Keeping Access Control while migrating to the Cloud

Zdenek Nejedly  
*Computing & Communications Services (CCS), University of Guelph*

Abstract

How do we control access to our apps once they are in the cloud? How do we keep up with the evolution of Identity products? And how can we do it better next time in such a highly heterogeneous environment? The University of Guelph has many of its local and cloud IT services controlled by the central Web Access Management solution. It was the Sun Access Manager in 2008, which was replaced by the Oracle Access Manager 11g in between 2011 and 2012, and now OAM 11g R2. Come to learn the patterns for bringing web applications into Access Management, increasing security, availability, and overall value for the enterprise.

Table of Contents

1. About Web Access Management and our goal ............................................................... 1  
2. Case study - Web Access Management at the University of Guelph.............................. 2  
3. Save resources: centralize, delegate, and verify................................................................ 4  
4. Automation: extend the functionality with the OAM Policy API .................................... 6  
5. Cloud or on-premise: exercise the access control .......................................................... 6  
6. Learn from the transitions: making the product changes transparent to the applications ... 7  
7. Lessons learned from the upgrade to OAM 11g R2...................................................... 8  
Conclusions .......................................................................................................................... 9  
Acknowledgements ............................................................................................................ 9  
About the Author ................................................................................................................ 9  
Appendix 1: Creating a simple audit report for the OAM Administration Console with BI Publisher....... 10  
Appendix 2: Creating a REST client for identification of websites with access to specific personal data .. 14

1. About Web Access Management and our goal

The Web Access Management (WAM) as part of the Identity and Access Management (IAM) or Identity Management (a category used by the Oracle Fusion Middleware) provides several functions including authentication (may implement federated protocols like SAML or social identities with OpenID), authorization, session management (Single Sign On), management of identity attributes (user’s personal data), and auditing of access to resources. The Oracle Access Manager 11g Release 2 [1,2] supports these five functions as well as additional ones, e.g., password management.

The implementation options for the Oracle Access Manager (and other IAM products) range from the deployment and support in-house by an in-house team to moving to the cloud service managed by another party. The benefits of each approach depend on multiple factors, for example, the timing requirements, availability and cost of local and external resources, cost of ongoing integration and customization, legislative requirements.
The goal of this presentation is to provide you with tips and share the lessons learned to increase the value of your investment in OAM and make your role as a Web Access Management practitioner easier whether you are, for example, a developer, OAM administrator, IAM architect, or a strategist. Some of these tips may be applicable in any implementation approach, e.g., the automation of the Oracle Access Manager (OAM) policy management with a RESTful client, and some may apply in the specific case when the OAM is deployed on-premise and supported by the local team as is the case at the University of Guelph.

2. Case study - Web Access Management at the University of Guelph

The University of Guelph is a midsize comprehensive Canadian university with about 28,200 undergraduate and graduate students [3]. The University is "dedicated to cultivating the essentials for quality of life - water, food, environment, animal and human health, community, commerce, culture and learning" [3].

During the past 8 years of the WAM initiative at the University of Guelph we experienced three major transitions (Figure 1). First we deployed the Sun Microsystems Sun Access Manager 7.1 as part of the Sun Java System in 2007/2008 and started bringing the campus IT services from application-managed authentication and authorization to the central WAM. In 2010, Oracle Corporation acquired Sun Microsystems and the Oracle Access Manager became Oracle’s strategic WAM product. Given these market conditions and the existing investment in the (former Sun) Directory Services Enterprise Edition, the University of Guelph transitioned from SAM to OAM 11g (11.1.1.5) in 2011/2012. Finally, we upgraded to the latest version of OAM (11g Release 2 Patch Set 2) in 2014/2015. WAM at Guelph is a business critical service protecting major enterprise services, departmental websites, and cloud applications.

Figure 1: Transitions from Sun Access Manager to Oracle Access Manager 11g and finally to OAM 11g Release 2
At the high level, the Oracle Access Manager consists of the centrally deployed access servers that implement, for example, administration, session, and authorization policy engine and the webgates (Policy Enforcement Points) which are deployed on the web servers they protect. The University of Guelph central OAM infrastructure consists of the standalone development OAM, a three-node test OAM cluster, and a three-node production OAM cluster plus temporary OAM deployments as needed (Figure 2). Integration of OAM webgates with campus services is prototyped and validated on the dedicated client development environment with Linux RedHat and Windows Servers matching the deployment platforms and webserver type of our OAM clients. The development and test servers are virtualized (VMware) while the production OAM cluster (both application and database tiers) is deployed on physical Dell hardware. Currently the OAM technology stack includes the Oracle HTTP Server, WebLogic Server 10.3.6, OAM 11gR2PS2, and Oracle Database Server 11gR2 Standard Edition; all running on RedHat Linux. The OAM cluster is load balanced by the Cisco hardware load balancer (LBR) where the https is also terminated.

