3

Using Event Processing in Business Applications

This chapter explores the business implications of the two seminal aspects of event processing: event-driven architecture (EDA) and complex event processing (CEP). Each is valuable by itself, but they're even more interesting when used together as event-driven CEP. Event-driven CEP is the underlying technology for continuous intelligence systems in which computers analyze events as they occur instead of after the fact. The main reason for the recent upsurge of interest in event processing is that continuous intelligence has become practical to use in a wide variety of business situations. This chapter explains the nature of EDA and CEP and describes how they are applied separately and together to improve the timeliness, agility, and information availability in business.

Event-Driven Architecture

"EDA" refers to a particular style of event processing; it is not an umbrella term for every activity related to events. Progressing from the general to the specific (see Figure 3-1), EDA fits this way into the overall scheme of event processing:

1. **Events** Things that happen or, viewed another way, changes in the state of anything. Broadly speaking, everything in the world participates in events, so the concept is too general to help in designing business processes or IT systems.
2. **Business events** Events that are meaningful in a business context. Other events occur in our personal lives, politics, science, sports, the weather, and other realms but they're not directly relevant to business computing. Every application system used in business during the past 50 years has processed business events in a sense, so it's still a very broad area.
3. **Event objects** Discrete reports of events. An address change on a paper form is an event object that a person can use. Computers, of course, use electronic, machine-readable event objects, such as XML documents. When people say "event processing" in an IT context, they usually imply event *object* processing. Intent is essential to the definition of event object—it is intended to convey information, not just store information. As one wag put it, "An event (*object*) is

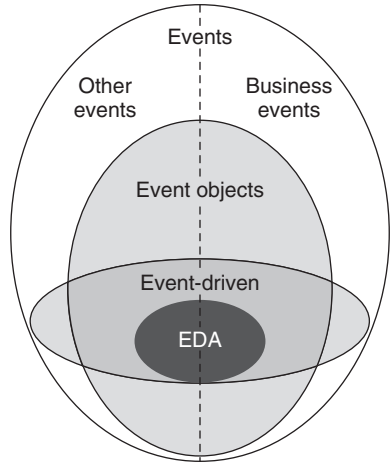

Figure 3-1: Positioning EDA in an event-filled world

a message with an attitude." An amorphous set of data scattered about in various databases, files, electronic documents, and sections of memory doesn't constitute an event object even if it is all related to a real-world business event. That's data about an event, but it isn't a discrete report that can be used to notify another software component. Event objects don't need to convey everything known about the event; they may be only a simple electronic signal. Chapter 7 drills down deeper into the design of event objects for business applications.

4. **Event-driven** The behavior of an entity that acts when it recognizes an event. A person is event-driven when he or she reacts immediately upon finding out that something has happened, perhaps by seeing it or hearing about it. A person can be event-driven without receiving a tangible event object. When software is event-driven, however, it is event *object*–driven, meaning it has received news about an event in a discrete, intentional, electronic form. The notion of being "event-driven" is implemented in many different ways in computer systems (see Chapter 7). Something can be event-driven without being EDA, just as something can be service-oriented (see Chapter 9) without being service-oriented architecture (SOA).

5. **EDA** An architectural style in which one or more of the components in a software system are event-driven and minimally coupled. "Minimally coupled" means that the only relationship between the event producer component and the event consumer component is the one-way transfer of event objects.

A business application implements EDA if it complies with five principles:

- **Reports current events** A notification reports a discrete occurrence as it happens.

- **Pushes notifications** Notifications are "pushed" by the event producer, not "pulled" by the event consumer. The producer decides when to send the notification because it knows about the event before the consumer does.
- **Responds immediately response** The consumer does something in response immediately after it recognizes an event.
- **Communicates one-way** Notification is a "fire-and-forget" type of communication. The producer emits the notification and goes on to do other work, or it may end processing and shut down. It does not get a reply from the consumer.
- **Is free of commands** A notification is a report, not a specific request or command. It does not prescribe the action the event consumer will perform.

The first three of these principles describe what it means to be event-driven. The last two specify the meaning of "minimally coupled." If an event producer tells the event consumer what action to perform, the consumer is event-driven but the application is not EDA because the two parties had to explicitly agree on the function of the consumer. If an event consumer sends a reply back to the producer, it is event-driven but not EDA because the producer is dependent on the consumer. Software engineers have used EDA for years, although they were more likely to call it "continuous processing," "message-driven processing," "data-driven," or "document-driven." An EDA system always adheres to these principles at a conceptual level but the implementation details are more complex, as we'll explore in Chapter 6.

The next four sections of this chapter explore how the EDA principles are used to improve timeliness, agility, data consistency, and information dissemination in business scenarios. Following that, the latter half of the chapter describes CEP and explains how event-driven CEP enables situation awareness.