Event Processing IN ACTION

SAMPLE CHAPTER

Opher Etzion
Peter Niblett

FOREWORD BY DAVID LUCKHAM

MANNING

Event Processing in Action

by Opher Etzion and Peter Niblett

Chapter 1

brief contents

Part 1

The basics

In this book we discuss events. Many people associate the word *event* with a meaningful happening in their lives such as a graduation party or wedding. If you wanted to plan a wedding event, you would need to understand the basics, such as terminology for discussing plans with service providers, the family constraints that affect the nature of the event, and the different types of ceremonies available.

We start this book by explaining the basics of event processing. If you are already familiar with the basics you can browse through part 1, taking note of the terminology that we use in this book. If you're not sure that you know what event processing is all about, this part of the book should fill in the blanks for you.

Part 1 consists of chapters 1 and 2. Chapter 1 provides a bird's-eye view of event processing. It explains what we mean by *events* and *event-driven behavior*. It includes some examples and relates them to different types of event processing. It also positions event processing within the enterprise computing universe and introduces the Fast Flower Delivery example application that will accompany us throughout the book. Chapter 2 discusses the architectural concepts and building blocks required to construct event processing systems.

1 Entering the world of event processing

I am more and more convinced that our happiness or unhappiness depends far more on the way we meet the events of life, than on the nature of those events themselves.

—Wilhelm von Humboldt

Some people say that event processing will be the next big development in computing; others say that event processing is old hat with nothing new. There is some truth in both viewpoints but before exploring them let's clear up the confusion about what *event processing* actually is. This confusion stems from misconceptions and mixed messages from vendors and analysts, and also from lack of standards, lack of agreement on terms, and lack of understanding of the basic issues. This book aims to clear up this confusion.

We use the phrase *event processing* in this book, much as the terms *database management* and *artificial intelligence* are used, to refer to an entire subject area. Our definition of event processing is therefore fairly broad, namely, any form of computing that performs operations on events. We elaborate on what this means as the chapter progresses.

This chapter includes the following:

- An explanation of what we mean by events, using examples from daily life
- Examples of computerized event processing in use, and the reasons for its use
- An introduction to the main concepts of event processing
- The idea of a dedicated event processing platform and its business value

- The relationship of event processing to other computing concepts
- An introduction to the Fast Flower Delivery application that accompanies us throughout this book
- A description of the *Event Processing in Action* website, http://www.ep-ts.com/EventProcessingInAction

Before clearing up the confusion surrounding event processing, we need to look at its background and examples of event processing in daily life.

1.1 Event-driven behavior and event-driven computing

We intend this book to disperse the fog around event processing by supplying a clear, comprehensive, and consistent view of what event processing is. We start by explaining the concept of event-driven behavior in daily life, and then look at how this carries across to the world of event-driven computing. As we go through the book we define the terms we use. You can find an alphabetically sorted list of these definitions in appendix A.

1.1.1 What we mean by events

Before going further we should clarify what we mean by an *event*.

> **EVENT** An *event* is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. The word *event* is also used to mean a programming entity that represents such an occurrence in a computing system.

We give two meanings to the word *event*. The first meaning refers to an actual occurrence (the *something that has happened*) in the real world or in some other system. The second meaning takes us into the realm of computerized event processing, where the word *event* is used to mean a programming entity that represents this occurrence. It's easy to get caught up in a rather pedantic discussion about the difference between these two, but in practice we can safely use the word *event* to have either meaning as it's usually easy to tell the meaning from the context. You should note that a single event occurrence can be represented by many event entities, and also be aware that a given event entity might capture only some of the facets of a particular event occurrence.

As this is such an important term, it's worth commenting on three of the phrases used in the definition. The first of these is "system or domain." In event processing we are chiefly concerned with real-world events–that is, events that occur in real life, such as an order being placed or a plane landing. But the techniques of event processing can also be applied to things that happen in artificial domains such as training simulators, virtual worlds, and similar virtual environments.

Next, the definition includes things that are "contemplated as having happened." It is possible to have events that don't correspond to actual occurrences. To explain what we mean, imagine a fraud detection system being used in a financial institution. Such a system monitors financial transactions and generates events when it suspects

that a fraud is being conducted. These systems can generate false positives, so further investigation is usually required before you can be sure whether a fraud has actually taken place or not.

Finally, the definition contains the phrase "programming entity." Elsewhere in this book we use the phrase *event object*, as this sounds more natural than *event programming entity*, but we need to make it clear that we aren't necessarily talking about an object as defined in object-oriented programming. In some contexts you might encounter events represented as OO objects, but you can also find events that appear in other forms, such as records in a relational database, structures in a language like C or COBOL, or messages transmitted between systems. We have therefore chosen to use the more general expression *programming entity* in our definition.

The word *event* is sometimes used in event processing literature to refer to a type or class of events rather than to a specific event instance. In this book we either use the term *event type* in such cases, or we use the name of the type as an adjective, for example, `Account Overdrawn` event, unless it is obvious from the context that we mean the type rather than a specific instance.

1.1.2 Event-driven behavior in daily life

The event concept is simple yet powerful. Suppose you are working on your laptop in a coffee shop, and since you entered this coffee shop several things have happened: people have come in and out and the waitress has brought you coffee. These are all events, but none of them is particularly exciting. Imagine now what would happen if a robber entered the coffee shop and demanded people's money. This would disrupt the peaceful atmosphere and compel people to react. Suppose that someone surrenders his wallet to the robber, triggering further events. After recovering from the shock, the victim might call credit card companies to cancel his stolen credit cards, which in turn would trigger further activities.

Let's leave scary scenarios and look at how coffee shops work.[1] Some work in a synchronous fashion: a customer walks up to the counter and asks for coffee and pastry; the person behind the counter puts the pastry into the microwave, prepares the coffee, takes the pastry out of the microwave, takes the payment, gives the tray to the customer, and then turns to serve the next customer. Another way to organize things is to have the person behind the counter take the order and payment and then move on to serve the next customer, while other people in the background deal with the pastry and coffee. When both are ready they'll call the customer, or bring the items directly to the table, allowing the customer to sit down, take out a laptop, and write books while waiting. Figure 1.1 illustrates these two approaches.

[1] The synchronous/asynchronous natures of coffee shops were discussed by Gregor Hohpe in his article, "Gregor Hohpe: Your Coffee Shop Doesn't Use Two-Phase Commit." IEEE Software 22(2): 64-66 (2005) http://www.computer.org/portal/web/csdl/abs/mags/so/2005/02/s2064abs.htm.

