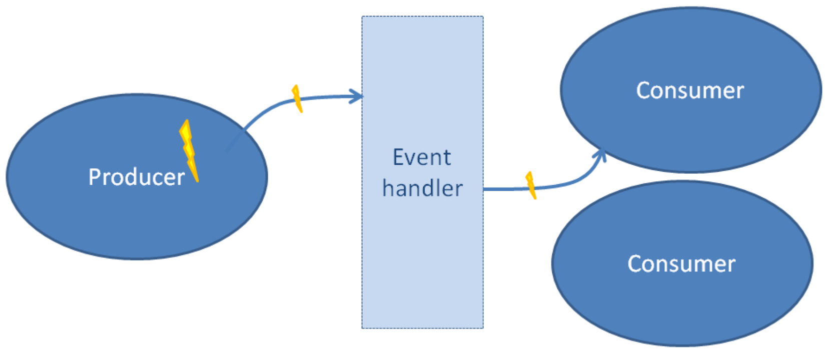
FUSION COLUMN FOR THE ODTUG TECHNICAL JOURNAL - MARCH 2010

A WORLD OF EVENTS WHERE NEWS TRAVELS FAST

Our IT systems are moving towards real-time operations: users expect to always see the current data, deviations or exceptions should be responded to immediately and as Isaac Newton already pointed out: actions in one place may need to trigger reactions somewhere else.

In this real time environment, it is important to rapidly get new information to anyone who may be interested. However, for various reasons we do not want to burden the producer of the information with the task of relaying the message to all interested parties. And this is where events have entered our applications and architecture.



Examples of events:
-User clicked on button
-Field was changed
-Record in table is being updated
-Result of query has changed
-Tom posted message on forum
-Order was dispatched
-Customer has changed address

Examples of event consumers:
-Server side actionListener
-clientListener, onValidateItem trg
-Database table trigger
-Change Notification Handler
-Activity Stream
-Complex Event Processor
-Synch CRM BPEL process

Figure - Events are signals sent by a producer and propagated by a central event intermediary to subscribed consumers

This notion of events that get published on the one hand and that are distributed to interested consumers on the other is not new at all. Oracle Forms, like Visual Basic, MS Access and other 4GL environments, makes intensive use of the event model where triggers can be hooked into fine grained (user interface) events like field navigation and update. Application Express has a similar concept and browsers too support events at mouse and field level on which JavaScript listeners can be registered.

Even the database has an event infrastructure: we can register triggers on tables that are activated whenever the specified DML event (insert, update or delete) takes place. Calling the trigger is not the responsibility of whoever is updating the records; the update event is captured by the database and all registered event consumers - in this case the database triggers - are notified.

Decoupling

Events can play a very important role in the architecture of complex IT environments. They can be used to fuse together components that are not even aware of each other. Events can help achieve reuse, agility and synergy between stand alone components.

Events are not targeted to a specific consumer. Events are published on some central, generic infrastructure that the event producers use in a fire-and-forget mode. After publishing their event, these producers have no longer responsibility for it. They have done their job by handing the event to the central event coordinating facility and no longer have strings attached to the event. In fact, they may later on even consume their own event just like any other consumer – as the event is not specifically theirs. Events typically have a header and a payload. The header contains meta-data such as the event type and a timestamp. The body or payload contains the details that describe the facts of the event instance that business logic in the consumer will process.

