Disclaimer

This document is subject to change without notification and will not comprehensively cover the issues encountered in any customer situation.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS.

The Authors
This document is produced by the Business Process Choreographer Level 3 Support team in Boeblingen, and the Boeblingen Lab services team.

Matthew Hargreaves
WebSphere BPM Development, Customer Support Teamlead
IBM Global Business Services, Enterprise Application Integration

Boris Feist
Senior Accredited IT Specialist, ISSW WW Technology Practice
IBM Software Group, Application and Integration Middleware Software

Acknowledgments:

The authors wish to acknowledge the valuable input for the following Business Process Management Development colleges: Anke Robeller, Gunnar Wilmsmann, Hermann Akermann.
# Table of Contents

1 Scope............................................................................................................................3  
2 Important Parameters and Properties.................................................................3  
   2.1 BPEL Process Model Settings.................................................................3  
      2.1.1 BPEL Transaction Boundaries.........................................................4  
      2.1.2 Continue-on-error Flag.................................................................4  
   2.2 SCA Component Parameters.................................................................5  
      2.2.1 Suspend Transaction Flag............................................................5  
      2.2.2 Join Transaction Flag.................................................................5  
      2.2.3 Invocation Style ...........................................................................6  
   2.3 Server Settings.................................................................6  
      2.3.1 Global Transaction Timeout..........................................................6  
   2.4 Settings For Distributed Transaction Propagation (WS-AT)..........................7  
      2.4.1 Enabling Propagation From the Web Service Caller (BPEL Process)....................7  
      2.4.2 Enabling Participation of the Web Service in the Propagated Transaction........8  
3 Server Behavior Effecting Transactionality & Fault Handling.....................8  
   3.1 Fault Type..............................................................................................................9  
      3.1.1 Exception Propagation.................................................................9  
      3.1.2 Fault / Exception Type & Rollback................................................9  
   3.2 BPEL Transaction Handling...........................................................................10  
      3.2.1 Repeated ServiceRuntimeExceptions During Transaction Retry..............14  
   3.3 Fault Handling.................................................................................................15  
      3.3.1 Fault Handling – Sequence of Triggered Handlers.........................15  
      3.3.2 Fault Handling - Transactional Considerations...............................16  
   3.4 Service Integration Bus Retry & 'Maximum Failed Deliveries'.................16  
4 Recommendations.............................................................................................17  
5 References & Further Reading........................................................................19
1 Scope

This document intends to provide an insight into fault handing and transactionality of web service invocations from a BPEL process. It explains the different fault types that can lead to failures and rollbacks and some modeling constructs and properties that influence the fault handling and transactionality.

The current version of the document focuses on web services invocations via SOAP/HTTP binding and is limited to JMS-based BPEL navigation (default up to Process Server v6.2). It does not explain differing behavior when using Work manager based navigation, nor does it cover quality-of-service interaction with SOAP/JMS web services, or web service invocations not originating from a BPEL process.

The document does not exhaustively cover all settings effecting transactionality. For example, some of the settings on deployments descriptors and explicit calls to the transaction service are not covered her. The intent of the document is to provide an understanding of how transaction boundaries can effect the error handling behavior around web service invocations from a business process running in Websphere Process Server. It is intended to assist in achieving deterministic behavior during application design and system test.

The test scenarios were verified on systems in the IBM Böblingen Lab and are based on the following generally available software version: Websphere Process Server version 6.2.

2 Important Parameters and Properties

This section will explain the most important parameters that need to be adjusted to achieve the desired behavior of a transactional integration application. As WebSphere Process Server provides a highly sophisticated infrastructure, where configurable properties exist at several layers. As settings at one layer can affect those of another layer, we need to be aware of them as a whole.

The following subsections discuss parameters of the SCA infrastructure as well as parameters that are tied to BPEL processes. Furthermore they mention server-side settings that influence the execution of BPEL integration applications.

2.1 BPEL Process Model Settings

There are a couple of major factors in a BPEL process, which influence how a process behaves in an error situation. One factor is the definition of BPEL transaction boundaries, another is the type of fault that occurs either within the process of at an interacting service. Yet another important factor is the policy, which controls how a business process deals with an error situation, e.g whether it continues processing or waits at the point of failure.
2.1.1 BPEL Transaction Boundaries

BPEL processes are generally distinguished into two categories, short runners (microflows) and long runners (macroflows). The most significant characteristic of a microflow is that it is always executed in a single thread and transaction, which spans from the beginning to the end of the process. A macroflow, on the other hand, consists of several transactions. The activity types, the transactional behavior property on the activities and the process engine behaviour are the factors that influence the scope of a transaction. A transaction boundary is a point in the process flow, where the entire process instance state is persisted into the Process Server database. The next transaction will load the process instance state from the database and will process the next transaction, potentially in a new thread.

