them in the work list of the new user (or substitute). In other words: **the SAP Workflow System reflects all organizational changes automatically**.

Now, determining agents of a task by addressing *all* jobs or persons which have the task in their job description is sometimes a too static way of work enactment. Thus the SAP workflow tools offer several additional mechanisms in order to select agents dynamically depending on the actual workflow data:

a) A customer complaint, for example, should not be routed to *all* customer service staff members, which could do the task according to their job profile, but only to those who are actually responsible for the specific customer (or product, sales region etc.). Therefore, the agent of a task can be determined by a role resolution program which has access to the customer's name and the other workflow data. Another possibility is to determine the agent of a step when starting the workflow, or as the result of a preceding step.

b) It is also possible to select an agent indirectly by referencing an agent of a previous step, the workflow initiator, the manager of the agent of step 5 and other predefined roles.

All these mechanisms could also be used to **exclude agents**, who should *not* work on a task. An invoice, for example, has to be approved by two different persons when it is above a certain amount. Or consider a workflow for business trip approval by the respective manager. Here it is possible to specify that a staff member is not allowed to approve his/her own business trips, even when he/she temporarily replaces his/her manager.

**Triggering** and **terminating events** provide further flexibility in SAP Business Workflow. In the task repository one can define events which trigger or terminate a task automatically. Events are signalled by the business objects indicating a workflow relevant state transition, see below. For example, a complaint received by fax or EDI could start a service workflow to process it. When the customer with a complaint reports that the problem has been solved, the workflow is completed automatically. A company using the SAP Workflow System can specify which task (if any) is to be started or completed when an event occurs in the system. "Invoice.created", for example, can start the (modified) workflow template SAP delivers with R/3, a completely different customer-defined workflow or no process at all, if creation and verification of an invoice is carried out by the same user within one step. Triggering or completing a task by an event can be dependent on a condition (cf. event-condition-action-rules) and can also be disabled temporarily [Gers95].

## 4. Process Model

The graphical presentation of a process uses a special sort of activity/state chart called "event-driven process chains" [KeMe94], see figure 2: Activities (rectangles) and resulting states are linked by connectors *xor* and *and*. This mechanism was extended by the usual flow elements of block-structured programming languages (e.g. Modula) and further mechanisms necessary to control long-running activities [WäRe92]: Besides sequential and parallel blocks (*fork, join*), we have *if, case, while* and *until* loops. To support parallel business flows in a database environment, it is also possible to process each line of a table (e.g. items of an invoice) in parallel. At a join node only $n$ from $m$ parallel incoming paths might be declared necessary to continue the flow. Since a workflow is also modelled as a task in the task repository, process definitions can be nested. Automated process control is supported by suspending or cancelling individual steps or the whole workflow. An important link to the object level are steps, which can wait for asynchronous state transition signals (events) from object instances which are of interest to the workflow, see below. A workflow or step can also be completed by an event $e$ even without processing the referenced task, if the task declares event $e$ to be one of its terminating events. For example, an event *employment stop* triggered by the CEO could stop all ongoing application workflows. If a step has more than one terminating event, the workflow can branch on that event in order to model a specific response to the event which actually terminated the step.

A workflow and each of its steps have a private data pool containing (pointers to) the involved objects and other variables. The data flow definition, a part of the process definition, specifies how flow variables are mapped to the input and output variables of a step. This creates two separate name spaces for workflow and

step data. As a consequence, both can be implemented independently and, more importantly, steps can be easily reused in other workflows. To tolerate crashes and other system failures, workflow and step data are managed transparently for the flow and step programmer in a database.

