Introduction in Eventing in SOA Suite 11g

Ronald van Luttikhuizen
Vennster
Utrecht, The Netherlands

Keywords:

Events, EDA, Oracle SOA Suite 11g, SOA, JMS, AQ, EDN

Introduction

Services and events are highly complementary instead of competing paradigms in the IT landscape. Oracle SOA Suite 11g emphasizes the importance of events by supporting the Event Delivery Network (or EDN) in the SCA infrastructure. This session provides an introduction of eventing in SOA Suite 11g. It will start by explaining the basics of events, introduce several messaging patterns such as fire-and-forget and publish/subscribe, and explain some real life examples of using events in an SOA landscape. It will then dive into the underlying eventing infrastructure of Oracle WebLogic Server 11g and Oracle SOA Suite 11g that is based on JMS and AQ, and demo their use both inside and outside SCA composites using resource adapters, PL/SQL, and Java. The session will finish with the discussion of Oracle SOA Suite 11g's EDN and the use of sensors and monitors in SCA composites.

What is eventing and why use it?

Events are the occurrences of something that is relevant to you or your business, and that can require you to act upon. Events are also called notifications, or changes in state.

Events can be categorized into:

- Business events. Businesses deal with (and are causing) events all the time: a customer moving to a different address, a new purchase order, receiving an invoice from a supplier, sending a bill to a partner, and so on. Entities in a business' ecosystem such as employees, partners, suppliers and customers all react to these events. They initiate new processes, perform activities, propagate events, and so on. It is therefore logical to incorporate events when modelling business processes, which are at the heart of SOA. Just as in all modelling practices, one wants to model and focus on important aspects. In case of SOA this includes processes, services, and events.
- Technical events. From a technical point of view events can achieve asynchronicity and decoupling. By using publish/subscribe and queuing mechanisms, software components are not required to know of each other's existence. They simply subscribe to a topic and act based on received events. The other way around, if components have some (intermediate) result or state, they can share this with the rest of the world by publishing an event and then forget about it. These components don't need to know what other components are interested in this information. Of course you'll need some glue (i.e. eventing middleware) to implement this.

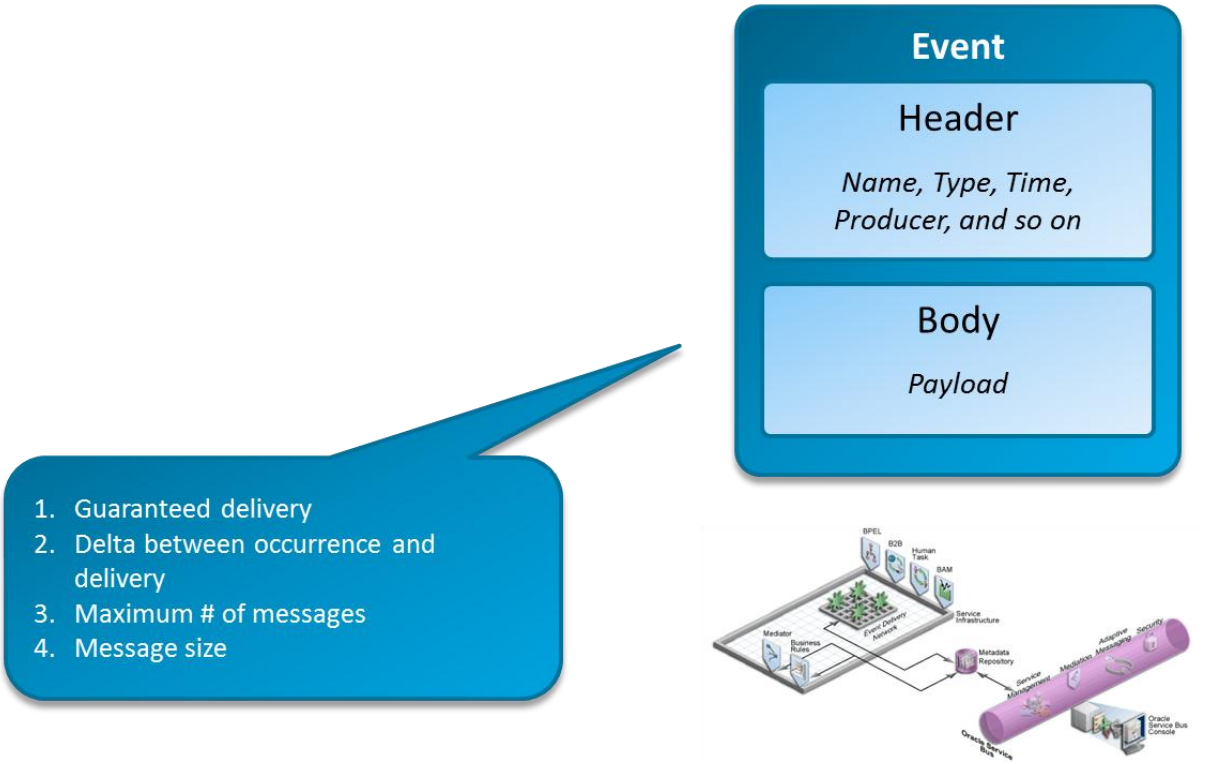
Events can be time-based, meaning they occur at a fixed time or in regular intervals. For example an event at the end of every month that triggers a billing process. Also, the absence of events (non-

