AN INTRODUCTION TO MESSAGING TECHNOLOGY

MARK EHR
POLARSOFT LIMITED
2465 CENTRAL AVENUE SUITE 205
BOULDER, COLORADO 80301
MARKEHR@POLARSOFTLTD.COM

ABSTRACT

The world of messaging technology can be a complex and confusing one. Many people travel a difficult road in trying to understand just what messaging is, how (and if) it should be applied to a given business need, and which messaging vendor to choose. This white paper provides a technical introduction to messaging technologies, describes major messaging products, and defines advantages and disadvantages of messaging technology.

Table of Contents

Table of Figures

SECTION 1. A HISTORY OF MESSAGING

EARLY MESSAGING SYSTEMS

Some of the first messaging systems that were placed into production were electronic mail, otherwise known as "e-mail". E-mail systems were (and still are) the most prevalent application of messaging technology, and serve as a very good model for describing the general characteristics of any messaging architecture.

CHARACTERISTICS OF AN E-MAIL SYSTEM

All e-mail systems share the same common characteristics:

- Asynchronous transmission & reception of messages
- Resilience to program and network outages
- Application Programming Interface (API) for sending and receiving messages
- Simple, unique, global message addressing scheme

ASYNCHRONOUS TRANSMISSION AND RECEPTION OF MESSAGES

E-mail systems, by their nature, cannot assume that the recipient of a given message will be available to receive the message at the same time that it is sent. This concept, known as asynchronous messaging, is an important one—senders and receivers function asynchronously of one another.

When an e-mail message is sent, it is queued at the e-mail server pending delivery. The e-mail client is then free to continue processing, and the responsibility for delivery of the message is placed on the e-mail server. The recipient of the message need not be connected to the network (or even turned on!) at the time that the message is sent.

Unless the recipient of the e-mail happens to have an account on the same server as the sender, the e-mail message must be sent to the appropriate e-mail server upon which the recipient has an account (a "mail queue", if you will). Unless the recipient has an account on the sending e-mail server, the message is passed from one e-mail server to another until the message arrives at the recipient's queue. In every case, when the sending server has passed the message to another, responsibility for delivery falls on the next server. If the message arrives at the destination e-mail server "domain" and is undeliverable (for a variety of reasons), an error or "bounce back" message is returned to the sender. Advanced routing logic allows e-mail messages to be routed from e-mail server to e-mail server along the most efficient path.

Once the message arrives in the recipient's queue, the e-mail recipient is then free to receive e-mail at its leisure, be it minutes, hours, days, or weeks later. When the e-mail has been received, it is generally deleted from the e-mail server queue. The sender may request confirmation that the message has been received, which causes another message to be sent to the sender by the recipient's e-mail server at the time of delivery.

The key concept here is that neither the sender nor the recipient are connected to each other in any way other than via their respective e-mail servers.

RESILIENCE TO PROGRAM AND NETWORK OUTAGES

Due to the nature of the architecture of e-mail systems, they are extremely resilient to program and network outages. If a given network segment is down at the time that a message is trying to be delivered, it simply remains in the server's outbound queue until it is able to be delivered. Likewise, if an e-mail recipient's e-mail program is not running, the message will remain in the queue until it is retrieved. If the message is undeliverable after a given amount of time, the sender receives an error message.

API FOR SENDING AND RECEIVING MESSAGES

Another key messaging concept is that of a common API for sending and receiving messages. E-mail systems commonly have two separate APIs for sending and receiving: Post Office Protocol (POP) for sending and Simple Mail Transport Protocol (SMTP) for receiving. Full messaging systems typically have just one API for sending and receiving messages.

MESSAGES ARE ADDRESSED USING A UNIQUE ADDRESSING SCHEME

In order for a message to be sent and received successfully, it must be addressed using a commonly accepted method that uniquely identifies the recipient on the network. Since the network can be the Internet, the addressing scheme must be able to handle global addresses.

Unless the recipient's e-mail system is in the same domain as the sender, the message must be fully qualified. An example of a domain name would be a company name plus an extension—"ibm.com", for example. A fully qualified e-mail address includes the recipient's account name plus a domain name. The domain name allows the sending e-mail server to figure out which receiving e-mail server to send the message to.

A summary diagram that depicts a simple e-mail messaging architecture is shown below:

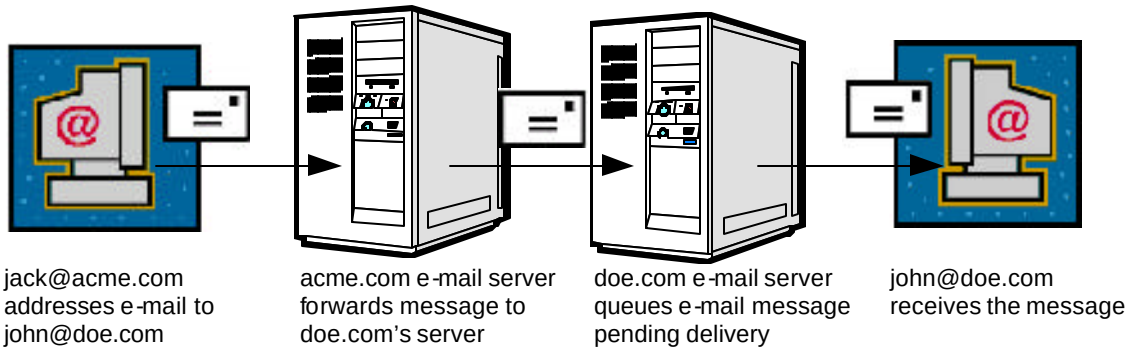

FIGURE 1: SIMPLE E-MAIL MESSAGING ARCHITECTURE

The messaging concepts shown by this diagram are:

- Messages are sent from one PC to another via message servers
- Messages are stored in queues on the message servers
- PCs using messaging communicate asynchronously
- Messages must have a unique recipient address for delivery to succeed

LATE 1980'S TO MID-1990'S: THE CLIENT/SERVER REVOLUTION

Beginning in the late 1980's and continuing through the late 1990's, client/server technologies were the primary type of solution being developed for mission-critical applications. The client/server paradigm, prior to the World Wide Web, consisted of a "fat client" application that typically contained a client application API, business logic, presentation logic, and communications protocols. Client applications would pass API commands to a server, which would process the commands and return results back to the client application. In this paradigm, business logic may be contained in the client application or on the server platform.

CLIENT/SERVER REQUIRES A CONTINUOUS PROGRAM-TO-PROGRAM CONNECTION

One of the requirements for client/server architectures is that the client applications typically remain connected to the server during the duration of the command-processing-result interchange. For instance, a client application would connect to a database server, passing security information to the server. After authentication, the client application would pass a database request, generally in the form of an SQL statement, to the database server for processing. The database server would process the request and generate a relational result set, which would be passed back to the client application.

One of the problems with this architecture was that if the communications link between the client and server were to break due to LAN or applications problems, the chain of command-processing-result would also break and the request would have to be re-started from scratch. In recent years, there have been some high- and continuous-availability products released which help address this issue, but it still remains a problem for client/server applications.

CLIENT/SERVER APIS AND NETWORK PROTOCOLS MUST MATCH

Another problem with client/server was that specific APIs and network protocols were required between the client and the server, unless complex "bridge" middleware was installed in between to provide translation. For example, an Oracle PL/SQL client application normally cannot communicate with a Sybase database server unless an Oracle-to-Sybase gateway middleware server is installed. In another example, an Oracle PL/SQL client application on a Novell network (SPX/IPX) cannot communicate with an Oracle database server on a UNIX server running TCP/IP without some intervening middleware to provide the protocol translation. This problem has been solved to some degree by increasing acceptance of TCP/IP as the standard enterprise-wide communications protocol.

