Oracle8*i*™ Advanced Queuing

An Oracle Technical White Paper
February 1999

ORACLE

INTRODUCTION

Many of today's enterprises have a large number of applications which perform well-defined tasks including accounting, manufacturing, inventory control, and shipping. There is a growing need for application integration allowing corporations to participate in global *business level* transactions,(i.e., order-fulfillment). Corporations are coming to the realization that applications cannot afford to be isolated islands of automation. The business drivers for application integration include corporate mergers and acquisitions, deployment of best-of-breed packaged applications from multiple vendors, and supply-chain management. Electronic commerce is driving companies to integrate their applications with those of their partners, suppliers, and distributors.

Integrating applications is an extremely complex and challenging task exacerbated by the fact that enterprise applications are geographically widely distributed, autonomously developed, and heterogeneous. Message queuing is the most viable technology for integrating these applications. Message queuing provides three essential features for the integration of such applications; an asynchronous communication model, reliable communication in the face of network failures, and a loosely coupled transaction model.

DESIGN OBJECTIVE OF ADVANCED QUEUING

This paper discusses Oracle8*i* Advanced Queuing — the industry's first database-integrated message queuing product. As a result of integrating asynchronous message queuing into the database itself, Advanced Queuing has brought unprecedented levels of reliability, availability, scalability, and operational simplicity to the world of message queuing. The design objective of Advanced Queuing is to go beyond the guaranteed delivery functionality of typical message oriented middleware and to provide a comprehensive solution for message management — one of the harder problems in message queuing based application integration. Message management involves tracking messages through their entire life-cycle, determining the state and location of messages at any given time, tracking the relationship between messages, and exception management.

APPLICATION INTEGRATION — A CASE FOR MESSAGE QUEUING

Enterprises have been using a variety of ad-hoc methods to integrate their applications including batch-processing, load and extract programs, and data synchronization using batch file transfer. These methods are all useful in some limited circumstances, but they do not provide a formal infrastructure for enterprise application integration. Message queuing is the most suitable technology for this purpose. To understand the reason, we will first investigate the characteristics of enterprise applications, then derive the technical requirements, and finally demonstrate how message queuing best addresses these requirements.

CHARACTERISTICS OF ENTERPRISE APPLICATIONS

In the context of integration, we will discuss the following three prominent characteristics of enterprise applications.

1. *Distributed* — Enterprise applications tend to be widely distributed across local and wide area networks.

2. *Autonomous* — Today's IT departments are highly decentralized. Autonomous lines of business own the applications and make purchasing decisions. This autonomy of ownership implies that there is no central authority that can mandate compliance to an enterprise-wide standard. This is especially true for inter-enterprise applications. Therefore, applications changed independent of each other and internal changes are not reflected in corresponding changes to others. Autonomous applications follow their own schedule for upgrades, maintenance, downtimes, and backups.

3. *Heterogeneous* — Extreme heterogeneity is the hallmark of today's enterprise applications. The autonomous factor described above is a major contributor to this state of affairs. Enterprise applications have heterogeneous internal architectures (host base single-tier, two-tier, three-tier),security models (roles, privileges, authorization, and access control lists), and data models. Real or de-facto standards cannot remedy the heterogeneous nature of applications. Data and semantic models of autonomous applications will always be different.

REQUIREMENTS — LOOSE COUPLING

When discussing the integration of applications, we have to consider three dimensions of the interaction between applications: *transaction model*, *communication model*, and *interface model*. Considering the three characteristics of applications listed above, it is safe to say that only a *loose coupling* is practical for each of the dimensions.

Tight coupling is simply not feasible for the following reasons:

- *Transactional Coupling* — Using the two-phase commit protocol to keep two or more databases in sync is an example of tight transactional coupling. The drawback with this approach is that if one of the databases fails during the two-phase commit protocol, then parts (or whole) of the other database become unavailable until the recovery of the failed database or databases. Under some extreme conditions both databases could become unavailable for an unacceptably long duration. Therefore, it is impractical to implement global distributed transactions that span multiple applications and last the duration of the business activity.

- *Communication Coupling* — Applications go through a sequence of steps in executing their business logic. One or more of these steps may require sending or receiving information from other applications. An example of tight communication coupling is the use of synchronous communication protocols to send and receive information. Synchronous communication based applications cannot proceed with their work if the application(s) at the other end become unavailable for any reason. Furthermore, the speed and performance of the one application directly affect the others. Thus, tight communication coupling creates an unacceptable inter-dependency between applications.