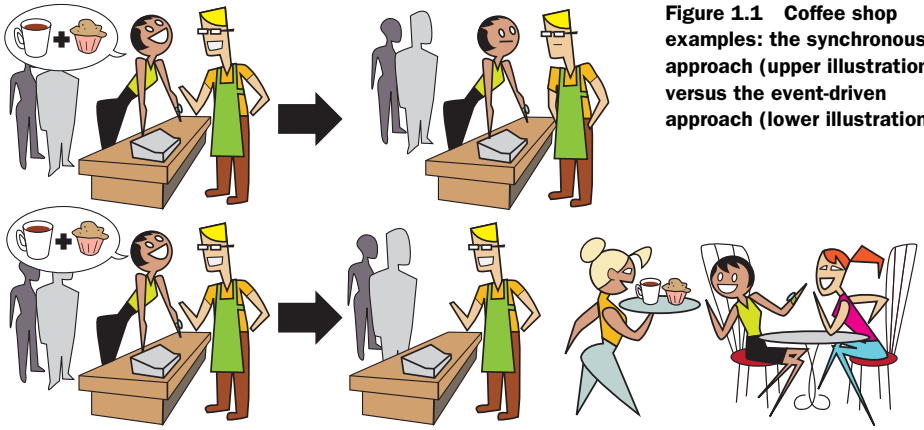

Figure 1.1 Coffee shop examples: the synchronous approach (upper illustration) versus the event-driven approach (lower illustration)

The first approach resembles traditional information systems—you issue a request and wait for the response, typically doing nothing in between. The second approach is asynchronous, and everything is based around events. In particular there is an `order is completed` event which is a combination of two other independent events: `coffee is ready` and `pastry is heated`. The `order is completed` event is a routine event for the coffee shop and customers, whereas the robbery is an unexpected event. Both result in reactions from those involved.

We encounter events all the time in our daily lives. Some events are quite basic—the phone rings, an email arrives, somebody knocks at the door, or a book falls on the floor. Some events are unexpected—the robbery event, coffee being spilled on the laptop creating a short circuit, a late flight causing you to miss a connection. It's also an event when you then find out that your luggage didn't arrive. Not all unexpected events are negative. They can include winning the lottery, getting a large order from an unexpected customer, or finding a significant amount of natural gas under the sea.

Some events can be observed very easily, for example, things that we see and hear during our daily activities; some require us to do something first, for example, subscribing to news groups, or reading a newspaper. In other cases we need to do some work in order to detect the event that actually happened, as all we can observe are its symptoms. As an example, suppose your laptop couldn't connect to your wireless home router. This might be a symptom of an event which had occurred earlier, and identifying the event could require 90 minutes of investigation by a skilled technician. As another example, suppose you recently noticed that your family's consumption of milk had increased, and you needed to add an additional carton of milk to your weekly grocery list. You might reach this conclusion when you ran out of milk in three consecutive weeks, convincing you that this was a consistent phenomenon. In this example `running out of milk` is an observable event, whereas the `milk consumption has increased` event is a higher level event that can be deduced from observing lower level events.

The main reason for learning that events have occurred is that it gives us the opportunity to react to them. In the previous example you might react to the `milk`

`consumption has increased` event by increasing your weekly purchase of milk. Many of the events around us are outside the scope of our interest. Some events are background noise and do not require any reaction, but some do require reaction, and those we call *situation*s.

SITUATION A *situation*[2] is an event occurrence that might require a reaction.

One of the main themes in event processing is the detection and reporting of situations so that they can be reacted to. The reaction might be as simple as picking up the phone or changing the weekly shopping list, or it might be more complicated. If we miss a flight connection there may be several alternative reactions depending on the time of the day, the airport where we are stranded, the airline policies, and the number of other passengers in the same situation.

In these examples, event processing has been performed by human beings. Let's now move from the world of people to the world of information systems.

1.1.3 Examples of computerized event processing

Using events in information systems is not new. In the early days of computing, events appeared in the form of exceptions whose role was to interrupt the regular flow of execution and cause alternative processing to happen. For example, if a program tried to divide by zero an exception event would be raised that enabled the programmer to end the program with an error message, or to perform corrective action and then continue with the computation process. More recently, events were featured in graphical user interface systems (such as Smalltalk or Java AWT) where user interface components (for example, buttons or menus) are designed to react to UI events such as mouse clicks or key presses.

In this book we are mainly concerned with computing events that correspond to events that occur in the real world. These examples show different ways in which automated event processing is used today:

Example 1: A patient is hooked up to multiple monitors that perform various measurements. The measurements take the form of events, which are then analyzed by an event processing system. A physician can configure this system, on a patient-by-patient basis, so that a nurse is alerted if certain combinations of measurements are detected within a certain time period, and so that the physician is alerted if other combinations occur. This example demonstrates the use of event processing to allow timely response to emergency situations, or as part of a personalized diagnosis program.

Example 2: In an airline luggage handling system a radio-frequency identification (RFID) tag is attached to every piece of luggage. RFID readers are located along the luggage route (the sorting device, the cart going to the aircraft, the aircraft's unloading dock, and more). Events from the RFID readers are analyzed to provide exception alerts,

[2] The term *situation* has been used in this way by several people including Asaf Adi and Opher Etzion: "Amit - the situation manager." VLDB J. (VLDB) 13(2):177-203 (2004). http://www.springerlink.com/content/nb1qa1d02vvdre00/.

such as luggage is on the wrong cart; luggage did not arrive at the aircraft; luggage did not even arrive at the sorting device; as well as a routine alert when luggage is approaching the carousel. This example demonstrates the use of event processing for detecting and eliminating errors within an automated processing system. The fact that an event did not occur is one of the most common event processing patterns, called the *absence pattern*, discussed in chapter 9. Some people refer to it as a "non-event."

Example 3: A manufacturing plant with restricted access zones uses RFID tags to monitor compliance with safety regulations. Each person working or visiting the plant carries an RFID tag, and an RFID reader in each zone generates an event when it detects the presence of that person. These events can then be analyzed to detect safety violations—simple ones such as a person entering an unauthorized zone, or more complex ones such as an authorized person working unaccompanied in a zone which requires the presence of two or more authorized people. This example demonstrates the use of geospatial event processing to detect policy violations.

Example 4: A personal banking system allows bank customers to set up alerts when certain events occur, such as when the sum withdrawn from a given customer's accounts within a single day is greater than $10,000; or when a particular investment portfolio has gone up by more than 5 percent since the start of the trading day. This example demonstrates the use of event processing for personalized information dissemination.

Example 5: A financial institution wishes to detect frauds or a financial regulator wishes to catch illegal trading patterns. They collect events from banking or trading systems and analyze them. Certain patterns of activity might suggest that a person is possibly (but not necessarily) in the process of committing a fraud or other illegal activity. This example demonstrates the use of event processing to detect evolving phenomena.

Example 6: An emergency control system informs and directs first responders and people at risk in case of an incident (for example, a fire or the leakage of hazardous materials). In this case the event is a report on an incident, and the main focus of the system is the dissemination of information: who should be informed about what and at what time, given the nature of the incident.

Example 7: An online trading system matches buy requests and sell requests in an auction. It needs to maintain fairness practices (for example, first person to make the bid has first chance). In this case the complexity lies in the matching itself. The events are the buy and sell requests and the matching process is required to match using patterns that apply fairness criteria such as priority based on order, matching conditions, and prior information about the level of risk of the buyer and seller based on trade history. This example demonstrates the use of event processing to dynamically manage business processes.

Example 8: A manufacturing plant management system diagnoses mechanical failures based on observable symptoms. In this case the events are symptoms, describing things that do not work properly, and the main purpose of the event processing is to find the root cause of these symptoms. This example demonstrates the use of event processing for problem determination and resolution.

Example 9: A road tolling system detects the entry and exit points of a vehicle using a toll road and bills the owner. Vehicles are detected based on analysis of a video stream that captures their license plates. Here the main difficulty is extracting and interpreting the vehicles' license plates from the video stream in order to generate the events themselves. This example demonstrates the use of event processing to trigger business processes, where the events need to be obtained as a result of analysis.