In the late 1990's, another major drawback of the "classic" fat client/server architecture emerged – maintenance of client applications became prohibitively expensive and time consuming. For instance, a major insurance company deployed a client/server customer service application written in PowerBuilder to 10,000 PCs distributed throughout the U.S. In order to update the client application, an automated process had to be instituted to replace program executables and DLLs on all 10,000 PCs via the company wide area network, which took a tremendous amount of bandwidth and a long time to complete, not to mention the difficulty in maintaining the process. If the update were to fail on a PC, a support event was created resulting in tremendous expense to repair the application, not to mention the opportunity cost of the user's downtime.

MID- TO LATE 1990'S: WEB/INTERNET, THIN CLIENTS AND "RARELY CONNECTED" USERS

In the mid- to late 1990's, several shifts in the way that companies conducted business began to appear:

- Some users became "rarely connected," connecting to the company network sporadically, many times over low-bandwidth, highly volatile dialup or wireless connections.
- The Internet and the World Wide Web gained popularity at an astounding rate.
- The availability of cheap, reliable Internet connections caused many companies to begin utilizing the Internet for field personnel needing connectivity to the corporate LAN instead of expensive point-to-point solutions (like T1s) or proprietary dial-up modem lines. These were termed Virtual Private Networks (VPN's). VPN's included robust end-to-end encryption, which allowed sensitive corporate data to be carried by the Internet without compromising security.

The advent of these major changes caused a rapid shift in the way that new applications were developed and in the underlying architectures that supported them.

MANY USERS BECAME "RARELY CONNECTED"

Another major trend in the 1990's was for corporate computer users to be "rarely connected" to the corporate LAN. There were many driving reasons for this shift in corporate computing:

- Sales Force Automation (SFA) & Customer Relationship Management (CRM)
- Telecommuting
- Personal Digital Assistants (PDAs), Cell Phones, and Pagers

Sales Force Automation (SFA) & Customer Relationship Management (CRM)

The concepts of Sales Force Automation and Customer Relationship Management came into vogue in the late 1990's. The idea was to provide sales and customer service representatives with as much information as possible relating to the customer, regardless of whether they were using a desktop PC, a laptop, or a Personal Digital Assistant. Many of these workers spend a high percentage of their time away from their offices meeting

with customers, attending trade shows, and the like, which results in their becoming "rarely connected" users.

Whether the user was a sales representative that needed access to a customer database or a technical support engineer that needed remote access to a case management system, the issues were the same. Indeed, the trend was for these systems to be one and the same, with the information content customized to the role of the person.

For example, the sales representative would have access to the customer database, order entry system, and a marketing encyclopedia, and would be able to check the status of support cases, while a technical support representative would have access to the customer database and read/write access to the support case system. The holy grail of SFA and CRM was that one integrated system, regardless of the platform being used (i.e. laptop, palmtop, desktop, etc.) should include all of those functions.

Telecommuting

Telecommuting began gaining popularity in the 1990's for a variety of reasons:

- Natural disasters closed some company buildings for months, especially in California, and having workers telecommute became a highly desirable option. This also became a perk for many employees, and companies began using telecommuting as a recruiting tool.
- Powerful laptop computers with desktop functionality became available for a reasonable cost.
- Availability of higher bandwidth data connections to the home increased with DSL, ISDN, and Cable Modems.
- Some workers began "job sharing", where two or more people, each working part time, fill one full-time position.

Since telecommuters typically use company-provided laptops or their own home PCs, the idea of a thin client makes a lot of sense. When a company's internal systems department loses control over what is installed on a PC (since the company may not even own it), the less company software that is permanently installed on it, the better.

Personal Digital Assistants (PDAs)

FIGURE 2: A PERSONAL DIGITAL ASSISTANT (PDA)

In addition to laptops, palm-sized Personal Digital Assistants gained a foothold in the computing market, either supplementing or replacing them in some applications. The availability of PDAs with large memory capacity and reasonable processing speeds fueled the development of embedded, "small fingerprint" applications and databases which could communicate with larger LAN- and mainframe-based databases.

Many companies are aggressively integrating these devices into their "legacy" applications for a variety of reasons, including inventory, order entry, case management, as well as the normal PDA functions which include address lists, calendar, to-do lists, notes, and Internet (Web and e-mail) access. Many of these applications require integration between large database systems and smaller embedded systems, over low speed, unreliable networks.

Internet- and Web-enabled PDAs, Cell Phones, and Pagers

FIGURE 3: INTERNET- AND WEB-ENABLED DEVICES

(From left: Mitsubishi T250 phone, Palm VII, Motorola Talkabout T900 2-way pager, and Motorola PF1500 "1 1/2 way" pager)

A major development that began in the late 1990's was the integration of the Internet into PDAs, cell phones, and pagers. PDAs and cell phones which could send and receive e-mail and surf the Web were introduced, and pagers became two-way, enabling short messages including e-mail to be sent and received.

The trend in the industry is for all of these devices to be integrated into one device—an "information appliance" with wireless Internet connectivity, PDA, and phone capabilities.

COMMON ELEMENTS

All of the above-mentioned trends share many of the same characteristics:

- Limited Bandwidth
- Distributed Application Requirements
- Questionable Connection Quality

Limited bandwidth

Compared with a dedicated LAN connection, which typically has a bandwidth of either 10 megabits per second (Mbps) or 100 Mbps, most of these devices have limited bandwidth.

Some examples follow:

- Wireless data - two-way pagers, Palm VII, etc: 9.6 Kbps to 19.2 Kbps (.0096 to .0192 Mbps!)
- Dial-up "56K" modem: 56 Kbps (limited by FCC to 51 Kbps, or .051 Mbps!)
- ISDN (Integrated Services Digital Network): 64 Kbps or 128 Kbps (.064 Mbps/.128 Mbps, respectively)
- DSL (Digital Subscriber Line): between 256 Kbps and 1440 Kbps (.256 Mbps / 1.44 Mbps, respectively)
- Cable modems: between 320 Kbps and 2 Mbps

Distributed application requirements

With many of these applications running on Web, laptop, or hand-held devices, most of the business logic is performed on another "mid-tier" platform—especially since the same application may need to run on a variety of these devices! The Java programming language may make ubiquitous thin-client based applications a reality some day, but at this point Java is still in the process of being developed and deployed on platforms other than "traditional" Web browsers.

Questionable connection quality

Wireless and modem connections are notoriously finicky—and even ISDN, DSL, and cable modems have sporadic outages. Any of these can wreak havoc on traditional client/server applications, which require constant connections.

Since most of these users are rarely connected to the network, and when they are, it isn't for a long time, all of these uses dictate that information be ready for distribution at the time that the user becomes connected to the network. Short connection times, unreliable connections, limited bandwidth, limited processor speed and memory, and Web-based applications make fat client applications not practical for any of these uses. The push for thin clients dictates that as much of the logic as possible be moved from the client platform to a "middle tier" application server.

And all of this needs to be available 24 hours a day, 365 days a year! It's a different world.

LATE 1990'S AND EARLY 2000'S: APPLICATION SERVERS, CORPORATE WEB PORTALS, AND BUSINESS-TO-BUSINESS APPLICATIONS

APPLICATION SERVERS

In the early 2000's, the push is now toward thin-client applications that are primarily Web-based. Even "classic" client/server applications are being re-architected to move as much business logic as possible off of the client platform to a middle-tier server, known as an Application Server. A variety of vendors have released Application Servers, including Sybase Enterprise Application Server, Lotus Domino, and SilverStream.

