Oracle SOA Tips and Tricks

Ahmed Aboulnaga
Raastech
USA

Keywords:

SOA Suite, WebLogic Server, OSB, BAM, OWSM, OSR, Integration, Oracle Fusion Middleware

Introduction

A typical integration infrastructure can leverage a multitude of products such as Oracle SOA Suite, Oracle Service Bus (OSB), Oracle Business Activity Monitoring (BAM), Oracle Web Services Manager (OWSM), and Oracle Service Registry (OSR). This whitepaper covers a wide selection of tidbits for each of these products; some being popular best practices while others are little known but valuable recommendations, covering important areas such as reporting, authentication, adapters, batching, JMS destinations, and more, all within the various Oracle Fusion Middleware 11g products mentioned above.

This whitepaper is intended for experienced SOA developers and administrators.

Business Activity Monitoring (BAM) Usage and Considerations

Oracle Business Activity Monitoring (BAM) is included as part of the Oracle SOA Suite license and provides reporting capabilities. Data can be published into BAM via a variety of approaches such as web services or the BAM Adapter. Though BAM provides, real-time dashboards, analytics, and monitoring of business processes, it is not a business intelligence tool such as Oracle Business Intelligence Enterprise Edition (OBIEE). BAM is typically seen being used by the integration team, capturing business process metrics and metadata.

In a past project, we had opted to make a call to BAM from each one of our integrations, capturing key metadata such as timestamp, composite instance id, order number, customer number, and so forth (actually, this metadata was first dumped into a JMS queue before being consumed and published to BAM). This, for example, allowed us to use BAM to query all orders placed by a customer, and trace it back to the appropriate composite if needed. This resulted in huge amounts of records being published to BAM; in the order of millions. What we learned is that BAM should not be treated as a data repository, even for key metadata or sensor related information. BAM simply does not work well with huge amounts of data, and it may be due to how this data is maintained in the BAM Active Data Cache (ADC).

Thus, our findings were:

- Do not use BAM as a data repository, and consider other better suited products for this (e.g., OBIEE).
- BAM is not intended for extremely large amounts of data.

Furthermore, BAM reporting is exceptionally weak. It is not possible to provide user sorting at the column level, there is no way to dynamically modify column widths in the reports, navigation is cumbersome, and search capabilities are weak (e.g., particularly working with NULL data).

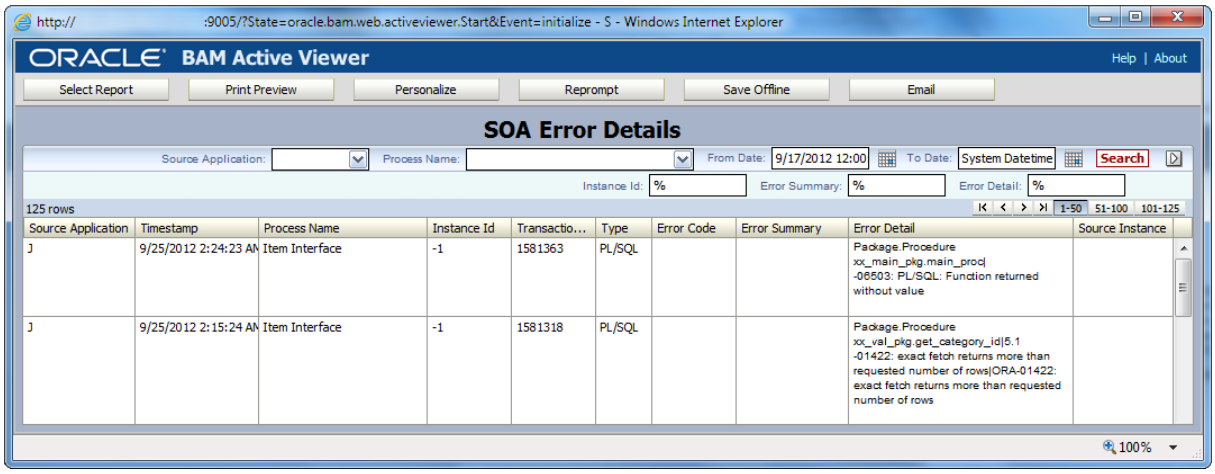

Illustration. 1: BAM reporting capabilities are exceptionally weak.

Furthermore, it is still surprising that the BAM web application runs only on Internet Explorer. Though BAM 11g is now finally a Java-based application (the previous version was Windows-only), having a web application run only on Internet Explorer is simply unacceptable. Illustration 2 proves this. We hope/expect Oracle to rectify this in future releases.

