Event-Driven Architecture: Events in the Driver's Seat

by Vijay Sathya
Lead Software Developer

SHAW SYSTEMS WHITE PAPER

How many times have you encountered a situation in which you do not know precise details of the requirements, but yet you want the solutions to take care of those unknown requirements? Is there an architectural style that a Software Architect can follow to ensure that today's solutions will also meet tomorrow's needs? Event-Driven Architecture (EDA) is an architectural pattern that not only helps build robust software solutions, but also provides tremendous benefits in terms of cost savings.

Any large financial institution will confront many challenges when it comes to building IT solutions. In earlier days, all of our end-to-end business transactions were processed within the boundary of our enterprises. In today's world, there is an ever-increasing need to interact with other business partners, federal agencies, third parties providing services, and so on. For example, institutions providing auto loans have to interact with a multitude of auto dealers to process their internal Dealer Floor Plan systems. Users have to continually update the dealer's revolving credit limits using   payments and payoffs, while at the same time verifying with outside agencies Vehicle ID Numbers at the time of acquisition of individual units. They must also interact with an internal retail loan servicing application during the consumer loan's life cycle and provide information regarding payment status to credit bureau agencies.

The frequent changes to business rules offer another set of challenges. In addition, there may be a few unknown systems with which you may have to exchange information in the future. To meet all these challenges you need to build software that will effectively listen to all these known and unknown needs and will respond well, while at the same time keeping costs low.

Events are at the heart of an Event-Driven Architecture. Events are not new: for example, many Windows-based applications track user actions, such as movement of the mouse, as an event, and a typical Graphical User Interface application reacts to those events. Such a programming model is known as Event-Driven Programming. Event-Driven Architecture takes that concept to the next level by building applications essentially to listen for and react to business events. Business events will be lot more coarse (unlike fine-grained mouse clicks or mouse drag and drops), such as "Payment Received Event" from a consumer loan account holder.

An event is defined as something happening that merits the attention of the business. In a typical financial institution, such events constantly occur within the institution's boundaries and from outside. The solution you build needs to respond to those events appropriately. The response could be synchronous or asynchronous. There may be one or more systems within your enterprise that will be interested in knowing the occurrence of such events, and their processing interests may vary widely.

For example, in a bank providing auto loans when a dealer concludes a sale, a "Vehicle Sale" event may arrive from that particular dealer. The floor plan system in the bank would like to register its interest in processing that event to update the dealer account for the remaining balance in revolving credit, while the internal retail system would like to register its needs to board the details of the new consumer loan. The accounting system would like to register its interest to process the movement of asset balances from one category to another.

A typical Event-Driven Architecture-based environment comprises event producers publishing events to an EDA-based application, and the various event consumers subscribe their interests with the message backbone of the enterprise. A message producer may send an event after translating the raw data into an event understandable to the application. On the receiving side, there are listeners working in conjunction with a message backbone, which is basically a message hub through which all events flow. This hub is a Message-Oriented Middleware (MOM) whose implementation could consist of an Enterprise Service Bus (ESB) and/or Java Messaging Services (JMS) provider. The Messages will be delivered to all service components (possibly from different applications) that have registered their interests in processing that particular event. In the course of processing an event, a service component in turn may produce another new but different event and publish it to the message hub. This is where you begin to see the power of reusability of components in EDA.

CONTINUED>>

www.shawsystems.com   ©2012 Shaw Systems Associates, Inc.

Events carry data from sources. Just like an event signifying occurrence of something interesting, it can also carry information about non-occurrence of an activity. For example, an end-of-day event producer can produce a "Delinquency Update Event," carrying account numbers of all loan accounts for which payment is due on that processing date and payments have not yet been received. Consumers of the event can decide to skip or process that event based on data carried.

In traditional legacy architectures, requests are processed through a central controller that acts to control the workflow as part of the processing of requests. The controller logic needs to get modified whenever there are any changes to workflow or any new integration requirements arise. However, in EDA, the controller is absent and is replaced with a listener-based mechanism that hands over events to a number of registered service components. These service components could be components you have already developed as part of Service-Oriented Architecture (SOA).

Removal of the centralized controller and being able to consume events by a list of autonomous service components in parallel helps to build a very robust architecture that can handle unanticipated requirements. For example, if you need to process a particular event also in a newly acquired company's legacy system, you can do it very easily. All you may have to do is register the new application with the listener to process the particular event and provide the necessary adapter, if needed, to handle data format differences.

Separating event producers and event consumers with a Message-Oriented Middleware in between is at the center of reaping the enormous benefits of very high agility to meet future needs. Producers of events do not know who the consumers are and whether those consumers will end up producing additional events as a byproduct of processing. Loose coupling obviously helps to reduce the cost of software maintenance.

Event-based processing provides the ability to reuse the service components in processing inputs from a variety of sources. Based on volume of processing and other considerations, transactions may arrive from multiple sources for processing. In a typical bank, if the volume of payment transactions is high, they may be submitted for processing overnight through end of day batch runs. Individual payment transactions may be keyed into the system by an internal user using a Graphical User Interface. Alternatively, a payment processing application may submit a payment transaction as it receives the payment to an accounting system through a web service call and wait for the response. Respective adapters such as a File adapter or Web Services adapter can act as the event producers, creating the event in the format understood by service-side components, while EDA helps to reuse (wherever it makes sense) service components, developed as part of an SOA and maximize return on investments on those components.

Another important benefit EDA provides is the ability to build very useful monitoring and alert applications. As events flow through the system, actionable intelligence can be derived from those discrete events. Using Complex Event Processing (CEP) techniques, correlation between different types of events can be identified and service components can be built to react to the combination of such events. For example, it can help to detect fraud or an intruder by observing login failure events. Or a monitoring system can look for occurrences of "Payoff" events from dealers that are not followed by "Add Consumer Account" events to identify those lost retail loan opportunities. Such real-time analysis can be displayed on a dashboard. Also, we can develop expert applications which will listen to multiple event types, correlate the events, and take actions based on data carried in those event types.

Event-Driven Architecture offers high agility and expandability to integrate with future applications while providing real-time analysis and monitoring as events occur, ensuring that today's solutions will also meet tomorrow's needs.