CORPORATE WEB PORTALS

Business requirements in the 2000's require constant access to corporate data by employees, customers, and business partners anytime, anywhere. The most common mechanism by which to accomplish this is via the World Wide Web. This type of access has been termed "Web Portals", in which a "portal" is opened between legacy corporate data and the outside world. Depending on who is accessing the portal, different information is available. Security becomes a major issue, as does availability, as the Internet economy expects 24x7 availability, 365 days per year—something that many of the legacy corporate systems (especially mainframe) were not designed to provide.

Web portals also introduce the concept of "personalized access" to corporate resources—after identifying yourself to a Web site, you are presented with screens of information that have been customized based on what you are interested in and/or are allowed to access.

Corporate Portals are often mixed up with other types of Internet portals, including Web portals such as Yahoo!. They are very different concepts.

BUSINESS-TO-BUSINESS (B2B) APPLICATIONS

The final major initiative in the early 2000's is the focus of Business-To-Business (B2B) applications. These applications are used to facilitate inter-business processes such as supply-chain management, funds transfer, sharing of data for "coopetition", etc. As you might guess, the majority of these applications are Internet-based, and security is an even bigger concern—many of these transactions may involve hundreds of thousands or even millions of dollars!

MESSAGING HELPS SOLVE MANY OF THESE PROBLEMS!

First, it is important to understand that messaging is not the "holy grail" which solves all of these problems. Like any technology, messaging has its strengths and weaknesses, and the critical factor is ensuring that the technology is applied correctly.

Briefly, messaging products enable heterogeneous programs to talk to each other across a network of unlike components—processors, operating systems, subsystems, and communications protocols—using a simple and consistent API.

Messaging, in and of itself, does not necessarily solve all of these problems, but a combination of messaging and complementary products, when architected correctly, can solve many of these issues.

SECTION 2. MESSAGING CONCEPTS

BASE MESSAGING ARCHITECTURAL COMPONENTS

Now that you have an understanding of the business factors that are driving applications architects toward messaging architectures, let's drill down into the mechanics of messaging. All messaging architectures share four basic elements:

- Messages
- Queues
- Queue Managers
- Nodes

Each of these elements will be discussed in detail in this section, along with some other important messaging concepts.

MESSAGES

The basic unit of any messaging architecture is the message. Messaging technology allows programs to communicate with each other by sending each other data in a generic format, known as a message, instead of calling one another directly. For example, a "classic" client/server application would communicate using a proprietary vendor API, such as Sybase Open Client/Open Server. These APIs were designed for a specific purpose (Open Client/Open Server is a relational database API), and as such, they do not lend themselves to other uses.

Before messaging, if program "A" wished to communicate with program "B", it would have to make a connection using one of a variety of network protocols (i.e. TCP/IP Sockets, Novell SPX/IPX, etc). Significant coding and knowledge of network protocols was essential, as both PCs needed to be running the same protocol and API. Continuous network connections were also essential; if the network went down, applications had to have their own recovery logic built-in.

Applications that use pure messaging, on the other hand, function at a lower level in the hierarchy—instead of proprietary APIs and network dependencies, messaging applications simply send "messages" to one another via a simple API. At this level, more application coding is required, however—one reason that proprietary APIs exist is because they provide a higher level of functionality. We will cover this in detail later.

MESSAGES DEFINED

What exactly **is** a message? Merriam-Webster defines it as follows:

> Main Entry: [1]**mes·sage**
> Pronunciation: `'me-sij`
> Function: *noun*
> Etymology: Middle English, from Old French, from Medieval Latin *missaticum,* from Latin *missus*, past participle of *mittere*

Date: 14th century
1 : a communication in writing, in speech, or by signals
2 : a messenger's errand or function
3 : an underlying theme or idea

A more germane definition would be as follows:

Main Entry: **[1]mes·sage**
Pronunciation: `'me-sij`
Function: *noun*
Etymology: Late twentieth century American, from International Business Machines and others
Date: 20th century
1 : a generic unit of information (bits) which may be sent from one computer program to another

Messages may contain any data, so long as it may be represented as a stream of bits! So long as the sending program and receiving program know what to expect from one another, the messaging system does not care what the format of the data is that it is delivering. With messaging, program "A' makes a simple API call specifying which "node," or recipient, to send the message to, and the messaging architecture takes care of the rest.

Applications do not have to be using the same network protocol, operating system, language, etc.—the messaging architecture takes care of required translations. If the receiving application is down, or the network is down, the message queue stores the message until the recipient is ready to receive it (see queuing in the next section).

EXAMPLE MESSAGES

Question: what exactly can an application place in a message? Answer: anything!

- E-mail
- Database requests & results
- Binary data (images, music, etc.)
- Entire files (i.e. IBM MQSeries allows messages up to 100 megabytes in size, although most vendors have a two-to-four megabyte message size limit)

BASE ELEMENTS OF A MESSAGE

All messages, regardless of vendor architecture, contain the following base elements:

Message type

The message type indicates the basic purpose of a message, such as 'send', 'reply', 'status', etc.

Destination

This is the message recipient's unique address. The format varies depending on vendor, but it must uniquely identify the recipient on the network. Some messaging systems

allow multiple destinations to be specified for one source message; this is commonly known as multicasting.

Message data

This is the actual "body" of the message, and may literally contain anything made up of bits. The maximum size of a message varies depending on the vendor, but multiple messages may be "chained" if necessary.

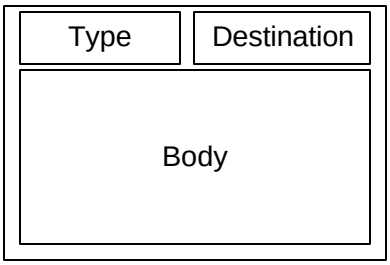

FIGURE 4: ELEMENTS OF A MESSAGE

QUEUES

Messages are placed into queues, which are containers that hold messages until they are delivered. Queues allow the sender and receiver to function asynchronously--in other words, if the receiving application is not available at the time the message is sent, the message remains in the queue until ready to be received. This allows programs to run independently of one another, at different speeds, times, and locations, without requiring a logical connection between them. Messages may be defined with a timeout parameter, which allows a message to expire if not delivered within a given period of time. Some messaging systems can notify the sender if a message expired, and others allow the sender to query the status of a message to check if it has been delivered successfully.

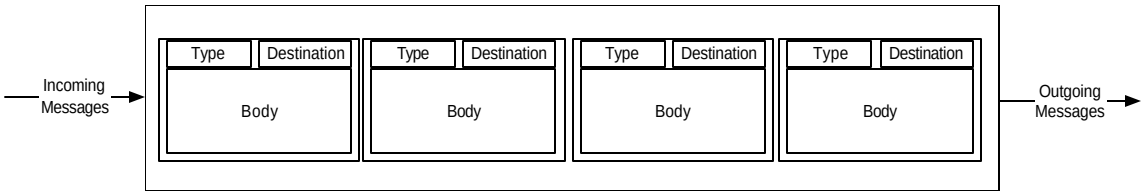

FIGURE 5: A MESSAGE QUEUE

QUEUE MANAGERS

Queue managers (QMs) manage the message queues, serving as the intermediary between nodes sending messages to one another. The QM stores messages in a queue until recipients are ready to receive them.

Some QMs are capable of multicasting. Multicasting allows a message sent from a single application to be distributed to multiple recipients – this is analogous to sending an e-mail message to a distribution list. Intelligent e-mail and messaging systems require the sender to transmit only one copy of the message, and the server then sends multiple copies to the specified recipients.

Publish/subscribe and push technologies utilize this concept--more detail on this later in the paper.

NODES

Nodes are the logical ends of a network connection. Applied to messaging, a node is an application that is either sending or receiving messages from another node.

