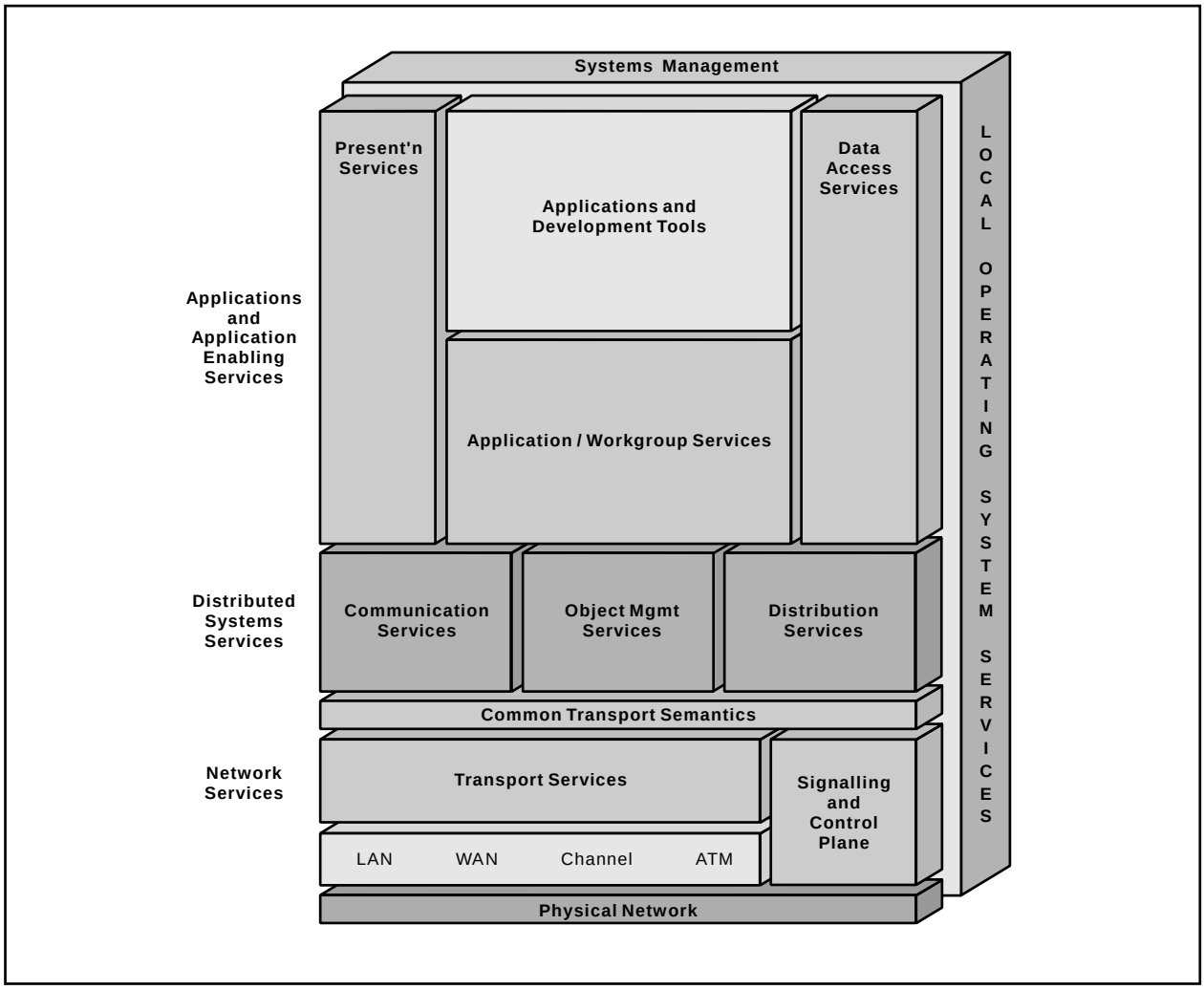
Open Blueprint

Transaction Monitor Resource Manager



GC23-3931-01

Open Blueprint

Transaction Monitor Resource Manager

GC23-3931-01

About This Paper

Open, distributed computing of all forms, including client/server and network computing, is the model that is driving the rapid evolution of information technology today. The Open Blueprint structure is IBM's industry-leading architectural framework for distributed computing in a multivendor, heterogeneous environment. This paper describes the Transaction Monitor resource manager component of the Open Blueprint and its relationships with other Open Blueprint components.

The Open Blueprint structure continues to accommodate advances in technology and incorporate emerging standards and protocols as information technology needs and capabilities evolve. For example, the structure now incorporates digital library, object-oriented and mobile technologies, and support for internet-enabled applications. Thus, this document is a snapshot at a particular point in time. The Open Blueprint structure will continue to evolve as new technologies emerge.

This paper is one in a series of papers available in the *Open Blueprint Technical Reference Library* collection, SBOF-8702 (hardcopy) or SK2T-2478 (CD-ROM). The intent of this technical library is to provide detailed information about each Open Blueprint component. The authors of these papers are the developers and designers directly responsible for the components, so you might observe differences in style, scope, and format between this paper and others.

Readers who are less familiar with a particular component can refer to the referenced materials to gain basic background knowledge not included in the papers. For a general technical overview of the Open Blueprint, see the *Open Blueprint Technical Overview*, GC23-3808.

