Exception Handling in Service-Oriented Architecture

Applies to:

Business Experts

Summary

Centrally managing exceptions has been a difficult task for many enterprises. This article describes the importance of Exception Management in a Service-Oriented Architecture.

Author: Rohit Kumar Shukla

Company: HCL Technologies

Created on: 26 November 2006

Author Bio

Rohit Kumar Shukla is a business consultant working with HCL Technologies. After gaining rich experience in SAP R/3 and SAP NetWeaver platform, he is now involved in driving enterprise SOA and BPX strategies.

© 2006 SAP AG

Table of Contents

Exception Management Challenges

Service-Oriented Architecture (SOA) emphasizes loosely coupled and reusable services based on industry standards. Stabilization of and support for Web services standards in most application platforms have made it possible to realize SOA.

However, SOA makes the applications more distributed than before. Applications communicate more and more with services hosted by other departments, by business partners, or by service providers. If exception management is not viewed from this perspective, troubleshooting may become a nightmare. A great deal of time could be wasted in communicating across different groups just to resolve an exception. This poses the following new challenges in exception management:

1. Exception Logging

An application may interact with many services to accomplish a business process. If service providers and consumers store their error logs in different locations (multiple databases, files etc.), support teams must look at multiple logs to identify the cause and resolve the error.

The error logs of services and applications consuming these services may contain information in various formats as they may use different database schemas or different log file formats. It is very difficult to query/report against these distributed logs stored in different places.

During the development of SOA applications, it may not be possible to set up a development environment hosting all the services consumed by the application for debugging purposes. This could be because source code is not available or because it is difficult to set up.

In all of these cases, the error log becomes a key source of debugging information. To resolve these issues, service consumers and service providers need to log the exceptions in a centralized repository. A central repository is essential not only for production, but also for development and test environments for use during the development and testing stages. Using a central repository, support teams troubleshooting an exception have a single place to view the logs. It is usually preferable to keep the repository as a database for easy reporting access.

© 2006 SAP AG

2. Exception Handling

One of the important aspects of exception handling is propagating sufficient information to upstream nodes when an exception occurs. However, when an exception happens inside a Web service, the details of the exception and contextual information is available only within the Web service. Web service specifications provide a `SOAP DETAIL` element in the `SOAP FAULT` structure to carry the exception details, but it is not mandatory for the service to populate the details. Also, the format/schema for carrying the exception details is not defined. This may lead to services populating the `SOAP DETAIL` element with their own custom format or ignoring this element completely. So service consumers either do not get the exception details or they get this information in different formats from different services. For example, an application using 3 services would have to have complex exception handling logic to deal with 3 different formats of exceptions.

Web services do not have the capability of maintaining stack trace information, which is very important for root cause analysis of any exception. The errors logged by services need to be traced back across each service node until the end consumer is reached to perform a root cause analysis.

To solve these issues, common exceptions need to be converted to a standard, predefined exception message to promote consistency and prevent ambiguity to the service consumers. For example, HTTP errors like 404 Not Found, 401 Access Denied, 500 Internal Server Error, etc. can arise because of issues in accessing underlying services, even though the applications they are directly interacting with work without error. A user using a Web site may get an “Access Denied” error when he clicks on a button because an underlying service being used is denying access. This may confuse the user, as he might have been successfully authenticated by the application he is accessing.

These challenges can be addressed by defining an enterprise XML schema for the exception message, logging the exception to a centralized store in every service node and providing adequate information to correlate log entries across service nodes right up to the end consumer. Some standard exceptions that are expected in SOA as described above can be translated to a more meaningful message which will remove any ambiguity and create a consistent feel across applications.

3. Exception Notification

Since the services may have SLAs associated with them, it should be possible to notify appropriate people about the exception and the associated SLA in a timely manner. It should also be possible to communicate some of these exceptions to an enterprise IT management system or another service or application which depends on the service that failed.

This issue can be resolved by an exception management solution that can report the exceptions in real time to the appropriate people and systems involved.

Conclusion

From the above analysis, we can see that there is a clear need for an exception management solution that addresses these challenges in SOA.

It would be ideal to host the solution as a service—providing exception management as an enterprise service would eliminate the need for having multiple implementation frameworks and components for logging and notification catering to different application platforms. Such a service would prevent applications from customizing these frameworks and components and creating needless extra versions. This would greatly help governance, as exception management policies would be easily enforced. It would be possible to know how many applications are really using a service, making it easier for developers, as they would see a consistent exception management framework as they move from project to project.

© 2006 SAP AG

Related Content

Modeling Concepts in Service-Oriented Architecture

Data Management in Service-Oriented Architecture

Ten Myths about Service-Oriented Architecture

© 2006 SAP AG

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.

© 2006 SAP AG