Special Edition Compliments of IBM

EVENT PROCESSING

Designing IT Systems for Agile Companies

K. Mani Chandy | W. Roy Schulte

We at IBM are excited about event processing. As the planet becomes increasingly instrumented, interconnected, and intelligent, event processing is emerging as a key technology for mitigating risk, seizing opportunities, and achieving greater corporate agility.

Smart cities, smart healthcare, smart retail…these are just a few of the areas where event processing is adding more intelligent technology to our lives. To help you appreciate how the "discipline of event processing enables many of the biggest improvements in business processes and IT systems," it is our pleasure to present this minibook excerpt from *Event Processing: Designing IT Systems for Agile Companies* by Dr. K. Mani Chandy and W. Roy Schulte.

Sandy Carter
Vice President, SOA, BPM, and WebSphere Marketing
IBM Software Group

About the Authors

K. Mani Chandy has been a professor of computer science at the California Institute of Technology since 1987 and was a professor in the computer science department at the University of Texas at Austin from 1970 to 1987. He has a PhD from MIT in Electrical Engineering. He has published three books, edited several, and written pioneering papers in computer performance modeling and distributed computing. He is a member of the National Academy of Engineering.

W. Roy Schulte is Vice President and Distinguished Analyst at Gartner, Inc., in Stamford, Connecticut. He was the lead author of the 1996 Gartner report that introduced the term *SOA* to the industry. He also originated research into the field of message brokers, coined the term *business activity monitoring* (BAM), and wrote the first analyst reports on the zero-latency enterprise and the enterprise service bus (ESB). He has more than 25 years of experience spanning user enterprises, IT vendors, and analyst firms. He has a BS from MIT and an MS from MIT's Sloan School of Management.

Event Processing: Designing IT Systems for Agile Companies

K. Mani Chandy
W. Roy Schulte

New York Chicago San Francisco
Lisbon London Madrid Mexico City
Milan New Delhi San Juan
Seoul Singapore Sydney Toronto

The McGraw-Hill Companies

Event Processing: Designing IT Systems for Agile Companies

1 2 3 4 5 6 7 8 9 0 DOC DOC 0 1 9

ISBN 978-0-07-163711-4
MHID 0-07-163711-7

Sponsoring Editor
Roger Stewart
Editorial Supervisor
Janet Walden
Project Manager
Patricia Wallenburg
Acquisitions Coordinator
Joya Anthony
Copy Editor
Lisa Theobald
Proofreader
Paul Tyler
Production Supervisor
Jean Bodeaux
Composition
TypeWriting
Illustration
TypeWriting
Art Director, Cover
Jeff Weeks

Contents

1	Event Processing Overview	1
2	How EDA and CEP Improve Business Processes	8
3	Event Processing Patterns in Business	20
4	Costs and Benefits of Event Processing Applications	31
5	Types of Event Processing Applications	39
6	Positioning Event Processing in the IT World	49

Preface

The discipline of event processing enables many of the biggest improvements in business processes and IT systems in the current era. Unlike many technological advances, event processing has a direct, tangible impact on the lives of business people. It changes the way they do their jobs by giving them better visibility into what is happening in their company and its external environment. It also improves a company's reaction time to unforeseeable situations, reduces the end-to-end elapsed time of business processes, and improves the quality and availability of information.

When you can see what is going on in your sales operations, customer contact center, supply chain, or service network in near–real time through a dashboard on your web browser, you are using event processing. When an insurance company condenses a 25-day process for paying a claim down to 9 days, it is probably using event processing. When an application system can be adapted to support a new set of requirements in days rather than weeks, it is likely using aspects of event processing as well.

Event processing is an increasingly important part of enterprise service-oriented architecture (SOA) and business process management (BPM) strategies. Event processing is synergistic to SOA and BPM, making them more effective and better able to respond quickly to escalating business requirements.

What It's All About

An *event* is just what you think it is—something that happens. We react to events every day, and we're always learning better ways to handle and think about them. Events are all around us and have always been all around us, like the air we breathe and the winds that we exploit to fill our sails and create windmill power. For many centuries, people used sails and windmills without understanding the Bernoulli principle or much else about the scientific properties of airflow. However, thanks to advances in the fields of aerodynamics, engine design, and material science, we now use the power of air currents to fly airplanes, helicopters, and space ships. Even our windmills are more powerful and efficient because they use specially shaped impeller blades made of high-tech materials rather than simple sails or blades mounted on wheels.

In a similar way, companies that deal with millions of events in their daily operations can learn to use events to their benefit. Analysts have an intuitive understanding of how to design business processes that are triggered by events. However, they could accomplish much more if they had a better understanding of how events actually work and used modern design patterns and software tools to harness the power of events.

We need to distinguish the general concept of *responding* to real-world events—which is practiced more or less continuously by every company and person—from the design discipline known as *event processing*. The design discipline of event processing encompasses the principles, reference architectures, design patterns, and best practices that are the subject of this book. Many business analysts, managers, and application architects have only an informal and limited knowledge of this subject. Event pro-

cessing is not included in many business and computer science courses and textbooks, or it is treated as a side issue rather than a focus. Most people are more familiar with the characteristics of non-event-based systems, such as *time-driven* and *request-driven* systems, because those design styles are more commonly used in business applications (although the terms time-driven and request-driven aren't explicitly recognized in many cases).

In some crucial ways, event processing changes the rules of the game. Event-driven design differs from conventional design in a way that can be compared to the difference between left-brain and right-brain thinking.

Why It Matters More Now

If event processing is such a great idea, why hasn't everyone been doing it all along? Event processing is underutilized partly because relatively little data on current business events has been available in digital form until recently. In the past, many events were either undetected or they were detected but not reported in a digital form that could be sent over a network or manipulated by a computer. Now, more events are detected and represented electronically, although, unfortunately, many are still not readily accessible to the people, devices, or IT systems that could benefit from them.

The amount of available event data is rapidly expanding because of the decreasing costs and increasing speed of computers and networks, and the unifying power of the World Wide Web and its communication standards. We are blessed with an explosion of event "streams" flowing over corporate networks—data from websites, enterprise application systems, e-mail systems, cell phones, RFID readers, GPS systems, and a variety of other sensors and devices. This wealth of event data will grow as the cost of the relevant technologies continues to drop and companies create new sources of event data in their operations and the outside environment. Our challenge is to make better use of this data.

The other reason that formal event processing has not been widely used in the past is that competition and customer demands were less urgent. Companies had more time to respond to events than they have today. A person driving 30 miles per hour doesn't need as much advance warning of upcoming curves or obstructions on the road as a person driving at 60 miles per hour. Companies today are operating at a faster pace, so early notification of emerging business threats and opportunities is ever more important. Companies that know how to leverage event processing have an advantage over those that don't.

The goal of this book is to make the knowledge of the event-processing design discipline more widely available. If you understand how event processing really works, you'll design business processes and IT systems that will offer your company better timeliness, agility, and information availability.

1

Event Processing Overview

This chapter introduces you to events, event processing systems, and event-driven architecture (EDA). You'll learn about the relevance of event processing to contemporary management strategies and begin to understand how event processing can improve a company's timeliness, agility, and information availability.

Introduction to Events and Event Processing

An event is anything that happens. A *business event* is an event that is meaningful for conducting commercial, industrial, governmental, or trade activities. Examples include receiving a customer order, making a bank payment, experiencing a power outage, changing a customer address, suffering a network security breach, detecting signs of attempted fraud, hiring an employee, or spotting a change in a competitor's price. Events small and large take place all day, every day in every corner of a company and its environment.

A fundamental characteristic of events is that they cannot be entirely foreseen—a company doesn't know in advance when an event will occur or the details of the event's nature. Customers place orders, competitors change prices, and the dishonest attempt fraud according to their own wants and needs, not on a schedule or plan known to the company or its employees.

A company, person, or animal is said to be *event-driven* when it acts in direct response to an event. The event acts as a stimulus, triggering some reaction. Some events represent threats that must be addressed. For example, a zebra that encounters a lion on the savannah will be event-driven to run away as fast as possible. Or a bank that picks up signs of credit card fraud will be event-driven to stop accepting charges on that card immediately.

Other events are positive opportunities that can be exploited. For example, a sales manager who receives a report that snow blower sales in the New York region are exceeding expectations will be event-driven to order an extra shipment of snow blowers to take advantage of the market conditions. A trader who detects a price differential for a certain commodity in two different markets will be event-driven to buy in one market and immediately sell in the other to profit from arbitrage.

Every company or person exhibits event-driven behavior some of the time, but many activities are time-driven or request-driven. A typical corporate business process has event-driven, time-driven, and request-driven aspects. Event-driven behavior is the best approach when dealing with unpredictable factors and situations. The person or system responding to an event may be guided by a general policy or standard oper-

ating procedure but doesn't know when to implement the policy or procedure or exactly how to apply it to a particular situation until the actual event is detected. Time-driven behavior occurs when the nature and timing of an activity can be planned in advance. Request-driven behavior is appropriate when the nature of the activity is understood and jointly agreed on by multiple parties but the timing is not predictable.

Because the world is full of events, airplanes need pilots, not just flight plans. A flight plan can describe a trip in advance, taking into account landmarks, distances, weight, historical data on fuel consumption, weather forecasts, and reported wind velocity. But a pilot is required to carry out the plan and make ongoing adjustments, as the plane will inevitably encounter other planes and changes in the weather, and perhaps have mechanical problems or experience other unforeseeable conditions while en route to its destination. Most of a pilot's job is event-driven.

All companies and people pay attention to events because the real world is complex and dynamic, and no one has complete foreknowledge of what will happen. Activities that are strongly affected by external factors—things that are outside the control of the company—are particularly suited to event-driven behavior. For example, field sales and service operations, customer contact centers, web-based sales and marketing, other customer-facing activities, supply chain management, transportation operations, and anything to do with competitive markets are fertile ground for event-driven behavior and event processing systems. Internal processes that are more under the control of the company have more time-driven and request-driven aspects, but even they are sometimes event-driven because unexpected circumstances arise everywhere.

Being event-driven in a real-world sense is clearly nothing new. If your company is advised to become an event-driven enterprise, that doesn't mean that your company should start handling events—your company already does that. What it means is that you should improve *how* your business processes and IT systems handle business events. Too often, companies respond to events using outdated processes that were designed for a slower paced, less-automated world. They don't take full advantage of the wealth of event data that is available in enterprise application systems, on the Web, or from other potential sources of event data. People making minute-by-minute, operational decisions often don't get information on current events soon enough. In other cases, people are overwhelmed by the volume of event data that is sent their way so they miss the truly important observations. Most companies could streamline, simplify, accelerate, and improve operations in hundreds of ways by applying the discipline of event processing.

Business Drivers

Companies use the discipline of event processing to improve business performance, not make their IT departments run better (although event processing can sometimes do that, too).

Today's companies face escalating demands, as shown in the first column in Figure 1-1.

Competition is growing as new companies emerge. Globalization is also increasing the number of competitors that any business is likely to encounter in a traditional geographic market. Many markets are consolidating as weaker players lose ground to

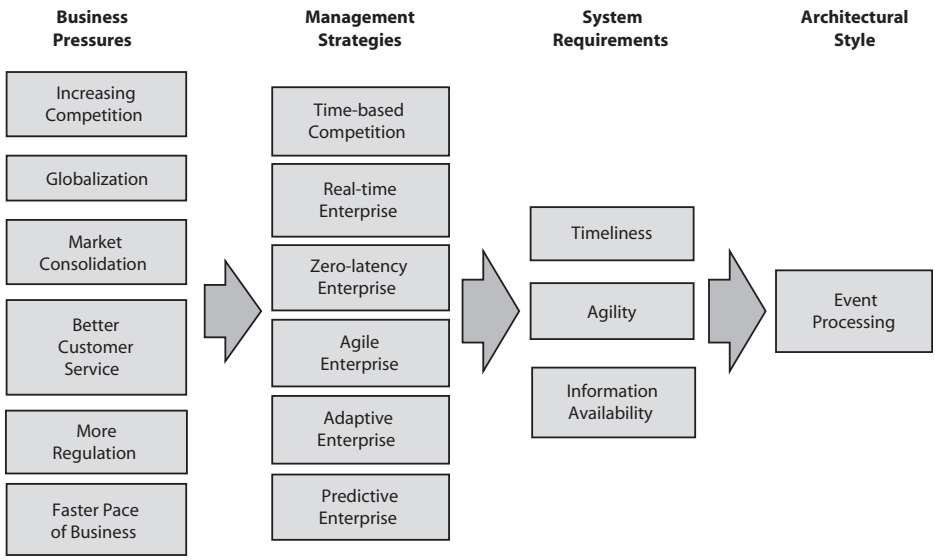


Figure 1-1: Business pressures motivate the use of event processing.

stronger companies. Companies strive to outdo each other by providing faster and better customer service, new products, and, in many cases, lower prices. Customer expectations keep rising, putting pressure on every company to improve its operations. When someone applies for a loan or an insurance policy, she expects a quicker decision and more visibility into the process than was available in earlier days. When a customer submits an insurance claim, requests a repair service, or ships a package or letter, he wants quick action and may want to know the status of his work item at all times. When he places an order online, he expects that the goods will arrive sooner than in the past and wants a tracking number so he can anticipate the delivery time.