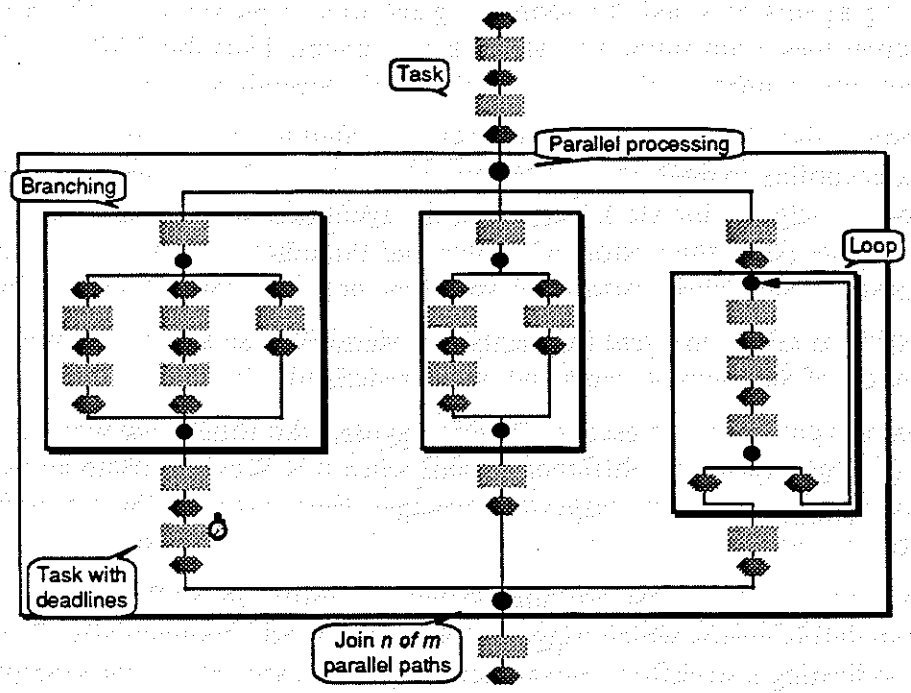



*Figure 2: Sample Workflow Definition.*

Although workflows are mainly intended for well-structured processes, some limited deviations and modifications of the process script are inevitable to handle unusual, and therefore not predefined exceptions. To allow a quick response to ad hoc changes during a workflow execution, the SAP workflow tools support

- skipping optional steps
- dynamic determination of the task of a step and its deadlines (similar to dynamic agent determination)
- inserting additional steps and objects (notes, PC documents etc.) into a workflow.

Besides that, users with the required authorization can change the attributes and processing states of steps or workflows. For example, a manager or person responsible for a process could update the deadlines of a step, or change the list of responsible agents etc.

## 5. Object Model

A workflow step implementing a task calls one or more methods of the involved business objects. The delivered business objects comprise orders, deliveries, invoices, production plans, materials, plants etc. Workflows, steps and objects are also modelled as (meta) object classes in the repository.

The object level does not replace the existing SAP runtime system. Rather it provides a suitable basis for workflow computing on top of existing applications. It is a type of wrapper implementing a unified calling and data transfer interface to the various non-object-oriented transaction programs, 4GL procedures and so on. By providing an RPC-like API, OLE2 [OLE94] and (planned) CORBA [OMG91] it is also possible to implement business objects outside the SAP System. A delegation mechanism allows the customer to overwrite classes delivered by SAP with user-defined modifications or extensions. Besides that the state-of-the-art object repository supports selected multiple inheritance, aggregating and linking objects. The graphical object browser integrates the SAP Development Workbench and Data Dictionary in order to define an object class by simply clicking the mouse to reference existing tables, transactions, 4GL programs etc.

Object identifiers, attributes, methods and events are the four important interfaces to the process level:

1. Via the object id one can find all process instances in the system which work on a given object.
2. An object's attributes can be used in a workflow branching condition by the usual dot notation (e.g. "invoice.amount > $ 10 000") without coding any SQL or 4GL statements to query the required values.
3. Besides an arbitrary number of input/output parameters, a method could produce one distinguished result value and several exceptions (indicating application errors) which can be used in the workflow definition for an implicit branch after a step. The same is possible for the completion and deadline events of a task.
4. Events indicate workflow-relevant state transitions of an object, as mentioned above (created, changed, released, deleted, and so on). Like a method, an event can have parameters which a subscribing task can pick up to respond more specifically to that event. Note, that the reaction to an event is not part of the application program implementing an object, but is defined in the task repository (triggering and terminating events) or in the workflow definition (steps waiting for an event of an object instance). This separation of the "publisher" of an event and its "subscribers" responding to it allows you to define and change them independently. The implementation reflects this separation: Creating an event is implemented by using a transactional RPC-like function call within the publisher's sphere of control. The subscribers, however, run asynchronously in their own logical unit of work. This mechanism is also used in SAP Business Workflow to integrate dialog transactions, which produce their result in an asynchronous commit procedure outside the sphere of control of a method call.

