10. WebSphere MQ

10.1 Übersicht

10.1.1 Client/Server-Operation

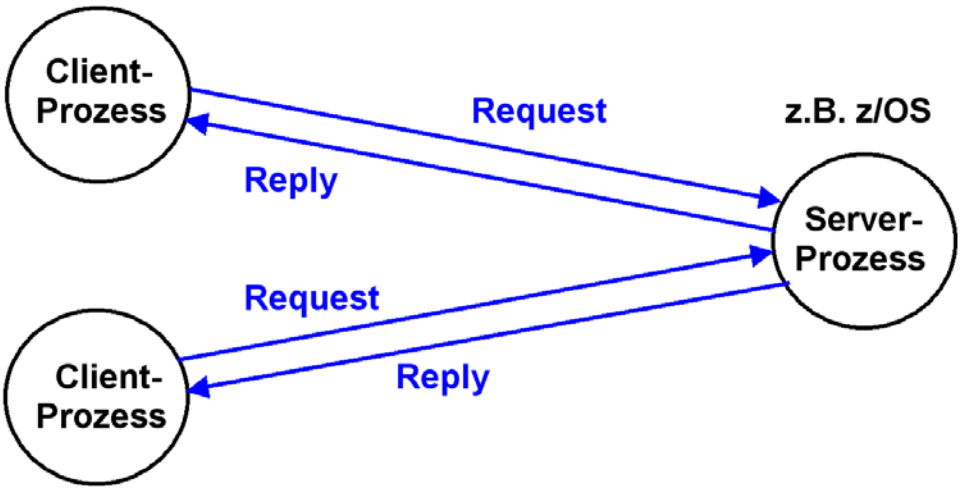


Abb. 10.1.1
Remote Procedure Call

In einer Client/Server Konfiguration bietet ein Server seine Dienste (Service) einer Menge a priori unbekannter Klienten (Clients) an. Client-Prozesse auf einem Client-System (z.B. einem PC) rufen Dienste (Services) auf einem Server-System auf. Ein Beispiel ist der Zugriff auf eine URL auf einem Apache-Webserver. Als Dienstleistung gibt Apache das gewünschte Dokument zurück. Ein Server bietet Dienste für jeden Client an, der seine Dienste anfordert.

Wir verwenden die folgenden Begriffe:

- **Client:** **ruft einen Dienst (Service) auf einem Server auf**
- **Server:** **Rechner, der Dienst-Software (als Service bezeichnet) ausführt**
- **Service:** **Software Prozess, der auf einem Server ausgeführt wird (möglicherweise auf mehreren Servern)**
- **Interaktion:** **Anfrage/Reaktion (Request/Reply)y**

Ein physischer Server kann gleichzeitig viele verschiedene Serverdienste (Services) anbieten. Beispielsweise bietet ein CICS Server multiple CICS Servies (identifiziert durch unterschiedliche TRIDs) einer Vielzahl von CICS Klienten an.

Der größere Teil aller Anwendungen in Wirtschaft und Verwaltung läuft auf Client/Server Systemen.

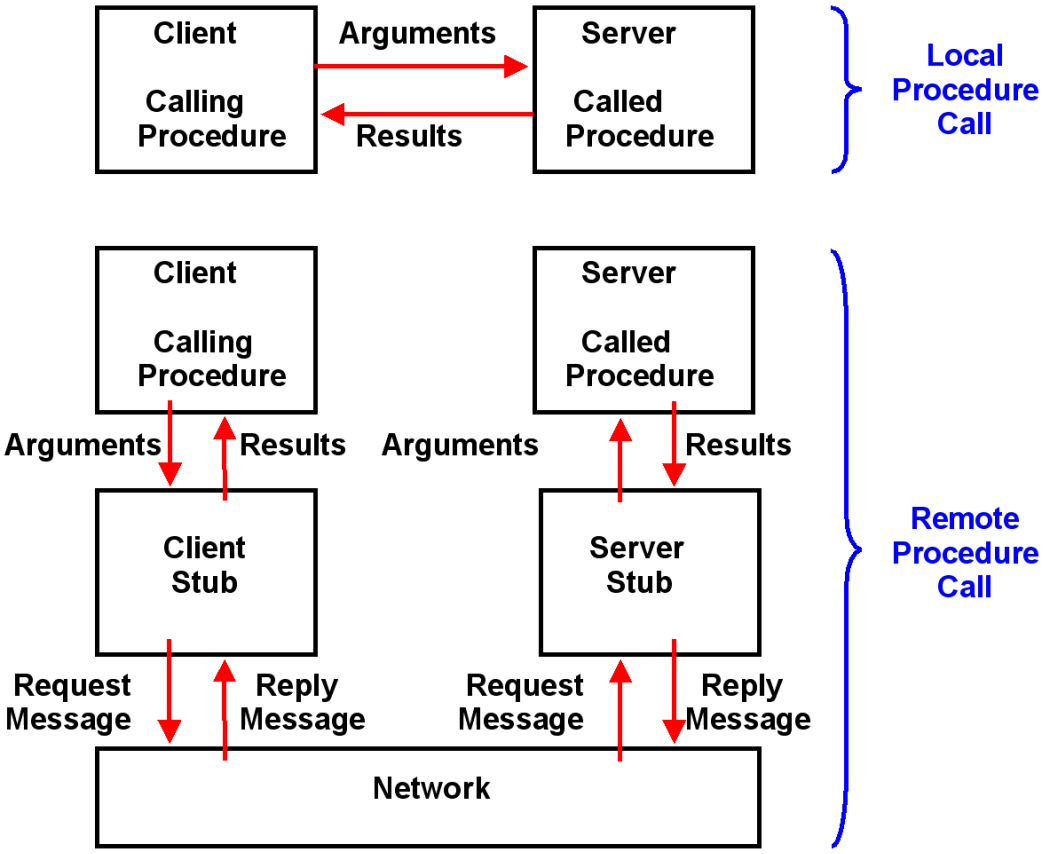


Abb. 10.1.2
Laufzeitumgebung des Remote Procedure Calls

Local Procedure Call

Eine aufgerufene Prozedur (Server) wird innerhalb des gleichen Adressraums wie die aufrufende Prozedur (Client) ausgeführt.

Remote Procedure Call

Client und Server laufen als zwei getrennte Prozesse, in der Regel auf zwei verschiedenen Rechnern. (Eine Konfiguration mit zwei getrennten Prozessen, die jeweils in ihrem eigenen virtuellen Adressraum auf dem gleichen Rechner laufen, ist ebenfalls möglich).

Beide Prozesse kommunizieren über Stubs. Stubs bilden lokale Prozeduraufrufe auf Netzwerk RPC Funktionsaufrufe ab.

Der klassische RPC benutzt die Bezeichnung Server Stub. Java und CORBA verwenden den Begriff Skeleton anstelle von Server-Stub. Die Bezeichnungen Server Stub und Skeleton sind austauschbar.

Stubs bzw. Skeletons implementieren eine RPC Runtime. Sie übersetzen Aufrufe der Client API in Sockets und dann in Nachrichten, die über das Netz übertragen werden.

Der Remote Procedure Call (RPC) erweckt den Anschein, als ob ein Client eine Prozedur aufruft, die sich in dem gleichen Adressraum befindet. In Wirklichkeit ruft die Client-Anwendung eine lokale Stub Prozedur auf, (anstelle des eigentlichen Codes, der die aufgerufene Prozedur implementiert). Der Client und Server haben jeweils einen eigenen Adressraum, und können sich (Normalfall) auf unterschiedlichen Maschinen befinden.

10.1.2 Funktionsweise des RPC

Stubs werden kompiliert und mit der Client-Anwendung verbunden. Statt dem eigentlichen Code, der die Remote-Verfahren implementiert, bewirkt der Client-Stub-Code:

- **Ermitteln der erforderlichen Parameter von dem Client-Adressraum.**
- **Übersetzen der Parameter in ein Standard-Format (Network Data Representation, NDR) für die Übertragung über das Netzwerk.**
- **Anruf von Funktionen in der RPC-Client-Laufzeitbibliothek, um die Anforderung und seine Parameter an den Server zu senden.**

Der Server führt die folgenden Schritte aus, um die Remote Procedure aufzurufen:

- **Die Server RPC-Laufzeitbibliothek Funktionen akzeptieren die Anfrage und rufen die Server-Stub Prozedur auf.**
- **Der Server Stub empfängt die Parameter von dem Netzwerk-Puffer und konvertiert sie von dem Netzwerk Übertragungsformat (NDR) in das Format, welches der Server benötigt.**
- **Der Server-Stub ruft die eigentliche Prozedur auf dem Server auf.**

Die Remote-Prozedur läuft jetzt. Sie erzeugt möglicherweise Ausgabeparameter und einen Rückgabewert. Wenn die Remote-Prozedur abgeschlossen ist, werden die resultierenden Daten in einer ähnlichen Folge von Schritten an den Client zurückgegeben:

- **Die Remote-Prozedur übergibt Ihre Ausgabeparameter und Rückgabewert Daten an den Server-Stub.**
- **Der Server-Stub wandelt die Daten auf das Format um, das für die Übertragung über das Netzwerk benötigt wird, und übergibt sie an die RPC-Laufzeitbibliothek.**
- **Die Server RPC-Laufzeitbibliothek überträgt die Daten über das Netzwerk an den Client Rechner-Computer.**

Der Client akzeptiert die Daten und übergibt sie an die aufrufende Funktion:

- **Die Client RPC-Laufzeitbibliothek erhält die Remote-Prozedur Rückgabewerte und gibt sie an den Client-Stub. weiter.**
- **Der Client-Stub wandelt die Daten aus dem NDR-Format in das Format um, das der Client-Rechner erwartet. Der Stub schreibt Daten in den Client-Speicher und liefert das Ergebnis an das aufrufende Programm auf dem Client.**
- **Die aufrufende Prozedur fährt fort, als ob sich die aufgerufene Prozedur im gleichen Adressraum befunden hätte.**

10.1.3 RPC Implementierungen

Sockets für TCP/IP , sowie LU 6.2 und APPC für SNA, sind Protokolle auf der niedrigsten Verbindungsstufe (Schicht 5 des OSI Modells, siehe Abschnitt 18.1).

Darüberliegende Protokolle (Schicht 6 und 7) werden allgemein als Remote Procedure Call (RPC) bezeichnet. Es existieren eine Vielzahl von RPCs, deren APIs (Application Programming Interface) in den meisten Fällen inkompatibel sind:

Klassische RPCs

- **SUN RPC (Standard in Solaris und vielen Linux Versionen),**
- **DCE RPC (unterstützt von Microsoft und IBM, besonders auch von z/OS),**
- **CPI-C (von IBM, ursprünglich für SNA entwickelt).**

Klassische RPCs sind immer noch populär aufgrund der überlegenen Leistung. Linux, Unix und z/OS-Systeme unterstützen sowohl den Sun RPC als auch den DCE RPC.

Spezialisierte RPC

- **DPL**

CICS Distributed Program Link, sehr performant, leistungsfähig und einfach zu bedienen, aber nur nutzbar zwischen CICS-Servern.

Objekt orientierte RPCs

- **Corba**
- **RMI**
- **DotNet der Fa. Microsoft, auch als .Net / COM+ / DCOM / ActiveX / DNA / ASP.NET bezeichnet. Die Abgrenzung der Begriffe ändert sich häufig. Wir benutzen durchgängig den Begriff DotNet. DotNet verwendet eine Erweiterung des DCE RPC.**

Corba, RMI und DotNet sind objektorientiert RPCs. RPC-Services werden als Objekte implementiert. Ein Client ruft eine Methode eines bestimmten Objekts auf.

Web Services RPC (SOAP RPC)

- **Der Web Service RPC benutzt das Simple Object Access Protocol (SOAP).**

Für den RPC sind Prozedur, Unterprogramm, Subroutine und Methode (objektorientierte Programmierung) austauschbare Begriffe. Eine Procedur implementiert z.B. einen serverseitigen Service. Ein Prozedur-Aufruf (request) besteht aus einer Nachricht, welche den Namen der aufgerufenen Prozedur sowie eine Liste der übergebenen Parameter enthält.

10.1.4 Synchroner und Asynchroner RPC

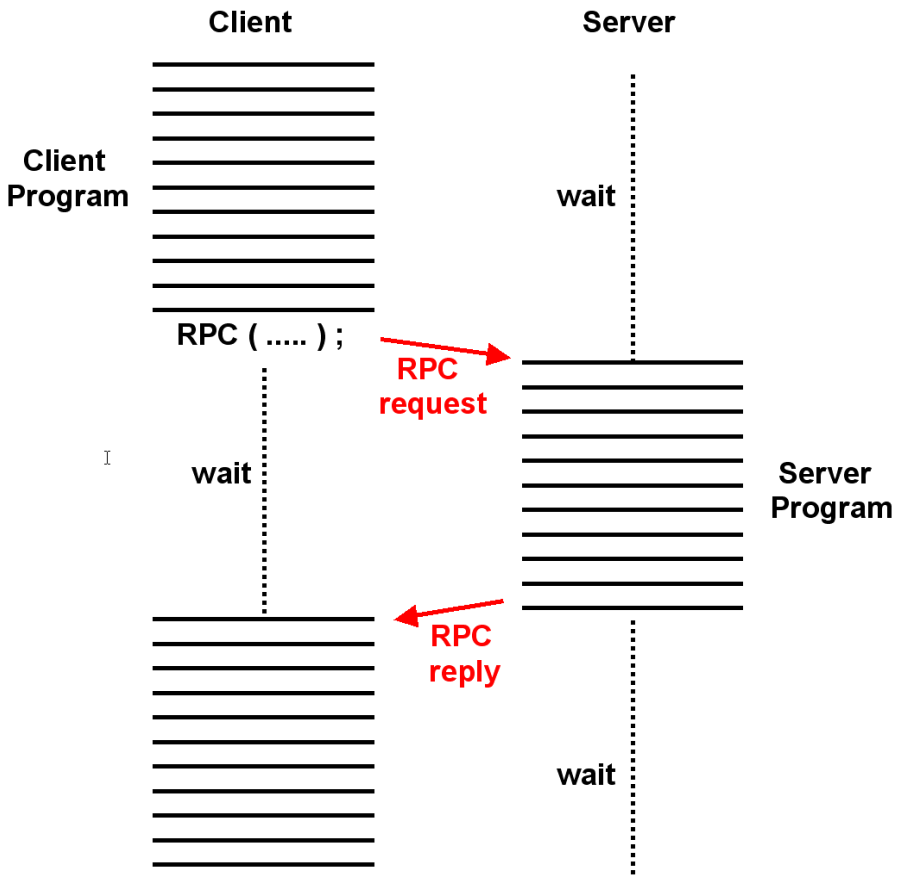


Abb. 10.1.3
Synchroner RPC

Die meisten RPCs werden als "synchrone RPCs" implementiert.

Der Klient sendet eine Nachricht an den Server. Das Eintreffen der Nachricht bewirkt den Aufruf eines von mehreren Services, die der Server bereitstellt. Der Klient wartet (blockiert), bis das Ergebnis des Service Aufrufes verfügbar ist.

Ein asynchrones RPC-Client-Programm wartet nicht auf eine Antwort von dem Server. MQSeries und MDB (Message Driven Beans, siehe Abschnitt 15.4.9) sind Beispiele für einen asynchronen Service Aufruf: Der Klient wartet nicht auf eine Antwort des Servers.

Electronic Mail ist ein Beispiel für einen eher primitiven asynchronen RPC.

10.1.5 Message Based Queuing

Message Based Queuing (MBQ) wird auch als Message Oriented Middleware (MOM) bezeichnet

MBQ ist ein Verfahren zur asynchronen Programm zu Programm-Kommunikation. Es kann verwendet werden, um einen asynchronen RPC implementieren. Programme bekommen die Fähigkeit, dass sie Informationen senden und empfangen können, ohne dass eine direkte Verbindung zwischen ihnen besteht. Programme kommunizieren, indem sie Nachrichten in Message-Queues hinterlegen, und Nachrichten von Message-Queues abrufen.

Führende MBQ Produkte sind:

- **IBM MQSeries, jetzt umbenannt in WebSphere MQ**
- **Microsoft MS Message Queue Server (MSMQ)**
- **Oracle BEA MessageQ**

MQSeries ist seit 1993 verfügbar und ist der unangefochtene Marktführer. Das Produkt ist für mehr als 35 Betriebssystem-Plattformen verfügbar, einschließlich AIX, DG / UX, HP-UX, i5/OS, Sequent, Sinix, Solaris, Tandem, TPF, TrueUnix, VMS / VAX, Windows und z/OS.

Anders als bei anderen MBQ Produkte ist Microsoft MSMQ nur auf Windows-Plattformen verfügbar. Es biete auch keine native Unterstützung für JMS (Java Messaging Service) an.

Apache ActiveMQ ist das populärste Open-Source MBQ Produkt.

Message Based Queuing ist attraktiv für die Integration von heterogenen Hardware, Software und Anwendungsumgebungen.