events) that you would expect to occur can be considered events as well. For example, a checked-in bag that doesn't arrive in the plane while you would expect it to be.

Not all things that happen are interesting for an enterprise; we need to differentiate between notable events and ordinary events. Events can be seen as a continuing stream of facts that happen. These events need to be filtered and combined so that only important events remain that are of interest to you.

Events and services

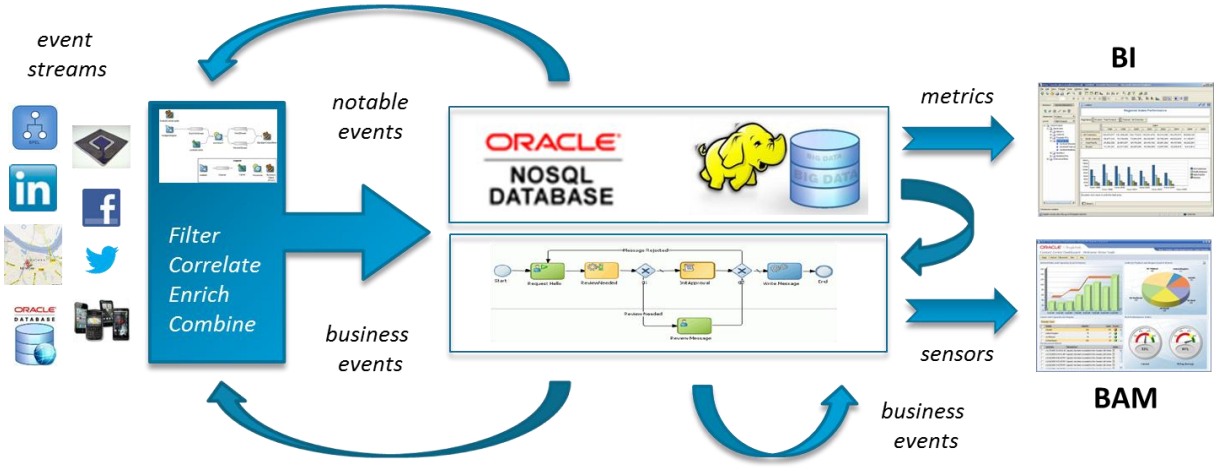
A Service-Oriented Architecture contains services as well as events. Like services, events consist of a contract, interface, and implementation. The interface specifies the payload of the event and important metadata such as time of publication, the implementation could be JMS, AQ, or some other technology, and the contract describes the conditions under which the event is published and consumed (e.g. guaranteed delivery, message size, delivery time after the occurrence of the event, and so on).