## 6. Summary

The process and object orientation and principally the integrated organizational model of the SAP Business Workflow tools allow a high degree of flexibility in modelling and executing business processes. Thereby they support individual and evolutionary business process engineering in an enterprise.

## References

[BuGa94] Buck-Emden, R.; Galimow, J.: Die Client-Server-Technologie des Systems R/3. Bonn 1993.

[Gers95] Gerstner, R.: A Workflow-Oriented Architecture of Application Systems based on Events (in German). Proc. Modelling Office Information Systems, Bamberg, Germany, October 1995.

[GrRe92] Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann,1992

[HaCh93] Hamer, M.; Champy, J.: Reengineering the Corporation. New York 1993.

[KeMe94] Keller, G.; Meinhardt, S.: SAP R/3 Analyzer: Optimizing Business Processes using the R/3 Reference Process Model. SAP AG, Walldorf 1994.

[OLE94] Microsoft OLE Custom Controls. Preliminary Specification, Jan. 1994.

[OMG91] Object Management Group: The Common Object Request Broker. Document No. 91.12.1, 1991

[SAP-WF] SAP Business Workflow. R/3 System: Functionality en Detail. SAP AG, Walldorf, 1995.

[SchDr95] Schlögel, C., Drittler, B.: Workflow-integrated Organizational Management. SAP-Info *Business Workflow,* SAP AG, Walldorf, 1995

[WäRe92] Wächter, H., Reuter, A.: The ConTract Model. In: A.K. Elmargarmid (Ed.): Transaction Models for Advanced Database Applications. Morgan Kaufmann Publishers, San Mateo, CA, 1992

[WfMC94] Workflow Management Coalition: The Workflow Reference Model. Doc. TC00-1003, 1994

# Workflow Monitoring: Queries On Logs or Temporal Databases?

## Position paper, submitted to HPTS'95

**Gerhard Weikum** *

Department of Compter Science
University of the Saarland
P.O. Box 151150, D-66041 Saarbruecken, Germany
E-mail: weikum@cs.uni-sb.de

### 1 The Need for Workflow Monitoring and Workflow History Evaluation

Workflow management is a challenging subject that is being tackled, from different angles, by computer industry, computer science, and business science [Me94, GHS95, HC93]. A workflow consists of a set of processing steps together with some specification of the control and data flow between the steps. In addition, some deviation from the specified control flow should be possible to allow for intellectual exception handling and decision making. The ultimate goal is to automate the computerized processing of workflows in large-scale distributed environments, without harnessing the flexibility of human intervention. Application examples that would enormously benefit from even partial solutions include loan processing in banks, insurance claim processing, health care administration, submission and processing of tax declarations, and so on, not to mention the often-cited, canonical example of travel planning.

The focus of computer science in this context is on specification methods for workflows, and on mechanisms for reliable execution in distributed systems with use of and extensions to transactional technology [Be93, Da93, De94, Hsu93, LA94, Mo95, RS94, WR92]. In contrast, business science is more concerned with the re-engineering of inefficient business processes at a rather informal level [Ma93, Sch94]. One aspect of workflow management that has not received much attention by neither community is the monitoring of workflow executions. I claim that this aspect bears a huge potential for improving the transparency, efficiency, and accuracy of workflows and is crucial for the success of business process reengineering. In fact, I believe that workflow monitoring is an important link between the computer-science and the business-science perspective of workflow management. In the following, I discuss the rationale for this position statement and the research issues that are entailed by it.

In workflow applications, steps may be processed in a fully automatic manner (e.g., checking the completeness of a tax declaration), solely intellecutally (e.g., deciding upon the medical treatment of a hospital patient), or by a combination of both (e.g., approving a loan request based on information from a customer database). In any case, a workflow management system should be aware of all these steps and should be informed about the corresponding input and

* This position paper is based on a joint project, between the University of the Saarland, the Union Bank of Switzerland, and ETH Zurich, on "Middleware for Enterprise-wide Workflow Management" (MENTOR).

output data. This information is the basis for making the dynamics of distributed business processes comprehensible to the involved agents themselves and to a process engineering staff. For example, a bank agent who is in charge of a certain step in a complex workflow may want to check the current status of other steps that are being processed in parallel, in order to assess the criticality of the workflow with respect to deadlines, risk versus potential profit, and so on. A process engineer, on the other hand, would be keen to learn about the (average and maximum) duration of and total work time involved in various steps, in order to identify opportunities for improvement. Then, workflows with a high turn-around time could, for example, be reorganized by streamlining the control flow and the organizational responsibilities (e.g., combining steps or running more steps in parallel).

