



IMPLEMENTATION OF ERROR HANDLING

Author: Natascha Schoenfeld
Creation Date: June 16, 2007
Last Updated: June 16, 2007
Document Ref: <Document Reference Number>
Version: DRAFT 1A

Approvals:

<Approver 1>

<Approver 2>

Table of Contents

Table of Contents	2
Purpose	3
Software	4
Overview	5
Inbound interactions	5
Outbound interactions	5
Requirements	5
Solution	6
Consequently categorize connectivity failures	6
Implement the Fault Handler	7
Model the the Fault Handling	12
Appendix- Description of sources	14

Purpose

The purpose of this document is to describe the main implementation features of the error handling solution implemented for the Toll Collect System in Germany, in order to provide other Oracle staff members with an overview for a reusable solutions.

All code parts have been developed by T-Systems Berlin and therefore should be treated with high confidentiality.

Software

mySAP Adapter 10.1.3.1 and SOA Suite (BPEL PM) 10.1.3.1

Overview

There are 21 interfaces, all of them interacting with SAP. For Outbound interactions with SAP custom RFC modules are used, for inbound interactions with SAP custom iDocs are used.

Inbound interactions

There is no consequent behavior in the error categories thrown by mySAP and Oracle Technology adapters.

- mySAP adapter cannot use rejection handlers (as the Oracle Technology adapters) for failing inbound interactions (delivery of SAP iDocs to BPEL).
- Inbound interactions with the AQ adapter throw rejection Messages (Doc Link) both for invalid message formats and inavailability of the dehydration store. Opposed to the documentation no FATAL error handler can be used for handling the infrastructure failures.
- Inbound interactions with the AQ adapter do not result in rejected messages, if oc4j data sources are used. The implication is using JCA connection pooling for inbound interactions with the AQ Adapter.

Outbound interactions

- Both connectivity and data validation errors during outbound interactions with SAP cause the BPEL PM to throw binding errors. This behavior prevents the implementation of retry mechanisms for handling connectivity errors.
- There is a different behavior between AQ and DB Adapter regarding handling of SQL Codes during outbound interactions. For example the AQ adapter will throw a remote fault if it faces ORA- value too long error.

Requirements

- Consequent and centralized error handling for inbound and outbound interactions for all adapters.
- The administrator should be able to signal an instance to resume its work, without additional efforts.
- Prevent instances to be created while partners for outbound interactions (SAP, AQ or DB) are unavailable (Throttling).

Solution

Consequently categorize connectivity failures.

Implement an error filter for being able to control the behavior of the mySAP Adapter in case of SAP connectivity problems. Error filters can be also be implemented for technology adapters, in our case those were not used.



```
<binding name="jcabinding"
    type="Z_SD_MAUT_PRB_INPUT_3:Z_SD_MAUT_PRB_INPUT_3PortType">
    <jca:binding AddonNamespace="urn:sap-
com:document:sap:rfc:functions.response"

XMLRecordConverterCallout="com.ibi.afjca.oracle.XMLRecordConverterImpl"

ExceptionFilter="com.tsystems.etc.bpel.SAPExceptionFilter" />
    <operation name="Z_SD_MAUT_PRB_INPUT_3">
        <jca:operation FunctionName="PROCESS"

InteractionSpec="com.ibi.afjca.cci.IWAFInteractionSpec">
    <input>
        <jca:header/>
    </input>
    <output>
        <jca:header/>
    </output>
    </jca:operation>
</operation>
</binding>
```

Install the filter class under \$BPEL_HOME/ system/ classes and restart the server

