Patricia Seybold Group

Trusted Advisors to Customer-Centric Executives

Event-Driven Architecture Overview

Event-Driven SOA Is Just Part of the EDA Story

By Brenda M. Michelson
Sr. VP and Sr. Consultant, Patricia Seybold Group

Direct link: **http://dx.doi.org/10.1571/bda2-2-06cc**

Event-Driven Architecture Overview

Event-Driven SOA Is Just Part of the EDA Story

By Brenda M. Michelson, Sr. VP and Sr. Consultant, Patricia Seybold Group *February 2, 2006*

INTRODUCTION

Service-Oriented Architecture and Event-Driven Architecture

Over the last year, every time we wrote or spoke about service-oriented architecture (SOA), [1] we couldn't help but include SOA's interaction with event-driven architecture (EDA). We see, and speak to, two distinct interactions.

In the first interaction, the occurrence of an event (a notable thing that happens inside or outside your business) can trigger the invocation of one or many services. Those services may perform simple functions, or entire business processes. This interaction between events and services is commonly referred to as event-driven SOA. We describe this as a style of SOA.[2]

In the second interaction, a service may generate an event. The event may signify a problem or impending problem, an opportunity, a threshold, or a deviation. Upon generation, the event is immediately disseminated to all interested parties (human or automated). The interested parties evaluate the event, and optionally take action. The event-driven action may include the invocation of a service, the triggering of a business process, and/or further information publication/syndication. In this interaction, the service is purely one of many event sources in a broader event-driven architecture.

Over the last few months, we have seen increasing interest in the SOA-EDA connection, both from enterprises and the vendor community. Enterprise architects and IT strategists are considering how to best create (evolve) an environment for IT and business agility. Thought leaders at leading SOA and integration vendors are coalescing visions and product strategies for SOA, integration, and event processing. This is all great to see.

However, we do have one concern and a point of caution for readers. Many SOA evangelists are only telling part of the event-driven architecture story: that of event-driven SOA. We fear this will lead to short-sided event architecture implementations. Enterprises could miss out on opportunities for real-time information flow and analysis, and complex event processing.

As a service to our architecture readers who think holistically, we have developed this overview report

[1] Service orientation is an architectural concept that refers to the loose coupling of a service (an abstract resource with a defined job) and its provider (the physical asset(s) that perform the job tasks). A requestor only knows what the service's job is and how to request it. The service is the only one that knows its implementation. SOA is an IT architecture strategy for business solution (and infrastructure solution) delivery based on the concept of service orientation.

[2] The two primary styles of SOA used in business solution development are composite application development and flow. In composite applications, the user interaction drives a request for one or many services. Most of the service invocations are synchronous in nature. A composite application typically serves one business domain. Composite applications are often delivered in a portal.

In flow, business process and/or events drive the service invocations. The service invocations are a mix of asynchronous and synchronous; however, the overall flow is usually long running and asynchronous. A flow typically crosscuts business domains and often extends outside of the enterprise. For more information, see our "'Service-Oriented World' Cheat Sheet," June 2, 2005, http://dx.doi.org/10.1571/bda6-2-05cc.

on event-driven architecture. In this report, we explain key event concepts, walk through event processing flows, and identify the major implementation components of an event-driven architecture.

EVENT-DRIVEN ARCHITECTURE BASICS

Event Basics

WHAT IS AN EVENT? An event is a notable thing that happens inside or outside your business. An event (business or system) may signify a problem or impending problem, an opportunity, a threshold, or a deviation.

Specification and Occurrence. The term event is often used interchangeably to refer to both the specification (definition) of the event, and each individual occurrence (instance) of the event.

Define in Business Terms. For an event to be meaningful to downstream subscribers (human and automated) it is imperative that the event (name and body) is specified in business terms, not data or application terms.

WHAT IS IN AN EVENT? Each event occurrence has an event header and event body. The event header contains elements describing the event occurrence, such as the event specification ID, event type, event name, event timestamp, event occurrence number, and event creator. These elements are consistent, across event specifications.

Event Body. The event body describes what happened. For example, if a retailer specified a low inventory threshold event, the event body would contain the information to communicate which product fell below the allowable threshold.