Who Should Read This Paper

This paper is intended for audiences requiring technical detail about the Transaction Monitor Resource Manager in the Open Blueprint. These include:

- Customers who are planning technology or architecture investments
- Software vendors who are developing products to interoperate with other products that support the Open Blueprint
- Consultants and service providers who offer integration services to customers

© **Copyright International Business Machines Corporation 1995, 1996. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 1995, 1996

Summary of Changes

This revision includes:

- A description of how relationships between the Transaction Monitor resource manager and other Open Blueprint resource managers support network computing
- A description of advanced transaction models

© Copyright IBM Corp. 1995, 1996

Open Blueprint Transaction Monitor Resource Manager

This paper describes both the attributes and the features of the Transaction Monitor resource manager.

Introduction

Transaction processing (TP) and transaction monitors represent an important part of general client/server computing, offering a set of facilities that are indispensable for building commercial-strength applications. As client/server applications mature and become the basis for mission-critical solutions, it becomes apparent that for such solutions to be effective, a component is required in the system that manages the interactions between all the participants (sometimes likened to the software equivalent of the symphony orchestra's conductor). This is the traditional role of the online transaction processing (OLTP) monitor, in which transactions, rather than being mere business events, represent a philosophy of application design that guarantees robustness in a distributed system. This philosophy has been summed up as follows:

> *"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."*[1]

In the Open Blueprint structure, transaction processing consists of two sets of function. The Transaction Manager resource manager, one of the Open Blueprint Distributed Systems Services, represents the core of the transaction processing environment. The Transaction Manager resource manager provides services that provide a foundation for distributed transaction processing applications by providing functions that:

- Delineate a group of operations as a transaction
- Record a transaction's transactional state
- Complete the work in it's entirety or discarding all the operations
- Control resource usage on a transactional basis
- Facilitate recovery of the system back to a known state following a failure

The Transaction Manager resource manager is described more fully in the *Open Blueprint Transaction Manager Resource Manager* component description paper.

The Open Blueprint also defines an application service called the Transaction Monitor resource manager. A *transaction monitor* is that system software (or middleware) that provides end user (or client) applications with access to a variety of application and system services (such middleware is often referred to as an *application server*). Additionally, a transaction monitor environment provides a broad set of the elements necessary to create, manage, and run user programs. Transaction monitor software specifically addresses aspects of program execution, security, system management, and transactional and service integrity. From a user's perspective, today's transaction processing monitors (such as CICS, IMS, AS/400, and Encina) are characterized by a number of attributes which can be conveniently grouped into six main categories that represent the integration components of a complete transaction processing system:

- System and Data Integrity Services
- Presentation Services
- Communications Services
- Data Access
- System/Program Services
- System Management services

© Copyright IBM Corp. 1995, 1996

Attributes of the Transaction Monitor Resource Manager

The Transaction Monitor resource manager is extremely well-suited to handling client/server applications in complex environments for the following reasons:

- **Integrity**

  The Transaction Monitor resource manager provides a robust, general-purpose environment which enforces the Atomicity, Consistency, Isolation, and Durability (ACID) discipline without requiring any complex application code to be written. It achieves this by using Transaction Manager resource manager functions to ensure the integrity of distributed resources.

- **Robustness**

  The ACID principles that underlie the Transaction Monitor resource manager provide appropriate levels of protection between applications and resource managers and between applications themselves.

  The Transaction Monitor also provides an environment in which concurrent applications (or instances of an application) are isolated and protected from one another and in which the business logic is usually separated from both client processing (presentation services) and resource processing (data access).

  Individual parts of an application can be tightly-coupled using a two-phase commit protocol or loosely-coupled using transactional queues.

- **High Performance**

  The Transaction Monitor resource manager provides a “pre-warmed” environment by having resources pre-allocated, exploiting early binding to resource managers and other optimizations. A pool of pre-loaded application servers not only reduces overall system resource requirements but avoids the overhead associated with starting up a new process for each client request.

  Another key strength of the Transaction Monitor resource manager is its ability to provide efficient use of (often limited) resources through superior process management and support for both static and dynamic load balancing. Requests can be prioritized and server processes can be replicated as required, either on the same server node or on different nodes. These facilities are especially important for systems that run on SMP hardware.

  Facilities are also provided to allow priority scheduling to be given to differing classes of work.

- **High and Continuous Availability**

  The Transaction Monitor resource manager operates within many different failure scenarios. Because the monitor is aware at all times of the current state of all client/server resources under its control, the point of failure can always be detected and failed processes restarted as required.

- **Security**

  The Transaction Monitor resource manager extends the functions of the Security resource managers by providing role-based authorization models. Access to resources can be controlled with reference to any combination of the user identity requesting the access, the type of action being requested (transaction-based security), the system or terminal from which the request has been initiated or the time of day.

- **Scalability**

  The discipline of developing applications for the transaction monitor environment is one which leads to the development of modular procedures that separate the function required from the data to be processed. As more functions are added, the Transaction Monitor resource manager is able to distribute that function over multiple servers. Enforcement of the ACID principles ensures that disparate functions work together in a consistent way.

In addition, the Transaction Monitor provides an easy way to mix differing resource managers in a heterogeneous environment. Coordination across the resource managers is managed by the Transaction Monitor resource manager itself. This allows the environment to grow without requiring any alterations to the existing applications or application architecture.

Thus, the Transaction Monitor resource manager provides a ready-built framework for building, running and administering a distributed application. This framework provides an excellent basis for rapid development of client/server applications when complemented with the appropriate set of graphical user interface (GUI) tools.

The combined effect is that the Transaction Monitor resource manager provides significant cost benefits, not only in providing a running start for application development but also by funneling a large number of clients into a small number of server processes, by reducing access costs to resource managers.