- *Interface Coupling* — Every application provides some interface by which other applications can interact with them. An example of tight interface coupling is the use of functions as interfaces. Interface Definition Language (IDL) used by remote procedure calls and common object request broker architecture is an example of a functional interface. The basic premise behind functional interfaces is that the signatures of these functions never change — the signature is a binding contract between applications. However, in reality, interfaces often change and all the applications must change correspondingly. Functional interfaces create an unacceptable inter-dependency between applications.

MESSAGE QUEUING — ADDRESSING THE REQUIREMENTS

The previous section described significance of loose coupling between interacting applications, especially when considering transactions, communications, and interfaces. It also highlighted the drawbacks of transaction coordination through two phase commit, synchronous communication, and functional interfaces. This section introduces message queuing and explains how it addresses the requirements listed above.

MESSAGE QUEUING — HOW IT WORKS

Messages are discrete pieces of information (data) exchanged by applications. Messages represent business activities and objects including purchase orders, sales orders, bills, bill of materials, and checks. Typically, messages consist of two parts[1]:

- *Payload* — This is the actual information communicated. This could be unstructured information, structured data (relational tables) or self describing data (Internet standard XML).
- *Control Information* — Information that controls the behavior of the message such as the destination, expiration time, priority, and recipients.

Applications interact with other applications using message queuing as follows. They create a message and hand it off to the message queuing system. The message queuing system *guarantees* the delivery of the message to its destination. If the destination is temporarily unavailable for any reason (machine, network or application failure), the message queuing system *stores* the message persistently and later *forwards* it to the destination.

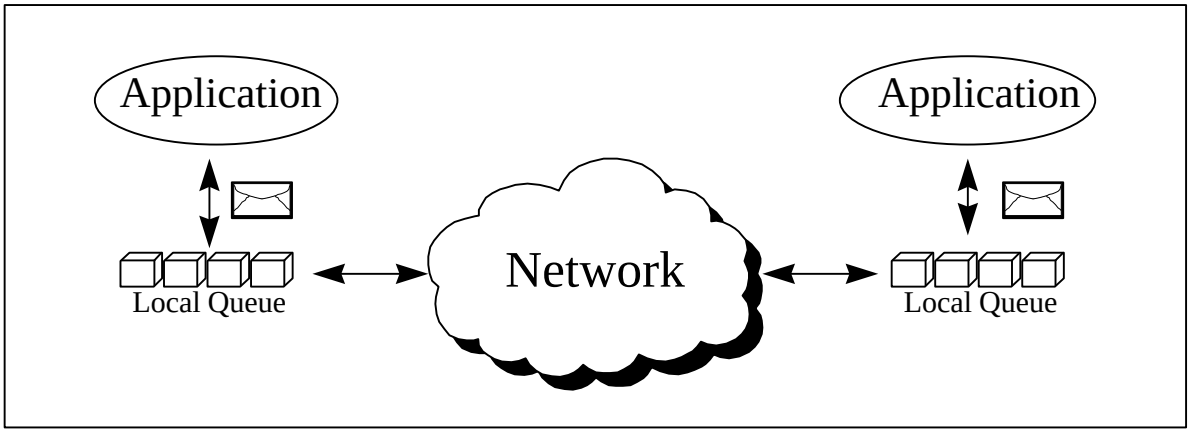

Fig 1. Applications Communicate Using Message Queuing

- *Loose Coupling* — Message queuing enables the integration of applications in a loosely coupled manner. It provides loose coupling of transactions, communication, and interfaces.
- *Transaction Coupling* — Using message queuing, one application sends a message and other applications receive it. These two activities take place in two separate and unrelated transactions.

[1] *As noted later, Oracle Advanced Queuing messages are unique in that they have a third part, a "history" associated with them.*

There is no global transaction monitor to coordinate the two transactions and the outcome of one transaction has no impact on the other. So, each application maintains its transactional autonomy and is completely de-coupled from others.

- *Communication Coupling* — Applications use messages to communicate *asynchronous*ly. After sending a message, an application goes on with its work without waiting for a reply. It does not wait to verify whether the message reached its destination. In fact, depending on the type of application, there may be no reply at all. Unlike in synchronous communication, the applications are de-coupled — the sending and receiving applications need not be running at the same time or at the same speed.

- *Interface Coupling* — In message queuing, the messages themselves are the interfaces between applications, not functions. Every application defines it external interface by specifying the set of messages that it consumes as input and the set of messages it produces as output. Even though the structure of messages is a binding contract between applications, it is more amenable to changes — intermediate brokers *transform* the messages into the native format of the other applications. Message transformation is a mature and proven technology with a number of successful products already on the market. Messages are data and data transformation is comparatively easier to deal with than changes in function signatures.

Message queuing ideally suits the integration of distributed, heterogeneous, and autonomous applications in a loosely coupled manner.

