Technical Note: LogicalApps Web Services

Introduction

ACTIVE Access Governor™ detects segregation-of-duties (SOD) conflicts within an organization, either preventing them from occurring or uncovering them so that they can be properly managed. Designed for use with Oracle Applications, Access Governor identifies conflicts at both the responsibility and function levels.

LogicalApps Web Services offers an API through which a user-provisioning system can communicate with Access Governor. As a user is assigned new Oracle responsibilities, the provisioning system uses API methods to request SOD evaluation; Access Governor generates information about conflicts the new assignment has generated, and the provisioning system uses API methods to receive that information.

Access Governor Overview

Users of Access Governor create segregation-of-duties rules, each of which specifies two or more Oracle responsibilities or functions that should not be assigned simultaneously to an individual person. Each SOD rule applies one of four “control types”:

- A Prevent rule denies access to conflicting responsibilities or functions.
- An Allow with Rules SOD rule permits access to conflicting responsibilities or functions if one or more additional rules, written in an application called Form Rules, mitigate the conflict by modifying Oracle forms. This control type (like the Prevent type) requires no approval.
- An Approve with Rules SOD rule also permits access to conflicting responsibilities or functions on condition that one or more Form Rules mitigate the conflict. In this case, however, the SOD rule designates an “owner,” and access to the conflicting entities is granted only if the owner approves.
- An Approval Required rule also designates an owner who can either approve a conflict or reject it. In this case, no Form Rule is attached to the SOD rule, and for access to be granted to conflicting responsibilities or functions, no condition need be met other than that the owner approve the conflict.

When SOD rules are satisfactorily configured, an Access Governor user “generates conflicts” — causes Access Governor to note Oracle ERP users whose work assignments violate the rules. If Oracle users have been assigned responsibilities or functions before SOD rules were in force to define them as conflicting, Access Governor records the conflicts, but administrators would need to resolve them manually within the Oracle ERP instance.

After conflicts have been generated, the SOD rules remain in force, to be used subsequently in uncovering conflicts as new Oracle users are created or existing users are assigned new responsibilities.

For more detailed information about Access Governor, see the *ACTIVE Access Governor User's Guide*.

Web Services Overview

Through the Web Services API, a client user-provisioning system can request that Access Governor perform "simulation" for an Oracle user. The API also enables the client to receive results from Access Governor.

In this context, "simulation" means that Access Governor evaluates SOD rules to determine which would be violated if the user were to be assigned one or more responsibilities. To do so, Access Governor would use the rules that were in force when conflicts were most recently generated.

A typical request includes the user name of an Oracle user whose account is being created or modified, the data source (or sources) in which the user account is to exist, and the responsibilities the user is being assigned.

A typical response constitutes a set of conflicts. Each includes information such as the name of the SOD rule that has been violated; the elements of the rule, including the conflicting functions, responsibilities, and applications; a status, which provides the control type of the rule that has been violated; the owner of the rule; the user whose responsibility assignment violated the rule; and more.

The Web Services API consists of nine classes, which offer methods that make the exchange of this information possible:

- SODWS is the interface to the web service.
- WebServiceUtil provides methods that help to retrieve the web service.
- SimulationRequest provides methods used to formulate a request by the provisioning system to have Access Governor run SOD analysis.
- OCSimulationRequest inherits methods from the SimulationRequest class and adds to them.
- SimulationResponsibilityEntry provides methods which, in a request, convey information about a responsibility that is to be assigned to a user.
- SimulationResult provides methods to convey conflict data from Access Governor back to the provisioning system.
- OCSimulationResult inherits methods from the SimulationResult class and adds to them.
- SimulationConflict provides methods which, in a result, convey information about SOD rule violations.
- SimulationException provides exception handling.

In addition to initiating requests and accepting results, the client user-provisioning system must be coded to handle the results it receives. When Access Governor returns a Prevent conflict, the provisioning system must be able to remove the responsibility assignment that generated the conflict. When Access Governor returns an Approval Required or Approve with Rules conflict, the provisioning system must be able to send notification to the owner of the SOD rule that detected the conflict, and handle the response: end-date the responsibility assignment if the owner rejects the conflict or remove the end date if the owner approves.

Moreover, a single responsibility assignment may (and often does) violate more than one SOD rule, and the client user-provisioning system must be coded to respond appropriately when an assignment violates rules of more than one control type. For example, your organization may determine that if a responsibility assignment triggers both Prevent and Approval Required SOD rules, the assignment should be prevented and so there is no need to send a notification for the Approval Required conflict. The conventional "pecking order" for control types, in descending priority, is Prevent, Approval Required, Approve with Rules, Allow with Rules, and no conflict, but you are free to determine and implement your own order.