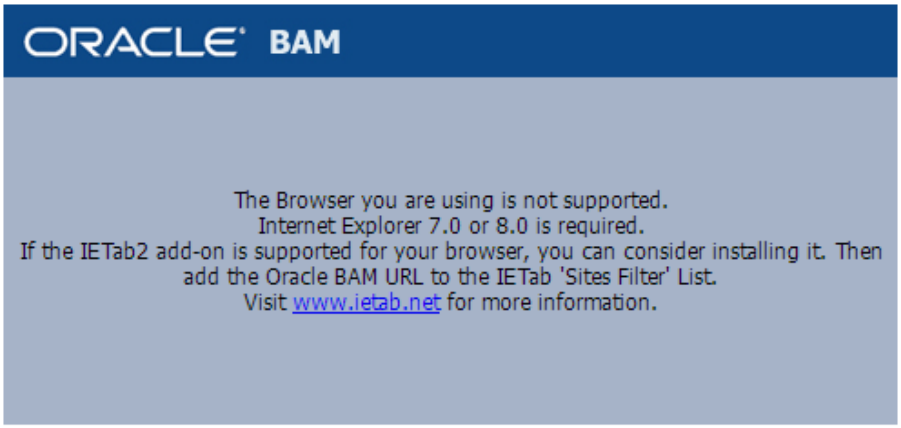

Illustration. 2: BAM only runs on Internet Explorer.

Addressing the earlier concern of BAM's inability to handle massive amounts of data, one consideration that is often overlooked and may help alleviate the situation is to purge the data objects. This can simply be done via BAM alerts, as shown in Illustration 3, as long as the database object is not based off of a database table. If that were the case, then simply create a scheduled PL/SQL package that purges data older than a predetermined age.

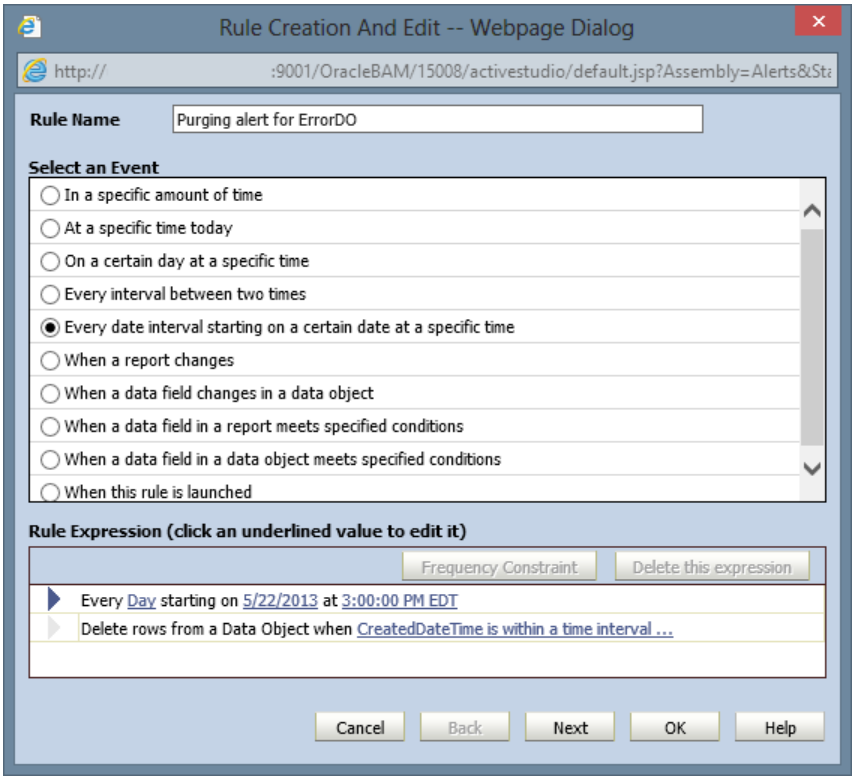

Illustration. 3: Purging data objects using BAM alerts.

BAM Adapter Recommendations

The BAM Adapter is used within the SOA Suite composite to publish data into BAM. This is by far one of the simpler methods of publishing data into BAM. This adapter comes in two flavours; RMI and SOAP. It must first be configured within WebLogic Server (as shown in the screenshot in Illustration 4), and this JNDI is used within the SOA composite.