ORACLE ADVANCED QUEUING — IMPLEMENTATION OVERVIEW

This section provides a brief overview of the implementation of Oracle Advanced Queuing. The features are discussed in-depth later.

Oracle Advanced Queuing is implemented as an integral part of the Oracle database management system. Advanced Queues are regular relational database tables which have been enhanced to support queuing operations like enqueue and dequeue as shown in Figure 2. Messages in queues correspond to rows in a table. Advanced Queues can store either unstructured (raw) messages or structured (Oracle Objects™) messages.

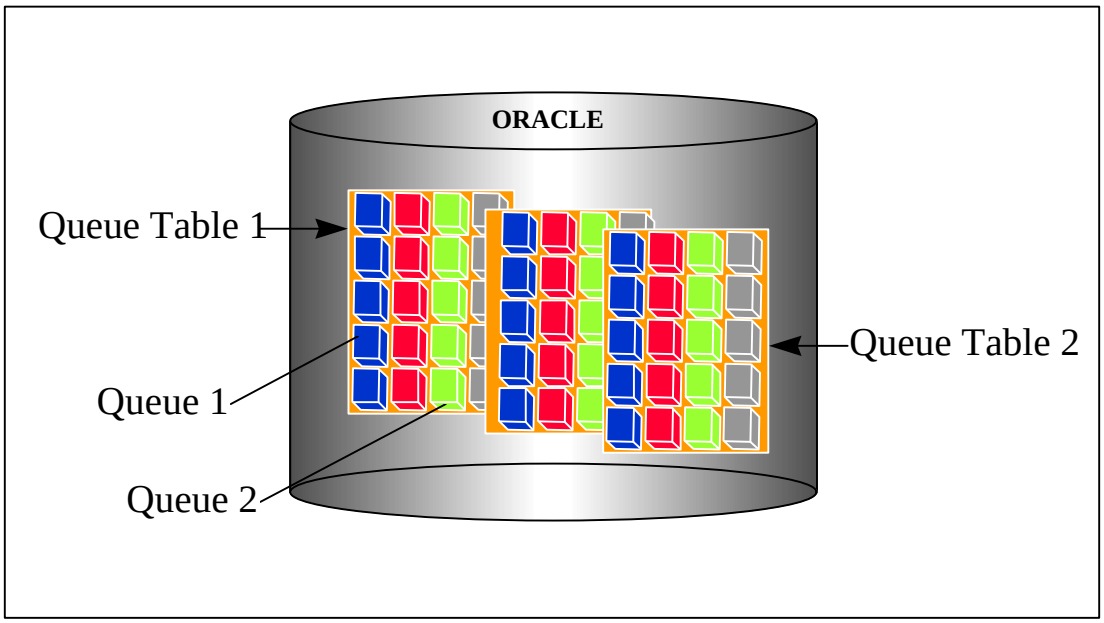

Fig 2. Advanced Queues are Implemented as Regular Database Queue

The three primary operational interfaces to the queues are Enqueue, Dequeue, and Listen. Application use queues by executing the following sequence of steps:

1. Create a queue table and create one or more queues in the queue table
2. Start the queues
3. Enqueue and Dequeue to or from the queue

Oracle Advanced Queuing propagates messages between queues located in different database instances as shown in Figure 3.