It is, of course, assumed that Access Governor has been installed and used to create a set of SOD rules. When you use the Web Services API to integrate Access Governor with your user-provisioning system, you create SOD rules essentially as you would if you were using Access Governor on its own. See the *Active Access Governor User's Guide* for details.

There is, however, one significant exception: When Access Governor is used on its own, one can write an SOD rule that designates an approval group as the conflict reviewer. Moreover, one can select a workflow role or a responsibility as an SOD rule owner. (If, for example, a rule designates Purchasing Super User as its owner and does not specify an approval group, everyone assigned the Purchasing Super User responsibility would receive approval requests generated by that rule.)

The Web Services API does not accommodate these features. Its getApprover method returns the rule owner to your user-provisioning system, but it provides no method to return an approval group. Moreover, the owner must be a person, not a role or a responsibility.

Web Services Environment

LogicalApps provides an archive file, IDM_JAVA_API.zip, which contains a set of jar files that constitute the Web Services API. When you extract the contents of the archive, the jar files are located at IDM_JAVA_API/client/lib. These files need to be placed in the classpath of the runtime environment of your client.

The Web Services API uses Java version 1.4.2.

Web Services Documentation

Two documents tell you to use the Web Services API. One is JavaDocs, an on-line manual that offers detailed listing of the API classes and the methods they contain. The Web Services JavaDocs manual is also provided in the archive file. The path to the document is IDM_JAVA_API/javadoc/html. To open the document, navigate to that directory and select the index.html file.

This document serves as an overview of the Web Services API, explaining how it interacts with ACTIVE Access Governor and, in the next chapter, providing sample client code, with annotations.

A Sample Client

The following sample code represents the portion of a client user-provisioning system that would use Web Services API calls to request that ACTIVE Governance™ perform SOD analysis and to receive responses. You are assumed to be a Java programmer and to be able to read much of this code without commentary. Annotations are intended to explain data items required for SOD processing.

```
import com.logicalapps.onecenter.sod.ws.OCSimulationRequest;
import com.logicalapps.onecenter.sod.ws.OCSimulationResult;
import com.logicalapps.onecenter.sod.ws.SODWS;
import com.logicalapps.sod.SimulationConflict;
import com.logicalapps.sod.SimulationRequest;
import com.logicalapps.sod.SimulationResponsibilityEntry;
import com.logicalapps.sod.SimulationResult;
import com.logicalapps.webservice.WebServiceUtil;
import org.apache.wsif.WSIFService;
import java.util.*;
                                                                                  1
public class SODWebServiceTest_Sample_Client
{
    public static String location = "http://localhost:8080/ags/services/SODServiceProcessor";

    public static boolean testBasicConnectionAndSecurity()
    {
        System.out.println("Connecting to web services, please wait: " + location);
        try{
               WSIFService service = WebServiceUtil.getRawService(location + "?wsdl", null,
                "SODService", "username", "password", null, null);                       2
               //System.out.println("service "+service+" "+service.getStub(SODWS.class));
               return true;
        }catch(Exception ex){ex.printStackTrace(); return true;}
    }

    public static void testBasicService()
        throws Exception
    {
        // 1. Map the data structure so WSIF knows how to unwrap from SOAP

        Map map = new HashMap();
        map.put("OCSimulationRequest", OCSimulationRequest.class);
        map.put("OCSimulationResult", OCSimulationResult.class);
        map.put("SimulationRequest", SimulationRequest.class);
        map.put("SimulationResult", SimulationResult.class);
        map.put("SimulationResponsibilityEntry", SimulationResponsibilityEntry.class);
        map.put("SimulationConflict", SimulationConflict.class);
        map.put("String", String.class);                                                  2

        // 2. Locate the web service and tell WSIF to create a stub/proxy for it
        SODWS sodService = (SODWS) WebServiceUtil.getService("SODService", location, SODWS.class,
         "username", "password", "urn:SODWS", map);

        // 3. Call the web service and get the result
3       OCSimulationRequest request = new OCSimulationRequest();
        request.setUsername("EVALUATION_USER");
        request.addDatasourceName("CLIENT_ERP");
                                               4
```

5

```
request.addResponsibilityEntry("Payables", "Payables Manager", "Standard",
  Calendar.getInstance().getTime(), null);
 request.addResponsibilityEntry("Purchasing", "Purchasing Super User", "Standard",
  Calendar.getInstance().getTime(), null);
 request.addResponsibilityEntry("Application Object Library", "Application Developer",
  "Standard", Calendar.getInstance().getTime(), null);
 request.addResponsibilityEntry("System Administration", "System Administrator", "Standard",
  Calendar.getInstance().getTime(), null);

  OCSimulationResult[] results = null;
 OCSimulationResult result = null;

 try {
     System.out.println("calling SOD test simulation");
     results = sodService.simulate(request, true);
     result = results[0];
 } catch (Exception e) {
        System.out.println(e);
     e.printStackTrace();
 }
```