OTHER MESSAGING CONCEPTS

This section describes some messaging concepts that you will find useful as you read this paper.

ASYNCHRONOUS MESSAGING

By definition, messaging is an asynchronous technology. In other words, the sender and recipient do not have to be connected to the network at the same time—messages will be queued until the recipient is available.

The sending application does have the option of checking to see if a message has been delivered. Some messaging systems can also notify the sender if a message has not been received within a given timeout period.

SYNCHRONOUS MESSAGING

Messaging systems may also be used in synchronous mode. The inverse of asynchronous mode, synchronous messaging involves sending a message to a recipient and then waiting for a response. While synchronous transactions were not the intent of messaging systems, most vendors support this mode to support classic client/server transaction characteristics.

"PUSH" VS. "PULL" TECHNOLOGIES

In the late 1990's, "push" technologies were all the rage. The idea was that a server could "push" information to a client application—for instance, a news alert could be sent to a client application when it occurs instead of the application having to "poll," or "pull" the information on a timed basis. This has evolved into a variation of push, known as "publish/subscribe," which is described below.

PUBLISH/SUBSCRIBE TECHNOLOGIES

Publish/subscribe, as the name implies, is the process in which a client application "registers," or "subscribes" to a particular stream of information. When information is "published," it is sent to the list of subscribers using multicast technology.

GUARANTEED "ONCE-ONLY" DELIVERY

Another important messaging concept is that of guaranteed, "once only" delivery of messages. This means that when a message is sent, the messaging system guarantees that it will be delivered, once and only once.

TRANSACTIONAL INTEGRITY / GUARANTEED UNITS OF WORK

One of the newer messaging features is the ability to group several messages together into a single "unit of work." This means that all messages live or die based on the assumption that they all must succeed or none of them succeeds. As an example, MQSeries can be used to send update messages to several database systems (providing they are XA compliant), and the MQ server can be used to provide multi-phase commits across the database platforms. If any of the messages fails (e.g. the database update failed), then all of the other updates are automatically rolled back.

MESSAGE BROKERS AND MESSAGE HUBS

Traditionally, one of the fundamental difficulties in implementing messaging applications is that they are difficult to develop since they do not have client/server database connectivity built in to them. As a result, access to database systems and other non-messaging applications had to be written into the application, which created a significant amount of development and maintenance effort.

Message brokers, also known as message hubs, solve many of these problems by allowing messages to be examined as they pass through the message server. Rules may be defined that cause actions based on a rule matching a particular characteristic of a given message; actions may include database access, interfaces to other messaging or middleware systems, or business logic.

For instance, a message broker could be used to provide access to a relational database for a messaging client. The client application would place a SQL statement inside of a message and send it to a queue manager. After the queue manager enqueues the message, the message broker would take the SQL statement from the message, submit it for processing against the database, and return the results to the sending application via a "reply" message. This processing could take place using either synchronous or asynchronous messaging.

Examples of these products include Sybase Enterprise Event Broker and IBM MQSeries Integrator.

MESSAGING APPLICATION PROGRAMMING INTERFACES

As mentioned previously, applications that communicate using messaging do so via an Application Programming Interface (API). Depending on the vendor, the API may include program function calls (supported by most major languages) as well as Component Object Model (COM) and callable objects (such as Visual Basic VBX objects).

Generally, most major languages are supported, including C, C++, COBOL, Visual Basic, PowerBuilder, Java, etc. This includes Web-based "thin" clients, which typically use Java and/or JDBC connections.

SECTION 3. MESSAGING ARCHITECTURES

Unlike some of the more traditional architectures that include client/server, messaging is extremely flexible. While flexible may also mean powerful, the fact that there are so many choices when architecting a system means that it is easy to design it the wrong way.

APPLICATION TO APPLICATION ARCHITECTURES

Messaging systems have the following possible architectures:

ONE-TO-ONE

The one-to-one architecture is a true "peer-to-peer" system, where one program sends a message to another, and the receiving program can reply if necessary. E-mail in its simplest form is a one-to-one system (although it may also be one-to-many). Another example of one-to-one would be a "chat" program, in which two PC users chat with one another via a split-screen windowed application.

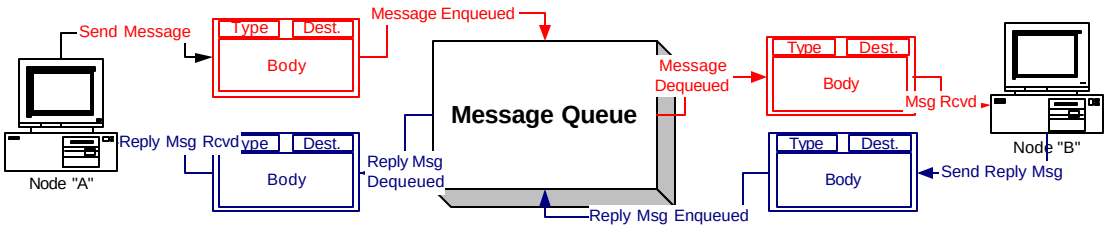

FIGURE 6: ONE-TO-ONE MESSAGING ARCHITECTURE

ONE-TO-MANY

A one-to-many architecture, as the name implies, is one program sending a message to many recipients. This may employ multicasting technology, in which one message is sent with multiple recipients, or it may be the same message sent many times (depending on the messaging system being used). E-mail is a logical application of one-to-many, as are publish/subscribe applications.

An example of a one-to-many application which makes use of publish/subscribe would be SkyTel's nationwide paging system. Their pagers have the capability of receiving news updates from a variety of sources. The individual paging subscriber registers interest (subscribes) to one or more categories via a Web interface. When news arrives, the paging system automatically sends it out to the list of pagers that subscribed to that category.

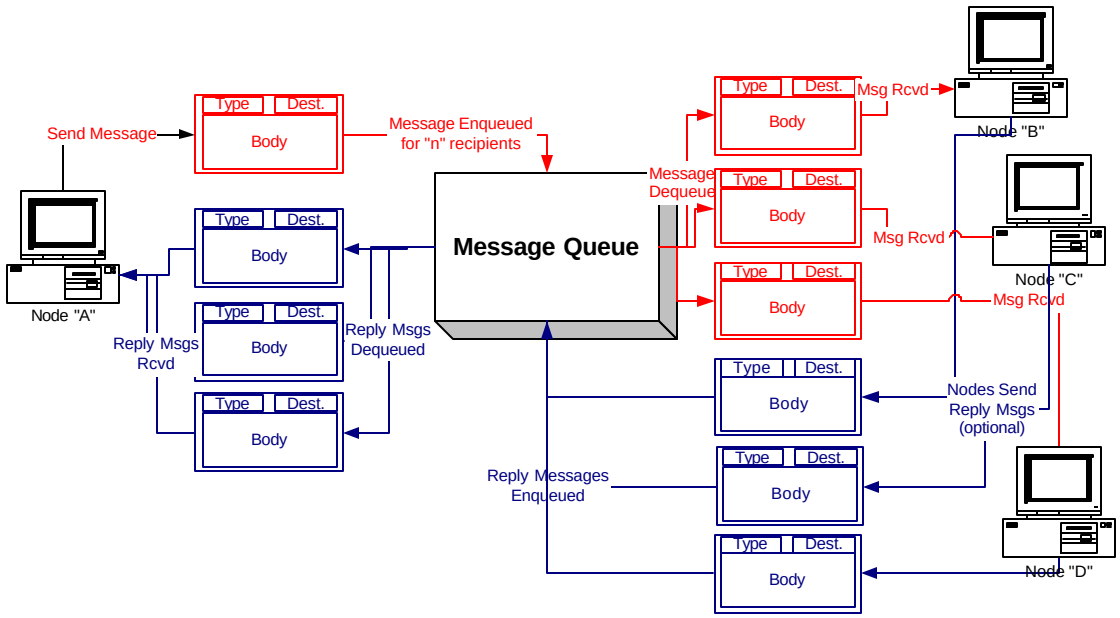

FIGURE 7: ONE-TO-MANY (PUBLISH/SUBSCRIBE) ARCHITECTURE

MANY-TO-ONE

Many-to-one architectures involve many sending applications communicating with one receiving application. This is essentially a client/server architecture, where many clients communicate with a single server. Client/server architectures are possible with many-to-one messaging technology; in particular, decision-support client/server applications lend themselves to messaging due to their lower time sensitivity and asynchronous nature.