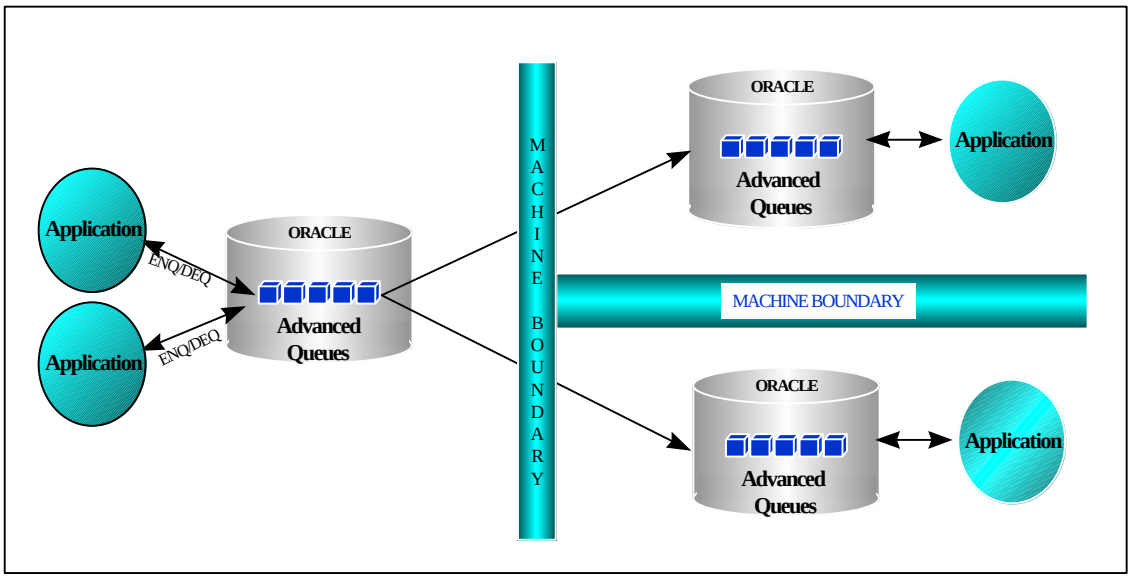

Fig 3. Advanced Queuing Transparently Propagates Messages Over the Network

A dedicated queue monitor process enforces the timing specification for messages.

ORACLE ADVANCED QUEUING — DESIGN OBJECTIVES

Oracle8*i* Advanced Queuing is the industry's only database integrated message queuing solution that not only offers all the functionality of message-oriented middleware products, but also addresses the hard problem of message management. Advanced Queuing has the robustness, scalability, and availability characteristics expected of an enterprise level infrastructure — mainly because it is integrated with, and leverages the database whenever possible.

THE HARD PROBLEMS

The previous section examined the suitability of message queuing as the foundation for integrating applications. However, message queuing introduces some very hard problems which message-oriented middleware products on the market did not address. Oracle Advanced Queuing was specifically designed to solve these hard problems. Any product that is incapable of addressing these issues cannot be the infrastructure for intra/inter enterprise application integration.

This section discusses how advanced queuing addresses two of the hardest problems in message queuing based application integration — *message management* and *operational simplicity*.

Message Management

Advanced Queuing has a number of features specifically designed to address the problem of message management. It offers the following message management functionality.

- *End-to-End Tracking:* — Advanced Queuing tracks messages through their entire life-cycle. At any given time it is possible to determine the precise location and state of a message (waiting, ready, consumed or expired). Every message carries its history with it — all the nodes that it has visited and which of the recipients have actually received it. This functionality can be attributed to the fact that unlike typical message-oriented middleware products, Advanced Queuing queues are not black boxes. Queues are regular relational tables and applications query them using standard SQL. Advanced Queuing owes much of its uniqueness and functionality due to its dual nature — queues are structures that support queuing behavior, but are regular database tables as well.

  Application can optionally retain messages in queues even after they have been consumed or propagated to a remote queue. So every node tracks the enqueue and dequeue time for every message as well as the actual time that applications consumed a message.

- *Relationship Tracking* — As messages move between applications, they split, merge, and clone. Messages create new messages and replies. Advanced Queuing automatically tracks relationships between messages. It tracks the following types of relationships:

  – *Cause-Effect Relationship ¾* Tracking which message caused the creation of other messages. For example, it is useful, to know which bill caused which check to be issued.

  – *Copy-Original Relationship ¾* When a message shows up in a queue, it could have reached there in one of two ways; an application created it for the first time or it was propagated from another queue. In the latter case, Advanced Queuing enables applications to identify the original message and its location. Oracle Advanced Queuing automatically documents the origin of every message.

  – *Request-Reply Relationship ¾* Messages are often created as replies to other messages. Advanced Queuing tracks this relationship between messages.

- *Exception Handling* — Every Advanced Queuing message has a precisely defined lifetime. Messages not consumed by all the prescribed subscribers within the specified lifetime, are automatically moved to an exception queue. Applications can find expired messages in the exception queue to determine the cause. Most importantly, no message is ever lost without trace.

- *Automatic Auditing* — Advanced Queuing automatically audits all messages by allowing applications to optionally retain messages indefinitely in queues. In many circumstances, it is legally required to retain messages for a specified duration in their *original* state. Many applications also log messages for dispute resolution. Advanced Queuing provides automatic logging of messages without any effort on the part of the application.

- *Message Warehousing* — There is one very useful side-effect that arises from retaining messages. Messages, like data, retained and accumulated over time constitute a very valuable warehouse. Enterprises use message warehouses to analyze the efficiency of a company's business processes.

MESSAGE MANAGEMENT — WHY IS IT IMPORTANT?

Message management as outlined above assumes great significance when we consider the distributed and autonomous natures of applications. In these circumstances there is no one central coordinating authority to manage messages, track them, handle the exception cases, and resolve disputes. It becomes the responsibility of every application to manage messages themselves. Advanced Queuing completely relieves the application of the this burden. Since the infrastructure provides the functionality, it manages messages in a consistent and reliable manner.