The event body must be fully described so any interested party can use the information without having to go back to the source system. For the low inventory threshold event, the event body would contain not only the product identifier, but also the product description, and the point in time inventory and threshold levels. To ensure events are understood by all consumers, a clear business lexicon or ontology should be used.

WHAT IS AN EVENT-DRIVEN ARCHITECTURE? In an event-driven architecture, a notable thing happens inside or outside your business, which disseminates immediately to all interested parties (human or automated). The interested parties evaluate the event, and optionally take action. The event-driven action may include the invocation of a service, the triggering of a business process, and/or further information publication/syndication.

Extreme Loose Coupling. By its nature, an event-driven architecture is extremely loosely coupled, and highly distributed. The creator (source) of the event only knows the event transpired. The creator has no knowledge of the event's subsequent processing, or the interested parties. The traceability of an event through a dynamic multipath event network can be difficult. Thus, event-driven architectures are best used for asynchronous flows of work and information.

Event Processing Styles

There are three general styles of event processing: simple, stream, and complex. The three styles are often used together in a mature event-driven architecture.

SIMPLE EVENT PROCESSING. In simple event processing, a notable event happens, initiating downstream action(s). Simple event processing is commonly used to drive the real-time flow of work—taking lag time and cost out of a business.

STREAM EVENT PROCESSING. In stream event processing, both ordinary and notable events happen. Ordinary events (orders, RFID transmissions) are both screened for notability and streamed to information subscribers. Stream event processing is commonly used to drive the real-time flow of information in and around the enterprise—enabling in-time decision making.

COMPLEX EVENT PROCESSING. Complex event processing (CEP) deals with evaluating a confluence of events and then taking action. The events (notable or ordinary) may cross event types and occur over a long period of time. The event correlation may be casual, temporal, or spatial. CEP requires the employment of sophisticated event interpreters, event-pattern definition and matching, and correlation techniques. CEP is commonly used to detect and

respond to business anomalies, threats, and opportunities.

EVENT PROCESSING

In this section, we present example event flows for each style of event processing. The flows are represented logically, broken out into four layers. Before jumping into the event flows, we describe the logical layers.

Event Flow Layers

An event flow starts with the event being generated and culminates with the execution of any downstream (event-driven) activity. The four logical layers are as follows:

EVENT GENERATORS. Every event is generated from a source. The source might be an application, data store, service, business process, transmitter, sensor, or collaboration tool (IM, email). An ordinary event may be evaluated for notability by an event preprocessor (router, filter), resulting in the generation of a new notable event.

Because of the variety of event generators, not all events will be generated in the required format for event processing. In those cases, the events need to be transformed to the required (enterprise standard)[3] format prior to being deposited in the event channel.

EVENT CHANNEL. The event channel, typically a messaging backbone, transports standard formatted events between event generators, event processing engines, and downstream subscribers.

EVENT PROCESSING. In the event processing layer, upon receipt, events are evaluated against event processing rules, and actions are initiated. The event processing rules and actions are defined in accordance to the needs of the interested parties, not of the event generators.

The actions include invoking a service, initiating a business process, publishing the event out to a subscription hub, directly notifying humans or systems, generating a new event, and/or capturing the event for historical purposes.

Events are processed by engines. A simple engine processes each event occurrence independently. A complex engine processes new event occurrences in context of prior and future events.

DOWNSTREAM EVENT-DRIVEN ACTIVITY. A single event, or event correlation, may initiate numerous downstream activities. The invocation of the activity might be a push by the event processing engine (service invocation, business process initiation, notification) or a pull by subscribers of event publications. Subscribers might be humans, applications, active business processes, data warehouses, performance dashboards, and/or automated agents. Events should be published in the standard event format. Transformation to subscriber-specific formats[4] is typically done by an enterprise integration backbone.

Example Event Flows

We are presenting simple examples to illustrate the concepts and capabilities of an event-driven architecture. The inherent flexibility at each layer—what events are generated, how they are processed, and who receives them—is the real power of event. As you review our examples, think of your own business and potential event flows that can streamline work and information delivery.