E-Mail wie SMTP, das Simple Mail Transport Protocol, und X.500 sind primitive MBQ Anwendungen. Merkmale sind:

- **Eine Nachricht wird gesendet.**
- **Die Nachricht kann (oder auch nicht) am Ziel ankommen.**
- **Der Empfänger ist (in der Regel) ein Mensch, der beschließt, die Mail, sofort oder später oder auch gar nicht zu beantworten.**

MBQ unterscheidet sich von elektronischer Post:

- **MBQ bietet ACID Eigenschaften, die sicherstellen, dass eine Nachricht genau einmal ausgeliefert wird.**

Ein MBQ Nachricht wird durch ein Anwendungsprogramm empfangen, das weiß, was damit zu tun ist.

Message Based Queuing ist eine Methode der Programm-zu-Programm-Kommunikation. Programme innerhalb einer Anwendung kommunizieren miteinander durch das Schreiben und Abrufen von anwendungsspezifischen Daten (Nachrichten) zu/von Warteschlangen (Queues), ohne über eine private, dedizierte logische Verbindung miteinander verknüpft zu sein. Messaging bedeutet, dass Programme miteinander durch Senden von Daten in Nachrichten kommunizieren und nicht durch einen direkt Aufruf.

Queuing bedeutet, dass Programme über Nachrichten kommunizieren, die in Queues gespeichert werden. Anders als bei einem synchronen RPC, müssen Programme, die über Queues kommunizieren, nicht gleichzeitig ausgeführt werden. Ein Java-Objekt kann eine Methode eines anderen Java-Objekts durch Senden einer Nachricht aufrufen. Dies ist jedoch eine synchronen und keine asynchrone Kommunikation.

Beim asynchronen Messaging fährt das sendende Programm mit seiner Verarbeitung fort, ohne auf eine Antwort seiner Nachricht zu warten. Im Gegensatz dazu wartet ein synchrones RPC Programm auf eine Antwort, ehe es die Verarbeitung fortgesetzt.

Beim Message Queuing muss sich der Programmierer nicht darum kümmern, ob das Zielprogramm beschäftigt oder nicht verfügbar ist. Er macht sich noch nicht einmal Gedanken, ob der Server heruntergefahren oder die Verbindung zum Server gestört ist. Der Programmierer sendet Nachrichten an eine entfernte (remote) Queue, die einem Anwendungsprogramm zugeordnet ist. Letzteres kann zu diesem Zeitpunkt verfügbar sein oder auch nicht. WebSphere MQ kümmert sich um den Transport zu der Zielanwendung. Ggf. startet es diese sogar.

Für den Anwender ist das zugrunde liegende Kommunikationsprotokoll transparent.

10.1.6 WebSphere MQ

Das IBM MQSeries Software-Produkt ist seit langer Zeit verfügbar. Andere Hersteller vertreiben alternative MBQ Produkte. IBM MQSeries hat jedoch im Markt eine ähnlich dominierende Bedeutung wie CICS.

Kürzlich hat IBM entschieden, MQSeries in WebSphere MQ umzubenennen. Sowohl die alte Bezeichnung MQSeries, der neue Begriff WebSphere MQ und die Kurzform MQ werden mehr oder weniger synonym verwendet.

IBM WebSphere entstand als IBMs Java Application Server-Produkt, heute bekannt unter dem Namen WebSphere Application Server (WAS). IBM hat beschlossen, den Namen WebSphere wiederzuverwenden, um eine Reihe von mehr oder weniger verwandten Software-Produkten (einschließlich WAS) zu kennzeichnen. Neben dem WebSphere Application Server enthält die WebSphere-Familie Produkte wie den WebSphere Designer, den WebSphere Integration Developer, WebSphere Process Server oder WebSphere Portal Server. Es gibt über 200 Software-Produkte in der WebSphere-Familie. Einige der Produkte sind gut miteinander integriert, andere nicht. WebSphere MQ ist ein ziemlich eigenständiges Add-on, das unabhängig von dem WebSphere Java Application Server eingesetzt werden kann.

Die meisten Personen, wenn sie den Namen WebSphere benutzen, meinen damit den WebSphere Application Server (WAS).

Es gibt 2 Versionen von WebSphere MQ:

- **Integrierte Version,**
- **Unabhängige (nicht integrierte) Version, die nur wenige Beziehungen mit dem Rest der WebSphere-Produktfamilie hat.**

Viele Personen benutzen immer noch den Namen MQSeries für die unabhängige (nicht integrierte) Version von WebSphere MQ. Der WebSphere Application Server (WAS) Implementierung des Java Message Service (JMS)-Standards verwendet hierfür eine integrierte Version von WebSphere MQ.

10.1.7 Program-to-Program Communication

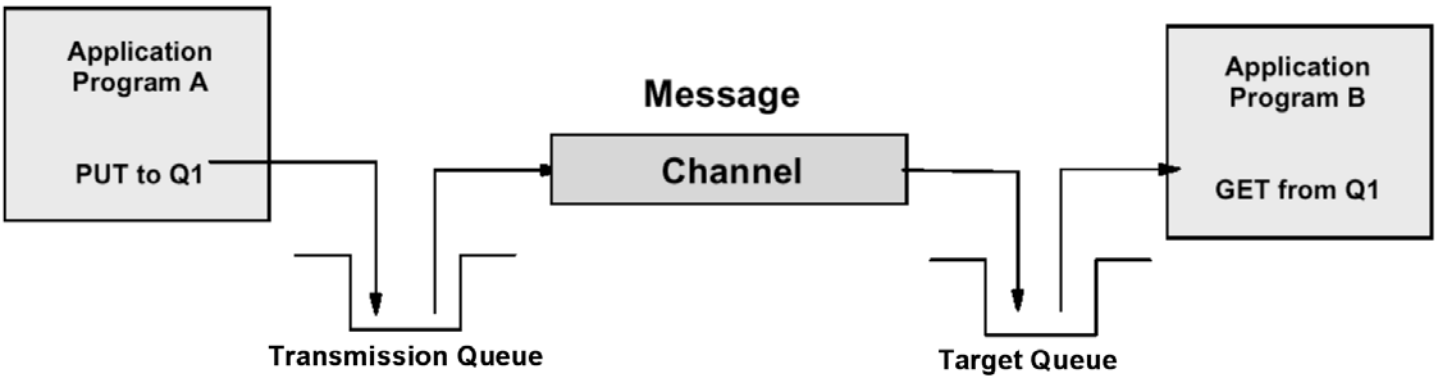


Abb. 10.1.4
MQSeries Operation

Ein Anwendungsprogramm kann Nachrichten an einen anderen Anwendungsprogramm senden, das auf einem entfernten System, wie etwa einem Server oder einem Host läuft.

Der sendende Anwendungsprogramm A speichert die Nachricht in eine lokale Transmission Queue mit dem MQPUT Befehl . Von dort aus wird die Nachricht über eine "Message Channel" zu einer entfernten Target Queue übertragen. Das empfangende Anwendungs-Programm B kann nun die Nachricht aus der Target Queue mit dem MQGET Befehl zu einem Zeitpunkt seiner Wahl abrufen.

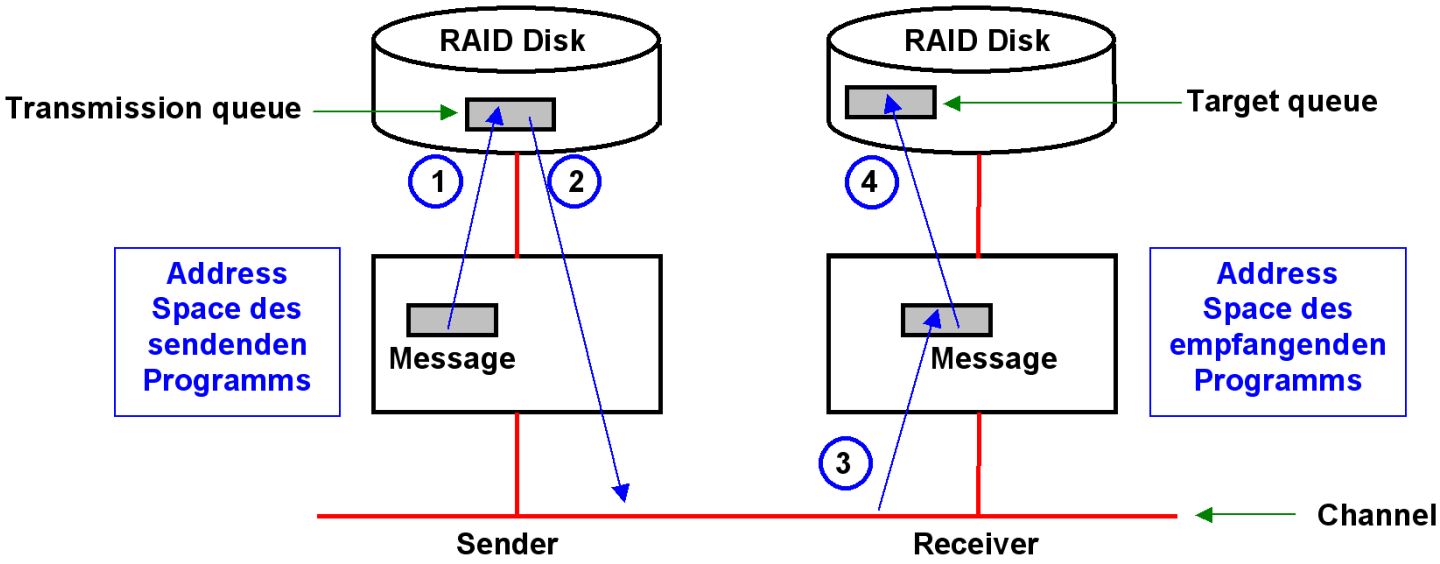


Abb. 10.1.5
Persistente und Non-Persistente Nachrichten

Die "Message Channel" (oder Channel) ist ein MQSeries Konstrukt, das benutzt wird, um Nachrichten von einer Transmission Queue in eine Target Queue zu übertragen. Es ist ein Schicht 5 Protokoll, vergleichbar mit Telnet, 3270, http oder SMTP. Es benutzt TCP/IP (oder SNA) als den darunter liegenden Schicht 4 Transportmechanismus.

WebSphere MQ unterscheidet zwischen persistenten und nicht-persistente Nachrichten. Die Übertragung von persistenten Nachrichten ist gewährleistet, und überlebt Systemausfälle, ähnlich wie Nachrichten, die CICS in seine Log Files schreibt. Nicht-persistente Nachrichten können nach einem Systemausfall nicht wiederhergestellt werden.

Die Persistenz wird mit Hilfe des Speicherns der Nachricht in einem lokalen RAID Festplatten-Subsystem erreicht.

Persistente Nachrichten implementieren eine Kommunikation mit ACID-Eigenschaften (die Zustellung der Nachricht genau einmal wird garantiert). Die Nachricht wird dauerhaft (persistent) durch den Absender gespeichert, ehe sie gesendet wird. Der Empfänger speichert die Nachricht ebenfalls persistent, ehe sie verwendet wird, und bestätigt den Empfang an den Absender.

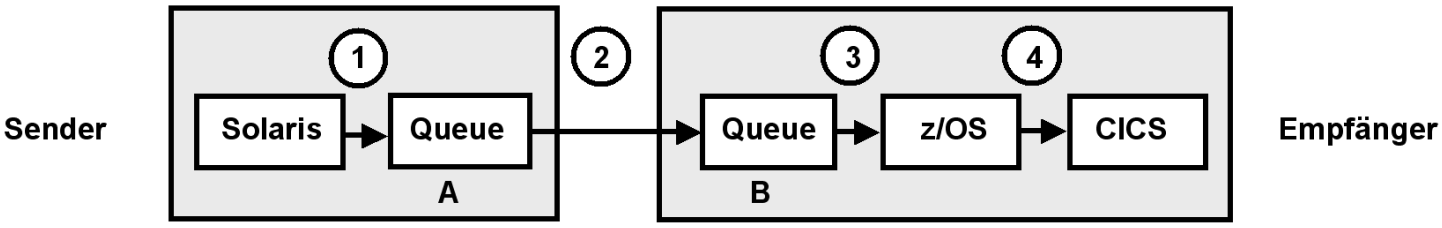


Abb. 10.1.6
Datenübertragung unter ACID Bedingungen

Die persistente Nachrichtenübertragung erfolgt in mehreren Schritten.

Schritt 1: Der Sender speichert die Nachricht persistent in der lokalen Transmission Queue A.

Schritte 2 - 3: Die Nachricht wird über das Netzwerk an die Target Queue B übertragen. Wenn die Übertragung erfolgreich war, sendet der Empfänger eine Bestätigung. Wenn erfolglos, wird der Vorgang beliebig oft wiederholt.

Schritt 4: Der empfangende Anwendungsprogramm (ein CICS Programm in diesem Beispiel) kann nun die Nachricht aus Queue B zu einem Zeitpunkt seiner Wahl abrufen. Dieser Zeitpunkt kann beliebig weit in der Zukunft liegen.

Wenn ein Anwendungsprogramm eine Nachricht in einer Queue speichert, gewährleistet MQSeries, dass die Nachricht:

- **Sicher gespeichert wird,**
- **recoverable ist, und**
- **nur einmal, und genau einmal (exactly once) an die empfangende Anwendung ausgeliefert wird.**

Die Kommunikation ist einseitig.

10.1.8 Middleware

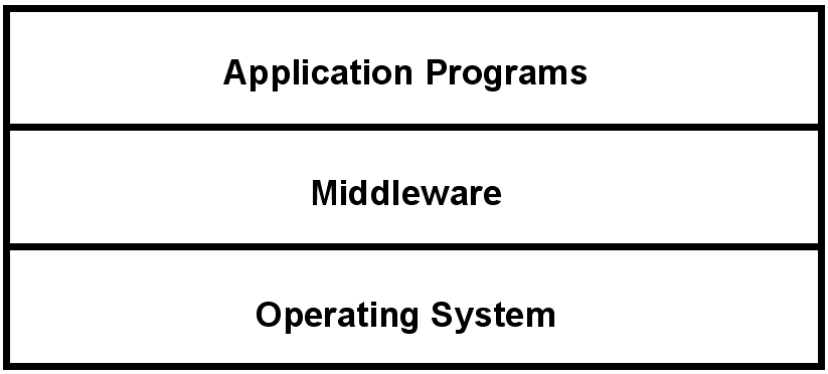


Abb. 10.1.7
Middleware ist eine Laufzeitumgebung für Anwendungssoftware

Der CICS Nucleus ist eine Runtime-Ausführungsumgebung, welche die gleichzeitige Ausführung von mehreren unabhängigen Anwendungsprogrammen (Transaktionen) ermöglicht. Ähnliche ermöglicht ein RPC-Server oder CICS Server die gleichzeitige Ausführung von mehreren unabhängigen Anwendungsprogrammen. Der WebSphere Application Server (WAS) ist eine Laufzeit-Ausführungsumgebung, die gleichzeitige Ausführung von mehreren unabhängigen Java-Anwendungsprogrammen ermöglicht, unter Benutzung von Einrichtungen wie einem Java Servlet-Container und einem Enterprise Java Bean (EJB) Container.

Der Queue Manager ist eine Runtime-Ausführungsumgebung, die die gleichzeitige Ausführung von mehreren unabhängigen WebSphere MQ Anwendungsprogrammen ermöglicht.

Software wie der CICS Nucleus, der RPC-Server, der Web Application Server und der Queue Manager wird gemeinhin als Middleware bezeichnet. Sie alle stellen eine gemeinsame Laufzeitumgebung für mehrere unabhängige Anwendungsprogramme zur Verfügung.

10.1.9 Queue Manager

Der Kern von WebSphere MQ ist der (Message) Queue-Manager (MQM), ein WebSphere MQ Run-Time Programm.

Der Queue Manager stellt eine Laufzeitumgebung für MQ Anwendungen dar, ähnlich wie der CICS Nucleus eine Laufzeitumgebung für CICS Anwendungen darstellt.

Ehe ein Anwendungsprogramm WebSphere MQ auf einem Rechner verwendet, muss ein Queue Manager gestartet werden. Der Queue-Manager besitzt und verwaltet die Ressourcen, die von WebSphere MQ verwendet werden. Zu diesen Ressourcen gehören:

- **Page Sets, die WebSphere MQ Objektdefinitionen und Message Daten enthalten,**
- **Log Files, die verwendet werden, um Nachrichten und Objekte wiederherzustellen falls ein Queue Manager Fehler auftritt,**
- **Prozessor Speicherplatz,**
- **Anschlüsse, über die verschiedenen Anwendungsumgebungen (z.B. CICS, IMS, Batch) auf die WebSphere MQ API zugreifen können,**
- **Den WebSphere MQ Channel-Initiator (später besprochen), der die Kommunikation zwischen WebSphere MQ auf einem sendenden und einem empfangenden Rechner ermöglicht,**

Der Queue-Manager hat einen Namen, und Anwendungen können sich mit einem entfernten Queue Manager über seinen Namen verbinden.