The main difference between services and events is that a service consumer needs to know about the service provider: the service to invoke, the operation to invoke, the request payload of the operation, and the service endpoint (or at least where to find the service). When using events, the originator of the event just publishes the event to the eventing platform and continues processing. The publisher doesn't need to know who is interested in the event, what to do with the event, and how to deliver these events to the subscribers.

Oracle and Event-Driven Architecture (EDA)

Oracle offers several middleware products that help you to implement an application landscape that is capable of handling and analysing events. An example of such a landscape is show in the following figure.



Enterprise deal with a continuous stream of events that originate from different sources with a high volume (RFID, log files, social media, smart devices, IT systems, and so on). Not all of these events are of interest to the enterprise. Oracle Event Processing (OEP), formerly known as Complex Event Processing (CEP) can be used to filter, correlate, enrich, and combine these events so that only important events remain.

Business events are handed over to the BPM and SOA platform and initiate new processes and services; for example a complaint by a customer triggering a CRM process. These processes and services generate business events themselves that are consumed by other processes and services, or sent back to OEP for further processing. The processes and services also publish sensor data regarding the performance and progress of process instances. These sensors are combined by Oracle Business Activity Monitoring into dashboard that provide runtime insight into the performance of these processes.

Raw and/or the filtered (notable) events can be fed into a Big Data platform consisting of Oracle NoSQL Database and Hadoop for further storage and analysis. From here, business events can be created and published to BPM and SOA platform. Important metrics can be fed into Oracle's BI tooling for further analysis by business users.

Messaging patterns

There are several basic patterns for exchanging events, among others:

- Queuing;
- Publish-subscribe;
- Fire-and-forget;
- Event-stream processing.

<u>Queuing</u>

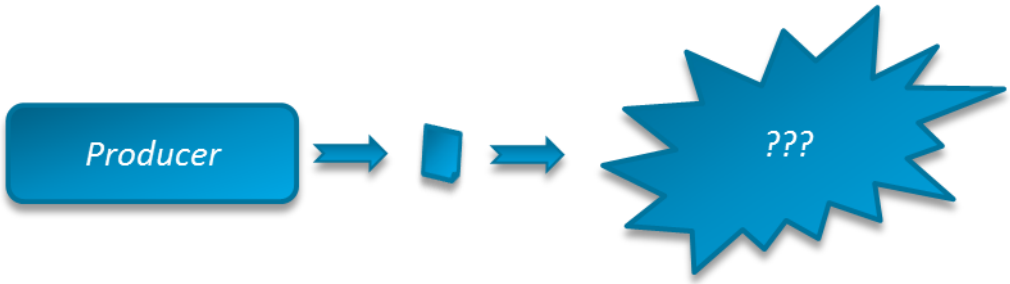
Queuing is a basic pattern in which producers create events and publish them (enqueue) onto an intermediary that is called “queue”. Consumers can retrieve these messages from the queue which is called dequeuing. Once a consumer has dequeued the message, it is removed from the queue.

