# Event-Driven Architecture and Service-Oriented Architecture

Bobby Woolf, IBM Software Services for WebSphere

*This paper explores what event-driven architecture is, and how it relates to service-oriented architecture and enterprise service bus.*

## What is EDA?

*Event-driven architecture* (EDA) is a technique for integrating components and applications by sending and receiving event notifications. An *event* is an occurrence in one application or component that others may be interested in knowing about. An *emitter* posts an announcement of an event and *handlers* receive notification of it. The announcements are transmitted as *event messages*.

The emitters and handlers must be connected. They connections can be direct (aka point-to-point), but the connections can be better managed if the applications are connected via a broker (aka bus). A broker can implement publish/subscribe channels, useful for publishing events. Handlers register interest by subscribing to a topic, and emitters announce an event by publishing an event message onto the topic. A topic can have multiple event handlers subscribed, but could have just one or none at all.

A broker for transmitting event messages can be further enhanced into an *event processing network*, a more sophisticated connection between emitters and handlers which performs mediation for event messages. It can perform *event correlation*, detecting several separate but related events and combining them into a single event or set of coordinated events.

Not all handlers receive all events. A hander controls what events it receives notification of by selecting which types of events to register interest in, such as which topics it subscribes to. Once a handler receives notification, it reacts accordingly, perhaps by doing nothing.

## EDA and SOA

Some discussions say that service-oriented architecture (SOA) and EDA go together nicely; others say that they're competing technologies, such that an application can implement one or the other but not both. I believe that there’s a lot of synergy between SOA and EDA, and that for a sufficiently complex integration solution, one might well use both architectures. Some communication is performed in an SOA sort of way, while other communication is performed in an EDA sort of way.

So when should an integration design use SOA or EDA? A component should use SOA to send a service request when:

- It knows exactly what service it wants to have run
- It wants it run the service exactly once

- It wants to be notified when the service completes successfully
- It wants to receive the results of the service invocation

This is the classic SOA interaction pattern of a consumer invoking a provider.

A component should use EDA to announce an event when:

- It wants to notify all consumers that might be interested
- It doesn't know what consumers may be interested in an event
- It doesn't know how each consumer may react to an event, and indeed expects that different consumers may react differently
- The communication is one-way: The announcer doesn't want to wait for the consumers to react, indeed doesn't know when they've all reacted, and isn't interested in the results from the reactions

In this EDA interaction, the announcer sends a notification and then is unconcerned with the outcome.

## EDA, SOA, and Services

A significant area of commonality between EDA and SOA is that they both invoke services, but the service selection is different.

The key insight is this: How will an event handler implement its reaction to an event? The reaction should be implemented as a service, a context-free task that can be reused by any code wishing to perform that task. In this way, the hander does not implement the reaction; it delegates that to the service implementation. The handler implements the logic for examining an event, determining whether to react, and determining which service to use to perform the reaction.

With SOA, the service consumer's implementation explicitly specifies the service to invoke, invokes it exactly once, and waits for the result before proceeding. With EDA, emitter simply publishes an event and is finished. Each handler decides how to react and invokes the services to react properly.

So both architectures use services, but differ in how they select which services to invoke.

## EDA and SOA Programming Techniques

The service invocations themselves are implemented differently.

An SOA consumer sends out a service request, a command message instructing the receiver what service to perform. An EDA emitter sends out a notification, an event message capturing details about the change that occurred for the receivers. It is the receiver, the event handler, that decides what service to invoke and what context to pass it.

In SOA, a service request is handled by a *service activator*, which receives a service request, invokes a service, and then optionally sends a response back to the requestor. It's a connector that sits in front of the service to enable the service to be invoked SOA-style.

In EDA, an event handler is a connector with more intelligence. It too sits in front of the service, but listens for events, filters them, and determines which ones are actually important. When it decides that a particular event is important, it reacts by invoking an appropriate service.

A service activator always invokes the same service. A single event handler will usually delegate to a single service, but different handlers for the same event type may each delegate to a different service.

A single service implementation may be executed both SOA-style and EDA style. Front the service implementation with a service activator to execute it SOA-style, and with an event handler to execute it EDA-style. A service capable of reacting to several different events would be fronted by a like number of different event handlers.

## EDA, SOA, and ESBs

An enterprise service bus (ESB) can and should be used to connect event emitters and event handlers, just like it's used to connect service consumers and service providers.

The discussion around ESBs has generally been in terms of how it fits into an SOA, with the service consumers on one side of the bus and the service providers on the other side. The bus connects the consumers to the providers so that the consumers can invoke the services.

EDA uses an ESB in a similar fashion, except to transmit event notifications rather than service requests and responses. In this way, the emitters and handlers don't know about each other directly, they just share the same bus channels. An emitter posts an event by sending it out on the bus, which transmits the notification to the appropriate handlers, which receive the event and react appropriately.

So the bus is really useful both for direct services invocation SOA style, and for indirect event notification EDA style.

In SOA, an ESB provides the opportunity for *mediations* in the form of request routing, message transformation, and protocol conversion. EDA event notification can likewise benefit from protocol conversion and message transformation. The SOA notion of routing is replaced by an EDA notion of publishing—matching an emitter to the interested handlers, often implemented using a publish/subscribe channel.

Mediations can also be stateful—splitting, combining, and reordering messages to create new content that is more meaningful to the receiver. In EDA, this stateful mediation can evolve into an event processing network, handlers in the ESB that correlate independent but related events and emit new events that announce when a correlation has occurred.

## Conclusions

EDA and SOA are not entirely different, but in fact have many similarities and can be used as alternative integration techniques that are quite complimentary. A complex integration solution may wish to leverage both approaches. Both implement their reusable functionality as services. An ESB greatly facilitates implementing both integration styles.