While some activities in a macroflow, such as a human task activity, always require a transaction boundary at some point, the transactional behavior property on the activities allows us to take some influence. The available values are “Commit before”, “Commit after”, “Participates” and “Requires own”. For instance, by using “Participates” on several adjoining activities, we can request that the engine normally executes these activities in one transaction. This can be set on the process activity in the process model e.g. using Websphere Integration Developer.

It is important to understand, however, that the transactional behavior property is merely a hint to the engine to interpret the process that way. It does not enforce transactional boundaries. In case of failure, the engine may successively narrow the transaction scopes during a retry, to process as far as possible up to the point of failure.

2.1.2 Continue-on-error Flag

The continue-on-error flag can be enabled on the process level or on particular activities. If set to “false” and a fault is not caught on the enclosing scope, or handled through a local fault handler, the flag will cause the activity or process to stop. This can be helpful to allow a human or automated administrator to repair a situation via the API or tools such as the BPC Explorer. This can include restarting a stopped process instance or carrying out other administrative actions on the activity such as force retry, skip, jump, force Navigate, etc.

Simply viewed, Continue-on-error only plays a role if a fault is not handled on the enclosing scope of the fault origin or local fault handler of an activity.

Some further comments on the use of continue on error are given in the recommendations section of this paper.

References:
2.2 SCA Component Parameters

SCA invocation style and transaction settings mainly influence the way in which SCA components and external services interact, and how they participate in transactions. These settings help to determine the scope of the transaction. Settings on both the calling and called components together (in addition to further none-SCA server side settings) must all be considered together when determining the boundaries of the transaction.

When it comes to propagation of a transaction at the point of invocation the following from the information center provides useful guidance:

"A number of conditions need to exist in order for a transaction to propagate successfully:

- A transaction qualifier must be set on the implementation, and the qualifier must specify a global transaction.
- The implementation must be invoked synchronously.
- The component must join a propagated transaction.
- The transaction cannot be suspended on any of the participating references.
- The preferred interaction style on a participating interface cannot influence the invocation style to become asynchronous."

Reference:

The SCA parameters effecting the above are discussed in the following subsections.

2.2.1 Suspend Transaction Flag

The “suspend transaction” flag on the reference of an SCA component controls, whether a synchronously called service shall be part of the transaction of the SCA component or not. If the value is set to “true”, the SCA component suspends its own transaction and waits for the call to return. After the return of the call it resumes processing the current transaction. It the value is set to “false” the called service may – depending on its capability to do so – join in the transaction of the caller. For instance, a value of “false” may allow a WS-AT enabled web service to join in the transaction of a BPEL process.

Reference:

2.2.2 Join Transaction Flag

The join transaction qualifier can be set on the import binding for synchronous invocations. It is ignored for asynchronous invocations.
2.2.3 Invocation Style

“The invocation style of a service call refers to the type of call the client makes when invoking the service. The call can be either synchronous or asynchronous. The style affects whether or not a transaction can propagate to the target service. Global transactions can only propagate if the client invokes the target service synchronously. Global transactions do not propagate when the client calls the services asynchronously, regardless of how the qualifiers that affect transaction propagation are set.”

To ensure the desired Invocation style is used please refer to the following article

Reference:

“In general, when considering an interaction from one component (source or client) to another (target), the service client determines what type of invocation will be used. “

The invocation style on an import binding can be set to “synchronous” or “asynchronous”. It is usually this parameter that will determine the invocation style, however there are exceptions since some components can not, or should not, be called synchronously. See the full article (link above) on developer works for details.

2.3 Server Settings

This subsection describes settings that are configured globally on the application server.

2.3.1 Global Transaction Timeout

The global transaction timeout, which can be configured on a WebSphere Application Server or WebSphere Process Server system, will ultimately cause a transaction to roll back if it runs for longer than the set timeout interval.

Rollback may not be immediate since the timeout is set by the transaction service and may not be noted immediately by the active thread.
Transactionality and Fault Handling in WebSphere Process Server Web Service Invocations

For more details in settings effecting the Websphere transaction service, see the following:

Reference:
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?
topic=com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/udat_contranserv.html

Illustration 1: Global Transaction Lifetime Timeout Setting

2.4 Settings For Distributed Transaction Propagation (WS-AT).

In addition to the usual transactional settings on the SCA components and BPEL activities, it is possible to enable propagation of transactions in a distributed environment.