Example 10: A social networking site starts a multiparty chat when five people from a group are online. In this case an event occurs when a person goes online or offline. Event processing is used to analyze these events to decide when to start a chat session. This example demonstrates the use of event processing for real-time collaboration.

This list of examples isn't intended to be a comprehensive review of event processing applications. At the end of this chapter we give references, which detail the types of applications in which event processing is employed, and the business value of using these applications. In the next section we provide a classification of event processing systems according to their business use.

1.1.4 Categories of event processing applications

The examples given in the previous section and many others can be classified as shown in figure 1.2:

- *Observation*—Event processing is used to monitor a system or process by looking for exceptional behavior and generating alerts when such behavior occurs. In such cases the reaction, if any, is left to the consumers of the alerts; the job of the event processing application is to produce the alerts only. The patient

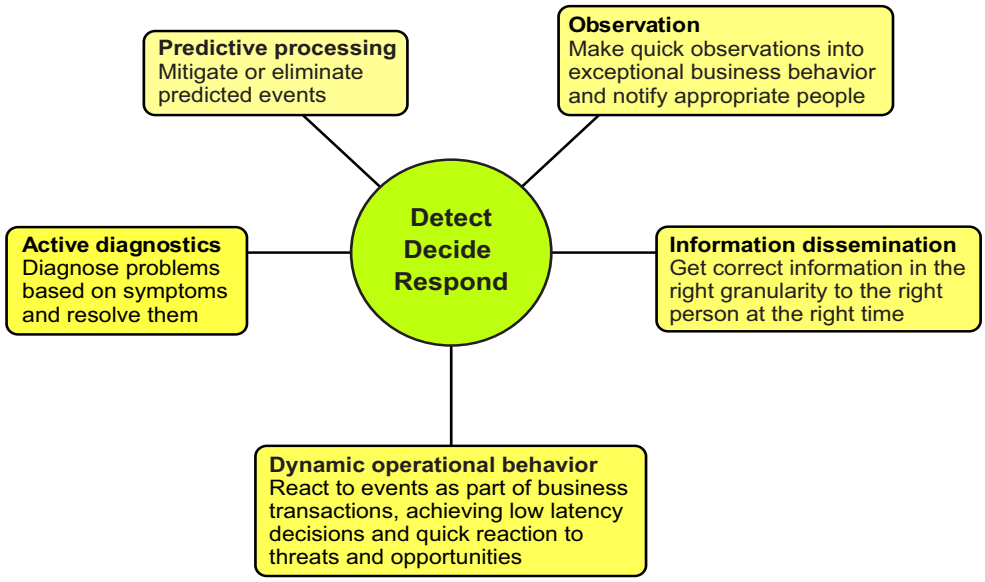

Figure 1.2 Five categories of an event processing application

monitoring system (example 1) comes under this heading, as does the luggage monitoring system (example 2) and the safety regulation compliance system (example 3).

- *Information dissemination*—Another reason for using event processing is to deliver the right information to the right consumer at the right granularity at the right time, in other words, personalized information delivery. Examples of this type are the personalized banking system (example 4) and the emergency system that sends alerts to first responders (example 6).
- *Dynamic operational behavior*—Event processing is often used to drive the actions performed by a system dynamically so as to react to incoming events. The online trading system (example 7) falls into this category, as do examples 9 and 10. In all these examples the output of the application is directly affected by the input events.
- *Active diagnostics*—Here the goal of the event processing application is to diagnose a problem, based on observed symptoms. The mechanical failure case (example 8) is such an example; a help-desk system is another example.
- *Predictive processing*—Here the goal is to identify events before they have happened, so that they can be eliminated or their effects mitigated. The fraud detection system (example 5) is of this kind.

These different classes of use are not exclusive of each other; a specific application may fall into several of the categories that we have listed.

You may notice that we have started using the term *event processing* without explaining what we mean by it. We now introduce the term briefly, though of course it is the subject of the rest of this book.

1.2 Introduction to event processing

In the previous section we introduced the idea of events in real life and their representation in computing systems, and we looked at some of the uses of event processing. We now look at what we mean by *event processing* and describe some of the components of an event processing application and an event processing platform.

1.2.1 What we mean by event processing

In the introduction to this chapter we explained that we are using a fairly broad definition of event processing:

> **EVENT PROCESSING** *Event processing* is computing that performs operations on events. Common event processing operations include reading, creating, transforming, and deleting events.

There are two major themes within this area:

- The design, coding and operation of applications that use events, either directly or indirectly. We saw some examples of applications like this in section

1.1.3. We refer to this as *event-based programming*, although it is sometimes also called *event-driven architecture*.[3]

- The processing operations that you can perform on events as part of such an application. These include filtering out certain events, changing an event instance from one form to another, and examining a collection of events to find a particular pattern. These operations can often result in new event instances being generated. Many of the examples in section 1.1.3 used event processing operations to analyze low-level events and derive further events from them.

It's possible, of course, to write event-based programs without using explicit event processing operations, and people have been doing event-based programming for many years. The three things that distinguish event processing from simple event-based programming, and which open up such a rich range of possibilities are as follows:

- *Abstraction*—Operations that form the event processing logic can be separated from the application logic, allowing them to be modified without having to change the producing and consuming applications.
- *Decoupling*—The events detected and produced by one particular application can be consumed and acted on by completely different applications. There's no need for producing and consuming applications to be aware of each others' existence, and they may be distributed anywhere in the world. An event emitted by a single producing application can be acted on by many consuming applications. Conversely you can arrange for an application to consume events produced by many different producing applications.
- *Real-world focus*—Event processing frequently deals with events that occur, or could occur, in the real world.

Let's pick up on the *real-world focus* idea and explore it in a bit more detail.

1.2.2 Event processing and its relationship to the real world

All the examples that we listed in section 1.1.3 use event processing to detect or report on situations, events that occur in the real world that may require a reaction, human or automated. In this section we use a couple of further examples to explore the relationship between events in the real world and their representation in an event processing system. These examples illustrate two kinds of relationships:

- *Deterministic*—There is an exact mapping between a situation in the real world and its representation in the event processing system.
- *Approximate*—The event processing system provides an approximation to real world events.

Our first example is a system that detects violations of payment on a toll bridge. In this case the situation occurs when a vehicle goes through a highway toll booth (figure 1.3) without paying.

[3] See the book by Chandy and Schulte that we reference at the end of this chapter.

Figure 1.3 A highway toll booth. In this example there is an exact mapping between the real-world events at the toll bridge and their representation in the event processing system.

This can happen in two ways:

1. A vehicle uses an automatic payment system lane without having the device that identifies the car.
2. A vehicle uses a manual lane and manages to sneak through without paying.

In both cases the system detects the situation and sends a picture with the vehicle license plate to the officer on duty on the other side of the bridge. From the event processing perspective we can devise a simple event processing application that detects the fact that a vehicle did not pay and sends this observation along with the picture to a consumer (the officer). This is a deterministic example; the derived event that flows to the consumer implies that the situation has occurred. Conversely when the officer does not receive an event, he or she can assume that no violation has occurred.

In our second example, events in the event processing system only approximate the real world. The setting is a service provider's help desk (figure 1.4). The situation involves a customer who gets so frustrated with the service that he threatens to cancel his account. The service provider wants to detect this situation so that it can assign a skilled customer relations officer to call the customer.

