Enterprise Middleware Series

A White Paper

Reliable Queuing Using BEA TUXEDO®

May 1996

BEA

Enterprise Middleware Solutions

Preview

This paper describes the benefits of reliable queuing as a communication paradigm for building distributed business applications. As an alternative to on-line request/response communication, reliable queuing allows business applications to communicate through the use of stable-storage queues in an asynchronous or time-independent manner. BEA TUXEDO offers such a reliable queuing mechanism. Its main features and benefits are presented here.

Contents

1. Introduction: On-Line Versus Queuing

When a company sends a customer a bill via mail, the company does not require the person to be at home when the bill is put in the mail. Neither does the postal service require that the person be at home when the bill is delivered. Likewise, when the person who receives the bill sends a payment, he does not require that the company actually be open for business on the day the payment is sent. On the other hand, prior to the advent of answering machines, telephone conversations demanded that the person called (the "callee") answer the phone before anything meaningful could happen. Answering machines changed the requirement that the callee be available. Instead one could leave a message, which was heard later. Sometimes, leaving a message is unsatisfactory, because the caller requires information immediately from the callee. The important point is that sometimes communication demands that the communicating parties both be simultaneously available at the time communication takes place. Other communications may take place without the simultaneous availability of the two parties.

In the case when both parties must be available, we say that the communication is synchronous, immediate or on-line. When the parties communicate by leaving messages, we say that the communication is asynchronous. Asynchronous communication requires that messages be left somewhere. In the case of mail, the message is left in your mail box. In the case of an answering machine, the message is left on the recording medium of the machine.

In computer systems, software modules may also demand that a partner be available, or perhaps may communicate via messages. Typically, a partner must be available when the caller needs some immediate information (e.g., a customer record), or requires that some immediate action be taken (e.g., printing out a bill to give to a waiting customer). When modules communicate via messages, the place the messages are held while awaiting "pick up" are called message queues. These queues are storage areas, either in main memory or on disk. Main memory queues eliminate the overhead of disk writing, but have the potential for loss should the computer stop operation before the intended target retrieves from them. Disk-based queues have the advantage that messages can be stored reliably while machines or networks are down. The price for this reliability is the extra cost of writing messages to, and reading them from, disk.

1.1 How Queuing Can Help in a Computerized System

There are many places in a business process where queuing messages is useful. Business processes that use queuing do so because their tasks are accomplished asynchronously (i.e., without knowledge of whether "downstream" systems are available, or when they might be ready to process messages).

Throughout this paper, we will use the example of a mail order firm, named Acme Clothing, or just AC, to illustrate the use of queuing. This section provides some examples.

1.1.1 Work Flow

Many business tasks consist of a series of steps to perform a job. Each step takes some input, modifies the business (e.g., adds a customer's name to a database), and generates some outputs for subsequent processes. In a simple case, this is what e-mail systems do. An initial message is generated and sent to a worker. The worker reads it, takes some action, and either replies or passes it on to a co-worker. The process is repeated until everything connected with the initial stimulus is completed. Another example, on a larger scale, replaces individual workers by whole computer systems. That is, work comes into a system and is put on a work queue. It is processed by software in that system and subsequently put onto the queue of the next system. The process repeats until all of the business functions associated with the initial input are completed.

For example, when a new customer orders merchandise from AC by phone, the AC customer representative uses the AC Customer Interface System (CIS) to create an entry for the customer and take the customer's order. Once the order is taken, the following actions are performed by the CIS software:

- notification of SHIPSYS, the AC shipping system,to send the merchandize to the customer,
- notification of CATALOGSYS, AC's in-house catalog system, of the customer's identity, so a current catalog is mailed,
- send a request for payment to CREDITCO, the customer's credit card company, and
- send a notice to GIFTCO, the company AC has contracted with to send a new customer a "Welcome Customer Gift".

