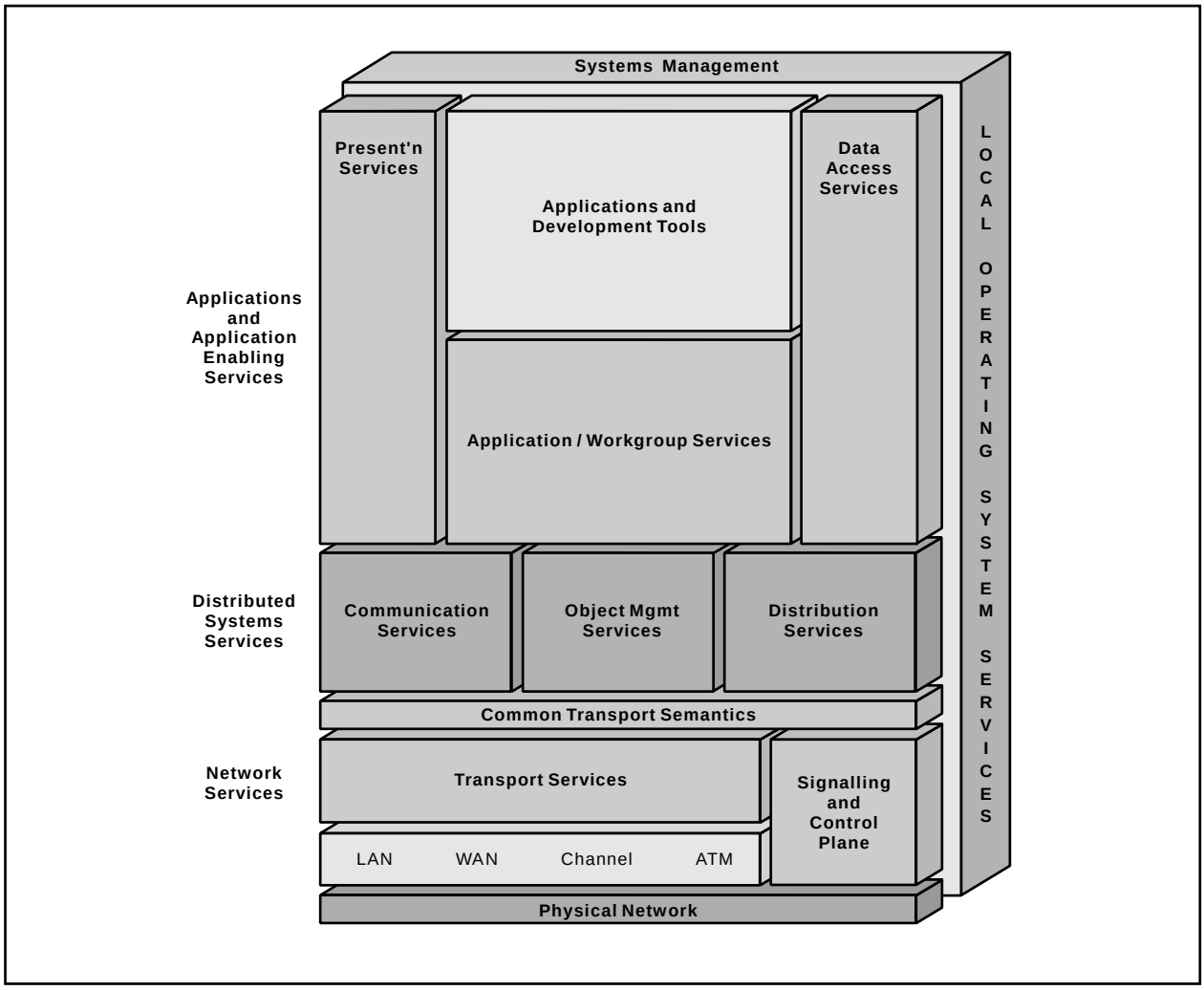
Open Blueprint

Transaction Manager
Resource Manager



GC23-3930-01

Open Blueprint

Transaction Manager Resource Manager

GC23-3930-01

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 Manager 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 Manager 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

Contents

Summary of Changes 1

The Transaction Manager Resource Manager 3
Introduction 3

Transaction Processing Overview 5

The Open Blueprint Transaction Manager Resource Manager 9
Logging of Transaction State 10

Transactional Interoperability 13
Interoperability Characteristics 13
Open Blueprint Capabilities 14

Notices 17
Trademarks 17

Communicating Your Comments to IBM 19

Figures

1. Distributed Transaction Processing Model 6
2. Distributed Client/Server Application Model 7
3. Transaction Manager Resource Manager Characteristics and Support 10

Summary of Changes

This revision includes additional information about transaction management.

