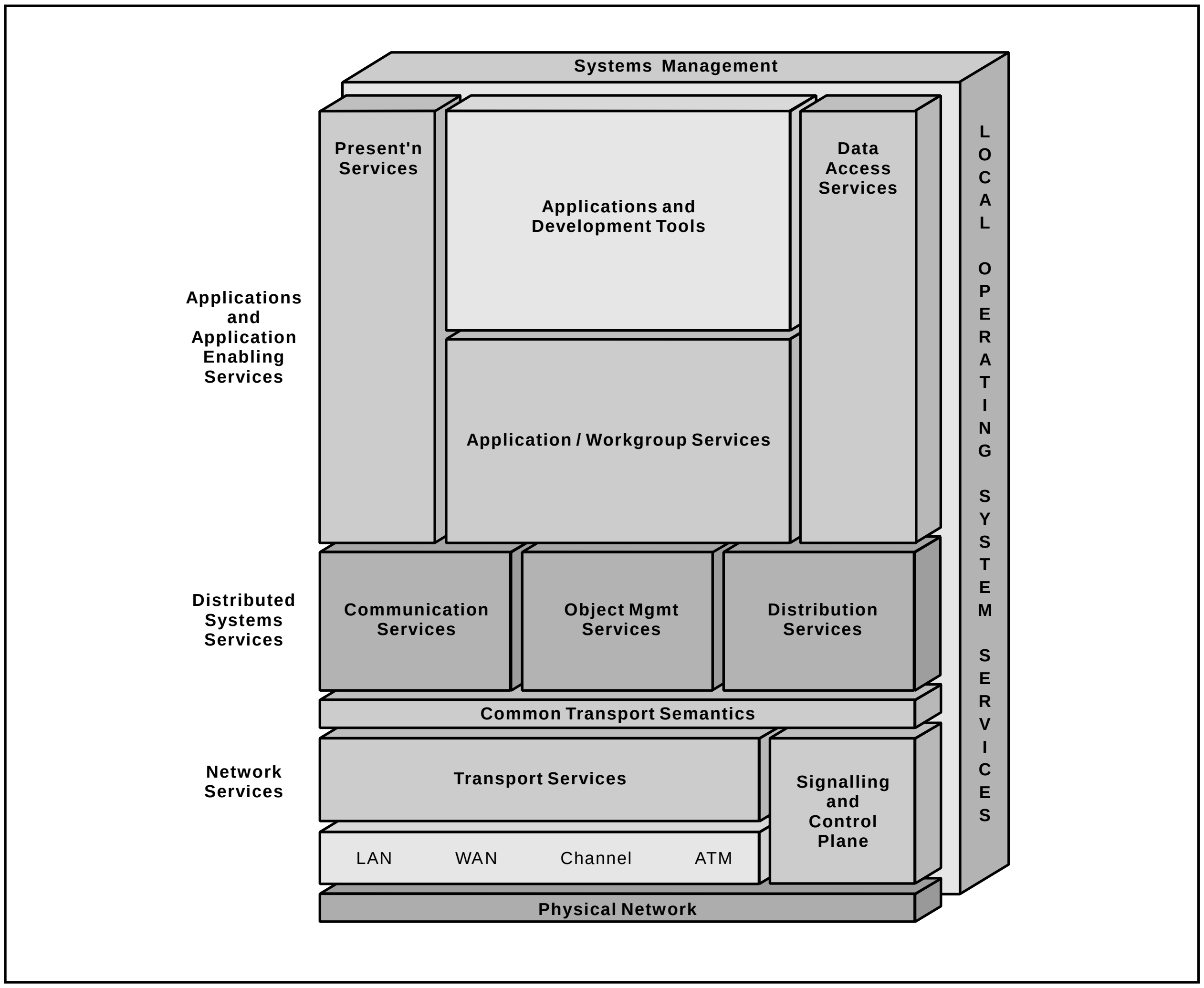
Open Blueprint

IBM

Messaging and Queuing Resource Manager


Systems Management
Present'n Services
Applications and Development Tools
Data Access Services
LOCAL OPERATING SYSTEM SERVICES
Applications and Application Enabling Services
Application / Workgroup Services
Distributed Systems Services
Communication Services
Object Mgmt Services
Distribution Services
Common Transport Semantics
Network Services
Transport Services
Signalling and Control Plane
LAN
WAN
Channel
ATM
Physical Network

GC23-3920-01

Open Blueprint

Messaging and Queuing Resource Manager

GC23-3920-01

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 Messaging and Queuing 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 Messaging and Queuing 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

Figures

© Copyright IBM Corp. 1995, 1996

Summary of Changes

This revision describes:

- The addition of the MQCMIT and MQBACK functions
- Advanced messaging facilities, which include support for mobile computing and the Internet
- Relationships with the Collaboration and HTTP resource managers

© Copyright IBM Corp. 1995, 1996

Messaging and Queuing Resource Manager

Message queuing (MQ) is a technique used to communicate from program to program. It can be used within any application where programs need to communicate with each other. MQ communication is performed by the programs sending messages to each other using queues. The messages contain application data that is passed from one part of the application to another. The parts of the application can be on the same system or on separate systems in a network.

The message queuing service is provided by a Messaging and Queuing resource manager which owns and manages the queues, and which provides the message queue application programming interface (MQI).

Technical Overview and Concepts

Examples of application areas where the messaging model is useful include:

- **Automated Teller Machines (ATMs)**. With modern banking facilities, it is common for wall-mounted ATMs to provide other facilities besides dispensing cash on request. With all these facilities, the ATM must make inquiries on databases (to obtain the balance of an account, for example), or to request that a function be performed by another program. These inquiries and requests are passed on as MQ messages and sent to specific queues within the banking system.

  This type of application is characterized by short messages and very high message rates, with message delivery times on the order of tenths of seconds. It is critical for any application which involves monetary transactions that messages are not lost, and are delivered only once.