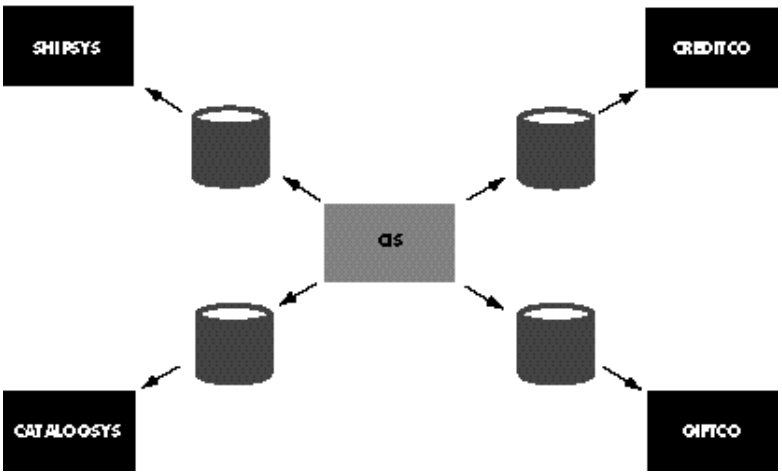

Figure 1. Work Flow at Acme Clothing

Figure 1 shows the relationship of these systems, with work flowing from CIS to them. Note that it may not be possible to complete a number of these work items while, or immediately after, the order is taken by CIS. For example, whereas AC is open 24 hours a day, the GIFTCO's system may only be operational from 8am to 5pm. Thus, Figure 1 shows that the work destined for these other systems is sent to stable-storage message queues for later processing by the receiving systems.

1.1.2 Unavailable Systems

Many times information needs to flow to a system and the system is not available. It may not be possible, cost effective, or necessary to have it available all the time. In this case, queuing can be used to buffer up inputs to the downstream system and submit them when it becomes available. As previously mentioned, this is the case with GIFTCO, whose operational hours are not the same as those of AC. CREDITCO, on the other hand, is supposed to be on-line 24 hours a day. Sometimes, however, the CREDITCO system is brought down for maintenance, which doesn't correspond to the AC maintenance schedule. Because AC likes immediate payment from CREDITCO for customer purchases, requests to CREDITCO for payment are immediately attempted by CIS. If CREDITCO is off-line for maintenance, CIS puts the requests for payment in a queue, and submits them as soon as CREDITCO comes back on-line.

1.1.3 Overloaded Systems

Sometimes the downstream system cannot process inputs as fast as they are generated by the upstream system. Reasons for this include the following:

- the downstream system runs on a less powerful computer,
- the downstream system shares a computer with other tasks, or
- the downstream processing is more complicated than the upstream system.

If the mismatch in processing capability between the upstream and downstream system is temporary, queues can be used as a buffer between the two systems. Assuming that the queues don't overflow, information can move between the systems. Suppose, for example, that during the daytime of peak holiday season, AC customer representatives can create shipping orders faster that the shipping department can satisfy them. During the night hours, customer requests decline, and the shipping department can "catch up". To accommodate this scenario, AC uses queues to buffer the flow of information from CIS to SHIPSYS.

1.1.4 Cheaper Fault-Tolerance

Generally, businesses would like to insure that their systems always complete their work, and are always available. Traditionally, this is achieved by expensive fault-tolerant systems. The expense usually comes from the replication of hardware. Using queuing, it is possible to achieve the desired results by only replicating part of the system. The idea is the following:

- capture the input on a fault-tolerant sub-system, and put the remainder of the system onnon-fault-tolerant hardware,
- if the remainder of the system (on non-fault-tolerant hardware) is available, send the message on,
- if the remainder of the system is unavailable, save the message until it becomes available and then submit it.