6

```
 Collection conflicts = result.getConflicts();
 System.out.println("conflicts size: "+conflicts.size());
 Iterator i = conflicts.iterator();

 while (i.hasNext()) {
     SimulationConflict con = (SimulationConflict) i.next();
     // remove variable data
     int requestId = con.getRequestId();
     int userId = con.getUserId();
     String userName = con.getUserName();
     con.setRequestId(-1);
     con.setUserId(-1);
     con.setUserName(null);

     System.out.println("======");
     System.out.println("verifying conflict:");
     System.out.println("requestId: " + con.getRequestId());
     System.out.println("userId: " + con.getUserId());
     System.out.println("userName: " + con.getUserName());
     System.out.println("conflictId: " + con.getConflictId());
     System.out.println("conflictName: " + con.getConflictName());
     System.out.println("approver: " + con.getApprover());
     System.out.println("applicationId: " + con.getApplicationId());
     System.out.println("applicationName: " + con.getApplicationName());
     System.out.println("responsibilityId: " + con.getResponsibilityId());
     System.out.println("responsibilityName: " + con.getResponsibilityName());
     System.out.println("conflictingApplicationId: " + con.getConflictingApplicationId());
     System.out.println("conflictingApplicationName: " + con.getConflictingApplicationName());
     System.out.println("conflictingResponsibilityId: " + con.getConflictingResponsibilityId());
     System.out.println("conflictingResponsibilityName: " + con.getConflictingResponsibilityName());
     System.out.println("functionId: " + con.getFunctionId());
     System.out.println("functionName: " + con.getFunctionName());
     System.out.println("conflictingFunctionId: " + con.getConflictingFunctionId());
     System.out.println("conflictingFunctionName: " + con.getConflictingFunctionName());
     System.out.println("actionType: " + con.getActionType());
     System.out.println("status: " + con.getStatus());
     System.out.println("sameSOB: " + con.getSameSOB());
     System.out.println("sameOU: " + con.getSameOU());
     System.out.println("==================================================================");

     con.setRequestId(requestId);
     con.setUserId(userId);
     con.setUserName(userName);
 }
```

7

```
        for (int ii=0; ii < results.length; ii++) {
            results[ii].setUserName("EVALUATION_USER");
            sodService.createSODAction(results[ii], false);
        }
    }

    public static void main(String[] args)throws Exception
    {
       if(testBasicConnectionAndSecurity())
       {
              testBasicService();
       }
       else
       {
              System.out.println("Error in invoking SOD Webservice");
       }
    }
}
```

Within this code, the lines marked by numbers require the following explanations:

1 This line establishes the location at which SOD services are provided. Access Governor runs on an ACTIVE Governance Platform, and in this line you would replace the word *localhost* with the name of the host system on which the Platform runs, and the number *8080* with the port dedicated to the application server that supports the Platform. Other elements of this line would be entered exactly as they are shown.

2 Two lines are identified by the number *2*. Each is involved in establishing the web service, and in each the word *username* must be replaced by the name of a user configured on the ACTIVE Governance Platform, and the word *password* must be replaced by the password configured for that user.

3 This block of the code calls the web service to request that SOD processing be performed for a user who is being assigned Oracle responsibilities. The phrase *EVALUATION_USER* must be replaced by the username of the Oracle user who is being assigned responsibilities. This would typically be a parameterized value passed to the code. (Note that this value also appears, and would also need to be replaced, in another line near the end of this code sample.)

4 This line specifies the Oracle instance on which the user is being assigned responsibilities, and the phrase *CLIENT_ERP* must be replaced by the name of the instance (again, a parameterized value passed to the code). There may be more than one of these lines, specifying more than one Oracle instance.

5 These lines specify Oracle responsibilities being assigned to the user; as parameters, each takes the name of an application to which the responsibility belongs, the responsibility itself, the Oracle security group, and the start and end dates for the responsibility assignment. Of course, the values shown are examples and would be replaced by values passed to the code.

6 The line that begins "OCSimulationResult[]" specifies an array of objects; each object is of type OCSimulationResult. Each of these objects consists of SOD analysis results for each Oracle ERP instance. The length of this OCSimulationResult array always equals the number of Oracle ERP instances added to the OCSimulationRequst object.

7 These lines use methods from the SimulationConflict class to provide information about conflicts that Access Governor has detected in the Oracle user's responsibility assignments. Among them, getConflictName provides the name of the SOD rule that has been violated, getApprover provides the name of the rule owner (and not, as discussed earlier, the name of an approval group), getStatus provides the name of the control type configured for the rule, and getActionType provides an index number corresponding to the control type: *1* represents Approval Required, *2* represents Prevent, *3* represents Allow with Rules, and *5* represents Approve with Rules (*4* is not used).