A decision support application may require quite a bit of processing in order to formulate a result set. With messaging, the request is sent to the server, which begins processing. The client application is then freed to continue other processing; when the result is available, the server can send a reply message to the sender that notifies it to retrieve the result set. If the sending application is no longer active when the result set is ready, it can be queued until the application is ready to receive it.

As an example, a manager may request a report that takes many hours to process—if it is ready after she has gone home for the evening, the results will be waiting for her upon her return the next morning.

MANY-TO-MANY

Many-to-many applications are very complex and really show the power of messaging. One of the most popular uses of many-to-many architectures are workflow applications, in which several complex processes must be completed, sometimes in a given sequence, in order to complete a particular objective. If this is confusing, an example may help.

An excellent example of a workflow application may be seen on the trading floor of major stock exchanges. In order for a successful stock trade to be completed, several steps must be completed (these steps are the workflow). Some steps must be completed sequentially, and other steps may happen in parallel. If any of these steps fails, the trade must be declared void. In addition, SEC regulations require that all of these steps be completed within a fifteen-minute time window, or the trade is void. An example of some of the steps involved in a trade could be as follows (note that this does in no way depict a real trading system—your mileage may vary!).

1. A Web-based day trader decides to purchase 1,000 shares of stock in the Burgundian Crossbow Company (ticker symbol BCBC). After signing on to her account and checking the stock price, she enters the trade order and presses the "OK" button.
2. At the Web-based trading company (STOCKSRUS.COM), an application server receives the buy order and generates a message with the following data items in the body of the message:
   - The day trader's account number and password
   - The ticker symbol of the stock she wishes to purchase
   - The quantity of shares she wishes to purchase
   - A timestamp indicating the time the trade request was received
3. The ticker symbol must be validated to be correct:
   - A copy of the message is sent to a trading system to validate the ticker symbol
4. The account number and password of the day trader must be verified:
   - A copy of the message is sent to the system that manages subscriber accounts for verification of the account number and password
5. The current price of the stock must be retrieved, and the total amount of the purchase must be calculated:
   - A message is sent to a trading system to retrieve the current "buy" price of the stock; this also triggers a search to locate one or more sellers that add up to 1,000 shares of BCBC stock
6. The purchase amount is sent to the subscriber accounts system to verify that adequate funds are available for the purchase:
   - The purchase must be approved by a trading company representative; a message is sent to their PC, and they click "approve" if everything seems to be in order
7. The complete "package" is sent to the trading floor system, which executes the trade.
8. In order for the trade to execute, four transactions must be executed (committed) together:

- The purchase amount is debited from the purchaser's account;
- The purchase amount is credited to the seller's account;
- The shares are electronically debited from the seller's brokerage account;
- The shares are electronically credited to the buyer's brokerage account.

9. The trading floor system sends a confirmation message to the trading company's Web application server, which sends an e-mail to the purchaser confirming the purchase.

There are many complex "workflow" applications like this in today's business climate, and messaging is a perfect architecture for many of them. Indeed, this type of application is nearly impossible to do with anything but a messaging architecture—in fact, TIBCO built their company on trading floor applications (more on TIBCO in the next section).

SECTION 4. MESSAGING VENDORS

There are many messaging vendors, including some of the biggest names in the industry as well as some of the smallest. The market leader at the time of this writing is IBM MQSeries, with a 65% market share in 1999; other leaders include TIBCO and Microsoft.

IBM MQSERIES

MQSeries is developed and maintained by IBM's Hursley, England division—the same division that develops and maintains CICS.

MQSeries supports over 35 server and/or client operating systems:

- AIX (RS/6000)
- HP/UX
- OS/2
- OS/390 (MVS)
- OS/400 (AS/400)
- Sun Solaris
- Tandem NSK
- VSE/ESA
- Windows 3.x, 95, 98, NT, and 2000

According to IBM literature, MQSeries attained a 65% market share in 1999 (WinterGreen Research). It is used by over 7,000 customers, including seventy of the Fortune 100 corporations and 2/3 of the world's leading banks. One major US organization uses MQSeries to transmit over 250 million messages per day!

MQSeries is a mature product (IBM defined the first message-oriented API in the early 1990's), with several additional components to help solve architectural issues:

MQSERIES INTEGRATOR

MQSeries Integrator (MQI) is a middleware component that allows MQSeries to act as an "Application Integration Broker," or "message hub" between messaging applications. MQI acts as a "way station" between applications, allowing logic to be placed in the middle tier (and not in the application). It intercepts messages and provides real-time processing as they arrive in message queues.

The Application Integration Broker market grew 90% in 1999 to over $550 million, and according to WinterGreen Research, IBM holds a 12% market share, at $66.5M. Other competitors in this market space include NEON (New Era of Networks) (10.8%), Mercator (10%), and TIBCO (9.1%). There are over 50 vendors in this market space, although the top four vendors hold nearly 50% of the market. Interestingly enough, IBM has bundled several NEON products into MQI.

MQI contains three significant value-added features:

PUBLISH / SUBSCRIBE

Publish/subscribe, as described earlier, allows MQ client applications to "subscribe" to particular messages; when a message arrives that matches the subscription criteria, it may be "published" to one or more MQ clients.

MESSAGE FORMATTER

Messages may be parsed and formatted between sending and receiving nodes, providing the ability to transform data. This functionality is provided by the NEON (New Era of Networks) Formatter utility, which is bundled with the MQI product.

RULES-BASED MESSAGE EVALUATION

This feature allows messages to be converted into computer commands (such as spawning an executable program on the MQ server platform), which may carry out actions based on the results. An example of this would be an application that performs database access based on an incoming message (which could contain the criteria for the access); the results generated could then be returned to the calling MQ application using a "reply" message. This feature allows rules to be defined that specify which message criteria to match, and which command to execute when a match occurs. This is also based on a NEON product that is bundled with the MQI product.

"SEAMLESS" DATABASE INTEGRATION

Using the features outlined previously, MQI can be used as middleware between normally incompatible database systems. Using rules-based message evaluation and the message formatter, relational data being moved from one platform to another may be transformed "on the fly," on the MQ server, not at the application (or gateway) level.

XML SUPPORT

IBM has announced that MQSeries Integrator version 2.0 has new XML (Extensible Markup Language) support. This means that MQI can take data from existing applications and transform it into XML, which is usable by any XML-enabled application including Web browsers. MQI can also take XML data and format it for use by older non-XML applications.

MQSERIES WORKFLOW

MQSeries Workflow is a product which allows complex workflow applications to be modeled and implemented using messaging. A workflow application is defined as one in which many different stages must be completed, often by different people and/or systems,

and if any of the stages are not successful, the entire workflow transaction fails. IBM defines workflow as "a number of related tasks to achieve a defined business outcome." Characteristics for a workflow application include:

- Multiple activities
- Many people
- Various procedures
- Multiple information sources

Workflow applications bind the procedures and logic with the people and the information sources that perform them. It is no longer acceptable to engineer "vertical", custom designed processes that are difficult to maintain as business needs change. Workflow applications attempt to make these processes more "horizontal", or reusable, in design—this makes them easier to adapt and re-use as business needs change.