Publish/subscribe

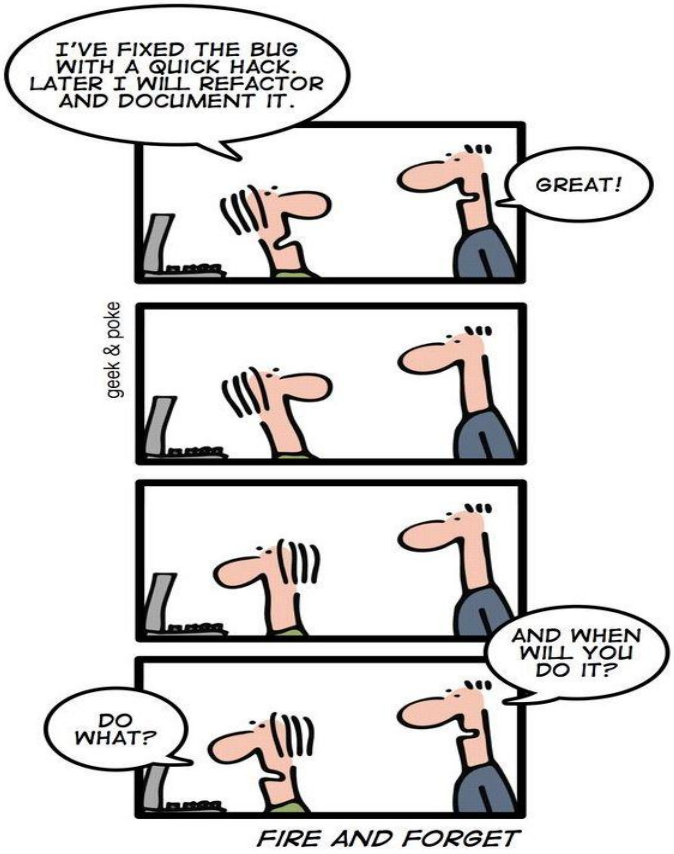
The difference between queuing and publish/subscribe is that publish/subscribe supports multiple consumers (zero, one, or more). The same message can thus be delivered to multiple subscribers. The intermediary to which consumers subscribe often is called “topic”. When events are durable, the eventing platform only removes the event after it has been delivered to all consumers.

Fire-and-forget

Fire-and-forget isn’t necessarily different from queuing or publish/subscribe. However, it emphasizes message exchanges in which producers have no knowledge of the subscribers.

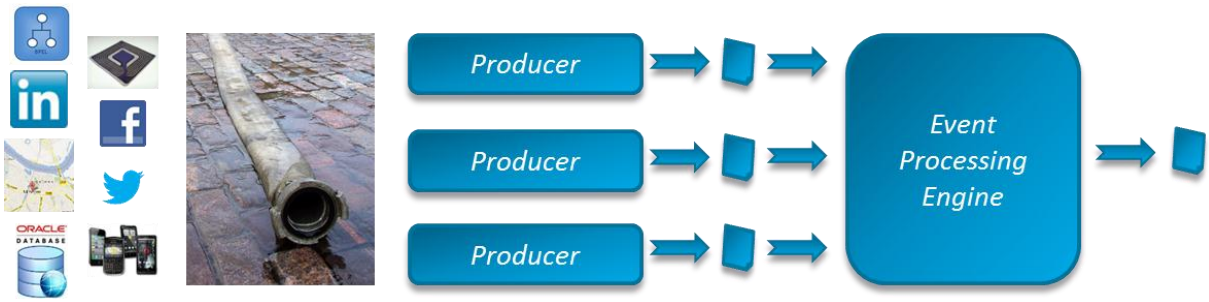


The producer generates the event, publishes it to the event platform and immediately continues work without respect to what happened to the published event.



Event-stream processing

Event-stream processing is a set of technologies rather than a pattern for processing continuous streams of events. These events are filtered, enriched, correlated, and combined to transform raw events into meaningful events. For this purpose, specific query languages are used that often include a time-dimension: if certain events occur within a specific timeframe (e.g. cash withdrawals from the same account from different continents within an hour), a business event (e.g. possible fraud) is generated.



Implementation

There are numerous implementations that support eventing. Examples of platforms offered by Oracle are:

- Advanced Queueing (AQ);

- Java Message Service (JMS);
- Event Delivery Network (EDN);
- Oracle Event Processing (OEP).

Others implementations include:

- Third party Message-Oriented Middleware (MOM);
- Social Media: Twitter, Facebook, LinkedIn, Google Talk;
- Java Action Listeners;
- AMQP implementations;
- And so on, and son …

Oracle SOA Suite lets you take advantage of AQ and JMS through JCA Resource Adapters, and EDN natively. You can also use Monitoring Objects, Sensors, and Composite Sensors to push data from running process and service instances to BAM, Enterprise Manager, and other tools.