SCALABILITY, RELIABILITY, AVAILABILITY, AND OPERATIONAL SIMPLICITY

Enterprises integrate applications within an enterprise and between enterprises in order to automate critical business processes. Enterprises depend on these applications to run their essential business activities. It is very important for the infrastructure that integrates and coordinates these applications to be as reliable and available as the applications themselves. *The integration infrastructure cannot afford to be the weak link in the integration chain, especially given its mission-critical purpose.*

- *Leveraging the Database ¾* Oracle Advanced Queuing is an exceptionally robust and scalable message queuing system chiefly because it leverages the database wherever possible. Advanced Queuing is an integral part of the database and so automatically and transparently inherits all the superior operational characteristics of the Oracle8*i* database management system. The Oracle database server is a proven high-performance database management system used successfully in thousands of high-end OLTP and DSS environments. The Oracle database server is the result of over two decades of research and development. Oracle® Parallel Server provides very high degrees of availability. It scales very well and can handle thousands of concurrent users. Oracle is the current leader in both TPC-C and TPC-D benchmarks.

- *Robustness* — Oracle Advanced Queuing does not lose any messages under any circumstances. Even if the system crashes, it can be quickly recovered without losing messages. True to the spirit of autonomy, the failure of one system does not impact other systems.

- *Scalability* — A message queuing infrastructure should accommodate new applications as they are included into the integrated family of applications. It should not run out of steam as message through-put requirements increase. All Oracle Advanced Queuing operations such as enqueue, dequeue, and propagation scale linearly with the number of processors in a symmetric multiple processor (SMP) environment.

- *Operational Simplicity* — Enterprise applications are invariably database applications. By integrating the message queuing infrastructure with the database, Advanced Queuing has been able to dramatically simplify the day-to-day operations of the entire system. Advanced Queuing is transactionally integrated with the database. Transactionally, the database and the message queuing system are treated as a single resource — a single transaction applies to the database as well as the queue. A separate transaction coordinator is not required. The elimination of a separate transaction coordinator simplifies management and recovery (compare Figure 3 and Figure 4). It also improves performance by eliminating the overhead of the two phase commit protocol.

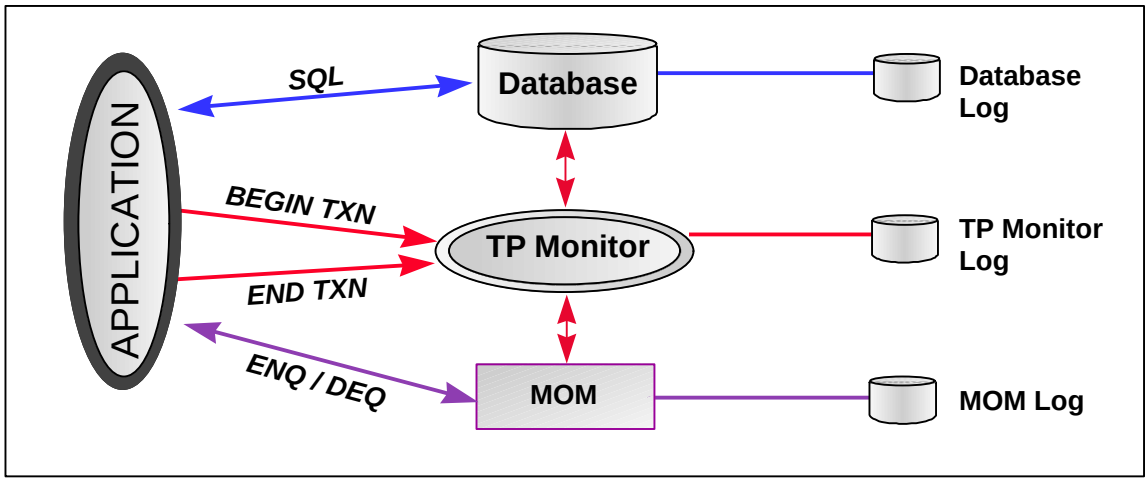

Fig 4. Message-Oriented Middleware Products Are Not Transactionally Integrated with the Database