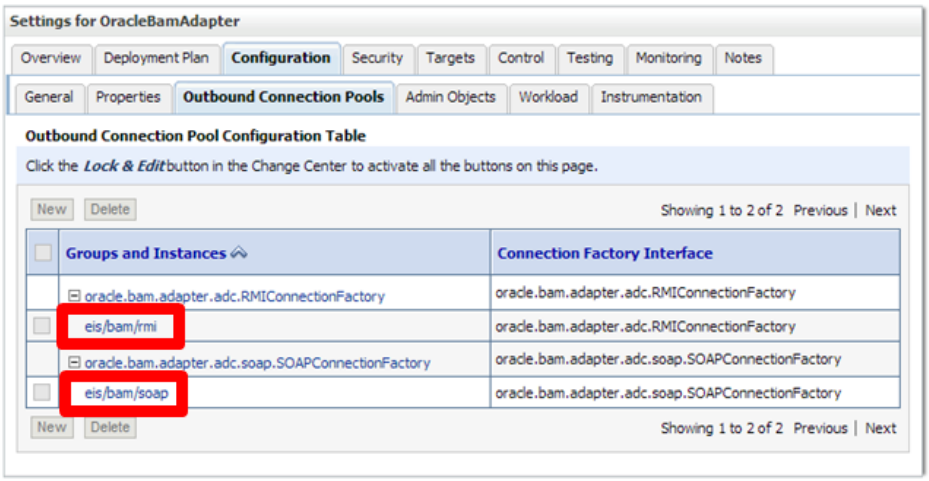

Illustration. 4: Configuring the BAM adapter in WebLogic Server.

There are two main reasons why to avoid the RMI version of the BAM Adapter. It is not as efficient as SOAP, and secondly there is a bug in BAM Adapter which leads to BPEL threads not being released (if used from within BPEL).

BAM Batching via the InBatch Property

The BAM Adapter supports the "InBatch" property. When enabled, this causes the BAM Adapter to publish the data to BAM in batches. This reduces the chatter between the SOA and BAM and is terribly efficient. For example, instead of 10,000 composite instances making 10,000 calls to BAM, they may be batched and only 500 calls are made instead. There are a few considerations regarding the behaviour of this property, as well as failover consequences.

The InBatch property can be set at both design time and runtime (as shown in Illustration 5).

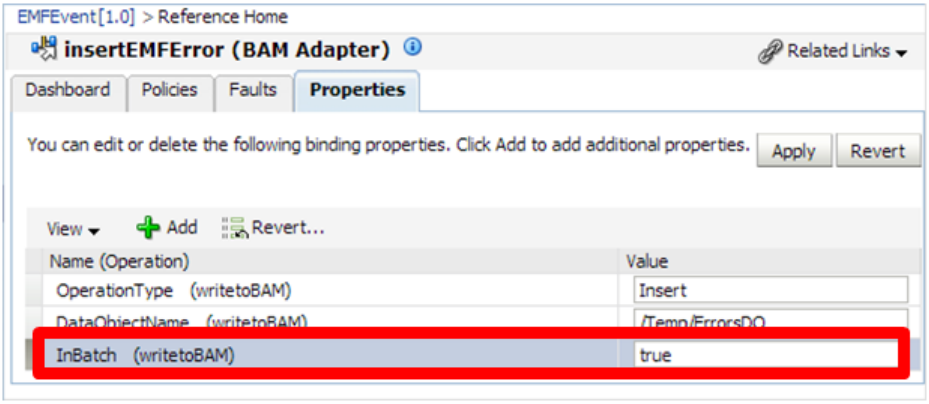

Illustration. 5: Setting the InBatch property with the BAM Adapter at runtime.

When a SOA composite publishes data to BAM where InBatch=Y, the composite will report this invocation as successful 100% of the time, even though the data has not made it to BAM yet. Later on, this data is published as part of a batch to BAM. However, if there happens to be an exception thrown (e.g., data validation error) and the data fails to be published, the composite never becomes aware of this. The error in the soa_server1 logs simply note that the batch failed to publish, with no indication as to why. The bam_server1 logs show details of the error though. Furthermore, if a single record sent to BAM fails, the entire batch fails.

If the BAM server happens to be down, this is not a problem. The SOA server batches the data, and publishes it to BAM once it is back online. However, this data continues to reside in the JVM, and if the SOA server is restarted before BAM is back online, the data is lost. Clearly there is a limit on how much data can be batched because of the size of the JVM.

Oracle Service Bus (OSB) Usage and Considerations

Oracle Service Bus (OSB) provides routing, mediation, and orchestration capabilities. It is a leading product for integration development and included as part of the Oracle SOA Suite license.