Advanced Queuing (AQ)

Oracle AQ is a valid choice as eventing platform when integrating processes and services implemented in SOA Suite with Oracle RDBMS-based applications. AQ stores events in queue tables and provides an PL/SQL interface to interact with queues and perform dequeuing and enqueuing.

A simplified script to create a database schema, grant the necessary rights, and create a queue is shown in the following snippet:

```
CREATE USER JMSUSER IDENTIFIED BY JMSUSER;
GRANT CONNECT, RESOURCE, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to
JMSUSER;
GRANT EXECUTE ON DBMS_AQADM TO JMSUSER;
GRANT EXECUTE ON DBMS_AQ TO JMSUSER;

DBMS_AQADM.CREATE_QUEUE_TABLE(
    Queue_table        => 'JMSUSER.NEW_CUST_ADDRESS_QT',
    Queue_payload_type => 'SYS.XMLTYPE',
    Sort_list          => 'PRIORITY, ENQ_TIME',
    Multiple_consumers => 'TRUE');

DBMS_AQADM.CREATE_QUEUE(
    Queue_name              => 'JMSUSER.NEW_CUST_ADDRESS_QUEUE',
    Queue_table             => 'JMSUSER.NEW_CUST_ADDRESS_QT',
    Max_retries             => 5,
    Retention_time          => 10512000,
    Retry_delay             => 5);

DBMS_AQADM.START_QUEUE(Queue_name                                    =>
'JMSUSER.NEW_CUST_ADDRESS_QUEUE');
```

AQ has the following characteristics:

- Implementation: Oracle Database
- Payload types: RAW, Oracle Object, and JMS-based types

- Integration: PL/SQL, AQ JCA Resource Adapter
- Delivery: Single-consumer and Multi-consumer

AQ allows for the configuration of various settings including priority of messages, filtering of messages, correlation of messages, and retry settings. AQ queues can participate in local and global transactions.

Java Message Service (JMS)
JMS is a Java-standard that is implemented by Java EE-compliant application servers such as Oracle WebLogic. JMS is a good choice as eventing platform when integrating processes and services implemented in SOA Suite with Java-based applications.

It has the following characteristics:

- Implementation: Oracle WebLogic Server
- Persistency: in-memory and file-based persistency
- Delivery: queues and topics (publish/subscribe)
- Integration: Java, Message-Driven Beans (MDB), JMS JCA Resource Adapter, JMS over AQ (hybrid)

JMS allows for the configuration of various settings including priority of messages, filtering of messages, correlation of messages, and retry settings. JMS queues can participate in local and global transactions.

Event Delivery Network (EDN)
EDN was introduced in SOA Suite 11g and provides a light-weight eventing platform used mainly between SOA composites (BPEL and Mediator components). EDN uses AQ or JMS as underlying eventing platform, but abstracts away their technical details at the expense of (advanced) configurability and features. The Event Definition Language (EDL) is used to define events based on XML message types.

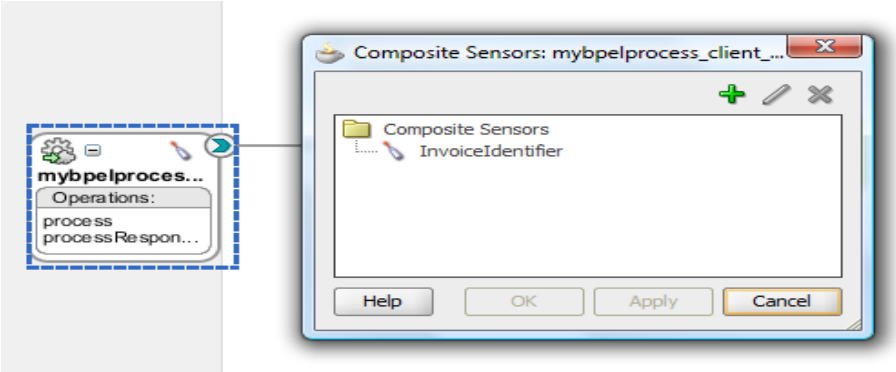
EDN has the following characteristics:

- Implementation: AQ (EDN-DB) or JMS (EDN-JMS)
- Delivery: Publish/subscribe (no durable subscribers)
- Integration: BPEL, Mediator, ADF BC, PL/SQL and Java API
- Payload: XML

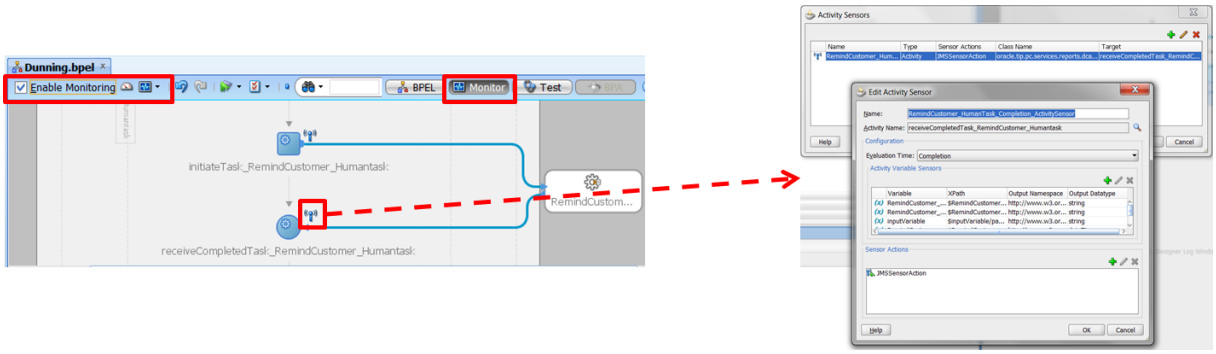
Monitors & Sensors
Monitors and sensors are features in SOA Suite that can be used to push relevant data from running process and service instances to monitoring tools such as BAM, Enterprise Manager, and other 3rd party tools. Monitors and sensors have minimum impact on running processes and services since faults in sensor data evaluations and publication doesn't stop the process instance itself.

- Monitoring Objects can be used to push data to BAM and other tools that require process instance data;
- Composite Sensors can be used to push data to Enterprise Manager to facilitate searching of process and service instances.

The following figure shows how composite sensors can be added to SOA composites within JDeveloper. In this case, an InvoiceIdentifier is added to an exposed service. The sensor data is evaluated at runtime based on the input data. IT operations can use these composite sensors values to search for process instances in Enterprise Manager next to out-of-the-box criteria like start date and instance state.



The following figure shows how monitors and sensors can be defined in JDeveloper. These sensor values can be directly published to BAM, or published to other eventing platforms such as JMS and AQ.



Summary and Best Practices

- Processes, services and events are complimentary instead of competing paradigms. SOA is not only about (synchronous) services and processes describing what should be done. You also need to take into account events that define when important changes in state occur.
- Model events in your BPM and SOA architecture and design.
- Use events to notify running processes and to decouple your processes and services.
- Do not use queuing alone as messaging pattern, also investigate the use of publish/subscribe and other patterns.
- Expand your service registry and repository to include events and event artifacts besides processes and services.
- There isn't a single best technology that should be used as eventing platform. Oracle provides various options such as AQ, JMS, EDN, and OEP that all have their own added value. Important aspects when choosing an eventing platform are integration capabilities with other components, functionality, and the volume and type of events you are dealing with.

Contact address:

Name
Vennster
Postbus 31457
6503 CL, Nijmegen, the Netherlands

Phone:	+31 (0)6 52456043
Email	ronald.van.luttikhuizen@vennster.nl
Internet:	vennster.nl
Twitter:	rluttikhuizen
LinkedIn:	http://nl.linkedin.com/in/soaarchitecture