The Transaction Manager Resource Manager

This paper describes the Open Blueprint Transaction Manager resource manager and its relationship to both the X/Open Distributed Transaction Processing (DTP) model and the Object Management Group (OMG) Transaction Processing Model.

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 functions. The Transaction Manager resource manager, one of the Open Blueprint Distribution 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 discard all operations
- Control resource usage on a transactional basis
- Facilitate the recovery of the system back to a known state following a failure

The principal sub-components of a Transaction Manager resource manager are:

- **Transaction Coordinator**. Supports functions for beginning, ending and aborting transactions
- **Transaction State Logging Service**. Provides functions for recording the state of executing transactions
- **Concurrency Services**. Allow multiple transactions to serialize access to shared resources

The Open Blueprint structure 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 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 Services

- System and Program Services
- System Management services

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

Transaction Processing Overview

Transaction processing environments evolved as system designers recognized a series of common requirements in the online systems they were building. One of the key features of online systems is that they ensure the transactional integrity of their data.

Transactional integrity is achieved by using a *transaction* (sometimes called a *logical unit of work*), which is a unit of computation that contains what are commonly referred to as the ACID properties. These properties are:

- **Atomicity**. Transactions are indivisible units of work; all of its actions either succeed or fail. Thus, programmers and users need not be concerned that a failure will leave resources in an inconsistent state.
- **Consistency**. After a transaction has executed, it must either leave the system in a correct state or abort. If the transaction is unable to reach a stable end state, it must return the system to its initial state. Thus, the integrity of the resources is maintained.
- **Isolation** (or serializability). Concurrent transactions must behave as if they were executed serially; that is, the behavior of one transaction will not be affected by another that happens to be executing concurrently. This means that a transaction must serialize all access to shared resources. The changes to shared resources that are made by one transaction must not become visible outside of the transaction unit until that transaction completes.
- **Durability** (or permanence). The effects of a transaction are permanent when that transaction is successfully completed (committed). These effects are not lost or undone as a result of subsequent system failures.

In a distributed system, the transaction becomes the fundamental unit of recovery, consistency and concurrency. Resources that are managed under transactional control are known as *recoverable resources*. The Transaction Manager resource manager provides synchronization services that allow multiple resource managers to act together to ensure that the resources they own retain their integrity.

A typical example is a two-part financial application that credits one account and debits another. If a failure occurs after the credit but before the debit, the application would want to back out the credit. The Transaction Manager resource manager interacts with the other resource managers that are involved in the credit and debit to ensure that either both actions complete or that the accounts remain unaltered. If the two accounts are located in different systems, the Transaction Manager resource managers in each system cooperate to eliminate any effects of a failed transaction.

A resource manager is responsible for ensuring the integrity of the resources that it owns; this responsibility includes recovery from physical or logical damage, backing out of incomplete changes, and retrying operations. Isolation is achieved by locking of shared resources for the duration of a transaction. Atomicity is achieved by using recovery techniques that include the logging of the transactional state and data and a commit protocol (generally a two-phase commit protocol).

The Transaction Manager resource manager's transaction coordinator provides the services necessary to coordinate the atomic completion of a distributed transaction (one that consists of operations on resources managed by many resource managers, possibly at several different locations). Resource managers that are part of a transaction register their interest with their local Transaction Manager resource managers. The two-phase commit protocol is used between a Transaction Manager resource manager and other resource managers or other transaction managers to request that the involved parties perform the following:

1. Prepare to commit the changes (the first phase, which tests each resource manager to make sure that a commit can be performed)
2. Complete the commitment of the changes (the second phase)

An instance of the Transaction Manager resource manager operates in each system. When activity takes place outside of a system, an Open Blueprint Communications resource manager is responsible for mediating between the Transaction Manager resource managers in each system. The Transaction Manager resource manager itself is not aware of the distribution or the type of communication.