Die Aufgabe des Queue Managers ist es, Warteschlangen und Nachrichten für Anwendungen zu verwalten. Es stellt eine API, die Message Queuing Interface (MQI), zur Kommunikation mit Anwendungen zur Verfügung. Anwendungsprogramme benutzen Funktionen des Queue Managers über API-Aufrufe (Commands). Zum Beispiel legt der MQPUT API-Aufruf eine Nachricht in eine Queue, die durch einentferntes Programm über den API-Aufruf MQGET gelesen werden kann. Dieses Szenario ist in Abb. 10.1.9 dargestellt.

Der Queue Manager kann Nachrichten für den Fall von Anwendung- oder Systemausfällen zwischenspeichern. Nachrichten werden in einer Warteschlange gespeichert, bis eine Empfangsbestätigung durch die MQI eintrifft.

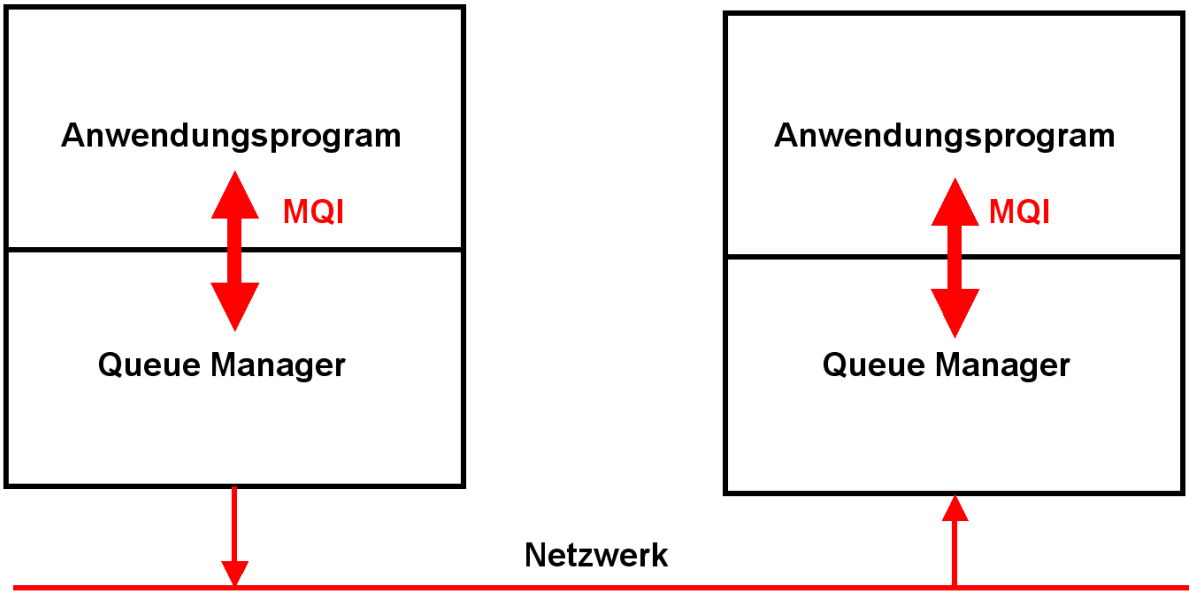


Abb. 10.1.8
MQI als Schnittstelle zwischen Queue Manager und Anwendungsprogramm

WebSphere MQ ermöglicht es Anwendungsprogrammen, miteinander über ein Netzwerk ungleicher Komponenten, wie unterschiedlicher Prozessoren, Subsysteme, Betriebssysteme, Kommunikationsprotokolle und Programmen, die in unterschiedlichen Programmiersprachen geschrieben sind, zu kommunizieren.

Abb. 10.1.8 zeigt die wichtigsten Teile eines WebSphere MQ-Anwendung zur Laufzeit. Anwendungsprogramme benutzen WebSphere MQ-API-Aufrufe, die Message Queue Interface (MQI), um mit einem Queue Manager zu kommunizieren. Die MQI ist eine konsistente Application Program Interface (API) für viele, sehr unterschiedliche Plattformen. Der Queue Manager ist die Laufzeit-Umgebung (runtime environment) von WebSphere MQ.

Anwendungsprogramme können in verschiedenen Programmiersprachen geschrieben werden, einschließlich Java. Der gleiche Queuing-Mechanismus gilt für alle Plattformen. Dies gilt auch für die derzeitig 13 API-Aufrufe der MQI.

10.1.10 Message Queuing Interface

Die wichtigsten Message Queuing Interface (MQI) Befehle sind:

MQCONN	**Verbinden (Connect) mit einem (in der Regel entfernten) Queue Manager**
MQOPEN	**Öffnen (Open) einer spezifischen Queue**
MQPUT	**Eine Message in eine Transmission Queue stellen**
MQGET	**Eine Message aus einer Target Queue auslesen**
MQCLOSE	**Schließen (Close) einer Queue**
MQDISC	**Verbindung zu einem Queue Manager auflösen (Disconnect)**

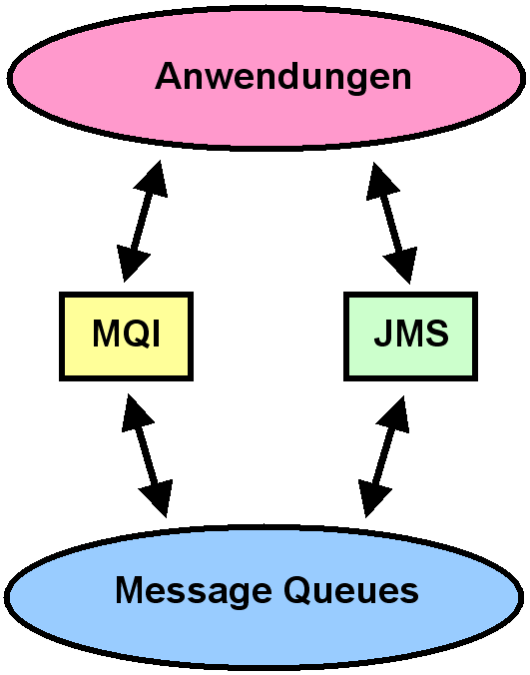


Abb. 10.1.9
JMS als Alternative zu MQI für Java Anwendungen

An Stelle der MQI kann Websphere MQ die „Java Message Service“ (JMS) Interface benutzen. Die beiden wichtigsten APIs sind:

Message Queue Interface (MQI)
- **MQI Funktionen sind verfügbar in den folgenden Sprachen:**
  z/OS Assembler, C/C++, COBOL, Lotus Script, Java, PL/1, VisualBasic, C#, viele andere.

Java Message Service (JMS)
- **Java Standard, der von WebSphere MQ unterstützt wird**
- **Abstracts WebSphereMQ Details**

JMS ist eine von mehreren Schnittstellen für JEE / Enterprise Java Beans.

Die älteste und am weitesten verbreitete API ist die MQI. JMS kann nur von Java-Anwendungen verwendet werden.

10.1.11 Queue Transmission

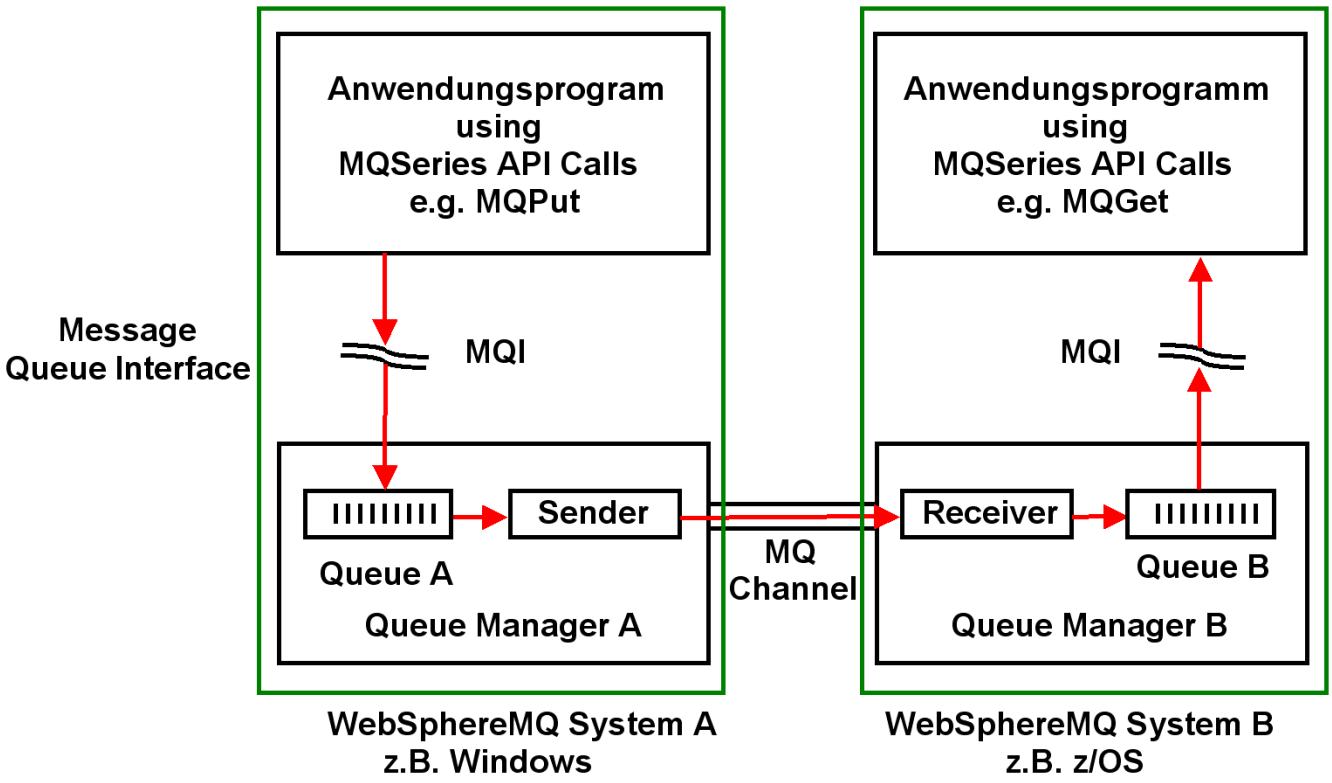


Abb. 10.1.10
WebSphere MQ Operation

Das Anwendungsprogramm stellt eine Nachricht in die Queue A (Transmission Queue), mit Hilfe des MQPUT Calls, ein Bestandteil der Message Queue Interface (MQI). Queue A ist Bestandteil von Queue Manager A. Dieser benutzt sie, um die Nachricht über den MQ-Channel an den Empfänger von Queue Manager B zu senden. Queue Manager B besitzt Queue B (Target Queue, auch als Request Queue, Application Queue, oder Destination Queue bezeichnet), wo die Nachricht gespeichert wird. Das empfangende Anwendungsprogramm kann die Nachricht aus der Target Queue B zu einem von ihm gewählten Zeitpunkt abrufen, mit Hilfe der MQI, beispielsweise mit dem MQI Befehl MQGET.

10.1.12 Sneakernet

Die Infrastruktur eines großen Unternehmens besteht neben dem Mainframe aus einer Vielzahl unterschiedlicher Rechner mit unterschiedlichen Betriebssystemen. Sehr häufig dienen die Ergebnisse auf einem Rechner als Input für den nächsten Rechner.

In der Vergangenheit war es üblich, die Ergebnisse eines Rechners in der Form von Magnetbändern oder Lochkarten zu Fuß zum nächsten Rechner zu tragen. Dies geschah durch einen menschlichen Operator, daher der Ausdruck „Sneakernet“. Noch in den 90er Jahren wurden jeden Abend umfangreiche Daten, die im Werk Bremen der Firma Daimler Benz anfielen, in der Form von Magnetbändern in einen PKW geladen, um am nächsten Morgen im Werk Sindelfingen verarbeitet zu werden.

Andrew S. Tanenbaum: „Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway.“

WebSphere MQ ist ein perfekter Ersatz für das Sneakernet: Einheitliche Schnittstellen, Sicherheit, sowie asynchrones Verhalten.

10.1.13 Unterstützung einer Service Oriented Architecture

WebSphere MQ ist eine Schlüsselkomponente bei der Umsetzung einer Enterprise Application Integration (EAI) oder einer Service-Oriented Architecture (SOA)-Strategie, in dem es das Messaging-Backbone für über 80 verschiedene Platformen bereitstellt. Die wachsende Bedeutung von EAI und SOA und das Wachstum von Web Services und anderen Verbindungs-Mechanismen sind eindeutig wichtige Entwicklungen.

Einzelheiten hierzu in Band 2, Abschnitt 20.4.

10.2 Queues und Channels

10.2.1 Messages

Abb. 10.2.1
WebSphere MQ Nachrichtenformat

Eine Nachricht besteht aus zwei Teilen:

1. **Message Header (Nachrichtenkopf),**
2. **Daten, die von einem Programm zu einem anderen geschickt werden.**

Der Message Header beinhaltet in jedem Fall einen „Message Descriptor“ (MQMD), kann aber weitere Information beinhalten. Der Message Descriptor identifiziert die Nachricht (Message-ID) und enthält Steuerinformationen (auch als Attribute bezeichnet) wie Nachrichtentyp, Name der Target Queue, Expiration Time, Korrelations-ID, die Priorität und den Namen der lokalen Queue für eine mögliche Antwort.

Eine Nachricht kann bis zu 100 MByte lang sein.

10.2.2 Was ist eine Queue

Die Begriffe “Queue” und “Message Queue” sind austauschbar.

Eine Queue ist eine Datenstruktur die verwendet wird, um Nachrichten zu speichern, bis sie von einer Anwendung abgefragt werden. Es ist einfach ein Speicherplatz der Nachrichten enthält. Die Nachrichten werden entweder von einem Anwendungsprogramm in die Queue gesetzt, oder durch einen Queue Manager als Teil seiner Operation.

Der Queue-Manager ist für die Queues verantwortlich, die er besitzt, für die Speicherung aller Nachrichten, die er empfängt, und das Auslesen von Nachrichten als Reaktion auf eine Anforderung eines Anwendungsprogramms.

Es gibt lokale Queues, die im Besitz des lokalen Queue-Managers sind, und entfernte (remote) Queues (z.B. eine Target Queue), die einem anderen Queue-Manager gehören.

Eine Queue ist ein benanntes Objekt (bis zu 48 Zeichen lang, so lang wie ein z/OS-Dataset-Name), der von einem Queue Typ definiert wird. Der Queue Typ kann eine lokale Queue spezifizieren. Alternativ kann es eine lokale Definition einer Target-Queue sein, die sich auf einem entfernten Rechner mit einem entfernten Queue-Manager befindet.

Lokale Queues sind physische Queues; der Queue-Manager kann eine Nachricht in ihnen speichern. Ein Queue Manager behandelt nicht lokale Queues als Beschreibungen von Queues, die Definitionen enthalten, die durch einen entfernten lokalen Queue-Manager verwendet werden können.

Der MQPUT Befehl speichert eine Nachricht in der Transmission Queue. Es definiert auch in seinem Message Descriptor (MQMD) eine entfernte Target-Queue (die MQI Syntax nennt dies ein "Remote-Queue-Object"). Diese Informationen werden von dem lokalen Queue-Manager verwendet, um die Nachricht an die entsprechende Target Queue eines anderen entfernten Queue-Managers zu übertragen.

10.2.3 Was ist ein Channel?

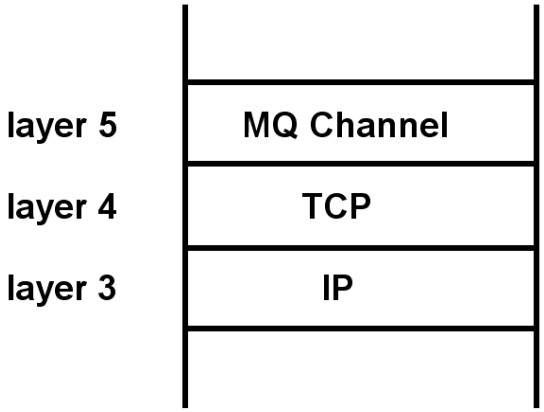


Abb. 10.2.2
Der MQ Channel ist eine höhere Abstraktionsebene als TCP/IP (oder SNA)

Alle Netzwerk-Kommunikation in WebSphere MQ wird über einen "Channel" (auch als "Message Channel" oder "MQ Channel" bezeichnet) durchgeführt. Insbesondere die Übertragung von Nachrichten von einer Transmission Queue an eine Target Queue eines entfernten Queue Managers erfolgt über einen Channel..

Ein Channel ist eine Punkt-zu-Punkt-Verbindung, die zwei verschiedene Queue-Manager miteinander verknüpft. Es ist eine logische Verbindung in der Schicht 5 des OSI-Modells. Telnet, 3270, SMTP, http und ftp sind weitere Beispiele für Schicht 5 Protokolle. Ein Channel wird durch ein sendendes und ein empfangendes- Programm realisiert, die über ein Netzwerk miteinander verbunden sind.

Channels verbergen die zugrundeliegenden Schicht 3 und Schicht 4 Kommunikationsprotokolle (zum Beispiel TCP und IP) vor den Anwendungen. Anwendungsprogrammierer brauchen nicht die physische Adresse des Programms zu kennen, dem sie eine Nachricht senden. Sie stellen ihre Nachrichten in einer Queue, und überlassen es dem Queue-Manager, sich um die Adresse des Ziel-Rechners zu kümmern, und darum, wie die Nachricht dorthin kommt. WebSphere MQ weiß, was zu tun ist, wenn das entfernte System nicht verfügbar ist oder das Zielprogramm nicht läuft oder beschäftigt ist.