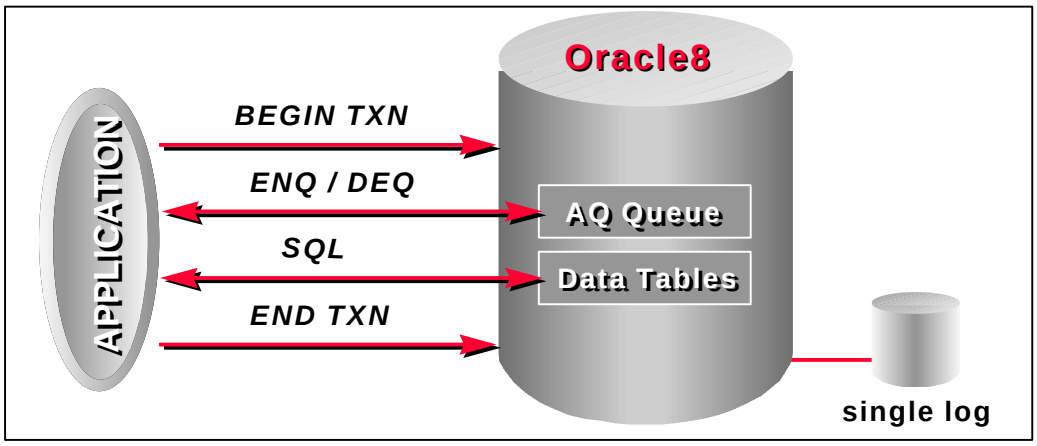

Fig 5. Integrated Message Queuing Simplifies Administration and Point-In-Time Recovery

The *Oracle® Enterprise Manager* completely supports Oracle Advanced Queuing. Through the Enterprise Manager applications you can:

1. Create, drop, stop, and start queues
2. Add and remove subscribers
3. Schedule propagation of messages from local to remote queues
4. Display queue statistics

A single tool, the Enterprise Manager manages the queuing system and the database system providing a single, integrated view of both systems. Therefore, the overall cost of management is reduced.

ORACLE ADVANCED QUEUING — FEATURE OVERVIEW

This section will discuss Advanced Queuing features under two categories: *basic queuing* and *publish and subscribe*.

ADVANCED QUEUING — BASIC FEATURES

- *Blocking Dequeue ¾* Applications retrieve messages from queues through a dequeue function call. The dequeue call blocks if there are no messages in the queue. It returns only when messages show up in the queue. With blocking dequeue, applications do not have to poll for messages, conserving

valuable processing resource and enabling the system to scale. Oracle8*i* enables applications to block on multiple queues with a single call, using the Listen function.

- *Retention ¾* Applications can optionally retain messages in a queue even after they have consumed or propagated to a remote queue. These time(s) at which the message was dequeued or propagated is recorded in the message. Retention is the one feature that enables life-cycle tracking. Do not consider retention merely as a quality of service (qos) issue. It can potentially change application design. Consider the following scenario; an application sends a message and expects a reply in return. Question: When the application gets a reply message, how can it identify the original request message? Two possible approaches are listed below, both unsatisfactory:

  – *Approach One* — All the applications agree on a protocol whereby they attach the original request to every reply. One drawback is that all applications have to agree on this protocol. This violates the autonomy requirement. The other drawback is that if messages are large and hop back and forth a number of times, performance suffers.

  – *Approach Two* — Every application logs every message it sends in a persistent store. The message queuing system records the identifier of the original request message in every reply. When an application gets a reply message, it can use the identifier to locate the original request. The drawback with this solution is that the application has to bear the burden of logging and fetching messages. This seemingly simple exercise would constitute a significant part of the total application development effort.

  Advanced Queuing completely relieves the application of this burden by retaining messages and providing a powerful query language to locate them.

  Do not confuse retention with the logging functionality of typical message-oriented middleware products. Applications retrieve message retained in Advanced Queuing queues very efficiently using standard SQL. Message logs that cannot be queried are not very useful.

- *Multiple Recipients:* - An application can send a message to multiple recipients (applications) with a single enqueue call. Advanced Queuing will ensure that all applications get the message without making multiple copies of the message. This feature ensures that the message queuing system can scale to handle large number of recipients and message sizes. A queue administrator can specify the list of subscribers who can receive messages from the queue. An application can override this default list by specifying a recipient list on a per message basis.

- *Time Specification:* - Every message has a well-specified lifetime. Applications can specify a delay interval and an expiration for every message. The time between the delay and expiration interval is the active lifetime of the message. Messages are ready for consumption after the delay interval and unavailable after the expiration interval. Specifying a delay interval is like post-dating a check. Advanced Queuing honors time specifications even for propagated messages.

- *Exception Handling* — Messages not consumed within their active lifetime are automatically moved to an exception queue. Advanced Queuing creates a default exception queue if the application does not explicitly create one. Applications can check the exception queue to determine why a message expired. With the automatic exception handling Advanced Queuing ensures that no message is ever lost without trace.

- *Poison Message Protection* — Poison messages (badly formed messages) might cause the receiving applications to crash. When applications crash, the transaction is rolled back and the messages end up in the queue again. Applications will find these messages again the next time around and so can never get out of this vicious loop. Advanced Queuing allows applications to specify a maximum *retry count* to handle poison messages. The retry count specifies the maximum number of times an application can attempt to dequeue a message. After the number of attempts exceeds the specified maximum retry count, messages are automatically moved to the exception queue.