Some of the advantages of a workflow application are:

- Faster execution of processes
- Higher productivity through automation
- Better service to customers at reduced costs
- Improved quality of process execution
- Process enforcement helps achieve ISO 9000 compliance

MQSeries Workflow provides a graphical way to define workflow applications. A workflow model is built first which defines the various stages in the workflow; after the model is completed, it is "translated" to run the actual business processes. For each stage in the workflow, MQ Workflow, using MQSeries, moves the work from one person to the next, ensuring delivery from person to person (or system to system). MQ Workflow can leverage existing systems as stages in a workflow, or new systems may be built and "plugged in" as the system is developed.

IBM has announced that it has plans to add support for MQSeries Workflow to its Enterprise Information Portal, which will allow Web-based workflow applications to be developed rapidly.

WORKFLOW APPLICATIONS

Some example workflow applications are:

- Trading floor applications
- Insurance underwriting
- Technical/customer service case tracking
- Tax return processing
- Credit application processing

MQSERIES WORKFLOW SUPPORTED PLATFORMS

Server platforms

The MQSeries Workflow 3.2.2 server runs on the following operating systems and is easily scalable from Windows up to mainframes:

- Microsoft Windows NT
- Microsoft Windows 2000
- IBM OS/2
- IBM AIX
- HP-UX
- SUN Solaris
- IBM OS/390

Client platforms

The MQSeries Workflow 3.2.2 client runs on the following operating systems:

- Microsoft Windows NT
- Microsoft Windows 2000
- Microsoft Windows 95/98
- Client for Lotus Notes

The MQSeries Workflow 3.2.2 client for Lotus Notes runs on:

- IBM OS/2
- Microsoft Windows NT
- Microsoft Windows 2000
- Microsoft Windows 95/98

MQSERIES BRIDGES

IBM provides a series of MQSeries "bridges" which allow MQSeries to "bridge" into data sources, extending the MQ functionality. Available bridges include:

- CICS/ESA (IBM)
- ODBC Data Sources (MQSoftware's SQL Bridge for MQSeries)

MQ bridges are available from IBM and MQSoftware. MQSoftware is an independent software vendor specializing in add-on MQSeries products. MQSoftware's SQL Bridge product for MQSeries allows MQSeries to access any ODBC compliant data source, including DB2, Oracle, Sybase, Informix, etc.

MQSERIES ADAPTORS

IBM has a family of custom MQSeries "adaptors" that may be purchased or built to access non-traditional data sources and applications. For example, various vendors have built MQSeries adaptors for MRP (Material Resource Planning) applications, including SAP R/3, Baan IVb, and J.D. Edwards OneWorld.

Adaptors allow MQSeries to transparently access data that are stored in proprietary back-end systems without the necessity of bridges or transformations. IBM has announced intent to provide an "MQSeries Adaptor Offering" tool in June 2000 that will allow custom adaptors to be written by third parties. The adaptor builder tool will also support XML.

IBM also intends to provide an MQSeries Internet pass-thru function that can be used to simplify the passage of MQSeries channel protocols through corporate firewalls and to be sent using the HTTP protocol.

MQSERIES EVERYPLACE

IBM has been involved in the "palm" computing market for some time. They were the first to "OEM" (Original Equipment Manufacture) the Palm Computing platform, selling it under their own name (with a black case). Their latest foray into the market is MQSeries Everyplace, which extends the MQSeries model to laptops and personal digital assistants (PDAs). IBM has termed this "Pervasive Messaging", and these rarely connected devices are the perfect vehicles for messaging applications, as they are oftentimes subject to "hostile communications environments."

As of June 15, 2000, the product will be Generally Available for the Palm OS (Palm Computing and Handspring PDAs), as well as Java on Windows (Windows CE, 95, 98, NT, 2000) and EPOC32 (Psion PDAs). IBM has stated intent to continue porting it to additional devices, including cell phones and embedded systems (like sensors and probes).

TIBCO

Palo Alto, California-based TIBCO (The Information Bus Company) (NASDAQ: TIBX), had its genesis in providing Wall Street vertical message-based trading systems. They have now shifted their focus to e-business, although trading floor systems remain a significant part of their business. A principal stockholder in TIBCO is Cisco Systems, and Reuters is a majority stockholder.

TIBCO is the successor to a portion of the business of Teknekron Software Systems, Inc. Teknekron developed software, known as the TIB technology, for the integration and delivery of market data, such as stock quotes, news and other financial information, in trading rooms of large banks and financial services institutions. In 1992, Teknekron expanded its development efforts to include solutions designed to enable complex and disparate manufacturing equipment and software applications—primarily in the semiconductor fabrication market—to communicate within the factory environment. Teknekron was acquired by Reuters in 1994. Following the acquisition, continued

development of the TIB technology was undertaken to expand its use in the financial services markets. In January 1997, TIBCO Software Inc. was established as an entity separate from Teknekron.

TIBCO claims to have 600 enterprise customers, including Cisco Systems, Yahoo!, Ariba, NEC, 3Com, Sun Microsystems, SAP, Philips, AT&T and AOL/Netscape.

TIBCO's product line consists of the following products:

TIB/RENDEZVOUS

TIB/Rendezvous is the foundation of the ActiveEnterprise messaging system. It supports publish/subscribe as well as request/reply messaging, and facilitates personalized information delivery. TIB/Rendezvous leverages the networking protocols of the Internet to offer a range of service levels in the delivery of information and the execution of transactions. It provides efficient, reliable information delivery and high scalability, and can be embedded in an enterprise's existing information system.

TIB/ETX

TIB/ETX is a transaction-based messaging system designed for use in environments that require a greater degree of transaction management and control than is provided by a standard messaging solution. TIB/ETX provides a transactional form of publish/ subscribe messaging similar to traditional computer transaction models.

TIB/OBJECTBUS

TIB/ObjectBus is an object request broker, or ORB. ORBs enable computer systems to operate more efficiently by employing reusable, self-contained pieces of software code known as objects. TIB/ObjectBus allows TIBCO's products to integrate with CORBA 2.0, a major programming standard for object-oriented applications. TIB/ObjectBus can be fully integrated with TIBCO's messaging software, combining the efficiency of an object-oriented computing model with the scalability, performance and ease of use benefits of TIB/Rendezvous.

TIB/INCONCERT

TIB/InConcert allows the definition of document-oriented process workflow, as specified through graphical user interfaces. TIB/IntegrationManager manages and executes automated system-to-system processes, while TIB/InConcert manages and executes the document-oriented workflows that involve human and computer processes. Together TIB/InConcert and TIB/IntegrationManager span the full range of requirements for process and workflow automation and execution within and between enterprises.

TIB/INTEGRATIONMANAGER

TIB/IntegrationManager controls the flow of information and system-to-system communication among applications and components in the TIB environment. It allows the enterprise to define business rules that govern information processing between applications and where information should go and under what conditions.

TIB/IntegrationManager coordinates the message transport and transformation functions of the TIB products for internal process automation and B2B trading systems.

TIB/MESSAGEBROKER

TIB/MessageBroker is a message routing and transformation system. It combines and transforms data from applications into formats that can be understood by other applications, and routes data according to pre-defined rules. TIB/MessageBroker also allows an enterprise to conduct transactions and exchange information with customers and business partners. Unlike many competing technologies, TIB/MessageBroker requires no independent database or third-party messaging system.