![Diagram of OAM Virtual environments for development and test](image)

**Figure 2:** The University of Guelph central OAM infrastructure - development, test, and production

The production OAM cluster serves about 40 OAM application domains with 60 OAM webgates deployed on various editions of Apache httpd server, Oracle HTTP Server, and Microsoft IIS. The number of concurrent and unique user SSO sessions reaches about 25,000 daily.

The LBR production configuration for OAM contains two server farms for the OAM authentication web requests (plus other server farms for the administration console, etc.): the primary server farm listening on a DNS-
advertised IP address and its clone listening on an IP which is not advertised in DNS. The secondary/cloned LBR server farm (Figure 2) is particularly valuable when the production OAM cluster is being patched or reconfigured. When the patched production cluster node needs to be individually validated it is first brought up in the secondary LBR server farm, validated without any public web traffic (Coherence and WebLogic cluster communication is not impacted), and then activated in the primary LBR server farm (Figure 3).

Figure 3: Validating the OAM cluster with a secondary LBR server farm - the first two steps of the cycle

3. Save resources: centralize, delegate, and verify

OAM webgate installed on a reverse proxy is one of the popular WAM integration patterns. The reverse proxy becomes necessary in cases when the service to be protected does not support direct installation of the webgate. Examples of such cases at the University of Guelph included the use of reverse proxy in front of the Oracle HTTP Server (OHS) when the Sun Access Manager did not yet support OHS, or a reverse proxy in front of Windows Server 2008 R2 when the Oracle Access Manager did not yet support IIS 7.5. A current example of use of a reverse proxy with an OAM webgate at Guelph is to provide authentication and authorization for campus users for a cloud services hosted outside of the campus by an independent service provider who does not support other means of integration, e.g., federated SSO.

The second goal of deployment of the OAM webgate on the reverse proxy is to minimize the effort of the OAM team to expand and support the OAM service - the webgate installation can be shared between many websites behind the proxy thus lowering the support costs. The University of Guelph provides centrally managed web hosting to its departments behind a load-balanced cluster of reverse proxies with apache webserver (Figure 4). Addition of OAM to this centralized environment allowed us to shorten WAM intake of an application from hours otherwise required to schedule, optionally prototype, install, configure, and test a new webgate to minutes required to define the access policies. Given the new option of delegation introduced in OAM 11g Release 2, we can delegate the web access policy management to trusted partners or teams working closely with the clients thus decreasing the implementation time and resource costs further and improving the OAM partner’s experience.

Delegation of OAM administration recognizes two types of administrators: a) System Administrators and b) Application Administrators with editing privileges to policies and resources within the delegated domains. The Application Administrators cannot view or modify OAM system properties and have read-only access to shared objects like authentication schemes, host identifiers, and resource types. In terms of the application domain objects (resources, authentication and authorization policies), the delegation provides guidance rather than sand
boxing or security. An Application Administrator assigned to one application domain can create access policies which may have security or privacy implications in another domain. Therefore, auditing of the administration tasks and diagnostic tools verifying the implemented OAM access policies against the required protocol are essential.

OAM supports extensive auditing of run-time and administration events. It also includes a restricted license for the Oracle Business Intelligence Publisher (BIP) [4] which can be used to design and run audit reports. Several such reports are provided along with the OAM by Oracle. In this presentation we illustrate how to create a simple custom BIP report capturing Administration Console events (Appendix 1): the report queries the OAM audit database and extracts the administration tasks, username and IP address of the OAM administrator executing the task as well as the original and new OAM parameters. When the audit database is managed in a way to retain the administration events, this custom report provides a sequential record of OAM administration tasks executed via the console, REST policy API, or the WebLogic Scripting Tool (WLST).

Figure 4: University of Guelph central reverse proxy cluster with OAM

While the audit reports provide historical view of administration activity, as the number of administered access point increases a view of effective policy implications is required. Here is an example of such use case specifically in privacy: identify what OAM application domains (which may denote clusters of web content like web sites or web applications) have access to specific personal attributes (user profile data). The next section describes a solution applicable to this use case.
4. Automation: extend the functionality with the OAM Policy API

With increasing deployment of OAM webgates within the organization the service and integration management requirements raise quickly and the limited resources require increased automation. In addition to native OAM tools, the University of Guelph uses 3-rd party tools for monitoring: Dell (formerly Quest Software) Foglight Performance Monitoring for the Oracle Database [5] and Nagios [6] for infrastructure monitoring. Beside the standard Nagios server or network tests, the service availability monitoring covers the full SSO lifecycle: OAM login, protected application test, OAM logout - with pattern assertions to diagnose potential issues at the specific stage, e.g., the load balancer, web tier, WebLogic server with OAM, the database tier, server and network infrastructure or the protected service itself.