Web Services Atomic Transaction (WS-AT) is a standard and technology that provides transactional support for Web services. It allows distributed Web Service applications, and the resources they use, to take part in distributed global transactions. Some relevant settings relating to WS-AT can be made in the deployment descriptors and are noted in the following subsections.

Reference:
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?
topic=com.ibm.websphere.wsfep.multiplatform.doc/info/ae/ae/cjta_wstran.html

2.4.1 Enabling Propagation From the Web Service Caller (BPEL Process).

In general terms this would be done on the via the deployment descriptor of the EJB module. However within the scope of the scenarios handled by this document this should not be necessary for the Process EJB since propagation it is handled by the system.
2.4.2 Enabling Participation of the Web Service in the Propagated Transaction.

Again this is realized in the deployment descriptor. For example, using Websphere Integration Developer - if the invoked service is implemented as a Servlet, the settings to allow WS-AT participation in an incoming request can be set as seen in the image below.

Illustration 2. allowing service execution in WS-AT context.

3 Server Behavior Effecting Transactionality & Fault Handling.

Websphere Process Server provides a robust infrastructure, with built-in ability to
handle some of the error situations that may occur. We also need to be aware of these to fully understand the transactionality of all an business process application based on Webspere Process Server.

The following subsections describe how the runtime automatically reacts to situations such as rollbacks and faults. Reference is also made to some of the previously discussed parameters and properties.

3.1 Fault Type

In regard to transaction and fault handling in WebSphere Process Server we distinguish two types of fault:

- Service Runtime Faults (or Exceptions), which are unmodeled faults. System exceptions usually result in a ServiceRuntimeFault.
- Service Business Faults (or Exceptions), which are modeled in the service interface

3.1.1 Exception Propagation

Service Business Faults are declared (modeled in the service interface) and are therefore expected to be handled by the caller. This allows for a business fault to be handled by a fault handler in the business process, and to continue process navigation in case of a successful fault handling.

Service Runtime Faults on the other hand are not normally expected. In a synchronous invocation scenario they would not normally be returned by SCA to the caller, however Business Process (BPEL) component or Human task component are the exceptions to the rule. This means it becomes possible to react to a ServiceRuntimeException within a BPEL process, for example via a ‘catch all’ defined in a business process’ fault handler.

3.1.2 Fault / Exception Type & Rollback.

The type of fault dictates, whether a current transaction requires a rollback or can still commit.

A ServiceBusinessException does not implicitly set the “rollbackOnly” flag to “true”. Instead the Business exception is returned to the caller via a Business Fault. Thus a business fault does not necessarily lead to a rollback.

On the other hand a ServiceRuntimeException in an SCA target component, automatically sets a so-called “rollbackOnly” flag to “true”. This causes the transaction to roll back when it hits the next transaction boundary.

In a synchronous call if the source process component runs in the same transaction as the target component this would result in the process navigation also being rolled back.
However, if the target runs in a separate transaction (as is always the case in asynchronous invocation) the ServiceRuntimeException will be returned to the process without implicitly causing a rollback in the process navigation transaction.

### 3.2 BPEL Transaction Handling

Previously it was explained that long running processes persist to the database at the end of every transaction. In addition the process engine sends itself a navigational JMS message to the so-called int queue. A queue listener will read the message and, according to its information, reload the process instance state from the Process Server database.

If the BPEL transaction cannot complete for some reason and it comes to a rollback, the navigational message, which triggered the transaction, is rolled back to the int queue. At the same time a counter is increased. After three delivery attempts, the message will not be put back into the int queue. Instead, it will go to the retention queue, where it remains, until another navigational action (triggered by another process instance) leads to a transfer back to the int queue. The retention queue also disposes of a counter, which will eventually lead to the message being put into the hold queue rather than the retention queue. Once a message is on the hold queue, it needs to be dealt with manually. This is typically done by correcting the situation that caused the transaction to fail and then replaying messages on the hold queue.

The following picture explains the navigation message handling in case of a rollback.

---

1
Illustration 3: Rollback and Replay of BPEL Transactions

**Figure 1:**
Navigation from one transaction to the next in a long-running flow is performed by navigational messages (here called n1, n2 and so forth). A finished transaction puts a navigational message into the int queue. The message contains instructions as to how to continue navigation of the flow in the next transaction.