The important thing is that the system can always accept work, and gives the promise of eventually processing it. In the case of AC, the company always wants customers to be able to place orders, 24 hours a day. To do this, AC uses fault-tolerant computers to run CIS, while it uses less expensive computers to run SHIPSYS. The issues in deferring processing are a bit tricky. Generally, you want to be guaranteed that the queued work is eventually processed. But you may not want the work processed more than once. In the case of AC, it is willing to defer processing of the shipping request if the computer running SHIPSYS goes down, but it also wants to be sure that any particular order is shipped exactly once, even if the shipping computer fails in the middle of the transfer of the shipping request from the CIS computer to SHIPSYS computer. That is, if the downstream system comes up, starts processing, but fails in the middle, you want to resubmit the message until the down- stream system "gets it right". AC, whose company motto is "We Really Deliver!" wants to make sure that its customers get what they pay for! But AC also wants to ensure that SHIPSYS doesn't process any message more than once. We describe techniques to ensure the processing of a message exactly once later in this paper.

2. Characteristics of a Good Queuing System

Users of queuing systems include:

- Application programmers who write programs that use queues.

- Administrators who configure and monitor queues and applications that access queues.
- End users who use applications that access queues.

Each user has his or her own requirements. The application programmer needs a simple but powerful interface to manipulate items in the queue. At the simplest level, the operations are to put an item into a queue (i.e., enqueue a message) or to get one out (i.e., dequeue a message). We take for granted that the queuing system works in a distributed environment. That is, access to queues may be remote from the machine on which the queue resides. In particular, details of where the queue resides, and whether or not the machine it resides on is of the same kind as the enqueuer or dequeuer should be hidden. The programmer should be able to concentrate on the application using the queuing system, not make up for deficiencies in the queuing system itself. Since the purpose of a queuing system is to store and make available stored messages, this it should do reliably and efficiently.

An important aspect of using a queuing system is when to remove an item from a queue. That is, should an item be removed immediately when a program dequeues it? What happens if just after dequeuing the message, the computer fails? Then the message has been dequeued but never processed. For some types of applications this may be tolerable, but for others it may be a disaster. What one would really like is to have a message removed when the dequeuer is finished processing it. But we also want to be careful not to process a message multiple times.

The natural way to handle messages put in a queue is that the first message put into the queue is the first message taken out (this is called “First-In, First-Out”, or simply “FIFO”). For example, if a queue is being used to hold customer requests for tickets, and the tickets are being sold on a “first-come-first-served” basis, the queue should be accessed in FIFO order. But applications may need other dequeuing orders as well. Perhaps the most recent message put into a queue is the most important one to be read. AC honors customer requests for expedited service. To do this, AC customer reps use CIS to mark certain shipping requests as high priority. SHIPSYS, in turn, accesses the queue it receives from CIS in priority order.

An important user of the queuing system is the system administrator. This person needs the tools to be able to create, monitor, re-prioritize, extend, and otherwise oversee the queues. So, easy-to-use administrative tools, including administrative program interfaces (for customization) are needed.

End users of computers systems that use queuing require systems that efficiently accept and process their requests, guarantee delivery of messages, provide assurance that messages are processed exactly once, and work in a mixed-vendor environment. Because the AC company uses such a system, its customer representatives can offer superior service (“expedited processing”) and never have to explain why an order was not delivered or delivered multiple times. AC customer reps are always doing new business for AC!

3. Features of BEA TUXEDO Queuing

3.1 Queuing as an Extension of BEA TUXEDO

A major advantage of BEA TUXEDO is that it provides several simple yet powerful communication tools[1], one of which is its reliable queuing feature known as /Q. These tools make writing distributed applications simpler by hiding hardware and networking details from application programmers. Thus, programmers can concentrate on writing their application rather than worrying about how programs communicate with each other.

One of BEA TUXEDO's features that makes it easier to write distributed programs is "typed buffers". Typed buffers are used by programs to send data among themselves as well as to and from BEA TUXEDO queues. A typed buffer is a data structure that not only contains application data but also information about that data. This information allows BEA TUXEDO to perform data format conversions transparently as data moves to programs or queues residing on systems having different data representation formats. The advantage of typed buffers is that application data can be sent "as is".