The OAM REST API for access policies management is one of the new options in OAM 11g Release 2 [7]. It supports creation, modification, removal and querying of OAM policy configuration and provides opportunities for automation. An example use case related to privacy is described in Appendix 2: the business goal is to determine which OAM-protected web sites (URLs) have access to particular personal data, e.g., a directory / LDAP attribute like a student number. The RESTful client that has been designed to provide this functionality searches the OAM policies with the REST API, detects the authorization policies which utilize the given attribute, looks up the related resources, and lists the applicable Application Domains and URLs that identify the services which have access to the private data. The client is implemented in java with web services framework Jersey [8] and also utilizes classes generated with XML binding compiler [9] - please see Appendix 2 for detailed instructions.

5. Cloud or on-premise: exercise the access control

The Access Management tier facilitates the first contact with the web application from the user’s perspective (authentication) as well as the continuing interaction, e.g., access authorization. These aspects give WAM a unique ability to participate in routing of the user’s web requests.

As an example, when the University of Guelph decided to move its student email and calendaring to the cloud (Google Apps for Education - GAFE - [10]) while providing similar services to University employees locally with Zimbra Collaboration Suite [11] it faced a challenge how to make the access to these two services transparent to the University community. In Higher Education, the IT users often have multiple and sometimes seemingly incompatible roles, e.g., being a student and an employee at the same time, and these roles change.

The solution was a custom middleware module which routes the user to the given email system based on their roles. The middleware is protected by the Oracle Access Manager which supplies the user attributes for the routing decision engine. Additionally the middleware poses as a SAML2 Identity Provider to GAFE and thus can provide

Figure 5: Steps to create a REST client with OAM Policy API

Figure 6: Customized request flow with the middleware
custom responses to University users who, for example, try to access the University instance of GAFE but are not eligible for GAFE and, therefore, not provisioned there. In the reference implementation of the GAFE Single Sign On [12], GAFE redirects the user directly to the SAML Identity Provider which authenticates the users and returns the username back to GAFE as a SAML artifact via the browser redirection. In our customized flow (Figure 6) utilized in this use case, GAFE is configured to redirect the user to the middleware instead and the middleware decides whether the user is to be directed either to the IdP and then to GAFE or to the on-premise instance of Zimbra.

6. Learn from the transitions: making the product changes transparent to the applications

Some WAM functions, e.g. authentication or authorization, are transparent to the protected services but the attribute delivery function creates a data format dependency. And if the format or data mapping evolves and changes then the dependent applications may need to change, too.

Some datasources like LDAP support multivalued attributes and the WAM products may map these multivalued attributes to a single value (to be delivered in a request header) with the original multiple values delimited by a separator. There is no standard delimiter supported by the WAM products. The Sun Access Manager used the vertical bar a.k.a. pipe '|' as the multiple-value delimiter. Comma was the separator in the Oracle Access Manager 11g. The comma is frequently used in LDAP values, for example, in distinguished names which can also appear in multivalued attributes and parsing of such values becomes non-trivial. And finally, the colon became the default separator in OAM 11gR2 and escaping of the separator character within the OAM responses was added. OAM 11gR2 also supports redefinition of the separator and the escape character.

Another changing aspect among different products or versions of the same product that we experienced was the management of empty values. While the SAM would return an empty string, the OAM 11g delivered NOT_FOUND constant which changed in OAM 11g Release 2 to NULL or NOT_FOUND depending on whether the attribute was null or not defined.

A change of the separator or the way the empty values are transformed during the transition between two WAM products or versions means that the protected applications receiving the attributes need to be updated unless there is an interface between the WAM product and the application. Such update of applications across the OAM deployment particularly the legacy ones or 3rd party can become resource intensive and costly.

When we deployed our first WAM product - the Sun Access Manager - we started distributing simple but consistent 'WAM toolkits' to our Access Management clients and partners. The main business goal was to simplify the identity aspect of web application development and make the transition from application-managed authentication and authorization to central Access Management as easy as possible. The toolkits provided a unified interface with convenient function calls in php, java, ColdFusion, and PL/SQL (for APEX) to retrieve the user profile data and it also kept the delivery implementation (HTTP headers or session variables, etc) transparent to the applications (Figure 7). For example, a php developer would retrieve the user's display name (LDAP attribute) with the following snippet:

```
app

WAM toolkit

request
object

isStudent()

GetHTTPRequestData()

parse()

lastName=Smith
firstName=Jane
coopNmber=NULL
affiliation=member:student

transform()

Figure 7: Retrieving the user's profile data - the application calls the toolkit's function isStudent(), which in turn retrieves the raw HTTP headers from the request object, parses and transforms them.
```
Hello <?php echo WAMGetDisplayName(); ?>

while a ColdFusion developer would code:

Hello #WAMGetDisplayName()#

If for any reason the delivery parameters of the attributes to the application had to change then only the interface - the WAM toolkits - needed to be updated not the applications. The WAM toolkits have proven to be essential in our transitions from Sun to Oracle Access Manager and later to OAM 11gR2.