TIB/ADAPTERS

TIB/Adapters are software components that link applications to the TIB environment, thus enabling these applications to communicate with each other. A TIB/Adapter is the single point of integration for the application. TIB/Adapter products connect leading enterprise applications and complementary middleware products to the TIB environment. A series of standard TIB/Adapters are designed to link applications and other software developed by SAP, Siebel Systems, PeopleSoft, IBM and Oracle, among others, to the TIB.

TIB/ADAPTER

TIB/Adapter SDK allows TIBCO customers and systems integrators to build custom TIB/Adapters to link applications to the TIB environment. The TIB/Adapter SDK product provides a common framework for the rapid development of new TIB/Adapters.

TIB/HAWK

TIB/Hawk is TIBCO's monitoring and applications management product. Through a graphical user interface, TIB/Hawk can be configured to monitor systems and applications in a local or wide area network and act autonomously when pre-defined conditions occur.

TIB/BUSINESSCONNECT

TIB/BusinessConnect, scheduled for release in 2000, is a server software product built around TIB/IntegrationManager and TIB/MessageBroker that will enhance TIBCO's capabilities for B2B trading activities and simplify the implementation of B2B solutions. TIB/BusinessConnect is used to define trading relationships through a graphical user interface and then execute the resulting transactions between trading partners. TIB/BusinessConnect supports the core technologies and standards for B2B (business to business) e-commerce.

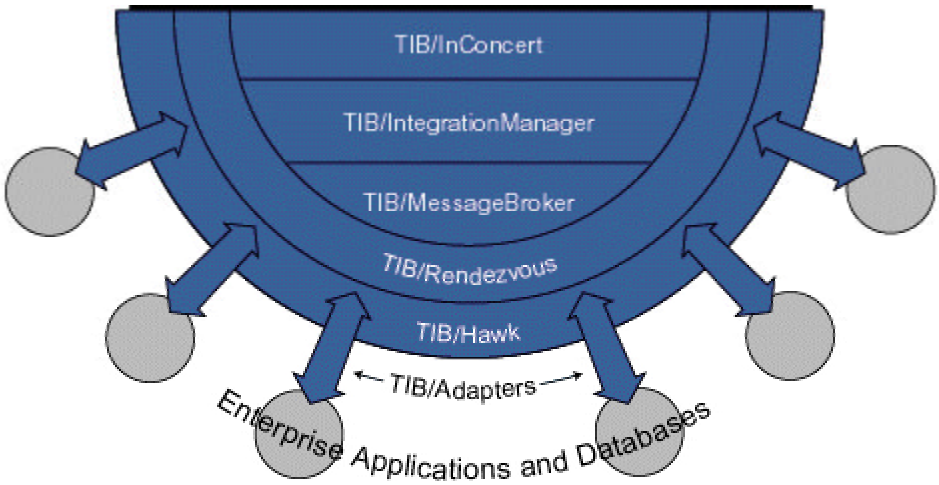

FIGURE 8: TIBCO ACTIVE ENTERPRISE PRODUCTS

The foundation of the TIBCO messaging product line is the TIB/Rendezvous product. This product is available on the following platforms:

- AIX (RS/6000)
- Digital UNIX (Alpha processors)
- HP-UX
- Linux (Intel processors)
- Sun Solaris (Sparc processors)
- DEC VMS (Alpha processors)
- Windows 95/98/NT/2000 (Intel processors)

Note the lack of IBM AS/400 and OS/390 support, which eliminates many potential Fortune 1000 customers.

MICROSOFT MQ

Microsoft (NYSE: MSFT) MQ (MSMQ) is an interesting "niche" product that happens to be published by the 800-pound gorilla of the industry. Starting with NT Server 4.0, it was built into the product, making it the most cost-effective messaging system on the market today. As Microsoft has discovered, the easiest way to build market share is to give a product away!

Unfortunately, MSMQ is only available on 32-bit Windows platforms (95/98/NT/2000), which significantly limits its appeal to the Fortune 1000 market. There are a series of "connectors" to interface MSMQ into other messaging systems; these are described later in this section.

MSMQ APIs

MSMQ has two APIs available for applications to use:

- C function calls
- COM (Component Object Model) objects (C++/Visual Basic/Java applications)

COM is required in order to use many of the more advanced features of MSMQ (like distributed commit logic using NT's Distributed Transaction Coordinator or the Microsoft Transaction Server). As it sounds, application development using COM requires a serious investment in Microsoft technology.

One of MSMQ's limitations is message size, which is limited to 4 megabytes. Multiple messages may be "chained" together, but this limitation makes moving large quantities of data difficult and slow.

MSMQ CONNECTORS

Microsoft MQ has "connector" functionality that allows 3rd-party "bridges" to connect MSMQ to other messaging systems. Microsoft has licensed FalconMQ from Level 8 Systems to provide MSMQ to IBM MQ Series integration. A free version of FalconMQ is bundled with Microsoft Back Office on the SNA Server CD; Level 8 also sells FalconMQ on non-NT platforms to integrate MSMQ with other MQSeries systems.

MSMQ MARKET PENETRATION

Based on purely anecdotal evidence, there do not seem to be many customers using MSMQ in a production environment. There are undoubtedly systems in use, but due to its limited interoperability and server platform support, this product will most likely continue to have low market penetration.

BEA SYSTEMS MESSAGEQ

San Jose, California-based BEA Systems (NASDAQ: BEAS), incorporated in 1995, is best known for its "TUXEDO" product. TUXEDO is a transaction processor (TP) that dates back to the Bell companies; BEA obtained worldwide distribution rights to it in 1996. One of BEA's latest products is their entry into the messaging arena with MessageQ.

BEA has an impressive list of customers including Federal Express, amazon.com, United Airlines, E*Trade, DirecTV and Quest. Like many software vendors, BEA has recently released a series of products designed for the e-commerce market as well as a messaging offering known as "MessageQ".

In 1999, BEA reported gross sales of $464 million, up from $289 million in the previous year, of which 44% was derived from services. BEA has over 2,100 employees worldwide. Although BEA has been posting per-share losses for several years due to high investment in product development and strategic acquisitions, they are financially stable with approximately $779 million in cash reserves against $564 million in long-term debt.

BEA's core business has been providing infrastructure for high-volume transaction systems, such as telecommunications billing applications, commercial bank ATM networks and account management systems, credit card billing systems and securities trading account management systems. With the growth of internet-based systems, BEA has been extending its product line to accommodate these new areas.

The features and benefits of MessageQ are as follows:

- Scalability up to thousands of messages per second
- Synchronous as well as asynchronous message delivery
- Recoverable messaging on all BEA MessageQ clients and servers
- Platform support: UNIX, Windows, Win NT, OpenVMS, IBM, and Unisys mainframes
- Easy — Common API, FML support
- Interoperability using TCP/IP and LU6.2
- Publish/subscribe mechanism
- Self-describing messages allow for conversion between heterogeneous platforms
- Messages may be up to 4MB in size
- Connectivity to BEA applications as well as MQSeries and other applications including SAP

Market share data for BEA's MessageQ product were not available at the time of this writing, but it appears that BEA most likely sells MessageQ to its existing customer base as an add-on product, and its appeal to non-BEA customers is likely to be very limited.

SECTION 5. ADD-ON MESSAGING PRODUCTS

Since the market space for messaging systems has matured, there is not much room for new messaging architectures to take hold. As a result, the main area of growth in the market has been in value-added products that work with the existing messaging vendors' systems.

One of the problems with the early messaging systems was that they required a significant amount of coding in order to accomplish virtually any task. The messaging system provided all of the fundamentals (guaranteed delivery, simple API, etc.), but fell short by not providing any sort of database access or other mechanisms for manipulating messages as they passed through the server.