- **Travel Reservations**. Each travel operator (airline, hotel chain, rental car agency) can have their own queue for reservations. Requests for seats, room or car reservations, and confirmations of bookings, are all MQ messages that are directed to the appropriate queues. This type of application is characterized by many users, short messages, very high message rates, and message delivery times on the order of seconds.

- **Mobile Sales Agents**. Mobile sales agents take their portable computers with them when visiting customers and might only be connected to their company networks when they dial in or return to the office. They need their standard applications to work whether or not they are connected to the network. The message queuing style of communications allows applications to operate even when they are not connected to the network.

- **Enterprise Integration**. Many enterprises have LAN-based work group facilities such as Lotus Notes that are not connected to their business applications and that are frequently mainframe-based. The need for business efficiency demands that the different environments are integrated so that work group users have access to corporate data and applications.

- **Internet Access**. The growth of the Internet has caused companies to want to make corporate data available to their customers over the Internet. This data is often scattered over various machines within the company. The Messaging and Queuing resource manager can be used as a bridge from a World Wide Web (WWW) server to the business machines.

Though all these applications can have vastly differing characteristics such as message lengths, message rates, and performance requirements, they can all be designed using the facilities of the single messaging model.

© Copyright IBM Corp. 1995, 1996

Applications and Programs

In the context of message queuing, an application is any set of programs that communicate with each other using messages. Each program performs a well-defined and self-contained function in response to a specific request, and executes within one node of a network, or within one operating system image within a node. The programs communicate requests to each other using messages placed on queues.

In a distributed application, each program that forms the application can be executing in different nodes in a network.

Message queuing can be used for most types of distributed application structure, from simple examples with a few programs in a client/server structure to much more complex structures involving many programs that are connected using combinations of client/server, chaining, and parallel execution.

Messages

A *message* is simply a string of bits and bytes that have some meaning to one or more programs.

The message is a block of data, in any form, that is being passed from one program to another. Some examples of typical messages are:

- A message sent from one program to another requesting that some service be performed, together with the input data that is required to perform the service.
- A reply from one program to another indicating that a service has been performed, together with some indication of the status of the initial request.
- A message that is sent from one program to another using the lowest-cost delivery mechanism. Often, this type of message does not require a reply.
- One of a series of messages forming part of a dialogue between two programs.

The programs usually impose some structure on the string of bytes—that is, the programs use an agreed to function protocol and view the string as consisting of a sequence of components, each having a particular data type and meaning to the application. For example, the first component might be a four-byte unsigned binary integer containing an account number, the second component might be a 20-byte character string containing a customer name, and so on.

MQ has no architectural limitation on the length of the data that can be carried in a message. The data can be a small number of bytes, as in the case of an electronic transaction in a travel reservation system. Message queuing also allows messages to be extremely long. For example, messages can contain image data or audio data (such as voice or music).

Initially, implementations might place limits on the maximum size of the messages they can handle, or they might have path lengths optimized for messages within a specific range of lengths.

Message Queues

A *message queue* is a named storage area for storing messages. The messages on the queue are *in transit* between one program and another. They were placed on the queue by one program and are waiting to be retrieved by another.

Message queues can exist without programs being active. Each queue belongs to an instance of the Messaging and Queuing resource manager, not to the program that might be retrieving the messages from the queue for processing. The queue manager maintains the queue; a queue manager can own many queues.

Each queue has a name and a set of *queue attributes*. The queue name is used by a program to identify the queue to be accessed. The queue attributes affect the way the queue is processed by the queue manager and includes the following:

- Permanent or temporary queue
- Storage space allocated to the queue
- The number of messages on the queue

Programs usually need to know only the names of the queues that they are using and do not need to be aware of the detailed queue attributes. However, facilities are provided by the MQ resource manager to allow a program to query the values of certain queue attributes.

Each message has a priority associated with it, which affects how it is added to the queue. Messages within a queue are maintained in first-in-first-out-within-priority sequence. However, messages can be read from the queue in an order which is different from the order they occur on the queue. For example, a program can process the reply to a particular message that it sent earlier. The program can retrieve this reply from the queue even though it is not the first message.

Physically, a message queue can be represented in two ways:

- As a buffer or buffers in main storage
- As a file or files on disk or other permanent storage device

A single queue can reside entirely in main storage, entirely on disk, or in both places. The physical management of message queues is entirely the responsibility of the queue manager, and such details are not made apparent to the programs. For the program, a message queue is simply a black box in which messages accumulate. Programs can only access message queues by using the MQI verbs.

Messaging and Queuing Resource Manager

The Messaging and Queuing resource manager is the component in the Open Blueprint structure that provides the message queuing facilities used by application programs. The MQ resource manager consists of a client component, which accepts MQI requests from programs, and a server component, which provides the queuing function. When the application program and the queue manager are on the same system, for example in a mobile environment, the MQ client and server components are combined.

Logically, the full function of the MQ resource manager can be considered as being distributed across a set of interconnected queue managers within the network. Each queue manager services those MQI requests that are made by the programs that execute within its system or client systems. Depending on the request, a queue manager might need to communicate over the network with other queue managers.

For message-queuing services to be available, there must be at least one queue manager or MQ client on a system. However, there can be more than one queue manager on a system (for example, to keep development work separate from production work).

The Messaging and Queuing resource manager has the following important characteristics:

- **Asynchronism**. Message queuing is *naturally asynchronous*; program A sends a message to program B, but program B need not be there to receive it. The message is not lost, but is retained by the message queuing service. Program B processes the queued message when it starts execution, which might be immediately but can also be some time later.

  Program A may or may not expect a reply from B, but A need not suspend execution waiting for B to reply. Instead, A can perform other work and then process the reply from B when it arrives. Program A can even terminate before the reply arrives; the reply is not lost—it is retained by the message queuing service for A to retrieve when it next executes.

Message queuing can be used for asynchronous communication between programs, and it can also be used for a synchronous communication. In the previous example, having sent a message to program B, program A can wait for a reply from B before continuing with its execution.

- **Assured Once Only Delivery**. When an application puts a message on a queue, the application can be sure that the MQ network will not lose the message. MQ also ensures that the message is only delivered once to the receiving application, which greatly relieves that application development burden, and which distinguishes MQ from other messaging systems.
- **Local and Remote Transparency**. Message queuing can be used between any two programs, whether those programs are executing within one machine or in different nodes within a network.