The question often arises on when to use OSB versus BPEL versus Mediator for integration development. Simply put, there is a lot of overlap between OSB and SOA composite

functionalities, but OSB is far more superior in performance because it is stateless. In addition, it offers capabilities not supported in composite development such caching, throttling, and service virtualization.

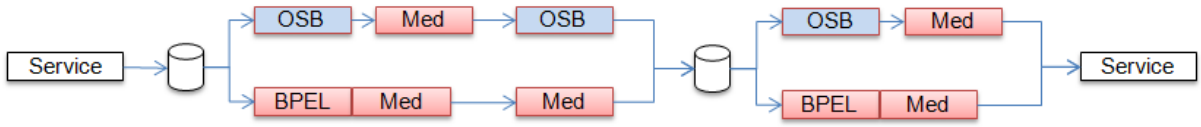

Illustration. 6: Design of a load test comparing OSB and BPEL performance.

Illustration 6 demonstrates two flows designed nearly identically and containing the same business logic, one flow leveraging BPEL while the other OSB. In internal testing, publishing 1000 messages through the OSB route completed within 3:23 minutes. The BPEL flow took 10:19 minutes to complete the same flow. In these tests, OSB outperformed BPEL by 68%.

However, OSB has some severe limitations; the most serious being within transaction visibility, tracing, and metrics. It is either difficult, cumbersome, or impossible to find out if transactions have gone through the OSB engine and detailed, dynamic metrics are unavailable. It is in our opinion that transaction visibility trumps performance.

Furthremore, OSB 11g cannot reference artifacts in the MDS (MetaData Store). Custom code is required for DVM support, and even then, multi-input DVMs are not supported.

Using the OSB Report Action for Payload Auditing

It is possible to use the Report Action though to capture incoming payloads. This is advantageous during runtime when troubleshooting issues, and is somewhat similar to setting the composite audit level to “development” within a SOA composite.

The Report Action can help capture SOAP headers and SOAP bodies (though not together in a single payload). To do so, in the Proxy Service on the request stage:

1. Add a Report action for $header
2. Add a Report action for $body
3. Specify at least one “Key Name” for both
4. Add the “…Request” string to the name and use the same value from the body

As shown in Illustration 7, this provides the ability to filter by index. For example, it now becomes possible to search for a message that has a specific order number. By clicking though the report index, it is possible to view the captured payload.

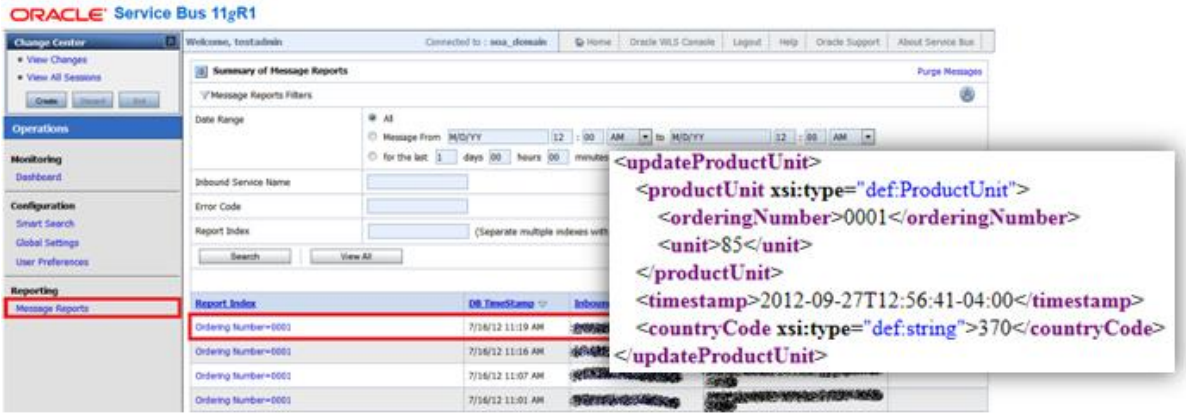

Illustration. 7: Viewing a payload within the OSB console.

Note that header and body payloads are captured and displayed separately. But enabling this on both the request and response (i.e., in the event that every OSB service would have 4 report actions) can quadruple the overall latency, meaning that the service is four times slower. This is because the data must be persisted, affecting performance. In fact, this is the only OSB functionality that is persisted.

JCA Adapters in OSB

Though SOA Suite offers a multitude of powerful JCA adapters, such as the File Adapter, FTP Adapter, Database Adapter, JMS Adapter, Oracle Applications Adapter, and BAM Adapter, OSB instead leverages a series of native adapters, such as the native FTP adapter. Sadly, these adapters are much simpler (and weaker) than their JCA counterparts.