This model for distributed transaction processing is illustrated in Figure 1 below.

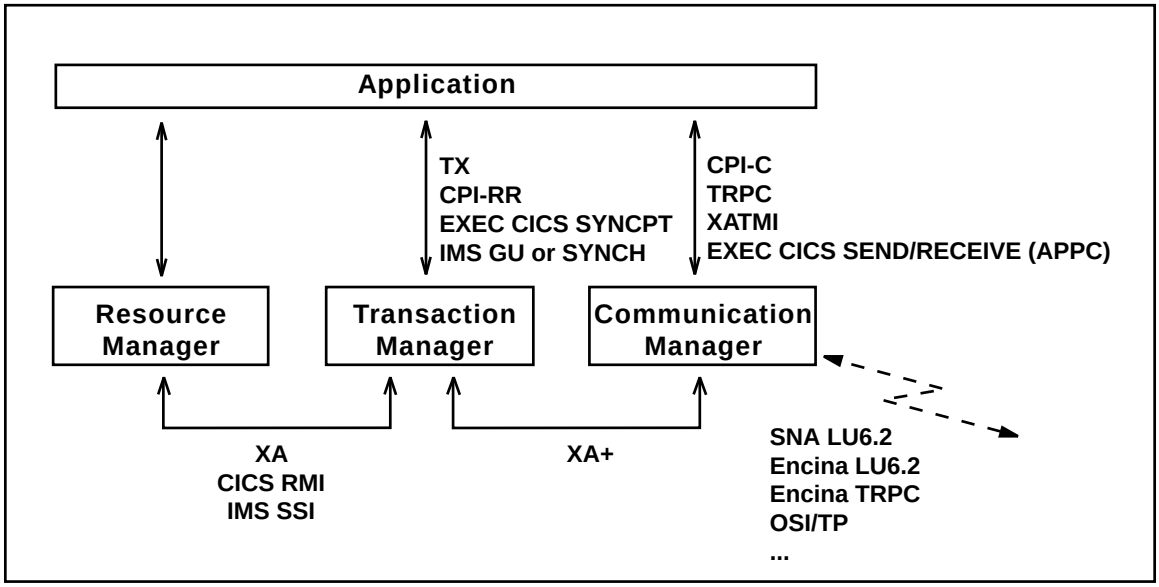


Figure 1. Distributed Transaction Processing Model

Figure 1 is the generic distributed transaction model implemented by products such as CICS, IMS, AS/400 and Encina; it also corresponds closely to the X/Open Distributed Transaction Processing Model.

To see how this model works with the example debit/credit application, assume that the account from which the money is removed is local to the system in which the application is running, but that the money is being transferred into an account that is owned by a resource manager on some other system. The application indicates that it wants to begin a new unit of work by calling its local Transaction Manager resource manager. The application then calls the local resource manager to request that it debit the first account. Because the second account is located on a remote system, the application then calls a Communication resource manager to request a credit to the remote account. This request is received by a Communication resource manager at the remote system, which registers this unit of work with its own Transaction Manager resource manager, then passes the request to the appropriate resource manager. To complete the unit of work, the application calls its local Transaction Manager resource manager, which initiates the prepare phase of the commit protocol, passing the request to prepare to both the local resource manager and a Communication resource manager. The Communication resource manager relays the prepare request to the remote system, where the request passes from Communication resource manager to Transaction Manager resource manager and eventually to the remote resource manager. The resource managers respond, and the local Transaction Manager resource manager (which is the coordinating Transaction Manager resource manager) takes the appropriate action for the second phase of the commit protocol.

In the world of objects, the model illustrated in Figure 2 below, (as defined by the OMG), is logically equivalent to the X/Open model, with the role of the Transaction Monitor resource manager defined as the

Transaction Service. The X/Open model might appear to be somewhat simpler because the function of an Open Blueprint Communication resource manager is handled within the Object Request Broker (ORB). This model is illustrated by Figure 2 on page 7.