  If program A wishes to communicate with program B, it does not need to know whether B executes within the same node as A or in some other node. The programming statements that are necessary for program A to send a message to program B, where A and B execute within the one node, are identical to those that would be required if program B were in a different node from A. In fact, program B can be moved (by the network administrator) from one node to a different node without requiring that the source code for program A be changed or that the program be recompiled.
- **Application Structure Support**. The MQ model supports many types of distributed application structures, ranging from simple structures such as client/server that contain just a few programs, to complicated structures with many inter-communicating programs.
- **Ease of Use**. MQ has a simple model that requires only a small number of MQI verbs. The programming interface hides the underlying complexities that are involved in communication protocols, operating system services, and heterogeneous systems, and makes it easier for programmers to develop application programs.

  The fact that the MQI is a simple programming interface makes it very easy for an application designer to structure an application program into several separate program modules, with well-defined interfaces between modules. This encourages a very modular application design, which enables ease of programming and the re-use of program code.

Message Queuing Interface

This section is a summary of the principal MQI verbs used by application programs and gives an outline of the various functions provided by the MQI.

The same message queuing programming interface is used, regardless of programming language used to write the application programs.

MQCONN	Establishes a connection from a program to a particular queue manager. This function is necessary before any other MQI functions can be used.
MQOPEN	Creates or opens a particular queue. This function is necessary before any other function can be performed on the queue. MQOPEN checks that the program has the appropriate authority to access the queue.
MQPUT	Puts a message on a queue.
MQPUT1	This function is semantically equivalent to the sequence MQOPEN, MQPUT, MQCLOSE. It improves the usability of the programming interface because the function of sending a single message to a queue is a common function within some types of application programs.
MQGET	Gets a message off a queue. This function is used by a program to take a message off the queue so the program can process the message. This function allows several ways of selecting the next message that is to be obtained off the queue.

MQCMIT	Commits a sequence of messages that have been processed by either the MQGET or MQPUT verb (or both) as part of a unit of work. Only Messaging and Queuing resource manager messages are committed. Coordination of Messaging and Queuing resource manager resources with other recoverable resources requires a transaction manager. The transaction manager commit verb must be used rather than MQCMIT.
MQBACK	Backs out a sequence of messages. This function backs out messages that have been processed by either the MQGET or MQPUT verbs (or both) as part of a unit of work. Only Messaging and Queuing Resource Manager messages are backed out. Coordination of Messaging and Queuing resource manager resources with other recoverable resources requires a transaction manager. The transaction manager back out verb must be used rather than MQBACK.
MQINQ	Inquires about queue attributes. This function allows a program to inquire on the various attributes of a queue. For example, it can be used to determine the number of messages on a queue.
MQSET	Sets queue attributes. This function allows a program to change the value of certain queue attributes.
MQCLOSE	Closes a queue and optionally requests that a queue be deleted. This function is used when the program has finished accessing a queue.
MQDISC	Breaks the connection between a program and a particular queue manager.

The Message Queuing Interface is available as procedural calls and will be available as object-oriented (OO) class libraries for System Object Model (SOM), C++, and LotusScript.

Program-to-Program Communications

Programs communicate by agreeing to use specifically named message queues.

For example, program A puts messages onto the queue named ABC, which is the queue that program B has agreed to read from, while program B puts messages onto the queue named XYZ, which is the queue that program A has agreed to read from. The location of these queues is not apparent to the programs, because each program interacts only with its local Messaging and Queuing resource manager client, and the network of interconnected queue managers is responsible for moving the messages around so that they eventually appear on the intended queues[1].

A program puts a message on a queue by using the MQPUT verb and supplies the message data and some ancillary information that is carried with the message. The program can continue processing. A receiving program retrieves a message from a queue using the MQGET verb. This function returns the message data to the program with the ancillary information.

After putting a message on a queue, a program might later receive a reply to the message it sent, or the reply might be directed to another application program in the suite. This depends on the design of the application suite.

[1] Of course, someone has to know where the queues reside, but that person is not the application programmer; it is the system administrator who is responsible for defining the queues and their locations and installing the applications that use those queues.

Local Applications: If programs A and B are connected to the same queue manager, the logical structure looks like Figure 1 below.

Program A *puts* a message on the queue that it knows is being serviced by program B. Program B *gets* the message off the queue when it wants to process the next message on the queue.

In this local case, both programs are interacting with the same queue manager (QM1). Routing a message from program A to program B does not involve sending data across any network connection.