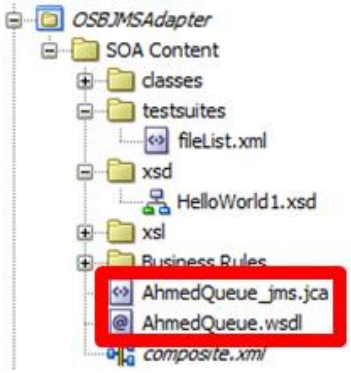

Illustration. 8: Creating a Database Adapter in JDeveloper 11g.

However, it is possible to create a Database Adapter within JDeveloper 11g (see Illustration 8), and manually importing the .JCA and .WSDL files generated to OSB. Though not supported for all adapters (e.g., FTP Adapter), leveraging the JCA Database Adapter gives the ability to take advantage of pooling and timeout features not afforded by the native adapter. In addition, the adapter configuration can be shared between OSB and SOA Suite services.

OSB Caching with Coherence

Though we highlighted numerous disadvantages within OSB, there are several positive ones such as throttling and caching.

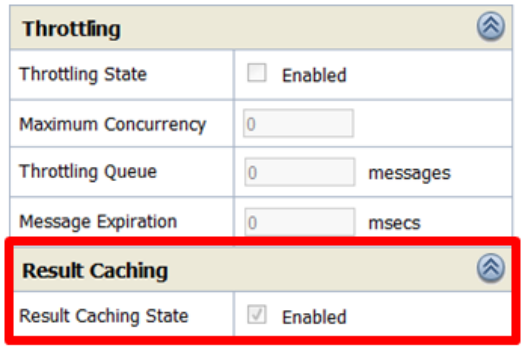

Illustration. 9: Enabling Result Caching within the OSB console.

Result Caching is a feature that can be enabled on the Business Service. It is intended to be used with data that is not frequently updated. For example, if you were running an online web store, typically the product info changes infrequently. If this data were stored in the database, it can become inefficient requerying this data every time the page is loaded. This is a perfect example of where Result Caching can provide some benefit, and the backend database will not be re-queried until the cache has expired. This can result in a 15% to 25% performance improvement with the Database Adapter.

Result Caching can be enabled and disabled at runtime as needed.

OSB Issues and Resolutions on Exalogic

Exalogic is touted by Oracle as an engineered system. Simply put, it is an highly optimized piece of hardware and yields tremendous performance benefits. On OSB 11g installations on Exalogic, however, the following problems were experienced:

- Client requests get a read time out error
- The BEA-380000 error "Request Entity Too Large" appears in the logs
- OSB invocations retry multiple times every 5 minutes
- OSB invocations invoke target services twice

To resolve these issues, a few additional steps need to be performed on the Proxy Services and Business Services.

For all Business Services

1. Under HTTP Transport Configuration
2. Set "Use Chunked Streaming Mode" to "Disabled"

For all Proxy Services

1. View Message Flow
2. Edit Route Node(s)
3. Check on "Quality of Service" and set it to "Exactly Once"

Securing Services with Oracle Web Services Manager (OWSM) Policies

Most services developed on SOA Suite listen on HTTPS. This allows all communication from the calling clients to travel through a secured channel (i.e., not in clear text). Though this helps in protecting the data in transit, it does not necessarily secure the service itself. The service can still be called by any client through simple web service client tools such as SoapUI. Thus, it is recommended to additionally secure the service itself. One such method is enabling authentication on the service. Any caller would have to supply a username and password in the SOAP header, thus preventing rogue clients from making unsolicited service requests.

Enabling authentication is very simply and leverages the Oracle Web Services Manager (OWSM) 11g policy engine.

For the OSB proxy service (see Illustration 10):

1. Click on the Proxy Service
2. Click on the "Policies" tab
3. Select "From OWSM Policy Store" and click "Add"
4. Select the policy "oracle/wss_username_token_service_policy"
5. Update, activate, and submit the changes

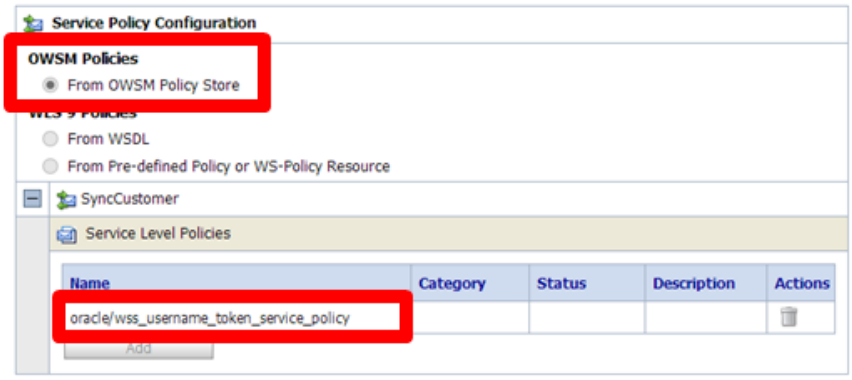

Illustration. 10: Securing an OSB proxy service using an OWSM policy.

