Project P817-PF

Database Technologies for Large Scale Databases in Telecommunication

Deliverable 1

Overview of Very Large Database Technologies and Telecommunication Applications using such Databases

Volume 3 of 5: Annex 2 - Data manipulation and management issues

Suggested readers:

- Users of very large database information systems
- IT managers responsible for database technology within the PNOs
- Database designers, developers, testers, and application designers
- Technology trend watchers
- People employed in innovation units and R&D departments.

For full publication

March 1999

EURESCOM PARTICIPANTS in Project P817-PF are:

- BT
- Deutsche Telekom AG
- Koninklijke KPN N.V.
- Tele Danmark A/S
- Telia AB
- Telefonica S.A.
- Portugal Telecom S.A.

This document has been approved by EURESCOM Board of Governors for distribution to all EURESCOM Shareholders.

Preface

(Edited by EURESCOM Permanent Staff)

The Project will investigate different database technologies to support high performance and very large databases. It will focus on state-of-the-art, commercially available database technology, such as data warehouses, parallel databases, multi-dimensional databases, real-time databases and replication servers. Another important area of concern will be on the overall architecture of the database and the application tools and the different interaction patterns between them. Special attention will be given to service management and service provisioning, covering issues such as data warehouses to support customer care and market intelligence and database technology for web based application (e.g. Electronic Commerce).

The Project started in January 1998 and will end in December 1999. It is a partially funded Project with an overall budget of 162 MM and additional costs of around 20.000 ECU. The Participants of the Project are BT, DK, DT, NL, PT, ST and TE. The Project is led by Professor Willem Jonker from NL.

This is the first of four Deliverables of the Project and is titled: “Overview of very large scale Database Technologies and Telecommunication Applications using such Databases”. The Deliverable consists of five Volumes, of which this Main Report is the first. The other Volumes contain the Annexes. Other Deliverables are: D2 “Architecture and Interaction Report”, D3 “Experiments: Definition” and D4 “Experiments: Results and Conclusions”.

This Deliverable contains an extensive state-of-the-art technological overview of very large database technologies. It addresses low-cost hardware to support very large databases, multimedia databases, web-related database technology and data warehouses. Contained is a first mapping of technologies onto applications in the service management and service provisioning domain.

Executive Summary

This annex contains a part of the results of the literature study to database technologies for very large databases. The other parts can be found in other annexes. This document is subdivided in three parts viz.:

- Transaction Processing Monitors: describing the concepts and added-value of transaction processing monitors together with available products
- Retrieval and Manipulation: describing aspects of data retrieval and manipulation in distributed databases, plus an overview of current commercial database systems.
- Backup and Recovery: describing backup and recovery strategies and their necessity in the context of high-available very large database applications.

Transaction Processing Monitors reside between the client and the (database) server. Their main goals are to ensure that transactions are correctly processed, often in a heterogeneous environment, and that workload is equally distributed among available systems in compliance with security rules. The most mature products are Tuxedo, Encina, TOP END and CICS. Grip and Microsoft Transaction Server (MTS) lack some features and standards support. If you are looking for enterprise-wide capacity, consider Top End and Tuxedo. If your project is medium sized, consider Encina as well. If you look for a product to support a vast number of different platforms then Tuxedo may be the product to choose. If DCE is already used as underlying middleware then Encina should be considered. Regarding support of objects or components MTS is clearly leading the field with a tight integration of transaction concepts into the COM component model. Tuxedo and Encina will support the competing CORBA object model from the OMG. There seems to be a consolidation on the market for TP Monitors. On the one hand Microsoft has discovered the TP Monitor market and will certainly gain a big portion of the NT server market. On the other side the former TP Monitor competitors are merging which leaves only IBM (CICS and Encina) and BEA Sytems (Tuxedo and TOP END) as the old ones. The future will heavily depend on the market decision about object and component models such as DCOM, CORBA and JavaBeans and the easy access to integrated development tools.

Retrieval and manipulation of data in different database architectures has various options for finding optimal solutions for database applications. In recent years many architectural options have been discussed in the field of distributed and federated databases and various algorithms have been implemented to optimise the handling of data and to optimise methodologies to implement database applications. Nevertheless, retrieval and manipulation in different architectures apply similar theoretical principals for optimising the interaction between applications and database systems. Efficient query and request execution is an important criterion when retrieving large amounts of data. This part also covers a number of commercial database products competing in the VLDB segment, most of which run on various hardware platforms. The DBMSs are generally supported by a range of tools for e.g. data replication and data retrieval.

Being able to backup and recover data is essential for an organisation as no system (not even a fault-tolerant one) is free of failures. Moreover, errors are not only caused by hard- and software failures but also by (un)willfull wrong user actions. Some types of failures can be corrected by the DBMS immediately (e.g. wrong user operations)

but others need a recovery action from a backup device (e.g. disk crashes). Depending on issues like the type of system, the availability requirements, the size of the database etc., one can choose from two levels of backup and recovery. The first is on the operating system level and the second on the database level. Products of the former are often operating system dependent and DBMS independent and products of the latter the other way around. Wich product to choose depends on the mentioned issues.

List of Authors

Part 1

Berend Boll Deutsche Telekom Berkom GmbH, Germany

Part 2

Frank Norman Tele Danmark

Wolfgang Müller Deutsche Telekom

Part 3

Sabine Gerl Deutsche Telekom Berkom GmbH, Germany

Andres Peñarrubia Telefonica, Spain

Table of Contents

Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
ACL	Access Control List. Used to define security restictions for objects/resources
COM	Microsoft's Component Object Model
CORBA	OMG's Common Object Request Broker Architecture
DBA	Database Administrator
DBMS	Database Management System
DBS	Data Base System
DCE	Open Group's Distributed Computing Environment
DCOM	Microsoft's Distributed Component Object Model
DDL	Data Definition Language
DML	Database Manipulation Language
DRM	Disaster Recovery Manager
DSA	Database Server Architecture
DTP Model	Distributed Transaction Processing Model, defined by the Open Group.
FDBS	Federated Database System
GIF	Graphics Interchange Format
HSM	Hierarchical Storage Management
HTML	Hypertext Markup Language
IDL	Interface Definition Language
JDBC	Java Database Connectivity
LOB	Line-Of-Business
MDBS	Multi Database System
MOM	Message-Oriented Middleware
MPP	Massively Parallel Processing
NCA	Network Computing Architecture
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
OMG	Object Management Group
Open Group	None-profit, vendor-independent, international consortium. Has created the DTP Model and the XA Standard for Transaction Processing.
ORB	Object Request Broker
ORDBMS	Object-Relational DBMS

PDF	Portable Document Format
RDBMS	Relational DBMS
RPC	Remote Procedure Call
SMP	Symmetric Multiprocessing
TLI	Transport Layer Interface (APIs such as CPI-C the SNA peer-to-peer protocol and Named Pipes)
TP Monitor	Transaction Processing Monitor
TPC	Transaction Processing Performance Council
tpmC	Transaction Per Minute measured in accordance with TPC's C standard (TPC-C).
UDF	User-defined Function
UDT	User-defined Data Type
ULL	United Modeling Language
VLDB	Very Large Database
VLDB	Very Large Database
VLDB	Very Large Database
XA	API used to co-ordinate transaction updates across resource managers

Definitions

ACID properties	the four transaction properties: atomicity, consistency, integrity, durability.
Conversational	Kind of communication. Unlike request-response each request in a conversation goes to the same service. The service remains state information about the conversation. There is no need to send state information with each client request.
Publish-Subscribe	Kind of communication. (Publisher) Components are able to send events and other components (Subscribers) are able to subscribe to a special event. Everytime the subsribed event happens within the publisher component the subscriber component gets notified by a message.
Queue	Kind of communication. A queue provides time-independent communication. Request and Responses are stored in a queue and could be accessed asynchronously.
Request-response	Kind of communication. The client issues a request to a service and then waits for a response before performing other operations (an example is a RPC)
Resource managers	a piece of software that manages shared resources
server class	a group of processes that are able to run the code of the application program.
two-phase commit	Protocol for distributed transactions

Part 1Transaction Processing Monitors

1 Introduction

"The idea of distributed systems without transaction management is like a society without contract law. One does not necessarily want the laws, but one does need a way to resolve matters when disputes occur. Nowhere is this more applicable than in the PC and client/server worlds." - Jim Gray (May, 1993)

2 Concepts of Transactions

Transactions are fundamental in all software applications, especially in distributed database applications. They provide a basic model of success or failure by ensuring that a unit of work must be completed in its entirety.