Please note, for the sake of completeness, our event flow illustrations (1-3) include more actions in the processing layer, and more subscriber types than we reference in the examples. The illustrations flow from left to right, starting with event generators.

SIMPLE EVENT PROCESSING. In Illustration 1, we show a simple event processing flow for an online bookseller's inventory position optimization. When a customer places an order for books, the

[3] To date, no industry standards exist for business event definition or notation. IBM has defined a common base event for systems events. The common base event is included in the Web Services Distributed Management (WSDM) specification. The common base event format evolved from IBM's Tivoli (systems management) product line. For more information see: http://xml.coverpages.org/IBMCommonBaseEventV111.pdf

[4] For human event subscribers, we recommend the use of Really Simple Syndication (RSS).

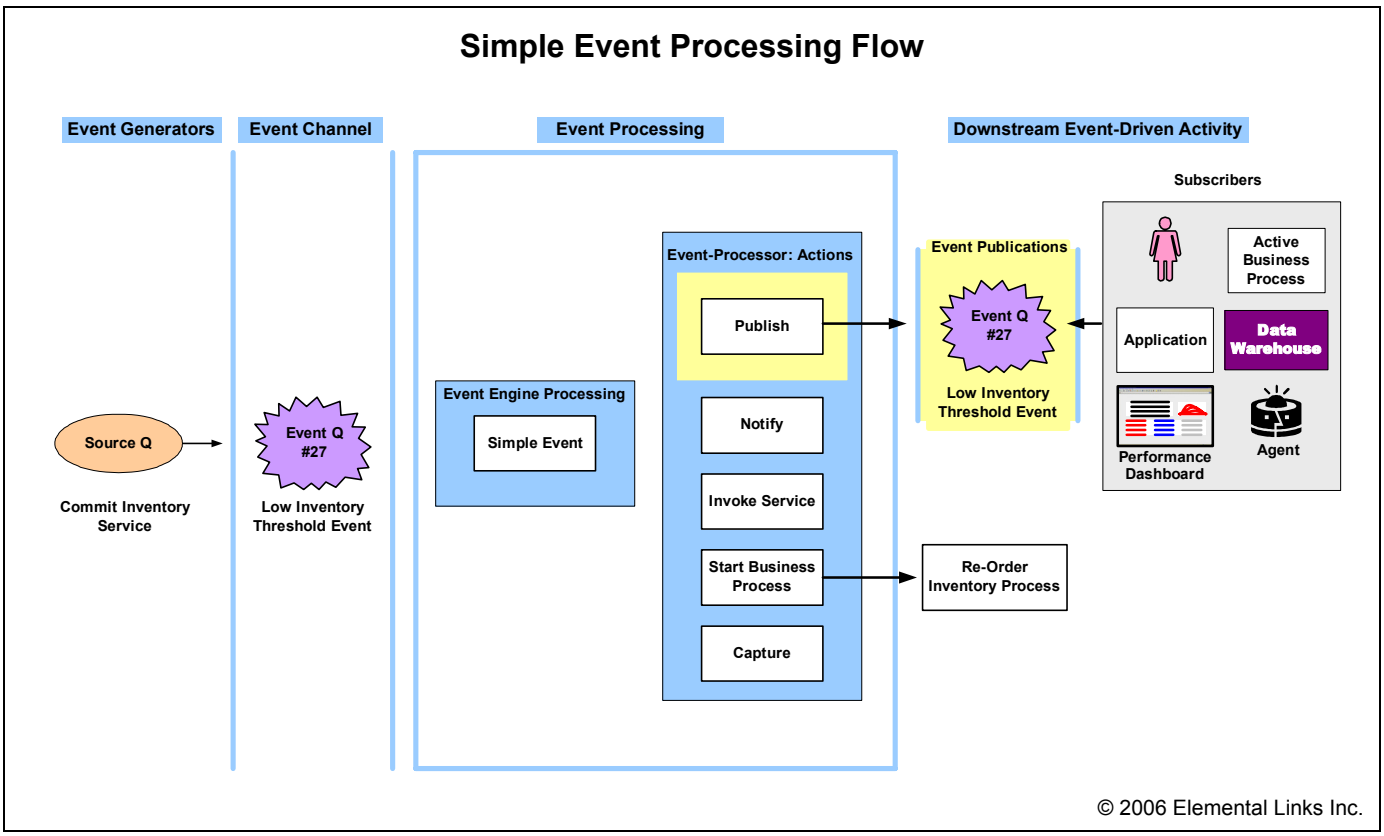

Illustration 1. This illustration shows a simple event processing flow for inventory position optimization.