For the SOA composite service (see Illustration 11):

1. Navigate to the composite, scroll down, and click on the reference
2. Click on the "Policies" tab
3. Attach the policy "oracle/wss_username_token_service_policy"

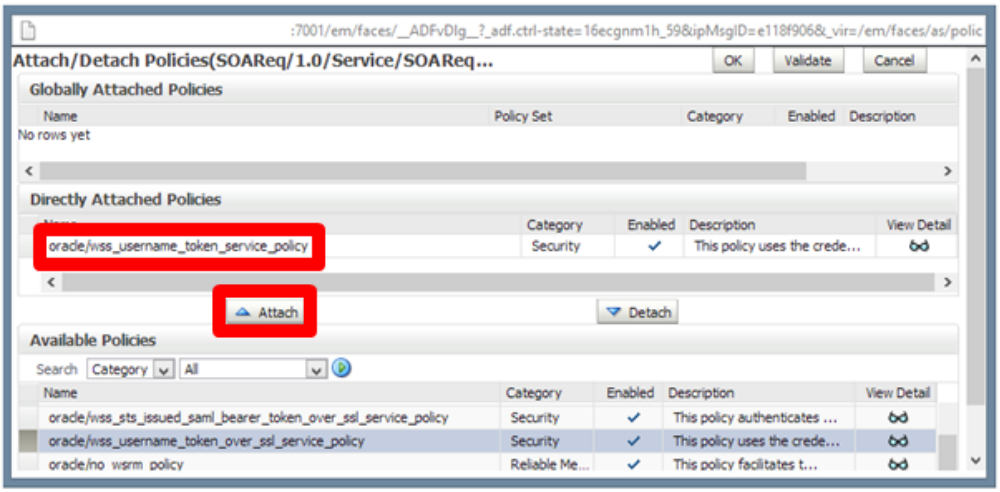

Illustration. 11: Securing a SOA composite service using an OWSM policy.

Now that the policies are secured, the only change the calling client has to make is to add a SOAP header with the UsernameToken as shown below:

```
<soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <wsse:Security soap:mustUnderstand="1" xmlns:wsse="http://
   docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">
    <wsse:UsernameToken wsu:Id="UsernameToken-
tk6qMWkQ5hl3Md2INvp16Q22"
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd">
      <wsse:Username>oratest</wsse:Username>
      <wsse:Password Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-
       profile-1.0#PasswordText">password123</wsse:Password>
    </wsse:UsernameToken>
  </wsse:Security>
</soap:Header>
```

Avoid Mediator Parallel Routing Rules

Mediator executes routing rules either sequentially or in parallel. Even if a Mediator service is designed as a one-way service and a sequential routing rule is used, it behaves similarly to a synchronous transaction (that is, the client holds onto the request until the request is completed, and faults are propagated back up to the client). Thus, despite their superior performance, sequential routing rules should not be used for true asynchronous transactions. On the other hand, by using parallel routing rules, services can be designed to be truly asynchronous. However, the service engine executes these requests in a rather unique fashion. Let's say you have five Mediator services deployed to your SOA server and each of these has a single parallel routing rule. Let's say that the average time it takes to complete execution of each service is one second. If you receive 10 requests on the first Mediator service and the

Mediator Service Engine is configured with 10 threads, you would expect all requests to be completed within one second. That is not the case.

The Mediator Service Engine loops through every composite, which has a parallel routing rule every X seconds, where X is the "Parallel Locker Thread Sleep" setting. In our example, the service engine will loop through the five Mediator composites, one at a time, and see if there are any requests to process from the composite's internal queue. It processes only one request from that composite, then moves on to the next composite. It finds no request, so it moves on to the next, and the next, until it loops back to the first composite, where it would then process its next request. The engine behaves this way by design in order to prevent starving of threads caused by load on a single composite. What the engine wants to avoid is that, if you have a Mediator service that has received hundreds of thousands of requests and another one having received two requests, each service is given a fair amount of time to be serviced, otherwise the two requests may have to wait for hours to execute.