From a business point of view a transaction changes the state of the enterprise, for example a customer paying a bill which constitutes in changes of the order status and a change on balance sheets.

From a technical point of view we define a transaction *as "a collection of actions that is governed by the ACID-properties*" ([5]).

2.1 ACID Properties

The ACID properties properties describe the key features of transactions:

- **Atomicity**. Either all changes to the state happen or none do. This includes changes to databases, message queues or all other actions under transaction control.
- **Consistency**. The transaction as a whole is a correct transformation of the state. The actions undertaken do not violate any of the integrity constraints associated with the state.
- **Isolation**. Each transaction runs as though there are no concurrent transactions.
- **Durability**. The effects of a committed transaction survive failures.

Database and TP systems both provide these ACID properties. They use locks, logs, multiversions, two-phase-commit, on-line dumps, and other techniques to provide this simple failure model.

2.2 Two Phase Commit Protocol

The *two-phase commit* protocol is currently the accepted standard protocol to achieve the ACID properties in a distributed transaction environment. Each distributed transaction has a coordinator, who initiates and coordinates the transaction.

In the **first phase** the coordinator (*root node*) informs all participating *subordinate nodes* of the modifications of the transaction. This is done via the *prepare-to-commit* message. Then the coordinator waits for the answers of the subordinate nodes. In case of success he gets a *ready-to-commit* message from each of the subordinate nodes. The root node logs this fact in a safe place for recovery from a root node failure.

If any of the subordinate nodes fails and does not send a *ready-to-commit* message to the root node then the whole transaction will be aborted.

In the **second phase** the coordinator sends a *commit* message to all subordinate nodes. They commit their actions and answer with a *complete* message. The protocol is illustrated in the figure below.

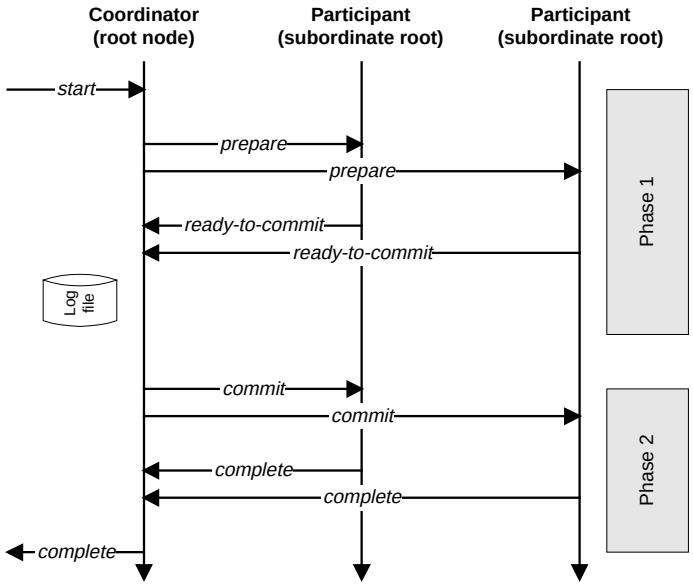

Figure 1. Two-Phase Commit Protocol

Most TP Monitors could easily handle transactions that spanned across 100 two-phase commit engines. However, the two phase commit has some limitations:

- **Performance overhead**. There is a message overhead, because the protocol does not distinguish for the different kind of transactions. That means also for read-transactions all the messages of the two-phase commit protocol will be sent, even if they are not really needed.
- **Hazard windows**. At special times a failure of a node can lead to a problem. For example, if the root node crashes after the first phase, the subordinate rotes may be left in disarray. There are workarounds, but they tend to be tricky. So the architecture should be built in a way, that the root node is located on a fault-tolerant system.

3 Concepts of TP Monitors

3.1 Why should you use a TP Monitor?

Based on the TPC ranking of February 1998, the success of TP Monitors was clearly demonstrated by the fact that every single test environment of the top 20 TPC-C benchmark results (ranked by transactions per minute) included a TP Monitor. If the same results are ranked by the price/performance ratio, 18 of the top 20 used a TP Monitor ([1], [12]).

Why are TP Monitors so popular in modern architectures and what problems do they address?

To understand this one has to take a look at how application architectures are build. All applications consist of three parts:

- presentation layer (GUI) which resides on the client
- application layer which could reside on a separate application servers
- data layer which could resides on a separate database severs.

In many applications the is no clear separation of those layers within the code. The same code could do work for all three layers. Good structured applications separate these layers on the code (software) and on the hardware level.

2-tier applications are applications that have the application layer integrated within the presentation layer on the client and/or the data layer (as Remote Procedures) an the database server.

3-tier application separate the application and most often run them on special application servers. Still with a separation of an application layer one could run an application layer services physically on the client or on the database server. But the point is that there exists a separation and a possibility to redistribute the application layer services on special purpose machines based on workload and performance issues.

Basically TP Monitors provide an architecture to build 3-tiered client/server applications. According to Standish Group, in 1996 57% of mission-critical applications were built with TP Monitors. This is because former 2-tiered architectures have the following problems:

- For each active client the databases must maintain a connection which consumes machine resources, reducing performance as the number of clients rises.
- 2-tiered applications scale well to a point and then degrade quickly.
- Reuse is difficult, because 2-tiered application code like stored-procedures is tightly bound to specific database systems
- Transactional access to multiple data sources is only possible via gateways. But gateways integrate applications on the data level which is "politically" and technically unstable and not adoptive to change ("politically" refers to the problem, that the owner of the data might not be willing to give access at the data level outside of his department).
- Database stored-procedures could not execute under global transaction control. They could not be nested and programmed in a modular basis. Also they are vendor-specific.
- Outside of trusted LAN environments the security model used in 2-tiered systems doesn't work well, because it focuses on granting users access to data. Once administrators give a user write or change access to a table, the user can do almost anything to the data. There is no security on the application level.
- There is no transaction mechanism for objects (CORBA) or components (COM, JavaBeans).

TP Monitors address all of the above problems and despite their rare usage in average client/server-applications they have a famous history in the mainframe area. Nowadays they tend to raise more and more attention because of the development of commercial applications on the Internet.

3.2 Standards and Architecture

A TP Monitor could be described as an operating system for transaction processing. It delivers the architecture to distribute and manage transactions over a heterogeneous infrastructure. This implicitly forces the application architecture to be 3-tier, because a TP Monitor is a type of middleware.