bookseller immediately commits inventory via a commit inventory service. The commit inventory service shifts inventory from available to reserved, and then checks the remaining available inventory against the optimal inventory thresholds. If the stock falls below the inventory threshold, the commit inventory service generates a Low Inventory Threshold event. This event is shown in Illustration 1 as Event Q, occurrence 27.

The Low Inventory Threshold event is deposited into the event channel and received by the simple event processing engine. The processing rules for this event type (Low Inventory Threshold) dictate two actions. A re-order inventory process is initiated, and the event is published for subscription. The subscribers are the inventory buyer, and the inventory manager's performance dashboard. The re-order inventory process could be a straight-through process, or it could require human review and approval.

STREAM EVENT PROCESSING. In Illustration 2, we show three stream event processing flows for a multichannel consumer electronics retailer. In the first flow, top left hand corner of Illustration 2, an RFID sensor is emitting events every time a product leaves the warehouse. The consumer electronics retailer wants to be informed when high-end products leave the warehouse. To meet this requirement, a local event filter has rules to filter out events for items priced less than $4,000. One event, Event A, occurrence 2, is for a $5,000 plasma TV. This event is reformatted to a standard event format, and placed in the event channel.[5] The simple event processing engine receives the event, and following the rules for high-end products leaving the warehouse, publishes the event. The event is subscribed to by the inventory manager's performance dashboard.

[5] It is an architectural decision if the local event filter should be aware of the proprietary RFID format, or if all RFID events should be transformed to a standard format (event formatter), and then evaluated by a local event filter.

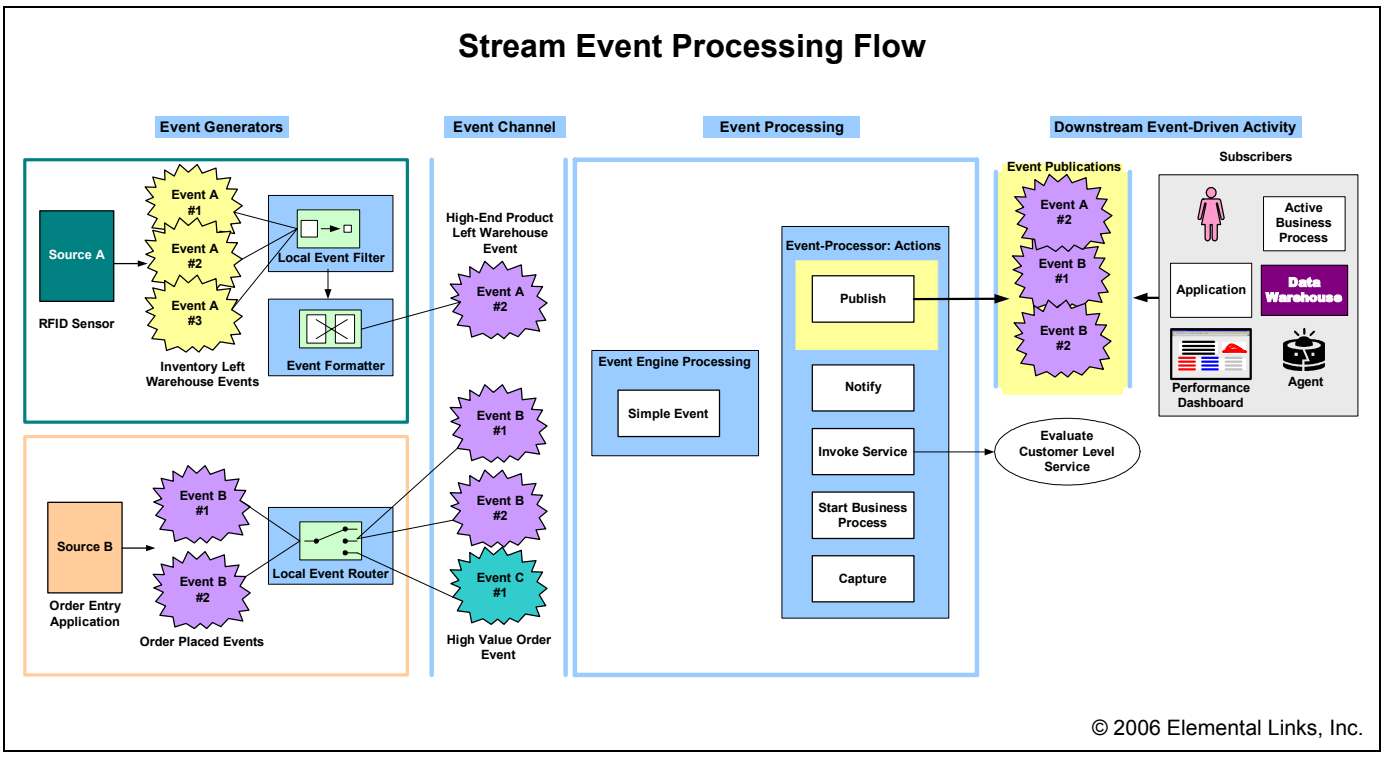

Illustration 2. This illustration shows three stream event processing flows for a multichannel consumer electronics retailer.

The second and third flows originate from the order entry application shown at the bottom left hand corner of Illustration 2. For every customer order placed, an *ordinary* Order Placed event is generated. All ordinary Order Placed events are streamed to a data warehouse, through the event channel, and the publication action of the simple event engine. These events are shown in Illustration 2 as Event B, occurrences 1 and 2.

For high value customer orders, an order totaling more than $1,500, the consumer electronics retailer wants to immediately reward the customer with a promotion in status. To meet this requirement, a local event router evaluates the order total of each ordinary Order Placed event, and generates a new *notable* High Value Order event when the $1,500 condition is met. This event is shown in Illustration 2 as Event C, occurrence 1.

The local router is employed because the burden of event routing or redistribution is on the event processing system, not the source application. The source of the event should only have to say once "this thing happened," regardless of how the event is used in the enterprise.

Once the High Value Order event is generated, it is deposited into the event channel, and processed by the simple event engine. The action associated with the processing rules for this event is to invoke the Evaluate Customer Level service. This flow is also an example of event-driven SOA.

COMPLEX EVENT PROCESSING. In Illustration 3, we show three complex event processing flows for the same multichannel consumer electronics retailer. In the first flow, top left hand corner of Illustration 3, a business-to-business order gateway is supposed to be emitting System Heartbeat events every 15 minutes. The System Heartbeat events inform IT operations the gateway is up and running. The absence of a heartbeat event indicates a failure. If the order gateway is down, business customers are likely to place an order with a competitor.

The complex event engine tracks the timestamp of the last received System Heartbeat event. If 15 minutes have elapsed, the event processing actions associated with non-arrival are initiated. In this case,

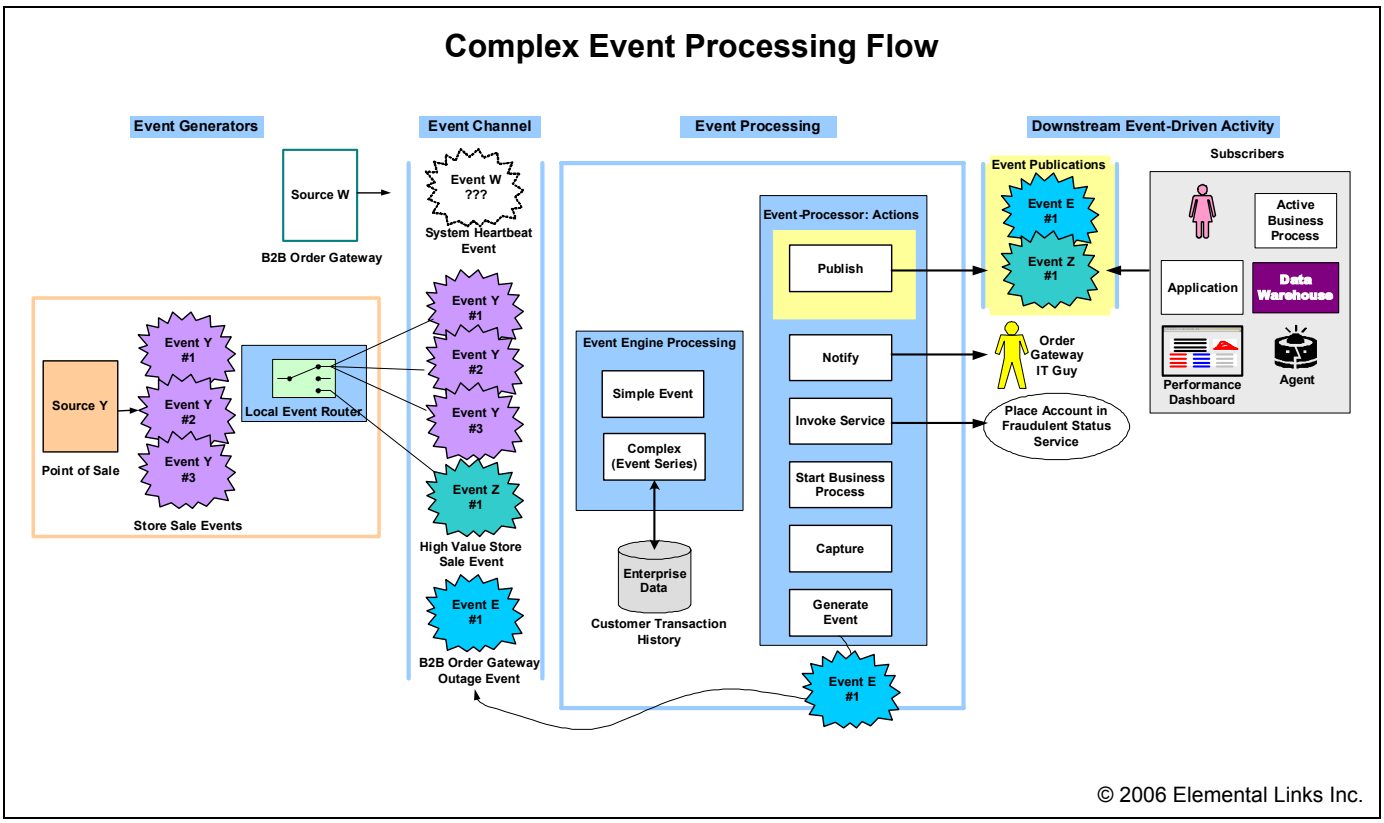

Illustration 3. This illustration shows three complex event processing flows for a multichannel consumer electronics retailer.

the order gateway IT guy is immediately notified (paged) and a new B2B Order Gateway Failure event is generated.

This new B2B Order Gateway event is deposited into the event channel. When the B2B Order Gateway Failure event is received by the simple event engine, a publishing action is performed. The enterprise problem management system subscribes to this event for resolution tracking.

The second and third flows show two variations of fraud detection. Both flows originate with the point-of-sale application shown at the bottom left hand corner of Illustration 3. Similar to the customer order example above, for every store sale, an ordinary Store Sale event is generated (Event Y, occurrences 1-3). These events are evaluated by a local event router, producing High Value Store Sale events (Event Z, occurrence 1) for transactions more than $1,500. All of the Store Sale events, ordinary and notable (high value), are deposited in the event channel.

In the first fraud detection flow, the complex event engine checks for multiple transactions (ordinary store sale events) by the same customer (credit card), over a short amount of time (10 minutes), at different locations over a large distance (20 miles). If these conditions are met, the place account in fraudulent status service is invoked.

In the second fraud detection flow, on receipt of a High Value Store Sale event, the complex event engine does an inquiry on the customer's past purchases to determine if this purchase should be marked suspicious.[6] If the current purchase amount deviates more than 50 percent from largest historical purchase, the event (Z, occurrence 1) is published as suspicious. A customer advocate team subscribes to

[6] To facilitate the enterprise database query (customer order history), complex event processing systems have the capability to cache relevant enterprise information.

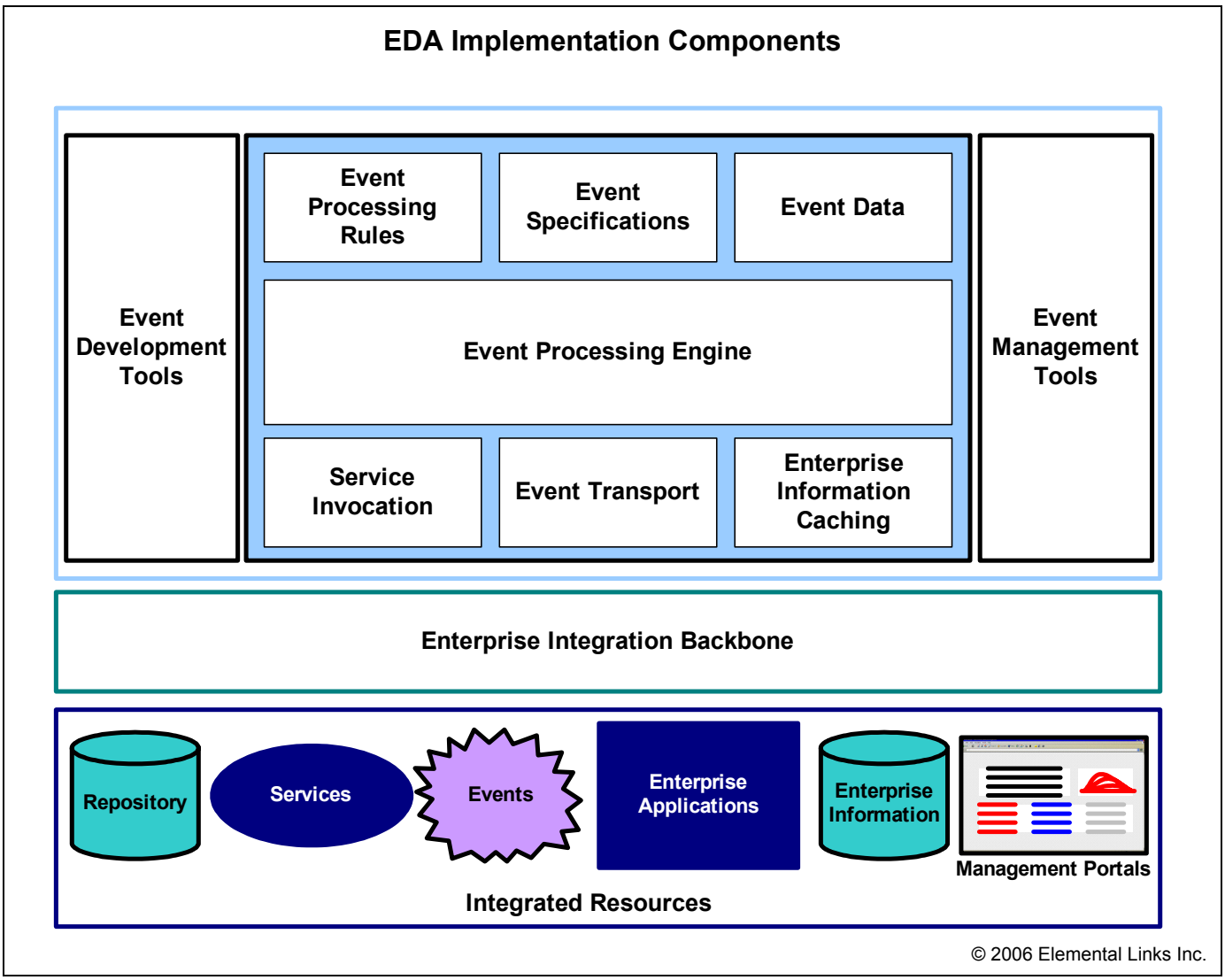

Illustration 4. This illustration shows the major implementation components for an event-driven architecture.

these events and places calls to registered cardholders.

As you can see from our examples, the possibilities are almost limitless. We find it particularly powerful to combine services and business processes with events. The services and business processes can be event generators, event-driven actions, or both.

EVENT-DRIVEN ARCHITECTURE IMPLEMENTATION COMPONENTS

In the event basics and event processing sections, we surfaced many of the implementation components required for an event-driven architecture. Illustration 4 shows these components in a layered architecture manner. The components can be broken out in five categories:

- **Event Metadata.** A good event-driven architecture has a strong metadata architecture. Event metadata includes event specifications and event processing rules. Event specifications must be made available to event generators, event format transformers, event processing engines, and subscribers. While no standards currently exist for event definition and processing notation, it is only a matter of time.

- **Event Processing.** The cores of event processing are the engine and the event occurrence data. Simple event engines are often homegrown. Complex event engines should be acquired from a CEP engine provider.[7] Event occurrence data is typically persisted and retained for audit and trend analysis.

- **Event Tooling.** Event development tools are required to define event specifications and processing rules, and to manage subscriptions. Event management tools provide administration and monitoring of the event processing infrastructure, monitoring of event flows, and visibility into event generation and processing statistics.

- **Enterprise Integration.** An enterprise integration backbone[8] plays a large role in event-driven architecture. Some of the required integration services are: event preprocessing (filters, routes, and transformations), event channel transport, service invocation, business process invocation, publication and subscription, and enterprise information access.

- **Sources and Targets.** These are the resources of the enterprise—applications, services, business processes, data stores, people, and automated agents—that generate events and/or perform an event-driven action.

The topology configurations for these components will vary for each enterprise, based on the event flows, event occurrence volumes, enterprise integration backbone, and location/distribution of sources and targets.

[7] Some of the providers offering event processing engines are: AptSoft, IBM, KnowNow, Progress Software (parent of Sonic), StreamBase, and Tibco.

[8] For more information on creating an enterprise integration backbone, see our Networked Integration Environment reports at http://dx.doi.org/10.1571/bda3-31-05cc and http://dx.doi.org/10.1571/bda4-14-05cc, and our ESB Evaluation Framework at http://dx.doi.org/10.1571/fw7-28-05cc.

CONCLUSION, NEXT STEPS

In this event-driven architecture overview report, we demonstrated the power of event-driven architecture in conjunction with service-oriented architecture, and as a broader architectural strategy. As we've previously shared in our business-driven architecture series, the most viable, agile architectures will be comprised of a blend of architecture strategies, including service-oriented architecture, event-driven architecture, process-based architecture, federated information, enterprise integration and open source adoption. The best choices are the ones that match your business!

Ongoing, we will monitor developments in the event-driven architecture space, including progress on standards, new product offerings, and innovative uses of events and event processing. We will share items that catch our attention in formal research papers, our blogs, and/or conversation.

More immediately, our Enterprise Service Bus Rides℠ series will include tests for simple event processing flows.

ESB RESEARCH REFERENCE LIST

- **"Creating A Blended Architectural Portfolio:** Candidate Strategies for Your Blended Architecture*,"* http://dx.doi.org/10.1571/bda10-14-04cc
- **"Design a Fluid Enterprise Using an Adaptive, Customer Centric IT Architecture:** *Creating a Blended IT Architectural Portfolio to Support Your Customer-Centric Business,"* http://dx.doi.org/10.1571/bda12-9-04cc
- **"The Evolution of Service-Oriented Architecture:** From Integration to Business Scenario Development*,"* http://dx.doi.org/10.1571/soa1-6-05cc
- **"Integration Scenarios and the Networked Integration Environment:** *Pillars of Your IT Integration Strategy for Customer Experience,"* http://dx.doi.org/10.1571/bda4-14-05cc
- **"A New Service-Oriented Architecture Maturity Model:** SOA MM from Sonic Software, Systinet, AmberPoint, and BearingPoint Now Public," http://dx.doi.org/10.1571/psgp11-3-05cc