MESSAGE BROKERS

Message Brokers provide this functionality, making development of message-based systems much easier by reducing the amount of coding required. Message brokers are referred to by a variety of names, including "integrators", "event brokers", and many others.

The main concept behind a message broker is this: when a message arrives in a given queue, the broker examines the message and may perform a variety of functions depending on the contents of the message. For example, if the message were to contain a SQL query, the broker could take the contents of the message and send the request to a database server using ODBC. The result of the query could then be placed in a "reply" message and sent back to the sender, all with a minimal amount of coding. Event brokers also may have interfaces to other messaging systems—Sybase Event Broker, for example, can participate in a variety of messaging architectures including IBM MQSeries and Tibco.

Some message brokers include:

- Sybase Enterprise Event Broker (SEEB)
- IBM MQSeries Integrator
- TIBCO TIB/Message Broker

TRANSACTION SERVERS

Transaction Servers allow transactional applications to be written easily and with a minimal amount of transaction-specific coding. For example, an application may need to update several databases at once, performing a "multi-phase" commit across all of them at once. Without a transaction server, this is a very difficult task to accomplish, as the application must perform all of the commit and rollback logic itself. A transaction server provides a set of common transaction services to applications, greatly simplifying the task of developing and maintaining these applications.

The first transactions were mainframe-based. These included CICS (Customer Information and Control System) and IMS (Information Management System). Over the

past few years, LAN-based transaction servers have begun to be deployed. These include ports of CICS to non-mainframe operating systems (like Window NT and various flavors of UNIX) as well as new products like Microsoft's Transaction Server (MTS).

Many of the messaging architectures interface with transaction monitors, which allow transactions to be sent from one monitor to another using messaging technology. A good example of this is IBM CICS.

MESSAGING BRIDGES

As mentioned previously, one of the drawbacks to messaging systems is their lack of connectivity to other systems. To solve the problem, various vendors have developed messaging "bridge" technology that provides this connectivity. One example is MQSoftware's SQL Bridge for MQSeries. SQL Bridge connects MQSeries to any ODBC-compliant relational database for seamless SQL integration, thus reducing application development time significantly.

SECTION 6. CONCLUSION

Messaging systems, when implemented properly, can be an amazingly powerful tool in the arsenal of an applications architect. Candidate applications for messaging technology should include at least some of the following characteristics:

- Relatively time-insensitive (asynchronous transactional requirements)
- Disparate operating systems, network protocols, applications, and/or data sources
- Complex application requirements (many data sources, workflow requirements, rarely connected users, etc.)

As with any technology, it is critical that messaging be considered and implemented in applications that can really leverage the advantages of the architecture. When used incorrectly, applications using messaging can be a disaster. There are tradeoffs to be dealt with in exchange for the power and flexibility that messaging offers. These tradeoffs are usually seen in the areas of performance and scalability, coupled with a higher investment to acquire the software and develop applications, and higher maintenance.

Take your time and really analyze the pros and cons of each application that is a candidate for messaging. Take into account the application requirements and contrast those with the benefits of all options open to you. If messaging seems to fit the bill, then the application will benefit greatly from messaging. If it doesn't, there are plenty of alternative technologies to consider.

Polarsoft Limited specializes in continually available business solutions utilizing a variety of middleware products, including messaging. You may contact us at (877) 440-7778 or via our Web page at http://www.polarsoft.com.

Good luck!

SECTION 7. APPENDICES

APPENDIX I: GLOSSARY AND ACRONYMS

This section will help you to understand some of the terms and acronyms that are used in this document.

Asynchronous Messaging is the concept in which the sender and receiver are not directly linked to one another. All communications take place with a messaging server, which ensures that messages are delivered to the proper recipient.

CICS (Customer Information Control System) is a TP monitor from IBM that was originally developed to provide transaction processing for IBM mainframes. It controls the interaction between applications and users and allows programmers to develop screen displays without detailed knowledge of the terminals being used. CICS is also available on non-mainframe platforms including the RS/6000, AS/400, NT, and OS/2 -based PCs.

CORBA (Common Object Request Broker Architecture) is an architecture that enables pieces of programs, called objects, to communicate with one another regardless of what programming language they were written in or what operating system they're running on. CORBA was developed by an industry consortium known as the Object Management Group (OMG).

ERP (Enterprise Resource Planning) is a business management system that integrates all facets of the business, including planning, manufacturing, sales, and marketing. As the ERP methodology has become more popular, software applications have emerged to help business managers implement ERP.

FML (Field Manipulation Language) a widely accepted full-featured self-describing message format, which allows applications to easily pass data from one to another. FML is a pre-cursor to XML.

HTML (HyperText Markup Language) is the language that encodes the content and presentation of a Web page, allowing them to appear the same on all Web browsers regardless of platform.

MRP (Material Resource Planning) is a system for effectively managing material requirements in a manufacturing process. MRP is a set of techniques that uses bills of material data, inventory data, and a master production schedule to calculate requirements for materials. It makes recommendations to release replenishment orders for materials.

ODBC (Open DataBase Connectivity) is an API that allows applications to easily connect to a variety of data sources regardless of vendor and platform.

ORB (Object Request Broker) is a component in the CORBA programming model that acts as the middleware between clients and servers. In the CORBA model, a client can request a service without knowing anything about what servers are attached to the network. The various ORBs receive the requests, forward them to the appropriate servers, and then hand the results back to the client.

PDA (Personal Digital Assistant) is any one of the small "personal"-sized computing devices including the Palm Computing Platform and Pocket PCs running Microsoft Windows CE.

Publish/Subscribe is a three-tier architecture in which "subscribers" are matched with "publishers" via a middleware broker. An example would be where an application that is interested in receiving information from a data source "subscribes" to the data; when updates become available, the server automatically sends the updated information to the subscriber. This is also known as "push" technology.

T1 Line is a dedicated phone connection supporting data rates of 1.544 Megabits per second (Mbps). A T-1 line actually consists of 24 individual channels, each of which supports 64Kbits per second. Each 64Kbit/second channel can be configured to carry voice or data traffic. Most telephone companies allow you to buy just some of these individual channels, known as fractional T-1 access. T-1 lines are sometimes referred to as DS1 lines.

T3 Line is a dedicated phone connection supporting data rates of about 43 Megabits per second (Mbps), or 28 times the speed of a T1 line. A T-3 line actually consists of 672 individual channels, each of which supports 64 Kbps. T-3 lines are used mainly by Internet Service Providers (ISPs) connecting to the Internet backbone and for the backbone itself. T-3 lines are sometimes referred to as DS3 lines.

VPN (Virtual Private Network) is a private network constructed using a public network (the Internet) to connect nodes. These systems use encryption and other security mechanisms to ensure that only authorized users can access the network and that the data cannot be intercepted.

XML (eXtensible Markup Language) is an extension of the HTML specification, which provides a standardized (vendor neutral) mechanism for applications to pass self-describing data to one another.

APPENDIX II: WEB RESOURCES

BEA Systems Home Page

http://www.beasys.com/

BEA Systems MessageQ Product Home Page

http://www.beasys.com/products/messageq/index.html

IBM MQSeries Home Page

http://www-4.ibm.com/software/ts/mqseries/

MQSoftware (independent software vendor of MQSeries products and services)

http://www.mqsoftware.com/

MQSoftware SQL Bridge for MQSeries:

http://www.mqsoftware.com/products/prodsum/sqlbridge.html

Polarsoft Limited

http://www.polarsoft.com/

TIBCO (The Information Bus Company)

http://www.tibco.com

SECTION 8. INDEX