A TP Monitor does three things extremely well:

- **Process management** includes starting server processes, funneling work to them, monitoring their execution and balancing their workloads.
- **Transaction management** means that the TP Monitor guarantees the ACID properties to all the programs that run under its protection.
- **Client/Server communication management** allows clients (and services) to invoke an application component in a variety of ways - including request-response, conversations, queuing, publish-subscribe or broadcast.

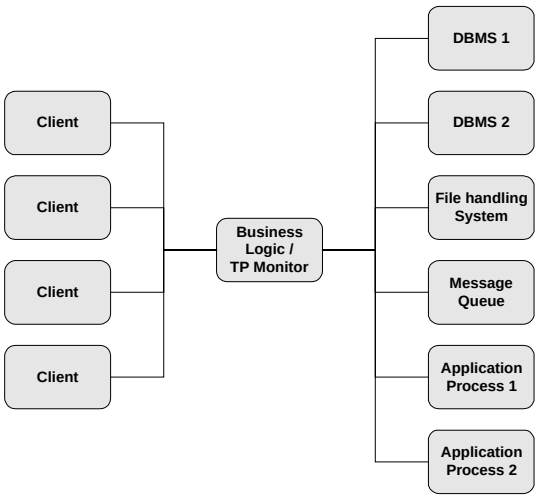

Figure 2. 3-tier client/server with TP Monitor

A TP Monitor consists of several components. The *Open Group's Distributed Transaction Processing Model (1994*) ([10]), which has achieved wide acceptance in the industry, defines the following components:

- The **application program** contains the business logic. It defines the transaction boundaries through calls it makes to the transaction manager. It controls the operations performed against the data through calls to the resource managers.
- **Resource managers** are components that provide ACID access to shared resources like databases, file systems, message queuing systems, application components and remote TP Monitors.
- The **transaction manager** creates transactions, assigns transaction identifiers to them, monitors their progress and coordinates their outcome.
- The **Communication Resource Manager** controls the communications between distributed applications.

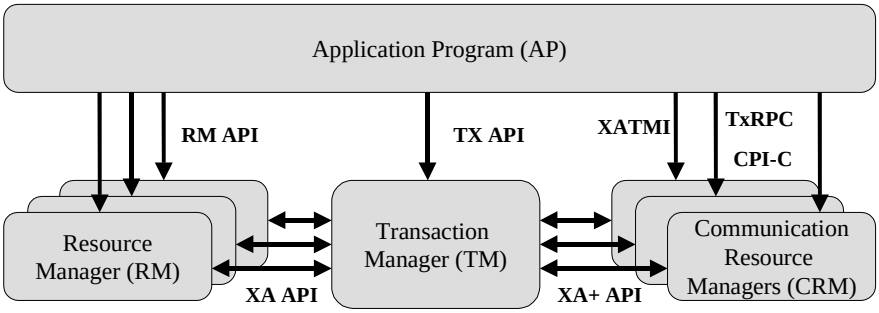

Figure 3. X/Open 1994 Distributed Transaction Processing Model

The following interfaces exist between the components:

- **RM API** is used to query and update resources owned by a resource manager. Typically the provider of the resource manager defines this interface. For example, the API for a relational database would be an SQL API.
- **TX API** is used to signal the transaction manager the beginning, the commitment or the abortion of a transaction.
- **XA API** is used to coordinate transaction updates across resource managers *(two-phase commit protocol).*
- **XA+ API** defines the communication between the communications resource managers and the transaction manager.

  This interface, however, was never ratified, so not all the vendors use it. On the whole, the XA+ interface is relatively unimportant as, generally, it is used internally within the product.
- **XATMI, TxRPC and CPI-C** are transaction communication programming interfaces.

  There is no general standard. XATMI is based on BEAs Tuxedo's Application to Transaction Monitor Interface (ATMI); TxRPC is based on the Distributed Processing Environment (DCE) RPC interface and CPI-C is based on IBMs peer-to-peer conversational interface.

The role of these components and interfaces within a 3-tier architecture is visualized with the following picture. The client (presentation layer) communicates with the Application Program (AP). The access to the data-layer is done via the Resource Manager (RM) component. If several distributed TP Monitors are involved within a transaction, the Communication Resource Managers (CRM) are responsible for the necessary communication involved.

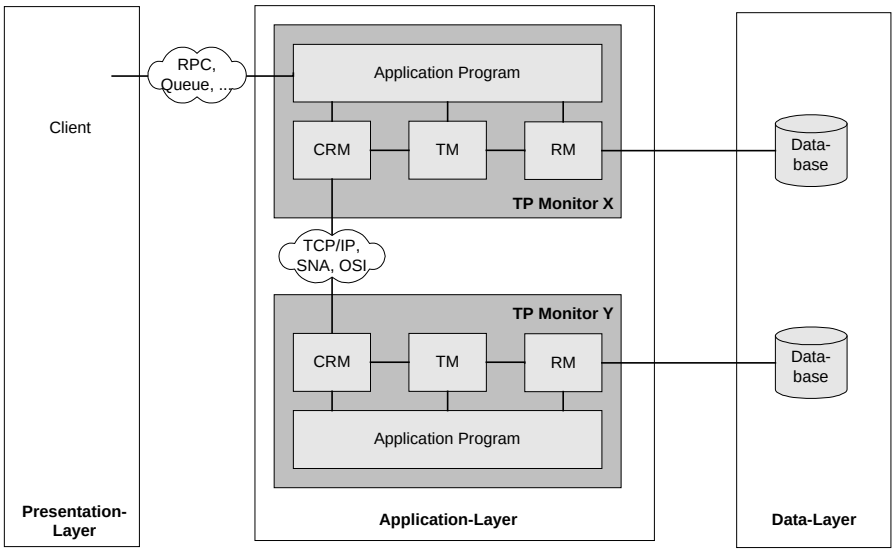

Figure 4. TP Monitor within a 3-tier architecture

Actual implemenations of TP Monitors consists of serveral other components but these differ from product to product. Therefore the OpenGroup DTP Model could only be used to understand the main function of a TP Monitor: distributed transaction management.

Other components include modules for client/server communication such as queues (which all TP Monitors have now included), administration tools, directory services and many more. We refer here to the commercial product chapter of this part.

The key interface is XA, because it is the interface between the resource manager from one vendor and the DTPM from the middleware vendors.

XA is not a precise standard, nor does it comprise source code which can be licensed. It is a specification against which vendors are expected to write their own implementations. There are no conformance tests for the X/Open XA specification, so it is not possible for any vendor to state that it is 'XA-compliant'; all that vendors can claim is that they have used the specification and produced an XA implementation which conforms to it. The situation is complicated even more by the fact, that the commitee which devised the XA Model and the specifications has now disbanded.

The DBMSs that definitely support the XA standard and also work with all DTPMs that support the standard are Oracle, Informix, SQL Server and DB2/6000.

In the following chapter we will describe special features of TP Monitors in more detail.

3.3 Transaction management

TP Monitors are operation systems for business transactions. The unit of management, execution and recovery is the transaction. The job of the TP Monitor is to ensure the ACID properties even in a distributed resource environment while maintaining a high transaction throughput.

The ACID properties are achieved through co-operation between the transaction manager and resource managers. All synchronisation, commit and rollback actions are co-ordinated by the transaction manager via the XA interface and the 2-phase-commit protocol.

3.4 Process management

3.4.1 Server classes

TP Monitors main job is managing processes. They keep pools of pre-started application processes or threads (called *server classes*). Each process or thread in the server class is able to run the application. The TP Monitor balances the work between them. Each application can have one or more server classes.

These server processes are pre-warmed. They are already loaded in memory, have a context and are ready to start instantly. If they finish their work for one client request they stay in memory and wait for the next request.

3.4.2 Reduced server resources

Keeping and sharing these pre-warmed processes dramatically reduces the number of concurrent processes and therefore makes it possible to support huge number of clients.

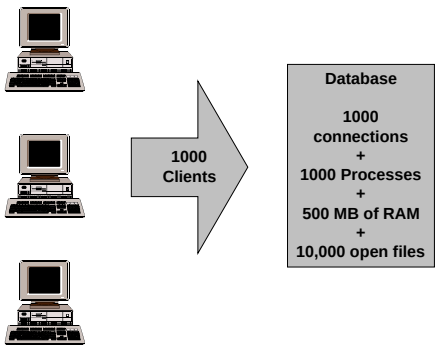

Figure 5. Process Management without TP Monitor

This kind of process management could be described as a pooling and funnelling. It provides scalability for huge database applications because it addresses the problem that databases establish and maintain a separate database connection for each client. It is because of this feature that all leading TPC-C benchmarks are obtained by using TP Monitors ([12]).

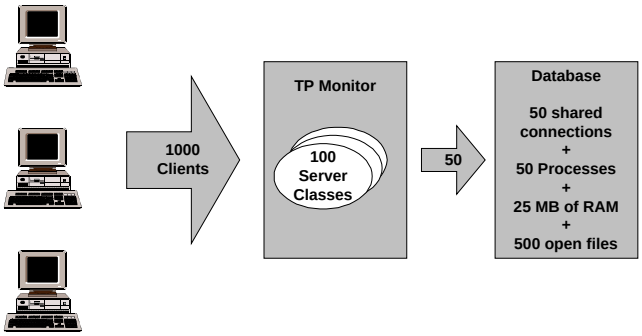

Figure 6. Process Management with TP Monitor - pooling and funneling

3.4.3 Dynamic load balancing

If the number of incoming client requests exceeds the number of processes in a server class, the TP Monitor may dynamically start new processes or even new server classes. The server classes can also be distributed across multiple CPUs in SMP or MPP environments. This is called load balancing. It could be done via manual adminstration or automatically by the TP Monitor. In the latter case it is called *dynamic load balancing*.

This load balancing could be done either for appication layer or data layer processes. For data layer processes the database server bottleneck could be released by having several (replicated) databases over which the TP Monitor could distribute the workload.

3.5 Robustness

TP systems mask failures in a number of ways. At the most basic level, they use the ACID transaction mechanism to define the scope of failure. If a service fails, the TP Monitor backs out and restarts the transaction that was in progress. If a node fails, it could migrate server classes at that node to other nodes. When the failed node restarts, the TP system's transaction log governs restart and recovery of the node's resource managers.

In that way the whole application layer running as server classes on potentially distributed application servers is highly available. Each failure (local process/server class or total machine) could be masked by the TP Monitor by restarting or migrating the server class. Also the data layer is robust to failure, because a resource manager under TP Monitor control masks the real database server. If a database server crashes, the TP Monitor could restart the database server or migrate the server classes of the resource manager to a fallback database server. In either cases (failure of application or data layer) the failures will be fixed by the TP Monitor without disturbing the client. the TP Monitor is acting as a *self-healing system*, it not just handles faults, it automatically corrects them.

This could be done because the client is only connected to one TP Monitor. The TP Monitor handles all further connections to different application, database, file services etc. The TP Monitor handles failures of servers and redirects the requests if necessary. That implies that the TP Monitor itself should be located on a fault-tolerant platform. Still it is possible to run several TP monitor server processes on different server machines that could take over the work of each other in case of failure, so that even this communication links has a fallback.

Different to a 2-tier approach a client could crash and leave an open transaction with locks on the database, because not the client but the TP Monitor controls the connections to the database. Therefore the TP Monitor could rollback connections for a crashed client.

TP systems can also use database replicas in a fallback scheme - leaving the data replication to the underlying database system. If a primary database site fails, the TP Monitor sends the transactions to the fallback replica of the database. This hides server failures from clients - giving the illusion of instant fail-over.

Moreover TP Monitors deliver a wide range of synchronous and asynchronous communication links for application building including RPCs, message queues

(MOMs), ORB/DCOM-invocation, conversational peer-to-peer and event-based publish-and-subscribe communication. Therefore based on the infrastructure and the nature of the services, the best communication links could be chosen.

In short the following services are delivered:

- Failover Server
- Automatic retry to re-establish connection on failure
- Automatic restart of process on client, server or middleware
- Automatic redirection of requests to new server instance on server failure.
- Appropriate and reliable communication links.
- Transaction deadlocks are dissolved.

3.6 Scalability

3.6.1 Shared process resources

TP Monitors are the best at database funneling, because they are able to use only a handful of database connections while supporting thousands of clients (see section on Process Management). Therefore by reducing the connection overhead on the database server they make the database side much more scalable.

In a 3-tier TP Monitor architecture for database access there are typically at least a factor of 10 fewer database connections necessary than in a straightforward 2-tier implementation.

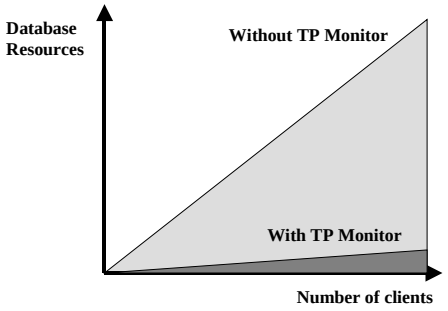

Figure 7. Scalability of the database layer

3.6.2 Flexible hardware requirements

Scalability is not only enhanced on the database side. With dynamic load balancing the load could be distributed over several machines. By doing this the whole architecture becomes very flexible on the hardware side. To increase the total processing capabilities of the 3-tier architecture one could either upgrade server machines or increase the number of servers. The decision could be made based on costs and robustness of the hardware and on company policies.

3.7 Performance

Performance could be sped up by different approaches. First of all it is based on the effective handling of processes in one machine. TP Monitors deliver an architecture

for multi-threading/multi-tasking on the middleware and the application layer. That includes:

- Automatic creation of new threads/tasks based on loads
- Automatic deletion of threads/tasks when load reduces
- Application parameters dynamically altered at runtime.

TP Monitors are speeding up performance by keeping the pool of processes in memory together with their pre-allocated resources. In this *pre-warmed environment* each client request could be instantaneously processed without the normal starting phase.

Another way to enhance performance is to use more processors/machines to share the workload. TP Monitor support load balancing for:

- multiple processors in one machine (SMP or MPP)
- multiple machines (nodes)
- multiple middleware, application and resource services (which may be started on demand as stated above).

3.8 Security

The TP Monitor provides a convenient way to define users and roles and to specify the security attributes of each service in an *access control list (ACL)*. It authenticates clients and checks their authority on each request, rejecting those that violate the security policy. Doing this it delivers a fine granulate security on the service/application level which adds a lot of value to security on the database or communication line levcl.

Aspects of security typically include:

- **Role**. Based on roles, users have security restrictions to use the services.
- **Workstation**. It is defined which physical workstation is allowed to request what service.
- **Time**. A service might be limited to a special period of time.

For example, in a payment systems, clerks might be allowed to make payments from in-house workstations during business hours.

Furthermore a TP Monitor supports general communication security features as

- Authentication
- Authorization
- Encryption.

3.9 Transaction profiles

It is possible to assign profiles to transactions/server classes. Attributes of profiles include:

- **Priority**. Transaction could have different priorities which are used by the TP Monitor to do load-balancing.

- **Security**. Defines who is allowed to use the service.

3.10 Administration

As described in the above sections processes are clustered into pools in service classes. Typically, short-running or high-priority services are packaged together, and batch or low-priority work is packaged in separate low-priority server classes.

After packaging the services, an administrator could assign security attributes to them.

Change is constant, and TP Monitors manage it on the fly. They can, for example, automatically install a service by creating a server class for it *(dynamic load-balancing)*. Even more interesting, they can upgrade an existing service in place by installing the new version, creating new service classes that use it, and gradually killing off old server classes as they complete their tasks *(on-the-fly software release upgrades)*. Of course, the new version must use the same request-reply interface as the old one.

This load-balancing feature could also be used for planned outages in case of maintenance for 24x7 applications to shift all load to a back-up machine.

The registration and start-up/shut-down of resources is unbundled from the application, because the application uses services of the TP Monitor which are mapped to available resources. Therefore resources could be added and removed on the fly. There is no direct connection between a resource and an application (i.e. via an IP-number or DNS-name of a Database-Server).

Overall TP Monitors help to configure and manage client/server interactions. They help system administrators to install, configure and tune the whole system, including application services, servers and large populations of clients.

Administration features include:

- Remote installation of middleware and applications
- Remote configuration
- Performance monitoring of applications, databases, network, middleware
- Remote start up/shut down of application, server, communication link and middleware
- Third party tool support
- Central administration
- Fault diagnosis with alerts/alarms, logs and analysis programs.

Even if it should be possible to administer processes and load-balancing manually, the preferred option should always be an automatic, self-administrating and self-healing system.

3.11 Costs

TP Monitors help to reduce overall costs in large, complex application systems servers. They do this in several ways:

- **Less expensive hardware**. By doing optimized load-balancing together with better performance, TP Monitors help to use resources more efficiently.

Moreover by funneling client request, the number of concurrent processes running on the resources (databases) could be drastically reduced (normally by the factor of 10). Therefore the TP Monitor architectures have lower hardware requirements.

- **Reduced Downtimes**. Because of the robust application architecture downtimes of applications could be reduced. This reduces lost profits because of downtimes.
- **Reduced license costs**. The funneling effect reduces the number of concurrent open connections to the database server, therefore reducing expensive license costs.
- **Development time savings.** By delivering an application architecture and forcing the developers to build the system over a 3-tier architecture system development and maintenance time is reduced (according to Standish Group by up to 50%).

According to Standish group this may result in total system cost savings of greater than 30% over a data-centric (2-tier) approach.

3.12 3-tier architecture framework

TP Monitors provide a framework to develop, build, run and administer client/server applications. Increasingly, visual tool vendors are integrating TP Monitors and making them transparent to the developer.

TP Monitors deliver shared state handling (transaction context) to exchange information between the services, freeing developers from this task.

They introduce an event-driven component based programming style on the server side. Services are created and only function calls or objects are exported not the data itself. This allows to keep adding functions and let the TP Monitor distribute them over multiple servers.

They provide a clear 3-tier architecture framework. It is almost impossible to violate this paradigma by "lazy" programmers. The application becomes strictly modularised with decoupled components. This leads to a state of the art architecture which fits best into current object and component paradigmas. Maintenance and re-use are best supported by such architectures.

3.13 When not to use a TP Monitor

Despite the fact that a TP Monitor has a lot of advantages you do not need it for all kind of applications and there are also some other drawbacks:

- **Few users**. Even if you have a VLDB with complex data you do not need a TP Monitor if you have just a few concurrent users at any time. TP Monitor helps nothing in managing data but it helps a lot in managing processes.
- **Raised Complexity**. If you include a new component like a TP Monitor in the architecture it raises the complexity to be mastered (at least in the beginning and regarding the knowledge of the people involved). So you need to have the necessary knowledge in your development and administration team. But in fact with big systems the complexity is not raised but lowered, because the whole application structure becomes better modularized and decoupled (see section on 3-tier architecture framework).

- **Vendor dependence**. Because the architecture of all TP Monitor systems is different it is not so easy to switch from one TP Monitor to another TP Monitor. Therefore for some part you are locked with a one vendor solution.

4 Commercial TP Monitors

The following comparison of the different available TP Monitors is heavily based on a study done by *Ovum Publications* ([11]) Ovum Publications, www.ovum.com, "Distributed TP Monitors", February 1997 and recent information from the product companies.

In general all of the products follow the OpenGroup standard DTP architecture described in Figure 3. They differ in their support of the XA-Interface between the Transaction manager and the Resource manager which is of the greatest importance for the for the usage of a TP Monitor.

Also some of them integrate the Transaction Manager and the Communication Resource Manager into one component (BEAs Tuxedo). But this has no importance for the usage of the TP Monitor because anyhow the interface between those two components is only used internally and should have no impact on the applications built on the TP Monitor. Also this XA+ interface between those components was never officially standardized.

So the differentiation between the products should be mostly done by

- supported resources
- supported communication protocols
- platform support
- company history, structure and future
- Internet support
- Object / Component support
- Price
- Easy usage
- Future developments
- market share
- your expertise in similar/complemtary tools and programming languages

4.1 BEA Systems Inc.’s Tuxedo

4.1.1 Summary

Key Points

- Transaction control available via XA compliant resource managers, such as Oracle, Informix, and others. Interoperates with MQSeries via third party or own gateways. Can also interoperate with other TP environments via the XAP OSI-TP standard and SNA (CICS) and R/3.

- Runs on OS/2, MS-DOS, Windows 3, Windows 95, Windows NT, Apple Macintosh, AS/400 and a wide range of UNIX flavours from the major system providers. Supports Bull GCOS8, ICL VME, MVS IMS, Unisys A series, and MVS/CICS via third party or own gateways. [Note: I am not aware of GCOS8 or VME functionality in the latest versions of the product, although I may not know of all the 3rd party versions.]
- Directly supports TCP/IP, via sockets or TLI. Indirectly supports SNA LU6.2.

Strengths

- Excellent directory services, with a labor-saving architecture and good administrative support tools, all suited to large-scale deployment
- Vast array of platforms (Hardware, Netware, Middleware) supported, with links to some environments which have no other strategic middleware support, such as GCOS and VME
- All technology is now available from one supplier, with a correspondingly clear strategy for development, support and evolution

Weaknesses

- BEA still has some way to go to integrate all the technology and people it has acquired - especially after buying TOP END from NCR.
- Guaranteed delivery services on the communication services side are not well developed
- Load balancing services should be more automated

4.1.2 History

AT&T started the development of Tuxedo in 1979 as part of an application called LMOS (Line Maintenance and Operation System). The product evolved internally within Bell Labs until 1989, when AT&T decided to license the technology to OEMs (value-added resellers).

At 1992 AT&T had spun off the development of Unix, languages and Tuxedo into a new group named Unix Systems Laboratories (USL). In 1993, Novell bought USL and started to develop plans which involved the integration of Tuxedo with Novell's Directory System and AppWare application development tools. These plans never worked out.

In September 1994, Novell released version 5 of Tuxedo. Enhancements to the product included support for DCE, extra platform support, a runtime trace feature, dynamic data-dependent routing and the 'domain' feature - used by systems administrators to configure Tuxedo servers into autonomous groups.

However, in February 1996, BEA Systems assumed all development, sales and support responsibilities for Tuxedo'. BEA was a start-up company specifically set up to acquire and develop middleware technology in the transaction processing area. Novell retained the right to develop the technology on NetWare platforms. BEA acquired the rights to develop the technology on all other platforms.

Despite the somewhat confusing language of the announcement, BEA has effective control of Tuxedo, the technology and its future development. It has obtained an exclusive 'licence' to the technology in perpetuity. The entire Tuxedo development,

support and sales team - in effect the entire Tuxedo business unit - transferred to BEA which now has about 100 to 150 developers working on Tuxedo, including many of the developers from AT&T and Bell Labs.

Release 6.1 was released in June 1996 and added the event broker, an administration API and ACLs.

1997 BEA Systems made a revenue of $ 61.6 Mio.

May 1998, BEA Systems announced to buy the competing TP Monitor TOP END from NCR.

4.1.3 Architecture

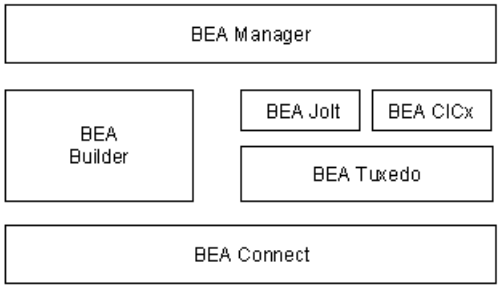

Figure 8. Tuxedo architecture

The Tuxedo architecture includes the following components:

- **Core System** provides the critical distributed application services: naming, message routing, load balancing, configuration management, transactional management, and security.
- The **Workstation** component off-loads processing to desktop systems. This allows applications to have remote clients, without requiring that the entire BEA TUXEDO infrastructure reside on every machine.
- **Queue Services** component provides a messaging framework for building distributed business workflow applications.
- **Domains** allow the configuration of servers into administratively autonomous groups called domains.
- **DCE Integration** is a set of utilities and libraries that allows integration between The Open Group's DCE.
- **BEA CICx** an emulator of the CICS transaction processing product which runs on Unix.
- **BEA Connect** are connectivity gateways to other TP environments (via SNA LU6.2 and OSI-TP protocol)
- **BEA Builder** covers tools that assist in the development and testing of Tuxedo-based applications.
- **BEA Manager** is the administration component
- **BEA Jolt** for Web integration

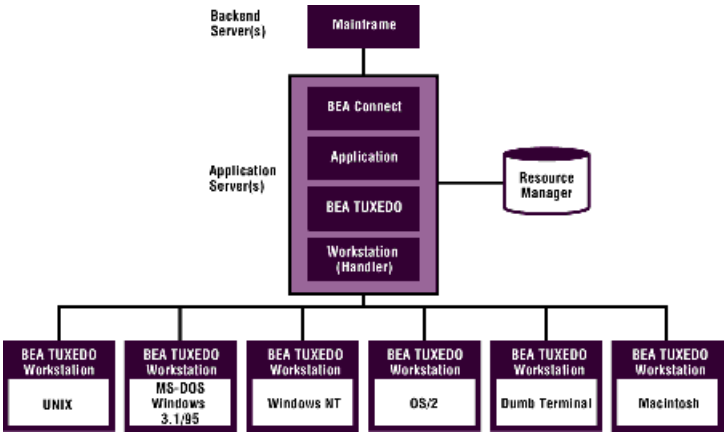

Figure 9. Tuxedo 3-tier architecture

Special features include:

- **Event Brokering System** which implements an event system based on the publish-and-subscribe programming paradigm. This allows for the notification of events on a subscription basis.
- **Security** is offered for both 40-bit and 128-bit Link Level Encryption add-on products to provide security for data on the network. It also supports service-level Access Control Lists (ACLs) for events, queues, and services.
- **Cobol Support**. A COBOL version of ATMI is provided.
- **Service Directory**. The Bulletin Board, located on every server node participating in an application, serves as the naming service for application objects, providing location transparency in the distributed environment. It also serves as the runtime repository of application statistics.
- **Internationalization**. In compliance with The Open Group's XPG standards, users can easily translate applications into the languages of their choice. Languages can also be mixed and matched within a single application.

4.1.4 Web Integration

BEA Jolt enables Java programs to make Tuxedo service requests from Java-enabled Web browsers across the Internet (or intranets). The aim of Jolt is to get round the restrictions of normal Internet communication.

Jolt consists of a collection of Java classes. It also replaces HTTP with its own enhanced Jolt Transaction Protocol.

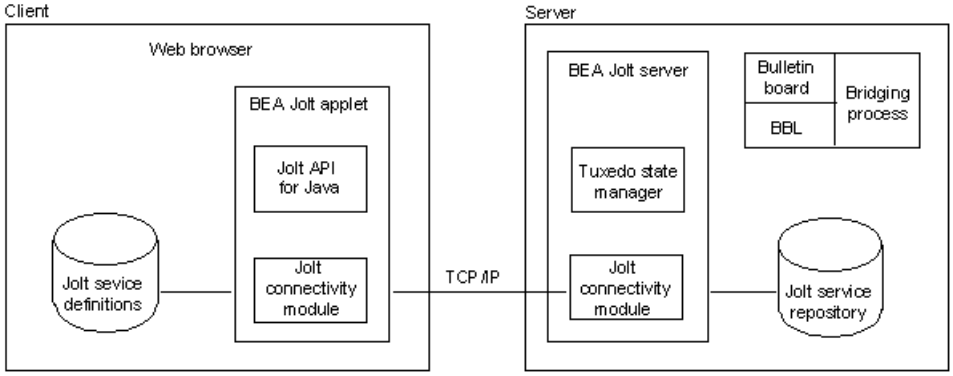

Figure 10. Jolt architecture

4.1.5 When to use

- You are developing object-based applications. Tuxedo works with non-object based applications, but it is especially suited to object based ones. In fact, you cannot implement object-based applications involving transactions without under-pinning them with a distributed TP Monitor like Tuxedo. (ORBs without TP underpinning are not secure.)
- You have a large number of proprietary and other platforms which you need to integrate. You use Oracle, DB2/6000, Microsoft SQL Server, Gresham ISAM-XA, Informix DBMSs or MQSeries, and you need to build transaction processing systems that update all these DBMSs/resource managers concurrently.
- You want to integrate Internet/intranet based applications with in-house applications for commercial transactions.

4.1.6 Future plans

There are plans to enhance Tuxedo in the following key areas:

- Java, internet integration – see newest release of Jolt 1.1: HTML client support via BEA Jolt Web Application Services and JavaBeans support for BEA Jolt client development via JoltBeans
- Exploiting the Object Technology by creating an object interface to ATMI for Java (BEA Jolt), COM, CORBA and C++ - the CORBA, COM and C++ interfaces are available using M3, Desktop Broker and BEA Builder for TUXEDO Active Expert
- **EJB Builder**, a graphical tool for building Enterprise JavaBeans (EJBs) applications.
- Exploiting the Object Technology by creating an object interface to ATMI for Java (BEA Jolt), COM, CORBA and C++
- The **Iceberg project** (release date June 1998) includes an updated version of BEA Tuxedo; a revised version of BEA ObjectBroker, formerly called Digital's CORBA ORB; and an integrated pairing of the two. This will integrate ORBS into Tuxedo.

- additional security features like with link level encryption, end-to-end encryption, digital signatures and built-in support for off-the-shelf security systems such as RSA - TUXEDO 6.4 provides most of these features.
- More support is also planned for multi-threading and more automated load balancing.
- tighter integration with CICS
- the BEA Manager will be supported on Windows NT and a browser-based administration tool will be released.

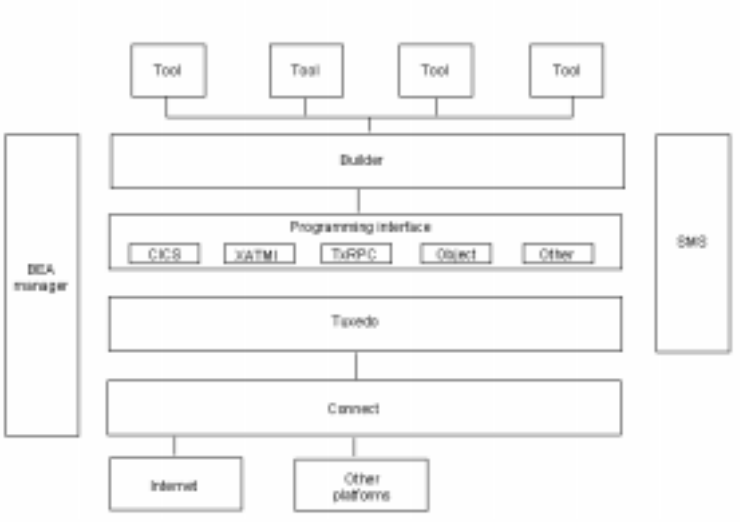

Figure 11. Tuxedo future plans

4.1.7 Pricing

Tuxedo costs about $ 5000 per development seat. The charges for runtimes are about $ 550 for concurrent use of a Tuxedo runtime and $ 125 for non-concurrent use per user per Tuxedo runtime.

Jolt is sold on a per-server basis and its cost is related to the number of users that can access any server. The minimum likely charge is about $ 3500.

4.2 IBM's TXSeries (Transarc's Encina)

4.2.1 Summary

Key points

- Based on DCE, X/Open, Corba II and JavaBeans standards
- Runs on AIX, HP-UX, SunOS, Solaris, Digital Unix, Windows 3, Windows NT. Third party versions from Hitachi (HI-UX), Stratus (FTX) and Bull (DPX). Two-phase commit access to MVS CICS through DPL and access to IMS through Encina client on MVS
- Supports TCP/IP and LU6.2 via integrated Peer-to-Peer gateway

Strengths

- Adds considerable value to DCE
- Extensive, well developed product set
- Excellent service offering with well defined, helpful consultancy and technical support
- Encina++ provides distributed object support

Weaknesses

- TXSeries supports only a small subset of the platforms DCE supports
- Third-party tool support is poor and the TXSeries interface is not easy to understand - the choices can be bewildering to a novice user
- Limited automatic failover features (no automatic redirection of requests to new server instances or new nodes).

4.2.2 History

Transarc has its roots in pioneering research into distributed databases, distributed file systems and support for transaction-based processing, which was undertaken at Carnegie Mellon university in the early 1980s.

Shortly after Transarc was founded in 1989, Hewlett-Packard, IBM, Transarc, Digital and Siemens met under the auspices of the OSF to create the architecture that was to become DCE. Transarc played a major role in providing the vision behind DCE. Transarc's AFS became the DFS component of DCE and was released as the first commercial version in 1994.

Encina in its first product version was released in 1992.

Transarc's close links with IBM resulted in an agreement with it that Encina should form the foundation for CICS on other platforms besides MVS. A joint team was formed to build CICS over Encina services, initially on AIX. About 60% of the code bases of Encina and Encina for CICS is common, but there are, in effect, two products for two markets (this is further explored in the CICS part).

IBM recently bought Transarc and bundled the two products Encina 2.5 and CICS 4.0 into a new product called IBM TXSeries 4.2. Also included in the product package; MQSeries, Domino Go Web Server and DCE servers and Gateways.

4.2.3 Architecture

TXSeries is a distributed transaction processing monitor, based on the XA standard and including support for both transactions and nested transactions. It supports two-phase commit across:

- heterogeneous DBMSs, file systems and message queuing systems (such as IBM MQSeries) supporting the XA standard
- the queues (RQS) and record oriented file system (SFS) provided by TXSeries.
- any LU6.2-based mainframe application(in the case of CICS this is full 2-phase 2-way transactional support through DPL (Distributed Program Link) which is a higher level call interface than LU6.2).

TXSeries provides synchronous and asynchronous support for client-server based applications. It supports synchronous processing using remote procedure calls and asynchronous processing using either its recoverable queuing service (RQS)or IBM MQSeries, which is included in the TXSeries server package

TXSeries is layered over DCE. The developer can access DCE services and TXSeries has been built to take advantage of them. Services such as time, security, threads support and directory are all provided by DCE.

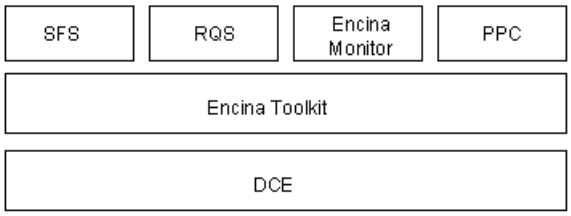

Figure 12. TXSeries architecture

The Encina Monitor

The Encina Monitor is the core component of TXSeries and consists of run-time and development libraries. It provides programmers with tools and high level APIs with which to develop applications, as well as run-time services such as load balancing, scheduling and fault tolerance services. The Encina Monitor comes with a GUI-based management and administration tool, called Enconsole, which is used to manage all resources in the DCE network (Encina servers, clients and DCE resources such as cells) from a central point.

The Monitor also acts as the central co-ordination agent for all the services. For example, it contains processing agents that receive and handle client requests, multi-thread services and multi-thread processes.

Further modules are:

- **Encina Toolkit** is built from several modules including the logging service, recovery service, locking service, volume service and TRAN - the two-phase commit engine.
- **Recoverable queuing service** (**RQS)** provides a message queue for message storage.
- **Encina structured file server (SFS)** is Transarc's own record-oriented file handling system
- **Peer-to-peer communication (PPC)** executive is a programming interface that provides support for peer-to-peer communication between TXSeries-based applications and either applications on the IBM MVS mainframe or other Unix applications, using the CPI-C (common program interface communication) and CPI-RR (common program interface resource recovery) interface.
- **Peer-to-peer communication (PPC)** gateway provides a context bridge between TCP/IP and SNA networks allowing LU6.2 'sync level syncpoint (synclevel2)' communication.
- **DE-Light** is a light-weight implementation of TXSeries which consists of three components: - the c client component, which runs on Windows

- the JavaBeans component which runs on any JDK 1.1-enabled browser
- the gateway, which runs on Solaris, AIX and HP-UX.

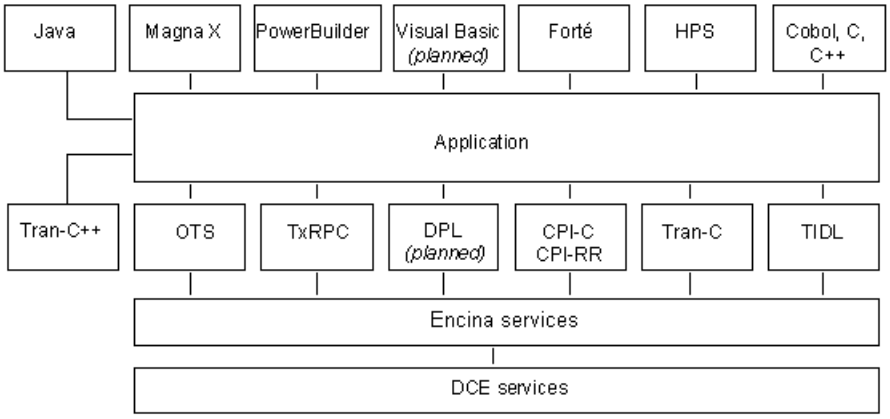

Figure 13. TXSeries interfaces

4.2.4 Web Integration

For DE-Light exists a **DE-Light Web client** that enables any Web browser supporting Java to access TXSeries and DCE services. DE-Light Web client does not need DCE or TXSeries on the client. It is implemented as a set of Java classes, which are downloaded automatically from the Web server each time the browser accesses the Web page referencing the client. DE-Light Web also has minimal requirements for RAM and disk.

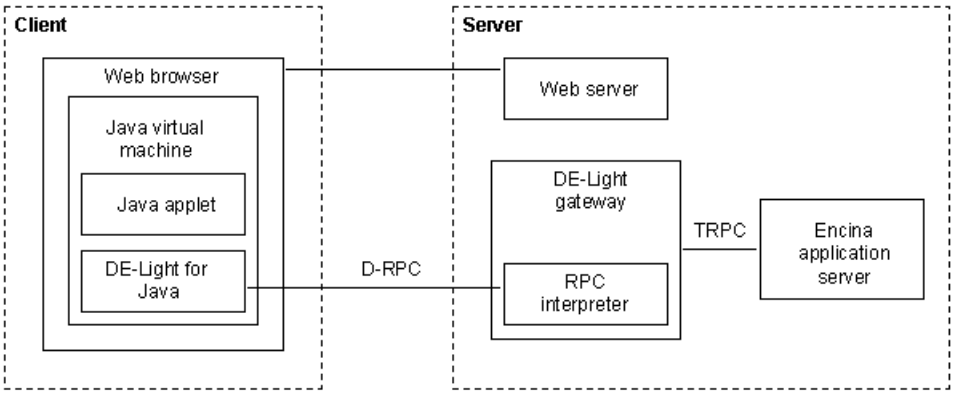

Figure 14. TXSeries Web Integration

4.2.5 When to use

You are already using or will be happy to use

- Your programmers are familiar with C, C++ or Corba OTS
- You use Oracle, DB2/6000, MS SQL Server, Sybase, CA-Ingres, Informix, ISAM-XA, MQSeries and/or any LU6.2-based mainframe transaction and you

need to build transaction processing systems that update or inter-operate with all these

- DBMSs/resource managers concurrently.
- You need to build applications that enable users to perform transactions or access files over the Internet - all Transarc's Web products are well thought out and useful.

4.2.6 Future plans

Future development plans include:

- Integration with Tivoli's TME, the systems management software from IBM's subsidiary
- Integration with Lotus Notes (available as sample code today)
- Enhancement of the DE-Light product (Full JavaBeans client support is currently available There are plans to provide an Enterprise JavaBeans environment later this year, which will be integrated with TXSeries in the future
- TXSeries currently provides Corba 2.0 OTS and OCCS services. These work with IONA's ORB today and there are plans to support other ORBS - such as IBM's Component Broker in the near future.
- Available since 4Q97, IBM provides direct TXSeries links to IMS, which will enable IMS transactions to become part of a transaction controlled using two-phase commit and specified using TxRPC
- Transarc will provide tools for automatic generation of COM objects from the TXSeries IDL in our next release. Today, integration with tools like Power Builder, Delphi, etc is achieved through calls to DLL libraries.
- Support for broadcast and multi-cast communication.

4.2.7 Pricing

Product	Price
TXSeries Server	$ 3,600
TXSeries Registered User	$ 80
TXSeries Unlimited Use (mid-tier)	$ 16,500

4.3 IBM's CICS

4.3.1 Summary

Key Points

- Supports NetBIOS (CICS clients only), TCP/IP (except the ESA, AS/400 and VSE platforms) and SNA (all platforms)

- CICS servers run on AIX, HP-UX, Solaris, Digital Unix, Sinix, OS/2, Windows NT, ESA, MVS, VSE and OS/400
- CICS clients run on AIX, OS/2, DOS, Apple Macintosh, Windows 3, Windows 95 and Windows NT

Strengths

- Development environment (API, tools, languages supported) and communication options for programmer well developed
- Context bridging and message routing support provided to bridge SNA and TCP/IP environments
- Worldwide IBM support, training and consultancy available

Weaknesses

- Directory services are weak and likely to need high administrator input to set up and maintain
- Central cross-platform administration of the environment is weak
- Underlying security services are not provided in a uniform way across all the environments and lack encryption facilities

4.3.2 History

Over the years IBM has produced three transaction monitors - CICS (Customer Information Control System), IMS (Information Management System) and TPF (Transaction Processing Facility), but CICS has become the dominant one.

During the 1960s IBM set up a team of six people at IBM Des Plaines to develop a common software architecture on the System 360 operation system what became CICS. The product was initially announced as Public Utility Customer Information Control System (PUCICS) in 1968. It was re-announced the following year at the time of full availability as CICS. In 1970, the CICS team moved to Palo Alto, California.

In 1974, CICS product development was moved to IBM's Hursley Laboratory in the UK. During the late 1970s and early 1980s, support for distributed processing was added to CICS. The main functions added were: transaction routing, function shipping and distributed transaction processing. Support for the co-ordination of other resource managers as part of a CICS transaction was introduced in the late 1970s.

The first version on Unix to be released was CICS/6000, based on Encina from Transarc and DCE. Only parts of the Encina product and DCE were used: IBM estimates that between 40% and 50% of the resulting code base is new code, the remainder being Encina and DCE.

The Windows NT version of CICS was originally based on the OS/2 code base, but is currently being transferred to the Encina code base.

The CICS development is now placed at IBM's Global Network Division.

4.3.3 Architecture

CICS comprises a **CICS core**, **application servers** and **listener services**. The **listener services** handle the communications links on the network, apply any security

and collect buffers received from the network software, breaking the buffers down into their components.

Each instance of CICS has a **schedule queue**: a shared memory queue, invisible to the programs but used within CICS to handle the scheduling of requests. As requests are received by the listeners, they place the requests on the schedule queue. CICS does not have a 'transmission queue' on the sending end to store requests.

One CICS instance on a machine (there can be more than one CICS instance on mainframes) can handle the messages or requests destined for all the programs, CICS files, or CICS transient data queues which are using that CICS instance. Thus, the scheduling queue will contain the request and message from multiple programs and destined for multiple programs, files or queues.

The CICS instance can also contain one or more **application servers**. These components handle the dequeuing of messages and requests from the scheduling queue and the transfer of these messages to the appropriate program, file or queue. An application server is not 'allocated' to a specific application program, file system or transient queue. Each application server simply takes whatever happens to be the next message on the schedule queue and then processes it.

In essence, the scheduling queue is organised in first-in-first-out order. CICS supports prioritisation of messages on the ESA machine, but this is only used to despatch messages, not to process them when a message is received.

Once the application server has taken the message off the queue, it will load the program where necessary and then wait until the program has completed its activity, the action has been completed by the file system or the message has been placed on the transient queue. Once the program has finished processing, the application server will clean up any data in memory and then go back to see if there are any messages on the schedule queue for it to process.

Once there are more messages/requests on the schedule queue than a specified limit (normally ten), new application servers are spawned automatically. When there are fewer messages than the limit, application servers are automatically taken out of service.

Further components include:

- **directory service**
- **CICS file system** (a record-based file-handling system)
- **temporary storage** (memory-based or disk-based queue)
- **transient data** (sequential files outside the file control functional area, which can be shared between transactions/tasks)
- **memory-based semaphore** (CICS-controlled area of memory, which acts like a semaphore - not under transaction control)
- **shared temporary storage**.