Conceptually, these information demands could be satisfied from a relational database as illustrated in Figure 1 in an oversimplified manner. Obviously, such a database constitutes an *application-level log* of all relevant events that occur during workflow executions. The log is further augmented with additional data such as step status, work time spent on a step, references to documents that describe input and output of a step, addresses of databases or imaging archives where these documents reside, and so on. I claim that the implementation of this application-level log is an important and largely unexplored issue in large-scale workflow management. In the following I point out two conceivable starting points for tackling this issue: the first approach aims to enrich an audit trail with additional functionality, and the second approach takes the viewpoint that a workflow log is a special kind of temporal database.

*Workflows*

| *WFId* | *WFType* | *Initiating Agent* | *Starting Date&Time* | *Completion Date&Time* | ... |
|---|---|---|---|---|---|
| 1001 | LoanRequest | B.J. Smith | Sept-13-1995 10:05 AM | Sept-20-1995 2:20 PM | |
| 1002 | PortfolioMgt | B.J. Smith | Sept-15-1995 11:30 AM | Sept-17-1995 8:30 AM | |
| 1003 | LoanRequest | A.B. Jones | Sept-20-1995 9:15 AM | still in progress | |
| ... | | | | | |

*Steps*

| *WFId* | *StepId* | *StepType* | *Agent InCharge* | *Starting Date&Time* | *Completion Date&Time* | *Work Time* | ... |
|---|---|---|---|---|---|---|---|
| 1001 | 1 | InitiateLoanRequest | B.J. Smith | Sept-13-1995 10:05 AM | Sept-13-1995 10:50 AM | 0:45 | |
| 1001 | 2 | CheckClientRating | P. Nutts | Sept-14-1995 9:05 AM | Sept-18-1995 10:30 AM | 1:30 | |
| 1001 | 3 | AssessRisk | RiskMgtSys | Sept-13-1995 11:22 AM | Sept-13-1995 11:24 AM | 0:02 | |
| 1001 | 4 | MakeDecision | T.W. Miller | Sept-20-1995 1:50 PM | Sept-20-1995 2:20 PM | 0:30 | |
| 1002 | 1 | EnterInvestorData | A.B. Jones | Sept-15-1995 11:30 AM | Sept-15-1995 12:25 PM | 0:55 | |
| ... | | | | | | | |

**Figure 1: Oversimplified illustration of a workflow monitoring database**

## 2 The Audit Trail Approach

A workflow audit trail is basically an append-only list of application-provided log entries. In addition, however, it has to satisfy some challenging requirements that go way beyond the capabilities of a conventional log file:

- Log entries (i.e., workflow events) must be retrievable via declarative queries. Example queries could be:
  - Which steps of which loan request workflows took place during the week from September 18 through September 24?
  - Is a particular workflow still on track and likely to meet a specified deadline?
- Queries against a workflow audit trail may involve complex temporal aggregations. Examples could be:
  - Which step type has the longest average duration in all completed loan request workflows that had an overall turn-around time of 2 weeks or longer?
  - What was the highest number of steps that B.J. Smith was involved in within three adjacent work days? (This involves aggregation over a moving time window.)
- Queries must be supported concurrently to the appending of new log entries.
- Log entries may have to be kept for years. Thus, the standard techniques for compacting and reusing log disk space are not applicable. Rather log entries have to be spooled onto tertiary storage and possibly translated into a different format and storage organization.