**Figure 2:**
When it comes to a rollback (see descriptions in this paper when exactly a rollback happens), the navigational message, which started the failing transaction, is rolled back to the int queue. The message with its continuation instructions is read again by the process engine and therefore the transaction is retried.

**Figure 3:**
After 3 tries without success the message will go to retention queue, as the engine assumes that further retries will remain unsuccessful due to a persisting problem. At the same time, a counter on the message keeps track of how often the message has already hit the queue. Navigation from here on continues only after the engine performs 12/20.
a successful navigation step (for any other process instance). This will lead to a replay of all messages in the retention queue back into the int queue.

Processing will resume as described in figure 2, as long as the retention queue counter is below the threshold.

The retry limit for the retention queue can be configured in the Business Flow Manager settings in the administrative console, as shown in the screen shot:

Illustration 4: Configuration of the Retry Limit

Figure 4:
When the navigation message is put into the retention queue and the counter for this message has reached the retry limit, the message will not be put back into the int queue. Assuming that there is a permanent problem completing this navigation step, the message will be forwarded to the hold queue.

Figure 5:
Messages on the hold queue, a type of failed event, aren’t processed off the queue...
automatically. However, failed events can be handled by the Failed Event Manager, either in the administrative console (see screen shot) or programmatically via the failed event manager API. By resubmitting the message, the process continues normal processing. If the process instance has been deleted in the meantime, resubmitting will discard the message.

How often is the retention queue hit by a message?

The retry limit on the Business Flow Manager specifies how often a message can be put into the retention queue before it is stored in the hold queue for manual recovery. In order to change the value, got to Application Servers --> <server_name> --> Business Flow Manager. The setting can also be applied on the cluster scope for high availability environments.
3.2.1 Repeated ServiceRuntimeExceptions During Transaction Retry.

In contrast to the described behavior above, the process engine (Business Flow Manager) helps to optimize calls to a service which repeatedly return a ServiceRuntimeException. If the invoked service raises the same runtime exception in successive transaction retry attempts, the runtime exception is stored via a separate transaction. During the next retry the service is not called again but the stored exception is restored and the navigation continues with the fault handling. See the section 'Fault Handling – sequence of triggered handlers' and 'Continue-on-error' for
further details.

3.3 Fault Handling

When a fault occurs in a process, the navigation continues with the fault handler or fault link.

A fault handler is a collection of activities that run when a fault is thrown on the activity with which the handler is associated. A fault link redirects the process navigation to an alternative (fault) process path when a fault is thrown on the activity with which the handler is associated.

3.3.1 Fault Handling – Sequence of Triggered Handlers

The following excerpt from the WebSphere Process Server information center explains the standard fault handling procedures of a BPEL process:

“Fault handlers can be specified for invoke activities, scopes, and on the process. Fault links can be specified for generalized-flow activities. Scopes and all basic activities, except for throw and re-throw activities, can be the source activity of a fault link.

A fault handler or fault link can catch a specific fault name, fault type, or both. When a fault occurs, Business Flow Manager uses the following rules to match the fault with a fault handler or fault link on the enclosing scope, or on the activity where the fault occurred.

- If an invoke activity without a fault handler or any other basic activity is the source of one or more fault links, Business Flow Manager tries to find a matching fault link. If a fault link is not available, it then tries to find a matching fault handler on the enclosing scope.

- If an invoke activity or a scope with one or more fault handlers is the source of one or more fault links, Business Flow Manager tries to find a matching fault handler. If a fault handler is not available, it runs the default fault handler and then tries to find a matching fault link. If a matching fault link is not available, it tries to find a matching fault handler on the enclosing scope.

- If the fault does not have any associated fault data, Business Flow Manager uses a fault handler or fault link with the matching fault name. If a fault handler or fault link is not found, it uses the catch-all fault handler or fault link if one is available. A fault without any data cannot be caught by a fault handler or fault link that has a fault variable defined.

- If the fault has associated fault data, Business Flow Manager uses a fault handler or fault link with the matching fault name and a fault variable with a type that matches the type of the fault data. If a fault handler or fault link is not found that matches the name and fault data type, it uses a fault handler or fault link
without a fault name and a fault variable with a type that matches the type of the fault data. If a suitable fault handler or fault link cannot be found, it uses the catch-all fault handler or fault link if one is available. A fault with data cannot be caught by a fault handler or fault link that does not have a fault variable defined.