New requirements also arise from external regulators and internal decision-makers. Governments and other official bodies impose an ever-growing amount of regulation that requires more reporting and better transparency of operations. Executives, managers, and operational decision-makers on every level want fresher and better information from dashboard displays, e-mail, and other forms of alerts so they can have a clearer picture of what is happening. Operational control activities, such as supply chain management, are far more effective today because of the increasing availability of current information about events. Technology makes it possible for companies to respond to all of these requirements, but it also encourages these requirements to grow even more over time.

These business pressures have inspired the development of numerous modern management strategies (see Figure 1-1, Management Strategies column). Time-based competition, real-time enterprise, and zero-latency enterprise strategies highlight the benefits of timeliness. Agile-enterprise and adaptive-enterprise strategies focus on the importance of flexibility and change. Predictive-enterprise strategies emphasize a combination of information availability and timeliness. Each strategy has its own set

of truths and benefits, but they all ultimately depend on improving a company's timeliness, agility, and information availability (System Requirements column) to varying degrees. For example, fast action isn't much good if you are doing the wrong things quickly, so you need good information as well as timeliness. Doing the right thing quickly is good, but you also need agility so you can adjust your behavior when the business requirements change, because the "right thing" will change over time.

This is not a book on business architecture, so we won't dwell on business pressures and management strategies. We mention them here to explain why people pursue event processing. For the purpose of this discussion, we assume that you and others in your company already know which of these business requirements and management strategies apply to your situation, or you can discover them by other means. We also take for granted that you appreciate the value of timeliness, agility, and information availability. Whether you aspire to these qualities under the umbrella of a specific management strategy or without a strategy, you probably can identify where and how these qualities would provide benefit to your company.

This book provides real-world examples of how event processing can improve your timeliness, agility, and information quality, but it won't attempt to identify every possible application. This book concentrates on explaining *how* to achieve these aspirations through the appropriate use of event processing (see Figure 1-1, Architectural Style column) rather than trying to convince you that these are good things. Note that here, and in many other places in this book, when the phrase *event processing* appears, it refers to the design discipline that implements the principles of event processing—not the mere act of responding to a business event.

The next three sections clarify the terms *timeliness*, *agility*, and *information availability*.

Timeliness

Two types of timeliness are identified here: timeliness that is exhibited by a "low latency" response to a particular input, and timeliness in the form of completing an end-to-end business process in a shorter elapsed time.

Reducing Latency for an Individual Activity

Latency is the time it takes for a system to respond to an input.

Latency in the user interface is one of the most critical factors in the design of an interactive system. Software engineers pay an enormous amount of attention to latency because of the impact that it has on human productivity. When online application systems first became popular in the 1970s, studies showed that applications that returned results to the display terminal within 400 milliseconds (0.4 second) after the user pressed ENTER on the keyboard provided excellent user efficiency for transactional applications. More-common systems with 1- to 2-second latencies were a bit less valuable because people noticed the pause and their rhythm was disrupted in subtle ways. Latency is still an important issue for surfing the Web and for social applications. Modern web designers know that if their site can't display a new web page within

7 seconds of a person clicking a hyperlink, many users will lose patience and abandon the thread they are pursuing (although their willingness to tolerate delay varies considerably, depending on the nature of the task).

Latency is also an issue in other aspects of IT systems and in business activities that don't use computers at all. Reducing latency can help a person or company in myriad ways, most of which are fairly obvious. For most business activities, faster is better up to a point, but beyond that point timeliness has little or no additional value. Event processing and particularly EDA help reduce latency, as we'll explain shortly.

Reducing Business Process Elapsed Time

An insurance company that used to pay 90 percent of automobile damage claims in 25 working days now pays 90 percent of those claims in 9 days. A computer vendor that previously fulfilled orders for PCs in two weeks now ships PCs to customers in 24 hours. Banks that traditionally settled foreign currency trades overnight now settle some currency trades within seconds using real time gross settlement (RTGS) systems.

These are examples of *multistep* business processes—sequences of activities carried out by one or more people, application systems, or business units. The benefit of reducing the end-to-end elapsed time for the process varies. In some cases, the benefit is increased customer satisfaction, leading to more revenue. In other cases, the company saves money by being able to hold less inventory, thereby reducing the carrying costs of inventory. In the case of RTGS, banks are reducing their risk.

If you consider process elapsed time closely, it turns out to be another way of looking at latency. The difference between a multistep process and an individual activity is in the eye of the beholder. Even the simplest activity can be considered a process because it can be decomposed into subactivities. For example, when you click a hyperlink to navigate the Web, you are initiating a short process that will end when the new page is displayed on your browser. To decrease the latency of an individual activity, you need to reduce the elapsed time of the process that constitutes that activity.

Latency is more useful than process elapsed time as a way to think about timeliness where you don't want to deal with the subactivities. On the other hand, process elapsed time is more appropriate when you have reasons to be aware of the component activites. Ultimately, the business only cares about minimizing the time between the beginning and the end of the activity or process, and EDA helps this in a wide range of business situations.

Agility

Enterprise agility is one of the most trite goals in business today, but it's trite for good reason. The world is changing faster than ever. An agile enterprise can readily adjust its behavior in response to environmental or internal changes—such as shifts in customer demand, competitors' activities, regulatory requirements, economic trends, supplier activities, and company circumstances. Agility is different from timeliness because it refers to a company's ability to change its activities rather than just perform them in less time.

Here we identify two forms of agility: instance agility and process agility.

Instance Agility

Instance agility is the ability of an entity to handle each instance (iteration) of a business process differently. For example, each car that comes off an assembly line can have its own unique combination of colors, tires, and audio equipment. In the extreme, no two cars may be alike. Another example is an insurance company cutting and pasting a different set of amendments ("endorsements") to create a custom insurance policy that is tailored to a customer's particular needs. Instance agility is sometimes called "mass customization," and in marketing terms, people speak of "precision marketing" or selling to a "market of one."

Process Agility

Process agility is the ability of an entity to change a whole process to support new kinds of products or services. For example, if customer demand for sport utility vehicles (SUVs) drops, an automobile manufacturer may decide to convert an SUV assembly line to produce smaller, fuel-efficient cars. Another example is an insurance company that enters a new market to insure boats by adapting the process, forms, people, and application systems that it uses to insure cars.

The effect of process agility is more durable than the effect of instance agility. Process changes affect every instance that is undertaken in the revised process. Every vehicle that comes off the modified assembly line will be a car rather than an SUV. By contrast, instance agility doesn't change the entire process; it merely provides ad hoc variability for each instance within the bounds of a defined process.

In subsequent sections, we explain how events play a role in enabling both kinds of agility.

Information Availability

This book addresses only a small part of the information management discipline, specifically focusing on three aspects that are most directly affected by event processing: data consistency, information dissemination, and situation awareness.

Data Consistency

The goal of *data consistency* initiatives is to get multiple business units and their respective application systems and databases to agree on the facts. For a variety of reasons, all companies inevitably have redundant versions of some data regarding customers, products, orders, employees, and other entities. This gives rise to the perennial problem of keeping data consistent ("in sync") across the disparate data stores. When a person, department, or application system receives or generates new data or a revised version of the data, the person or entity updates the local database.The person or system should also forward a copy of the new data to other departments and systems that are keeping track of that data.

Information Dissemination

The need to pass on information is one of the most common situations in business and in life. A person or system detects that something happened and sends a report to one or more people or systems. Some messages report routine events that are expected in the normal course of business, such as the arrival of a shipment. Other notifications report exceptions or abnormal situations, such as when a reverse 911 system notifies all the students on a campus through e-mail or Short Message Service (SMS) text messages on their cell phones that the school will close early because of a snowstorm. A notification that is intended to cause a response is often called an *alert*.

Situation Awareness

Situation (or *situational*) *awareness* means knowing what is going on so that you can decide what to do. The term originated in military applications to describe the goals of advanced command and control systems, but it is relevant in many business scenarios as well. Almost every company has operational activities that run continuously and must respond quickly to changing conditions. Situation awareness implies that you have an up-to-the-minute understanding of the environment and your own internal circumstances. The purpose of situation awareness is to help you make faster and better decisions. Situation awareness goes beyond mere dissemination because it implies that someone or something is able to synthesize multiple sources of input data to develop a holistic picture of what is happening.

Relating Business Drivers to System Design

Timeliness, agility, and information availability can't be achieved simply by speeding up traditional business processes or exhorting people to work harder and smarter with conventional applications. These goals generally require fundamental changes in the architecture of business processes and the application systems that support them. Often this involves more use of the event processing discipline.

Almost every step in every business process today is enabled in some direct or indirect manner by an IT system. You can't have timeliness, agility, and information availability in your business unless you also have timeliness, agility, and information availability in your IT systems. Of course, IT is only part of the solution. Improving the business also requires changes to the non-IT aspects of operations, including people's job descriptions, document and form design, the flow of work, corporate polices and procedures, the organization chart, and corporate governance. These are all part of business *systems* in the larger sense of the word.

Conclusion

Events—things that happen—are a central fact of life for companies, people, and animals. Companies can improve the timeliness, agility, and information quality of their operations if they handle events in a systematic way that leverages advances in the contemporary understanding of how events work.

2

How EDA and CEP Improve Business Processes

This chapter takes a closer look at the two major forms of event processing: simple event-at-a-time event-driven architecture (EDA) systems, and complex event processing (CEP) systems. In particular, we'll explore how event processing can improve the timeliness, agility, and information availability of IT systems, thereby improving how businesses operate.

Event-Driven Architecture

You know that a company or person is considered *event-driven* when it acts in response to an event. This concept of being event-driven can also be applied to software, and when it is, we describe it using the term *event-driven architecture*.

Characteristics of EDA with Simple Event Processing

Software components obviously can't deal directly with real-world business events such as a customer placing an order or a truck delivering a pallet of snow blowers, because these events are abstractions. Software components are capable of dealing with only machine-readable *event objects*, which contain data that report a real-world event. For example, an XML document that contains a purchase order is an event object—it is a record of something that happened or will happen. A lot of the data held on business computers is event data, consisting of event objects in messages, files, databases, in-memory data structures, or some other form. A large majority of all business applications deal with business events, but most of them don't deal with well-designed event *objects*.

When an event object is transmitted in the form of a message, the combination (event data in a message) is a *notification*—the same term used outside of computers for conveying news. For example, an e-mail message that contains an advanced shipping notice for a pallet of snow blowers is deemed a notification for both the IT and non-IT worlds.

Formally, EDA is an architectural style in which one or more components in a software system execute in response to receiving one or more event notifications. Note a subtle distinction here: a person is event-driven when he senses that something has happened, perhaps by seeing it or hearing it. When software is event-driven, however, it is actually *event-object* driven—it has received a notification in electronic form.

Unfortunately, the custom in this field is to overload the term *event*—it sometimes means event in the real-world sense of a happening but it can also refer to an event object, a special type of data. When programmers speak of events, they usually mean notification messages. When business people speak of events, they usually mean real-world happenings.

A software component that detects a business event and sends the notification is called an *event producer*. A software component that receives the notification and reacts is the *event consumer*.

Any large application system is a complex collection of programs and databases, often distributed across disparate computers in multiple locations. The communication among the components in such a system will reflect diverse architectural styles. The majority of the relationships will be request-driven, fewer will use EDA, and fewer still will be time-driven. Software relationships generally mirror real-world relationships: real-world activities that are driven by real-world events are usually best supported by EDA-based software components that are driven by software notifications.

Applying EDA to Business Processes

EDA systems are replacing time-driven systems in a growing number of situations. For example, many companies still use order fulfillment processes that are mostly time-driven and batch-based, with some individual request-driven activity steps. A customer order is initially captured in a request-driven order entry system. This could be an internal application under the control of a salesperson or an externally facing, web-based ordering system used directly by a customer. The order entry system puts the orders in a database or file where they accumulate throughout the day. At periodic intervals, typically once a day in the evening, a batch of orders is passed on to a fulfillment system in an operations department. After the operations department has picked the items from shelves and packed them, more data is relayed to the next functional areas, such as the billing and shipping departments, generally through other batch jobs. This batch-based multistep process is literally an event (object) processing system but it is not using EDA because the software processes the event objects according to a schedule rather than being driven by the arrival of event objects.

