An Oracle White Paper
March 2010

Managing Exceptions in Distributed Applications

ORACLE

Executive Overview

Exceptions are the cause of lost orders, lost revenue, and overall inefficiencies. Business operations teams seldom have the visibility they need to detect these issues or respond to them proactively. As a result, transaction failures and other exceptions lead to not only immediate revenue loss but also customer churn. Distributed applications such as service-oriented architecture (SOA) raise exception management challenges that are beyond the scope of traditional approaches. To meet the demands of today's applications, exception management must provide the ability to detect both software errors and business events in distributed applications, and then act upon these conditions in real time.

Introduction

Distributed applications—such as SOA, cloud, Web 2.0, business process management, and Web services systems—present tremendous new opportunities for organizations to develop modular business applications and achieve rapid integration across heterogeneous platforms. The organizations that deploy these loosely coupled systems use a variety of names for them, including composite applications and compositions.

No matter the name, these distributed, federated applications can deliver immediate benefits, such as simplicity of development and economy of costs. However, the real value of composite applications lies in the unprecedented flexibility that they bring and the resulting real-time enterprise:

- Application functionality can be exposed as new services for reuse by other applications.
- New business applications can be rapidly composed from services and exposed as services themselves.

- Business systems can be integrated from end to end, resulting in a real-time flow of information and a real-time business environment.

Composite applications and real-time environments are inherently vulnerable to exceptions. Exceptions typically materialize as unusual or unacceptable states in a business transaction. The consumers of a service—the customer or the end user of a composite application—are usually the first to experience them. Common examples include opaque messages on an e-commerce Website, delayed orders, lost packages, and rejected insurance claims.

To the enterprise, exceptions appear, at best, as entries in an error log that simply state “Error 00021C: Transaction rejected” and, at worst, as frantic calls from vexed customers. These issues can lead to resource-intensive “fire drills” within the IT operations team. At the same time, disrupted or failed business transactions result in lower customer satisfaction, lost orders, lost revenue, and overall inefficiencies. Business operations teams seldom have the visibility they need to respond to these challenges proactively.

Because exceptions impact customers up front, management of exceptions is of utmost significance to the business. To be successful with distributed applications, IT operations must be able to detect, diagnose, and resolve the exception quickly and efficiently—in minutes instead of hours or days. Business operations must be able to learn about important exceptions in real time with the context necessary to resolve them before a customer logs a complaint.

Fortunately, it is possible to ensure functionally robust distributed business environments. Exceptions in live business transactions are preventable by employing a management system that

- Detects and intercepts system errors, application errors, or business events at runtime
- Notifies authorized personnel (in IT and business operations) as well as end users of the situation
- Applies just-in-time corrective actions, when applicable, to the transaction to resolve anomalies while in flight or to compensate for them
- Provides end-to-end visibility into exceptional transactions for diagnosis or a manual response
- Collaborates with existing IT resources (in-house exception handling services, network system management, trouble ticketing, business process management)

To support enterprise-class distributed applications, exception management capabilities should handle more than just disruptive exceptions; they should empower the enterprise to better capitalize on revenue opportunities and extend its gains. By delivering these capabilities, exception management lays the foundation for a real-time business enterprise.

The Business Challenge: Exceptions in Distributed Applications

Exceptions are conditions that alter the normal flow of control in a software application and are an integral part of any computing environment. Consequently, it is not surprising that every single programming model has built-in constructs for handling exceptions.

Theoretically, every exception condition anticipated at development can be detected and handled in some fashion. However, the reality is quite different. A study by the National Institute of Standards and Technology estimated that the U.S. economy loses US$59.5 billion per year to bugs and glitches in software.[1] The study further found that almost 80 percent of the problems are discovered after the applications have been put in production. At the time of the study, that was the world of traditional monolithic applications. All components were simultaneously developed and designed for specific purposes. All shared a common technology. Most important, all components were built and owned by a single development team.

Enter SOA, the cloud, SOAP, Representational State Transfer, and other composite application enablers that crystallize traditionally monolithic "one-off" business applications into heterogeneous, distributed networks in which each node is an autonomous service. As more business applications are developed, these networks will overlap and intersect. Moreover, the mantra of composite applications is "reuse and repurpose." A service might be designed as part of one application and later revised, possibly without the knowledge of the original designer, for other applications.