use the filter in the WSDL's for the outbound interactions with SAP like the following example:



```

package com.tsystems.etc.bpel;

import java.util.Map;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import javax.resource.ResourceException;
import javax.resource.cci.InteractionSpec;
import oracle.tip.adapter.api.exception.PCResourceException;
import oracle.tip.adapter.api.exception.PCRetriableResourceException;
import org.apache.log4j.Logger;
import com.sap.mw.jco.JCO;

public class SAPEExceptionFilter implements
oracle.tip.adapter.api.exception.ExceptionFilter {
    private static final String REGEX =
".*com\\.sap\\.mw\\.jco\\.JCO\\$\\w*Exception: \\((\\d{3})\\)"
(JCO|RFC)_ERROR_[A-Z]+: .*";
    private static final Pattern PATTERN = Pattern.compile(REGEX,
Pattern.DOTALL);
    private static final Logger LOGGER =
Logger.getLogger("handlers.collaxa.cube.engine");
    public void init(InteractionSpec spec, Map props) {
        LOGGER.info(this + ".init(): " + spec);
        LOGGER.info(this + ".init(): " + props);
    }
    public PCResourceException applyFilter(ResourceException rex) {
        Exception ex = rex.getLinkedException();
        String msg = ex != null ? ex.getMessage() : null;
        int group = -1;
        // Extract JCO error code
        if (msg != null) {
            Matcher m = PATTERN.matcher(msg);
            if (m.matches()) {
                group = Integer.parseInt(m.group(1));
            }
        }
        // Retriable if JCO_ERROR_COMMUNICATION or
JCO_ERROR_LOGON_FAILURE
        PCResourceException pcex = null;
        switch (group) {
            case JCO.Exception.JCO_ERROR_COMMUNICATION:
            case JCO.Exception.JCO_ERROR_LOGON_FAILURE:
                pcex = new
PCRetriableResourceException(rex);
                break;
            default:
                pcex = new PCResourceException(rex);
        }
        // Set error message and code
        if (msg != null) {
            pcex.setEISErrorMessage(msg);
        }
        pcex.setEISErrorCode(String.valueOf(group));
        LOGGER.info(this + ".applyFilter(): " + rex + " linked to
" + ex + "
wrapped with " + pcex.getClass().getName());
        return pcex;
    }
    public void destroy() {}
}

```

Implement the Fault Handler

The fault handler is a BPEL-Process which supports different message types in its signature and can be initiated by different operations.

Fault Categories

Following fault categories have been defined:

- *Binding Faults*
Non-retriable faults for outbound interactions.
- *Remote Faults*
Communication/ Connectivity failures during Outbound Interactions, normally retrievable.
- *Standard Faults*
For example Transformation or Assign Faults during the process instance execution.
- *RejectionFaults*
Faults which occur during inbound interactions.

A lookup table in the database stores the fault categories.



```
CREATE TABLE t_fault_type (fault_type VARCHAR2(50) NOT NULL,
CONSTRAINT pk_fault_type PRIMARY KEY (fault_type)
);
INSERT INTO t_fault_type (fault_type) VALUES ('binding');
INSERT INTO t_fault_type (fault_type) VALUES ('fatal');
INSERT INTO t_fault_type (fault_type) VALUES ('rejection');
INSERT INTO t_fault_type (fault_type) VALUES ('remote');
INSERT INTO t_fault_type (fault_type) VALUES ('standard');
```

Fault Message Structure

The message types used are derived from the structure of the BPEL Runtime Message described in RuntimeFault.wsdl and JCAErrorHandler.xsd respectively (located on \$BPEL_HOME/bpel\system\xml1lib\jca.



```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified" elementFormDefault="qualified"

    targetNamespace="http://xmlns.oracle.com/FaultHandler/XSD"
    xmlns:tns="http://xmlns.oracle.com/FaultHandler/XSD"
    xmlns="http://www.w3.org/2001/XMLSchema">

    <element name="RuntimeFaultMessage"
type="tns:RuntimeFaultMessageType"/>
    <element name="StandardFaultMessage"
type="tns:StandardFaultMessageType"/>

    <complexType name="RuntimeFaultMessageType">
        <sequence>
            <element name="code" type="string"/>
            <element name="summary" type="string"/>
            <element name="detail" type="string"/>
            <element name="process" type="string"/>
            <element name="payload" type="string"/>
        </sequence>
    </complexType>
    <complexType name="StandardFaultMessageType">
        <sequence>
            <element name="fault" type="string"/>
            <element name="process" type="string"/>
            <element name="payload" type="string"/>
        </sequence>
    </complexType>
</schema>
```

Logging and Throttling for Inbound and Outbound interactions

The FaultHandler BPEL Process will be called by the adapter framework for all inbound interactions which fail due to different reasons (i.e Message Format incompatibility) . All Interface Processes use a specific

partnerLink property which guides the adapter framework to use this BPELProcess in order to handle inbound interaction failures.



```
<activationAgent
  className="oracle.tip.adapter.fw.agent.jca.JCAActivationAgent"
  partnerLink="PL_PRB_BLD_QUEUE">
    <property name="activationInstances">30</property>
    <property name="portType">Dequeue_ptt</property>
    <property
      name="rejectedMessageHandlers">bpel://handlers|FaultHandler|handleRejection|message</property>
    <property name="operation">Dequeue</property>
  </activationAgent>
```