Programs using BEA TUXEDO /Q place their application data in typed buffers which then become the messages stored reliably in disk-based queues. When a program retrieves a message from a queue, it receives the message in the form of a typed buffer.

3.2 Basic Queuing

For the application programmer, the API of /Q is both simple and powerful, consisting of two functions:

- tpenqueue - which puts a message in a queue, and
- tpdequeue - which takes a message from a queue.

Queues are addressed by simple string names, like "PAYMENT_QUEUE", and the name of the queue is one of the parameters of both tpenqueue and tpdequeue. But how does the queue get there in the first place? Queues are setup by the BEA TUXEDO administrator who defines the queues and how much space each is to contain. Moreover, queues are gathered into a collection called a queue space. The name of the queue space is also a parameter of tpenqueue and tpdequeue. To send a message one calls tpenqueue and supplies:

- the name of the queue space,
- the name of the queue,
- the message (i.e., the typed buffer) to be enqueued and its length,
- parameters controlling the delivery of the message, and
- flags controlling the operation of tpenqueue.

as in:

```
tpenqueue(queuespace, queuename, control_info, message, len, flags);
```

For example, one might call:

```
message = tpalloc("STRING", NULL, 500);  /* allocate a typed buffer */
strcpy(message, payment_record);         /* put data into typed buffer */
tpenqueue("CIS", "PAYMENT_QUEUE", control_info, message, len, flags);
```

to send a payment record message to the PAYMENT_QUEUE. Likewise, to dequeue a message one calls tpdequeue, providing:

- the name of the queue space,
- the name of the queue,
- options to control the dequeuing,
- a typed buffer into which to receive the message and its length, and
- flags controlling the operation of tpdequeue.

as in:

```
tpdequeue(queuespace, queuename, control_info, message, len, flags);
```

For example, to read the message enqueued above, CIS could run a program that calls:

```
/* get a typed buffer for receiving dequeued message into */

message = tpalloc("STRING", NULL, 500);

tpdequeue("CIS", "PAYMENT_QUEUE", control_info, &message, &len, flags);
```

Note that this API is quite powerful and simple. Nowhere does the programmer specify the location of the queue, or have to provide any extra code to convert the message to different machine representations. BEA TUXEDO performs all these functions transparently.

3.3 Advanced Queuing: Control Information

Every message enqueued has some control information associated with it. Most of this information is set by the application (e.g., the name of the reply queue), but some of the information is set by the BEA TUXEDO queuing software directly (e.g., the unique message identifier assigned to each enqueued message).

3.3.1 Time Release

When enqueuing a message, it is possible for the enqueuer to specify that the message not be dequeued until after some time. This time may either be absolute (e.g., 2am on January 7th), or relative to the time of enqueuing (e.g., five hours from now). Programs attempting to dequeue messages whose "time has not yet come" don't see them, even though the messages are in the queue.

This feature may be used, for example, by AC when it wishes to distribute price lists in advance of the time the prices become effective. CIS simply enqueues the prices in advance and specifies the dequeuing time as the absolute time the prices are to become effective. Thus, a dequeuing program whose purpose is to update prices as issued by AC, won't see any price updates until the effective time, whereupon the price messages will become available, and the program will dequeue them and post the new prices. The way to specify time delay is to manipulate the control structure, which is the third parameter to tpenqueue and tpdequeue. For example, to specify that a message is not to be dequeued for one hour (3600 seconds) after it is enqueued, one would code:

```
control_info->deq_time = 3600;          /* no dequeues until 3600 seconds from now */
control_info->flag = TPQTIME_REL;  /* indicates that a relative time value is set */
```

prior to calling tpenqueue.

3.3.2 Priorities

Messages can be assigned priorities. The priority given to a message determines where it is placed on the queue relative to other messages. Messages with higher priorities are placed closer to the head of the queue so that they are dequeued before messages with lower priorities.