Figure 1.4 A service provider help desk. In this example the mapping between real-world events and their representation in the event processing system is not as clear cut as in the toll booth example.

In this example things are not so straightforward. A human agent can detect frustrated customers by the tone of their voices or electronic messages, but this might not catch all frustrated customers. So in addition we might use an event processing system to look at patterns of user activity, for example, one that detects when a customer contacts the help desk three or more times within a single day. Although we can construct an event processing application that detects this pattern and sends a notification to the appropriate customer relations officer, the detection of this pattern is neither a necessary nor sufficient indication that the situation has occurred. In this case the derived event generated by the system is an approximation to the situation, as there may be false positives and false negatives. Ways of dealing with uncertainty in event processing are discussed briefly in part 3 of this book.

1.2.3 Reasons for using event processing

One question you might ask is why you might want to use an event processing approach in the first place. Here are some reasons:

- Your application might be naturally centered on events, as are many of the examples in section 1.1.3. They involve sensors that detect and report events, and the purpose of the application is to analyze and react to these events.
- Your application might need to identify and react to certain situations (either good or bad) as they occur. An event-driven approach, where changes in state are monitored as they happen, lets an application respond in a much more timely fashion than a batch approach where the detection process runs only intermittently.
- Event processing can give you a way of extending an existing application in a flexible, non-invasive manner. Rather than changing the original application to add the extra function, it's sometimes possible to instrument the original application by adding event producers to it (for example, by processing the log files that it produces). The additional functionality can then be implemented by processing the events generated by these event producers.
- Intermediary event processing logic can be separated out from the rest of the application. This can allow the application to be adapted quickly to meet new business requirements, sometimes by the application business users themselves.
- The application might involve analysis of a large amount of data in order to provide an output to be delivered to a human user or another application. This data can be organized into streams of events which are then distributed to multiple computing nodes allowing separate parts of the analysis to be performed in parallel.
- There are potential scalability and fault tolerance benefits to be gained by using an event-driven approach. An event-driven approach allows processing to be performed asynchronously, and so is well suited to applications where events happen in an irregular manner. If event activity suddenly spikes, it may be possible to defer some processing to a subsequent, quieter time.

We are not claiming that event processing is a universal solution, and that existing batch, online transaction processing, or database-centric applications should be rewritten using event processing technology. However, event processing plays an important part in complementing these approaches. With some applications an event-driven approach will give you a more timely response, better throughput, or greater flexibility through the separating out of event processing logic from the mainstream application code. It doesn't have to be a case of either/or; many applications incorporate a mixture of event-driven and other approaches.

Next we discuss the idea of an event processing platform and its business value.

1.3 The business value of an event processing platform

The examples in section 1.1.3 differ one from another, but all follow the same pattern: events are reported, sometimes by multiple event producers, and some processing of the events is performed. This processing can be made up of several phases, and at the end it creates additional events that are consumed either by humans or by automated processes.

The producers of the events may be different from their consumers. For example, in the luggage management system (example 2), events are produced by the check-in process (which emits an event when luggage is initially deposited) and by the various radio-frequency identification (RFID) readers, which emit events about the movement of the luggage in the system. The events generated by the event processing system are consumed by the luggage control system itself, by airport staff, or even by the passengers themselves.

Many event processing applications keep the logic that processes the events separate from the event producers and consumers, as shown in figure 1.5.

One consequence of adopting this pattern is that you can run the event processing logic on a dedicated event processing platform. Several commercial or open source event processing platforms are available (you can find some references on our website). An event processing platform provides some or all of the following:

- A language for expressing event processing logic
- Tools to design and test event processing logic

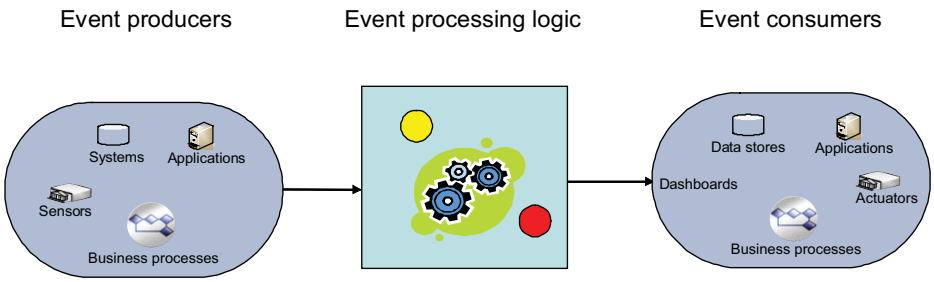

Figure 1.5 The structure of an event processing application, showing the separation of event processing logic from the event producers and event consumers

- A runtime to execute event processing logic
- An event distribution mechanism
- Operational management tools

People frequently ask this question: Is there business value in employing dedicated event processing software, such as an event processing platform, to construct such applications, or is it sufficient to understand the concepts of event processing and apply them using regular programming languages and existing software tools? From observing the current market and reading analyst reports one can draw two conclusions about the current usage of event processing software:

- The use of dedicated event processing software is growing rapidly; analysts claim that this is the fastest growing segment of enterprise application middleware.
- Dedicated event processing software is used in only a small fraction of its potential market.

In this section we briefly explain some of the criteria you can use to decide whether event processing software might have business value in a particular case.

1.3.1 Effectiveness issues

The use of event processing software can substantially reduce the total cost of ownership of an event processing application. The cost reduction comes from the level of abstraction, and is similar to the benefit you get from using a database management system to hold data rather than using a file system. Event processing software typically provides abstractions for handling events that are at a higher level than those provided by conventional programming languages. This can decrease the cost of development and maintenance, and thus the total cost of ownership. In some cases these higher level abstractions can also enable semi-technical persons to author event processing rules. We return to this point in chapter 12, when discussing the event processing of tomorrow.

Another effectiveness issue is business agility. You might need to change your event processing functionality relatively quickly, and it's much easier to make quick changes to functions that are expressed using higher level programming abstractions and that are detached from mainstream application logic. Furthermore, if the event processing logic is separated out in this way, nontechnical users might be able to author and maintain it. This would increase the speed of initial implementation and subsequent changes, allowing the event processing logic to be specified and implemented directly by the application's business users.

1.3.2 Efficiency issues

In some cases the sheer volume of events to be processed, or the complexity of the processing required, necessitate implementations that can scale up and meet high performance requirements. We discuss this issue further in chapter 10, when talking about implementation issues. In these cases, the use of software optimized for this

purpose might be crucial to achieve this goal, as it may not be easily achievable using regular programming.

Applications may also be concerned with other non-functional properties such as reliability, availability, and security. Event processing platforms can provide specific support for these properties, along with operational tools to manage event processing applications when they are put into production.

1.3.3 When to use dedicated event processing software

Generic software platforms are now being used in many areas. To take just two examples, organizations typically use database management systems to manage their data and process queries, and they use message-oriented middleware to connect enterprise applications. In a similar fashion, we think that for many applications it has become more cost effective to use an event processing software platform rather than to implement one's own.

As in any build-versus-buy decision, there are cases when it might not be cost effective to use a dedicated event processing platform. In some cases the event processing functionality required is quite simple, with no special performance requirements, and the usage of event processing within the enterprise is limited. In these cases it might not be worth investing the time or money required to purchase, learn, and assimilate event processing software. It might also be more reasonable to use a self-built solution when the required functionality is unique and is not expressible naturally within any of the languages provided by event processing platforms.