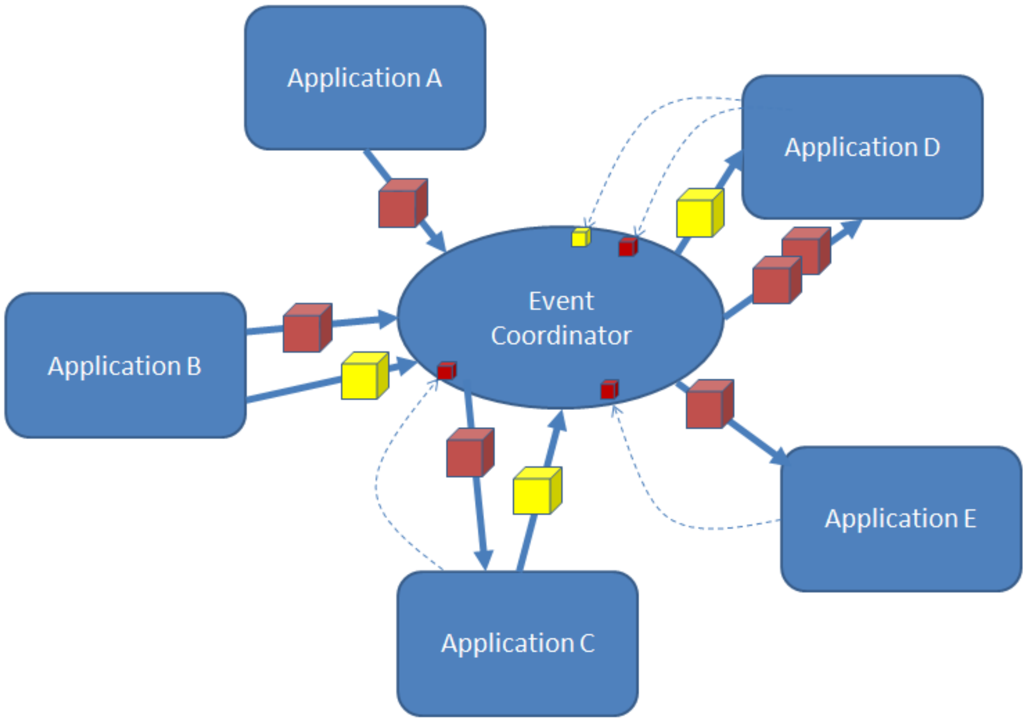


Figure: Decoupled interaction between six applications with an event coordinator handling the subscriptions (dashed lines) and forwarding the events

Anyone interested in occurrences of a specific type of event can subscribe to it, registering their interest in that event type with the central facility that coordinates the events. This coordinating facility receives all events that get published, relieve the event publisher of the responsibility for the event. After receiving an event, it will traverse all subscriptions for the particular event type, taking into account any filters that may have been defined on such subscriptions to see whether the specific event occurrence should actually be forwarded to the subscription's consumer and propagate the event to all qualifying consumers. The event coordinator may support mechanisms to retry delivery of an event upon initial failure or deferred delivery for currently unavailable consumers.

Note: none of the producers of events are aware of these registrations. They should continue to publish their events even if no subscriptions exist at all. They should neither know nor care.

From Poll to Push - don't call us, we'll call you

When we want to know about something that may happen in the near future, we can keep asking the person who will be the first to know whether there is any news yet. In IT terms that would be called 'polling'. It is generally not very efficient, as we waste our own time and the time of the person we keep harassing for the answer by asking a question over and over until at some point there is a meaningful answer. The alternative would be based on an event: when the thing we want to know about happens, the person who finds out first can shout it out loud or publish it in some other way, so everyone with an interest is informed. Obviously, this 'push' approach is much more efficient.

A little parable on moving from poll to push based:

Jenny is not necessarily a nosy person. She just happens to know a lot about what is going on in her corner of St Matthews Hospital. New nurses and doctors, staff calling in sick, rare new cases, VIP patients, extra special operations and staff-dates – Jenny knows it all. And her colleagues know that she does.

One day, Victor, one of the interns asked Jenny to let her know whenever an emergency operation would be scheduled. And Jenny was happy to oblige; whenever the schedule was overhauled because an emergency operation had to be performed, Jenny would page the intern.

Word got round and after a few weeks, other interns came to her with the same request. And Jenny was a good sport and added their names to her list of people to inform upon interjected operations. Then things started to get a little trickier when she was asked by one or two desperate single nurses to keep them informed of newly admitted apparently single male patients. By email this time, as paging would be somewhat ridiculous.

Victor came back to her and – eager to participate in more operations than he was lined up for – sweet talked her into letting him know whenever one of his colleagues who had been scheduled for scrubbing in would call in sick. And if she could please page him as well as leave a voicemail on his telephone – as he might be at home or en route.

Jenny could not cope anymore. Keeping track of all the people, the information they wanted and the channel through which they requested to get just became too much. She had her own job to do as well!

Then she found a perfect solution: instead of maintaining lists of people's interests and calling, paging or emailing them whenever a nugget of information fitting with their particular request had become available, she started to use the corporate Twitter. Every tidbit of information that came across her desk was turned into a tweet. And she told everyone who wanted information from her to just "Read the Feed!".

Life became so much easier. She tweeted her news flashes, not knowing nor caring by this time whether anyone would read it. New readers could join in, old ones could vanish from the crowd – temporarily or permanently, it did no longer affect her.

At some point there was a request to somehow filter her tweets – in order to distinguish between operation warning, sickness notifications and hunk alerts. She started to add hash-tags (#opr, #sck, #hnk) to her tweets and made everybody happy.

Her counterparts in other departments joined in and started to tweet their news as well, in a similar vein. Victor at some point happily scrubbed in on an operation in remote part of the hospital, thanks to the alert some other "Jenny" had twittered.