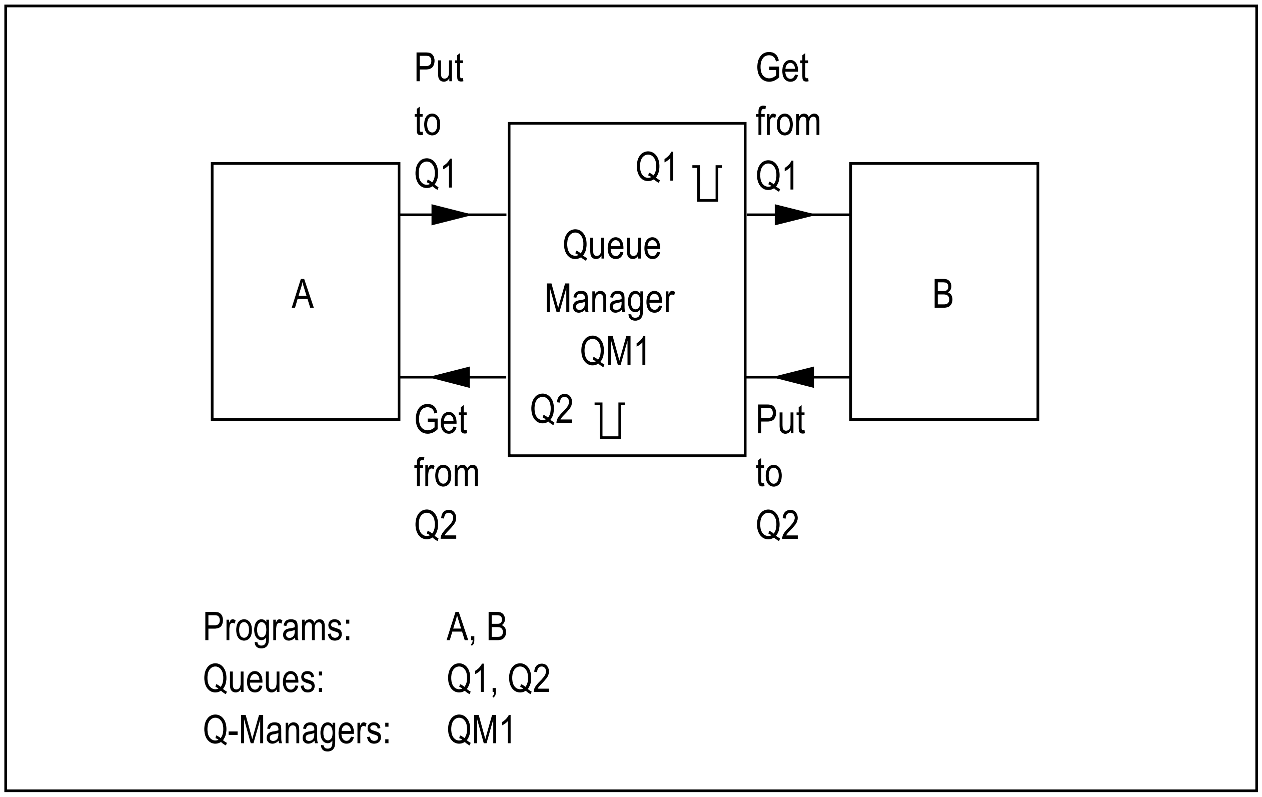


Figure 1. Logical Model of Local Applications

Programs accessing the queue manager using an MQ client are also considered local applications. Although they might be on different systems, only one queue manager is involved.

Remote Applications: If programs A and B reside on different nodes within the network, then the logical structure looks like Figure 2 on page 9.

As with the local application case above, program A *puts* a message on the queue that it knows is being serviced by program B, and program B *gets* the message off the queue. However, in this remote application case, A and B are interacting with different queue managers (QM1 and QM2). Sending a message from program A to program B now involves routing data across the network from one node to another. The applications are not aware of how the message is moved through the network or whether the message is sent over a wired or wireless local area network (LAN) or wide area network (WAN).

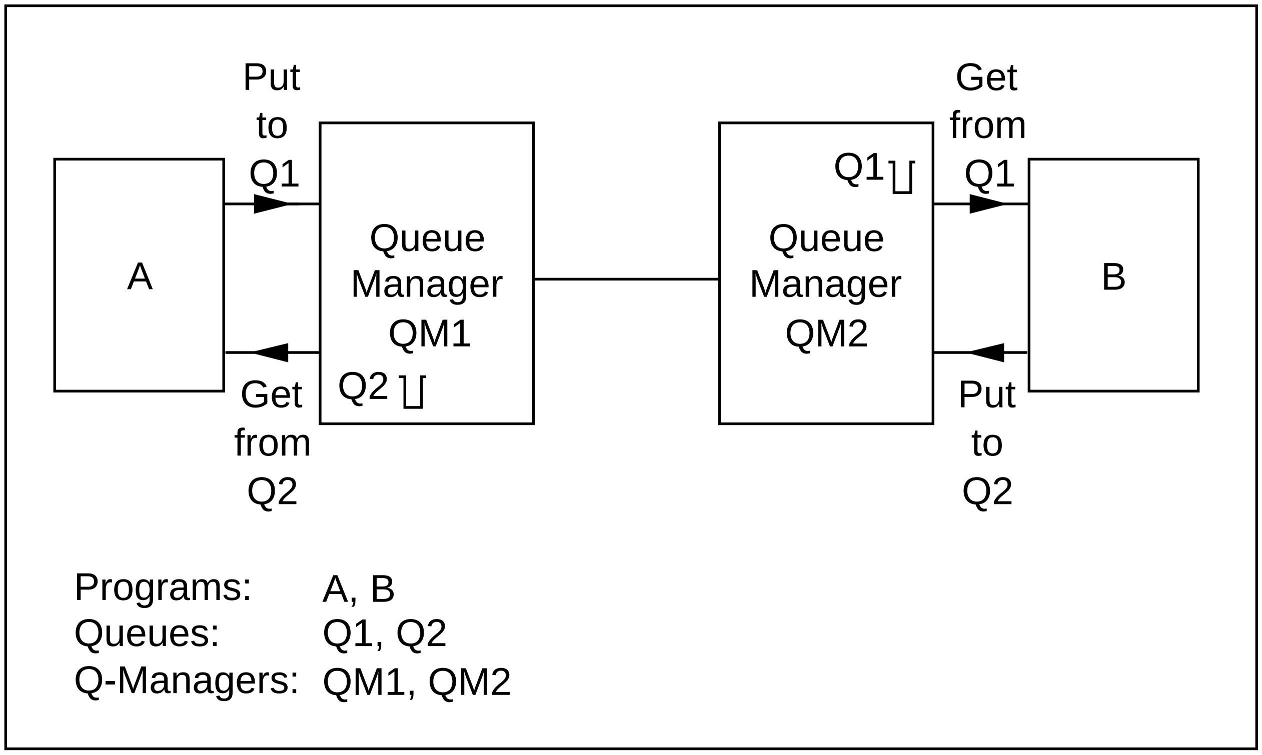


Figure 2. Logical Model of Remote Applications

Typical Applications

Message queuing can be used for most types of distributed application structures, ranging from simple structures with just a few programs involved, to much more complex structures with many programs that are connected using combinations of subroutining, chaining, and parallel execution.

A typical example of an application that uses subroutining and parallel execution of programs is illustrated in Figure 3 on page 10.