If you decide to build your own event processing application from the ground up, you can still follow the principles described in this book. You can structure your application as shown in figure 1.5 and develop a set of software components, each of them written to perform a specific function, corresponding to the event processing agents that we discuss in this book. These software components can communicate by sending event instances over a messaging system, such as Java Message Service (JMS). You might want to use an enterprise service bus to perform some of the simpler event processing functions, such as transformation, filtering, and routing.

We provide a set of additional references at the end of this chapter for readers interested in reading more about the business aspects of event processing.

1.4 Event processing and its connection to related concepts

Now that we've given a brief introduction to event processing, it's time to see how it relates to some other concepts that exist in the IT environment. We discuss the following areas: business process management (BPM), Business Activity Monitoring (BAM), business intelligence (BI), business rule management systems (BRMSs), Network and Systems Management (NSM), message-oriented middleware (MOM) and stream computing. This section covers a lot of ground at quite a high level, so at the end of the chapter we provide some references for the reader who is interested in finding out more about some of these areas.

1.4.1 Event-driven business process management

Business process management deals with computerized support of modeling, managing, orchestrating, and executing some or all of an enterprise's business processes. BPM software has evolved from workflow systems, and is now frequently included in service-oriented architecture platforms. We discuss the relationship of event processing to service-oriented architecture in the next chapter.

There are two major standards related to BPM software: BPMN (business process modeling notation),[4] an Object Management Group (OMG) standard that deals with the modeling side, and WS-BPEL (Business Process Execution Language),[5] an OASIS standard that deals with the execution side. The synergies between BPM and event processing are listed here:

- The BPM system can serve as an event producer, generating events that report on state changes within the BPM system, which are then analyzed by the event processing system, with the resulting derived events either being returned to the BPM system, or disseminated to other applications.
- The BPM system can act as an event consumer, reacting to situations detected by the event processing system after it has analyzed events from outside the BPM system. An event sent from an event processing system could interact with the BPM system in several ways: the event could trigger a new instance of business process, it could affect a decision point within the flow of a business process that is already running, or it could cause an existing process instance to stop running. We'll give an example from human resources management. In this example a fictional enterprise uses a managed business process to make management appointments. This process consists of a number of activities: advertising the position, identifying candidates, evaluating them, and reaching a decision. Several events could trigger this process to start, such as retirement or promotion of an existing manager. After the process has started it could be affected by a number of external situations. Let's look at two examples of external situations that affect the process. First, event processing could be used to detect that fewer candidates have applied than expected, and this could cause the process to launch an activity to encourage more applications, such as additional publicity or the use of a recruitment agency. Second, during this process there might be an organizational change that eliminates the position (in which case the process should be canceled) or an organizational change made to alter the set of skills required for the job (which might remove some candidates from consideration).

At the time of writing, there are some BPM products that embed their own ad hoc event processing capabilities. In the future we might see more integration between

[4] http://www.omg.org/spec/BPMN/
[5] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

BPM and event processing products, given the increasing importance of event processing to the BPM space. We provide some references at the end of the chapter for readers interested in pursuing this topic further.

1.4.2 Business Activity Monitoring (BAM)

Business Activity Monitoring (BAM) is a term coined by Gartner.[6] BAM software typically tracks key performance indicators (KPIs). For example, a book publisher uses the number of copies sold per month of an important book as a KPI. The BAM concept, in principle, is wider than KPI tracking and may include monitoring of any kind of business related event.

BAM software systems typically contain some event processing functionality. Systems that are centered on KPI tracking perform filtering, transformation, and most importantly aggregation of events. This can be done in batch mode, calculating KPIs at the end of each day, week, or month as appropriate, or in online mode so that the current value of the KPI can be continually tracked, typically on a dashboard. We cover the dashboard concept in chapter 5, while discussing event consumers.

Two trends in the evolution of BAM software will require more event processing functionality: the drive to provide richer types of observation, which necessitates more event processing functions such as event pattern detection; and the need to provide more online observations when data comes from multiple event sources. We anticipate that these trends will lead to more use of event processing software in tightly or loosely coupled integration with BAM software.

1.4.3 Business intelligence (BI)

Business intelligence (BI) is a collection of analytics techniques and software used to help organizations make decisions based on data that they have collected.

Today's BI systems differ from event processing systems in that they are request driven. A typical BI system takes as its input a set of data that has previously been collected in a data warehouse. It analyzes the data retrospectively, and does not respond to events as they happen. This approach is so different from event processing that it makes sense to think of business intelligence and event processing as separate disciplines, dealing with different problems.

We are beginning to see the appearance of software that provides online analytics for decision making. This is a kind of event-driven BI, because it involves versions of the BI analytics that can run online, triggered by events. The term *operational intelligence*[7] is sometimes used to describe this area. Whereas mainstream BI functionality will remain request oriented for the foreseeable future, event-driven BI is useful to some segments of the BI marketplace, and so we expect to see this functionality being included in BI software.

---

[6] http://www.gartner.com/resources/105500/105562/105562.pdf

[7] The Wikipedia entry for operational intelligence provides some explanation: http://en.wikipedia.org/wiki/Operational_intelligence.

1.4.4 Business rule management systems (BRMSs)

Business rule management systems (BRMSs) are software systems that execute rules, typically in the form of *condition-action* or *if-then,* which are kept separate from mainline application logic. This means that the rules can be modified without requiring change to the application code. The rules are expressed in declarative languages and are managed by dedicated software platforms.

There is often confusion between the concepts of BRMS and event processing, mainly due to the fact that some event processing software is based on rule-oriented languages, and the term *rule* is sometimes used to mean the basic processing primitive of event processing (in this book we use the term *event processing agent* instead). However BRMS and event processing systems have fundamental differences:

- Event processing is invoked by the occurrence of events; business rules are invoked by requests made from application logic.
- Business rules operate on states; event processing operates on events but can consult state.
- A business rule system makes inferences from a knowledge base of business rules; the main functionality of event processing is event filtering, transformation, and pattern detection.

These differences result in different functionality, different execution mechanisms, and different types of optimization. Attempts have been made to implement business rules functionality using event processing software and to implement event processing functionality using BRMS software, for example, adding temporal extensions to BRMS to express the equivalent of the event processing temporal patterns that we will examine in chapter 9. It is, however, difficult to optimize both types of functionality in a single implementation.

There are also synergies between event processing and business rules:

- Business rules can be used in the routing and filtering decisions made by event processing software.
- Event processing functionality can be expressed in a rules-based programming style.
- Business rules and event processing rules could be authored by the same person, particularly when the products involved aim their level of abstraction at the business user.
- The occurrence of an event, or the detection of a situation by event processing software, can be used to trigger a BRMS business rule. In this case the BRMS is a consumer of events produced by the event processing system.

These synergies, coupled with the growing quantity of applications that require both event processing and business rules functionality, may serve as a motivation for tighter integration between BRMS and event processing, including common programming models and common product packaging.

1.4.5 Network and System Management

Network and Systems Management (NSM) applications are event driven. One of NSM's major goals is to monitor error events (sometimes called *symptoms*) and analyze them to find their root causes. An underlying problem may give rise to many symptoms, for example, a malfunctioning network router could cause a number of services to fail. Network and Systems Management software uses a technique called *event correlation*[8] to examine symptoms and identify groups of symptoms that have a common root cause. These systems also look for particular patterns among the events that they monitor.