This process can be accelerated by redesigning it to use EDA. The customer's order is still captured immediately by a request-driven application, but rather than putting the order data into a file or database, the order capture application immediately sends a notification message to the next step in the process, the operations department. The order fulfillment function is a continuous activity that runs all day, ready to respond as soon as each new item of work is received. In EDA terminology, the order entry application is an event (object) producer and the order fulfillment system in the operations department is the event consumer. It does its part in the process and, in turn, forwards notification messages to subsequent steps in the process, such as the shipping and billing systems.

The Five Principles of EDA

The preceding example demonstrates the basic mechanics of EDA. By definition, an EDA system implements the following five principles:

- **Individuality** Each event is transmitted individually—the event producer doesn't allow a batch of events to accumulate before sending the first one.
- **Push** Notifications are "pushed" by the event producer, not "pulled" by the event consumer. This means that the event producer decides when it will send a notification. This minimizes the time lag between the real-world event and sending the notification message. By contrast, a pull, or request-driven, system always has some delay because the consumer doesn't know when to ask for the event data, so the message must wait for some period of time.
- **Immediacy** The consumer software component does something in response immediately after it receives the notification. (This is always true in a simple-event processing system, although, as you'll learn later, the consumer's "response" in a CEP system sometimes is merely saving the event data for later use.)
- **One-way** In EDA, 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 or any other returning message from the event consumer.
- **Free of commands** A notification, such as a customer order, is a news report, not a specific request or command. It does not prescribe the action the event consumer will perform. The consumer contains the logic that determines how it will respond.

EDA systems are commonly implemented with a publish-and-subscribe communication mechanism. A single notification message may be delivered to one or many consumers without requiring that the event producer send the message more than once. However, this characteristic is optional—not all EDA systems support one-to-many communication.

Advantages of EDA Relative to Time-Driven Systems

The advantages of event-driven systems compared to time-driven systems fall mostly in the categories of timeliness and information availability (see Figure 2-1). An EDA-based process has minimal delays, because each handoff from one activity to the next is immediate. Always-on, EDA applications have become more common as companies strive to improve customer service and reduce the time to recognize revenue. EDA systems exhibit shorter elapsed times for the end-to-end processes, which can lead to greater customer satisfaction, lower inventory carrying costs, and faster payment from the customer. However, in most cases, the process designers, business analysts, and software engineers that design and implement this process don't use the term *EDA* to describe what they are doing. They are more likely to call this "message-driven processing" or "document-driven processing," and the communication pattern that enables it is "push" or "publish-and-subscribe."

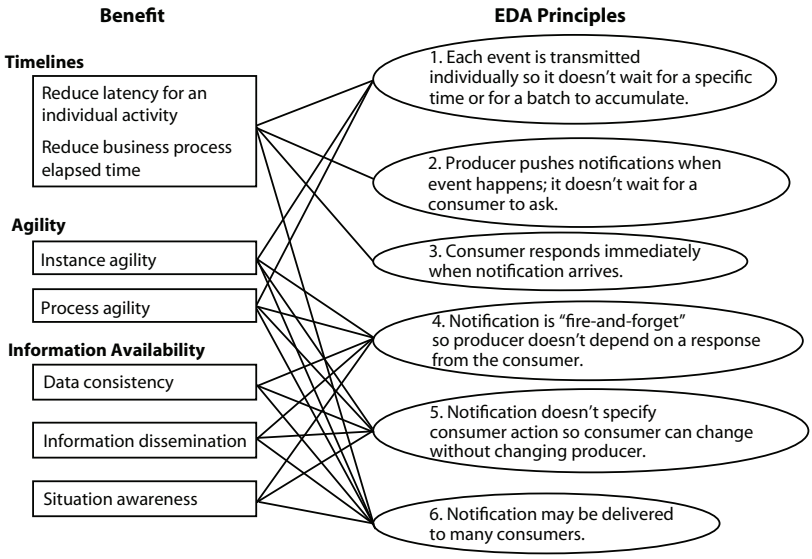


Figure 2-1: EDA benefits and principles.

For the same reasons that EDA improves the timeliness of multistep processes, it also improves the timeliness of data-consistency work compared to time-driven data-consistency. The most common method of synchronizing two or more databases is still to create a batch file in the application that has the most recent information and then transfer those updates to other interested applications in a nightly, point-to-point batch job. This can be done using custom programs or an extract-transform-load (ETL) utility. However, this approach results in a situation where the downstream systems and their databases operate with obsolete data until the batch synchronization job runs.

EDA is clearly superior to time-driven systems for purposes of immediate notification to people or systems. The point of Really Simple Syndication (RSS) feeds, reverse 911, and other kinds of notification or alerting systems is to convey the data as quickly as the communication channel will allow. A time-driven system obviously can't meet the business requirements of many situations involving imminent threats or opportunities.

Event-driven IT systems are only part of the solution for companies that want to reduce the latency of an activity or the elapsed time of a business process. Even before the IT systems are designed, process designers need to re-engineer the whole business process to eliminate unnecessary steps, combine steps, perform steps in parallel, and accelerate each individual step where practical (see Chapter 6 for more on business process management). Where practical, a process should be designed as a straight-through process (STP) by eliminating manual (human-based) activities. STP is increasingly popular in many industries, although they sometimes use other labels to describe it, such as "flow-through provisioning" (in telecommunications), "paperless acquisition" (in the military), "lights-out processing" (in manufacturing), or "no touch" or "hands-free processing" (in insurance). An important aspect of STP is reduc-

ing or eliminating the need to reenter data manually. This saves time, reduces clerical effort and cost, and, most importantly, eliminates an important source of error.

Processes that involve some human steps cannot run as fast as STP processes, but they can still use EDA and, therefore, run faster and have each instance tracked more precisely than batch processes. Human steps in an EDA design can be managed by workflow software that controls work lists and informs people as soon as a new work item is ready to process.

Advantages of EDA Relative to Request-Driven Systems

It's fairly obvious why EDA systems demonstrate better timeliness than batch systems, but it is less obvious why EDA systems have advantages over request-driven systems. Request-driven systems operate in an immediate, message-at-a-time fashion, just like event-driven systems, so they sometimes match the timeliness of EDA systems. However, request-driven systems don't implement the other characteristics of EDA, so they exhibit less agility in certain respects.

In a typical request-driven system, one party (for example, a software component acting as the requester) sends a request message to the other party (a server or service-provider component) and then waits for an answer before it can resume its work. This creates a dependency, because the requester can't finish its task if the server component is not running or if the network is down. Moreover, the team that designs and builds the requesting component must know a lot about what the server component is going to do. If the team that builds the server component changes the server in any way that affects the reply that is sent to the requester, the requesting component must be changed also. This is a type of *logic coupling* that makes the overall system more brittle, because changes can have unexpected (and often bad) and unforeseen effects on the process.

By contrast, EDA systems communicate in only one direction and don't expect a response, hence the label "fire-and-forget." Fewer things can go wrong and it is easier to change or add one component without changing another. For example, a company might want to add a step to its order fulfillment process to check the credit rating of the buyer before an order is filled. This activity could be added to the order capture step using a request-driven design pattern (see top diagram in Figure 2-2). However, the request-driven approach would be unnecessarily complicated. The order entry application would be modified to invoke a credit-checking service, wait for a reply, and then put the order into an accepted order message or a rejected order message for further processing. That would require modifying the order entry application, including recompiling, retesting, and redeploying the component.

Alternatively, the new function could be added to the business process using EDA (see lower diagram in Figure 2-2). The order entry application wouldn't be modified in any way—it would still capture the order and emit the notification (the customer order) as it did previously. However a new, event-driven credit-checking service would be inserted into the business process after the order capture and before the order fulfillment step. It would intercept the order by acting as the event consumer for the order in place of the order fulfillment system. The order entry application that emitted the notification (the customer order) didn't specify that the next step in the process would

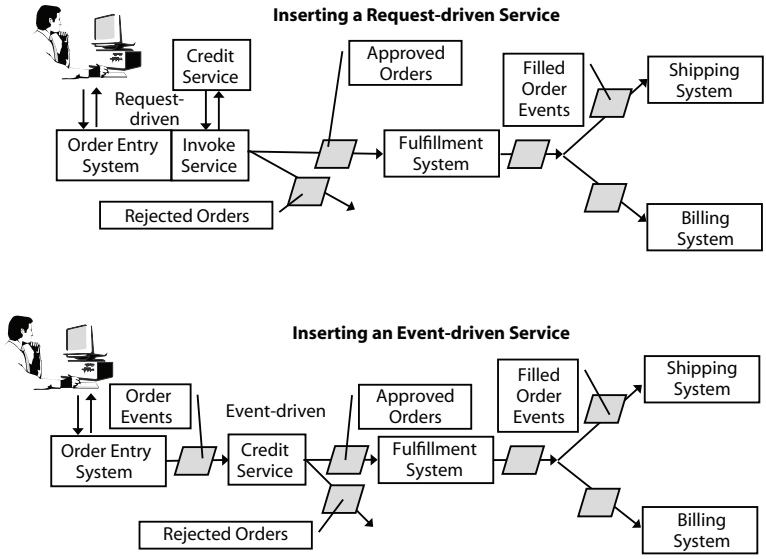


Figure 2-2: EDA facilitates process modifications and agility.

be order fulfillment, so it didn't have to change when the credit-checking step was inserted. The credit-checking service would then verify the credit of the buyer and send an accepted order notification to the order fulfillment system or rejected order notification to a component that handles problem cases. Neither the order entry nor order fulfillment components would require modification when this change was made. EDA makes this kind of "plugability" possible because a notification message is free of commands and the communication is fire-and-forget. By contrast, the request-driven design alternative required invasive changes in the application systems.

EDA systems are agile and can evolve in a piecemeal, incremental fashion. As long as an event producer component emits the right kind of notification, the event consumer component (the recipient) can run successfully. For its part, the event producer is entirely independent of the event consumer, so if the event consumer isn't running, or if it is changed to do something other than what it did previously, the event producer is not affected.

Nevertheless, request-driven interactions are more common than EDA relationships in business systems because the logic of business processes often requires that one component (the requester) gets some information back from the other (server) component. In those parts of the business processes, EDA just wouldn't work. It is a best practice to use EDA where there is a choice, but in many cases, there is no choice.

Complex Event Processing

Much of the industry buzz around events is focused on the second type of event processing, CEP. CEP isn't new, but the way computer systems are used to implement

CEP is new. For most of history, people did CEP in their heads, and even today computers are used in a minority of CEP calculations. However, some critical, high-value business processes depend on automated CEP, and it is spreading steadily out to other parts of the business world.

How Manual CEP Works

CEP has three phases: event capture, analysis, and response. Figure 2-3 shows how it works when people are at the center of the process.

1. **Event capture** People find out about events by observing or interacting with the world. They talk to other people, receive phone calls, use IT systems, read thermometers and other sensor devices, get text messages, navigate the Web, subscribe to RSS feeds, watch television, listen to the radio, and read business reports, mail, newspapers, e-mail messages, and other documents.
2. **Analysis** Each person analyzes the available event data and puts it into context. Part of what they do is "connect the dots"—that is, they distill raw facts about multiple business events into a few higher level insights ("complex events") that represent emerging threats or opportunities. This is the heart of CEP.
3. **Response** The person initiates a response to threats and opportunities that have been identified in the analysis phase. They may undertake some action personally, send a message to another person or other people, kick off a transaction in an IT system, or start up a machine or other device.

Although the label "complex" may initially seem off-putting, complex events actually simplify the data by summarizing what is happening. For example, a sales man-

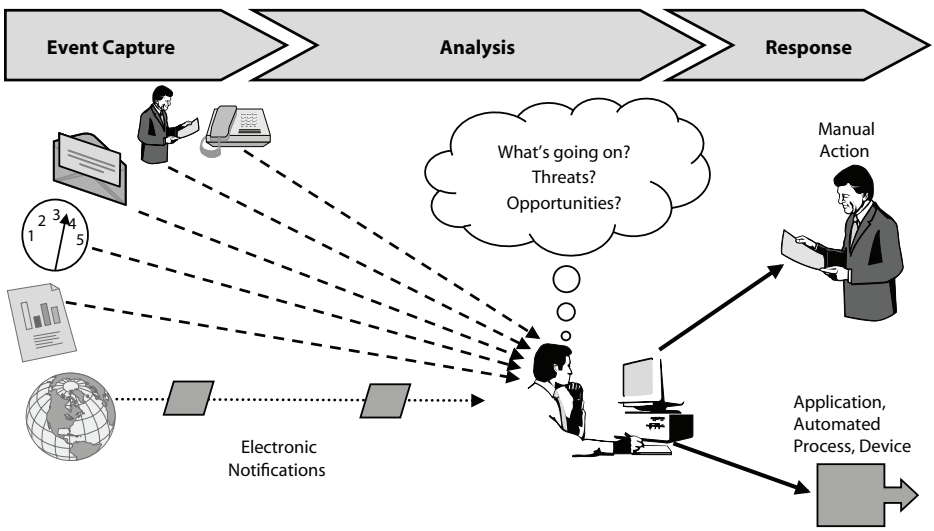


Figure 2-3: Human-based CEP.

ager at a power equipment distributor may receive e-mail reports from 20 stores in his region regarding weekly sales of snow blowers. Think of each sales report as an event object that records a store identifier, today's date, and the quantity of snow blowers sold this week. The manager can extract the data from the 20 e-mails, do a quick computation to determine total sales for the region, and compare the result with historical data to come to the realization that regional sales of snow blowers were 30 percent above average. In event processing terms, the fact that "snow blower sales volume was 30 percent above average this week" is a complex event. This complex event reflects the collective significance of the *base events*, the 20 individual sales reports. This complex event could then be combined with another type of event data—the weather forecast for next week predicting more snow—to derive another complex event that "this is an opportunity situation to sell more snow blowers." The response may be to order an additional supply of snow blowers for immediate delivery. No single event report was important by itself, but when analyzed together, they revealed a meaningful insight that led to a good decision about what action to take today to prepare for next week.