If a fault is raised that does not match any of the fault handler or fault link definitions, the default fault handler is started. The default fault handler is not specified explicitly. The default fault handler runs all of the available compensation handlers for the immediately enclosing scopes in the reverse order of completion of the corresponding scopes. If the scope is the source of one or more fault links, Business Flow Manager then tries to find a matching fault link. If a matching fault link is not available or the scope is not the source of any fault links, the default fault handler re-throws the fault to the next level, that is, the enclosing scope or the process. On this next level, Business Flow Manager again tries to match the fault to the fault handlers or fault links that are available.

If the fault is not caught by any of the specific fault handlers or fault links, or by the catch-all fault handler or catch-all fault link, the fault reaches the process scope, and the process ends in the failed state. Even if a fault handler catches the fault on the process scope and handles it, the process still ends in the failed state.

For full details of fault handling and the fault handler chain please refer to the following information center link:

Reference:

3.3.2 Fault Handling - Transactional Considerations

It is worth noting that transaction rollback is not always immediate. This can lead to the possibility of a fault handler or fault link being triggered even in case where a rollback is expected (where rollbackOnly on it set to true on the transaction). Note however that the rollback should still occur preventing the overall transaction, including activities in the fault handler, from being committed.

3.4 Service Integration Bus Retry & 'Maximum Failed Deliveries'

With a synchronous invocation, a single attempt to run the invoke activity results in a single attempt to run the back-end service. However, for asynchronous invocation SCA uses the Service Integration Bus (SIB) to transport messages between
components. The SIB infrastructure makes use of a further retry mechanism. This retry mechanism is independent of the BPEL retry mechanism. Since this occurs after the call out from the BPEL process a single attempt to run the invoke activity can result in multiple attempts to run the back-end service (runtime error case with rollbacks). In this case the transaction on the source (BPEL) side is committed and a message is sent via SCA & SIB. A new transaction is started to process the message however the transaction fails and is rolled back.

If the transaction fails repeatedly it will retry several times depending on the value of the 'maximum failed deliveries' on the SIB destination for the target module.

Further details can be found in the ‘Retry Configuration’ section of the following reference:

Reference:

4 Recommendations

Recognize transaction boundaries & exceptions that will effect transactional behavior.
Understanding the boundaries of your transactions and the circumstances that can cause rollback are more critical than being able to track and explain the exact number of rollbacks and retries observed at different levels in the infrastructure. If using transactions correctly it should not be important to know the exact number of retries made, more important is to know that the transactional resources will either be committed or rolled back, as a unit, within the transaction scope.

Use short runners in conjunction with WS-AT to enforce atomicity
Only a microflow guarantees that the entire flow runs in one transaction. So if you want to invoke several participants that are all capable of distributed transactions and the two phase commit protocol, then you can leverage the power of a distributed atomic transaction. If either the coordinator or any of the participants fail, the entire atomic transaction will roll back altogether. This saves you from writing manual compensation or undo logic.

Use long runners in conjunction with compensation
In contrast to a microflow, a macroflow does not allow you to enforce atomic transaction boundaries. As explained above, transaction scopes may be narrowed down by the engine, thus resulting in a partial execution of your originally intended transaction. If you need to use distributed atomic transactions, then you should externalize the portion of your macroflow that needs to be one transaction, into a microflow. This way, you can
set the macroflow transaction to span only the one invoke activity that calls the microflow, which in turn calls all the other transaction participants.

Instead of atomic transactions, there is a concept of “compensations”, which fits better with the concept of long-running flows. On a macroflow, you can define compensation activities on an activity level, or you can attach compensation handlers to scopes. In case of a failure, the engine will invoke the compensation logic from the innermost scope, which may in turn trigger compensation on higher levels. Compensation logic itself is modeled in exactly the same manner as the normal process logic is.

**Unset Continue on error with invokes in long running processes.**

With long running (macroflow) BPEL processes ensure a process Administrator is set, and unset “Continue On Error” for invoke activities, and a process administrator defined. This way the administrator has the possibility to repair the situation.
5 References & Further Reading:

WebSphere Process Server invocation styles

Exception handling in WebSphere Process Server and WebSphere Enterprise Service Bus

Transactions & SCA (Information Center)

Error handling in WebSphere Process Server, Part 1: Developing an error handling strategy

IBM WebSphere Process Server Best Practices in Error Prevention Strategies and Solution Recovery

Transactional properties and solution recovery

Transactionally integrating remote Web services with BPEL processes in WebSphere Process Server v6.0.2: