What is Event Driven Architecture (EDA) and Why Does it Matter?

Dr. K. Mani Chandy, Simon Ramo Professor of Computer Science, California Institute of Technology and
W. Roy Schulte, Vice President and Distinguished Analyst, Gartner Inc.
15 July, 2007

Event processing has emerged as one of the most important issues in IT today. Event processing encompasses two separate, although related, ideas:

1. **Architecture**: Event-Driven Architecture (EDA) is a style of application architecture centered on an asynchronous "push"- based communication model. EDA is the software architecture of choice for implementing "straight through" multistage business processes that deliver goods, services and information with minimum delay. Applications designed using EDA are also easier to modify than traditional applications as business requirements change.

2. **Sense and Respond**: The ability to respond rapidly and effectively to changing conditions is a powerful competitive advantage. Companies can use Complex-event Processing (CEP) techniques, a sophisticated form of EDA, to extract the information value from multiple events. CEP systems find patterns in event data to detect opportunities and threats. Timely alerts are then pushed to the appropriate recipients, often using Business Activity Monitoring (BAM) dashboards or similar end-user information delivery channels. The result is faster and better operational decisions and more timely responses to important situations.

Gartner has developed a new conference on event processing and BAM to be held from September 19th through 21st, 2007 in Orlando, Florida to address both aspects of event processing. See www.gartner.com/it/page.jsp?id=502259&tab=overview for more information. This article is an introduction to the field of event processing to provide background for those who will attend the conference or for anyone who wants to learn more about the subject. The article provides an overview of why architects, software engineers, business analysts and IT managers use EDA with one event at a time in business applications. It describes:

- Why design patterns used in typical applications don't enable the timely and effective behavior that businesses want.
- What EDA is and how it works to improve the responsiveness and effectiveness of business processes.

A second, related article (*"How Event Processing Improves Business Decision Making")* describes how CEP systems "connect the dots" and find patterns in multiple events to implement BAM and other operational business intelligence (BI) systems.

Sensing and Responding to Events As They Occur

We live in an event-driven world. Animals, people, companies and even countries survive and thrive based on their ability to act quickly on opportunities and threats. Zebras on the savannah sense and respond to the opportunity of a water hole or the threat of a lion. Boxers and armies look for opportunities to go on the offensive while watching for signs of imminent attacks. Wall Street traders use algorithmic trading systems that act on arbitrage opportunities in sub-second response times. Banks track credit card usage on a near-real-time basis to stop fraudulent charges as they occur.

Opportunities and threats appear at unpredictable times. So an event-driven entity must respond to events at times that are externally determined. They act when the world changes and not only according to pre-planned schedules. Event-driven business applications are initiated when a shipping company delivers goods, a customer submits a change of address or the price of a stock goes up. These happenings are ***business events***: meaningful changes in the state of something relevant to the business.

All companies have always been event-driven in a real-world sense because they respond to external events generated by customers, suppliers and competitors. Furthermore, most companies moved their time-sensitive applications from scheduled, batch modes of operation to on-line modes during the past forty years. So isn't business IT already event-driven? Unfortunately, in many cases, the answer is "no"! Companies have moved only their simple tasks and some limited parts of their more complex processes to responsive, event-driven modes of operation. They have yet to make the transition to a sense-and-respond style of behavior for many of the more important and complex aspects of their business. The work of IT during the next twenty years will be to complete the evolution of business processes from sequences of slow-moving, disjointed applications to more responsive end-to-end, event-based straight-through flows of action.

Why Now? What Changed?

Organizations were always event driven in a real-world sense, so why weren't their IT systems equally event-driven in the past and why will they become event-driven now? There are 3 reasons why EDA matters more now:

1. **Competitive pressure to deal with the increasing pace of business**: In the past companies could monitor changes in the slow-moving environment periodically – every quarter or month – then plan an appropriate response for the remainder of the period and finally execute the plan. Organizations now need to respond much more quickly to rapidly changing situations. Customers expect to receive the goods that they order sooner. They want to be able to go on the Web to look up the current balance in their account or the up-to-the-minute status of a service request rather than seeing data from the previous night.

2. **On-line responses**: More people are accessible electronically – through email, instant messaging and phones – than ever before. Organizations can proactively inform people as soon as critical events occur. Further, business process management software, which is increasingly widespread, allows processes to be triggered when critical events occur; thus integrating event processing into businesses is easier today than a decade ago. The opportunity for sense and respond IT applications increases as more people remain on-line for more of the time.

3. **IT capability**: Hardware and software resources enable sense and respond applications in IT today which were not possible in the past. The capacity of computers, communication and storage continues to grow exponentially. IT now has the capability to satisfy business needs to respond to conditions as they change. This capability wasn't available a decade ago.

Changing Roles of Request-Driven and Event-Driven Paradigms

Common patterns of interaction among people, people and computer systems and among software components in a computer system fall into three broad categories:

- scheduled
- pulled
- pushed

Scheduled activities work fine when the timing of the work is predictable and fast response to an external entity is not important. CEOs meet with their executive leadership teams every Monday at 8 AM, or a batch job runs on the computer every night at 11 PM. However, some jobs can't be done according to a fixed schedule. Imagine a lion where the visual cortex is out of touch with the part of the brain that controls the leg muscles. The lion would see a zebra and store that fact in its memory. Then, a few hours later, the part of the brain that controls the muscles would come alive through some internal clock and look through memory to see if it has an opportunity to get food. The zebra has been long gone by then. It sounds silly, but this is how many IT systems still work today. A customer logs in via the Web or calls a contact center to order a product and the order is captured on the spot because the order entry application is on-line. But responsiveness may end there because the next stage in the business process, the order fulfillment application, may be linked to the order entry system only through a batch process at the end of the day. The response is deferred to some time that is determined by the company's internal clock even though there is an economic advantage to the company to provide faster delivery to the customer.

Pull-based activities are better suited than scheduled activities for meeting immediate needs. A CEO may call the VP of Manufacturing to ask about the status of the plan to reduce energy consumption. Or a consumer can look up their current bank balance over the Web. Pull interactions are implemented using a request/reply communication pattern. IT has used request-reply patterns from the dawn of computing, starting with procedure

calls, through client-server interactions and now in many Web services and Service-oriented Architecture (SOA) applications.

Companies moved most of their time-sensitive, externally oriented business functions from batch (scheduled) to on-line (pull) so that they could be responsive to real world events. On-line, pull based functions are event-driven on the surface, however they are not using Event-driven Architecture (EDA) in their core so they can't satisfy end users' timeliness requirements for larger business processes or for situations in which the requester (information consumer) does not know when to ask for fresh information. For example, a credit card company does not expect its customers to inquire into their account activity every hour to look for fraud. The company proactively notifies customers when patterns of potentially fraudulent transactions have been detected. In other words, push, rather than pull, is needed for certain kinds of tasks.

Push-based patterns are the best solution for two different, although related, aspects of work. The first has just been described – the credit card company notifying a customer because it has found some information of potential importance. A push application anticipates consumer needs, proactively executes services and then sends critical information to consumers. By contrast, pull systems are inherently reactive.

The second area where push-based relationships are important is in multistep processes. The pull model is real-time and responsive only for a specific task executed by a single component and the components to which it delegates subtasks (also using a request/reply model). For example, the order-entry step in a order-to-cash process runs immediately using a pull-based model, but the remaining steps for order fulfillment, manufacturing, shipping and billing can't be executed using a pull-based model in practical terms because they are done by different application systems in different business units (maybe even in different companies cooperating in the same virtual enterprise).

A complex business process isn't executed in a single step. Rather, it is deconstructed into multiple simple stages, each of which is carried out by a different component – for example, by different people or by separate service-oriented architecture (SOA) services. If communication between components is carried out by combinations of pull and scheduled (batch) operations, the overall performance of the end-to-end process will be slow. Push-based, staged EDA processes consist of a sequence of functions, each executed by a separate software component, and each triggered by the arrival of a message pushed by the previous component in the process (this is sometimes called message-driven processing).

Few business problems call for EDA exclusively. Most enterprise applications should use a mix of pull, scheduled and push communication patterns because each pattern has its advantages. For simple tasks and for simple portions of larger processes, where an external entity is waiting for an immediate reply to a simple request, the pull-based model is still the right answer. For situations where there is no benefit to the customer or the company to operate in near-real-time, the right solution is still a scheduled batch job. However, as the pace of business accelerates and competition to provide faster and better

customer service grows, an end-to-end, sense-and-respond mode of operation is increasingly needed, driving more work toward EDA.

Event-driven Architecture (EDA)

There is more to EDA than just the "push" based communication model. For a system to implement EDA, it must have the particular characteristics described in this section.

Let's first look deeper into some definitions. As we described earlier, an ***event*** is the fact of something happening, such as a bank transaction, stock trade, customer order, address change, shipment delivery or buying a house. Of course, computers can't manipulate events because they are abstractions; so an application system must create an ***event object*** – an electronic signal or report of the event. The fact that Fred Smith withdrew $ 100 from his bank account at 10 AM today is an event. The computer record associated the withdrawal transaction, perhaps in the form of an XML message, is the event object. Computers can transmit and process event objects but not events. A message consisting of an event object is called a ***notification***. (Confusingly, some articles on event processing refer to event objects and messages containing event objects as events. So readers of these articles must consider the context to determine whether "event" refers to a happening or a report of the happening.)

Event processing is defined as computing that performs operations on event objects. Event processing includes creating, reading, deleting, transforming and responding to event objects. An event processing system has at least two components, (1) a sensor or source that senses events and emits event objects, and (2) a consumer or responder that receives and responds to event objects. The act of sensing an event and generating an event object is separate from the act of responding to the receipt of an event object. For example, a smoke detector senses smoke in the atmosphere and generates an event object – an electronic signal – that is sent to the sprinkler system which responds to receipt of the event object by taking the action of turning on the sprinklers. In a degenerate case, the source and consumer are implemented within the same program. A staged EDA application has multiple components each of which may consume and emit event objects, where event objects generated by one component are consumed by others.

EDA is a style of software architecture that deals with "pushing" event objects. Not all event processing is EDA. Event objects can be manipulated in pull and scheduled modes of operation as well as in push mode. Event objects may be stored in information repositories for later data mining, or event objects may be packaged into remote procedure calls (RPC). An application uses EDA only if the three following conditions are met:

1. **Event objects are pushed.**

   Event objects are sent from an event source to the event consumer in asynchronous messages at times determined by the event source. Pushing event objects proactively reduces latency (the time required to respond to an event),

compared to waiting for consumers to pull event objects (for example, by repeatedly asking if any new data is available (polling)). The latency of a pull-based system increases as the time between polling requests increases, but frequent polling may be impractical because of its overhead.

2. **Components process events on arrival.**

   An event consumer responds as soon as it gets an event object. An end-to-end process can respond in a timely manner only if each stage of the process executes when it receives an event object rather than waiting for a scheduled time. Of course, during periods of high load, event objects may arrive faster than they can be processed; in such cases an event object is processed as soon as possible.

3. **Event objects do not specify operations.**

   An event object does not specify the operation that a consumer of the object must perform upon receiving the objects. This decreases the logical coupling between sources and consumers of event objects. The source simply sends a message reporting that an event has happened. The logic to decide what to do about the event is embodied in the consumer (or consumers). Therefore developers have the flexibility of changing or adding event consumers without modifying sources. By contrast, remote procedure call (RPC) mechanisms and most request/reply SOA services include a method name on which the service requestor and service provider must agree; so both requestor and provider must be changed in lock step. The enhanced "plug-ability" of EDA systems is a key advantage over common request/reply systems.

There are many types of event processing applications. A stage in a simple EDA process executes a business function each time it receives one single event object. Each event object is handled largely independently of other event objects. By contrast, CEP applications operate on multiple event objects at the same time (this important topic is explored in the related article mentioned earlier, "*How Event Processing Improves Business Decision Making.")*

Middleware for EDA

Staged EDA applications can be implemented using different types of middleware including message-oriented middleware (MOM) or one-way document-style SOAP messages. MOM functionality, in particular, provides features helpful for improving the integrity and efficiency of EDA processes. Sources and consumers of information are decoupled in time through the use of message queuing – a source can send an event object and terminate at any time because the MOM can store the object until the consumer is ready to receive it. MOM middleware can store messages to disk, guarantee exactly-once delivery to a consumer, balance load, handle failures, provide security and implement publish-and-subscribe (pub/sub) communication patterns.

EDA does not require the use of MOM or pub/sub, although they are often useful. Conversely, an application that uses MOM or pub/sub is not necessarily implementing the EDA architectural style. Pub/sub is also used to distribute data other than event objects – music for example. EDA is not the same as MOM or pub/sub.

Conclusion

EDA is under-utilized because application architects, software engineers and business analysts sometimes fall back on the familiar pull and scheduled models as a matter of habit. Moreover, some software developers, even some well educated ones, haven't used MOM so they find it easier to resort to writing out files or building database tables. This often results in business processes that are slower and less responsive than they should be.

Businesses need to be able to sense-and-respond more effectively to remain competitive, so they need to make more of their complex processes event-driven and straight-through. The push concept and the desirability of running some business processes straight-through are not difficult to grasp and they are being used more frequently every year as business pressures grow and as application developers get more comfortable with using them. Some parts of all new SOA and other business systems should use EDA, while other parts should still use pull and scheduled patterns. The key is to understand the advantages of each and how to employ them.

At the Gartner Event Processing and BAM conference in September 19-21, 2007, we will explore the best practices and software tools associated with EDA at length. The conference has a special emphasis on case studies, about half of which are dedicated to Capital Markets and Banking because of the wide use of event processing in those industries. Thirteen case study speakers will participate, along with three experts from academia and a number of Gartner analysts. We will also be joined by sixteen vendors of event processing products who will demonstrate the range of capabilities available on the market today. See www.gartner.com/it/page.jsp?id=502259&tab=overview for more information.

This article has focused on one aspect of EDA exploitation but there is another related aspect that is equally important: the use of CEP to enhance the quality and timeliness of operational business decisions. One of the indirect, but significant benefits of using simple, event-at-a-time EDA in push-based business processes, as described in this article, is that it provides a foundation for using business events for CEP purposes. That topic is covered in a related article, *"How Event Processing Improves Business Decision Making."*