Usually mail order companies like AC offer their customers the choice of regular or express shipping where the customer pays a premium for overnight delivery. When CIS enqueues orders to SHIPSYS, it marks those items for immediate delivery with a higher priority than those being sent via regular shipping. By doing so, SHIPSYS ensures that it dequeues those items that must be sent out immediately before retrieving any other orders. On enqueuing a message, /Q's control structure allows a message to be given a priority between 1 and 100, inclusive, where 100 is the highest priority. Thus, for CIS to enqueue an order that is to be delivered overnight, one would code:

```
control_info->priority = 100; /* set the enqueuing priority to the highest level */
```

control_info->flag = TPQPRIORITY; /* indicates that a priority value is set */

prior to calling tpenqueue.

3.3.3 Reply and Failure Queues

Before enqueuing a message, a program can name two queues, a reply queue and a failure queue, that should be used by downstream programs that encounter the message. When a message is dequeued, these two queue names indicate where reply or error messages generated as a result of processing the original message should be enqueued.

For example, when CIS enqueues a payment request to CREDITCO, it can name a reply queue where a positive acknowledgement of payment should be enqueued. In addition, it can name a failure queue that CREDITCO can use if it encounters an error processing the payment request. In the event that an AC customer representative incorrectly types a credit card number that CREDITCO cannot process, CREDITCO could enqueue a "verify card number" message on CIS' failure queue.

3.3.4 Correlation Identifiers

Applications can assign "tags" to messages such that they can be identified as they move from queue to queue and processed by different parts of the application. These tags are called "correlation identifiers". One use of correlation identifiers is to correlate request messages with replies. That is, a message can be assigned a correlation identifier before it is enqueued on the application's request queue. The program dequeuing the request message will process the message and attach that message's correlation identifier to the reply message that it generates. It then places the reply message on the application's reply queue. By doing so, the reply message is identified as being a reply for the request message that had the same correlation identifier.

One way CIS could use correlation identifiers is to track orders as they "move" through the system. That is, when a customer places an order, a unique number is generated that identifies the order. When SHIPSYS actually sends out an order's merchandise, it enqueues a message to CIS. This message contains as its correlation identifier the order number as well as other information about how the merchandise was sent. CIS periodically runs batch jobs that read messages from this queue and uses the correlation identifier to locate each order's record and updates its status. By doing so, AC customer representatives have up-to-date information on every order. A customer can be told exactly when an order was shipped, which courier was used, and when it should arrive at its destination.

3.4 Advanced Queuing: Queue Ordering

By default, messages are enqueued in the order specified by the administrator when a queue is created and dequeued in FIFO order. Recognizing that sometimes messages are not intended for immediate processing, and that orderings other than FIFO are needed, BEA TUXEDO provides a rich set of facilities for control of this type. These include the following capabilities:

- the ability to put messages into the queue in priority order,
- the ability to put messages into the queue in time order, and
- the ability to put messages into a queue in either FIFO or LIFO (i.e., "Last-in, First-out") orders.

The administrator can use combinations of the above orderings such that there can be up to three ordering criteria per queue. For example, a queue could be defined to have time as its primary, priority as its secondary, and FIFO as its final ordering criterion. Such a queue would be arranged

first in time order, with messages having the same time release arranged in priority order, with messages having the same time and the same priority sorted in FIFO order.

3.4.1 Ordering Overrides

Even though a queue has been defined to have a particular queue ordering, sometimes an application needs to override this ordering. The BEA TUXEDO queuing facility lets administrators configure queues such that programs enqueuing messages can override the assigned ordering. The two out-of-order features are:

- The ability to place a message at the head of a queue, thus "jumping ahead" of all other enqueued messages. In the case where a queue was configured to support only FIFO ordering, this override option allows for the expedited handling of messages placed at the head of the queue.
- The ability to put a message in a queue ahead of a specified message.