Given this understanding, implementing sequential routing rules may improve performance of Mediator anywhere from 4% to 509%.

Simply edit all *.mplan files for your asynchronous Mediator operations and change the executionType as follows:

OLD:	executionType="direct"
NEW:	executionType="queued"

Singleton Property with JMS Topics

There are two types of JMS destinations; Queues and Topics. Queues support single consumers and operate in a first-in-first-out (FIFO) fashion. Once a message is consumed from the queue, the message is gone. Topics, on the other hand, support multiple consumers. The message remains in the Topic until all consumers have consumed the message or until the message has expired, whichever comes first.

In the case of a clustered installation of Oracle SOA Suite 11g, there are cases where a message in a Topic may be consumed multiple times by the same consumer. This is incorrect behaviour and only happens with Topics created in a cluster. Thus, for the inbound JMS Adapter which consumes a message from a Topic on a clustered installation, it is recommended to use the "singleton" property to avoid consumption of the same message multiple times. Think of this as an additional check becing performed. Though there is a minor performance hit as a result, it ensures operational correctness.

This is done by simply added the "singleton" property to the JMS Adapter under the Services section in composite.xml as shown here:

```
<service name="JMSConsume" ui:wsdlLocation="JMSConsume.wsdl">
  <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/jms/JMS#wsdl.inter
face(Consume_ptt)"/>
  <binding.jca config="JMSConsume_jms.jca">
    <property name="singleton">true</property>
  </binding.jca>
</service>
```

For more information on the behaviour of Topics, please refer to the following blog posts:

> Single message consumption from distributed topic in WebLogic Server 11g
> http://blog.raastech.com/2012/04/single-message-consumption-from.html
>
> Understanding the "singleton" property with WebLogic JMS topics
> http://blog.raastech.com/2012/07/understanding-singleton-property-with.html

Limitations with Distributed JMS Topics

We actually discourage the use of JMS Topics, even though the previous section describes how to avoid a certain issue related to it. The reason is because it is not possible to have truly highly available JMS Topics with WebLogic Server 11g.

The expected behaviour of a highly available topic is that if a single consumer to the Topic exists, and this code is deployed to all 4 nodes of a SOA Suite cluster, the message should be consumed only once. Before the message is consumed, it should reside on and be is equally available to all nodes of the cluster, so that if any node fails, the message is still available and can be consumed without manual intervention.

Take the example where some Java (or SOA) code is deployed to 4 nodes of a cluster. A JMS topic is also created in this cluster.

The expectation is as follows:

1. If there is 1 consumer to this topic, we expect that despite this code being deployed to all 4 nodes of the cluster, that the message is consumed only once.
2. We expect that the message is equally available to all 4 nodes of the cluster, so if any 3 of the nodes fail, the message is still available and can be consumed without manual intervention.

Point #1 is not possible if you set the forwarding policy to "Replicated". Point #2 is not possible if the destination's forwarding policy is set to "Partitioned". Granted these are the only two options available when creating a distributed Topic, we are unable to satisfy the two basic and necessary requirements needed for high availability Topics.

In a clustered environment, there is no simple way to create a simple JMS destination that is targeted to all managed servers and accessible via a single JNDI. This issue one of WebLogic Server's biggest failings.

Understanding the PassThroughHeader Property

Do you need to use the "passThroughHeader" property? Take the scenario where Composite A calls Composite B which then calls Composite C. If there are any elements in the header in Composite A and these are not explicitly referenced (used) by Composite B, then they are dropped from the header. They will never reach Composite C. This is the default behaviour of Oracle SOA Suite 11g.

By adding the "passThroughHeader" property to the SOA component, elements in the header are passed from one composite to another. This is done as follows:

```
<component name="UpdateCustomer">
```

```
  <implementation.mediator src="UpdateCustomer.mplan"/>
  <property name="passThroughHeader">true</property>
</component>
```

Take the case of authentication information. Composite A may receive a username and password in the header which may not be used by Composite B, but Composite C may need it to authenticate against an external service. Without the use of the "passThroughHeader" property, it is lost.

Oracle Service Registry (OSR) Findings

When a service-oriented architecture is adopted in an organization, it usually starts by implementing a few web services. But over time, the number of services grows, from web services to polling services to authenticated services. Keeping track of these services is sometimes done through a spreadsheet, and many times never at all.