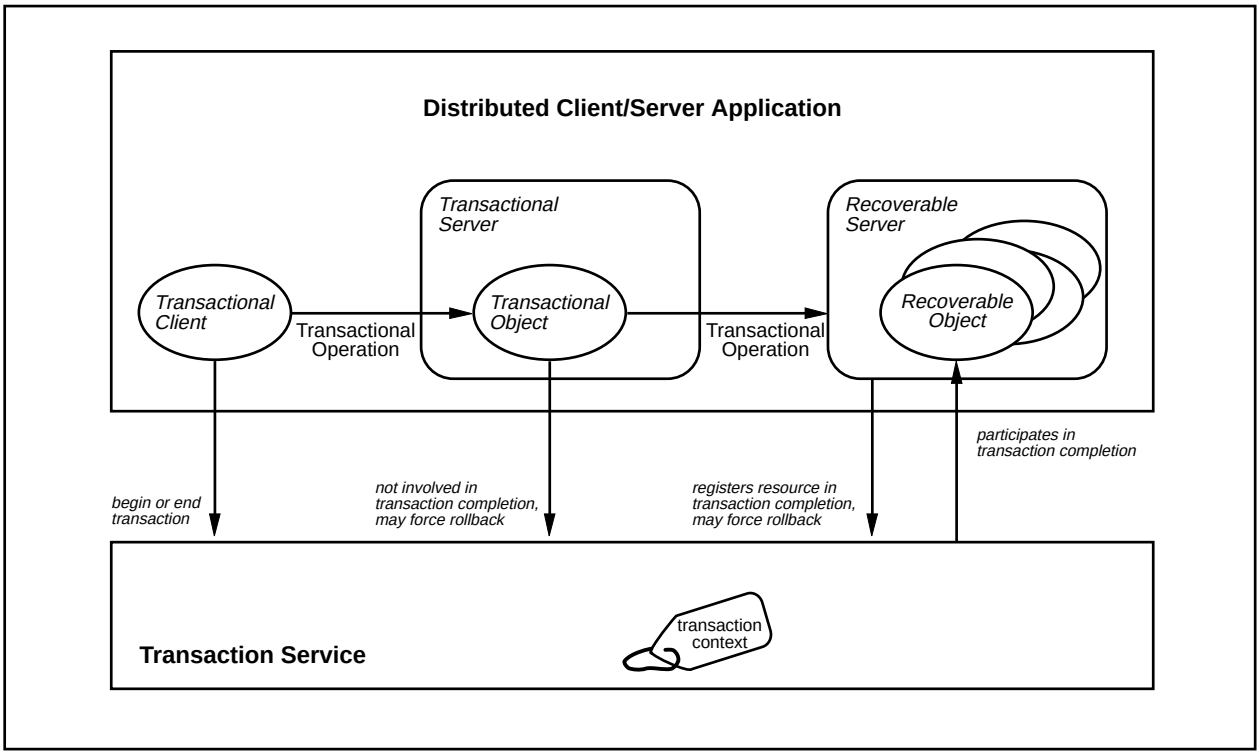


Figure 2. Distributed Client/Server Application Model

A typical application comprises a collection of objects. If an application is transactional, some of its objects will behave transactionally. Such objects belong in one of three categories:

- An object whose behavior can be affected by a transaction but which has no recoverable states or resources associated with it is called a *transactional object*. A collection of one or more transactional objects is called a *transactional server*. Transactional objects and servers do not participate in the completion of a transaction, but they can force the transaction to be rolled-back.

  A transactional server can be likened to the application logic invoked at a remote system using the X/Open Communication Manager.

- A *transactional client* is an object that brackets its method invocations with a series of begin/end requests to the Transaction Service. Within the scope of a transaction, a transactional client can send requests to both transactional and non-transactional objects. Thus, A transactional client is the initiator of a transaction.

- An object that owns and manages recoverable resources or states is called a *recoverable object*. Recoverable objects are transactional objects that have resources to protect and, as such, are affected by the outcome of any transaction in which they participate. A collection of one or more recoverable objects is called a *recoverable server*.

The Transaction Service provides recoverable objects with the transaction coordination services that they need. A recoverable object uses the transaction manager's registration interface to join and leave a transaction and must provide completion methods, which can be invoked by the Transaction Service as part of the two-phase commit process, which is executed at transaction completion. This enables the recoverable object to know when to make appropriate changes in its state at transaction boundaries.