Der Begriff "Channel" kann verschiedene Dinge in verschiedenen Umgebungen bedeuten. Eine MQ-Channel ist etwas ganz anderes als z.B. ein FICON-Channel (Abschnitt 5.2).

10.2.4 MQ Message Channel

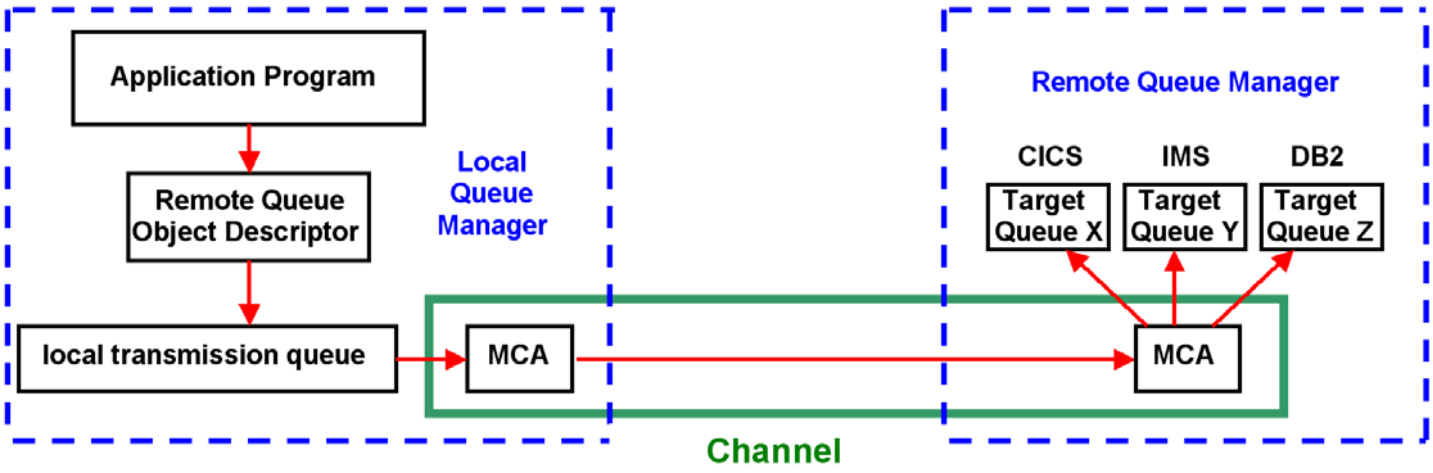


Abb. 10.2.3
Eine MQ Channel Verbindung wird mittels 2 MCAs erstellt

Wie funktioniert die Nachrichtenübertragung?

Ein Message Channel Agent (MCA) ist ein Programm, welches das Senden und Empfangen von Nachrichten steuert. Es gibt einen MCA an jedem Ende eines Channels. Ein MCA nimmt Nachrichten aus der Transmission Queue und legt sie auf die Kommunikationsverbindung. Der andere MCA empfängt Nachrichten und liefert sie an eine Target Queue des entfernten Queue Managers. Einem MQ-Channel wird durch ein Paar von MCAs und das Kommunikationsnetz dazwischen implementiert.

Jeder MQ Message Channel besteht aus zwei Message Channel Agents (MCA), je einem auf dem sendenden und dem empfangenden Rechner. Jeder MCA übernimmt eine der folgenden Rollen:

- **Der sendende MCA öffnet eine bestimmte Transmission Queue für exklusiven Input. Dies bedeutet, keine zwei Message Channels können konfiguriert werden, um Nachrichten von der gleichen Transmission Queue zu übernehmen. Der MCA erhält Nachrichten von einer bestimmten Transmission Queue und sendet sie an den Partner MCA.**

- **Der empfangende MCA empfängt Nachrichten von dem sendenden MCA. Für jede Nachricht, entfernt er den Transmission Queue Message Deskriptor aus der Nachricht und liest deren Inhalt. Er öffnet die Target Queue, die in dem Message Deskriptor für diese Nachricht angegeben ist und schreibt dann die Nachricht in die Target Queue.**

Bitte beachten: Ein Channel ist mit einer einzigen Transmission Queue verbunden, kann aber mit mehreren Target Queues verbunden sein.

MQSeries benutzt den Begriff „Objekt“, um damit Queues und Channels zu bezeichnen.

Um Nachrichten von einem lokalen Queue-Manager an einen entfernten Queue-Manager zu senden, muss der lokale Queue Manager eine Kommunikationsverbindung zum entfernten Queue Manager aufsetzen (zum Beispiel mit dem MQCONNECT Befehl). Der empfangende MCA muss auf dem entfernten Queue-Manager gestartet sein, um Nachrichten über die Kommunikationsverbindung zu empfangen. Diese Einweg-Verbindung, bestehend aus dem sendenden MCA, der Kommunikationsverbindung, und dem empfangenden MCA, wird als Channel bezeichnet. Der sendende MCA übernimmt Nachrichten von einer Transmission Queue und sendet sie über einen Channel an den empfangenden MCA. Dieser empfängt die Nachrichten und legt sie auf eine Target Queue.

Der Remote „Message Queue Object Descriptor“ (MQOD) enthält die Message Descriptor Information, die für den Transmission Queue Header benötigt wird.

MQ-Channels sind unidirektional. Für die bidirektionale Kommunikation müssen zwei Channels (ein Channel Paar) bestehend aus einem Sender Channel und einem Empfänger Channel definiert werden.

10.2.5 Target Queue Definition

Wenn Sie ein WebSphere MQ-Queue öffnen, führt der MQOPEN Befehl eine Namensauflösung durch

Die MQPUT Kommando eines Anwendungsprogramme schreibt eine Nachricht in einer virtuellen Remote Queue, die nur eine Definition einer Target Queue auf einem entfernten Rechner darstellt (remote Queue-Definition). Es ist die Aufgabe des lokalen Queue-Managers, die Nachricht in eine lokale Transmission Queue zu laden, und an die richtige Target Queue weiterzuleiten. Dieses Target Queue ist lokal für einen einem entfernten Queue-Manager auf einem Remote System.

Auf dem entfernten System (und seine Queue-Manager) können mehrere Target Queues existieren, die mit unterschiedlichen Anwendungsprogrammen assoziiert sind, zum Beispiel einer CICS-Transaktion, einer IMS-Transaktion oder einer DB2 Stored Procedure.

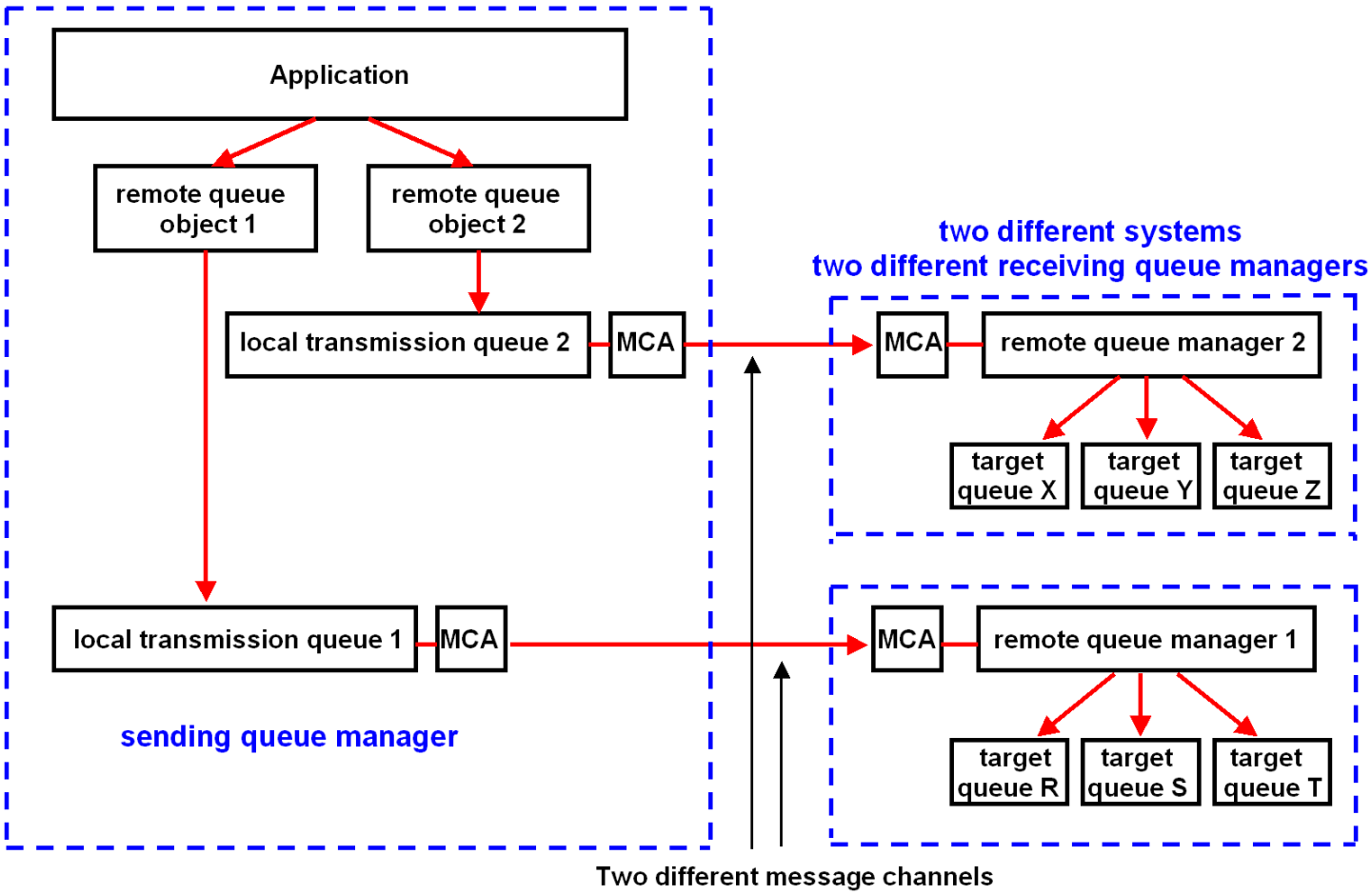


Abb. 10.2.4
Für jede Channel Verbindung wird eine getrennte lokale Transmission Queue benötigt

Eine einzelne Transmission Queue und ein einzelner MQ Message Channel kann mehrere Target Queues auf einem bestimmten Remote-System bedienen. Der Message Deskriptor in der Nachricht, die an das entfernte System gesendet wird, definiert die Target Queue, welche die Nachricht empfangen soll.

Wie in der folgenden Abbildung dargestellt, kann ein lokales System gleichzeitige mehrere WebSphere MQ Verbindungen zu mehreren Remote-Systemen aufrechterhalten. Jede Verbindung benötigt eine separate Transmission Queue und den damit verbundenen MQ Message Channel. Die MQPUT Kommando eines Anwendungsprogramme schreibt eine Nachricht an eine von mehreren entfernten Target Queues. Der lokale Queue-Manager stellt die Nachricht in die entsprechende Transmission Queue. Damit wird die Nachricht automatisch an den richtigen Remote Queue Manager ausgeliefert.

10.2.6 WebSpere MQ Netzwerk

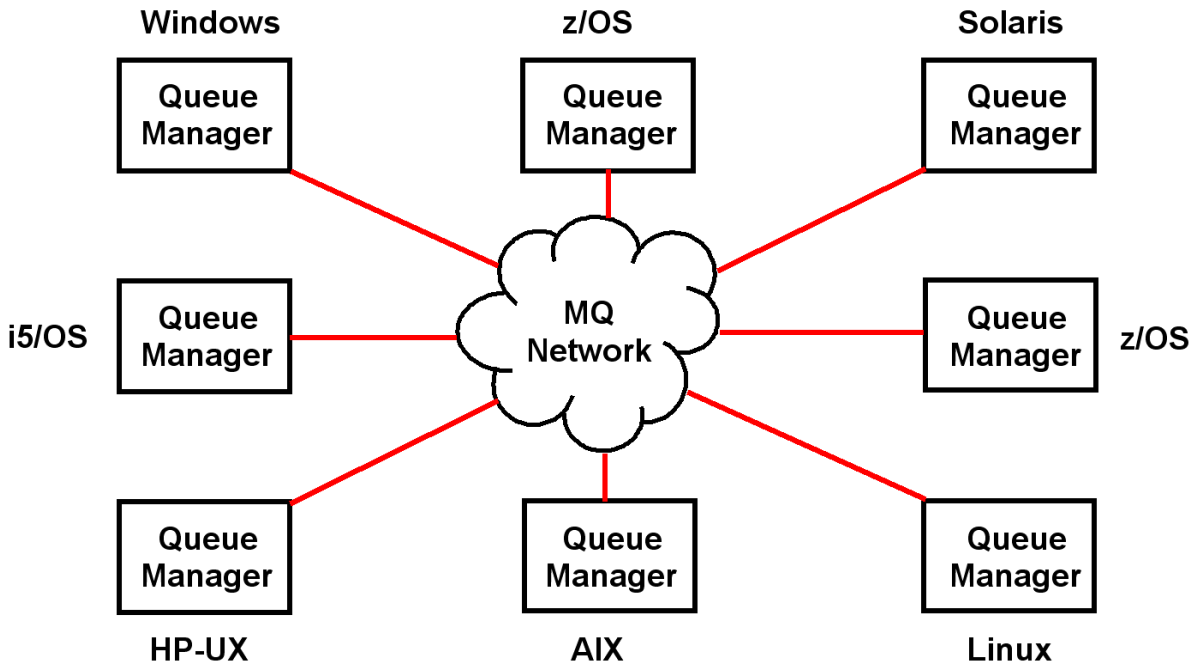


Abb. 10.2.5
Ein weltweites MQ Netzwerk kann Router ähnliche Funktionen haben

Alle Queue Manager in einem Netzwerk von WebSphere MQ Systeme müssen eindeutige Namen haben. Sie ordnen die Namen den Queue Managern zu, wenn Sie das WebSphere MQ-Netz definieren und installiere, genau so, wie Sie IP-Adressen definieren, wenn Sie eine IP-Netzwerk aufzubauen.

Denken Sie daran, genauso wie bei CICS oder Apache, WebSphere MQ ist es egal, auf welcher Betriebssystem-Plattform es läuft.

10.2.7 z/OS Channel Initiator

Unter z/OS läuft WebSphere MQ als z/OS-Subsystem in seinen eigenen Adressraum (Adressraum # 2 in dem in Abb. 10.2.6 gezeigten Beispiel). Innerhalb des Subsystems wird der Queue-Manager durch das Ausführen eines JCL-Prozedur gestartet. Diese spezifiziert die z/OS-Datasets, die Informationen über die Log Files, die Object Definitionen und Message Data (page sets) enthalten. Alle Queue-Manager in Ihrem Netzwerk müssen eindeutige Namen haben, auch wenn sie auf verschiedenen Systemen und Plattformen laufen.

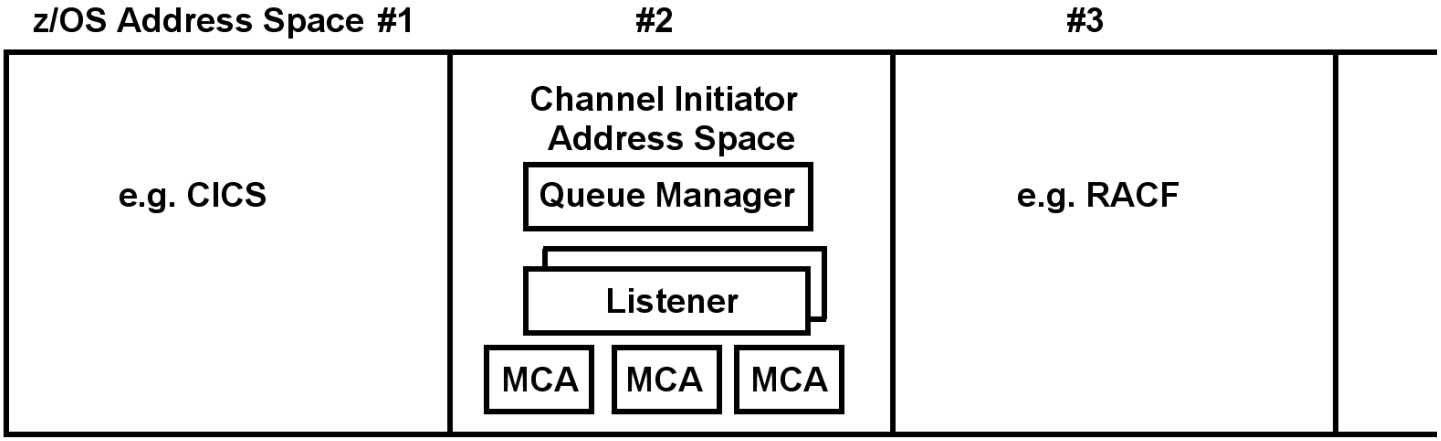


Abb. 10.2.6
Channel Initiator und Listener

Ein Channel Initiator ist ein Bestandteil eines Queue-Managers. In z/OS existiert ein Channel Initiator für jeden Queuemanager (es können mehrere Queue-Manager in getrennten Regionen existieren). Der Channel Initiator läuft zusammen mit seinen Queuemanager innerhalb desselben z/OS Adressenraums. Er beherbergt alle Message Channel Agents (MCAs) für einen Queue-Manager. Tausende von MCA Prozessen können innerhalb des Channels Initiators gleichzeitig laufen. Der Channel Initiator überwacht die vom System definierte Queue SYSTEM.CHANNEL.INITQ, die Initiation Queue für eine Transmission Queue (siehe hierzu den folgenden Abschnitt 10.3.3 und Abb. 10.3.5 dieses Kapitels).