The WAM toolkits/interfaces worked well for in-house designed applications. They may not be an option for off-the-shelf or 3rd-party custom services which cannot be customized to include the toolkits. In those cases, the OAM webgate on a reverse proxy inserted in front of the protected service can serve as the translation /corrective layer. For example, the apache httpd server supports real-time rewrite of the HTTP headers with the mod_headers module (Figure 8). As an example, the following pair of mod_headers directives placed in the virtual host containing the webgate would replace a colon with a comma in CA-UOGUELPH-EP-AFF header and then substitute NOT_FOUND for NULL in the CA-UOGUELPH-COOP header:

RequestHeader edit CA-UOGUELPH-EP : ,
RequestHeader edit CA-UOGUELPH-COOP NULL NOT_FOUND

7. Lessons learned from the upgrade to OAM 11g R2

The following approaches helped us to minimize the impact of the 11g R2 upgrade on our client services:

i) Prototyping of the upgrade steps in virtualized environment where we could quickly roll back or forward or select a new path/branch in upgrade while we were working with My Oracle Support. Our production servers are deployed on physical hardware for supportability and to minimize dependencies.

ii) Automating the upgrade steps, e.g., scripting of the one-off patch application after the upgrade of the IAM Suite binaries.

iii) Temporary pre-production runs of the enterprise services with the new OAM 11g R2 - the webgate's profile would be temporarily repointed to the upgraded test OAM 11gR2 cluster and then returned to the OAM 11g installation. We could not pre-test all client services and applications so we selected representative services, specifically:

   a) core applications like enterprise email, learning management system, or federated SSO /Shibboleth
   b) representatives of different platforms (apache httpd on Linux, different versions of IIS on Windows Server, OHS on Linux)
   c) third party applications that we cannot modify and which could thus force us to roll back the production upgrade if there was an issue
   d) applications/services with complex policies (conditions and/or responses)
iv) Retesting the upgrade process over and over to cover the various challenges, to identify the workarounds and patches needed in our environment. Some of these requirements or workarounds are described in the OAM Documentation Library [2], some available as Knowledge Base articles and some are being discussed in My Oracle Support Community. The list included, for example,
- Removing all base patches from WebLogic prior to the OAM upgrade and re-applying after the upgrade
- WebLogic Java policy workaround (Doc ID 1572620.1)
- Simple mode bug with java 6u28 and later (Doc ID 1513143.1)
- Workarounds for the new (in 11gr2) credential timeout (Doc ID 1579145.1) and login page idle timeout (Doc ID 1908294.1) - we found that some devices, e.g., Windows Phone 8.1 did not support the redirect pragma (as provided in the KB article) so we added also a javascript redirect.
- Patching Oracle Coherence from the default 3.7.1.1 to 3.7.1.13 to resolve Coherence failures resulting in SSO session creation issues and, eventually, causing the OAM to be unresponsive.

Conclusions

In this document, we have described some of our experiences with deployment, management, and integration of the Oracle Access Manager at the University of Guelph. The new option in OAM 11gR2 - delegation of policy administration - is beneficial to our central web hosting with the reverse proxy farm. Auditing of administration events and the REST policy API (another new option in OAM 11gR2) facilitate two different views of the access policies and are desirable when delegating the OAM policy administration. We have demonstrated how JAX-RS via Jersey simplifies creation of a RESTful client for management of OAM access policies and, also, how to design a simple report with the BI Publisher to track OAM administration changes. A middleware integrated with the OAM enabled us to facilitate routing to two components (cloud and on-premise) of an enterprise IT service. When we transitioned between three Access Management products we and our campus partners benefitted from a common application interface to user attributes. This interface (a toolkit) was convenient for the application developers and made the changes in the delivery of attributes between the WAM products transparent to the applications. Finally, we are providing several tips from our recent upgrade of OAM 11g to 11g Release 2.

Acknowledgements

This ongoing initiative - bringing OAM to the University of Guelph and expanding the community of WAM/SSO partners - requires effort of multiple teams from our central University of Guelph IT organization (CCS) while prototyping, deploying, and supporting the central OAM infrastructure, as well as support of our campus partners during integration with their web applications and services, technical assistance of My Oracle Support as well as feedback from the My Oracle Support Community.

In particular I am indebted to Bosco, Rob G., Paul H., Tim, Tony, Leo, Dennis and their CCS teammates for managing the servers, and assisting us with the integration with the University webhosting while implementing sometimes peculiar network and load balancer configurations for our OAM, Rob F. and Paul K. for assistance with the Zimbra integration, Jill and her Help Desk and Help Centre teams for helping us track and resolve run-time issues, my IAM team colleagues Hugh and Matt for their patience and support and to Matt for supporting Shibboleth, and finally to Kent and Brian for their managerial insights.