Transaction Monitor Resource Manager Structure

As shown in Figure 1 below, the Transaction Monitor resource manager is built on a client/server architecture, with clients and servers implemented as separate processes, each within the local operating system environment. A client can either run on the same system as the server or run remotely.

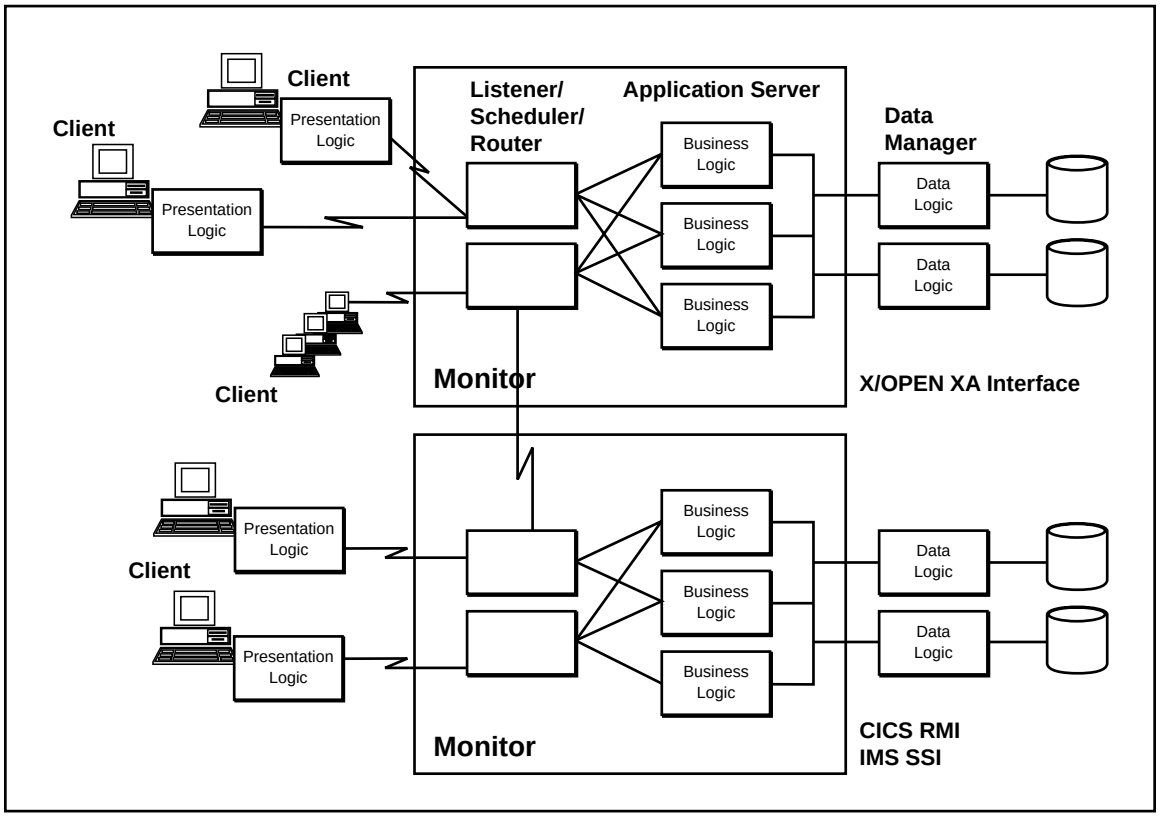


Figure 1. Transaction Monitor Structure

Most clients are concerned with presentation management, for example when the client is a remote application. In some configurations, elements of business logic can also be run on the client.

Transaction requests reach a server from a client and are caught by a front-end process that routes the requests to an application server process, placing them on a priority scheduling queue. Each application server (usually running in its own separate operating system process) takes the transaction requests from

the scheduling queue and executes the requests in priority order, causing the appropriate customer application to run. The number of processes (application servers) is controllable and will usually be far fewer than the number of clients. A collection of such application servers is often called a *region*. More than one region can be run on a system at any one time. An application server process runs only one application at a time but is capable of running any of the application types defined to the region. When the application terminates, the application server process is reused by the next application. Requests are passed from the clients to the application server by a scheduler function within the Transaction Monitor resource manager.

During system initialization, a number of application servers are created. At peak times, more servers can be created, up to a configurable limit. Then, as the workload eases, excess servers are terminated. The minimum and maximum number of application servers is controllable by the system administrator. Predictable response time is a key requirement for OLTP systems. The ability to control the number of applications running at any one time enables the system to be responsive to increased demand without allowing a large number of concurrent requests to flood the system and slow response time. This can be managed automatically by the Transaction Monitor resource manager itself.

Transaction Monitor Resource Manager Features

This section describes the various features of the Transaction Monitor resource manager in more detail. Some of these features are provided directly by the Transaction Monitor resource manager itself, but in many cases the Transaction Monitor resource manager simply provides access to the underlying Open Blueprint services from within its environment.

In traditional implementations, transaction monitor environments are usually provided by powerful, single, well-integrated products such as CICS, IMS, and AS/400. Each provides its own set of interfaces, such as the EXEC CICS API.

As distributed computing evolves to encompass a more heterogeneous environment, some of the functions traditionally associated with the transaction monitor environment will be provided as an integral part of the base operating system. Therefore, applications will be able to use more of the standard application programming interfaces (APIs) in addition to the specialized interfaces used today. Examples of this include the Encina programming model, which extends the standard Distributed Computing Environment (DCE) interfaces as defined by the Open Software Foundation and encompass the X/Open transactional interfaces (TX, XA, XA+ and CPI-C) and the distributed object environment, in which some of the transaction monitor function is incorporated into the Object Request Broker.