NSM products predate more general-purpose event processing software, and have continued to evolve in parallel with it. They have been focused on management applications, whereas event processing has concentrated more on business applications.[9] They also have different types of users (NSM users are typically system administrators) and different non-functional assumptions.

Some applications (such as Business Activity Monitoring) may require a combination of systems management and event processing applications, and this might be a motivation for developing more synergy between the two areas, but we expect that they will continue to move along separate tracks, at least for the near future.

1.4.6 Message-oriented middleware (MOM)

Message-oriented middleware (MOM) complements event processing, but also partially overlaps it. MOM provides a transport layer that event processing may employ as an infrastructure to implement event channels. The filtering functionality of event processing is similar to the filtering functionality in MOM, as is some of the transformation functionality. There are also differences:

- Whereas an event instance may be represented as a message, a message does not necessarily represent an event. For example, if you send a picture as a message using a MOM system, the picture does not necessarily represent an event.
- Whereas messages in message-oriented middleware can have temporal semantics, such as timestamps, expiry, and ordering, these are not hard requirements. In contrast, temporal properties are fundamental to event processing.
- MOM typically handles each message separately, whereas event processing usually includes functions that operate on collections of events, for example, aggregation and pattern detection.

Event processing may be implemented using a combination of MOM and dedicated event processing components, using the MOM to route event messages between event processing components and to perform filtering. You can find a good description of the

---

[8] A Computerworld article described event correlation in network and system management at http://www.computerworld.com/s/article/83396/Event_Correlation?taxonomyId=16&pageNumber=1.

[9] For an interview with Tom Bishop, CTO of BMC Software, on this topic see http://complexevents.com/wp-content/uploads/2008/11/an-answer-to-a-question3.pdf.

MOM pattern and related concepts in Hohpe and Woolf's *Enterprise Integration Patterns*.[10] This book includes several of the concepts that we cover; it provides a message-oriented view, whereas we describe the concepts from an event processing viewpoint.

1.4.7 Stream computing

The term *event stream processing* is sometimes used as an alias for event processing but sometimes just refers to a subset of it. Stream computing is a much wider subject, encompassing streams that contain data that we would not normally view as events, for example, video streams and audio streams. Stream computing uses an infrastructure based on a dataflow model, which may be used to run various types of applications.

Event processing intersects and synergizes with stream computing as follows:

- You can use a stream processing platform to implement the event processing functionality described in this book.
- Functions implemented on top of a stream computing framework can serve as event producers. For example, in a security application you might use a stream processing platform to extract events out of video streams, which are then forwarded to an event processing platform for further analysis.

Additional reading on stream computing is detailed at the end of this chapter.

We now introduce the Fast Flower Delivery application that accompanies this book and is demonstrated on the book's website.

1.5 The Fast Flower Delivery application

Before presenting numerous concepts and details, we introduce our example event processing application. This example is used throughout the book to explain and demonstrate the concepts and facilities of event processing. We have chosen a supply chain management example centered on flower delivery that illustrates many of the features found in event processing applications, but which can be understood without application-specific knowledge.

This example is demonstrated using the book's website:

http://www.ep-ts.com/EventProcessingInAction

The website provides you an opportunity to see this example implemented using a number of event processing platforms and tools, and lets you experiment with a number of these platforms so you can see how the concepts described in this book are applied in practice.

In this section we provide a specification of our example—the Fast Flower Delivery application—in a detailed, yet informal, type of specification. As we proceed through the book we examine each aspect of this application in some detail. Appendix B contains a more formal specification of the application, using elements of our modeling language. We start with figure 1.6 which summarizes the main event flows in the application:

[10] http://www.eaipatterns.com/

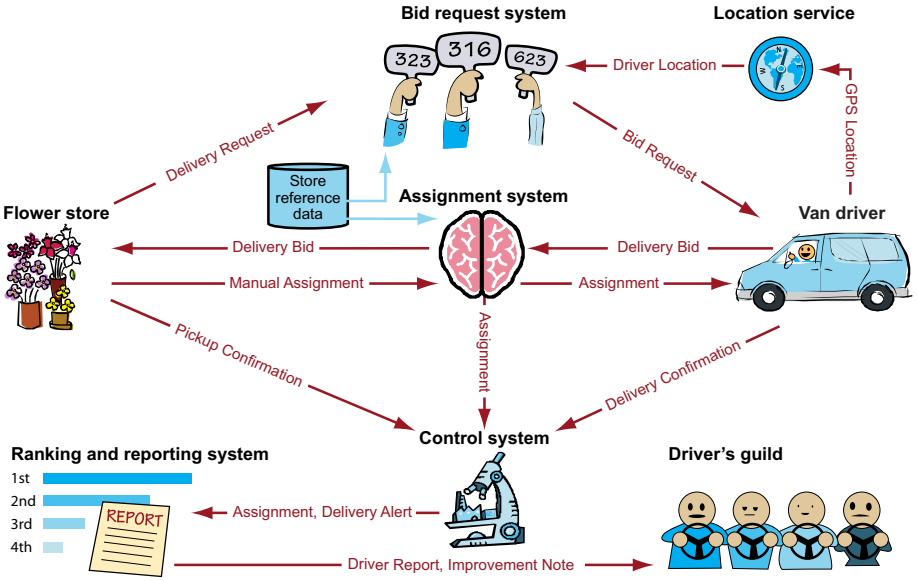

Figure 1.6 Parts of the Fast Flower Delivery application. The arrows represent the flow of events, and the pictures represent the various entities involved in the network.

1.5.1 General description

A consortium of flower stores in a large city has established an agreement with local independent van drivers to deliver flowers from the stores to their destinations. When a store gets a flower delivery order, it creates a request which is broadcast to relevant drivers within a certain distance from the store, with the time for pickup (typically *now*) and the required delivery time. A driver is then assigned and the customer is notified that a delivery has been scheduled. The driver makes the pickup and delivery, and the person receiving the flowers confirms the delivery time by signing for it on the driver's mobile device. The system maintains a ranking of each individual driver based on his or her ability to deliver flowers on time. Each store has a profile that can include a constraint on the ranking of its drivers; for example, a store can require its drivers to have a ranking greater than 10. The profile also indicates whether the store wants the system to assign drivers automatically, or whether it wants to receive several applications and then make its own choice.

1.5.2 Skeleton specification

Let's go through the various phases of the skeleton specification:

PHASE 1: BID PHASE

The communication between the store and the person who makes the order is outside the scope of the system, so as far as we're concerned a delivery's lifecycle starts when a store places a `Delivery Request` event into the system. The system *enriches* the `Delivery Request` event by adding to it the minimum ranking that the store is prepared to accept (each store has a different level of tolerance for service quality). Each van is

equipped with a GPS modem which periodically transmits a `GPS Location` event. The system *translates* these events, which contain raw latitude and longitude values, into events that indicate which region of the city the driver is currently in. When it receives a `Delivery Request` event, the system matches it to its list of drivers. It selects only those authorized drivers who satisfy the ranking requirements and who are currently in nearby regions. A `Bid Request` event is then broadcast to all these drivers.

PHASE 2: ASSIGNMENT PHASE

A driver responds to the `Bid Request` by sending a `Delivery Bid` event designating his or her current location and committing to a pickup time. Note that here the term *request* means a message that asks drivers to bid; it should not be confused with a service request issued in a request-response protocol. Two minutes after the `Bid Request` broadcast, the system starts the assignment process. This is either an automatic or a manual process, depending on the store's preference. If the process is manual the system collects the `Delivery Bid` events that match the original `Bid Request` and sends the five highest-ranked to the store. The store chooses one of these five drivers and creates the `Assignment` event itself. If the process is automatic, the first bidder among the selected drivers wins the bid, and the `Assignment` event is created by the event processing application. The pickup time and delivery time are set and the `Assignment` is sent to the driver.

There are also alerts associated with this process: if there are no bidders, an alert is sent both to the store and to the system manager; if the store has not performed its manual assignment within 1 minute of receiving its `Delivery Bid` events, both the store and system manager receive an alert.

PHASE 3: DELIVERY PROCESS

When the driver arrives to pick up the flowers from the store, the store sends a `Pickup Confirmation` event; when the driver delivers the flowers, the person receiving them confirms by signing the driver's mobile device, and this generates a `Delivery Confirmation` event. Both `Pickup Confirmation` and `Delivery Confirmation` events have associated timestamps, and this allows the system to generate alert events.

- A `Pickup Alert` is generated if a `Pickup Confirmation` wasn't reported within 5 minutes of the committed pickup time.
- A `Delivery Alert` is generated if a `Delivery Confirmation` wasn't reported within 10 minutes of the required delivery time.

PHASE 4: RANKING EVALUATION

The system evaluates each driver's ranking every time that driver completes 20 deliveries. If the driver did not have any `Delivery Alerts` during that period, the system generates a `Ranking Increase` event indicating that the driver's ranking has increased by one point. Conversely, if the driver has had more than five delivery alerts during that time, the system generates a `Ranking Decrease` to reduce the ranking by one point. If the system generates a `Ranking Increase` for a driver whose previous evaluation had been a `Ranking Decrease`, it generates an `Improvement Note` event.

PHASE 5: ACTIVITY MONITORING

The system aggregates assignment and other events and counts the number of assignments per day for each driver for each day in which the driver has been active. Once a month the system creates reports on drivers' performance, assessing the drivers according to the following criteria:

- A *permanently weak driver* is a driver with fewer than five assignments on all the days on which the driver has been active.
- An *idle driver* is a driver with at least one day of activity that had no assignments.
- A *consistently weak driver* is a driver whose assignments, when active, are at least two standard deviations lower than the average assignments per driver on that day.
- A *consistently strong driver* is a driver whose daily assignments are at least two standard deviations higher than the average number of assignments per driver on each day in question.
- An *improving driver* is a driver whose assignments increase or stay the same day by day.

As we have said, this application accompanies us throughout the book, and provides us with a good view into the various functions performed by an event processing system. You will find implementation material for this application on the book's website, which we describe next.

1.6 Using this book's website

The *Event Processing in Action* website is a valuable resource for you, because it bridges the concepts explained in this book with products and open source offerings in the event processing domain. The website's URL is

http://www.ep-ts.com/EventProcessing InAction

On the website you can find the following:

- A specification of the Fast Flower Delivery example application, containing material from section 1.5 of this chapter. As well as demonstrating the concepts of this book, this example helps you understand the various languages that can be accessed from the website.
- Links to the participating languages. Each of the participating languages has a link to a site from which you can download the language, documentation, and an implementation of the Fast Flower Delivery application.
- An open source editor allowing you to create definition elements using the modeling language that is introduced in this book.
- Excerpts from each of the chapters in this book.

We briefly discuss the part of the website which deals with the participating languages in the sections that follow.

THE LANGUAGE-BASED PART OF THE WEBSITE

In order to experience event processing programming you need to choose the language that you would like to use. The website contains examples in several different

languages. You can pick one of these languages and go into it in depth or look at several to get a better feel for the various ways to do event processing.

We include here a few examples illustrating some of the different language styles that you can find on the website. These examples show how to implement the automatic driver assignment step that occurs in phase 2 of the Fast Flower Delivery application.

APAMA EVENT PROCESSING LANGUAGE

The imperative language style is illustrated by the MonitorScript Event Programming Language (EPL) from Apama, a division of Progress Software. The syntax of MonitorScript is similar to Java or C++ as shown in listing 1.1.

Listing 1.1 Example of the MonitorScript event programming language from Apama

```
if isAuto then {
  DeliveryBid db;
  on DeliveryBid(store=dr.store):db within(ASSIGNMENT_TIME){
    assignmentTimer.quit();
    route Assignment(dr.requestId,
                     dr.store,
                     db.driver,
                     dr.addresseeLocationPointX,
                     dr.addresseeLocationPointY,
                     db.committedPickUpTime,
                     dr.requiredDeliveryTime);
     watchForPickUp(dr, db.driver, db.committedPickUpTime);
     watchForDelivery(dr, db.driver);
   }
}
```

1 Detect pattern

2 Send event

In this code snippet, `DeliveryBid` and `Assignment` are event types. The listener 1 reacts when it detects the first incoming `DeliveryBid` and then executes the block of code contained in curly braces { } that follow it. This includes statement 2, which creates and emits an `Assignment` event. This uses attributes from the original delivery request, as well as from the `DeliveryBid`.

ESPER

Esper is an open source event processing implementation, and like several other event processing platforms it uses a stream-oriented language that is an extension of SQL. It can be embedded into Java applications.

```
on pattern[every b=BidRequest(storeManual=true)->timer:interval(2 min)]
insert into AssignmentManual
select d.* from DeliveryBid d where requestId=b.requestId
order by ranking desc limit 5;
```

1 Detect pattern

2 Send event

As you can see, the syntax of this language is similar to SQL but it has been extended with the `on pattern` statement 1 which performs a similar role to the `on` statement in the MonitorScript EPL. A SQL `insert` statement 2 is used to send the `Assignment` event.

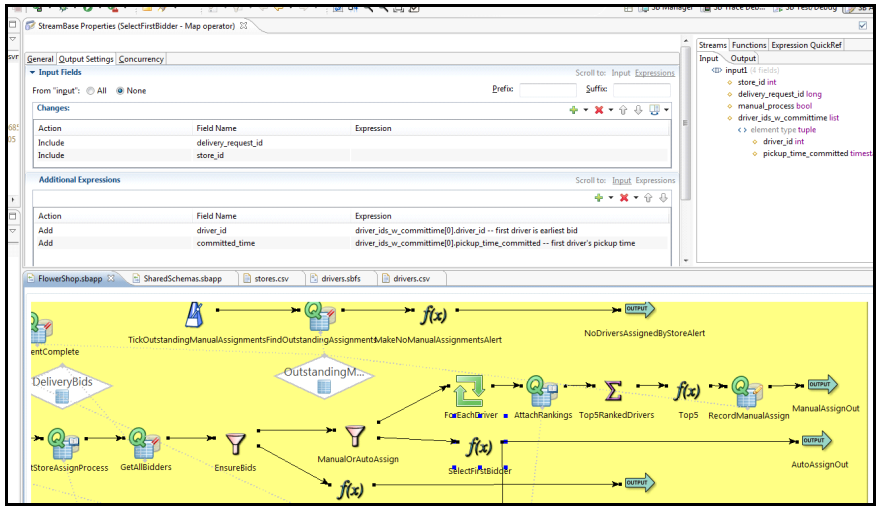

Figure 1.7 The graphical interface for StreamBase StreamSQL EventFlow

STREAMBASE STREAMSQL EVENTFLOW

The StreamBase product also uses an extension of SQL, but programmers are encouraged to use the graphical user interface provided by the StreamBase Studio shown in figure 1.7, known as StreamSQL EventFlow, instead of entering the StreamSQL as text.