Sie können den Channel Initiator benutzen, um Channels zu starten.

Um Nachrichten zu empfangen, muss ein Listener-Programm auf der Empfängerseite gestartet worden sein.

Das Abhören (monitor) von TCP/IP-Ports geschieht mit Hilfe des WebSphere MQ für z/OS Channel-Initiators. Der Listener, ein WebSphereMQ Komponente, und ein Teil des empfangenden Queuemanagers, überwacht Nachrichten, die auf einem TCP/IP-Port ankommen. Er erstellt einen Message Channel Agent (MCA), der diese Verbindung verarbeitet. Wenn eine Nachricht eintrifft, startet er den Message Channel Agent. Die MCA speichert die Nachricht in die Target-Queue, die in dem Message Descriptor angegeben ist.

Port 1414 ist der Standard TCP/IP Port für WebSphere MQ. Mehrere TCP/IP Listener mit verschiedenen Ports können innerhalb des Channels Initiators gestartet werden. Jeder Listener hört einem bestimmten TCP/IP-Port ab (monitors the Port). Ein Listener wird mit der START LISTENER Befehl innerhalb des Queue-Manager Subsystems gestartet.

Zusätzlich zu den üblichen TCP/IP Listeners unterstützt WebSphere MQ für z/OS auch SNA LU 6.2 Listener.

Das Anwendungsprogramm, das die eingehende Nachricht verarbeiten soll, kann manuell oder automatisch gestartet werden. Um das Programm automatisch zu starten, muss eine Initiation Queue und ein Anwendungsprogramm mit der lokalen Queue assoziiert werden, und der Trigger-Monitor muss laufen. Dies wird in Abschnitt 10.3 diskutiert.

10.2.8 WebSphere MQ auf dem gleichen z/OS System

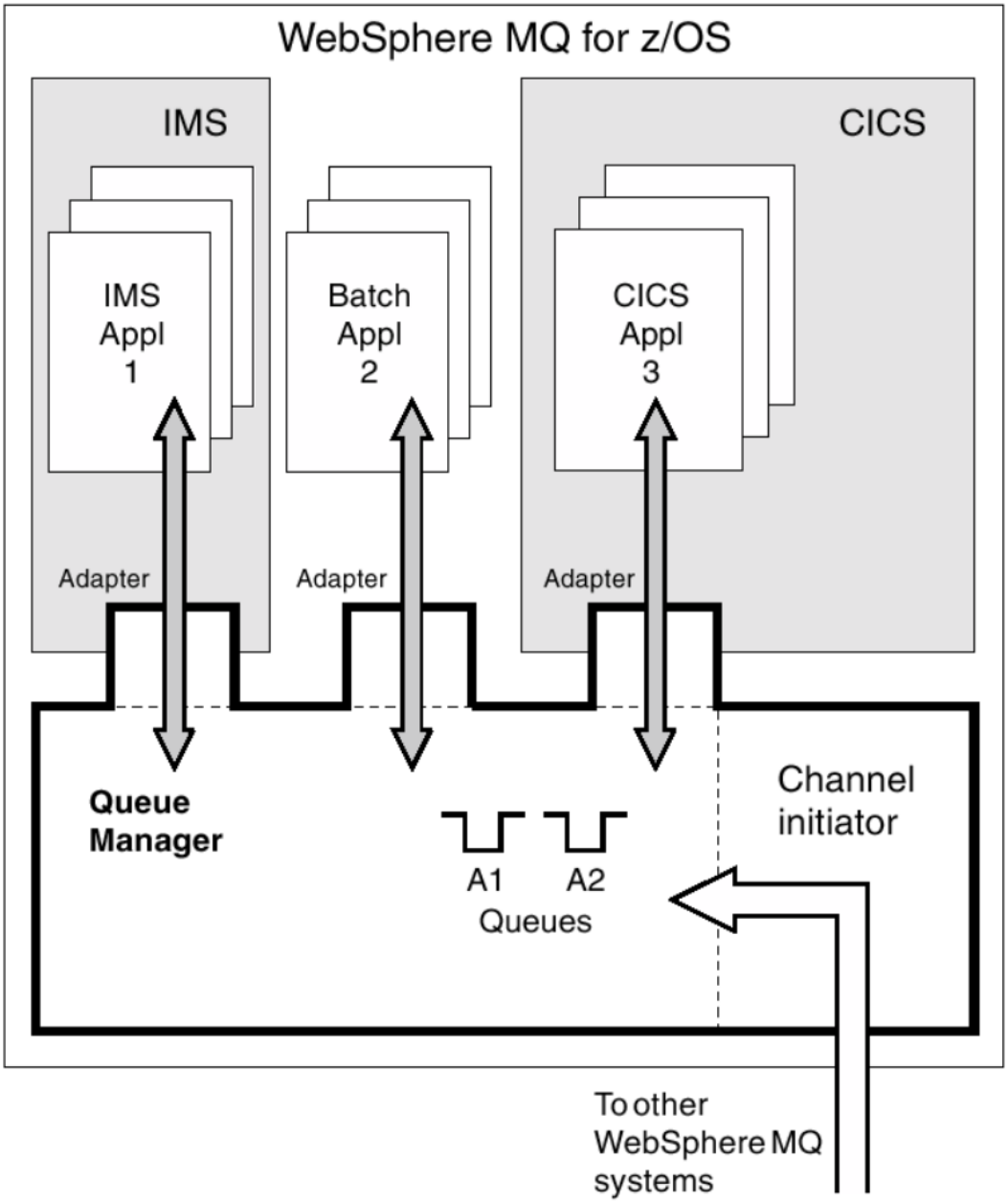


Abb. 10.2.7
WebSphere MQ kann benutzt werden, um Prozesse auf dem gleichen Rechner zu verbinden

MQ Kommunikation wird in der Regel zwischen entfernten Systemen durchgeführt. Es kann aber auch für eine Kommunikation zwischen zwei Regions auf dem gleichen z/OS System eingesetzt werden. Abb. 10.2.7 ist ein Beispiel für eine Programm-zu-Programm-Kommunikation in einem einzigen z/OS-System, zum Beispiel zwischen einer CICS Region und einer IMS-Region.

Eine Übungsaufgabe (Tutorial) für dieses Scenario läuft auf unserem Rechner. Sie ist zu finden unter

http://www.cedix.de/VorlesMirror/Band1/TutorialMQ1.pdf

10.2.9 MQPUT

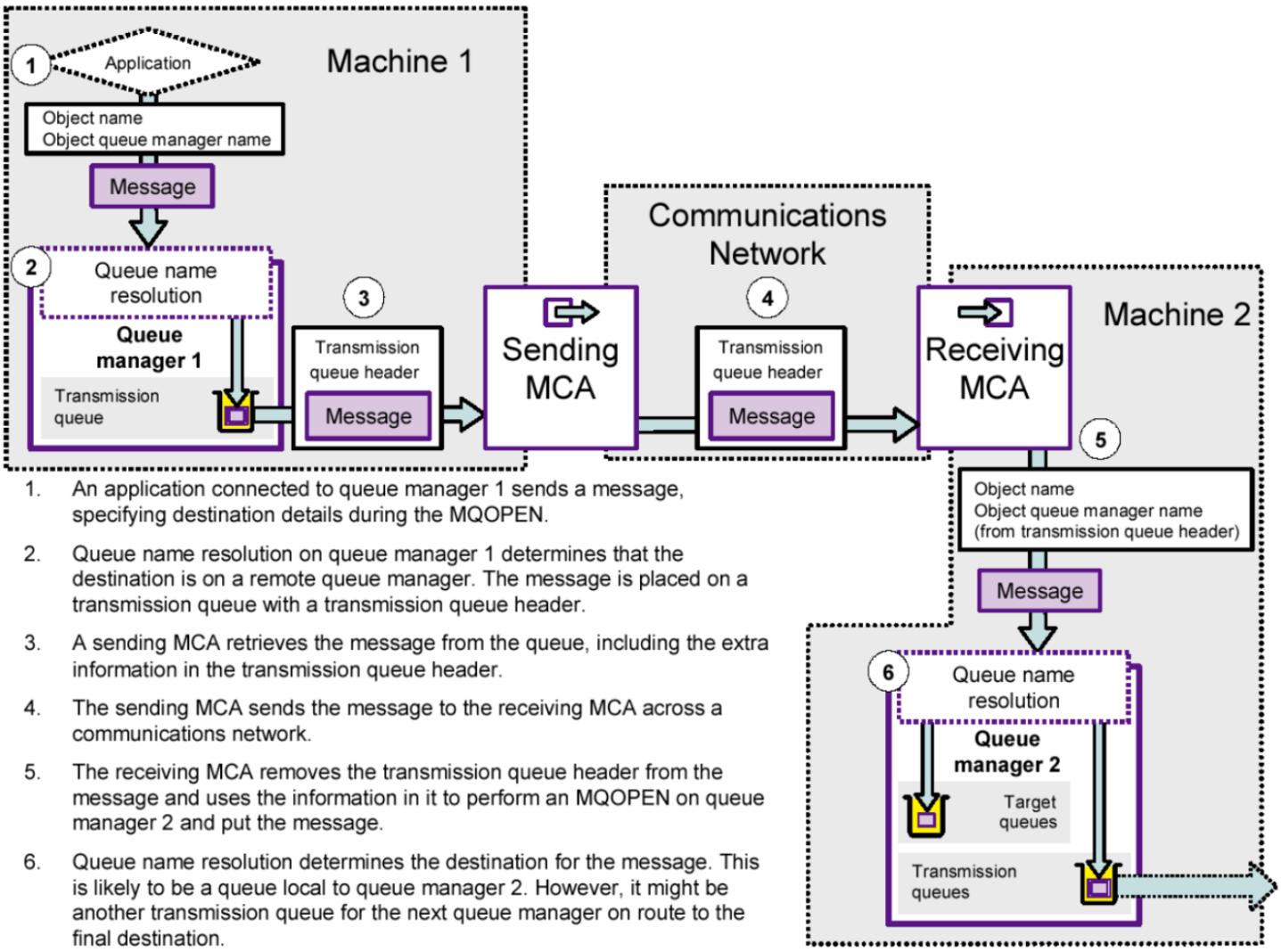
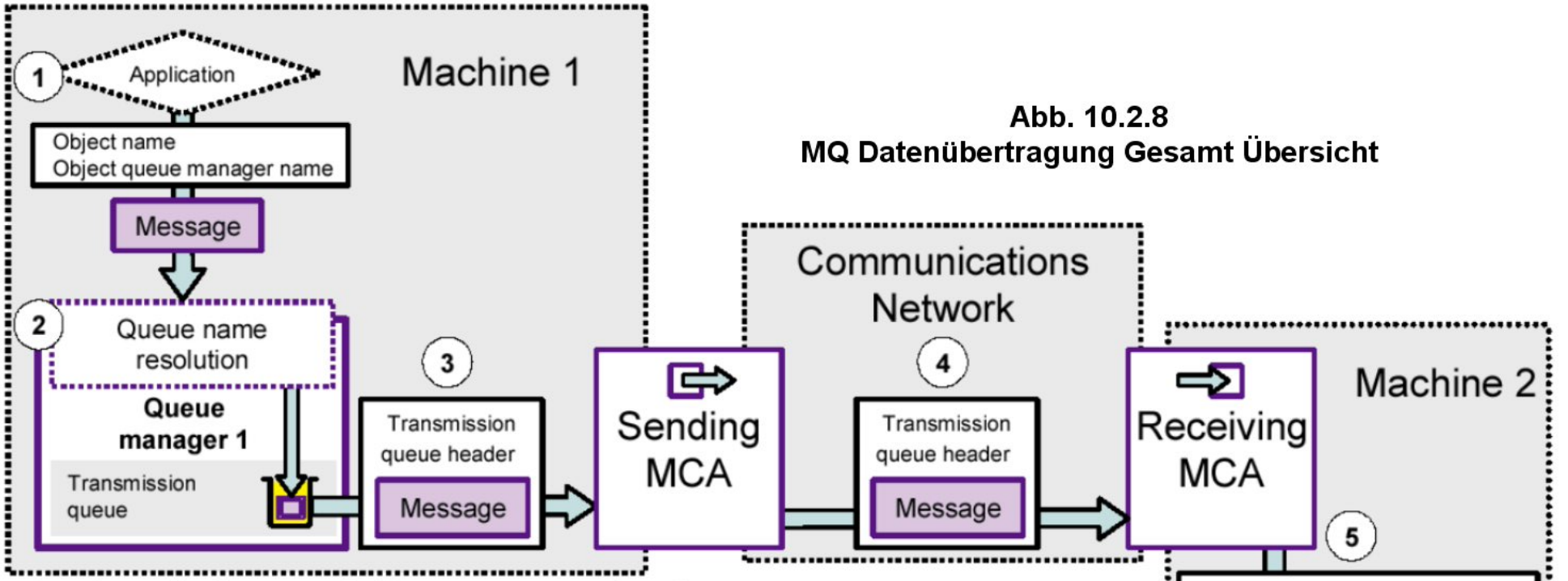


Abb. 10.2.8
MQ Datenübertragung Gesamt Übersicht

Beim Ausführen eines MQPUT Calls übergibt das Anwendungsprogramm neben der Nachricht einen Remote Message Queue Object Descriptor (MQOD) an den lokalen Queue Manager. Dieser enthält spezifisch den Namen des Entfernten Queue Managers und den Namen der entfernten Target Queue.

Abb. 10.2.8
MQ Datenübertragung Gesamt Übersicht



1. An application connected to queue manager 1 sends a message, specifying destination details during the MQOPEN.
2. Queue name resolution on queue manager 1 determines that the destination is on a remote queue manager. The message is placed on a transmission queue with a transmission queue header.
3. A sending MCA retrieves the message from the queue, including the extra information in the transmission queue header.
4. The sending MCA sends the message to the receiving MCA across a communications network.
5. The receiving MCA removes the transmission queue header from the message and uses the information in it to perform an MQOPEN on queue manager 2 and put the message.
6. Queue name resolution determines the destination for the message. This is likely to be a queue local to queue manager 2. However, it might be another transmission queue for the next queue manager on route to the final destination.

Während der Queue Namensauflösung durch den sendenden Queue-Manager, (wo der sendende MCA läuft), generiert der lokale Queue Manager hieraus den Message Descriptor, der in den Message Header (Transmission Queue Header) eingebaut wird. Er wird verwendet, um die Nachricht an den Queue-Manager zu übertragen, auf dem der empfangende MCA läuft. Dieser schreibt die Nachricht auf eine von mehreren Target Queues, die von dem empfangenden Queue-Manager gesteuert werden. Der Transmission Queue Header enthält die folgenden Informationen:

- **Remote Queue-Name:**
  Der Name der Target Queue für die Nachricht, die von der Queue-Manager während der Queue Namensauflösung ermittelt wurde. Wenn der MCA im entfernten Queue-Manager die Nachricht in eine Queue zu schreiben versucht, entnimmt er diesen Namen der Queue dem MQ Object Descriptor (MQOD) beim Öffnen der Queue. MQOD ist eine Struktur (in C++) oder ein Copybook (in Cobol), die als Parameter in dem MQOPEN Anruf des sendenden Anwendungsprogramms benutzt wurde.

- **Remote Queue Manager Name:**
  Dies ist der Name des Target-Queue-Managers, der die Target Queue hostet. Dies ist möglicherweise nicht der Name des Queue Managers, der die Nachricht empfängt. Dieser Fall kann auftreten, wenn dieser Queue Manager nicht die Final Destination für die Nachricht ist. Eine MQ Channel Verbindung kann über ein Netzwerk von Queue Managen zu dem empfangenden Queue Manager erstellt werden.

10.3 Trigger

10.3.1 MQ Client/Server Communication

In einer WebSphere MQ Client/Server-Konfiguration arbeiten ein Rechner und sein Queue Manager als Client, und ein anderer Rechner und sein Queue-Manager arbeitet als Server.

Streng genommen stellt WebSphere MQ eine Peer-to-Peer-Kommunikation dar. Es ist jedoch nützlich, den sendenden Queue-Manager als Client und dem empfangenden Queue-Manager als Server zu betrachten, weil WebSphere MQ häufig in einem Request / Response-Modus betrieben wird. Das sendende System erwartet (asnchronously) eine Reaktion von dem empfangenden System beim Senden einer Nachricht (Request-Message), und das empfangende System reagiert durch das Senden einer Response Message.