Transaction Manager Resource Manager Support

The Transaction Monitor resource manager provides access to the Transaction Manager resource manager services (described in the *Open Blueprint Technical Overview* and related white papers), either implicitly, as in the chained transaction model[2] implemented by CICS or IMS, or explicitly through direct access to the Transaction Manager resource manager interfaces. Thus, the Transaction Monitor resource manager allows an application to delineate a group of operations as a transaction, record the transactional state, and complete the work in its entirety or discard all the operations, to control resource usage on a transactional basis and to facilitate recovery of the system back to a known state following a failure.

More detailed information on the support required to assist applications with their recovery when a system restarts after a system failure is discussed in “Recovery Services” on page 11.

Presentation Support

Presentation support provides functions for data handling and validation capability. Examples include CICS Basic Mapping Services (CICS/BMS), IMS MFS, AIX with Motif/AIC/Widgets or GUI builders. Additionally, expanded client-server models such as intelligent agents include additional processing at the client on behalf of the user (for example, rules and learning).

Presentation support is designed to free an application from those tasks associated with gathering and displaying information to and from the client (for example, a terminal). Presentation support provides device independence by creating a logical rather than a physical form description with which an application interacts. Presentation support acts on the logical form description and reads/writes to the physical device. Examples include the presentation support's use of X-Windows, Motif, CICS/BMS, or IMS Message Formatting Services.

Associated with the presentation support is a screen generation tool that allows the developer to specify an application's display or screen layout function (for either GUI-based or character-based environments). These display generation facilities allow specification of standard components for GUI-based applications such as data entry areas, labels, boxes, buttons (radio and push), lists (drop-down, pop-up), tables and similarly character-based attributes such as fields (position, appearance, contents, and behavior). The screen generation facility also provides services to assist the developer in forms construction. These services are basic templates to build new forms, an editor to modify existing forms and debug facilities to insure proper operation. The facilities can also be provided through such tools as IBM's VisualAge, Powersoft's PowerBuilder, Microsoft's Visual Basic and Magna's Magna X.

The base functions will be provided by the Human Computer Interaction resource manager. In the transaction monitor environment, these can be enhanced with additional support for forms processing within the applications and transparent support of non-programmable terminals.

Communications Support and Interoperability

Communications support frees the application from those tasks associated with exchanging data between client and server and between applications. They provide transparent access between the client and the application through communications, directory and security functions. Applications (or servers) advertise the availability of a particular program (or interface) with the Directory resource manager. After a user (or client) request is issued, the directory transparently locates an available program and hides any distribution detail associated with the connection. Associated with the user request are security services to both authenticate the user's validity to access the program and to provide an authenticated connection between user and application.

The three common distributed computing programming models are all supported: conversational, remote procedure call, and messaging. Within the function are facilities to transparently transfer message exchanges across any number of transport protocols and to handle data representation conversions.

Interoperability with existing systems requires that conversion of distributed computing protocols (for example, IMS/ISC or CICS/ISC) be supported through gateways provided at the boundary of the operating environment. Additionally, mapping is required to accommodate interoperability between communication models (that is, between the contextual or connection-oriented conversations and independent or datagram-oriented request/response models). Historically, gateways have been constructed that address all aspects of interoperability (transaction, protocol, and model) in a unified implementation.

Data Access Support

Within the transaction monitor environment, interfaces are provided between the application and stored data. These interfaces allow the data to be read or updated, while preventing unauthorized access and protecting the data from corruption. The Transaction Monitor resource manager supports recoverable usage of a variety of database, record-oriented and byte-oriented file systems, and Persistence resource manager support.

Some systems provide functions for handling queues either to provide such services as time-independent processing (MQSeries, IMS Queue Manager, Encina RQS) or to use as transactional storage areas (CICS).

Data access support is designed to free the application from those tasks associated with data definition and data manipulation. This support delivers yet another degree of data independence by providing logical data representation. Additionally, standards have been defined for data access operations (such as SQL and ISAM). The X/Open XA interface, as supported by the Transaction Manager resource manager, is used to ensure the integrity of resources.

The Transaction Monitor resource manager traditionally provides value-add to the existing data access facility by defining recoverable services that are both:

- Permanent (exist for other programs to use)
- Temporary (only exist for the duration of the current program or transaction).

Additionally, the Transaction Monitor resource manager can provide any missing data manipulation functions, as it does with CICS File Control (variable length, relative access, or generic key). It also provides the convenience of common administration and common security across a variety of resource managers.

Application Management Support

Facilities are provided for efficient balancing of workload and efficient use of available (and possibly limited) resources to guarantee a consistent user response time. These facilities are:

- **Scheduling**

  Although the local operating services provide basic facilities for task or program scheduling and execution, a monitor provides a value add through the introduction of prioritization, triggering, and resource sharing and by extending the core process and program definition to include a specification level for user, application, transaction or service.