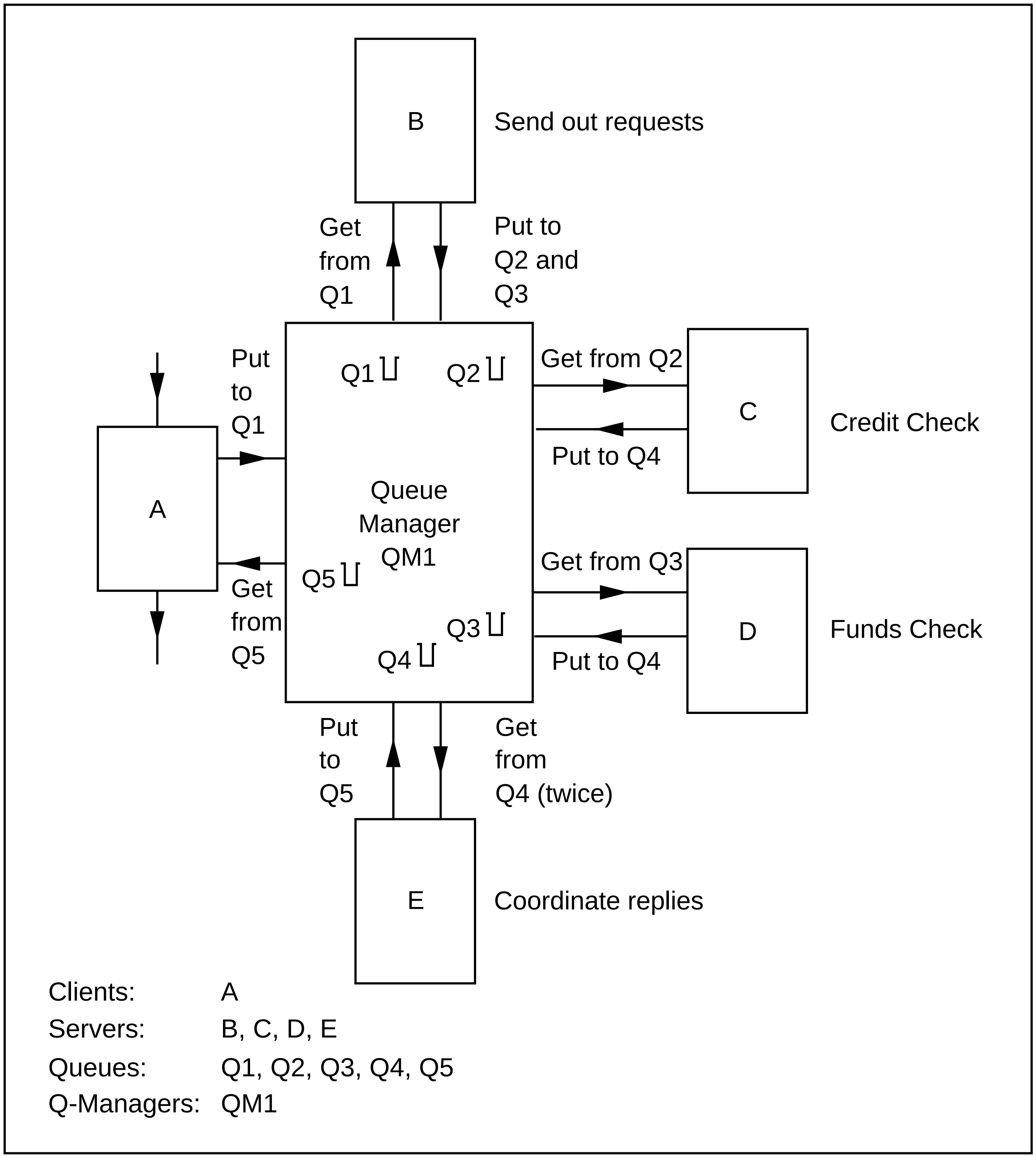


Figure 3. Program-to-Program Communication with Queues

In this example, program A determines from user input (not shown in Figure 3) that a certain service needs to be performed (for example, a loan application). Program A sends a message to queue Q1 requesting this service. Program B gets the message off the queue, and is aware that (1) two activities need to be carried out; the credit-worthiness check, and a check that sufficient funds are available, and that (2) the two activities can be performed in parallel. Program B sends messages to queues Q2 and Q3 requesting these activities. Programs C and D read these messages, and send their replies to queue Q4. Program E coordinates the replies, and sends the yes/no response back to queue Q5. Program A gets the reply, and takes the necessary action.

Figure 4 on page 11 is an application designer's view of Figure 3 using a lattice structure.

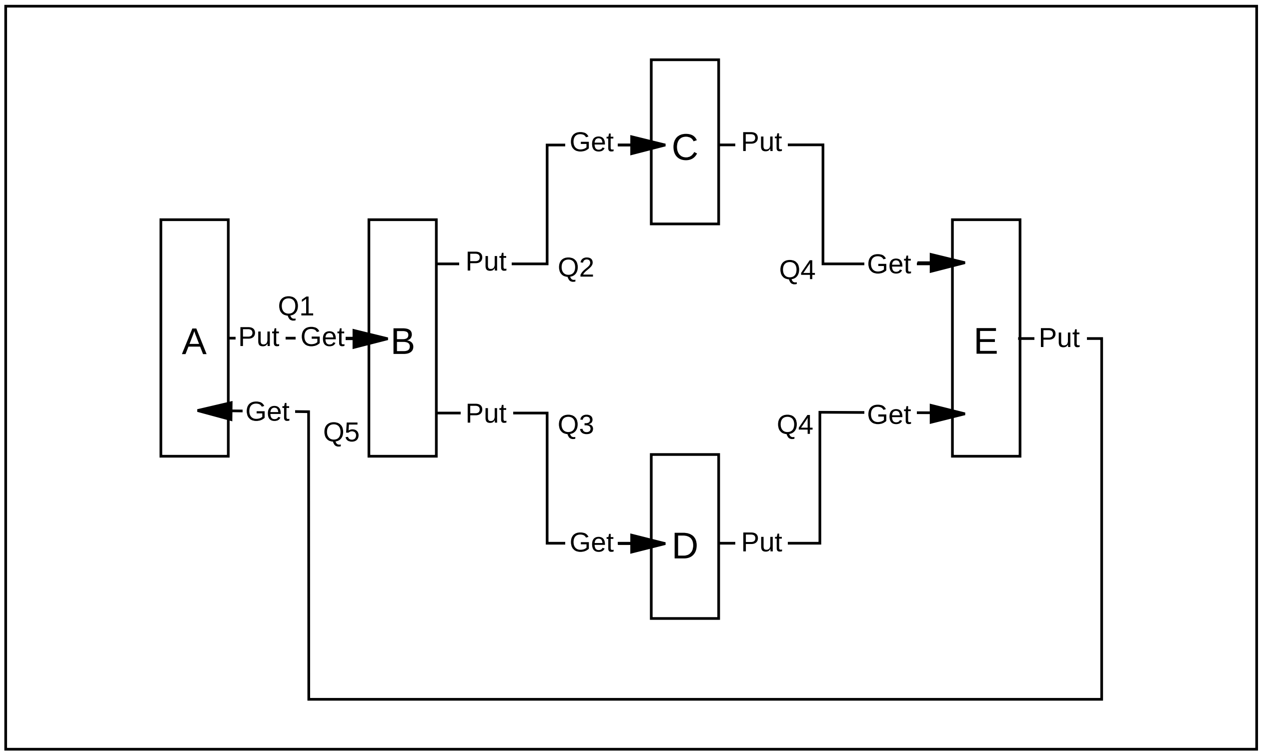


Figure 4. Application Designer's View of the Previous Figure

Client/Server Applications: Message queuing can be used for client/server applications, where many client programs send messages to a single server program. Figure 5 on page 12 shows the case of three clients A, B, and C sending messages to the server S. The server replies to three different queues using the Reply to Queue name in the message sent by the originator of the request. Two clients (A and B) are remote from the server, while one client (C) is local.

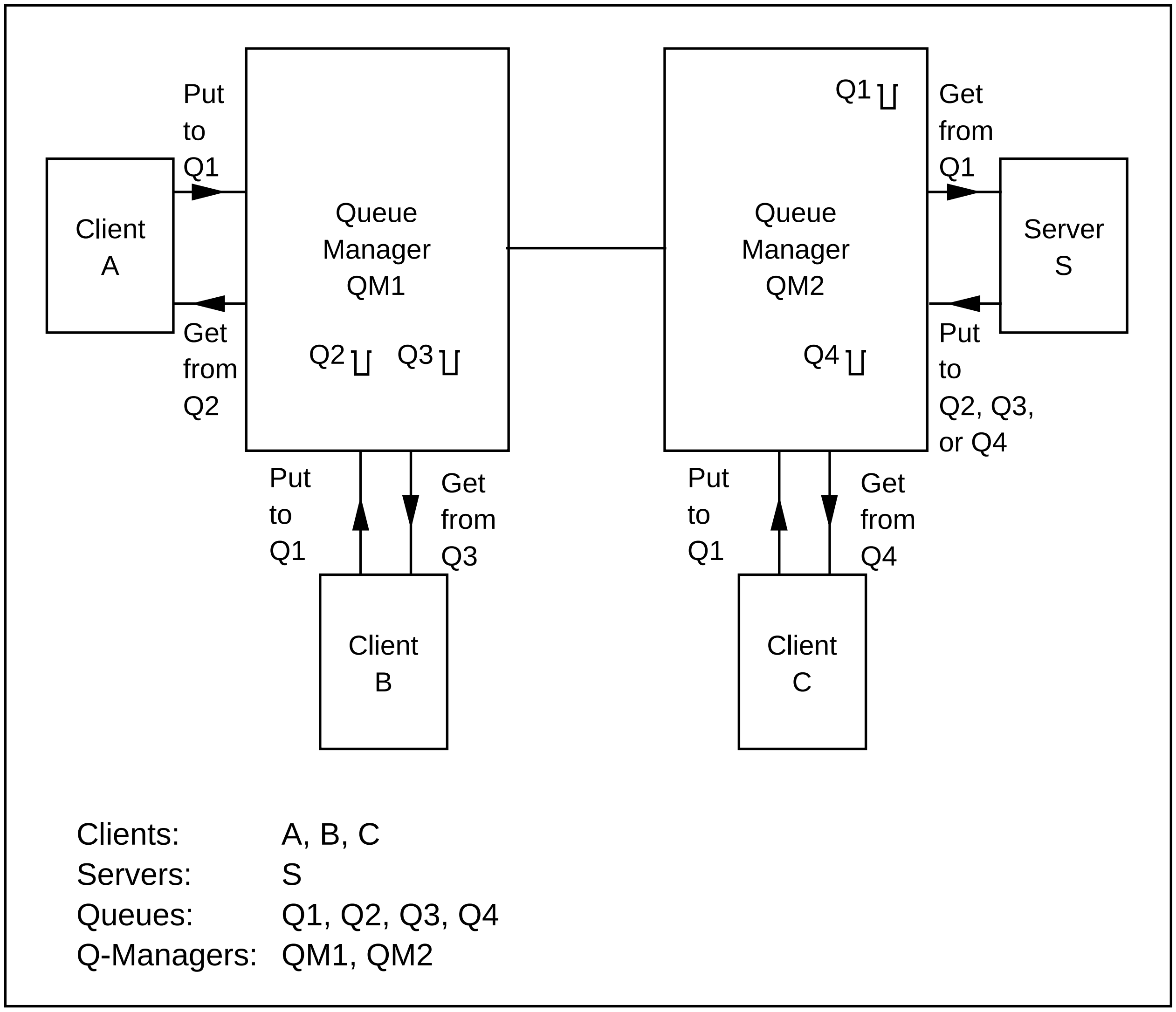


Figure 5. Logical Model of a Client/Server Application

Figure 6 on page 13 is an application designer's view of Figure 5 in tree structure (with the base on the right and the branches on the left) format.

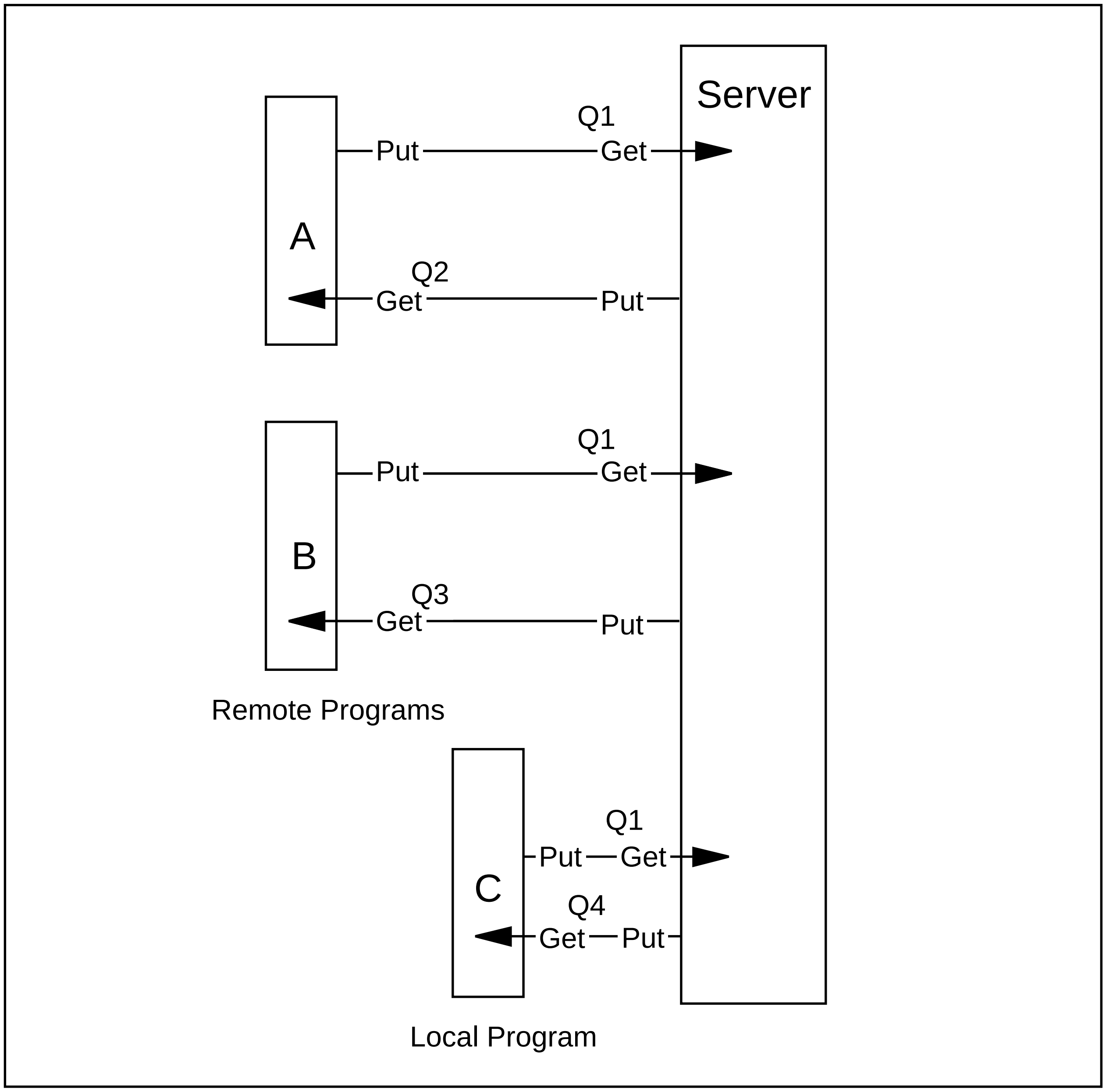


Figure 6. Application Designer's View of Client/Server Application

Starting Programs

A program that processes messages on a queue can be started in any of the following ways:

- By a user
- By the local operating system
- By another program

Programs that are started by the user and programs that are started by the operating system are similar; the execution of the program is not related to the status of the queue. For example, a user might start the program whenever the user logs on to the system—the program runs, drains the queue of messages (probably few in number), then terminates. Alternatively, the operating system might start the program whenever the system is started, or at some specific time of day. The program might run indefinitely or might run until the queue is empty. The queue manager need not take any special action in either case.

However, many applications require a more sophisticated way to start programs that process the arrival of messages on queues. For example, the application designer might want a particular program to start when a message arrives on a previously empty queue or when a high-priority message arrives on a particular queue.

The queue manager provides a mechanism, called *triggering*, that allows an application program to start one or more other programs in those situations. Triggering works as follows:

1. The application design requires that a program, A, is to be started when a message of priority 3 or higher arrives in queue Q1. The programmer needs to define queue Q1, specifying that triggering is required, and provides:
   a. The name of the program that is to be started (that is, program A)
   b. A threshold priority value of 3
   c. The name of an initiation queue (for example, INITQ1)
2. At execution time, the queue manager monitors the arrival of messages on queue Q1. When a message with a priority value that is equal to or higher than the threshold of 3 arrives on the queue, the queue manager places a system-defined trigger message on the initiation queue INITQ1. This message contains the name of the queue (Q1) that was triggered and the name of the program (Program A) to start.
3. Messages are read by a *trigger program*, which reads the trigger message off the initiation queue and starts program A. Program A can then open queue Q1 and read the priority 3 message.

This is a simple example of the MQ triggering mechanism. Because trigger programs can be supplied by the customer, the algorithms used in trigger programs can be as simple or as complicated as required. These algorithms can consider the time of day, the depth of other queues, or the number of copies of program A that are running already.

The responsibility for starting programs belongs to the trigger processing programs, which can be supplied by either the customer or by the messaging product.