## 3 The Temporal Database Approach

An alternative to the audit trail approach is to view the workflow history as a special kind of temporal database. Thus, significant leverage could be expected from the advances in temporal database systems [Ta93, Sn94], most notably, from the functionality provided by temporal query languages such as TSQL and the efficiency of temporal access structures such as the TSB-tree [ST94]. However, substantial extensions to the state of the art are necessary in this case as well:

- The temporal database that would hold the workflow history is inherently distributed. The data about workflow events arises at different sites where workflow steps are executed. Enterprise-wide workflow management (e.g., in a bank) encompasses in the order of ten thousand computers. Thus, it is important to partition the workflow history and manage it as a distributed database. This is particularly difficult as workflows themselves migrate between sites.
- The workflow history must be highly available. This may require sophisticated forms of replication or other techniques for masking failures and unavailability of sites. Again, this is particularly challenging because of the high dynamics and migratory nature of workflows.

- It may be practically infeasible to keep the entirety of workflow history data in an enterprise–wide homogeneous repository system. The reason lies in the autonomy of different branches and departments within large enterprises. Thus, different partitions of the workflow history may be based on different types of databases or other archival systems.

## 4 Conclusions

I believe that resolving the open questions discussed above poses exciting challenges to researchers and system engineers, regardless of which of the two approaches is pursued. I advocate the direction of building a distributed and highly available temporal database. It seems to me that the envisioned advanced workflow monitoring facilities can be adequately supported only by a full–fledged database system with support for temporal data; in other words: augmenting a workflow audit trail by an appropriate query engine would eventually lead to some kind of temporal database system anyway. I would expect, however, that architects of commercial systems may heavily disagree with this statement, especially given the fact that the implementation side of temporal database systems is developed rather poorly. On the other hand, an important application class like workflow monitoring may give a push to the currently undercovered research on efficient implementation of temporal database systems. As for the additional requirements of distributed queries and high availability, there should be significant leverage from the experience on distributed data management in general.

In any case, such advanced workflow monitoring facilities are completely lacking in the current generation of commercial workflow management systems, and are, perhaps, not yet the most crucial feature of the next–generation systems. In the long term, however, workflow management can realize its full potential and meet the high expectations only if enterprise–wide distributed workflows can be effectively monitored and workflow history data can be evaluated to provide the necessary feedback for the re–engineering of business processes.

## References

[Be93] P.A. Bernstein, Middleware: An Architecture for Distributed System Services, Technical Report, Digital Corporation, Cambridge Research Laboratory, 1993

[Da93] U. Dayal, H. Garcia-Molina, M. Hsu, B. Kao, M.-C. Shan, Third Generation TP Monitors: A Database Challenge, ACM SIGMOD Conference, 1993

[De94] P.J. Denning, The Fifteenth Level, Keynote Address, ACM SIGMETRICS Conference, 1994

[GHS95] D. Georgakopoulos, M. Hornick, A. Sheth, An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases Vol.3 No.2, pp. 119-153, 1995

[HC93] M. Hammer, J. Champy, Reengineering the Corporation, New York, 1993

[Hsu93] M. Hsu (Editor), IEEE Data Engineering Bulletin Vol.16 No.2, June 1993, Special Issue on Workflow and Extended Transaction Systems

[LA94] F. Leymann, W. Altenhuber, Managing Business Processes as an Information Resource, IBM Systems Journal Vol.33 No.2, 1994

[Ma93] T.W. Malone, K. Crowston, J. Lee, B. Pentland, Tools for Inventing Organizations: Toward a Handbook of Organizational Processes, Technical Report, MIT Center for Coordination Science, 1993

[Me94] W.P. Melling, Enterprise Information Architectures - They're Finally Changing, Invited Industrial Plenary Talk, ACM SIGMOD Conference, 1994

[Mo95] C. Mohan, G. Alonso, R. Günthör, M. Kamath, Exotica: A Research Perspective on Workflow Management Systems, 1995

[RS94] M. Rusinkiewicz, A. Sheth, Specification and Execution of Transactional Workflows, in: W. Kim (Editor), Modern Database Systems: The Object Model, Interoperability, and Beyond, ACM Press, 1994