  The operating system mechanisms for task and program scheduling include function for:

  – Transparent assignment of a process context to an execution context (by reusing processes or threads as needed)

  – The concept of shared libraries that allows many applications to reuse a copy of the program

  – The facility to assign fixed priorities to a process

  – The ability to either create a process beforehand or upon request

  – Concurrency service for synchronization of resource access

  The Transaction Monitor resource manager provides additional value by extending the scheduling functions to include the following:

  – Priority levels on an application, transaction, or service basis

  – Scheduling functions to provide program trigger on defined thresholds

  - Synchronization services at a resource level to allow programs to execute as the required total resource becomes available for the required execution environment

- **Workload Management**

  Workload management covers two areas:

  - Load balancing: assures the workload is assigned equally across the system
  - Load sharing: assures the workload is executed equally across the system

  The Transaction Monitor resource manager provides facilities to support workload management for transaction processing workloads, both within a single system and across multiple distributed systems.

- **Event Services**

  Although the system provides basic services and frameworks for event management, an event service provides additional value through the introduction of application execution based on an event such as time of day, time interval, transaction completion, or resource availability.

  The system service mechanisms for event management include core function to define the basic timer event. The monitor service provides additional value by defining higher-level processing of the timer event through a variety of time-dependent processing options (such as transactions executed at a particular time of day or after an elapsed period of time).

Some transaction processing systems provide their own implementation of program service functions and functions to facilitate and manage the sharing of resources. Examples include CICS task control, program control, storage control and timer service; or the IMS storage manager, dispatcher, and scheduler.

Systems Management Support

System management services are designed to assist the administrator by supplying a graphical user interface which provides both a single system image and single point of control for those tasks associated with system definition, control, and maintenance. The Transaction Monitor resource manager provides appropriate definitions and support to allow its entities and workload to be managed within the Open Blueprint Systems Management structure.

- **Configuration**

  Minimizes user involvement by automating definition of required user, application and system services (and defining any relationships and operational environment). Provisions are made to ensure an administrator is directed to the task at hand and to reduce or eliminate the need to supply definition for the supporting system (for example, installation of a CICS/6000 system should be insulated from any detail associated with Encina, DCE and AIX). Additionally, the configuration service needs to ensure that only the minimal amount of parameters need specification for a particular system definition by supplying as much default and generic definition as possible. Configuration services also need to discover changes in the systems and automatically create and destroy definitions as appropriate.

- **Update**

  Provides a mechanism for distribution of change of user, application and system service definition, or software level. The update service ensures that the system has both the latest system definition and the latest software (version, function, and fixes) by providing the mechanism to distribute changes.

- **Backup**

  Provides a facility that supplies information the user might require to facilitate reconstruction of events or data changes caused by an application. Usually, the backup data is stored in a recoverable file that can be used as an audit trail, a list of transactions on the system, or a record of modifications to the system services. The backup data can be used for additional purposes such as restart or recovery operations. Examples of backup services include IMS logging and archiving, CICS journaling, Encina logging, and AIX error logging.

Many of the backup services are provided by the underlying local system facilities. System reliability, availability, and serviceability (RAS) and monitoring facilities usually provide the necessary function to perform general audit services.

The Transaction Monitor resource manager provides a general logging service to store data that the user, application, or services specifies as recoverable. Usually, the Open Blueprint Data Access Services incorporate such services within their specific product implementation. Such a service is designed to store larger quantities of data, maintain the data for longer duration, and provide more degrees of data selection (and is often required to provide backup storage mechanisms to retain the data). The Transaction Monitor resource manager provides a transaction log service; however, the recoverable data requirements are unique and distinct from a generalized logging service.

- **Error Detection**

  Provides services for the collection of event data requiring some system intervention. These services are useful for problem diagnosis, correction, and notification (of any interested party, such as NetView).

  The system RAS services provide for an integrated data repository for all events critical to the operation of the system. Sufficient event information is gathered to allow subsequent problem diagnosis and guarantee problem resolution (in other words, any occurrence critical to the operation of the system or malfunction or error is recorded and provides a description, probable cause and necessary corrective actions). RAS services include specific system facilities for trace, error logging, and system dump. Common utilities provide uniform data reports for analysis of this gathered data in relationship to overall system activity or on a more granular basis by specific component within the system.

- **Monitoring**

  A collection of operational data to allow system-controlled tuning. Such services are useful for ensuring reliability and availability, performance management, and accounting.

  Reliability and availability services provide for continued operations in the event of a failure. The requirements placed on the system are for the following:

  – Monitoring (or a heartbeat function) to provide notification of an outage and failure

  – Configuration services to define the relationships and operating environment of user, application, and system services (as defined previously)

  – Data and system integrity services to provide global synchronization of resource access by the user, application and system services (as defined previously)

  When an application or service fails, work can be recovered by:

  – Restarting the failing process

  – Assigning a backup that provides similar function (generally operating somewhere else in the system)

  Restart and recovery services are addressed in “Recovery Services” on page 11.

  Additionally, fault tolerant environments are specifically designed to provide continuous operation and minimize disruption by providing *failover* processing, which uses a designated backup and automated switchover procedures.

  Performance management allows statistical information to be gathered on a user, application, service, or transaction usage basis. User-defined performance data is written to a system facility for analysis that is either specific to the transaction or in relationship to other system activity. The system can then tune the application execution environment based on the statistical information to achieve maximum throughput, guaranteed response times or maximum utilization of resources. Additionally, the execution environment can be reconfigured among other systems to ensure specific execution criteria.

Accounting is also provided by user, application, service, or transaction usage.

- **Security**

  Authentication ensures a validated and secure connection between the requester and service provider.

  Authorization allows the definition and enforcement of access levels by requester for usage of a service provider. For transaction processing environments it is generally desirable to have a single authorization step for granting access into the system. Additionally, based on the operating environment and conditions, authorization levels can be established by operating domain, user role, level of permission or, time of day criteria.