Recoverable objects and servers are equivalent to the X/Open Resource Manager.

The Transaction Service is seamlessly integrated with the Object Request Broker. The begin and end demarcation calls made by a transactional client are intercepted by the Object Request Broker and passed to the Transaction Service. When a begin call is received, the Transaction Service establishes a new *transaction context* and associates it with the client thread. When the transactional client issues subsequent method invocations against transactional or recoverable objects, the Object Request Broker automatically propagates the transaction context; the participating objects are not even aware that this is happening. The transaction context is not propagated to non-transactional objects.

Returning to the simple debit/credit application, suppose that each of the two accounts is represented by its own recoverable object. The processing that causes money to be transferred from one account to the other is then encapsulated in a transactional client object. The client begins the transaction by invoking a begin call to the Transaction Service. The begin call establishes a new transaction context. The client then invokes a debit method on the first account object. This account object invokes the Transaction Service to determine the current transaction context for this thread of execution, then registers with the Transaction Service for this transaction. The Transaction Service keeps track of all participants in the transaction. The client then invokes a credit method against the second account object. This account object then performs an identical sequence of steps on the first account object. Both of the account objects are now registered with the Transaction Service for this particular transaction. Finally, the client issues a commit. The commit is passed to the Transaction Service, which performs the first phase of the two-phase commit process by invoking each participant's prepare method. Depending on the outcome of the first phase, the Transaction Service invokes either the commit or back out methods for each of the participants.

Further, a bank can be implemented as a recoverable server which contains a set of recoverable account objects. Each service offered by the bank can be implemented as a transactional object, with a transactional server representing the total collection of such services.

The Open Blueprint Transaction Manager Resource Manager

The Open Blueprint Transaction Manager resource manager delivers a set of core services that provide the foundation for distributed transaction processing applications. The Transaction Manager resource manager supports three interfaces:

- An interface between the application program or its transaction monitor and the Transaction Manager resource manager that is used to indicate when a transaction begins and ends. In addition, when the transaction is ending, the application can use this interface to indicate whether the completion is either correct and the resources changes can be committed or is in error and the resource changes must be backed out.

  Particular examples of this interface include the X/Open TX interface, the CPI-RR interface (IBM and X/Open), the EXEC CICS SYNCPOINT interface or the IMS GU or SYNCH interface.

- An interface between the Transaction Manager resource manager and the resource managers for the interaction that enable the Transaction Manager resource manager to synchronize all the resource changes.

  Particular examples of this interface include the X/Open XA interface, the CICS RMI or the IMS SSI.

- An interface between the Transaction Manager resource manager and an Open Blueprint Communication resource manager that is used to inform a local Transaction Manager resource manager of the status of a distributed logical unit of work.

  An example of this interface is the XA+ interface (a preliminary X/Open specification). Other products such as CICS and IMS combine the function of Transaction Manager resource manager and a Communication resource manager, and therefore do not define this interface explicitly.

In the Transaction Manager resource manager, both procedural and object-oriented paradigms are supported directly for applications and as a basis for transaction processing monitors such as CICS and IMS or higher-level transaction processing object-oriented frameworks. In addition to the interfaces already described, the Transaction Manager resource manager also supports the OMG Object Transaction Service interfaces.

These functions build on other Open Blueprint resource managers such as Security, Naming, Events and Scheduling, which are described in the component description papers included in the Open Blueprint technical library.

As described previously, the Transaction Manager resource manager coordinates the set of applications participating in a transaction to ensure that the transaction has the atomicity and durability properties required and to this end, provides functions for beginning, ending and aborting transactions. To help achieve atomicity between applications participating in a transaction, the Transaction Manager resource manager provides an implementation of a distributed commitment protocol.

Differing transaction managers can exhibit differing characteristics. Figure 3 on page 10 lists some of these characteristics and indicates which are supported by the Open Blueprint Transaction Manager resource manager, an X/Open Transaction Manager, and an OMG Transaction Service.