This example should sound familiar to anyone who makes operational decisions or who builds operational intelligence applications for use by others. We've described a mundane decision-making scenario using event processing terms as a way of demonstrating the commonsense principles that underlie CEP. Most automated CEP systems involve many more than 20 event objects, and they come in much faster—every few seconds or even a few milliseconds apart. However, we can explain automated CEP using the same scenario.

Automated CEP

Automated CEP follows the same three phases as manual CEP, except each phase works in a slightly different way (see Figure 2-4).

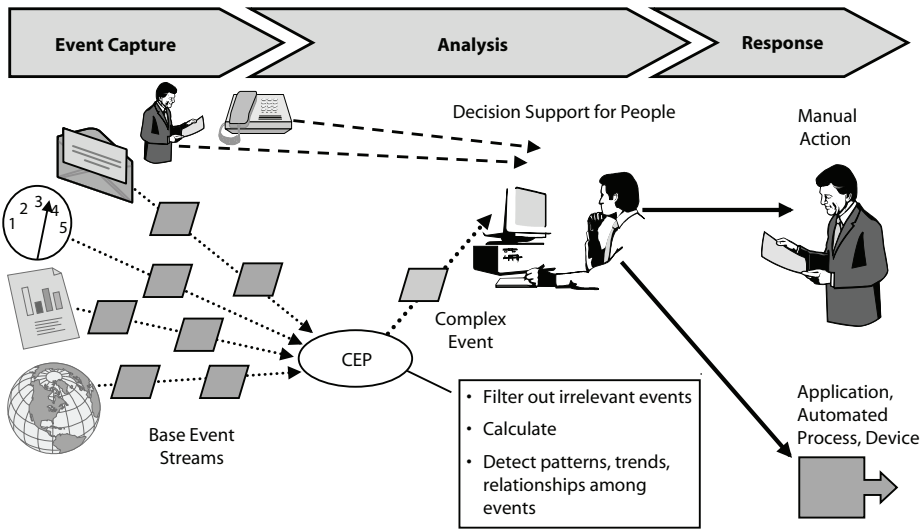


Figure 2-4: Partially automated CEP.

Event Capture

An increasing amount of business event data is available in digital form through a company's network or the Internet. Web-based RSS news feeds convey reports that used to come from newspapers, television, and radios. New types of market data are distributed from exchanges and clearinghouses. More of a company's internal functions are automated in application systems that can be tapped to provide near–real-time data about what is happening in sales, manufacturing, transportation, accounting, and other departments. Bar code and radio frequency identification (RFID) readers pump out reports on the location of specific items. Web page scrapers can poll a competitor's website every 10 minutes to see if the company's prices have changed. Some of this data wasn't generated in the past. Other data was available to people but computers couldn't read it. Now, however, more data is produced and most of it is electronic so it can be handled by automated CEP systems.

Analysis

CEP software can be used to handle some or all of the analysis phase. CEP software is any computer program that can read, write, delete, or perform calculations on two or more event objects. It can be packaged as a stand-alone CEP agent running on its own; it can be implemented as a section of code intertwined in a larger application program; or it can be built into a software framework or business event processing (BEP) platform that can be extended by a software developer to construct a complete application. CEP software is technically a type of *rule engine*. (See Chapter 6.)

CEP software may do the following:

- Read through all the incoming base event notifications and discard those that are irrelevant to the task at hand (called filtering, or screening, the data)
- Calculate totals, averages, maximums, minimums, and other figures
- Use predefined rules to look for patterns in the base events, such as trends, relative timing (the order in which events took place), or causal relationships (an event that led to another event)

The CEP computation produces one or more new complex events that summarize the significance of the available base event data. This is connecting the dots in a high performance, sophisticated fashion. Most key performance indicators (KPIs) used in business situations are actually complex events (but not all complex events are KPIs). Complex events may be *reactive*, summarizing past events, or *predictive*, identifying things that are likely to happen based on what has occurred recently compared to historical patterns.

Complex events may be used for decision support to help a person make a faster or better decision, or to trigger a response without human involvement in the analysis phase.

Decision Support

In the majority of cases, CEP software is used for decision support. It can't do the whole job of analyzing the situation by itself, so it is used to do some of the prelimi-

nary calculations. In the example of the power equipment distributor described earlier, the CEP software would read the sales reports from the 20 stores, compare it to sales history data, and produce the complex event that said "snow blower sales volume was 30 percent above average this week."

This event could be transmitted to the sales manager through a web-based business dashboard. For example, the complex event data could be represented on a satellite map, bar chart, or line plot, or it could be used to turn a "stop light" from green to yellow or red. In other cases, the CEP software puts the event into a message and sends it to a report-writing system, a spreadsheet utility, or another application system that will look up some additional data or perform some further calculations before sending the information to the appropriate person. Or perhaps an alert could be sent through e-mail, an SMS message, a phone system, or another notification channel.

In our example, the sales manager would never have to look at the individual sales reports; he could move directly on to the next phase of analysis, which involved considering the weather forecast. The CEP software has helped him get to a faster or better understanding of whether a threat or opportunity exists by offloading the mechanical aspects of event computation.

FULLY AUTOMATED ANALYSIS

Some decisions can be fully automated because the processing logic and business rules can be expressed in explicit algorithms that CEP software is capable of executing. This works only if the factors that go into the decision are well understood and business decision-makers, business analysts, and software engineers have worked together to articulate the processing rules and implement them in the CEP software.

Response

The response to automated or partially automated CEP analysis may be manual, partially automated, or fully automated. To trigger an automated response, the CEP system may kick off a transaction or service-oriented architecture (SOA) service in an IT system, initiate a business process through a software orchestration engine, or send a signal to an actuator to start a machine, lock a door, increase the heat, or perform some other action (see Figure 2-5).

Full automation of both the analysis and response phases is especially important in scenarios that call for an ultra–low-latency response. For example, CEP-based algorithmic trading systems in capital markets can make some buy and sell decisions for stocks or foreign currency in less than 5 milliseconds without human involvement. A person cannot think that fast or even type one character on a keyboard in 5 milliseconds, so people must be left out of the process. Some aspects of business have become similar to an "arms race," in which small differences in response time can lead to major financial gains or losses.

CEP Benefits

Partially or fully automated CEP improves a company's situation awareness and its ability to behave in an intelligent sense-and-respond manner. It offers four key benefits:

- **Improved quality of decisions** Computers can extract the information value from dozens, thousands, or millions of base events per second in real-world applications (as long as the events are simple). By contrast, a person can assimilate only a few events per second and thus cannot consider nearly as many factors when making a decision.
- **Faster response** Partially automated decision-support CEP systems save time because people don't have to do the manual calculations. Fully automated CEP systems are even faster because they don't have to wait for people to read and respond to the information.
- **Preventing data overload** CEP systems reduce the volume of unwanted, unnecessary data ("information glut") presented to people. In some cases, a CEP system may run for hours or days, turning millions of base events into thousands of complex events before detecting a complex event that must be brought to the attention of a person. CEP systems are often used to implement management-by-exception (MBE) strategies. People are disturbed less often, so they can reserve their attention for the few situations in which their involvement is important.
- **Reduced cost** CEP systems offload the drudgery of repetitive calculations and pattern detection comparisons from people to computers. This reduces the amount of human labor needed to analyze the data. For example, the spread of algorithmic-based program trading in capital markets has reduced the number of human traders operating on certain categories of investments.

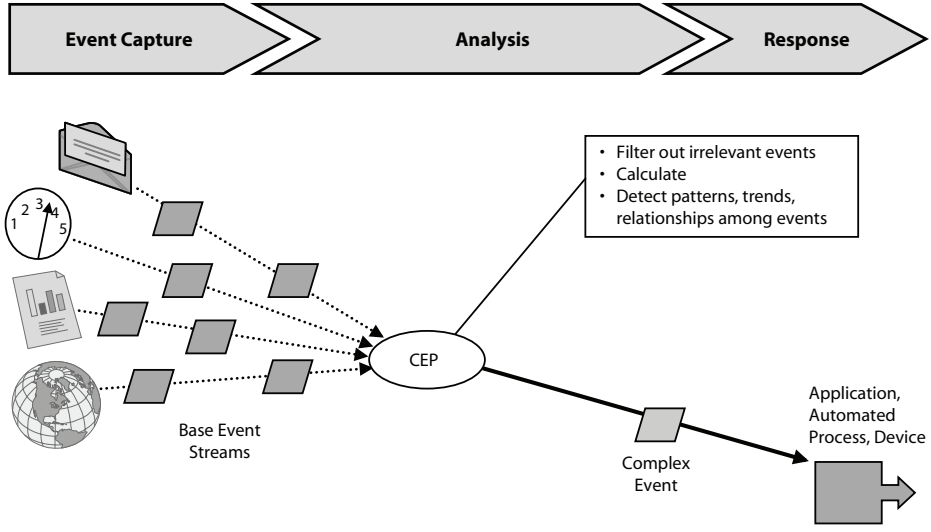


Figure 2-5: Fully automated CEP system.

Applying CEP in Business

Automated or partially automated CEP systems are applied in diverse ways in modern business situations.

Application Styles

Some CEP systems, such as the snow blower sales management example used in this chapter, are used for operational intelligence purposes. We consider them to be a type of *business intelligence* (BI) system, although they differ from traditional analyst-oriented BI systems because of their focus on immediate, operational decisions. (See Chapter 6 for more discussion on the relationship between CEP and BI.)

Other CEP systems are used to monitor business processes that are controlled by *business process management suites* (BPMs) or workflow software. However, most CEP systems are unrelated to BPM. Some are used in high-value, ultra–low-latency applications in the financial services industry. Others are used to monitor complex activities, such as supply chains, transportation operations, gambling casinos, hospital emergency rooms, factory floors, web-based gaming systems, and transportation management.

All the examples mentioned so far are *business event processing* (BEP) systems. A BEP system deals with business events, in contrast to other types of event processing systems that deal with technical or scientific events. BEP is the fastest growing type of event processing because companies are just starting to recognize the advantages of formal event (object) processing for business purposes. Business people who use the system must be actively involved in the development of BEP systems because they have a clear understanding of the requirements. Some BEP software tools even allow power users to adjust the application directly without requiring traditional software programming. Most earlier CEP systems dealt with system events that occurred within computers, networks, or automated devices. CEP was a key technology in network management, computer system management, and military applications by the 1990s, although the term CEP was not used until recently.

The CEP discussion so far has focused on systems that operate in EDA mode. However, that is not the only option. CEP can also be used in an offline time-driven mode or an online request-driven mode. These are described in the next chapter.

Conclusion

The contemporary discipline of event processing is based on two big ideas. The first is EDA—sensing and responding to individual events as soon as possible. The second is CEP—developing insight into what is happening by combining multiple individual data points (event objects) into higher level complex events that summarize the collective significance of the input data. In the next chapter, we'll take a closer look at how companies use these two ideas in their work.

3

Event Processing Patterns in Business

Many types of interactions occur among people and components in IT systems. Information flows and tasks are executed through these interactions. Chapter 1 introduced the basic types of interactions—time-driven, request-driven, and event-driven interactions. Each type offers some advantages over the others, so most IT systems employ combinations of the basic types. This chapter discusses the differences between the types of interaction in detail.

These differences highlight the fundamental concepts that are essential for understanding how to design in the event-driven architecture (EDA) style. A comparison of different types of interactions and a review of the historical contexts in which these interactions flourished can help us understand how different software styles developed in the past and where they are going in the future. Arguments about these architectural styles can get heated—discussions about whether a product is in the EDA style or whether service-oriented architecture (SOA) can be event-driven can get contentious. We can get to the heart of these issues by asking a few questions about the most elementary operations: How do people interact with each other? How do we expect software components to interact? Elucidating the differences between basic types of interactions gets to core issues without using terms that may be considered either pejorative or complimentary.

This chapter also covers the suitability of each type of interaction for different business problems.