Over time, most organizations encounter the following problems:

- No central service registry
- Creation of duplicate/redundant services
- Difficult to obtain interface specifics on existing services

One solution to address this is through the use of a private (or internal) UDDI registry. Oracle Service Registry (OSR) 11g is a UDDI compliant registry. OSR maintains web service metadata via four basic data elements within a UDDI data model. This information includes:

- businessEntity (modeling business information)
- businessService (high level service description)
- tModel (modeling a technology type or service type)
- bindingTemplate (mapping between businessService and tModels)

Last decade, it was envisioned that clients and consumers would dynamically look up services from a UDDI registry, then make the invocation to the queried service. This, among other reasons, provides location independency. Usage of this methodology did not catch on as intended, thus leveraging a UDDI registry (thus OSR) is a waste of time.

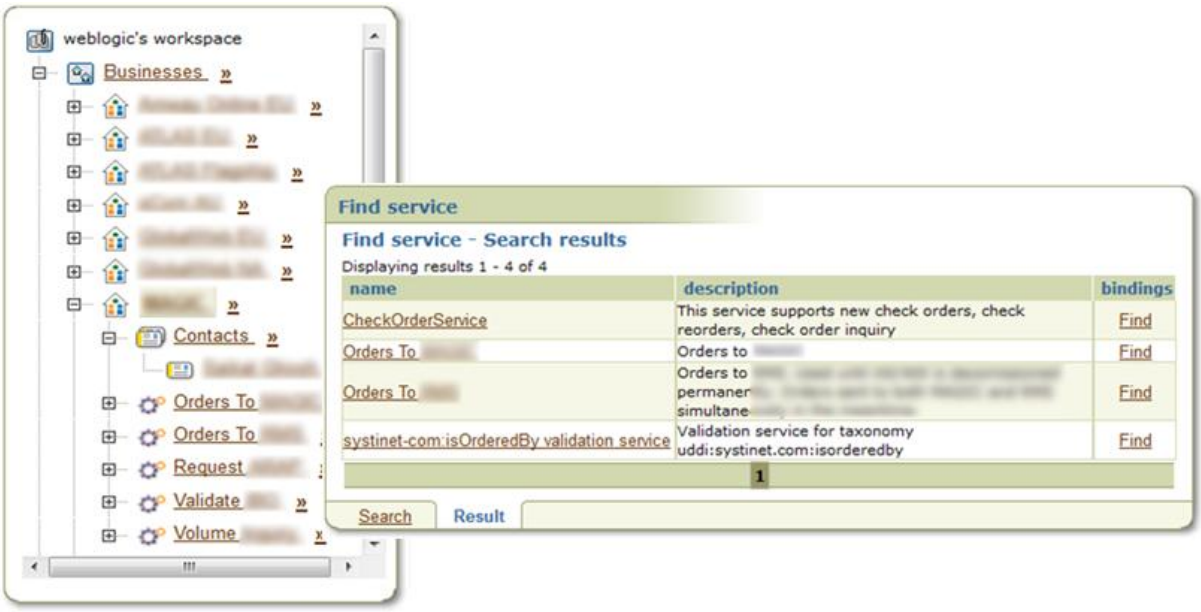

Illustration. 12: Screenshots from OSR 11g depicting 'businesses' and 'services'.

Though UDDI was a hot topic 10 years ago, it is not so much nowadays. Oracle is now recommending leveraging Oracle Enterprise Repository (OER) 11g instead. OER provides system of record for all SOA asset information, defines required architectures, standards, and assets for reuse, automatically detects usage, tracks compliance of services, and provides end-to-end traceability.

OER can work in conjunction with OSR to bridge design-time and runtime requirements. OER can further provide policy management, auto-assignment of assets, automated workflow, analytics, and much more.

Summary

This whitepaper covered random topics surrounding many Oracle integration product sets, with the aim of highlighting new areas that the reader may not have encountered or experienced before.

In summary, it is always important to leverage a product for what it is designed for instead of forcing it into your reference architecture. Furthermore, it is equally important to understand the strengths and limitations of each of them to help position them correctly within your framework.

In summary:

- BAM should not be thought of as a business intelligence tool.
- Though OSB is largely superior in performance compared to SOA composites, it has poor transactional and instance visibility.
- SOA Suite is a rich, complex, and fully featured product which truly covers all aspects of integration. It has a lot of aspects to consider, and learning about them takes time and experience.
- Avoid JMS Topics in WebLogic Server.
- Consider avoiding OSR and looking into OER instead, if governance is an area of concern for your organization.

Contact address:

Ahmed Aboulnaga
Raastech, Inc.
2201 Cooperative Way, Suite 600
Herndon, VA 20171
USA

Phone: +1-571-249-1155
Email ahmed@raastech.com
Internet: www.raastech.com