About the Author

Zdenek is an IT analyst at the University of Guelph and the architect, project manager, and subject matter expert for the Oracle Access Management initiative at Guelph. Zdenek holds the OCP DBA, OCA WebLogic Administrator, OCA PL/SQL Developer, and TOGAF 9 certifications [24].
Appendix 1: Creating a simple audit report for the OAM Administration Console with BI Publisher

Our business goal is to identify policy and system changes carried out by the OAM administrators either by the OAM console or via the OAM WebLogic Scripting Tool or the OAM Policy API. We will utilize the Oracle Access Manager support for auditing [13] and then create a sample report in the Oracle Business Intelligence Publisher (BIP) customized to our goal. The OAM already includes several other BIP reports out-of-the-box.

Step 1: Configure OAM auditing to a database store

a) Enable OAM auditing [14] - by default the audit events will be logged to a file on the Administration server and the OAM managed servers.
b) Create the audit database [15]
c) Add the audit schemas with the Repository Creation Utility [14]
d) Configure a corresponding datasource in WebLogic and enable logging for OAM components in the OAM Enterprise manager [16]. You may need to restart the OAM cluster to activate the database logging.
e) Regarding the database maintenance, at the minimum the old events may need to be periodically removed. Oracle provides several maintenance scripts to purge the data as well tuning guidance [17].

At this point the audit events from the entire OAM cluster will be logged in the database and can be conveniently via SQL. For example, the following SQL (for Oracle Database Server) will output the Administrative events logged in the last 24 hours (only a subset of available fields is used as in Figure 9):

```sql
set lines 300
set pages 20
col IAU_ID format 999,999,999
col TS format a30
col IAU_EVENTTYPE format a22
col IAU_AUDITUSER format a14
col IAU_INITIATOR format a30
col IAU_OLDATTRIBUTES format a40
col IAU_NEWATTRIBUTES format a40
SELECT a.IAU_ID, TO_CHAR(a.IAU_TSTZORIGINATING,'YYYY-MM-DD HH24:MI:SS.FF3') TS,
a.IAU_EVENTTYPE, b.IAU_AUDITUSER, b.IAU_INITIATOR, b.IAU_REMOTEIP,
a.IAU_OLDATTRIBUTES, a.IAU_NEWATTRIBUTES
FROM OAM a INNER JOIN  IAU_BASE b ON a.IAU_ID = b.IAU_ID WHERE
a.IAU_EVENTCATEGORY='AdminConsole' AND a.IAU_TSTZORIGINATING>SYSDATE-1 ORDER BY 2;
```

In the next several steps we will create a custom BIP report to provide a more effective way to visualize the audit data. The report will allow us to filter the data by username and the type of the administrative event and select a particular date window.

Step 2: Install and patch the Oracle Business Intelligence Publisher

The BIP report below has been designed and tested with BI Publisher 11.1.1.7 patched with the January 2015 patch to 11.1.1.7.150120. The Oracle installation documentation is available at [18].

Step 3: Create the Data Model

The BI Publisher Data Model defines the data source, structure, and relationships of the data, plus additional characteristics like parameters and lists of values used to filter the data - more details are available at [19].
Figure 9: Database entity diagram denoting the subset of columns used in the Data model

a) Add the data source (the new audit database) in the Administration>JDBC Connection section. You can use the prefix_IAU_VIEWER database user created by the RCU in Step (1c) above. Let's assume that the 'prefix' is 'tst2'
b) Create a new Data Model via Home>Create>Data Model.
c) Set the properties: in the left components pane, click on the root element "Data Model", set the Description and select the audit database. Click the save icon and define the data model name, e.g., AdminConsoleDataModel.
d) Create the data set: in the left components pane, click on the element "Data Sets", then New Data Set under the Diagram tab and select SQL Query.
e) In the SQL Query editor set the Name property, verify the Data source and paste the following in the SQL query field and then press OK:

```
SELECT a.IAU_ID, a.IAU_TSTZORIGINATING, a.IAU_EVENTTYPE, b.IAU_AUDITUSER, b.IAU_INITIATOR, b.IAU_REMOTEIP, a.IAU_OLDATTRIBUTES, a.IAU_NEWATTRIBUTES
FROM tst2_iau.oam A INNER JOIN tst2_iau.IAU_BASE b ON a.IAU_ID = b.IAU_ID
WHERE a.IAU_EVENTCATEGORY='AdminConsole' AND b.IAU_AUDITUSER LIKE nvl(:p_username,b.IAU_AUDITUSER)
AND a.iau_tstzoriginating > :p_startDate
AND a.iau_tstzoriginating < :p_endDate
AND a.IAU_EVENTTYPE IN (:p_eventType)
ORDER BY a.IAU_TSTZORIGINATING DESC
```
f) Confirm creation of the four parameters (p_username, p_startDate, p_endDate, p_eventType which match the four bind variables we have defined in the SQL statement in the new data set.
g) Set parameter properties: in the parameters window:
   i) move the p_username to the top and set the Display Label to Username
   ii) move p_startDate to the top second position, set the Data Type to Date, Display Label to Start Date, Text Field size to 10, Date Format String to yyyy-MM-dd and the default value to {$SYSDATE()-7$}
   iii) move p_endDate to the top second position, set the Data Type to Date, Display Label to End Date, Text Field size to 10, Date Format String to yyyy-MM-dd and the default value to {$SYSDATE()+1$}
   iv) leave the p_eventType as is for now and save the Data Model again
h) Define the list of possible event type values: in the left component pane select List of Values, create a new list of values, name it ListofEventTypes, and paste the following query in the SQL Query window:

```
SELECT DISTINCT(IAU_EVENTTYPE) FROM tst2_iau.oam WHERE IAU_EVENTCATEGORY='AdminConsole' ORDER BY 1
```
i) Define a pull-down menu for the p_eventType parameter: click on the p_eventType parameter in the component pane, set the Display Lable to Admin Event, Parameter Type to Menu and confirm the List of Values points to our
ListOfEventTypes, check Multiple Selection, and Can Select All with All values passed option. The parameter screen is captured in Figure 10.