In diesem Text benutzen wir den Begriff MQ-Client für einen Queue-Manager, der eine Nachricht an eine Target Queue sendet, und den Begriff MQ-Server für einen Queue-Manager, der die Target Queue unterhält, und der potentiell mit einer Antwort-Nachricht an den MQ Client reagiert.

Bitte denken Sie daran, WebSphere MQ ist eine unidirektionale Kommunikation. Um in einem Request / Response-Modus zu arbeiten, muss der MQ Server eine zweite unidirektionale Verbindung vom MQ-Server an den MQ-Client verwalten.

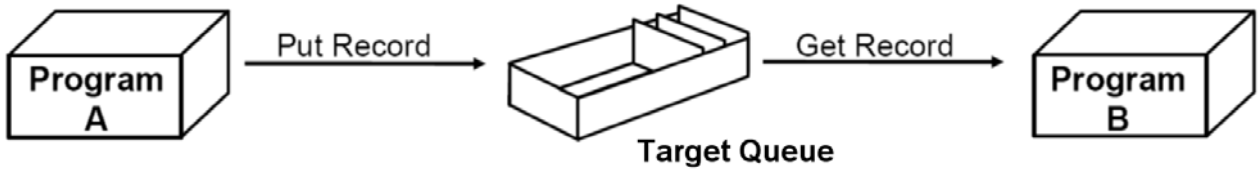


Abb. 10.3.1
Fire and forget

In seiner einfachsten "Fire and Forget"-Form, überträgt das sendende WebSphere MQ Programm A eine Nachricht an einen empfangenden Programm B und vergisst die ganze Sache. Es liegt an der empfangenden Programm B, die Nachricht irgend wann von der Target Queue zu extrahieren.

Die Nachricht befindet sich entweder auf dem sendenden Rechner oder dem Target Rechner, je nachdem, ob die Nachricht bereits übermittelt wurde oder nicht. Programm A kümmert das alles nicht.

Wenn das empfangende Programm B Programm nicht verfügbar ist, wird die Nachricht in einer Queue bleiben und später verarbeitet. Programm B kann den ganzen Tag lang warten, ehe es mit einem MQGET Befehl die Nachricht ausliest.

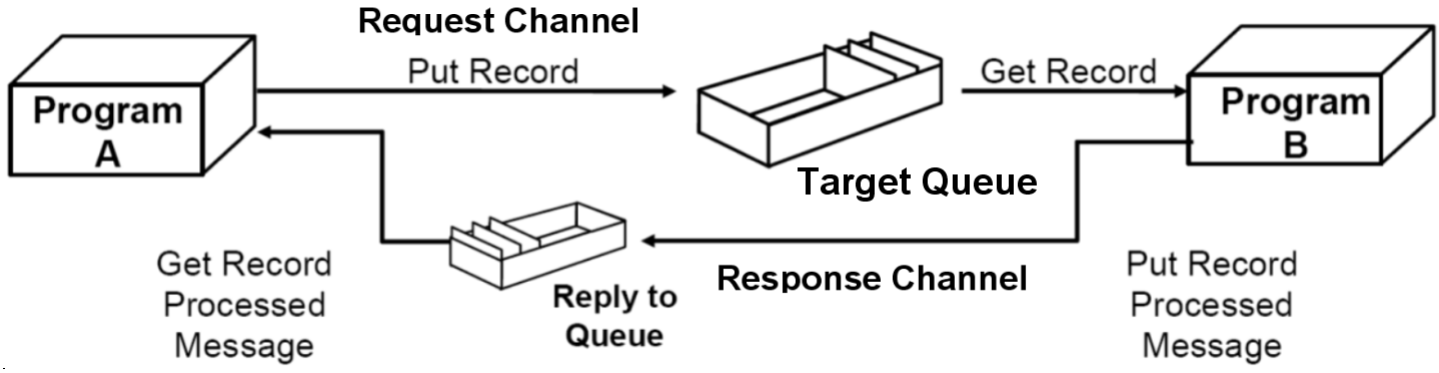


Abb. 10.3.2
Request / Response Modus

Eine Alternative ist, Programm B automatisch zu starten, wenn eine Nachricht eintrifft (oder nachdem eine bestimmten Anzahl von Nachrichten eingetroffen sind).

Obwohl WebSphere MQ asynchron arbeitet, kann ein "Request / Response" Modus einen synchronen Betrieb simulieren. Es wird recht häufig auf diese Weise verwendet. Dazu enthält der Message Descriptor der Request Nachricht eine "Reply to ...."-Anzeige.

Da WebSphere MQ asynchron und unidirektional arbeitet, werden zwei MQ-Kanäle mit zwei Message Channel Agent (MCA) Paaren für einen Request / Response Betrieb benötigt.

10.3.2 MQI Channel

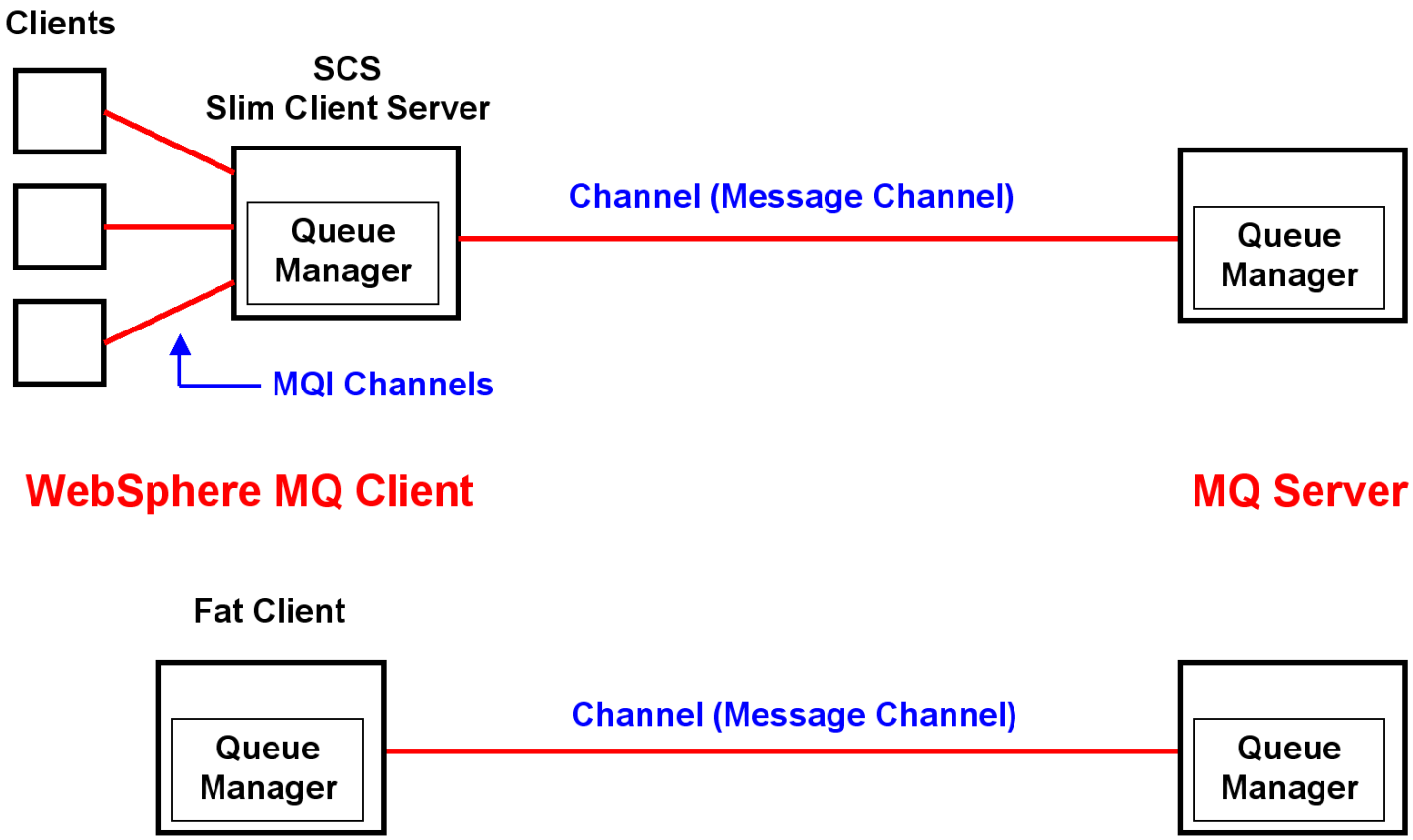


Abb. 10.3.3
MQI Channels verbinden Thin Clients mit einem Thin Client Server

Bei den WebSphere MQ Netzwerk-Kommunikationsverbindungen existieren zwei verschiedene Arten von Channels: Message Channel und MQI Channel

Message Channel

Ein Message Channel (in der Regel nur als Channel bezeichnet) ist das, was wir bisher diskutiert haben. Er verbindet zwei Queue Manager durch Message Channel Agents (MCAs) auf jeder Seite. Ein Message Channel ist unidirektional, besteht aus zwei Message Channel Agents (ein Sender und ein Empfänger) und einem Kommunikationsprotokoll. Ein MCA überträgt Nachrichten von einer Transmission Queue zu einer Kommunikationsverbindung, und von einer Kommunikationsverbindung zu einem Target Queue. Für eine bidirektionale Kommunikation ist es notwendig, ein Paar von Kanälen, bestehend aus einem Sender und einem Empfänger-Channel zu definieren.

Der Queue-Manager überträgt Nachrichten an andere Queue-Manager über Channels. Hierzu werden vorhandenen Netzwerk-Einrichtungen, in der Regel TCP / IP, benutzt.

Channels sind unidirektional. Die meisten der Zeit verstehen wir unter einem Channel einen Message Channel.

MQI Channel

Ein spezieller Fall eines Channels ist der Message Queue Interface (MQI) Channel. Es verbindet einen WebSphere MQ-Client (auch als Slim-Client bezeichnet) mit einem Queue-Manager. WebSphere MQ Slim Clients verfügen nicht über einen eigenen Queue-Manager.

Ein MQI Channel ist bidirektional.

WebSphere MQ unterscheidet zwischen verschiedenen Arten von Clients:

- **Slim client (oder WebSphere MQ client)**
- **Fat client**

Der Unterschied zwischen einem Slim Client und einem Fat Client besteht in der Art, wie Nachrichten gesendet werden. Die Transmission Queue befindet sich entweder in dem Endbenutzer-Arbeitsplatz (Fat Client) oder auf einem „Slim Client-Server“ (SCS). Ein Slim Client muss an einen WebSphere SCS angeschlossen sein, der die Queue-Manager Services zur Verfügung stellt. An einen SCS sind in der Regel viele Slim Clients angeschlossen.

Fat Clients haben eine lokalen Queue-Manager, Slim Clients nicht.

Ein Slim WebSphere MQ-Client, der sich nicht mit einem SCS verbinden kann, kann nicht arbeiten, weil sich der Queue-Manager und die Queues für den Slim Client auf dem SCS befinden. WebSphere MQ-Clients sind häufiger Slim Clients und seltener Fat Clients.

Ein Fat Client enthält einen eigenen Queue Manager. Für eine Gruppe von Slim Clients stellt ein WebSphere MQ Slim Client Server (SCS) Queue Manager Services zur Verfügung.

Hinweis: Die "WebSphere MQ-Client für Java" ist ein Slim Client.

10.3.3 Trigger Event

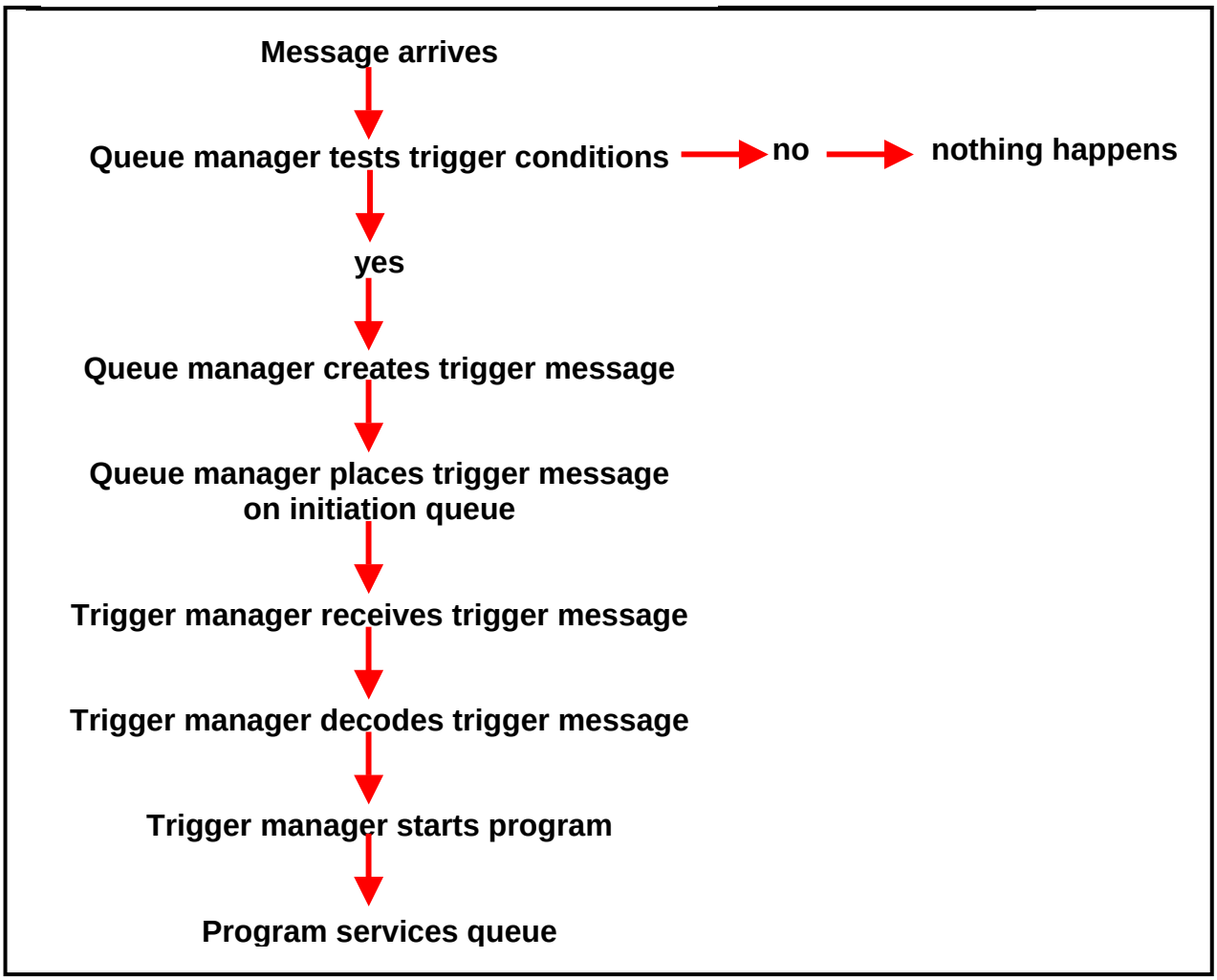


Abb. 10.3.4
Verarbeitung eines Trigger Events

Die Target Anwendung kann die Nachricht aus der Target Queue zu einem beliebigen Zeitpunkt extrahieren. Dies kann zu einer langen Wartezeit führen, ehe eine Antwort gesendet wird. Um pseudosynchron zu arbeiten, muss die Target-Anwendung "getriggert" werden. Die Target Anwendung wird informiert, dass eine Nachricht in der Target Queue angekommen ist, die ihre Aufmerksamkeit erfordert. Dazu wird ein „Triggering Mechanismus" benötigt.

Ein "Trigger-Event" ist ein Ereignis, das den entfernten Queue-Manager motiviert, eine "Trigger-Message" zu generieren. Ein Trigger-Event wird in der Regel in dem Deskriptor einer Nachricht angegeben.

Wenn Triggerung für eine Target Queue aktiviert ist und ein Trigger-Event auftritt, sendet der Queue Manager eine "Trigger Message" an eine lokale Queue, die als Initiation Queue bezeichnet wird. Das Vorhandensein der Trigger-Nachricht in der Initiation Queue zeigt einen Trigger Event an. Eine Initiation Queue kann mehrere Target Queues bedienen, aber eine Target Queue ist immer einer bestimmten Initiation Queue zugeordnet.