Unknowingly, Jenny migrated from a distinctly coupled communication pattern to a much more efficient, decoupled approach based on publishing news and messages in general (to the ether) rather than sending them to individual recipients. With a much lighter responsibility and workload, she can make even more people happy than she realizes.

Events in Fusion Middleware

In Fusion Middleware, events play an important role. Both within applications - to efficiently specify how actions performed by the end user should be dealt with - as well as between applications - to achieve interaction and integration in a decoupled manner.

At the level of fine grained User Interface events for example in ADF Faces - with clientListeners (that are invoked on the client side in JavaScript) and serverListeners that are invoked through AJAX calls by the ADF framework to respond to client events such as enter field, push button, change value and click right mouse button. Events such as valueChange and navigation play an important role in server side JavaServer Faces, as well as so called phaseListeners that allow developers to define hooks that are invoked on entry or completion of important stages in the request handling.

ADF Faces also has the concept of 'server push' through the Active Data Services. The idea behind server push is a strive for a real time presentation of current data, without the efficiency drain a polling model would introduce. Whenever the server finds out about an event - change in data, arrival of an email, new entry in an RSS feed or queue- that the user (interface) should know about, the server can push that event to the browser, to allow for an immediate partial refresh of the user interface. This middle-tier to browser push can be supported by a database to middle tier push through the database change notification in Oracle Database 11g.

People Oriented Events

Most events we discuss in this article are consumed by applications and frameworks. However, WebCenter also deals in events that are consumed by people. WebCenter keeps track of activities by users - such as uploading documents, writing notes, adding tags and links, commenting on a discussion forum or publishing a blog - and publishes an overview of all these events by all users in a community in the form of a feed - called activity stream - to all members of the community.

WebCenter also supports various types of communication - including Email, VOIP and Chat. Incoming messages are much like incoming events: they are not solicited and the generic intermediary makes sure the message is delivered (pushed) to the intended consumer. One important difference with events: communication usually is targeted at a receiver whereas evens are not.

Event Driven Architecture

Event Driven Architecture (EDA) made a lot of heads turn recently. As far as I am concerned, Extremely Decoupled Architecture would be a perfect secondary meaning to the acronym – as extreme decoupling is what EDA adds to ‘traditional’ SOA. In the world of EDA, events are the messages, replacing direct calls to external systems or services.

Oracle SOA Suite 11g introduced the Event Delivery Network (EDN). EDN is the central event intermediary. Applications can subscribe with the EDN to receive events of specific types - and that optionally need to satisfy certain payload filter conditions. Any application - SOA service, ADF application, database PL/SQL package - that is the first to learn about or even itself the cause of a (business) should publish that event to the EDN.

In fact, in addition to business process modeling and data modeling , organizations should consider to do rigorous event modeling to define the events that are of interest across multiple systems and departments in the organization. A catalog of business events could be drawn up. Any application with an interest in one or more of these recognized events can subscribe to them with the Event Delivery Network. And any system producing these events should publish them to the EDN.

Events on the EDN carry their data in an XML document that is defined through an XSD (XML Schema Definition) document. Subscribed event consumers know how to process events based on this data model .

Complex Event Processing

In addition to the fairly high level business events like signing of a long term agreement with some vendor or the completion of a stage in a business process, there are many - and I really mean many: up to many thousands events per minute - much finer grained events in most organizations. They usually carry only tiny crumbs of information like the current temperature in some fridge, a click on a navigation link in the patient self service web application, the heart beat of some device or the fact that some security badge was scanned to gain entry into a storage room.

An important aspect of the processing of these events is that much of it must be done in real-time. Ongoing business leads to continuous streams of events, that may require instantaneous actions. We cannot afford to just dump all the data into data warehouses that we will analyze later on – even though for some of the data that is of course exactly what we will do. The urgency of some of the reactions the events may prompt us to come up with is one reason. The sheer volume of the data is another: Thousands of low level events occurring every second represents tons of data that a data warehouse cannot comfortably handle. And should not handle, as the vast majority of those events represent entirely useless information – especially when considered by themselves.

So we need a way to process the zillions of events that the business operation keeps on producing. A processor that turns these events into information – usually by producing a higher level type of events that have some special business meaning, carry a larger information payload and are produced at much lower frequencies. Business events are typically derived from the continuous event streams by aggregation, identification of relevant deviations from target values and detection of special patterns such as a trend or a missing event. Many events are part of a chain of events - for example RFID sensor signals that track the progress of an item or different steps in a business process. When the next expected event is never reported, that in itself may warrant a business event as it may signify something seriously gone wrong.