At the bottom of this illustration you can see the control flow of the application. The driver assignment is performed by the `SelectFirstBidder` operator, represented by one of the icons in the flow. You configure details of this operator using the tabbed property sheets that appear at the top.

These examples show some of the different language styles that are available; more languages and language styles are displayed on the website. We conclude this chapter with table 1.1, which gives a list of languages currently available.

Table 1.1 Event processing languages classified by language style

Language style	Language or product name	Vendor or open source
Stream-oriented	Aleri	Aleri
	CCL	Aleri/Coral8
	Esper	Open source
	CQL	Open ESB IEP (open source)
	Oracle CEP	Oracle
	RTM Analyzer	RTM Realtime Monitoring
	SPADE	IBM
	StreamInsight	Microsoft
	StreamSQL EventFlow	StreamBase

Table 1.1 Event processing languages classified by language style *(continued)*

Language style	Language or product name	Vendor or open source
Rule–oriented:		
ECA Rules	Amit	IBM (Research Asset)
	AutoPilot M6	Nastel
	Reakt	RuleCore
	RulePoint	Informatica/Agent Logic
	SENACTIVE InTime	UC4 / SENACTIVE
	StarRules	Starview Technology
	Vantify	WestGlobal
	WebSphere Business Events	IBM
Inference rules	DROOLS Fusion	JBoss (open source)
	TIBCO BusinessEvents	TIBCO
Logic Programming	Etalis	Open source
	Prova	Open source
Imperative	MonitorScript	Progress Software
	Netcool Impact Policy Language	IBM

After you have chosen a language style, you can use links on the website to get more information about languages that implement it.

1.7 Summary

Although we process events all the time in our daily lives, traditional software paradigms have not been oriented towards event-driven functionality, but instead have focused on more synchronous request-driven interactions. Many kinds of applications benefit from an event-driven approach. Imagine that you have an employee who sits idle and only works when explicitly told exactly what to do—that isn't an effective way to handle a business. Likewise there are cases in which software is more effective if, rather than waiting to be told what to do, it can detect when an event has happened and can decide whether and how to react.

There can also be benefits in separating the logic that processes events from mainstream application logic, so that an event processing application is made up of event producers, event consumers, and intermediary event processing. Although you can implement event processing logic by hand in a conventional programming language, there are advantages in using a dedicated event processing platform. Such platforms

usually have their own languages and tools, and as we have seen, several different language styles are currently in use.

You should now be familiar at a high level with the basic concepts covered in this book. Reading the rest of the book will give you a more detailed understanding of these concepts and their proper use in constructing event processing applications, starting with the discussion of the technical principles behind event processing in chapter 2.

1.7.1 Additional reading

Chandy, K. M., and W. R. Schulte. 2009. *Event Processing: Designing IT Systems for Agile Companies, 1st ed.* McGraw-Hill Osborne Media. http://www.amazon.com/Event-Processing-Designing-Systems-Companies/dp/0071633502/ref=sr_1_1?ie=UTF8&s=books&qid=1258816511&sr=8-1. This book is an excellent reference on the business motivation for event processing, as well as the positioning of event processing against Service-Oriented Architecture (SOA) and other related concepts. It includes a discussion of terminology, for example, the various interpretations of the term *event*.

Luckham, David. 2002. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Professional. http://www.amazon.com/Power-Events-Introduction-Processing-Distributed/dp/0201727897/ref=sr_1_2?ie=UTF8&s=books&qid=1258816511&sr=8-2. This is the first book published about event processing. It introduced early versions of many of the concepts discussed in this book such as event processing networks and event processing agents.

Arasu, Arvind, et al.2003. STREAM: The Stanford Stream Data Manager. "SIGMOD Conference": 666. http://dblp.uni-trier.de/rec/bibtex/conf/sigmod/AbadiCCCCEGHMRSSTXYZ03. This article describes the STREAM project from Stanford University, which introduced the Continuous Query Language (CQL).

Zimmer, Detlef, and Rainer Unland. 1999. "On the Semantics of Complex Events in Active Database Management Systems." ICDE: 392-399. http://www.informatik.uni-trier.de/~ley/db/conf/icde/ZimmerU99.html. This article is a good survey of related concepts in active databases; several of the concepts we discuss later had their origin in work done in the context of active databases.

von Ammon, Rainer. 2009. "Event-Driven Business Process Management." Encyclopedia of Database Systems: 1068-1071. http://www.springerlink.com/content/p267j78082568086/. This is an encyclopedia entry explaining event-driven business process management.

1.7.2 Exercises

1.1 Provide your own examples of real-life activities which involve events and reactions to them.

1.2 Classify the ten examples given in section 1.1.3 into the categories given in section 1.1.4. Remember that an example can fit into more than one category.

1.3 Provide three of your own examples of the use of automated event processing, and analyze the benefits of automation.
1.4 List all event types used in the Fast Flower Delivery application.
1.5 List all the event processing agents in that application.
1.6 List the event types consumed and emitted by each event processing agent in the Fast Flower Delivery application.
1.7 Describe three additional functions that could be added to the Fast Flower Delivery application.

SOFTWARE ENGINEERING

Event Processing IN ACTION

Opher Etzion • Peter Niblett

Free ebook
SEE INSERT

Event processing apps collect, analyze, and react to events as they occur. They recognize event patterns—from the obvious to the complex, even predicting outcomes such as power shortages or customer dissatisfaction—and respond to them accordingly. In some applications, such as financial trading, fast reaction times are a must.

Event Processing in Action is a ground-breaking book that shows you how to use, design, and build event processing applications. It follows a detailed example to present the concepts and show you the how-tos of both architecture and implementation. The book and its accompanying website introduce the leading free and commercial tools available, along with several language implementations and many examples.

What's Inside

- Event processing concepts and applications
- The event-driven application lifecycle
- How to fit event-driven architectures into your enterprise apps
- Things to consider in your implementation

This book is written for software architects and developers. It requires no previous knowledge of event processing.

Dr. Opher Etzion is the chair of the Event Processing Technical Society and leads the Event Processing team at IBM's Haifa research lab. An IBM senior architect, **Peter Niblett** led IBM's work on the JMS interface definition, and chaired the OASIS Web Services Notification committee.

For online access to the authors and a free ebook for owners of this book, go to manning.com/EventProcessinginAction

"All you need to start building useful software."
—From the foreword by David Luckham

"A progressive approach, with pragmatic examples."
—Christophe Avare, Thales Research & Technologies

"Makes this complex subject very manageable."
—Tony Niemann
Zier Niemann Consulting

"Event processing distilled."
—Paul Benedict
Argus Health Systems

"A richly illustrated deep dive."
—David Dossot
Pug Pharm Productions Inc.

ISBN 13: 978-1-935182-21-4
ISBN 10: 1-935182-21-8
54999
9 781935 182214

MANNING $49.99 / Can $62.99 [INCLUDING eBOOK]