Even the consumers of a composite application are likely to change over time. For example, a service that had been used only internally might be made available to partners or the general public. In such a circumstance, the rules and expectations of use are likely to be radically different from the designer's original understanding or intentions.

There are more moving parts in a distributed application, more dynamism, and many conflicting data models within the same business application. The exception semantics of a service are typically unknown; even the original developer might not remember how the service might behave when not used in a so-called sunny day scenario. What's more, in complex, federated environments, your entire business application is at the mercy of services that you do not own or control.

As a result, services-oriented environments are more susceptible to exceptions than traditional application architectures. Potential losses due to exceptions can be much higher. In addition, software exceptions are not the only type of exceptions that afflict real-time business environments.

Business activity often results in states that cause revenue loss. For example, while committing a business transaction, a Platinum customer might find a particular item on back order. The software

[1] National Institute of Standards and Technology, *The Economic Impacts of Inadequate Infrastructure for Software Testing* (May 2002), www.nist.gov/director/prog-ofc/report02-3.pdf.

behaves as intended and rejects the order. Everything seems to be working as planned in the business system. But, is it?

Typically, business operations teams do not have visibility into the live state of business. At best, business events like this don't surface until they appear in reports, weeks later. By then, they are too old for response. These business inefficiencies, though individually insignificant, quickly add up to have a significant effect on the bottom line.

The Need: Management of Exceptions

Exceptions in distributed applications can take the form of system or application errors (for example, invalid data in requests, transport-level errors, inaccurate responses) or business events (for example, excessive weight of shipment, bad credit for a premier customer), which are not understood by existing applications.

Unfortunately, it is usually the customer (the consumer of a distributed application) who experiences exceptions before anyone within the enterprise. Common examples include opaque messages on an e-commerce Website ("Sorry, unable to process request at this time"), delayed orders, lost packages, and rejected insurance claims.

Each exception occurrence disrupts the customer's experience up front and has a direct impact on the business. Therefore, managing exceptions proactively is of the utmost importance to the business. But what does managing exceptions involve?

To begin with, exceptions must be detected as they occur—in real time. IT and business operations must be able to specify the criteria for spotting exceptions in live business transactions. Typically, these are message patterns that indicate unusual business activity, which might include incongruent reference data, discrepancies in data fields, error messages, and error codes. Sometimes, criteria can be crafted for detecting very specific conditions, for example, "Raise an exception if a premier customer's order is rejected due to mainframe error code D234200." Sometimes nonoccurrence of a message or a pattern is the symptom of an exception.

However, it is impossible to anticipate all patterns; therefore, it is essential that operations teams be able to cast a net as wide as possible across their business system to trap all possible exceptions.

Once an exception is spotted, IT or business operations must be made aware of it immediately. It might be important to alert one or more individuals across different teams. The technical details of an exception might be more pertinent to the IT operations staff. However, the business impact of this exception must be made known to the business operations staff. For example, the error code is of interest to the IT staff, whereas the rejected purchase order and the customer details are important to business analysts.

Those who are notified must then be able to quickly analyze the situation, understand the cause, and devise a remedy. To accomplish this, they need not only the exception message pattern but also the context of the business transaction in which it occurred. IT operations must be able to diagnose and resolve the exception quickly and efficiently—in minutes instead of hours and days. Business

operations must be able to learn about exceptions in real time, with the context necessary to formalize a resolution before customers log complaints.

For a given exception condition, a resolution might be well known. In such cases, it is possible to resolve the exception in flight by applying automated exception handling actions to the exceptional message. Often, the cause of an exception is a simple human error, such as an invalid address, bad identifier, or missing information. Fixing such exceptions might involve simple manual corrections to the exceptional message. IT or business operations staff must be able to apply such corrections quickly through the appropriate resolution tools; for example, sometimes, the only form of resolution is calling the affected end user on the phone. It's easy to see how managing exceptions proactively improves customer service.

Shortcomings of Traditional Approaches

Programmatic exception handling models have been the mainstay of exception management in business applications.