One of the components of Oracle Fusion Middleware is the Complex Event Processor (CEP). This tool can process thousands of events per second - to derive the aggregated values or find the meaningful deviations or patterns. Oracle CEP consumes events from various sources including file, JMS, Database, HTTP channels and custom sources. It can produce the elevated business events to those same channels that in turn are frequently connected to the Event Delivery Network in the SOA Suite or the Business Activity Monitor (BAM), another component in the SOA Suite.

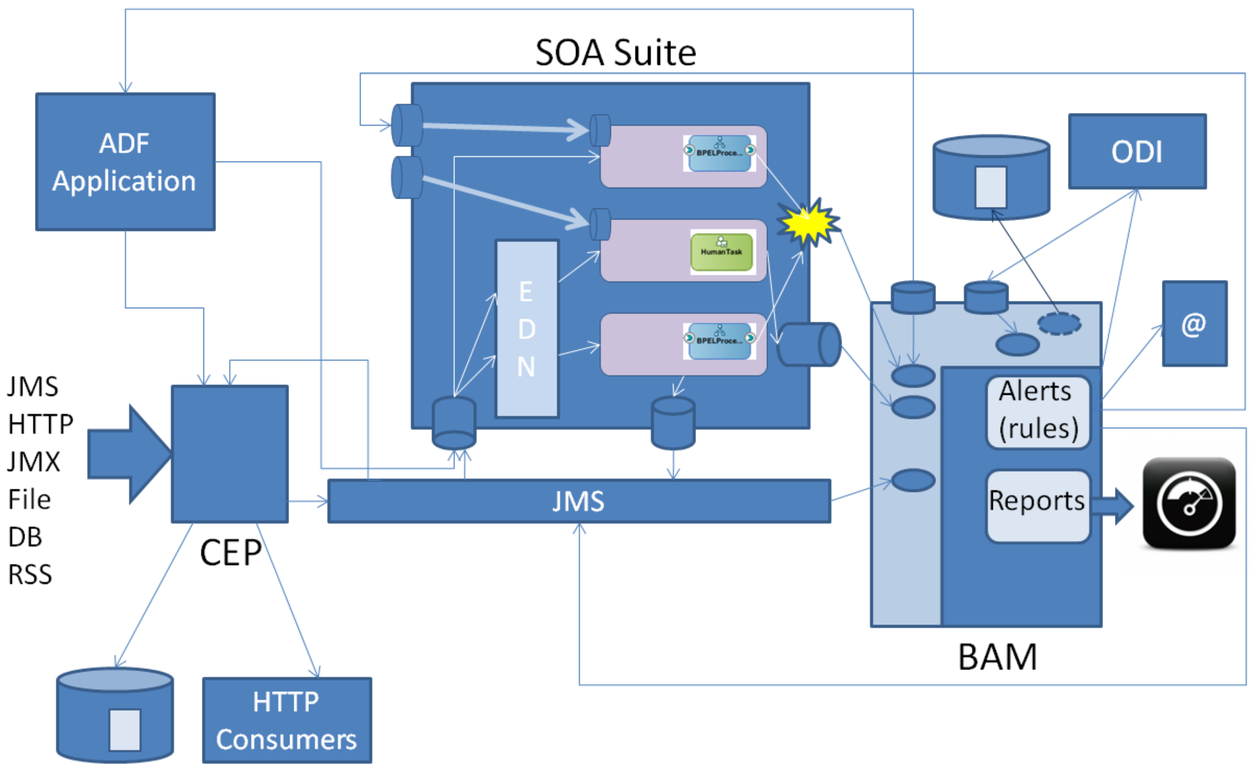
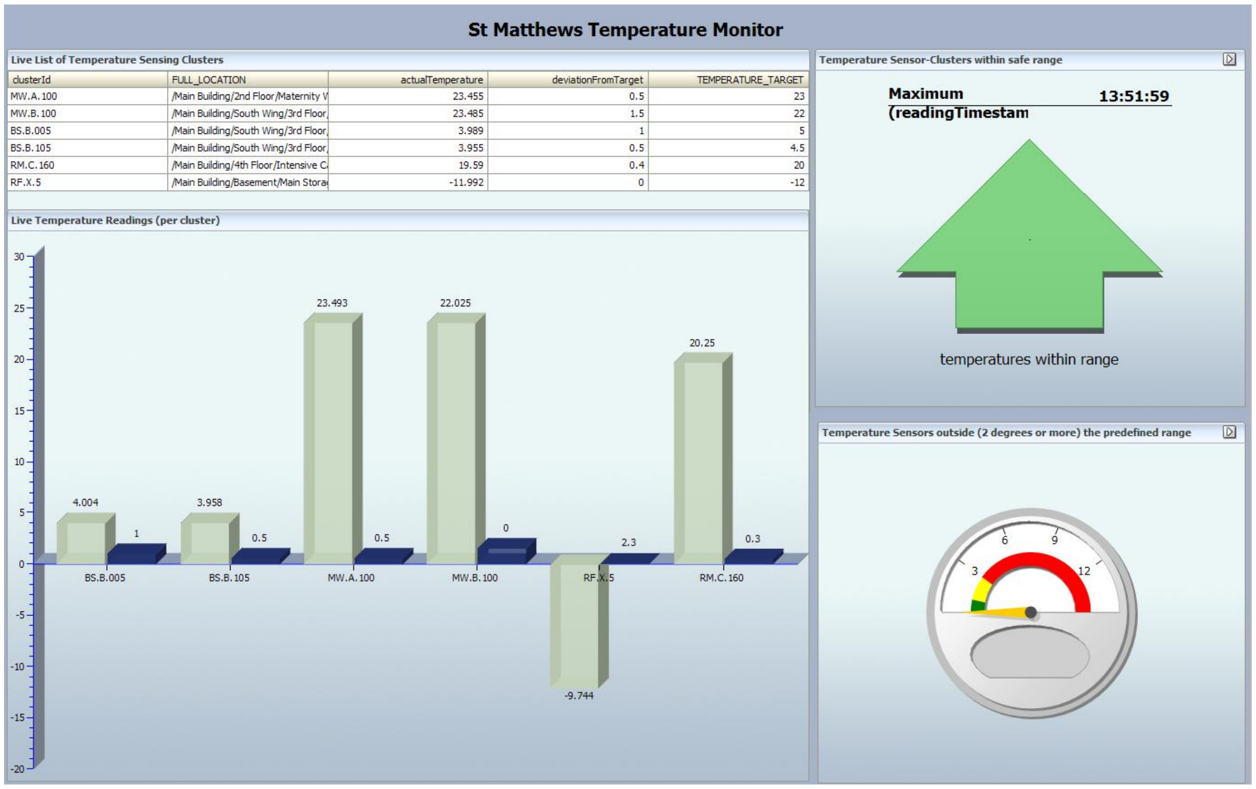