Categories of Event Processing

People with expertise in different areas of IT have different views of event processing: experts in databases, application servers, message oriented middleware (MOM), sensor networks, control systems, user interfaces, hardware architectures, and programming languages each brings his or her own valuable, but different, perspectives to event processing. A discipline of event processing that spans all branches of IT is only just emerging. A foundation for such a cross-discipline must start with the basics, and there is nothing more basic than an interaction. We identify features common to event processing in human organizations and IT by studying interactions among people in enterprises and among IT components. Indeed, most features of IT event processing systems are mirrored in human organizations.

Event processing is used for many purposes, including the following:

- **Business intelligence** Event processing applications analyze repositories of event data to help people understand the behavior of an enterprise and its environment.
- **Sense and respond** Event processing applications enable enterprises to sense and respond to events. Sense and respond systems provide users and businesses with situational awareness, the ability to predict future events, and assistance in responding to threats and opportunities.

These goals are not mutually exclusive; indeed, you will see that event-driven interactions help meet these goals and others as well. The following sections discuss interactions among people in organizations and then study the same interactions among components of IT systems. The term *agent* is used to indicate both a person and a component of an IT system. A study of interactions among agents is thus a study of how people and IT components interact. The humanities and social sciences study people and how they interact; here we draw upon only a few observations from these fields to help us understand event processing in its larger context.

People in an enterprise have expectations—formal or informal—of each other and of functional units such as marketing and finance. An enterprise functions because people have, and live up to, shared expectations. People and IT components communicate with one another using different types of interactions to meet different types of expectations. Expectations include being informed about critical events. A mother on a business trip expects her family at home to inform her if something important and unexpected happens; the president of a country expects to be informed when an attack occurs; a CEO expects to be notified when a factory burns down. People communicate partly to meet shared expectations. Likewise, components of IT systems create and communicate event objects to meet designers' expectations about component behavior. In software engineering, an expectation of a component is manifested as a formal *contract* specifying how that component must behave. When we study interactions, we will also analyze the expectations, both explicit and implicit, that people and IT components have of each other.

Business Intelligence

Enterprises improve by learning from history: The US National Transportation Safety Board analyzes flight data and voice recorders to determine what went wrong when a plane malfunctions or crashes. Hedge funds analyze historical records to evaluate trading strategies. The raw material of history is a record of events. A century ago, events were recorded in people's memories and on paper. Today, decreasing costs of sensors and electronic storage allow vast amounts of event data to be collected and stored digitally. Business intelligence (BI) systems and machine learning technologies enable event data to be analyzed efficiently. These technological developments have led to increasing reliance on IT to understand the behavior of the enterprise and its environment. (Chapter 6 compares event processing and BI in more detail.)

Complex event processing (CEP) is used in time-driven, request-driven, and EDA modes for BI purposes. For example, a batch report could be generated every night

at 8 P.M. from event log files or event databases. This is a time-driven event processing application because it deals with time-triggered processing of event data, although the event data is "at rest" in files rather than "in motion" in the form of notification messages. Offline or batch CEP involves the same types of computation found in online CEP—that is, the CEP software performs filtering, calculations, and pattern detection. Offline CEP appears in market surveillance, fraud detection, marketing, and regulatory applications, such as payment card industry (PCI) compliance.

CEP is request-driven in some situations. As with time-driven CEP, the event data used in this type of processing is at rest in files or databases. An ad hoc query can be run against the file or database using a BI tool or some other interactive application. The time at which the query is executed is determined by the person (or application) making the request, unrelated to when the event notifications arrived at the database. Request-driven event processing is a useful mode of operation for many situations, although it is not EDA.

Some of the most compelling CEP applications are, however, implemented as EDA. Military applications illustrate the importance of timeliness and the consequent use of CEP in EDA mode. For example, there is a critical need for timely CEP analysis of data from satellite and airborne sensors, such as the Global Hawk unmanned aerial spy plane used by the Air Force, because a late response can be as ineffective as no response at all.

EDA-based CEP is sometimes called *event-stream processing* (ESP) because the systems run continuously, performing calculations on notifications as they arrive. An *event stream* is a sequence of event objects where the sequence is generally (but not always) ordered in terms of increasing arrival times. The term ESP is typically used for a system where relatively few (typically less than a hundred), high throughput event streams (hundreds to millions of notifications per second) arrive through some messaging middleware mechanism and are processed quickly (typically in milliseconds per event). However, some experts use the term ESP as a synonym for CEP, so you should consider the context in which the term is used.

Responding to Events

In nature, organisms that do not respond appropriately to threats and opportunities die out. Organisms that expend too much energy responding to non-threats and non-opportunities die prematurely, too. Like any organism, business enterprises are required to respond to external and internal events, but they must respond more rapidly and more appropriately than ever. Enterprises respond rapidly by predicting and planning for future opportunities and threats. The interactions among people in an organization, as they sense and respond to changes, are event-driven. When the interacting agents are software components, they are said to implement EDA, as described in Chapter 2.

Time-, Request-, and Event-Driven Interactions

We introduced the three basic types of interaction, *time-driven*, *request-driven*, and *event-driven*, in Chapter 2. Here we will explore the nature of these interaction pat-

terns in more detail. The types of interaction are differentiated by the times at which interactions are initiated and the set of participants in interactions. Our goal here is to understand the unique features of event-driven interactions as they occur in human and business contexts, because EDA interactions in a software context exhibit the same benefits (and costs).

- **Time-driven** In a time-driven interaction, an agent, or a group of agents, initiates an interaction at a specified time. An example of such an interaction is a regular meeting of an executive committee that meets every Monday at 8 A.M. The times of the interaction (Mondays at 8 A.M.) and the participants in the interaction (the executive committee) are prespecified.
- **Request-driven** In a request-driven interaction, a client requests a service from a server and waits to receive a reply from that server. The interaction is initiated by the client and completes when the server replies. The participants in the interaction are the client and the server.
- **Event-driven** In an event-driven interaction, an agent initiates an interaction by creating an object describing an event. Here's an example of an event: Lehman Brothers declares bankruptcy on September 15, 2008. Lehman starts an event-driven interaction by creating an event object—the bankruptcy filing. The event object describes a state change: Lehman was not bankrupt on September 14 and became bankrupt on September 15. The bankruptcy filing does not include a description of which agents—organizations, individuals, and software—should read the filing and act upon the information. Hedge funds act upon the information by short-selling bank stocks; different mutual fund investors act upon the information in different ways and at different times; government agencies act upon the same information by preparing contingency plans. In general, an event object does not specify which agents should read the object, when it should be read, or what agents should do upon reading it. The set of participants in the interaction is open-ended, the time at which the interaction terminates is open-ended, and what agents do upon reading the event object is open-ended as well.

In technical terms, we consider an event to be a change in the state of a component of an enterprise or its environment. All interactions deal with state changes so all interactions can be treated as special cases of event processing. Time-driven interactions are triggered by time moving forward to a predetermined interaction point, and this movement of time is an event. Request-driven interactions are initiated by a client making a request, which is also an event. Time-, request-, and event-driven interactions are initiated by different types of events; however, the ways in which events are processed by each of these interactions is different.

We can illustrate the basic types of interaction by considering three simple commands from a client to a broker:

- **Time-driven** Buy 10 ounces of gold every morning at 8 A.m. provided the price is less than $800 until my funds are exhausted (see Figure 3-1).
- **Request-driven** Buy 10 ounces of gold right now provided the price is less than $800 per ounce (see Figure 3-2).

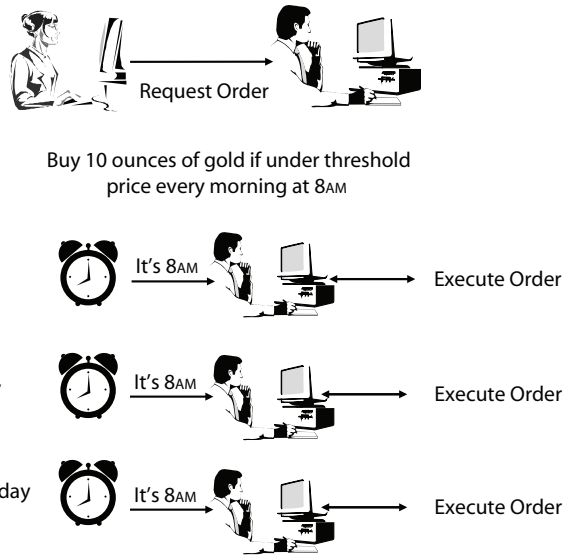


Figure 3-1: Time-driven interactions.

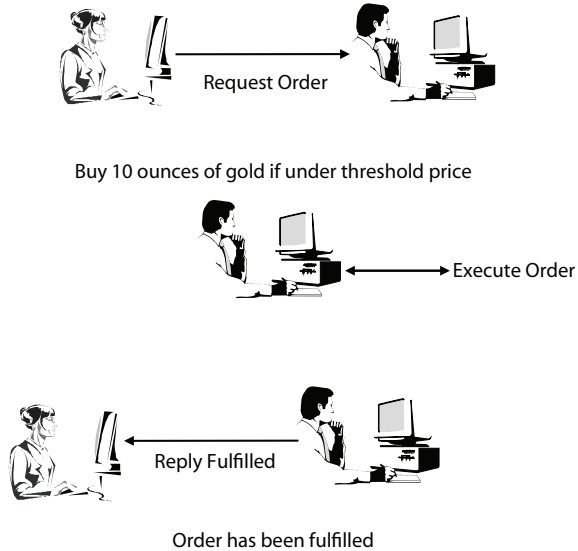


Figure 3-2: Request-driven interactions.

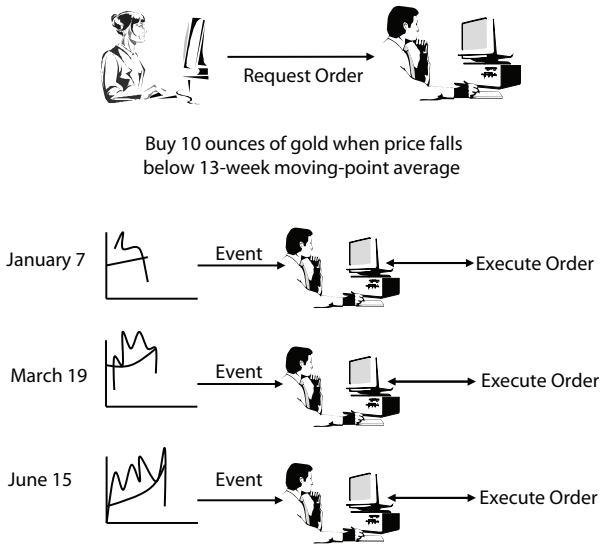


Figure 3-3: Event-driven interactions.

- **Event-driven** Until I tell you to stop buying, or my funds are exhausted, keep buying and selling gold according to a specified strategy. An example of a strategy is this: Exchange cash for gold when the 13-week moving-point average price drops below the 52-week moving-point average (see Figure 3-3).

Let's examine each of these types of interactions in more detail.

Time-Driven Interactions

In a time-driven interaction, the broker continues to perform tasks for the client at specified times. The broker's actions are triggered by time. Time-driven interactions are appropriate when tasks can be scheduled and rapid responses to unanticipated events are not critical. The benefits of time-driven interactions are evident from their ubiquity: for example, a CEO meets with the executive team every Monday at 8 A.M., or a batch computer job runs every night at 11 P.M. Organizations use time-driven interactions in combination with other interaction patterns. When a factory burns down, the factory manager doesn't wait till the following Monday to inform the CEO. Information about the event—factory burns down—is communicated to the CEO as soon as possible.

IT relied on time-driven interactions in the 1960s. Most computing jobs required long execution times, so job schedules optimized use of the scarcest resource: the computer. Such schedules were often largely time-driven. The legacy of time-driven interactions remains in many organizations though computers are no longer the scarcest resource. Today tasks are scheduled dynamically; this enables event processing applications to monitor the current situation and adjust task priorities dynamically. For instance, instead of running a batch application to check for medical

insurance fraud once a day, an EDA application can detect anomalous behavior as it occurs and immediately schedule execution of a more computationally intensive task to evaluate that behavior.

Request-Driven Interactions

In a request-driven interaction, a broker executes a service for a client and then does nothing on behalf of the client until the client makes another explicit request.

Chapter 2 described how request-driven software interactions are better suited than time-driven interactions for meeting urgent needs. The same principle applies to request-driven human interactions. For example, a CEO may call a CFO to ask about the status of a line of credit rather than wait to find out the status at the next scheduled weekly meeting. The thread of interaction begins with the CEO making a request to the CFO and ends with the CFO responding to the request. In that context, the CEO is the client and the CFO is server. A common example of a request-driven activity is a consumer looking up a bank balance at a website.