The compilation stage detects and eliminates syntactic errors. Anticipated anomalous business conditions are detected and handled via embedded logic, either in the application source code or in the business process driving the application. Business process management systems often handle exceptions in process definitions, at best, by hardwiring the process definition with corrective actions for a well-defined set of exceptions that might occur while executing the process. Unanticipated conditions and process exits are handled by writing the condition to a log file.

Debugging and testing practices aim to isolate and eliminate logical errors. Quality assurance teams spend countless hours putting the software through scripted production simulations. Then, it is up to the consumers of the production systems to report any exceptions to the technical support organization.

Humans essentially perform the role of exception detectors. The IT operations staff depends on applications and system logs to diagnose problems reported by customers. Patches are applied to applications if problems are deemed severe. Additionally, network system management (NSM) software is used to isolate runtime failures in the hardware or elements of the physical layer and trigger alerts.

The above methodologies are based on the following assumptions:

- The entire system is monolithic.
- The functional requirements are driven by a single business application.
- A central managing entity controls the message flow in the environment.
- Changes in the application components are administered centrally.
- Exceptions are expected to be limited and well defined.

However, as seen in the previous sections, composite applications are distributed, heterogeneous, and federated. Specifically,

• Services are shared and reused by many applications

• Not all components used in an application have been tailor-made for that application

• Application components, such as Web services, can change independently

What's more, distributed applications are dynamic and real-time systems. Traditional approaches for managing exceptions fall short.

The problem is that these models rely on the developer's anticipating all possible exceptional conditions and embedding custom instrumentation (as code) into the application to detect and handle them appropriately. It is infeasible for anyone to imagine all the types of interactions an end user might have with the services and code to prevent every possible runtime glitch that might occur.

Any coding-based technique is also inapplicable when the components in question are ones you do not own or control. How are you going to force a change upon such services?

Such hardwiring also means that any change or update to the exception handling capabilities requires programmatic changes to the application. IT is unable to respond in a timely fashion to the vagaries of a real-time business environment. Business requirements remain unmet, and IT maintenance overheads rise.

Distributed applications introduce a new order of complexity in detecting exceptional conditions. Because the custom instrumentation is isolated to the service into which it has been programmed, it has no context of the actual business transaction in which it is participating. Exceptional conditions that are related to business transactions spanning one or more services across different business applications cannot be detected by exception handling localized within one of the participants. This is particularly true of exceptions that materialize as business events.

The reliance of traditional models on the developer is also the primary reason for resource-intensive IT fire drills. The amount of information available to diagnose the problem is limited, often solely dependent on whether the developer was meticulous about exporting all the available information to an application log. The IT staff often has no way of knowing how a service was consumed when the exception occurred and in what context. There is no easy mechanism for the IT staff to capture the flow of business transactions end to end within the system, as they occur.

In a distributed environment, log entries are likely to be colocated with the service in question. Thus, diagnostic information is fragmented across one or more logs, potentially at geographically different locations or at least on different servers within the same organization. Typically, IT operations teams spend hours sifting through different logs trying to manually piece together the puzzle. It's no wonder that, when diagnosing a complaint, they spend 80 percent of their time trying to simply reproduce the problem.

It is also not surprising that postdeployment exceptions in business systems are the primary cause of finger-pointing and blame across different teams. Ultimately, it is the business that suffers.

The Solution: Exception Management for Distributed Applications

In a dynamic business environment, a state of equilibrium can be established only if the runtime exceptions are detected and intercepted in flight, and localized resolutions are executed to either eliminate these exceptions or mitigate their effect on the rest of the system. Such equilibrium is critical to creating a robust, enterprise-class distributed application. How do you achieve this equilibrium?