  The Transaction Monitor resource manager builds on the Security resource managers by providing these additional parameters (such as transaction-oriented security or authorization based on the identity of the initiating terminal).

Recovery Services

Restart involves creating a backup after detection of a failure. The configuration services create the backup in an operational processor (which is either the same or different from where the failure occurred and depends on the failure type). The backup is responsible for re-creating the execution environment from the point of failure from pertinent information retained in permanent storage. This information is used for automatic replay of in-flight transactions following such a failure.

Failover involves creation of an execution environment that consists of a primary process and a backup process, each running on separate processors (that is, a process pair). The basic definition of a process pair states that the primary process executes while the backup process remains quiescent. However, as long as primary and secondary processes perform different tasks, both can actively perform work. That is, they can provide failover support for one another. The primary and backup processes maintain synchronization by maintaining pertinent context information in a stable and shared storage area. In the event of failure the system need not notify the associated user, application, or service of the failure, because requests are automatically retargeted to the functional process.

If a system terminates before all transactions have completed normally, when the system is started again, the recoverable resources of that system must be restored to a known consistent state before normal processing can resume. This consistent state should reflect the changes made to resources by transactions which committed during the last execution of the system. The logic for transactional recovery is similar to the following:

- Restart the system (by starting the Transaction Manager and other resource managers)
- Reconstruct the Transaction Manager resource manager state (the transaction outcome)
- Restart the resource managers
- Exchange transaction outcome
- Relay log information to restore resource manager data

The specific procedures followed depend on the environment in which the application that comprises the transactions was running, so the procedures are different for object-based transactions, for CICS-based transactions, or for IMS-based transactions, for example. However, any recovery protocol mechanism performs the following actions:

- Determine the transactions that need to be undone. Replay the log information for transactions that are incomplete. Incomplete transactions are transactions that have a record of BEGIN_TRANSACTION but no END_TRANSACTION.
- Determine the transactions that need to be redone. Replay the log information for transactions that are complete. Complete transactions are transactions that have a record of a BEGIN_TRANSACTION and END_TRANSACTION.

In addition, a *recovery checkpoint* facility can be implemented to minimize the amount of time required for recovery. Checkpointing forces a copy of the current state to permanent storage (at some specified time interval) so that it only becomes necessary to replay the log information from the last checkpoint rather than the entire log.

Specific requirements include:

- Log processing at restart (to restore state)
- Implicit recovery (warm start)
- REDO/UNDO
- Sharable
- Explicit recovery (heuristic)
- Data and operation logging

A recovery manager can be designed to support restart recovery processing in each and all of the transaction management environments. It does not provide a recovery protocol but does provide logic to simplify the implementation of a recovery protocol by providing the following basic recovery primitives:

- **Recovery Analysis**. Constructs a list of transaction status.
- **Recovery Processing Assistance.** Assists data retrieval by providing log scan operations that return information associated with a given transaction or server (or both).
- **Recovery Checkpoint.** Forces a record (containing the Transaction Manager or other resource manager state) to stable storage allowing recovery processing time to be minimized.
- **Restart Processing.** System facilities to restart resource managers and associated initialization environment.
- **Recovery Interface.** Minimally an indication by the resource manager that it has restarted and is ready to begin transaction state recovery.
- **Resynchronization.** Capability to facilitate recovery by passing all outstanding transaction recovery status between Transaction Manager and resource manager on a single operation (that is, defining a resynchronization protocol).

The Transaction Monitor resource manager builds on the basic recovery primitives by providing a complete implementation of a recovery protocol and the necessary logic to ensure that recovery is driven at system restart.

Relationships to Other Resource Managers

There are many ways of bringing together the various Open Blueprint resource managers to meet the needs of real commercial processing systems. This section describes the ways in which the Transaction Monitor resource manager is being deployed in conjunction with the Workflow, Collaboration, and HTTP resource managers.

An Extended Model of Transaction Processing

Increasingly, collaboration and workflow systems are being deployed as front office components linked to specific back office systems that have long been automated. Hence the linkage between the front office components and the more classical transaction processing system is critically important. A typical workflow management system will provide functions which help to define, execute and re-engineer business processes, often across a heterogeneous system environment. The workflow manager is a coordinating agent which initiates the execution of work by end-users and the execution of programs in multiple, distributed systems. It can be likened to a river which carries the flow of work from port to port with extra value being added along the way. A workflow management system allows process (flow-of-control) definitions to be separated from the applications.

There is some overlap between traditional transaction monitor systems and modern workflow management systems. However, the current workflow models (and groupware in general) are not transaction oriented in the ACID sense; the systems are very good at reflecting the changing states of information over time but not as good at reflecting the current state of the data in real-time. In situations where this is an important requirement, there seems to be significant scope for combining the two technologies to infuse workflow with ACID properties.

A purchase processing example might be a helpful illustration.

A typical purchasing system begins with an approved purchase order requisition, which produces an official purchase order. These systems are typically extremely well automated, and most of the cost reductions to be gained from automation have been realized. The input to the system is an approved purchase order requisition. However, an approved purchase order requisition is typically the result of a process that begins when a person decides that they need to purchase something. The person creates a purchase order, and this requisition threads its way through the system until it is approved or rejected. When it is approved, the transaction (in the classic sense) is initiated. However, in reality a business transaction was initiated when someone created a purchase order requisition. Increasingly, particularly as enterprises re-engineer their business processes or establish some level of formalization of them, back-office systems have front-office business processing components.