Advantages of request-driven interactions are that the interactions have clear initiation points (client makes a request), clear termination points (client receives a reply), and well-defined participants (the client and the server). These advantages are manifested in both human and software interactions. In some cases, the request and reply can be treated logically as a transaction—a single indivisible atomic operation: a transaction is either completed successfully or aborted. For example, when a bank is transferring funds to another bank, you want either the transaction to succeed and funds debited from the first bank and added to the second, or the transaction to be aborted and the amounts in both banks to remain unchanged. You would be perturbed if a transaction you initiated to move funds from one bank to another either lost your money or never terminated. Though transactions can be implemented as sequences of event-driven interactions, they are more commonly treated as request-reply interactions.

The request-driven interaction pattern has been used in IT since the dawn of computing with good reason. Programming languages, since the 1960s, have used procedure calls in which a program calls a procedure and waits for a response. Remote procedure calls, remote method invocations, client-server interactions, and most web service interactions are request-driven. The long history and current ubiquity of request-driven interactions provide evidence of their value.

Event-Driven Interactions

In the event-driven interaction example, the broker continues to do tasks for the client; however, the actions of the broker are triggered by events and not by the clock or by requests. The example illustrates the use of CEP: The crossing of moving-point averages is a complex event that is detected by carrying out computations on a set of simple event objects that record prices of commodities over time.

The event-driven interaction example, though very simple, has features that merit discussion. The customer and broker have long-term expectations of their interac-

tion: the broker continues to carry out responses on behalf of the customer until told to stop. By contrast, a request-driven interaction terminates with the reply from the server. Indeed, the requester may not want the server to keep data about the request after the interaction is over. By contrast, in an EDA application, the user expects the application to maintain information about the user so that the application can continue to execute tasks on the user's behalf. For example, you expect your doctor to remember your medical information so that the doctor can send you an alert when an event—such as the recall of a drug that you are taking—occurs. The differences in expectations impact designs of systems based on time-, request-, and event-driven interactions.

Next, you'll study different types of interactions among people and among components in IT systems, which will help you understand the evolution of business-event processing.

Interactions in Human Organizations and IT

When a working parent goes on a business trip, he or she expects to receive a message if an emergency occurs at home. If the parent hasn't received such a message, he or she assumes that no emergency has occurred. This commonplace situation illustrates a characteristic of event-driven applications: *The absence of messages conveys information.*

The parent has a model of how the home functions and the parent expects to be notified when reality deviates significantly from this model. A parent can function only because he or she can estimate characteristics of the state of the home without monitoring the home continuously. In software engineering terms, the parent's expectation is a contract between the parent and a support system: family, friends, and others. Their joint expectation of what is, and what is not, normal in the home is an informal model of the home.

An important aspect of the parent's expectation is that it is long term: it does not start with a request and end with a reply. People function only because (possibly informal) contracts in EDA applications enable them to be aware of their environments without constant, continuous monitoring. CEP is fundamental to the operation of families, enterprises, and governments; all organizations function only because people depend on others to detect and respond to complex events.

Time-, request-, and event-driven interactions appear in many places in IT. User interactions with search engines provide simple examples of the basic types of interaction in software systems. When a user executes a simple search, say by entering key words in the search template, the user (the client) is making a request for a service (a web search) to the search engine (the server). The server responds to the request by executing the service and replying with the results of the service. Thus, the execution of a search follows the request-driven pattern of information flow.

Search engines and other web companies offer alert services: Airlines call customers if flights are delayed, stock trading companies send alerts when stock prices fall below thresholds, and news sources send alerts when stories that fit a profile are published. Applications that send alerts are examples of event-driven interactions.

Compare expectations that users have about using search engines and alerts engines. A user who inputs key words in a search engine expects to get a reply containing a list of documents dealing with the key words. The interaction between the user and the search engine ends when the user receives the reply to a search request. By contrast, a user who inputs the same words in an alerts engine expects to get periodic messages containing such lists. The interaction ends only when the user cancels the alert service. Remembering what users want and serving their needs over the long term is a distinguishing characteristic of event-driven applications.

Predictive Systems

Predictive systems that warn about forthcoming events are, as one might expect, primarily event-driven. The detection of past events leads to predictions of more serious events. For example, a hurricane far out at sea may not cause much immediate damage; however, the detection of a distant storm may result in a prediction that a hurricane will hit populated areas later. The detection of certain patterns of trade in foreign markets may lead to predictions about the probable direction of stock prices in local markets. Increases in click-through activity in a marketing campaign website may lead to predictions of greater call volume and sales. Predictive applications give systems more time to prepare for critical events, but they are generally less accurate than systems that detect past events.

Applications that predict future events may also be time-driven or request-driven. For example, the government predicts unemployment figures periodically, and a department, when requested to predict sales under an economic scenario, will reply to the request with a projection. The predictions are, however, about events; so, EDA is often associated with predictive systems.

Communicating Event Objects

A fundamental difference between EDA interactions and other types of interactions is the decoupling between the creation and reading of an event object. An event object is a record of a change in state; it does not specify which agents should read it, when they should read it, or what they should do upon reading it. This decoupling implies that an event-driven application must have a mechanism, distinct from events themselves, for ensuring that an event object can be read at appropriate times by appropriate agents who take appropriate action. We call this mechanism the event-object *dissemination network*.

An *event processing network* (EPN) consists of a set of event-object processing agents (EPAs) and a dissemination network that obtains event objects from EPAs that produce them and delivers (copies of) event objects to EPAs that consume them. EPAs include sensors, responders, CEP agents, and all the agents that carry out computations. The dissemination network is a component of an EPN; the EPN includes producers and consumers of event objects, whereas the dissemination network does not. As you will see later, the dissemination network and the sets of producers and consumers are often dynamic to adapt to changing conditions.

All mechanisms, such as message channels and databases, for receiving event objects from producers and delivering them to consumers are part of the dissemination network. A producer may send event objects to a consumer using low-level protocols or web services or some other notification mechanism. The dissemination network may include intermediaries that analyze messages and direct them to appropriate destinations. Message oriented middleware (MOM) and databases used for storing and communicating event objects are part of the dissemination network.

Many agents at different locations create event objects. Some event objects describe predictions of future events such as "a hurricane will strike between Tampa and Key West between 1800 and 2000 hours." The collection of event objects, over time and space, contains the raw material of the history and the predicted future of the enterprise. The dissemination network makes this valuable material *actionable* because it delivers the material to agents that need to act based on this information. The decoupling of event processing activity from communication, and flexible dissemination networks, makes EDA applications agile.

Networks that convey event objects are widespread. News feeds using RSS, ATOM syndication formats, podcasts, and a variety of content distribution networks—including peer-to-peer networks such as BitTorrent—are widely used. Features such as Internet multicast help make dissemination more efficient. Search engine companies that disseminate alerts about changes in websites use the Internet to communicate information. Voice mail and e-mail can be channels for conveying event objects. The details of implementations are not important at this stage. What is important is the separation between the production and consumption of event objects on the one hand and their dissemination on the other.

Publication-Subscription Networks

An important type of dissemination network is the publication-subscription (or pub/sub) network. Agents send event objects to the network by "publishing" them, and the event objects they send are called *publications*. An agent's interest profile, called a *subscription*, specifies the kinds of publications of interest to the agent. When an agent publishes an object, the network delivers copies of the object to all agents whose subscriptions match that publication. A clipping service that monitors news is an example of a pub/sub broker. The client subscribes with the clipping service by telling it what kinds of news stories are relevant. Thereafter, the clipping service sends the client publications that match the client's subscription.

Generally, a subscription is a simple match on a single publication: either the publication matches the subscription or it does not. Determining whether a publication matches a subscription and then sending the publication on to the subscriber is an example of simple event processing. Some pub/sub networks allow subscriptions to be complex; for example, they send information about all stocks whose daily moving-point average drops below its 13-week average. Such matches and consequent dissemination of information are examples of CEP.

A dissemination network may be required to deliver a publication that matches a subscription at prescribed times using different modalities: for example, it sends urgent

information to a mobile phone at all times; for less critical information, it calls the office phone during working hours and the home phone in the evening; it sends all other information by e-mail. It is helpful to separate the concerns of modalities of delivering information and the complexity of matching publications and subscriptions from the fundamental requirement of disseminating information based on simple matches. Though we choose to think of producers and consumers of event objects, and the dissemination network, as logically different components, they may be implemented within the same device and even the same code.

Conclusion

The pattern of interactions and decision-making among people in an enterprise is similar to the interaction patterns among components in IT systems. Basic types of interaction include time-driven, request-driven, and event-driven. Each of the basic types has advantages and therefore many applications employ combinations of these types. Focusing on the most elemental aspects of software architecture—how components interact—helps us understand the differences in features and benefits emphasized in different types of interactions. Different types of software interactions evolved at different points in the history of IT for good reasons; a study of these reasons helps us to estimate how software will continue to evolve. When discussing the relative merits of EDA, SOA, event-driven SOA (ED-SOA), asynchrony, synchrony, or other concepts, it is helpful to cut through to the basics, and ask "What is going on at the most basic level?" And the answer is this: interactions among people and among software components.

4

Costs and Benefits of Event Processing Applications

This chapter describes methods for evaluating the costs and benefits of event processing applications. This evaluation helps identify the range of problems for which event processing approaches are suitable.

The benefits and costs of event processing applications are related directly to business, because they visibly impact business people as well as customers and suppliers. Costs and benefits should be evaluated by, or in conjunction with, business users. Applications that help business people respond to events may change the way people work. For example, an application that detects risks and opportunities for traders in a company enables them to exploit these events and increase profits; however, a side effect of the application could be that trading patterns of different traders become increasingly similar. These types of influences are more likely to be apparent to people in the business.

Exploiting Events for Business Value

Millions of events are generated in enterprises every day, and many events are recorded in logs of various types. A study evaluating event processing applications for a business has the benefit of identifying valuable, but unexploited, event sources inside and outside the enterprise. The study will also identify technology trends, such as decreasing costs of sensors, which allow the business to exploit entirely new sources of events.

We organize the cost/benefit metrics into a collection of categories with the acronym *REACTS*, which stands for the following:

- Relevance of information to a user
- Effort in tailoring a user's interest profile to ensure that the user gets the information needed
- Accuracy of detected events
- Completeness of detected events
- Timeliness of responses
- Safety, security, privacy, and provenance of information

We first discuss each of these measures and later show how designers trade off one measure against another in developing different applications.

Relevance

Information can be accurate but irrelevant. The relevance of information to a person depends, in part, on the person's context. Information about road congestion due to an accident on a freeway in Los Angeles is relevant to people in the Los Angeles area but irrelevant to people elsewhere. People pay attention to different issues at different times, so the relevance of information to a person changes over time.

It is a best practice to allow end users (operations personnel or other business people) to control or modify the types of event notifications sent to them by the application. End users best understand tradeoffs between missing occasional critical information on the one hand and alarm fatigue—weariness from getting too many false alarms and too much irrelevant information—on the other. If an IT person or a top business executive decides which events will be pushed to the end user, their concern about costs of missing significant events may make them configure applications to push more data, including more irrelevant data, to operations personnel.

A benefit of a well-designed event-driven architecture (EDA) application is *attention amplification*: the application acquires data from many varied data sources, processes the data, identifies the information that deserves the most attention, and computes data that helps in executing responses. By filtering out less relevant information and prioritizing relevant information, the application enables users to gain longer periods of uninterrupted time to concentrate on important issues.

Relevance is one of the metrics that businesses use when deciding whether to continue doing business as usual or whether to develop EDA applications. For example, multilateral trading systems and stock exchanges can conduct market surveillance in traditional ways or use event processing to detect suspicious activity and highlight relevant information in dashboards. The increase in relevance of the data displayed using event-processing surveillance systems is leading to their widespread use.

Effort

This measure is the cost of the time required to develop an EDA application that meets the individual needs of each user. This effort falls into two broad categories: the effort in implementing the initial application and the effort in tuning the application for each user's needs. A large fraction of the cost of an application is incurred after the application is put into production.

Different users within an enterprise have different needs: A one-size-fits-all specification of events and responses doesn't work. People in many roles participate in tailoring a system to fit the needs of each user: Professional services consultants provided by systems integrators and vendors tailor software to fit business needs; IT staff in the enterprise learn event specification notations provided by vendors and set up business-oriented templates for end users; power business users create their own macros; and, finally, each business user spends time learning how to use tools to tailor the system to that user's individual needs. The cost of this effort is substantial.

Consider, for example, the effort required to set up an event-driven application to support trading of electric power. An application that facilitates trading in energy is

beneficial to the extent that the application results in traders making more profit at less risk. Though an application has sophisticated algorithms and novel computer architectures, it will be ignored by traders unless it helps them with their day-to-day operations. The traders themselves have to invest time to ensure that the event-processing system works for them; however, time spent by traders away from their trading desks is very expensive, but without this investment the system won't work effectively.

An effective way of managing the tradeoff between power and ease-of-use is to develop user interfaces that fit different roles of business users. For example, a package-tracking application alerts customers as a package makes its way along different points. Customers tailor the application to their specific needs by identifying what packages they want tracked and how they want to be alerted; in this case, tailoring the application is as simple as filling in slots in a template. Other users, such as operations managers in package-tracking applications, have different user interfaces tailored to their needs. This approach can be made more powerful by giving business users mechanisms to compose specification templates to get new specifications. Mashups in web development are an example of a technology that enables the combination of data from multiple tools.

Many EDA applications will be layered on top of existing applications in the business. A great deal of skill goes into identifying events in the underlying applications that can be exploited by the business, defining schemas for event objects, and determining business responses to events. Deciding what event objects to store requires business intuition because an event object may be used at some future time by an application that isn't currently anticipated. Recording and storing all nontrivial state changes is not a viable option either because that is far too much information. Though the costs of storage continue to decrease, event objects may be generated at such high rates that exploiting them at some future time will be an intractable task. Event design is as important as database design. The investment in skilled designers is a significant cost. Indeed, the major part of the cost of an EDA application is the cost of the time of business users, IT staff, and professional services in design, deployment, and continuing operations. But this investment yields significant benefits over the long term.

Accuracy

EDA applications are beneficial when they generate accurate responses and display accurate data. Some degree of inaccuracy is, however, likely. One type of inaccuracy is a *false positive*, the incorrect reporting of an event that does not actually occur. An example of a false positive is a false prediction that a tsunami will strike a beach at a specific time. An example of inaccuracy in an event parameter is a prediction that a destructive category five hurricane will hit a city when the actual storm that hits is only a minor tropical depression. Inappropriate decisions are made when an EDA application generates inaccurate data. A false tsunami warning results in beaches being evacuated unnecessarily, and an inaccurate prediction of a surge in the price of the stock of a company results in traders incurring losses.

Accuracy is different from relevance. Information can be accurate and irrelevant. Likewise, information about a hurricane about to strike your home is relevant to you

but can be inaccurate. Accuracy refers both to the accurate detection of an event and the execution of an accurate (or appropriate) response.

The costs of inaccuracy over the lifetime of an application depend on the frequency of inaccuracies and the average cost of an inaccurate response. In some applications, rare but massive losses are more costly over the lifetime of the application than frequent small losses. Société Générale lost over seven billion dollars due to a trader, and Barings Bank collapsed after a trader lost over a billion dollars.* Carefully designed event-driven CEP applications could have detected anomalous trading patterns and reduced the losses. Steps can be taken to prevent massive losses by carrying out sanity-checks of proposed actions: Unusual or high-risk responses are sent to another system for further approval while low-risk responses are executed directly. If an inappropriate high-cost response has been invoked then compensatory activities are initiated.

For example, algorithmic trading systems are designed to reduce costs of "fat-finger" errors. A *fat-finger trade* is an erroneous trade in which the actual amount traded is greater than the amount that the trader intended. For example, a fat-finger error may result in a trade of a million shares of a stock when the desired trade was only a thousand shares. The name *fat finger* derives from possible errors caused by a person with a fat finger who accidentally types extra zeros in the amount to be traded or leaves his finger on a key for too long. Some systems automatically detect and block probable fat-finger errors, and if necessary execute compensatory actions. Such systems are compositions of two event-driven subsystems: the first generates proposals for responses, and the second filters out suspicious responses and takes compensatory actions. Sanity checking reduces inaccuracy but slows down responses.

Completeness

A system that provides only accurate information but that does not provide all the information required to make decisions can be costly. An example of incomplete information is a false negative—a response that is not executed because an event was not detected. Consider the smart power-grid example in which electric utilities detect and respond to overload conditions. The events detected by the grid include the following: the state of the system has changed from acceptable load to overload, and the state has changed from overload to acceptable load. A false positive is an erroneous signal that claims that a particular state change took place when it really didn't. A false negative is the absence of a signal that a particular state change took place when it really did take place. For example, in the smart grid example, a false positive is a signal that the grid is overloaded when it is not; and a false negative is the absence of a signal that the grid is overloaded when it truly is overloaded. A consequence of a false positive in this application is that demand may be reduced unnecessarily and forcibly by turning off appliances, whereas a consequence of a false negative is a possible brownout.

* http://www.iht.com/articles/ap/2008/01/25/business/EU-FIN-France-Societe-Generale-Fraud.php (*International Herald Tribune*, 25 January 2008)
http://query.nytimes.com/gst/fullpage.html?res=990CEFD91F38F937A35750C0A963958260 (*New York Times*, 4 March 1995)

The costs of incomplete information are, in many cases, higher than the costs of inaccurate information. The cost of a false negative—no warning—when a tsunami strikes is measured in lives lost, property destroyed, and sea water inundating agricultural fields. The cost of a false warning of a tsunami includes the costs of clearing beaches unnecessarily, the negative impact on tourism due to unnecessary worry, and possibly alarm fatigue from too many false warnings. In this and many other applications (see the smart grid example), the cost of a single false negative is much higher than the cost of a single false positive.

The costs of false positives can be limited, in some cases, by sanity checks. For example, the costs of a false positive report, such as an apparent news item in a search engine repository that a company is about to go bankrupt, can be high if the immediate response is to sell huge numbers of shares of the company. A sell-off in United Airlines (UAL)* shares occurred after a false report about the company declaring bankruptcy appeared on a website on September 8, 2008. Shares bounced back after investors realized that the report was about an old story describing the company's 2002 bankruptcy filing. The false positive could, however, have been avoided by checking the news story with other sources. Sanity checks may improve accuracy but slow response.

Generally, enterprises cannot control costs of false negatives by sanity checking in the same way. Suppose a company had gone bankrupt, but a trader received no information about the bankruptcy. The trader could not deduce that the company was bankrupt from the absence of information about bankruptcy. In general, agents cannot deduce the existence of a false negative from the absence of information.

Business decisions are taken in the absence of complete information. Some degree of incompleteness is inevitable. An enterprise determines whether to adopt an event-processing application, in part, by evaluating the improvement in the quality of decisions due to more complete information engendered by EDA applications.

Timeliness

The effectiveness of a response depends on its timeliness. A tsunami warning issued after a tsunami strikes has little value. The value of a response to an event almost always decays with increased response time. The decrease in value of a response as response time increases is captured in a value-time function that plots response time on the X axis and value on the Y axis, as shown in Figure 4-1. The value of a response drops significantly when response time increases by even a few milliseconds in some applications, while in other applications the value of a response may not decrease much even when response times increase by minutes. The costs of not dealing rapidly with threats and opportunities in milliseconds are huge in missile defense and algorithmic trading applications, whereas the costs of not responding in minutes are not high in some accounting applications. Accuracy and completeness of responses are also critical in determining timeliness: responses based on less accurate and less complete information can be executed in less time. Designers developing event pro-

* http://www.webguild.org/2008/09/false-google-story-triggers-massive-ual-stock-sell-off.php

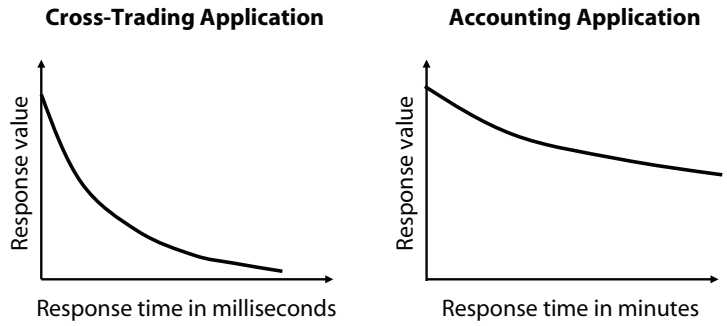


Figure 4-1: Value-time functions.

cessing applications today have to consider the value-time function as it is today and as it is likely to be in the future. The value of rapid responses is likely to increase as business moves faster.

An analysis of the tradeoff between incremental costs of systems that deliver faster responses with incremental benefits accruing from greater speed shows that reducing response times by milliseconds does have benefits in many fully automated applications such as cross-trading. In cross-trading, an asset manager matches and then executes trades directly between client buyers and client sellers of an item without first sending the requests (bids and asks) to an exchange. Regulations prohibit an asset manager from holding buy or sell requests for more than a short time before passing the requests on to an exchange. Reducing delay between the initiation of a buy or sell order and the detection of the order by the cross-trading application by even a few milliseconds is valuable. If orders can be held by the application for a few milliseconds more, then the application can hold more orders before sending them to the exchange, and this gives the application both more time to detect matches and more orders to match.

Timeliness of event-driven applications can be improved by improving any stage of the process: by receiving data more quickly, by responding more rapidly, by aggregating and analyzing data faster, or by predicting events farther into the future. For example, timeliness in algorithmic trading applications can be improved by obtaining trading data directly from exchanges rather than from slower consolidated feeds. Timeliness can also be improved by using faster message oriented middleware (MOM) software, faster processors, and effective use of parallel systems such as networks of multiprocessor machines. Designers of event processing systems make reasoned tradeoffs between timeliness and other parameters. This tradeoff must focus on the response time of the entire business application and not merely the time spent in a single component such as the event detector.

Security

Security is a concern for users of all software systems. It is, however, a particular concern for users of event-driven applications, because these applications are often critical to business. For example, electric power transmission companies that respond to

possible brownouts by turning appliances on and off in homes (via a smart electric power grid) must prevent hackers from breaking into the system and orchestrating attacks that cause brownouts. The problem of cyber attack wasn't as acute for utilities not so long ago when the companies' information technology networks didn't reach homes and expose more points to attackers. Security is becoming more important and more challenging with increasingly tight integration of critical infrastructure—electric power, gas, water, roads, and air-traffic—with sensors, responders, and IT woven into the infrastructure.

The broad area of security includes protection of privacy. Many event-driven applications acquire and store a great deal of personal information. An application that detects and responds to a customer's location by monitoring mobile phones acquires information on where the customer has been. Cars equipped with GPS and telemonitoring devices can inform remote vehicle management sites about speeds at which cars are driven at different times. Car insurance rates may depend on estimates of driving habits gathered from sensor data on cars. Many people don't object to making some event types, such as car locations, public. Many people make videos public—and videos are event objects. Event objects, like diamonds, are forever. Event objects can be analyzed by everybody who can get access to them at any time—even long after the event. Business intelligence (BI) can be used to learn a great deal about a person by fusing event objects recorded over the years. Copyright laws can be used to prevent proliferation of copies of event objects, but finding and destroying all copies is an intractable task. Privacy will become an increasingly important issue in EDA applications.

An attacker can find many ways to break into an EDA application. An attacker can *spoof* the enterprise, pretending to be a customer, and a hacker can spoof the customer, pretending to be the enterprise. In the case of the smart electric power grid, a customer can attempt to change the signals sent from her home to the grid to reduce her bill. More dangerously, malicious hackers can attempt to cause brownouts by sending signals to turn on appliances in homes, thus overloading the system. Sensors and responders are becoming more widespread and more exposed; this gives malicious hackers more points to attack the system. Preventing attackers from bringing down the system requires understanding the relationship between potential attacks and the underlying physical system; in the case of the electric power system, this requires understanding the effect of cyber-attack scenarios on the electromechanical system consisting of generators, transmission, distribution, and consumption systems. Cyber-physical security—understanding and managing the relationship between attacks to the cyber-infrastructure and the underlying physical, biological, or social system—is becoming increasingly important as EDA becomes widespread.

Not all attacks come from agents outside the enterprise; losses due to rogue traders at some major institutions were due to agents within the enterprise. Detection of insider attacks requires that CEP detect anomalous behavior that indicates attacks, and this is difficult when the attacker knows the detection algorithms. The area of security includes forensics—understanding what went wrong when a system was attacked or when errors were made—and BI tools operating on event-object repositories help here. An aspect of forensics is event *provenance*—a description of what data items were used to generate an event object and the process by which the event object was

generated. This highlights the importance of careful designs of event schemas and links between simple event objects and the complex event objects created from them.

Security problems are not new; however, event processing systems make them more severe. The cost of ensuring security is a significant part of the overall cost of the system, but the costs of not ensuring security are much higher.

Conclusion

Since EDA applications have direct, visible impact on business, particular attention must be paid to business participation in evaluations of costs and benefits. The analysis of costs and benefits will help you determine whether to implement an EDA application for a business problem or to use alternative technologies or make no changes at all. Some cost-benefit measures are more important in EDA applications than in other types of software; an evaluation of the REACTS measures for different EDA designs helps you determine the best designs for your business problem.

5

Types of Event Processing Applications

A list of business problems for which event processing is the preferred solution will help you determine where it could be applied in your organization; however, such a list will be incomplete and out of date the instant it is compiled, because of the increasing variety of business problems for which event processing can be used profitably. A better approach is to use a systematic framework to evaluate whether event processing in general, and event-driven architecture (EDA) and complex event processing (CEP) in particular, are appropriate for a given problem. We propose a framework built upon the material presented in the preceding chapters. We use this framework to evaluate the suitability of event processing for existing business problems, and in the next chapter we use the framework to predict how event processing, EDA, and CEP will evolve.