Figure - Events in Fusion Middleware, flowing between applications and components

BAM is used to create real-time dashboards that provide live insight into aggregates and deviations. BAM can also be used to monitor for special conditions and raise alerts when those conditions are no longer satisfied. The results from the BAM Server can also be fed to a WebService, a JMS Queue or an ADF application.



St Matthews Temperature Monitor

Live List of Temperature Sensing Clusters

clusterId	FULL_LOCATION	actualTemperature	deviationFromTarget	TEMPERATURE_TARGET
MW.A.100	/Main Building/2nd Floor/Maternity W	23.455	0.5	23
MW.B.100	/Main Building/South Wing/3rd Floor/	23.485	1.5	22
BS.B.005	/Main Building/South Wing/3rd Floor/	3.989	1	5
BS.B.105	/Main Building/South Wing/3rd Floor/	3.955	0.5	4.5
RM.C.160	/Main Building/4th Floor/Intensive C	19.59	0.4	20
RF.X.5	/Main Building/Basement/Main Stora	-11.992	0	-12

Live Temperature Readings (per cluster)
30
25
20
15
10
5
0
-5
-10
-15
-20
4.004
1
3.958
0.5
23.493
0.5
22.025
0
-9.744
2.3
20.25
0.3
BS.B.005
BS.B.105
MW.A.100
MW.B.100
RF.X.5
RM.C.160
Temperature Sensor-Clusters within safe range
Maximum (readingTimestam
13:51:59
temperatures within range
Temperature Sensors outside (2 degrees or more) the predefined range
3
6
9
12

Figure: Real Time Dashboard created with BAM based on aggregate temperature sensor reading events

Conclusion

Events are packets of information that may have relevancy to certain consumers. Events occur at various levels: from fine grained user interface events and row level data manipulation to high level business process steps. Applying events in our application architecture helps us to achieve interaction between systems and applications without introducing direct dependencies. When we have a central event coordinator - such as the Event Delivery Network in the SOA Suite or a custom AQ or JMS based infrastructure - we can introduce event types, subscriptions to events and subsequently start producing and consuming events . Event driven architecture (EDA) equals extremely decoupled architecture - a corner stone for business agility.