The FaultHandler BPEL Process will be called by the FaultHandlers of every interface BPEL Process for both logging and throttling purposes. Throttling affects all outbound interactions with the partner systems (AQ,DB and SAP). For achieving this goal the FaultHandlerProcess will switch the process lifecycle to *'inactive'* automatically after the count of the allowed failures per fault category is exceeded. Those parameters are pre-configured as BPEL Process Preferences in the Deployment descriptor of the FaultHandler BPEL Process.



```
<?xml version = '1.0' encoding = 'UTF-8'?>
<BPELSuitcase>
  <BPELProcess id="FaultHandler" src="FaultHandler.bpel">
    <partnerLinkBindings>
      <partnerLinkBinding name="client">
        <property name="wsdlLocation">FaultHandler.wsdl</property>
      </partnerLinkBinding>
    </partnerLinkBindings>
    <preferences>
      <property name="MAX_REJECTION"
        encryption="plaintext">3</property>
      <property name="MAX_BINDING_FAULT"
        encryption="plaintext">1</property>
      <property name="MAX_REMOTE_FAULT"
        encryption="plaintext">3</property>
      <property name="MAX_STANDARD_FAULT"
        encryption="plaintext">10</property>
    </preferences>
  </BPELProcess>
</BPELSuitcase>
```

The count of process instances which failed due to a specific fault will be logged into a database table (T_FAULT_COUNT). The count of failures stored in this table will be compared to the current configuration properties described above in order to decide if the lifecycle of the process should be set to *'inactive'*. Only the system administrator can reactivate the process lifecycle through the BPEL Console.



```
CREATE TABLE t_fault_count (  
    fault_date      DATE          NOT NULL,  
    fault_type      VARCHAR2(50)  NOT NULL,  
    fault_host      VARCHAR2(50)  NOT NULL,  
    fault_process   VARCHAR2(100) NOT NULL,  
    fault_count     NUMBER(8)     NOT NULL,  
    CONSTRAINT uk_fault_count UNIQUE (fault_date, fault_type, fault_host,  
    fault_process),  
    CONSTRAINT fk_fault_count FOREIGN KEY (fault_type) REFERENCES  
    t_fault_type (fault_type) ON DELETE CASCADE  
);  
  
CREATE INDEX ix_fault_count ON t_fault_count (  
    fault_date      DESC,  
    fault_type      ASC,  
    fault_host      ASC,  
    fault_process   ASC  
);
```

The FaultHandler BPEL Process will store logging information in the table T-FAULT for monitoring purposes. Note, that this is necessary, because the auditTrail of the domain in which the FaultHandler runs, is switched off.



```

CREATE TABLE t_fault (
  fault_id          NUMBER(12)          NOT NULL,
  fault_date        DATE                 NOT NULL,
  fault_type        VARCHAR2(50)        NOT NULL,
  fault_host        VARCHAR2(50)        NOT NULL,
  fault_process     VARCHAR2(100) NULL,
  fault_payload     CLOB                 NULL,
  fault_message     CLOB                 NOT NULL,
  fault_status      NUMBER(1)           DEFAULT 0,
  CONSTRAINT uk_fault UNIQUE (fault_id),
  CONSTRAINT chk_fault CHECK (fault_status IN (0, 1)),
  CONSTRAINT fk_fault FOREIGN KEY (fault_type) REFERENCES t_fault_type
(fault_type) ON DELETE CASCADE
);

CREATE INDEX ix_fault ON t_fault (
  fault_date        DESC,
  fault_type        ASC,
  fault_host        ASC,
  fault_process     ASC
);

CREATE SEQUENCE s_fault
  INCREMENT BY 1
  START WITH 1
  MAXVALUE 999999999999
  MINVALUE 1
  CYCLE
  CACHE 100
;

CREATE OR REPLACE TRIGGER ti_fault BEFORE INSERT ON t_fault
  FOR EACH ROW
  BEGIN
    FOR rec IN (SELECT s_fault.nextval AS new_id FROM DUAL)
    LOOP
      :new.fault_id := rec.new_id;
    END LOOP;
  END
;
/