Nehmen wir die folgende Situation an: Program A sendet eine Nachricht an ein Remote Programm B, das getriggert werden soll (siehe Abb. 10.3.4 und 10.3.5):

WebSphere MQ Application Programming Guide, March 2003, SC34-6064-03, p.196

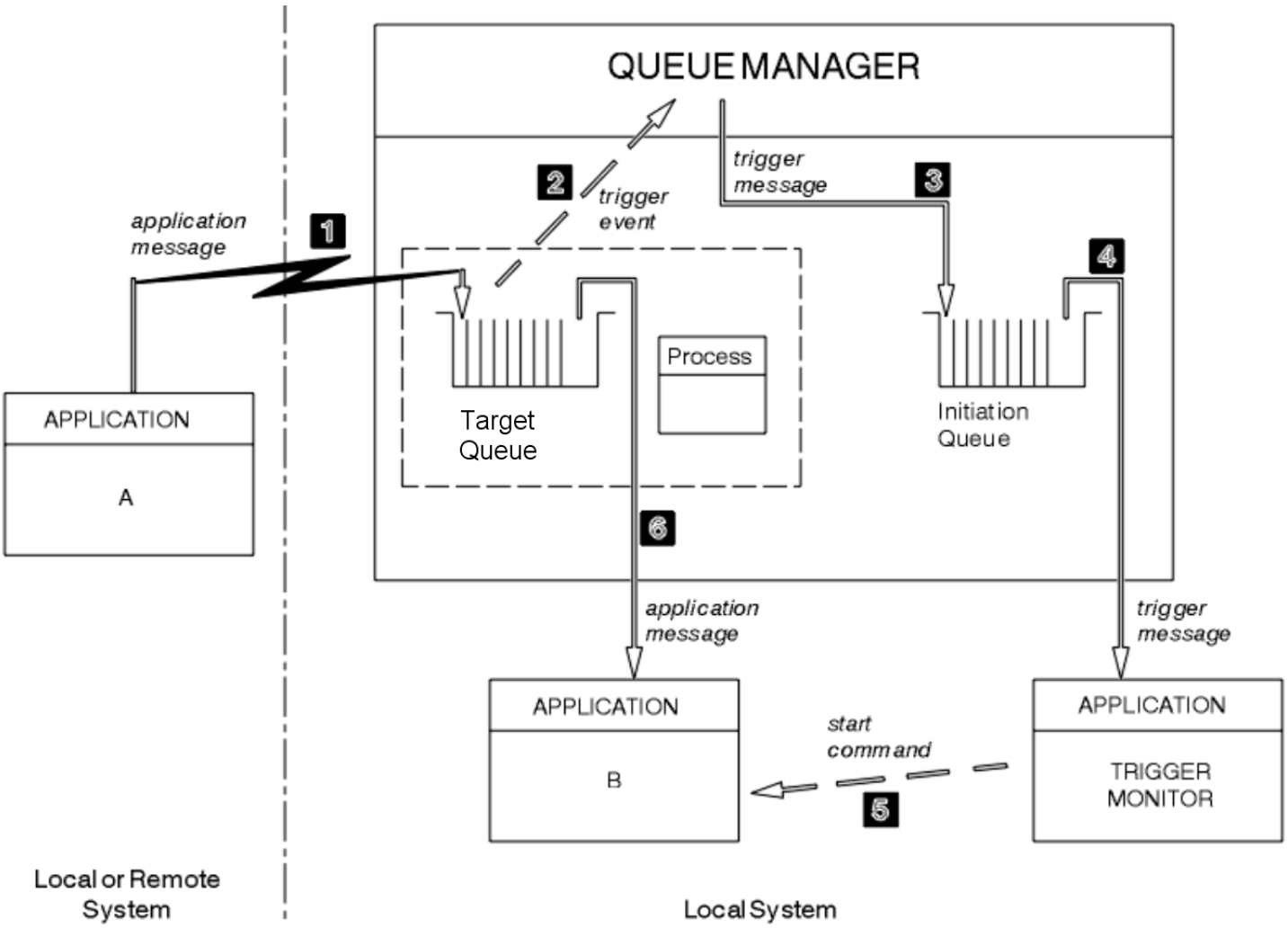


Abb. 10.3.5
Aufgabe der Initiation Queue

Die Ankunft einer Nachricht in einer getriggerten Queue startet eine komplexe Folge von Ereignissen.

1. **Programm A, das entfernt zu dem empfangenden Queue-Manager ist, stellt eine Nachricht in die Target Queue. Beachten Sie, dass kein Anwendungsprogramm diese Nachricht erwartet.**
2. **Der empfangende Queue-Manager überprüft die Nachricht. Er stellt fest, ob die Bedingungen erfüllt sind, unter denen er ein Trigger-Event erzeugen muss (Daten im Message Descriptor). Wenn ja, wird ein Trigger Event generiert.**
3. **Der Queue-Manager erstellt eine Trigger-Nachricht und legt sie in die spezifische Initiation Queue, welche der Target Queue zugeordnet ist. Dazu muss eine spezielle Anwendung (Trigger-Monitor) das Erscheinen der Trigger Message in der Initiation Queue zur Kenntnis nehmen..**
4. **Der Trigger Monitor liest die Trigger-Nachricht von der Initiation Queue.**
5. **Der Trigger-Monitor startet das Programm B (die Server-Anwendung).**
6. **Programm B öffnet die Target Queue und liest die Nachricht.**

10.3.4 WebSphereMQ Zugriff auf eine CICS Transaction

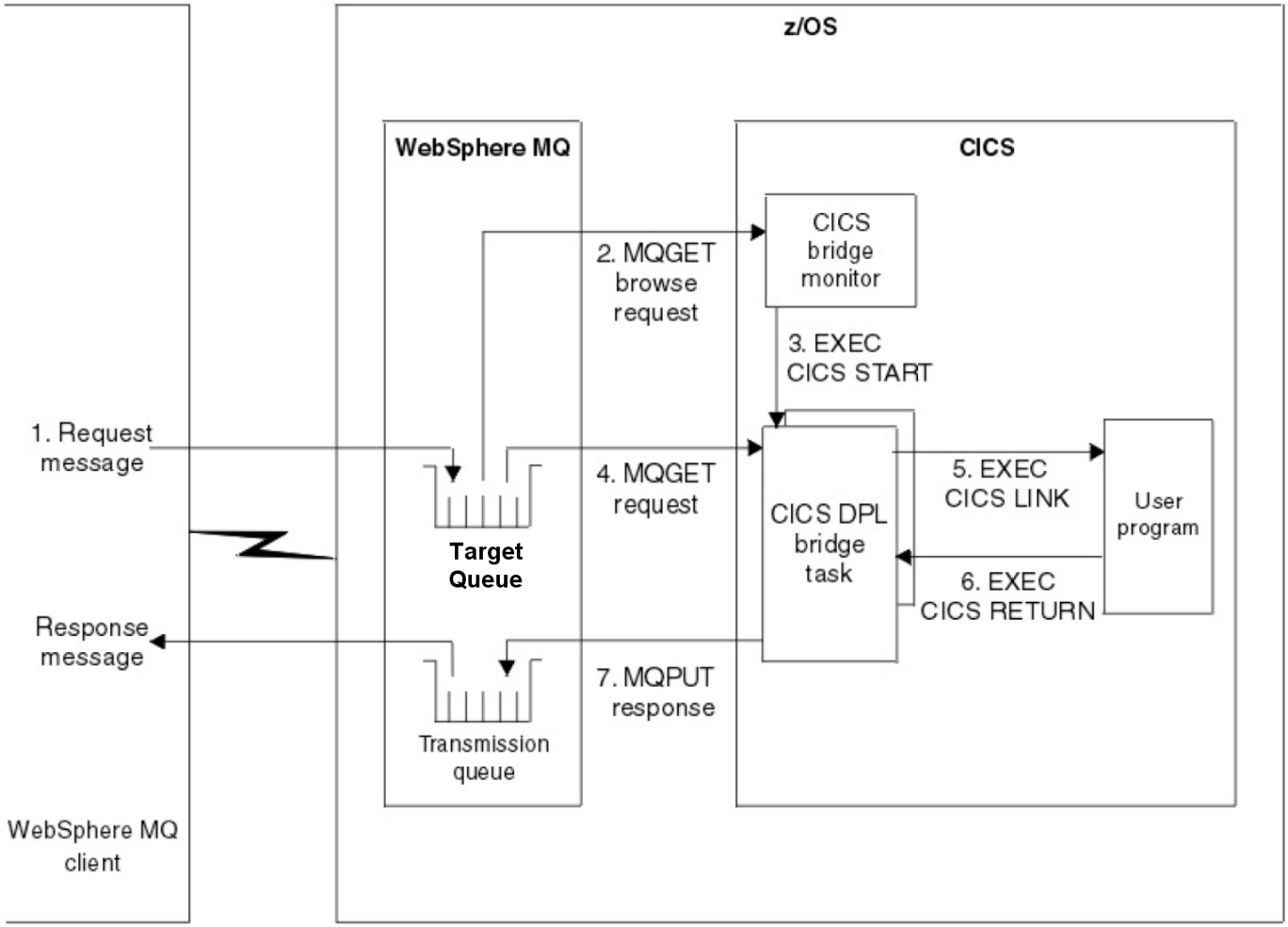


Abb. 10.3.6
Der CICS Bridge Monitor startet mittels des normalen DPL ein CICS Anwendungsprogramm

WebSphereMQ im Request/Response-Modus ist eine Alternative zur Verwendung eines 3270-Emulator beim Zugriff auf eine CICS Transaktion. Dies kann mit Hilfe des "MQ-CICS-Bridge" Software-Produktes durchgeführt werden. Die MQ-CICS-Bridge besteht aus 2 Komponenten, dem Bridge Monitor und der DPL Bridge Task.

Abb. 10.3.6 zeigt, wie es funktioniert:

1. **Eine Nachricht, die eine CICS Transaktion aufrufen soll, wird in der Target Queue empfangen. Die Nachricht enthält einen Trigger Event.**
2. **Der CICS Bridge Monitor übernimmt die Funktion des Trigger Monitors.**
3. **Der CICS Bridge Monitor startet eine spezielle "CICS DPL Bridge" Task.**
4. **Die CICS DPL Bridge Task liest (und entfernt) die Nachricht aus der Target Queue.**
5. **Der CICS DPL Bridge Task verwendet den Distributed Program Link (DPL) Aufruf und greift auf eine CICS Region zu (z.B. ein Application Owning Region - AOR). Diese startet die Transaktion und führt sie aus.**
6. **Die Transaktion übergibt das Ergebnis an die CICS DPL Bridge Task, z.B. über eine COMMAREA.**
7. **Der CICS DPL Bridge Task verwendet einen MQPUT Anruf, um das Ergebnis in die WebSphere MQ Transmission Queue zu speichern.**
8. **Der Queue-Manager transportiert die Nachricht in der Transmission Queue über eine MQ-Channel an den WebSphere MQ-Client.**

Eine detaillierte Beschreibung ist zu finden in einer Masterarbeit von Tobias Busse:
http://www.cedix.de/DiplArb/Busse04.pdf

10.3.5 Multiple Target Queues

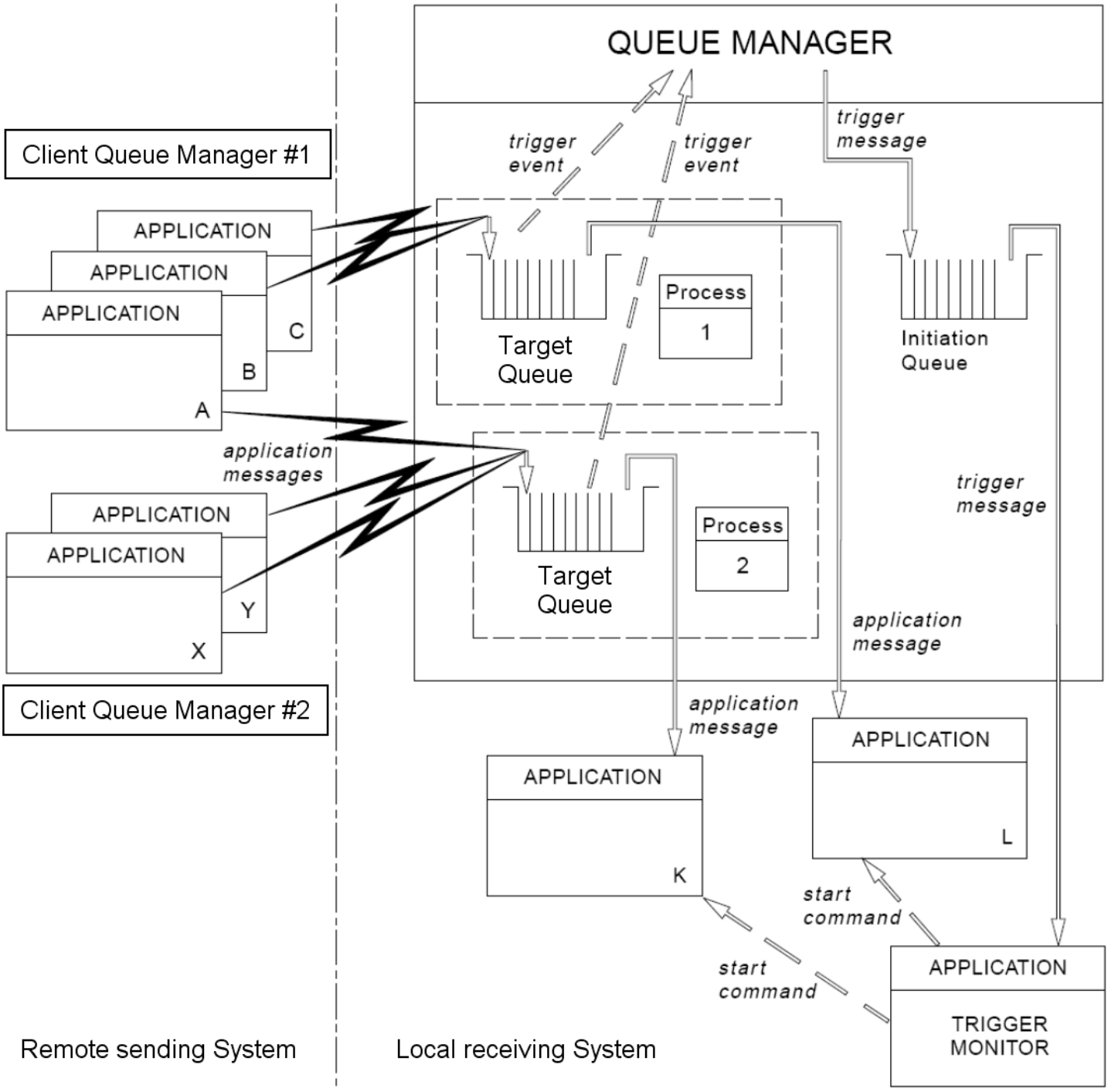


Abb. 10.3.7
Unterschiedliche Anwendungen kommunizieren mit unterschiedlichen Target Queues

Abb. 10.3.7 zeigt eine Situation, wo mehrere Programme, die entfernt zu dem empfangenden Queue-Manager sind, Nachrichten an zwei verschiedenen Target Queues senden. Mehrere Nachrichten können die gleiche Target Queue adressieren. Es ist angenommen, alle eingehenden Nachrichten zeigen Trigger-Events in ihren Deskriptoren an.

Obwohl mehrere Target Queues in Betrieb sind, und jede einem anderen Anwendungsprogramm zugeordnet ist, kann eine einzige Initiation Queue und ihr Trigger Monitor alle Triggering Events bedienen.

Abb. 10.3.7 zeigt zwei Server-Anwendungsprogramme K und L, die Nachrichten von verschiedenen Client-Anwendungsprogrammen A, B, C, X, Y empfangen können. Programme K und L werden von zwei verschiedenen Target Queues bedient.

Mit einer Target Queue ist ein Prozess-Definition Objekt assoziiert, welches Details über die Anwendung enthält, die die Nachricht verarbeiten soll. Der Queue Manager stellt die Informationen in eine Trigger Message, die in einer Initiation Queue gespeichert wird.

Der Trigger Monitor extrahiert diese Informationen aus der Trigger-Nachricht und startet die entsprechende Anwendung, um die Nachricht in jeder Target Queue zu behandeln. Nur eine Initiation Queue ist erforderlich, um entweder Anwendung K oder Anwendung L zu triggern.

10.4 MQI Programmierung

10.4.1 MQI Application Programming Interface

Die MQI API besteht aus 13 API Kommandos. Dies sind die 6 am häufigsten benutzten Kommandos:

MQCONN	**Verbindung mit einem (normalerweise entfernten) Queue Manager herstellen**
MQOPEN	**Öffnen (Open) einer spezifischen Queue**
MQPUT	**Eine Message in eine Queue schreiben**
MQGET	**Eine Message aus einer Queue auslesen**
MQCLOSE	**Eine Queue schließen (Close)**
MQDISC	**Verbindung mit einem Queue Manager auflösen**

Die übrigen 7 Kommandos sind:

MQPUT1	**Kombination von MQOPEN + MQPUT + MQCLOSE**
MQINQ	**Eigenschaften eines Objektes erfragen**
MQSET	**Eigenschaften (Properties) eines Objektes setzen**
MQCONNX	**Standard oder fast Path Bindings**
MQBEGIN	**Eine Unit of Work starten (database coordination)**
MQCMIT	**Commit eine Unit of Work**
MQBACK	**Back out**

Ein Client-Anwendungsprogramm benutzt verschiedene WebSphere MQ MQI Kommandos.