A term that has been adopted to refer to this broader and more business oriented view of a transaction is that of the *Extended Transaction Model* (ETM). By tightly linking front office business processes to existing back office processes, enterprises are able to achieve significant productivity and economic gains. To achieve this tight linkage and integration, enabling frameworks, tools, and guidelines need to be provided and exploited allowing interaction with existing enterprise transaction systems in a controlled and architected way. By viewing front office processes as a part of some extended transaction, it becomes clear that the key to developing rich collaboration applications is an application development/deployment environment that is part of (integrated with) the overall business processing system.

There are many different approaches to the solution of these problems. but in the context of IBM's Open Blueprint, we are interested in three examples in this section.

© Copyright IBM Corp. 1995, 1996

Integrating with the Collaboration Resource Manager

The purpose of the Extended Transaction Model implementation is to enable applications that are built in a Collaboration resource manager environment to access and process data or transactions which reside in an enterprise system.

Why do customers want to do this?

There are two principal reasons. First, all major enterprises already maintain, manage, and process a significant part of their mission-critical data in such enterprise systems but want to build new systems and applications which can exploit Collaboration resource manager function while continuing to access and utilize the existing business applications and data. Second, the Collaboration and Transaction Monitor resource managers provide different functions, each suited to solving a particular aspect of a business problem. Many applications require the facilities provided by both environments. The extended transaction model defines a strongly integrated way of bringing these environments together to build a complete solution.

The integration of these two complementary environments is as seamless as possible, whether viewed from the perspective of an end-user, an application developer, or a system administrator. This seamlessness is most important from the end-user perspective and is achieved by ensuring that the entire user interface is built within the collaboration environment. The end-user interacts with the application through a series or sequence of collaboration documents, forms, or views.

The applications are built in such a way that whenever an interaction is required with a database or enterprise application it is performed transparently and the data presented to the user through the Collaboration resource manager.

Consider the application development scenario for the extended transaction model.. The process for developing applications is divided into two distinct parts: *construction* and *assembly*. Construction involves building well-defined and well-encapsulated pieces of business logic, usually within the context of an RDBM or OLTP system. Assembly involves building a complete business application by putting together a number of smaller constructed pieces; in the extended transaction model, this is in the context of the collaboration environment (though this is by no means the only possible solution). Construction and assembly can be undertaken by the same person or can be assigned to different skill groups. Construction is often performed using a third-generation programming language and environment; assembly is typically done using a scripting language or visual assembly tools. Almost all of the user-interface implementation falls into the domain of the assembly phase. In the Open Blueprint implementation of the extended transaction model, languages such as Cobol, PL/I, or C++ can be used for construction and LotusScript, in conjunction with the Notes document editor, can be used as the basis for assembly.

Access to the enterprise systems from the LotusScript environment is provided (transparently to both the application and the programmer) across a number of protocols. These protocols can include the MQSeries protocols and an optimized route to existing CICS 3270 applications using the CICS InterSystem Communication protocols. Other protocols such as Transactional-RPC could be added in the future,if required. The choice of protocol to be used in a given configuration will be driven by the customer's existing infrastructure, the overall network topology, and decisions regarding skill requirements (for administrative tasks) at particular network nodes.

Integrating with the Workflow Resource Manager

Another example of the extended transaction model is the way in which applications can be integrated across Transaction Monitor and Workflow resource managers.

Traditional transaction processing applications running under the control of the Transaction Monitor resource manager can be extended by incorporation. They can run as activities in a workflow process when built as applications using the Workflow resource manager. The Workflow resource manager also allows the transactional applications to initiate workflow processes.

A Workflow resource manager run time server uses a process model definition to control a process instance. Suppose an activity step in such an application wants to invoke an application that runs under the control of the Transaction Monitor resource manager. Because the Workflow resource manager uses data containers to pass work in progress from one activity step to another, the application developer must write a script what can interpret these data containers and convert the information they hold into a form appropriate to the transaction monitor application.

In a similar fashion, the Workflow resource manager can be invoked by a transaction monitor application to initiate a business process. That application must provide the workflow data container values that are required by the Workflow resource manager process.

For maximum flexibility, communication between the two resource manager environments can use the facilities of the Messaging and Queuing resource manager.

Integrating with the HTTP Resource Manager

The World Wide Web was originally conceived as a mechanism for providing widespread access to many documents. One of the key aspects of Web access was that the data was not centralized and was usually accessed in a read-only mode. By contrast, transaction processing systems are designed to provide a highly reliable means to read and update centrally managed resources.

However, there are some important similarities between Web servers (such as the HTTP resource manager) and transactional systems (such as the Transaction Monitor resource manager):

- Both use standard data streams between the client and server. The HTTP resource manager uses Hypertext Transfer Protocol (HTTP) and Hypertext Markup Language (HTML). The Transaction Monitor resource manager uses IBM 3270.
- In both cases, the application is usually running on the server, and the client is dedicated to presentation.

The pervasiveness of Web browsers makes the World Wide Web an attractive front-end environment for accessing existing or new transaction processing systems. The full function of the World Wide Web can be used to develop the presentation component of the application while the integrity of the data remains protected within the transaction processing environment.

There are many ways of building the interface between the HTTP resource manager and the Transaction Monitor resource manager.

An Internet gateway for the Transaction Monitor resource manager allows transaction monitor applications that have been written to communicate with a standard terminal to be invoked from and within any standard Web browser. Mapping between the 3270 datastream built by the Transaction Monitor resource manager and the HTML, which is interpreted by the Web browser, is performed automatically.