Advanced Messaging Facilities

In addition to providing the straight-forward queuing mechanisms described so far, the Messaging and Queuing resource manager provides several other facilities that are useful to certain applications:

- **Alias Queues**. An alias queue is an alternate name for the target queue. The definition of the alias is maintained outside the program, so the target queue name can be changed without affecting the program.

  Multiple aliases can exist for the same target queue name, allowing several programs to use different names for the same target queue.
- **Persistent Messages**. Optionally, a message can be defined as being *persistent*, that is, saved on permanent media. Persistent messages are recovered after system and queue manager failures.
- **Mobile Support**. Mobile computing is characterized by a computer that is intermittently connected to the network. Mobile support includes users who only dial into the network when needed, users who take the computer home to do work but who reconnect in the office, and truly remote users who connect over a wireless connection. While disconnected from the network, the Messaging and Queuing resource manager stages messages reliably in queues on the user's computer. When reconnected, the messages flow to their destination. The Messaging and Queuing resource manager allows users to easily change the communication protocol used to connect to the network when they leave the office. It optimizes the flows to reduce network traffic over wireless networks and understands when it is best to send data based on traffic rates.

- **Internet Support**. The Messaging and Queuing resource manager supports passing messages over the Internet. In addition, it provides a gateway to a World Wide Web (WWW) server that converts Web browser requests to messages that flow to existing applications to make corporate data available.
- **Bridges**. Applications can communicate using the MQI. However, it is often desirable to allow access to existing applications that are not distributed. The Messaging and Queuing resource manager provides bridges to allow these applications to be driven by messages. For example, IMS applications that were originally written to be used from 3270 terminals can be driven unchanged by messages, allowing them to be front-ended easily.

Performance

Message queuing has several features that make it suitable for high-throughput, performance-critical applications. These features include the following:

- **Parallel Execution**. Message queuing allows an application to be designed so that some of its constituent programs can execute in parallel. These programs can execute in different nodes within the network. Parallel execution enables the application to provide an improved response to users and to provide improved system utilization.
- **Load Balancing**. Message queuing allows multiple copies of a program to process a single queue, with the programs executing in parallel. These programs can be started dynamically, depending on the number of messages on the queue, to provide extra queue processing capability at peak load times.
- **Session Concentration**. Message queuing allows applications to be designed so that no direct session connection is necessary between the programs that want to communicate. This reduces the total number of sessions required by the application, reduces the time used to start the network, and simplifies network control. This can be of particular advantage to applications that run on large networks. It also permits the Messaging and Queuing resource managers to optimize the movement of messages around the network using techniques such as batching of messages.
- **Multiple Connections**. Messaging and Queuing resource managers support multiple connections between them, allowing for the separation of messages with different performance requirements. For example, MQ managers can prevent high-priority messages from being delayed by the transmission of large, lower-priority messages.

Interoperability

Message Queuing is available on a large number of platforms and environments so that programs that execute in the different environments are able to communicate with each other.

Message queuing interfaces are being proposed to the Object Management Group (OMG) to extend the object standards to include messaging and asynchronous methods.

Systems Management

The Messaging and Queuing resource manager provides commands to create, start, stop, and destroy queue managers.

The resources of the queue manager, such as queues, are dynamically defined and managed using script commands, a programmable interface, or a system administration tool. The programmable interface manages by sending a message to a particular queue, so it is possible to administer both local and remote queue managers. Queue managers can also monitor themselves for particular situations, for example, when a queue is getting full or when the queue manager stops. When these events occur, a message is put on a monitoring queue and made available to system management applications.

These interfaces will be used by systems management agents to allow Open Blueprint systems management services to control the queue manager.

Relationship to Other Resource Managers

Message Queuing provides the messaging infrastructure for other resource managers to use. These include:

- **Workflow**. The Workflow resource manager uses the Messaging and Queuing resource manager as the underlying transport to control the workflow through the business process.
- **Transaction Monitor**. Messaging and Queuing resource manager can be used by transactions running in the transaction monitor environments.

The Messaging and Queuing resource manager uses other resource managers as follows:

- **Transaction Manager**. As an optional facility, message queuing can collaborate with the Transaction Manager resource manager to provide full recovery of message queuing resources on a logical unit of work basis. This allows updates to queues, for example, to be synchronized with the proper updating of other resources such as databases even in situations where hardware or software fail.

  Programs use the transaction manager application programming interface (API) to invoke commit, back out, and other recovery functions they require. The Messaging and Queuing resource manager uses the X/Open XA interface to the Transaction Manager resource manager, where appropriate.
- **Directory**. The Messaging and Queuing resource manager uses the Directory resource manager to find out which queue manager in the network owns a particular queue. When a queue is defined to a queue manager, the queue manager also optionally stores the name of the queue and the owning queue manager in the directory. When an application in the network opens that queue name, its queue manager uses the information in the directory to discover which queue manager owns the queue.
- **Security**. The facilities provided by the local operating system and the Security resource managers are used by the Messaging and Queuing resource manager to provide security for the resources it owns. These facilities are used, for example, to authenticate users, to guarantee that only messages from authorized users find their way onto particular queues, and to prohibit unauthorized programs from inspecting or changing the contents of queues or messages.

  The Security resource managers are also used to verify the connections between queue managers.
- **HTTP**. The Messaging and Queuing resource manager uses the computer graphics interface (CGI) of the HTTP resource manager to convert Hypertext Markup Language (HTML) requests to MQ messages, thus giving Web browsers access to Messaging and Queuing resource manager applications.
- **Collaboration**. The Messaging and Queuing resource manager provides interfaces to allow collaboration applications to invoke Messaging and Queuing resource manager functions, thus linking the collaborative and enterprise worlds. It is also possible for an MQ application to automatically insert data into a collaborative document.

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 valid intellectual property or other legally protectable 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 and/or other countries or both:

IBM
IBMLink
IMS
MQ
MQSeries
Open Blueprint
SOM
System Object Model

The following terms are trademarks of other companies:

C++	American Telephone and Telegraph Company, Incorporated
Lotus Notes	Lotus Development Corporation
X/Open	X/Open Company Limited

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.

GC23-3920-01