Distributed applications require an approach to management that includes capabilities for detecting, diagnosing, and resolving exceptions. Exception management capabilities must span the complete network of distributed application components, including components serving multiple applications within the business environment. Exception management needs to

- Identify the set of components that serve each business transaction
- Provide visibility into the service interactions within the scope of the business transaction
- Detect exceptions in these interactions
- Notify authorized personnel or other services of the situation, if necessary
- Apply (or allow an individual to manually administer) just-in-time corrective actions to the application to resolve the anomaly in flight
- Detect both software errors (system or application errors) and business events in distributed applications, and then act upon these conditions in real time

The nature of exception detection and resolution depends on the system or business orientation of the exception. The NSM software managing the physical infrastructure could best resolve a system error, for example, a service failure. The business-level resolutions necessary to handle a rejected order might already be available in business processes hosted by existing workflow or business process systems or perhaps as a custom exception handling service. To be effective within an enterprise, the system performing exception management must collaborate with these resources. It should accept system errors or business events detected by these other systems, and it should delegate resolutions when appropriate.

Abstraction of exception detection and handling from the actual application fosters a business system that can rapidly adapt to change. It also allows for dealing with exceptions in services that are beyond your control.

The distributed application paradigm provides compelling opportunities for implementing an exception manager to address the challenges described above. However, its advantages are not limited to handling errors. An effective enterprise exception management system also opens avenues for real-time business management in ways never before possible.

Business Transaction Management

With distributed applications, business transactions are executed by arranging existing applications and infrastructure to implement business processes. They incorporate a wide variety of technologies,

deployed across many platforms and organizational boundaries. Common technologies include intermediaries such as enterprise service buses (ESBs), process engines, middleware, and legacy and packaged applications. Composite applications incorporate a range of shared components, such as SOAP services, packaged applications, Enterprise JavaBeans, Plain Old Java Objects, ASP.NET, and databases. However, despite the complexity of such processes, they must behave as single, seamless transactions to business users.

One challenge for the operations staff trying to identify transaction failures and bottlenecks is that the problem could be anywhere. It could result from components in the implementation layer of the service, or it might actually be located in one of many replicated services or in the infrastructure supporting the orchestrated services. Some business transactions can be long running or asynchronous, involving human interaction and spanning multiple systems and several days. Due to this weak link effect, performance and capacity are key challenges for managing distributed transactions. A performance meltdown at any given point could be extremely difficult to track down—making it doubly difficult to meet expectations for performance and availability.

Business transaction management (BTM) addresses these requirements by providing real-time monitoring of each transaction flowing end to end across distributed applications. It covers two areas:

- **Performance management.** Tracks performance and throughput of the components and the transactions.
- **Failure management.** Calls for mechanisms to detect transaction failures and rapidly locates the root cause of the issue.

BTM provides instrumentation for tracking the transactions flowing across an application, as well as detection, alerting, and remediation of various types of unexpected business or technical conditions. It enables application support personnel to search for transactions based on message content and context, such as time of arrival, message type, and client credentials.

BTM can also provide insight into the business data of each message, allowing organizations to modify system behavior based on transaction-specific values. For example, if the services supporting Platinum customers approach thresholds at which service-level agreements will be breached, the BTM system can redirect traffic via routing or load balancing.

Done the right way, BTM will bring immediate and dramatic benefits. Its benefits include a significant reduction in failed transactions across services-based systems, much-lower mean time to repair, and reduced IT maintenance overheads. Of course, the most important benefit is customer retention, because fewer customers are lost due to failed transactions.

Conclusion

Composite applications bring tremendous opportunities for organizations to benefit from a flexible, standards-based framework for building modular applications and tying together information systems. However, these distributed, decentralized systems bring with them the increased likelihood of exceptions that can have adverse affects on business operations.

Distributed applications demand an infrastructure—whether homegrown or purchased—that allows you to detect, diagnose, and resolve exceptions. Exception management not only handles exceptions, but it also provides the ability to increase revenues by managing business operations in real time.

Gartner defines a real-time enterprise as "an enterprise that competes by using up-to-date information to progressively remove delays to the management and execution of its critical business processes."[2] Exception management for distributed applications will be a key enabler of such real-time business environments within an enterprise, empowering IT specialists and business managers to monitor their business environments and proactively act upon system- and business-level contingencies in real time—all without invading the existing business applications and while leveraging existing IT investments.

[2] Gartner, Gartner Research Note, *The Gartner Definition of Real-Time Enterprise,* October 2002.

Managing Exceptions in Distributed Applications
March 2010

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

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com