- *Resource Separation* — Advanced Queuing is transactionally integrated with database management system. A single transaction applies to the tables and queues — when a transaction rolls back, all changes to tables are undone along with enqueue/dequeue operations on queues. Sometimes it is desirable to treat the queuing system and the database system as separate resource managers. For example, an application might want to send a message irrespective of the outcome of the database transaction. Advanced Queuing applications can use the immediate option to achieve this separation. Enqueue and Dequeue with the immediate option can be considered as independent transactions unto themselves — their effect is independent of the outcome of the outer database transaction.

- *Statistics* — Advanced Queuing gathers statistics about queues unobtrusively and on-demand. The statistics collected include average length of the queue, number of messages in waiting state, number of messages in ready state, and number of expired messages. The Oracle Enterprise Manager displays statistics. Statistics are stored in an in-memory table and applications obtain it using standard SQL. The Advanced Queuing statistics collection is non-invasive — users can control if and when to gather statistics for a queue.

- *Propagation* — Advanced Queuing can transparently move messages between queues distributed across a local area or wide area network. An application can enqueue a message to a local queue and specify the destination of the message to be remote queue. Advanced Queuing will propagate the message to the remote queue even if the machine or network is temporarily unavailable. It will be retried until it succeeds. All time specifications (delay and expiration) are honored. Using Net8, Advanced Queuing can propagate messages across a variety of network protocols such as TCP/IP and IPX/SPX.

  Advanced Queuing propagation uses streaming to overlap network input/output with processing to minimize the impact of low bandwidth network. Administrators fine tune propagation by setting the appropriate values for a number of scheduling parameters. These parameters include the number of processes, frequency of polling, duration of propagation, and waiting interval. Propagation scales linearly with the number of processors in a symmetric multiple-processing environment.

- *Plug-In Architecture* — Oracle Advanced Queuing has an open, extensible architecture for propagation so applications can use other transport mechanisms to distribute the messages. For example, in some situations (trading floors of financial institutions) messages have to be distributed to a very large number of applications distributed across a local area network in near real time. TIBCO and Vitria have products that use IP multi-casting for this purpose. With the plug-in architecture it is easy to use these products to propagate advanced queuing messages. Message-oriented middleware products like IBM's MQSeries® can also be used as the transport mechanism wherever appropriate, like inter-operating with legacy systems like CICS. Applications can develop custom propagators if necessary.

- *Non Persistent Queues* — In many situations the strict requirement for guaranteed delivery of messages can be relaxed — a simple *best effort* is sufficient. Non persistent queues are used to distribute non-critical messages like stock ticks. A lost message is no cause for concern as the new one is around the corner anyway. Non-persistent queues do not store messages in stable storage. They have a higher through-put because messages are kept in memory and the overhead of writing to stable storage is eliminated.

PUBLISH AND SUBSCRIBE FEATURES

Earlier, we emphasized the importance of loose coupling for integrating distributed, autonomous, and heterogeneous applications. Publish and Subscribe is a message based information distribution model providing an even greater degree of loose coupling than simple message queuing. In message queuing, the sending application directs the message towards a set of specific application(s). This establishes a coupling between these applications, however loose. Publish and Subscribe eliminates even this degree of coupling as described below.

- *Publishers* just publish information as messages to an intermediary typically referred to as publish and subscribe engine or message broker. Publishers do not know or care whether other applications are interested in the message or not. The publishers decide when, what, and how to publish. In the client-server model, servers respond (produce information) only when they receive a request from a client. In the publish and subscribe model, no external entity drives the publishers.

- *Subscribers* receive information by expressing interest in certain types of messages. They do not care about the origins of the messages. The publish and subscribe engine (or message broker) delivers published messages to the interested subscribers. Depending on the service provided by the message broker, the subscribers can specify the time, format, location, and manner in which they expect to receive the information.

- *Publishers and Subscribers* do not directly connect to each other in the network sense. They are unaware about each other's location and architecture. Therefore, a publisher or a subscriber can come and go without impacting any other application.

Addressing Modes

Depending on how subscribers specify their interest in messages, we classify the publish and subscribe model as either *subject-based* or *content-based*.

- *Subject-Based Addressing* — In this model, a publisher associates every message with a certain subject when publishing a message. Subscribers subscribe to certain subjects and they receive all messages associated with that subject. Publishers and subscribers only have to agree on the name of the subject. Subject names are usually arranged in a tree-like hierarchy.

- *Content-Based Addressing* — In this model, publishers just publish a message without necessarily associating any subjects to the messages. Subscribers specify certain filtering rules for messages — they get only those messages whose content satisfies the rules. The message broker is responsible for evaluating every message to determine if its content satisfies the selection criterion for any subscriber.

PUBLISH AND SUBSCRIBE FEATURES OF ORACLE ADVANCED QUEUING

- *Content-Based Addressing* — Oracle Advanced Queuing supports content-based addressing. Subscribers express the filtering rules using SQL. Filtering conditions can be applied to the content (payload) of messages as well as to the control information. Advanced Queuing messages are normal database objects; any SQL operation that applies to data objects can also be applied to messages. A high performance rules engine evaluates and selects the messages. Subscribers receive only those messages that satisfy their specified rule. Note: the rules engine evaluates only those messages for which no explicit subscribers are specified.

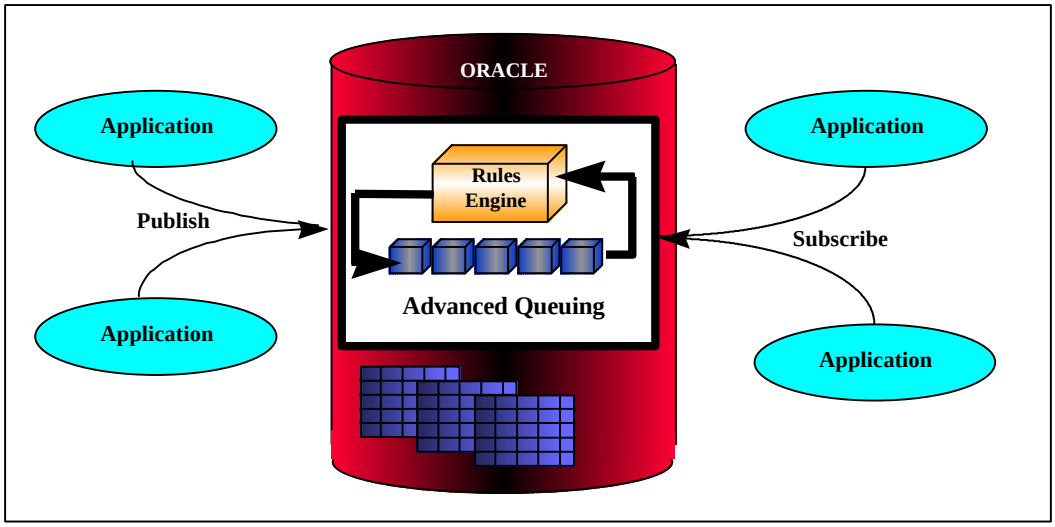

Fig 6. Rules Based Routing of Messages

- *Subject-Based Addressing* — Applications can simply implement subject-based addressing by mapping queue names to subject names.

- *Database Event Publication* — Important business activities occurring in an enterprise result in changes to a database. Therefore, it has been a common integration strategy to tap the database as the single source of events. IT organizations traditionally achieve this goal by techniques such as load/extract, intercepting database calls, and triggers. Each of them is trying to achieve the same goal — utilize the database as a source or *publisher* of events.

  Database event publication formalizes this concept by providing an elegant and powerful interface. Applications can subscribe to database events just as they subscribe to messages from other applications. Database events could be:

  – DML events like insert, delete, and update

  – System events like start up, shutdown, log on, and log off

  The database event publication is tightly integrated with Advanced Queuing. Database events are published by triggers to non-persistent queues. Applications have a single consistent interface allowing them to treat the database as a publisher just like any other application.

- *Asynchronous Notification* — Subscribers can register callback functions when they subscribe to messages. When messages are published, the database immediately invokes the appropriate callback function. This, in essence, is the "push" paradigm of information delivery. The benefits from this feature are:

  – Simpler application development model — applications only have to implement the callback function(s). The database automatically checks and retrieves the messages from the queue(s).

  – It removes the need for polling, thereby improving performance and eliminating latency.

  The applications do not have to be connected to the database to receive notifications. Applications can selectively enable or disable notifications.

ADVANCED QUEUING AVAILABILITY

Advanced Queuing is free with the Oracle8*i* Enterprise Edition.

SUMMARY

Oracle8*i* Advanced Queuing is a comprehensive, full service message queuing platform for integrating applications within and between enterprises. It is a full-featured, robust, and easy to manage enterprise infrastructure for mission-critical use. Oracle8*i* Advanced Queuing is the industry's first database-integrated message queuing product that goes beyond guaranteed delivery to address the hard problems of message management in a distributed, autonomous, and heterogeneous environment. The Internet has a tremendous potential as a cheap, ubiquitous, connectivity platform for connecting consumers to businesses, businesses to businesses, and applications to applications — Oracle8*i* is the first step towards making this potential a reality.

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
+1.650.506.7000
Fax +1.650.506.7200
http://www.oracle.com/