Feature	X/Open	OMG	Open Blueprint
Basic transaction co-ordination:			
■ thread/context management	✓	✓	✓
■ log record creation/formatting	x	✓	✓
■ presumed abort recovery	✓	✓	✓
■ unique TID allocation	✓	✓	✓
■ recoverable txn state management	✓	✓	✓
■ asynch call processing	✓	✓	✓
■ dynamic registration	✓	✓	✓
■ nested transactions	x	option	✓
■ rollback only transactions (selectable)	✓	✓	✓
X/Open interface support:			
■ XA, TX (possibly extended where required, e.g. for nested txns)	✓	option	✓
■ CPI-C/RR, XATMI, TxRPC	✓	option	✓
■ XA+	x	x	✓
Performance optimisations:			
■ single phase commit	✓	✓	✓
■ vote read only	✓	✓	✓
■ co-ordinator migration / last agent optimisation	x	x	✓
■ prompt finish	x	option	✓

Figure 3. Transaction Manager Resource Manager Characteristics and Support

The Transaction Manager resource manager also includes a set of System Object Model (SOM) objects that provide an implementation of the OMG Object Transaction Service. The OMG Object Transaction Service is implemented on the same procedural base components as the other Open Blueprint and Transaction Manager resource manager services.

Applications can interface directly to the OMG Object Transaction Service or can be built on higher-level transaction processing frameworks, which in turn are built up the transaction service.

Logging of Transaction State

One of the main purposes of the transaction manager is to assist applications to recover from various types of system failures. To provide this assistance, a transaction manager must log (record) information about the state of the system and the participating components in permanent storage. This is known as *information logging*.

A *log* is a permanent record of operations that is maintained as a file. The data stored in a log is used to recover from two general classes of failure:

- System Failure: crash and transaction recovery
- Media Corruption: media recovery

Standard or generic log interfaces include the ability to:

- Define logs
- Read and write records
- Archive data

A specific log implementation reflects the performance and storage requirements of a user. Thus, log implementations can differ in the following:

Log File An independent data repository(s) utilized to maintain records for a particular user.

Log Record A data entry in a log file (minimally containing a lock sequence number (LSN) and user data).

Chain A logical collection of log records that is used to facilitate efficient retrieval of log information.

Archive A mechanism to allow old log records to be moved to off-line storage or be discarded.

Restart Record Data used to optimize recovery (for example, an anchor point).

A simple, fast, and efficient logging service is incorporated into the Transaction Manager resource manager to log transactional state data and outcomes. Transaction state records contain only a minimal amount of data, such as the following:

- For the transaction coordinator, outcome and participant list
- For any other resource manager, identification of what is prepared and a pointer to the transaction coordinator address

Because the log does not usually exhaust the available online storage allocation, archiving capabilities are not usually required to support the Transaction Manager resource manager. The Transaction Manager resource manager logging service is not generic, because other resource managers provide their own internal logging, which is specifically tailored to their requirements.

Transactional Interoperability

Because the purpose of a transaction manager is to coordinate the activity of a set of resource managers, the effectiveness of any given transaction manager is, to some extent, determined by the transaction manager's ability to interoperate with a wide range of different transaction processing models, products, and environments. This final section describes some of the important interoperability considerations and the level of support provided in the Transaction Manager resource manager.

Interoperability Characteristics

Two forms of interoperability need to be considered, namely the way in which different styles of applications coexist and interact on a single system (*model interoperability*) and the way in which multiple systems interoperate (*network interoperability*).

Model Interoperability

Customers want to mix existing procedural applications with new object applications or to develop object applications that are augmented with code that uses a procedural paradigm. In addition, customers might want to mix applications based on an application server model, for example, an application that uses the CICS External Call Interface (ECI) to invoke a remote CICS transaction, and on a distributed unit of work model such as the X/Open DTP model. To do this successfully in a transaction environment, both implementations must share a single transaction.

A client-to-server application (such as IMS or CICS), the X/Open DTP model, and the OMG Object Transaction Service Model contain the following set of characteristics:

- A single transaction, which includes ORB and non-ORB applications and resources
- Interoperability between the OMG Object Transaction Service model, the X/Open DTP model, and the CICS client/server model
- Access to existing (non-object) programs and resource managers by objects
- Access to objects by existing programs and resource managers
- Coordination of the activities of both object and non-object resource managers by a single instance of the Transaction Manager resource manager

In addition to these characteristics, it is necessary to consider the network case of a single transaction that is distributed between an object and non-object system, each system with its own transaction service.