Queue administrators have the option of deciding that enforcing queue ordering is important to the fairness of the application and not allow for ordering overrides. For example, a brokerage application that enqueues market orders might configure its queues to have strict FIFO ordering such that customers' orders are sent to the market only in the order they arrive.

3.5 The Effect of Transactions

A transaction is a method to make sure that a number of operations are either all performed, or none of them are performed. Transactions are usually used to ensure that multiple updates to databases are all effected. Such updates may be within a single database system, or may be spread across multiple database systems. For example, if money is to be transferred from a savings to a checking account, two records may have to be updated: the savings account balance in the savings account record should be decreased by the amount being transferred and the checking account balance in the checking account record should be increased by the same amount. The desired effect is that both balances be updated. It is highly undesirable for one balance to be updated but not the other.

Suppose neither the checking nor savings accounts are on-line at the time someone wants to make a transfer. The system could still enqueue messages to both systems. But it would like to ensure that both messages are enqueued. To do this, the system must ensure that the systems holding the messages are coordinated. This BEA TUXEDO does. In fact, it can coordinate the update of a queue with the update of any other queue and the update of any database system. What happens if one of the messages can be put in its target queue but the other message can't be enqueued? In this case, the programmer calls the rollback function, and no messages are enqueued whatsoever. That is, a message is not really enqueued until the "commit point" of the transaction it is in is completed by a successful call to the commit operation.

Remember the problem we posed before about ensuring that an enqueued message is processed exactly once? The transactional properties of message queues help solve it. When a message is dequeued by a program using tpdequeue, the message is only removed from the queue if the dequeuer's transaction commits. If the transaction is rolled-back, or times-out, the message is left in the queue. Thus it will be processed again. Because the message is left in the queue as a result of the non-commitment of its enclosing transaction, all of the databases updated as a result of that transaction are also undone. That is, when the transaction is rolled back, not only is the message left in the queue, but the effects of processing it are undone. It is as if the message were never dequeued in the first place. So when it

is subsequently re-dequeued, it can be processed without fear of it already having been processed. In effect the system guarantees that it is processed exactly once!

4. Queuing Meets On-Line

Of course, it is possible for applications to use both synchronous and asynchronous communications. For example, an application may accept data from an input device, update some database, and enqueue work for a downstream system. It processes some of the data "now" and schedules some for later. Moreover the system can conditionally do either. For example, upon accepting work, the system may try to invoke parts of the application immediately (i.e., on-line). If those parts of the application are unavailable, it may be possible to queue the request for them. Thus, if the system can process the request on-line, it does so. If parts of the application can't be accessed, it queues data for them.

A BEA TUXEDO application written to try on-line processing before failing over to a reliable, stable-storage queue to capture requests might look something like:

```
request->acct = acct_no;   /* Populate request buffer with account...*/
request->amt = amount;     /* ...and amount information for payment request */
/* Send the request to the on-line service named "RequestPayment" */
if (tpcall("RequestPayment", request, req_len, &reply, &reply_len, 0) == -1) {
/* The call failed; check error codes */
switch(tperrno) {case TPENOENT: /* The service is not available; fall-through to next case... */
case TPETIME:   /* The service timed-out; probable cause: a network failure */
tpenqueue("CIS", "PAYMENT_QUEUE", qctl, request, req_len, 0);
break; ...}}
```

5. Conclusion

Asynchronous communication via queuing systems is a powerful tool to model real-world work flow and to overcome a variety of situations where it is not possible for cooperating systems to be immediately available for each other. BEA TUXEDO provides a feature-rich queuing system that can be used in an integrated fashion with its other distributed features to provide superior on-line and queuing software to build robust synchronous and asynchronous business applications.

References

1 BEA Systems, Inc., "Programming a Distributed Application: The BEA TUXEDO Approach", BEA TUXEDO White Paper.

BEA Systems, Inc. (BEA), 385 Moffett Park Drive, Suite 105, Sunnyvale, CA 94089-1208, USA.