![Figure 10: Creating the BIP Data Model](image)

i) Finally, save the data model and create a sample data set: proceed to the Data tab, click on View and then Save as Sample Data - this data sample will be used in the next step to create the layout.

**Step 4: Create the Layout**

The Layout defines the presentation of the data from the Data Model. In this step we will create the layout by generating it automatically. Details on design and customization of the layouts are available in the Oracle documentation at [20].

a) Create a new layout: click on the Create Report icon, then select Use Data Model, confirm our data model (AdminConsoleDataModel.xdm) is selected along with the ‘Guide Me’ option and click Next.
b) Select Landscape as the Page Option, confirm the Table Layout, and select Next.
c) Drag all column fields to the right window to create the table. Preview the Report to see the data in a table.
d) Click Next, finish the report, and save it as AdminConsoleReport.
e) Open the report, set the filtering parameters, for example the date window, username, or subset of administrative events to be listed, and choose the type, e.g., interactive or PDF (Figure 11).
Figure 11: Generated report layout for Administration Console
Appendix 2: Creating a REST client for identification of websites with access to specific personal data

Our business goal is to list Application Domains and specific URLs (thus web-based services) which have access to the selected user personal data (as defined in OAM for example based on a directory attribute). The design and deployment of the REST client described here requires knowledge of how to code and run a simple java application with the main() method. The sample code can be downloaded from [21]. Further details on how to create RESTful clients or servers with Jersey are available, for example, in [22].

Step 1: Prerequisites - set up your environment

a) Set up a java development environment of your choice.

b) Download the dependencies - depending on your preferred java IDE these dependencies may be already present. The RESTful client requires:

- Jersey [8]: implementation of Java API for RESTful services (JAX-RS) - a bundle with all 3rd party dependencies is provided on the project download page
- Project JAXB [9]: implementation of Java Architecture for XML Binding (JAXB)

c) Bring over the XML schema for generation of client-side java classes representing the OAM policy objects. You can find in the deployment subfolders of the OAM managed servers and the AdminServer. For example, you can use the following Linux command to locate it on your OAM Weblogic servers (the $OAM_DOMAIN denotes the root folder of the OAM WebLogic domain):

```
find $OAM_DOMAIN -name oam-policyadmin-11.1.2.0.0.xsd
```

d) Make sure you have network access to your development OAM server (including the OAM administration credentials)

The RESTful client described here has been created with Netbeans 8 [23], Jersey 2.16, JAXB 2.2.11, and Oracle Access Manager 11.1.2.2.4.

Step 2: Create the client-side java classes describing the RESTful service object

a) Generate the POJOs (plain old java object) with JAXB from the OAM provided XML schema. You can find the xjc compiler script in jaxb-ri subfolder in the unzipped JAXB download. In your java development environment run:

```
xjc -d sourceFolder -p packageName oam-policyadmin-11.1.2.0.0.xsd
```

This will create java classes representing the policy resources used by the OAM REST API.

b) As a workaround, add the XmlRootElement annotation to each generated class you are going to use for XML-java binding, for example, add:

```
@XmlRootElement(name = "ApplicationDomain")
```

to ApplicationDomain.java directly above the first @XmlAccessorType annotation. Remember to adjust the name attribute to the name of the class. Our RESTful client will need this workaround implemented for ApplicationDomain.java, ApplicationDomains.java, HostIdentifier.java, and HostIdentifiers.java

c) Define a separate Resources class to substitute in JAXB for the existing inner class ApplicationDomain.Resources - please see Resources.java in the provided source code.
Step 3: Create a REST client to list the OAM Host Identifiers

JAX-RS and its implementation Jersey simplify the process of creation of a RESTful client (Figure 12). We will start with a simple REST client which requests a list of all Host Identifiers from the OAM server. In the following step we will extend this client to a full application implementing our business goal: identifying the application domain which have access to a specific user data.