The OMG specification for the Object Transaction Service accommodates all of these requirements for implementations where interoperability with X/Open procedural applications is necessary.

Integration with other client/server models such as IMS or CICS is more complex, because these models assume that the server system always acts as the coordinator (although the CICS DTP model allows one CICS system to be subordinate to another).

Network Interoperability

Customers expect and require interoperability both among systems that are based on different network paradigms (such as the SNA transactional model or the ORB model) and among systems that are offered by multiple vendors (which might use similar paradigms). The particular models which need to be addressed can include the following:

- SNA LU6.2
- Encina PPC protocols
- Encina Transactional RPC
- OMG Object Request Broker

In the X/Open world, this level of interoperability is defined by the X/Open Communications Manager specification.

In the object world, interoperation occurs through the Object Request Broker (ORB), with four possible scenarios to consider:

- **Single transaction service and single ORB**. Multiple instances of a single transaction service must interoperate with itself using a single ORB.
- **Multiple transaction services and single ORB**. One transaction service must interoperate with a cooperating transaction service using a single ORB.
- **Single transaction service and multiple ORBs**. A single transaction service must interoperate with itself using different ORBs.
- **Multiple transaction services and multiple ORBs**. One transaction service must interoperate with a cooperating transaction service using different ORBs.

The OMG Object Transaction Service supports interoperability between transaction service applications and procedural applications using the X/Open DTP model. A single transaction management component acts as both the transaction service and an X/Open Transaction Manager resource manager.

Note: The OMG does not specify an implementation of the transaction service and allows for a separate and distinct transaction service and X/Open Transaction Manager. The requirement to associate the transaction between paradigms is called *implementation dependent* by the OMG.

Open Blueprint Capabilities

In the Open Blueprint environment, a transactional application can choose from a number of different programming paradigms, such as:

- EXEC CICS
- CICS External Call Interface (ECI) calls
- IMS DL/I
- Encina Transactional-C
- Object Services

The interoperability between the OMG Transaction Service and Encina applications is a natural consequence of the built-in synergy between the OMG and X/Open models. The reuse of an underlying procedural X/Open-based Transaction Manager resource manager provides the transaction association through the implementation of class libraries that map between the two models at the paradigm boundary. To interoperate with the CICS or IMS worlds (or more generally, the SNA LU6.2 world) additional extensions are required. This interoperability can (and should) be viewed from the perspective of the initiating application.

- **Transactional Objects**

  Require access to:

  – X/Open resource managers and communication managers

    In the OMG Transaction Service specification, this access is referred to as exporting transactions from the Transaction Service domain to the X/Open domain. (This is an example of model interoperability.)

  – Mainframe-managed resources (SNA LU6.2 model - for example, CICS, IMS or AS/400 resources). (This is an example of network interoperability.)

- **X/Open Applications**

  Require access to:

  – Transactional Objects

    In the OMG Transaction Service specification, this access is referred to as importing transactions from the X/Open domain to the Transaction Service domain.

  – Mainframe-managed resources (SNA LU6.2 model - for example, CICS, IMS or AS/400 resources)

  These scenarios include applications written to the Encina programming interfaces.

- **CICS or IMS applications**

  Require access to:

  – Transactional Objects

  – X/Open resource managers and communication managers

  (In general, these are both examples of model interoperability.)

Because CICS and IMS client applications do not explicitly delineate the scope of a unit of work, it is assumed that any extension of existing CICS or IMS client applications embrace either the transactional object or the X/Open models and should occur in such a way that the new model provides the required delineation function (that is, ECI calls or IMS DL/I calls become embedded within a unit of work defined by either the X/Open Transaction Manager Resource Manager or the OMG Object Transaction Service).

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

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 protected rights, any functionally equivalent product, program, or service may be used 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:

AS/400
CICS
IBM
IBMLink
IMS
Open Blueprint
SOM
System Object Model

The following terms are trademarks of other companies:

Encina	Transarc Corporation
X/Open	X/Open Company Limited in the U.K. and other countries

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

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

GC23-3930-01