Es beginnt mit den MQCON und MQOPEN Kommandos. In der Regel beziehen diese sich auf einen entfernten QUEUE-Manager und eine entfernte Target-Queue. Bedenken Sie, dass der Remote Queue Manager mehrere Target Queues enthalten kann. Mit MQOPEN wählen wir die richtige Target Queue.

Das Anwendungsprogramm führt dann seinen Code aus, in der Regel durch den Einsatz mehrerer MQPUT und/oder MQGET Anrufe.

Wenn es damit fertig ist, beendet es seine Arbeit durch Ausgabe von MQCLOSE und MQDISC Kommandos.

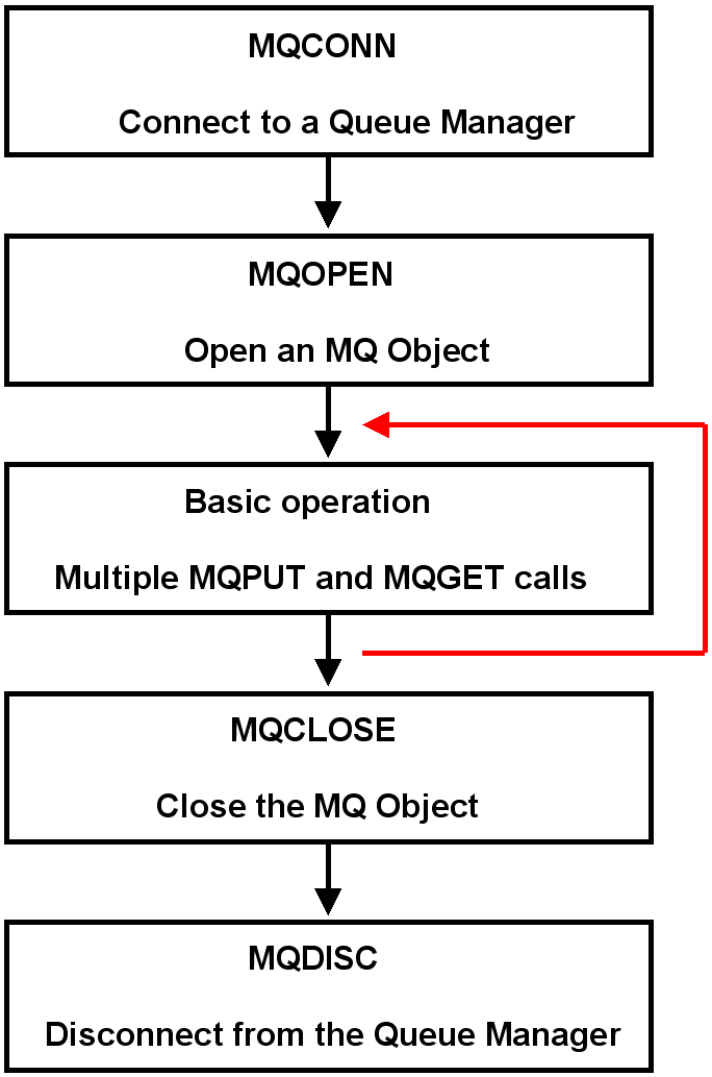


Abb. 10.4.1
Folge von MQI Kommandos in einem Client-Anwendungsprogramm

10.4.2 MQI Call Parameters

Es gibt zwei Typen von Parametern, die von allen Kommandos benutzt werden:

Handles

Diese werden von den Queue-Manager MQCONN und MQOPEN Kommandos zurückgegeben, und werden dann als Eingangsgrößen für die nachfolgenden MQPUT und MQGET Kommandos verwendet.

Der Begriff „Handle“ wird hier benutzt, um eine Message Channel Verbindung oder eine Target Queue eindeutig zu definieren.

Return codes

Zwei Return-Codes sind für alle Kommandos gebräuchlich: ein Completion-Code und ein Reason-Code.

Der Completion-Code gibt an, ob die Kommandoausführung erfolgreich war (mit einem MQCC_OK), oder ob ein Fehler aufgetreten ist (mit einem MQCC_FAILED).

Der Reason-Code ist MQCC_NONE, wenn der Completion-Code MQCC_OK ist. Wenn nicht, wird ein anderer Wert zurückgegeben, der die Ursache der Warnung oder des Fehler im Completion-Code erklärt.

10.4.3 Verbinden mit dem Queue Manager

Um eine MQSeries Aktivität mittels der MQI Schnittstelle zu starten, sollten Sie sich zunächst mit einem (in der Regel entfernten) QUEUE-Manager mit einem der beiden verfügbaren Verbindungs-Kommandos, MQCONN oder MQCONNX verbinden. Beispielsweise unter Verwendung von MQCONN:

```
MQCONN (QMgrName, Hconn, CompCode, Reason)
```

Als Input für MQCONN müssen wir den symbolischen Namen des (in der Regel entfernten) Queue-Managers (QMgrName) angeben.

Das Kommando gibt die folgenden Werte zurück:

- **Eine Connection Handle (`Hconn`) zu dem Queue-Manager. Hconn wird durch das Anwendungsprogramm in den folgenden Kommandos wendet, um diesen spezifischen Queue Manager zu adressieren.**
- **Die Ergebnis-Codes (Completion- und Reason-Codes).**

MQCONN bewirkt, dass der lokale Queue Manager eine Transmission Queue einrichtet, und ein Message Channel mit seinen beiden MCAs erstellt wird.

10.4.4 Öffnen von MQSeries Objekten

Das MQOPEN Kommando ermöglicht es einer Anwendung, Nachrichten in eine Queue zu schreiben oder Nachrichten aus einer Queue zu lesen.

```
MQOPEN (Hconn, ObjDesc, Options, Hobj, CompCode, Reason)
```

Das Kommando hat diese Input Parameter:

- **Eine Connection Handle. Der Wert von `Hconn` war von dem vorangehenden MQCONN Kommando zurückgegeben worden.**
- **Eine Beschreibung der Target Queue, die wir öffnen (Open) möchten. Dies geschieht in der Form eines MQ Object Descriptor (MQOD). MQOD ist eine Structure (in C++) oder ein Copybook (in Cobol). Diese Struktur identifiziert die Target Queue, die geöffnet werden soll.**
- **Eine oder mehrere Optionen. Eine mögliche Option ist es z.B., der Anwendung zu erlauben, eine Nachricht in die Target Queue zu stellen.**

Das Kommando gibt die folgenden Werte zurück:

- **Eine Object Handle Hobj, die den Zugriff auf die Target Queue mittels ihres Namens spezifiziert.**
- **Eine modifizierte Object-Descriptor Structure (MQOD, wenn eine Abänderung erforderlich ist).**
- **Die Result-Codes (Completion- und Reason-Codes).**

Weiterführende Information finden Sie in: MQ Application Programming Guide:
http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/csqzal05.pdf
Und in dem MQSeries Primer:
http://www.cedix.de/Literature/Textbooks/MQSerPrimer.pdf

10.4.5 Schreiben einer Nachricht in eine Queue

Wir benutzen das MQPUT Kommando, um eine Nachricht in eine Queue zu schreiben:

```
MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer, CompCode,
Reason)
```

Dieses Kommando erhält als Input Parameter:

- **Eine Connection Handle Hconn, die durch das MQCONN Kommando zurückgegeben wurde.**
- **Eine Queue Handle Hobj , die von dem MQOPEN Kommando zurückgegeben wurde.**
- **Eine Beschreibung der Nachricht, die Sie in die Queue stellen wollen, in der Form eines Message Descriptors.**
- **Control Informationen, in Form einer Put-Message Optionen (MQPMO) Struktur.**
- **Die Länge der Daten in dieser Nachricht.**
- **Die Nachricht selbst, enthalten in einem Puffer.**

Das Kommando gibt diese Werte zurück:

- **Die Result-Codes (Completion- und- Reason-Codes).**
- **Aktualisierte Message Descriptor und Optionen, wenn der Aufruf erfolgreich ausgeführt wurde.**

10.4.6 Lesen einer Nachricht aus einer Queue

Wir benutzen das MQGET Kommando, um Nachrichten aus einer Queue zu lesen:

```
MQGET (Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer, DataLength,
CompCode, Reason)
```

Die Eingabeparameter für diesen Aufruf sind:

- **Eine Connection Handle `Hconn`.**
- **Eine Queue Handle `Hobj`.**
- **Eine Beschreibung der Nachricht, die wir aus der Queue lesen wollen, in der Form einer MQMD Structure.**
- **Control Information in Form einer Get Message Options (MQGMO) Struktur.**
- **Die Größe des Puffers, in dem die Nachricht gespeichert werden soll.**
- **Die Adresse des Puffers.**

Die Ausgabe Parameter dieses Aufrufs sind:

- **Die Result-Codes (Completion- und Reason-Codes).**
- **Die Message in dem angegeben Puffer, wenn das Kommando erfolgreich abgeschlossen wurde.**
- **Die Get Message Options Struktur, modifiziert, um den Namen der Queue zu zeigen, aus dem die Nachricht abgerufen wurde.**
- **Die Message Descriptor Struktur, mit Informationen über die eingelesene Nachricht.**
- **Die tatsächliche Länge der Nachricht.**

10.4.7 Schließen des MQ Objektes

Um ein MQ Object zu schließen (Close) benutzen wir das MQCLOSE Kommando.

```
MQCLOSE (Hconn, Hobj, Options, CompCode, Reason)
```

Dieses Kommando benutzt die folgenden Eingabe Parameter:

- **Eine Connection Handle `Hconn`.**
- **Eine Queue Handle `Hobj` des Remote-Queue-Objekts das wir schließen wollen wir.**
- **Die Close Optionen.**

Die folgenden Parameter werden zurückgegeben:

- **Die Result-Codes (Completion und Reason Codes).**
- **Die Object Handle `Hobj`, auf den Wert MQHO_UNUSABLE_HOBJ zurückgesetzt.**

Typischerweise wird eine Queue gelöscht, sobald das Programm, das sie geschaffen hat, ein MQCLOSE Kommando für diese Queue ausführt. In diesem Fall wird die Close Option MQCO_NONE erzeugt.

10.4.8 Codefragment

Das folgende Codefragment zeigt die APIs, um eine Nachricht in eine Queue zu schreiben und eine Antwort von einem anderen Queue zu erhalten. Die C++ MQI wird benutzt.

Das Codefragment verwendet die folgenden Definitionen. Die Felder CompCode und Reason enthalten die Fertigstellung Codes für die APIs. Wenn Sie interessiert sind, können Sie sie in der Application Programming Reference Dokumentation finden:
http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/csqzak05.pdf

```
                                        // Definitions
MQHCONN HCon;                  // Connection handle
MQHOBJ HObj1;                  // Object handle for queue 1
MQHOBJ HObj2;                  // Object handle for queue 2
MQLONG CompCode, Reason;       // Return codes
MQLONG options;
MQOD od1 =      {MQOD_DEFAULT};  // Object descriptor for queue 1
MQOD od2 =  {MQOD_DEFAULT};  // Object descriptor for queue 2
MQMD md =   {MQMD_DEFAULT};  // Message descriptor
MQPMO pmo = {MQPMO_DEFAULT}; // Put message options
MQGMO gmo = {MQPMO_DEFAULT}; // Get message options

// 1 Connect application to a queue manager.
strcpy (QMName,"MYQMGR");
MQCONN (QMName, &HCon, &CompCode, &Reason);
// 2 Open a queue for output
strcpy (od1.ObjectName,"QUEUE1");
MQOPEN (HCon,&od1, MQOO_OUTPUT, &Hobj1, &CompCode, &Reason);
// 3 Put a message on the queue
MQPUT (HCon, Hobj1, &md, &pmo, 100, &buffer, &CompCode, &Reason);
// 4 Close the output queue
MQCLOSE (HCon, &Hobj1, MQCO_NONE, &CompCode, &Reason);
// 5 Open input queue
options = MQOO_INPUT_AS_Q_DEF;
strcpy (od2.ObjectName, "QUEUE2");
MQOPEN (HCon, &od2, options, &Hobj2, &CompCode, &Reason);
// 6 Get message
gmo.Options = MQGMO_NO_WAIT;
buflen = sizeof(buffer - 1);
memcpy (md.MsgId, MQMI_NONE, sizeof(md.MsgId);
memset (md.CorrelId, 0x00, sizeof(MQBYTE24));
MQGET (HCon, Hobj2, &md, &gmo, buflen, buffer, 100, &CompCode, &Reason);
// 7 Close the input queue
options = 0;
MQCLOSE (HCon, &Hobj2,options, &CompCode, &Reason);
// 8 Disconnect from queue manager
MQDISC (HCon, &CompCode, &Reason);
```

1 Dieses Statement verbindet die Anwendung mit dem dem QUEUE-Manager mit dem Namen MYQMGR. Wenn der Parameter QMName keinen Namen enthält, dann wird der Default-QUEUE-Manager verwendet. MQ speichert die Handle des Queue-Managers in der Variablen HCon. Diese Handle muss in allen nachfolgenden APIs genutzt werden.

2 Um eine Queue zu öffnen muss der Name der Queue in den Objektdeskriptor kopiert werden, der für diese Queue verwendet wird. Dieses Kommando öffnet QUEUE1 nur für Output (Open Option MQOO_OUTPUT). Die Handle der Queue und Werte in dem Objektdeskriptor werden zurückgegeben. Die Handle Hobj1 muss in dem MQPUT Kommando angegeben werden.

3 MQPUT platziert die Nachricht, die in einem Puffer steht, in eine Queue. Parameter für MQPUT sind:

- **Die Handle des Queue-Manager (aus MQCONN)**
- **Die Handle der Queue (von MQOPEN)**
- **Den Message Descriptor**
- **Eine Struktur, die Optionen für den MQPUT Befehl enthält (siehe MQ Application Programming Reference)**
- **Die Länge der Nachricht**
- **Der Puffer, der die Daten enthält**

4 Dieses Kommando schließt die Output Queue. Da die Queue vordefiniert ist, findet kein Close Processing statt (MQOC_NONE).

5 Dieses Kommando öffnet Queue2 nur für die Eingabe, unter Benutzung der Queue-Voreinstellungen (Defaults).

6 Für das Get Kommando wird die nowait Option verwendet. MQGET braucht die Länge des Puffers als Eingangs Parameter. Da keine Nachrichten-ID oder Korrelations-ID angegeben ist, wird die erste Nachricht aus der Queue gelesen. Sie können einen Warteintervall (in Millisekunden) hier spezifizieren. Sie können den Return Code überprüfen, um herauszufinden, ob die Zeit abgelaufen ist und keine Nachricht eingetroffen ist.

7 Dieses Kommando schließt die Eingabe-Queue.

8 Die Anwendung schließt die Verbindung mit dem Queue-Manager.

Die gezeigten Codebeispiele benutzen C++ und die MQI API. Sie würden sehr ähnlich mit anderen Programmiersprachen wie Cobol oder PL/1.aussehen

Java MQ-Code sieht anders aus, weil der JEE (Java Enterprise Edition) Standard für JMS (Java Message Service) eine eigene API festlegt, die sich von der MQI API unterscheidet. Java-Anwendungen verwenden normalerweise die JMS API anstelle der MQI API.

Es gibt 2 Versionen von WebSphere MQ. Die "non-integrated"-Version benötigt keinen WebSphere Java Application Server (WAS), und wird in der Regel für andere Programmiersprachen als Java verwendet. Die integrierte Version ist Teil der WebSphere Java Application Server (WAS), nutzt aber die gleiche Code-Basis wie die nicht-integrierte Version.

MQ Code Beispiele für Cobol, C + + und Java sind verfügbar unter;

http://www.cedix.de/Literature/Textbooks/MQ/index.html

Wenn Sie glauben, all dies ist kompliziert: Die meisten Unternehmen stehen vor dem Problem, bestehende Anwendungs-Software zu integrieren, die von unabhängiger Seite in der Vergangenheit entwickelt wurde, in verschiedenen Programmiersprachen implementiert wurde, und auf verschiedenen Hardware-, Betriebssystem-und Middleware-Platformen läuft. Viele Experten betrachten die Verwendung von WebSphere MQ als den einfachsten und effizientesten Weg, um die Integration von Anwendungen zu erreichen.

10.5 Weiterführende Information

Eine sehr gute Einführung
Dieter Wackerow: MQSeries Primer. www.redbooks.ibm.com/redpapers/pdfs/redp0021.pdf
Auch hier erhältlich:
http://www.cedix.de/Literature/Textbooks/MQSerPrimer.pdf

Ein interessantes Video
“WebSphere MQ Zero to Hello World in under 5 Minutes”
unter Benutzung von Linux ist zu finden unter
http://www.youtube.com/watch?v=wSCHLBftjDw

Vier vergleichsweise einfache WebSphere MQ Tutorials sind beschrieben unter
http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=%2Fcom.ibm.mq.explorer.tutorials.doc%2Fbi00112_.htm

Vielleicht interessiert Sie hier ein MQSeries Video vom IBM Vertrieb
"IBM WebSphere MQ on System z"
http://www.youtube.com/watch?v=rAJMXO2o59M

und WebSphere MQ im Allgemeinen
http://www.youtube.com/watch?v=kbxFDP6aWhw