First, we instantiate the client using the javax.ws.rs.client.ClientBuilder factory - please see line (1) in Figure 13, set the Basic authentication which the OAM Policy API uses it to identify the client (2), and obtain a WebTarget object for the base service URI (3) - this object can be later reused for specific REST resources.

The common service URI is
protocol://oam_host:port/oam/services/rest/11.1.2.0.0/ssa/policyadmin/
where protocol, oam_host, port depend on your OAM deployment.

```java
package oamclient;
import oamclient.pojo.Host;
import oamclient.pojo.HostIdentifier;
import oamclient.pojo.HostIdentifiers;
import java.util.List;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import org.glassfish.jersey.client.authentication.HttpAuthenticationFeature;
public class RestClient {
    private static final String COMMON_URI =
        "http://adm.identity.uoguelph.ca:7001/oam/services/rest/11.1.2.0.0/ssa/policyadmin/";

    public static void main(String[] args) {
        String username = "weblogic";
        String password = "passwd";
        WebTarget target = ClientBuilder.newClient()
            .register(HttpAuthenticationFeature.basic(username, password))
            .target(COMMON_URI);
```

**Figure 13**: Simple REST client - part 1 of 2 (using java method chaining)
Then (Figure 14), we create the WebTarget for the specific resource - in our case the resource URI is COMMON_URI/hostidentifier (4), set the media type to XML (5), set the class type of the object that will receive the data (6) - this is the class where the XML data will be translated into (un-marshalled), and retrieve the java List of Host Identifiers (7). Finally, we will output the name of the Host Identifier and all the included host names and ports (8) as intermediate diagnostic.

```java
List<HostIdentifier> lhi = target.path("hostidentifier")
    .request(javax.ws.rs.core.MediaType.APPLICATION_XML)
    .get(HostIdentifiers.class)
    .getHostIdentifier();

for (HostIdentifier hi : lhi) {
    System.out.println("Host Identifier: " + hi.getName());
    for (Host h : hi.getHosts().getHost()) {
        System.out.println("\t" + h.getHostName() + " : " + h.getPort());
    }
} // end of the main method
} // end of the class
```

Figure 14: Simple REST client - part 2 of 2

And now it is time to run our less-than-one-page-of-code client against OAM - Figure 15 shows part of the output.

```none
Host Identifier: IAMSuiteAgent
    IAMSuiteAgent : 80
    IAMSuiteAgent : null
    adm.identity.uoguelph.ca : 443

Host Identifier: IAMwebgateN3
    IAMwebgateN3 : null

Host Identifier: bobo.cs.uoguelph.ca_iis
    bobo.cs.uoguelph.ca : 443
    bobo.cs.uoguelph.ca_iis : null
    bobo.cs.uoguelph.ca : 8080
```

Figure 15: Sample output from the REST client
Step 4: Extend the client to lookup the URLs that receive the specific user data

Now we will add several REST calls to scan through the Application Domains in order to find all URLs that receive specific user data. In this case we are assuming the user data are defined only in the authorization policies. In general, they can be defined in authentication policies and passed to the user after login. Figure 16 shows the high level description of the process customized for the OAM policy object design.

<table>
<thead>
<tr>
<th>Read the user's attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request all host identifiers from the REST server</td>
</tr>
<tr>
<td>Request the identifiers of all application domains</td>
</tr>
<tr>
<td>FOR all application domains</td>
</tr>
<tr>
<td>Request the child objects of this domain</td>
</tr>
<tr>
<td>FOR all authorization policies in the given domain</td>
</tr>
<tr>
<td>FOR all policy responses in the given authorization policy</td>
</tr>
<tr>
<td>If the policy response contains the user attribute then</td>
</tr>
<tr>
<td>FOR all resource identifiers in this policy</td>
</tr>
<tr>
<td>If not cached then</td>
</tr>
<tr>
<td>Request the full definition of all resources in this domain</td>
</tr>
<tr>
<td>Lookup the resource path based on the resource identifier</td>
</tr>
<tr>
<td>Lookup the hostname based on the host identifier</td>
</tr>
<tr>
<td>Assemble the full resource URL from the hostname and path</td>
</tr>
<tr>
<td>Print the URL</td>
</tr>
</tbody>
</table>

**Figure 16:** Pseudocode for the full RESTful client: identifying the URLs of web resources which have access to the user's attribute

We will use three new REST calls:

1) lookup of all application domains (excluding the child objects)

2) lookup of detailed description of all URL resources within the given application domain

3) lookup of the specific application domain with the child objects

The first REST call (Figure 17) is similar to the lookup of host identifiers.

```java
List<ApplicationDomain> lapp = target.path("appdomain")
   .request(javax.ws.rs.core.MediaType.APPLICATION_XML)
   .get(ApplicationDomains.class)
   .getApplicationDomain();
```