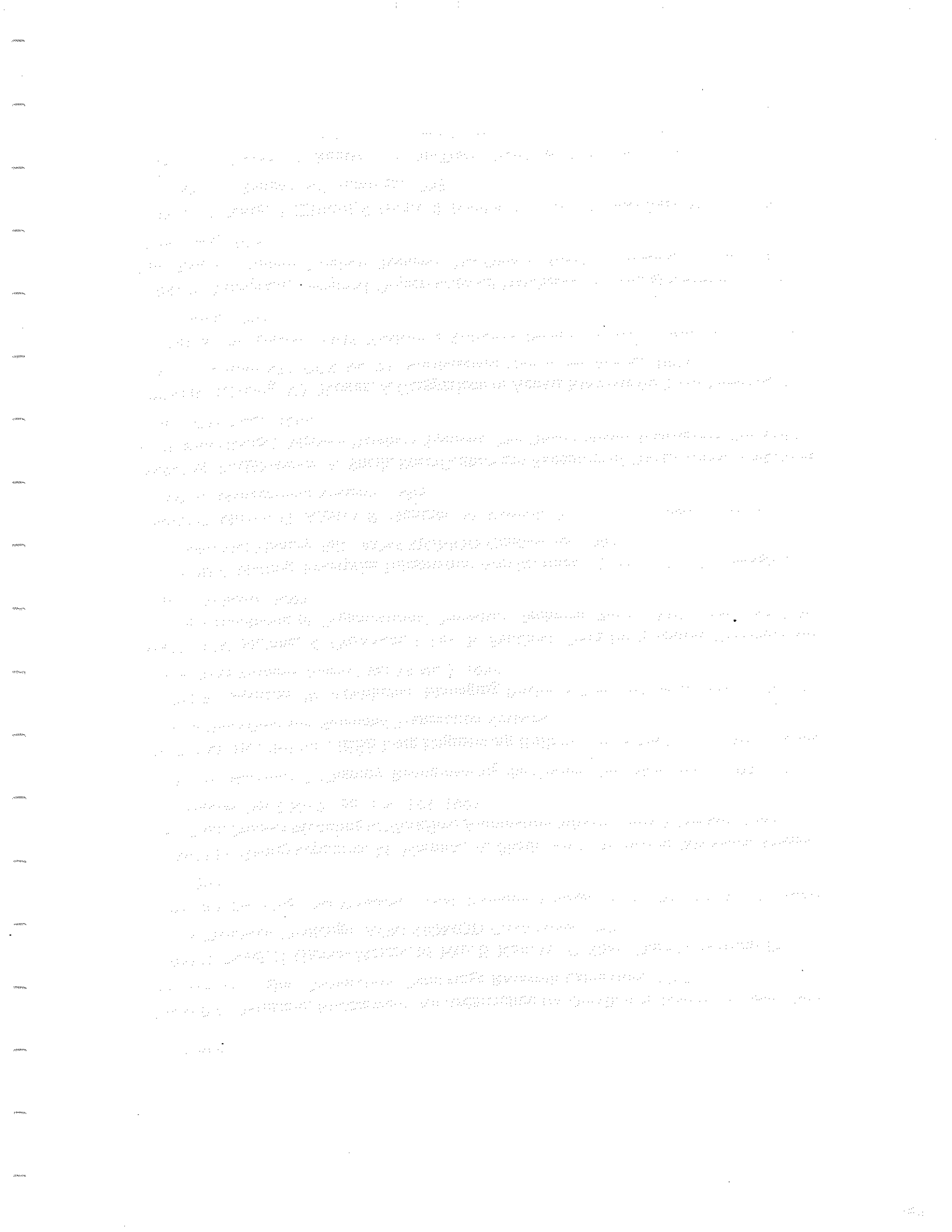
[ST94] B. Salzberg, V.J. Tsotras, A Comparison of Access Methods for Time Evolving Data, Technical Report NU-CCS-94-21, Northeastern University, Boston, 1994

[Sch94] A.-W. Scheer, ARIS Toolset: a Software Product is Born, Information Systems Vol.19 No.8, 1994

[Sn94] R. Snodgrass, Temporal Object-oriented Databases: A Critical Comparison, in: W. Kim (Editor), Modern Database Systems: The Object Model, Interoperability, and Beyond, ACM Press, 1994

[Ta93] A.U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R. Snodgrass (Editors), Temporal Databases, Benjamin/Cummings, 1993

[WR92] H. Wächter, A. Reuter, The ConTract Model, in: A.K. Elmagarmid (Editor), Database and Transaction Models for Advanced Applications, Morgan Kaufmann, 1992



# 6th High Performance Transaction Processing Workshop
# 17-20 September 1995
# Asilomar, CA.

## ——SUNDAY (Sept. 17, 1995)

| | |
|---|---|
| 1:00- 5:00PM | Registration |
| 6:00 PM | Dinner |
| 7:00-10:00PM | Reception |

## ——MONDAY (Sept. 18, 1995)

| | |
|---|---|
| 8:30 AM | Introductions: Don Haderle |
| 8:45 | Peter Weinberger - AT&T challenges in growth. |
| 9:45 | Break |
| 10:15-NOON | (Andreas Reuter, session chair) |
| | Frank Leymann (IBM):<br>FlowMark technology: Current state and future trends |
| | Jack Bissell (ATT/GIS):<br>Electronic Commerce at Walmart |
| 12:00 | LUNCH |
| 1:30-3:00PM | (Dieter Gawlick, session chair) |
| | Helmut Waechter (SAP):<br>Flexible Business Processing with SAP Business Workflow 3.0 |
| | Paul Oeuvray (Prologic):<br>Temporal Data Manager |
| | Friedemann Schwenkreis (University of Stuttgart):<br>APRICOTS - A Workflow Programming Environment |

## ——MONDAY (Sept. 18, 1995) Continued:

| | | |
|---|---|---|
| 3:00 | | Break |
| 3:30 | | Internet (Susan Malaika, session chair) |
| | | Michael Higgins (Netscape)<br>Netscape Secure Courier |
| | | Hamilton, Brown, Malaika<br>Discussion |
| 5:00 | | Mark Carges<br>Next Event for Client/Server |
| 5:30 | | <end of day> |
| 6:00 | | Dinner |
| 7:00 | | Night Session (D.Vaskevitch, session chair)<br>David to explain our participatory preparation earlier in the day |

## ——TUESDAY (Sept. 19, 1995)

| | | |
|---|---|---|
| 8:30 | AM | Messaging and Queuing (S. Malaika, session chair) |
| | | Jim Gray<br>Queues are DataBases |
| | | Jeff Eppinger<br>title missing |
| | | Roger Bamford<br>Oracle in Motion - support for asynch/disconnected transactions |
| 10:00 | | Break |

## ——TUESDAY (Sept. 19, 1995) Continued:

| | | |
|---|---|---|
| 10:30 | | Object Transaction Services (G. Copeland, session chair)<br>George Copeland<br>Introduction: Thesis is objects improve application productivity and cost. Large scale savings is possible through binary reuse.<br><br>Each of the following will describe their approach to how binary components can be used in transactions.<br><br>Jim Lyons - Tandem<br>Geoff Hambrick - IBM<br>Bob Atkinson - Microsoft<br>Mohsen Al-Ghosein - Microsoft<br>Charly Kleissner - Next |
| 12:00 | | Lunch |
| 1:30 | PM | Object Transaction Services, continued (G.Copeland, session chair)<br>Panel debate on alternative approaches with folk from previous session. |
| 3:00 | | Break |
| 3:30 | | Interesting Stuff (H.Garcia-Molina, session chair)<br><br>C. Mohan<br>Workflow Management, A Research Perspective<br><br>Jim Johnson<br>Charting the Seas of Information Technology<br><br>Wayne Duquaine<br>Notes on Third Generation Data Access<br><br>Gary Kelley<br>Informix TPCD performance (at MCI) |
| 5:30 | | Break |
| 6:00 | | Dinner |
| 7:00 | | Evening Services - Poster Boards (P. Helland leads the parade)<br>Pat will explain this ritual to us during the day. |

## ——WEDNESDAY (Sept. 20, 1995)

| | |
|---|---|
| 8:30 | TBD (Andreas Reuter to supply)<br>A Perspective on C/S Transactions from SAP |
| 9:30 | High Performance (S. DeFazio, session chair)<br>Pat O'Neill<br>DSS Performance is Now more significant than OLTP performance |
| 10:00 | Break |
| 10:15 | High Performance, continued (S. DeFazio, session chair)<br>Ranga Rengarajan<br>Subject is transaction performance (big tpc numbers)<br>Bill Baker<br>Subject is performance in multidimensional (IRI) |
| 11:15 | The Market Perspective (D.Haderle, chair)<br>Alfred Spector<br>Challenges from the Market |
| 11:45 | End (D. Haderle) |