```

The error handler uses <JavaExec> activities in order to perform the actions described above.



```
<bpelx:exec name="processRejection" language="java" version="1.5">
  <![CDATA[Logger logger =
    Logger.getLogger("default.collaxa.cube.engine");
    FaultHandlerHelper helper =
    FaultHandlerHelper.getInstance();
    Object process = null;
    Object payload =
    getVariableData("rejectionPayload");
    Object message =
    getVariableData("rejectionMessage");
    String host =
    helper.getHost(getVariableData("faultHandlerURL"));
    try {
      String rejectionId = ((Element)getVariableData("rejection",
"message")).getAttribute("tns:RejectionId");
      process = rejectionId.substring(12, rejectionId.indexOf('_',
12));
      payload = Base64Decoder.decode((String)payload);
      helper.store("rejection", host, process, payload, message);
      int count = helper.updateCount("rejection", host, process);
      String threshold = getPreference("MAX_REJECTION");
      if (threshold != null && count % Integer.parseInt(threshold) ==
0) {
        helper.retireProcess(getLocator(), process);
        String msg = "Maximal number of rejections: process " + process +
" retired.";
        addAuditTrailEntry(msg);
        logger.error(msg);
      }
    } catch (Exception exc) {
      addAuditTrailEntry(exc);
      logger.fatal("FaultHandler.processRejection", exc);
    }
  ]]>
</bpelx:exec>
```

Runtime Settings

The FaultHandler BPEL process uses the configuration properties *inMemoryOptimization=true* and *completionPersistPolicy=faulted* in order to be able to do its work even in case of dehydration store failure.

It is deployed in a domain where following domain settings have been applied:

Domain Settings
deliveryPersistPolicy=off.immediate
auditLevel=off
completionPersistLevel=all

Model the the Fault Handling

Inbound Interactions

The bpel.xml descriptor of all interfaces must be configured as described above In order to use the FaultHandler for inbound interactions. Note that inbound interactions with AQ adapter do not use the JDBC Data Sources, they use the JCA Connection pool. The reason to choose the JCA Pool was, that the transaction will not rollback to the AQ and the message is only recoverable through the BPEL recovery console. This behavior was not acceptable by the customer, especially for the most critical interface

moving 2 mio messages from AQ to SAP. If you use the JCA managed connections the rejection handler can be used.

Outbound Interactions

As a pre-requisite, all adapters used must be able to differentiate retrievable and non-retrievable errors in a consequent manner. there are two alternatives In order to achieve this behavior there are two alternatives.

1. Implement error filters. This is mandatory for the mySAP adapter in JCA Configuration, because it is always throwing a binding fault.
2. Apply patches below, because the AQ/ DB Adapter behave differently (throw always remote faults)



[Bug.5838073](#)(74) DB ADAPTER THROW REMOTE FAULTS FOR ORA ERRORS THAT ARE NOT RETRIEABLE:

[Bug.5877231](#)(74) AQ ADAPTER THROW REMOTE FAULTS FOR ORA ERRORS THAT ARE NOT RETRIEABLE:

Following patches should be applied in order to be able to log messages.



[Bug.5672007](#)(74) XPATH EXTENSIONS GETFAULTASSTRING AND GETFAULTNAME MISSING IN 10.1.3.1:

In all <catch> blocks there is a partnerlink to the FaultHandler, which will log the messages.

In case of remoteFaults, if the allowed retries are exceeded, the process will additionally write the instance information in a custom database table (SERVICE_EXCEPTIONS) and waits for a signal that the problem has been fixed. After having fixed the problem, the system administrator can enqueue a signal message into the SERVICE_EXCEPTIONS_Q, which will be picked up by an AQ Adapter , which is using native AQ Correlation. This signal comes through a <receive> activity from an AQ Adapter partner link. The message written in the SERVICE_EXCEPTIONS table will be deleted. The BPEL process will repeat the scope using the <replay> fault.

In this way the original message is available for resubmission within the process instance even if the process lifecycle might be *'inactive'*. This was important for inbound interactions with SAP, because it is not possible to notify SAP in case of delivery problems of an iDOC, that (from the SAP point of view is successfully delivered to the integration layer).

Following patch is required in order to implement this behavior.



[Bug.5567281](#)(74) PATCH01 WHEN PROCESS IS RETIRED, CANT CONTINUE DUE TO EP DEACTIVATION

The count of retries is configured as BPEL preference in the deployment descriptor of every interface. As mentioned, the <replay> fault is used to control the retry behavior.

Appendix- Description of sources

Directories:

ExceptionFilter: Java Code used to implement the error filter

FaultHandler: The BPEL Process for Fault Handling

prb_sap_leist: A sample interface using the fault handler

sql: Scripts to create Queues and tables for the signal.