**Figure 17:** Lookup of all application domains (without child objects)
The second REST call requires the query parameter `appdomainid` to specify the application domain - see line (1) below (Figure 18).

```java
public static List<Resource> getResources(WebTarget target, String appDomainId) {
    Resources rs = target.path("resource")
        .queryParam("appdomainid", appDomainId) // (1)
        .request(javax.ws.rs.core.MediaType.APPLICATION_XML)
        .get(Resources.class);
    return (rs!=null ? rs.getResource() : null);
}
```

**Figure 18: REST method to lookup all resources in the given application domain**

And the third REST method (Figure 19) requires one more parameter named `content` with value "full" to specify that we need to retrieve also the child objects - see line (3) in Figure 18. Notice the new name `id` of the parameter denoting the application domain id - see line (2).

```java
public static ApplicationDomain getApplicationDomain(WebTarget target, String appDomainId) {
    ApplicationDomains apps = target.path("appdomain")
        .queryParam("id", appDomainId) // (2)
        .queryParam("content","full") // (3)
        .request(javax.ws.rs.core.MediaType.APPLICATION_XML)
        .get(ApplicationDomains.class);
    return apps.getApplicationDomain().get(0);
}
```

**Figure 19: REST method to lookup the specific domain along with the child objects**

The remaining code in the client class implements the business logic to filter the authorization policies and match the resources with the corresponding host identifiers and the server names. The full java source code is available from the download site [21]. Several assumptions have been made to keep the code simple: the port numbers are not considered and the error conditions are not explicitly handled.

**Step 5: Extend the REST client for security and supportability**

Production deployment may require use of secure protocol (https) or addition of run-time error, e.g., based on the HTTP headers received from the REST server. Also new business goals may require the client to update certain policy resources with the PUT, POST, or DELETE REST commands. Come to the COLLABORATE15 session to discuss how the REST client described here can be extended further.
References

[1] Oracle Identity Management: Oracle Access Manager
http://www.oracle.com/technetwork/middleware/id-mgmt/overview/index-090417.html

[2] Oracle Fusion Middleware Online Documentation Library 11g Release 1 (11.1.2.2) Identity and Access Management
http://docs.oracle.com/cd/E40329_01/index.htm

[3] University of Guelph: About the University
http://www.uoguelph.ca/info/factsfigures/

[4] Oracle Business Intelligence: Business Intelligence Publisher
http://www.oracle.com/technetwork/middleware/bi-publisher/overview/index.html

http://software.dell.com/products/foglight-for-oracle/

[6] Nagios
http://www.nagios.org/

http://docs.oracle.com/cd/E40329_01/dev.1112/e27134/plcyadmn_rest.htm#BHAJEDCD

[8] Jersey RESTful web services in java - Reference implementation of JAX-RS
https://jersey.java.net/

[9] Project JAXB - Reference Implementation of JAXB, the Java Architecture for XML Binding
https://jaxb.java.net/

[10] Google Apps for Education

https://www.zimbra.com/

https://developers.google.com/google-apps/sso/saml_reference_implementation?cs=1

http://docs.oracle.com/cd/E40329_01/admin.1112/e27239/audit.htm#AIAA373

http://docs.oracle.com/cd/E40329_01/admin.1112/e27239/audit.htm#AIAA381

http://docs.oracle.com/html/E18558_01/fusion_requirements.htm#BCFJGDHA
[16] Fusion Middleware Application Security Guide 11g Release 1 (11.1.1), section Configuring and Managing Auditing
https://docs.oracle.com/cd/E28280_01/core.1111/e10043/audpolicy.htm#JISEC2003

https://docs.oracle.com/cd/E28280_01/core.1111/e10043/audpolicy.htm#JISEC3652

[18] Oracle Fusion Middleware: Quick Installation Guide for Oracle Business Intelligence 11g Release 1 (11.1.1)
http://docs.oracle.com/cd/E28280_01/bi.1111/e16518/toc.htm#BIEQI105

[19] Oracle® Fusion Middleware: Data Modeling Guide for Oracle Business Intelligence Publisher 11g Release 1 (11.1.1)
http://docs.oracle.com/cd/E28280_01/bi.1111/e22258/using_data_model_edit.htm#BIPDM124

[20] Oracle® Fusion Middleware: Report Designer’s Guide for Oracle Business Intelligence Publisher 11g Release 1 (11.1.1)
http://docs.oracle.com/cd/E28280_01/bi.1111/e22254/create_report_new.htm#BIPRD2090


[22] Bill Burke: RESTful Java with JAX-RS 2.0, Paperback, O'Reilly Media; Second Edition (Dec 2 2013)


[24] About the author
http://www.educause.edu/members/zdenek-nejedly
http://ca.linkedin.com/in/zdenekn

Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Zimbra logo and logo types are trademarks of Zimbra, Inc. Google and the Google logo are registered trademarks of Google Inc. Dell, the Dell logo, and Foglight, IntelliProfile, PerformaSure, and Tag and Follow are trademarks of Dell Inc. Nagios Enterprises owns the trademark Nagios and the Nagios logo. Other names may be trademarks of their respective owners.