Implementations of Transaction Monitor Java clients will allow a more sophisticated Web interface to be built for existing transaction environments within an enterprise, and will allow client applications to be downloaded and executed as required.

---

1 Jim Gray, “Where is Transaction Processing Headed?,” *OTM Spectrum Reports* (May 1993)

2 In the chained transaction model, termination of one unit of work implicitly causes a new unit of work to be started, so an application is always running within the context of some particular unit of work.

Appendix A. Advanced Transaction Models

Although the traditional transactional model with its ACID properties is in widespread use throughout the industry for building robust distributed applications, there are some areas in which the model has severe limitations. Most notably, these include:

- Loss of work when a long-running transaction fails before commit
- Concurrency conflicts caused by long-running transactions
- Relationships between transactions (such as control flow dependencies) need to be handled by the application
- Collaboration among transactions is not supported
- There is no support for application-level parallelism
- Transactions are able to recover *data* but not *activities* (application state)

One solution to such problems has been the introduction of the nested transaction model. However many other, more exotic, transaction models are being introduced, each of which compromises the ACID properties in some way, in order to support:

- Long-lived activities
- Open-ended activities
- Cooperative activities
- Local autonomy
- Reactive activities (active data and triggers)

Examples of such applications include VLSI design, CAD/CAM projects, office automation, software development, heterogeneous database, or real-time applications.

Although there is much research into these areas and much published material there are many difficult problems in these areas and hardly any realistic implementations. In general it is expected that such solutions will either be built upon or delivered alongside traditional transaction processing monitors.

Some of the more interesting and important examples include Sagas, ConTracts, Parallel Transactions (with rendezvous points), Cooperative Transactions and Split and Join Transactions.

Sagas

Sagas represent an evolution of the solution to the problem of how to deal with long-running transactions, following in a fairly natural way from the concepts of savepoints and chained transactions.

A **saga** is a linear sequence of regular ACID transactions such that:

- When each component transaction completes, it exposes its results by releasing locks.
- For each component transaction, a compensating transaction is defined by the application writer with the property of logically undoing each of the component transaction updates

© Copyright IBM Corp. 1995, 1996

If a saga aborts part way through the sequence its effects may then be undone by running (in reverse sequence) the compensating transaction associated with each of the component transactions which have successfully completed. That is, for a saga that consists of the transaction sequence $T^1,T^2,...,T^n$, with compensating transactions $C^n,...,C^2,C^1$ respectively, then aborting at stage n results in the following sequence of transactions being executed:

$T^1,T^2,...,T^n,<<saga\ aborted<>,C^n,...,C^2,C^1$

If the system fails while a saga is in progress, forward recovery of the saga can be effected by executing the remaining transactions in the sequence (the interrupted one plus the following ones). Provision of support for this model of transaction processing would be provided through a system component known as the **Saga Manager**.

Cooperating Transactions

The *Cooperative Transaction* model is introduced to allow for explicit interactions between collaborating applications on shared objects. Examples of this are multi-user design systems where several users can require cooperative access to parts of a drawing, parts of a software design (or code) or parts of a document. At such a level of cooperation the traditional notions of atomicity are not powerful enough to be able to model the complex rules of state transitions that these applications will want to support.

To see how such a model works, assume that transaction T_1 is currently executing and modifying a design object X and that a second transaction T_2 wishes to explicitly request access to this same design object X. An implementation of the cooperative transaction model provides methods by which transaction T_1 can decide whether it wishes to suspend its own activities on X and give it to T_2, with two underlying conditions:

- Transaction T_2 must understand that object X is provisional and must not try to change it
- Transaction T_2 must eventually return object X to T_1 so that T_1 may resume its work on X

It is important to realize that this handing out of objects which have not yet been committed and requesting them back later is substantially different from the traditional dynamic data-induced dependencies which arise as side-effects of the parallel execution of transactions. Such data dependencies are monitored and handled inside the TP Monitor system and not exposed to the applications; however, in the case of cooperative transactions it is assumed that applications inherently know of and handle such dependencies.

Appendix B. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
AS/400
IBM
IBMLink
IMS
CICS
CICS/6000
MQSeries
Open Blueprint
Visual Age

The following terms are trademarks of other companies:

C++	American Telephone and Telegraph Company, Incorporated
DCE	The Open Software Foundation, Incorporated
Encina	Transarc Corporation
Java	Sun Microsystems, Incorporated
Motif	The Open Software Foundation, Incorporated
Open Software Foundation	The Open Software Foundation, Incorporated
Visual Basic	Microsoft Corporation
Windows	Microsoft Corporation
X-Windows	Massachusets Institute of Technology

Microsoft is a registered trademark of Microsoft Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries.

© Copyright IBM Corp. 1995, 1996

Appendix C. Communicating Your Comments to IBM

If you especially like or dislike anything about this paper, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply. Feel free to comment on specific error or omissions, accuracy, organization, subject matter, or completeness of this paper.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

- If you prefer to send comments by FAX, use this number:

  United States and Canada: 1-800-227-5088.

- If you prefer to send comments electronically, use one of these ID's:
  - Internet: **USIB2HPD@VNET.IBM.COM**
  - IBM Mail Exchange: **USIB2HPD at IBMMAIL**
  - IBMLink: **CIBMORCF at RALVM13**

Make sure to include the following in your note:

- Title of this paper
- Page number or topic to which your comment applies

© Copyright IBM Corp. 1995, 1996

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC23-3931-01