Features Driving Demand for Event Processing

Here, we discuss features of business problems that drive increasing demand for event processing applications. The presence, or absence, of each of these features in a problem is one of the measures used to evaluate the appropriateness of event processing for the problem.

As discussed in earlier chapters, an important difference between event processing interactions and time- or request-driven interactions is that the creation of an event object is decoupled from its eventual use. The dissemination network is responsible for getting event objects from agents that produce them to the agents that need to act on them. A common structure of event processing networks (EPNs) is a "hub-and-spoke" with agents—including sensors, responders, and other types of event processing agents—at the spokes and the dissemination network at the hub. The spokes produce, consume, and process event objects. The hub receives event objects from the spokes and sends (copies of) event objects to the spokes that need them. An alternative view of the hub-and-spokes structure is that of an event-object bus (the hub) with producers pushing data on to the bus and consumers pulling event objects from the bus. An EPN is flexible to the extent that event-object producers and consumers can be modified, attached, and detached from the bus easily; and the flow of event objects through the bus can be modified easily. Most implementations of EDA are, indeed, flexible.

EPNs don't always have bus structures. Indeed, the dissemination network can be hard wired with each producer sending event objects in proprietary formats to specific ports of specific Internet Protocol (IP) addresses. Hard-wired networks are, how-

ever, generally less flexible. The bus structure is a logical structure, not a physical one. The bus may be implemented as a distributed system with many components. The specifics of the implementation are not important at this stage; what is important is the logical separation between producers and consumers of event objects on the one hand and the dissemination network on the other.

An EDA application can grow by accretion with the addition of agents and event-object types that were not planned for when the application was first designed. For example, a record of a credit card transaction is an event object. A credit card transaction is itself a request-reply interaction between a retailer and the bank that issued the card, but the record of the transaction is an event object that can be used by other agents at other times. Fraud detection applications act upon the information recorded about the credit card transaction to evaluate the probability of fraud. Customer relationship management (CRM) applications act upon the same information to determine whether the customer should be offered promotions. Risk-management applications act upon the same information to evaluate lines of credit. Corporate performance management (CPM) applications use the information on an aggregate level to tell top management about the overall health of the business. (Chapter 6 discusses performance management in more detail.)

If an EPN is represented by a hub-and-spoke structure, then the spokes are agents in applications such as CRM, fraud detection, risk management, and performance management, while the hub is the dissemination network. New applications, new producers, and new consumers of event objects can be added after the system is in operation. Adding spokes to a hub-and-spoke structure is generally easier than changing an organizational chart or some other graph structure. The key to flexibility is the dissemination network, and many flexible networks are available today.

Use event processing technologies when adaptability is a key requirement.

Looking Outside the Virtual Enterprise

Much of enterprise IT focused exclusively on services within the *virtual* enterprise—customers, suppliers, and the enterprise itself—from the 1950s through the 1980s. Now things have changed: IT is expected to help the enterprise respond rapidly to conditions outside the virtual enterprise.

An enterprise can ensure that interactions among all agents within the virtual enterprise follow well-defined protocols and speak the same language—that is, use the same schemas, the same meanings for phrases, and agreed-upon services. A supplier may be required to submit a quotation to a manufacturing company by making a request for a submit-quotation service provided by the manufacturer. The supplier's request must use the schemas and meanings specified by the manufacturer so that the service can process the request. The manufacturer, for example, may require a supplier of ball bearings to specify the size of a bearing in terms of diameters in millimeters as opposed to radii in inches. When you buy an item from an online retailer, you describe the item you want by filling out a form specified by the retailer; you are, in effect, using the retailer's language and the retailer's service. Agents within the vir-

tual enterprise can interact with each other using services and languages approved by both parties.

Agents outside the virtual enterprise may not interact with agents within it by using services and languages specified by the enterprise. An enterprise's competitor, for example, will not keep the enterprise informed about the competitor's actions by invoking the enterprise's services. Government agencies will not inform an enterprise about new regulations or unemployment figures by requesting services specified by the enterprise. The enterprise must actively acquire external information and make sense of it. Applications that respond to events outside the virtual enterprise are event-driven.

Use event processing technologies for applications that sense and respond to events outside the virtual enterprise.

Managing by Exception

Flying an airplane has been described as "hours and hours of boredom punctuated by moments of stark panic."* Like airplane instrumentation systems that alert crews about exceptional conditions, event processing systems help organizations deal with rare events. People aren't as effective as computers in remaining alert while monitoring the environment for signs of rare events. Event processing applications support management by exception (MBE)—the applications acquire and analyze large quantities of data, detect rare exceptional situations, initiate responses, and alert people. This frees organizations from having to pay attention over long periods to multiple high-throughput data streams so as to detect rare situations.

Scientific instruments measure and record everything, because everything may be relevant to data required to complete an experiment or test a theory. A seismic network, when used as a scientific measurement instrument, measures and records even very low intensity shaking—which may be due to a passing truck—because the measurements could prove useful to science. By contrast, a seismic network built for first responders focuses on detecting rare intensive shaking that indicates collapsed buildings, broken bridges, or other serious damage; such networks don't need to store records of vibrations due to passing trucks. A seismic network that helps first responders is a system that supports management by exception—it detects the important exception (severe shaking)—and helps responders deal with it. The many examples of management by exception include a CEP application that warns management about patterns of trading that indicate fraud, an EDA application that responds to overloading of the power grid, a CEP application that determines that peanut butter from a factory is contaminated with salmonella, or an EDA application that takes action when actual and planned expenditures deviate significantly. Management by exception is an inherently event-driven activity since the occurrence of the exception is an event.

* See http://www.amazon.com/Hangar-Flying-Alfred-DAmario/dp/1434355284, *Hangar Flying* by Alfred D'Amario, Publisher: AuthorHouse 2008.

Use event processing technologies for applications that support management by exception.

Responding to Situations That Change Rapidly

Some situations change so rapidly that applications cannot keep up with the changes. For some problems, applications that keep up with arriving data but drop items occasionally are better than applications that never drop items but fall far behind. For example, cross-trading applications make money by matching buy and sell offers; however, not identifying a match isn't a catastrophe. Normally, buy and sell orders are matched in an open exchange, but firms can match the buy order of one customer with the sell order of another within the firm, provided the transaction is executed in accordance with regulations. If the firm cannot match buy and sell orders within specified times, the firm is required to send the orders on to an exchange. Regulatory agencies impose severe penalties if the application falls behind and the firm holds buy and sell offers for longer than the permitted time. Event processing is more suitable than request-driven architectures for such applications that must keep up with arriving data but may lose data occasionally.

Online transaction processing (OLTP) applications, such as airline reservation systems, handle high data rates using request- or event-driven interactions. Ticket purchases are required to be transactions—that is, either the purchase is aborted (the customer doesn't pay for the ticket and the airline doesn't sell the ticket) or the purchase completes successfully. Many business interactions do not have to satisfy the stringent constraints of transactional processing, and EDA applications are particularly cost effective in these situations. Though EDA can be used for transactional systems, a more common view of transactional interactions is that they are request-reply interactions. EDA applications are often overlaid on top of request-driven applications where EDA applications capture and act on events generated by the request-driven substrate.

Use EDA for applications that must respond quickly to situations that change rapidly and asynchronously, and where interactions do not have to be transactional. Use EDA layered on top of transactional applications to capture and process events, and to respond to situations in the underlying transactional system.

Responding to Complex Unanticipated Situations

Let's compare two applications: an application that customers use to configure and purchase computers online, and an application that detects cybercrime. A customer specifies many parameters to configure a machine: the machine type (notebook, laptop, desktop, or tower), size of memory, numbers of hard drives and USB ports, and so on. Though the vendor does not anticipate each customer's specific request, the vendor does anticipate the types and ranges of requests. The interaction between the customer and the vendor is request-driven because the interaction must be a transaction

that is either completed successfully or is rolled back so that it is, in effect, not executed at all, and the customer waits expecting an immediate response from the vendor. Unlike transactions used to configure computers, the ranges of behaviors that indicate cybercrime may not be anticipated. Criminals attempt to remain undetected by refraining from the kinds of behavior expected by the enterprise. Detecting noncompliant behavior is particularly challenging when detection needs to occur rapidly. One of the many application areas of CEP is detecting noncompliance.

Responses to anomalous behaviors are, perforce, less well-defined and more fluid than responses to anticipated situations. Applications for configuring computers online can be request-driven because behaviors generally fall within anticipated ranges. Applications that detect cybercrime and other anomalous behaviors are partly event-driven because the detection and characterization of the behavior is a key part of the application.

CEP is useful when characterizations of anomalies are complex and when detection of anomalies requires analysis of data gathered from multiple agents across time. For example, warnings of tsunamis are based on fusion of data obtained over time from many sensors. Trading stock "ahead" and "interpositioning" of trades by trading specialists are violations of federal securities laws, and such trading violations are not detected from a single violation but from an analysis of trades over time. Simple exceptions, such as an out-of-range parameter in a form, can be handled without event processing technologies, but complex exceptions benefit from EDA in general and CEP in particular.

Use event processing technologies for applications that must react rapidly to complex unanticipated situations.

Summary of Features Favoring Event Processing

Event processing technologies are also used to deal with problems that don't have the features discussed here; however, most event-driven applications do have one or more of these features. We offer the following mnemonic, using the vowels in English, for features of business problems that favor event processing:

- **A** Agility and adaptability
- **E** Exception—management by exception; monitoring to detect the rare event
- **I** Instantaneous response to rapidly unfolding situations
- **O** Outside—responding to events outside the extended environment
- **U** Unanticipated—responding to unanticipated situations

We refer to the features of business problems that favor event processing as "problem features" or "the A-E-I-O-U features."

A Framework for Analyzing Problem Domains Suitable for Event Processing

Next, we propose a framework based on the material presented earlier that helps you determine whether EDA is the best choice for a problem you face. We use the framework to identify business problems suitable for EDA in different domains. The framework is based on the following:

- The types of contracts, or expectations, of agents in different interactions
- The cost/benefit measures
- The features of problems, such as the A-E-I-O-U features, that favor different types of interactions

The framework focuses on how information flow impacts what the business does and not on information as an end in itself. As a consequence, the framework must explicitly consider uncertainty, the likelihood of error, the costs of mistakes, and the benefits of timeliness. We review, very briefly, the elements that go into the framework.

A problem that has any of the A-E-I-O-U features is a candidate for event processing technology. The degree to which these features are pronounced in the problem determines whether event processing, often combined with other technologies, is the preferred choice. We also compare the expectations we have about the proposed application with expectations we have about components in time-, request-, and event-driven applications; if the expectations are similar to those for event-driven applications, that's a clue that EDA is appropriate. If we decide, based on the A-E-I-O-U features and comparison of expectations, to evaluate EDA solutions, we then compare the REACTS cost/benefit measures for an EDA solution with alternative approaches and choose the approach with the best cost/benefit ratio.

Business Problems Suitable for Event Processing

Many business problems are suitable for event processing, and an exhaustive list is not instructive. Our focus here is on applying a framework that will help you evaluate the suitability of event processing for any problem. Next, we'll carry out the exercise of applying the framework to a few domains.

Defense and Homeland Security

Problems in defense, security, and crisis management exhibit all the A-E-I-O-U features. Many of the devastating events handled by homeland security agencies, ranging from tsunamis to chemical spills, are rare. Systems continuously monitor the environment for signs of these events and take action when the events are detected. These situations unfold rapidly, asynchronously, and unpredictably, and responses to events must be rapid as well. Agencies must respond to critical events that occur out-

side the agencies. And though agencies plan extensively for crisis situations, each crisis has unanticipated features.

The kinds of expectations we have for components in defense and homeland security applications are more typical of expectations for components in EDA than for request-driven or time-driven applications. We expect components to sense and respond to conditions continuously. For example, they must sense when contaminants are in the water supply and take remedial action, predict when and where a hurricane will make landfall and warn citizens, and detect intrusion into a network and then identify and shut out the intruder. We don't expect a server merely to tell a client the condition of the water supply, the location of a hurricane, or the presence of an intruder when the client makes a request.

An analysis of each of the REACTS cost/benefit measures demonstrates the value of EDA in general and CEP in particular. Military applications are tuned so that only highly relevant information is pushed and so that soldiers can communicate information with little effort. The benefits of EDA applications, when compared with the alternatives in terms of accuracy, completeness, and timeliness, are self-evident. Security remains a problem, however, as EDA applications are as vulnerable to attack as are other types of applications. Nevertheless, military, security, and crisis-management problem domains exemplify areas where EDA and CEP have great and obvious business value: these problems have all the A-E-I-O-U features and have excellent cost/benefit values for the REACTS measures.

Track-and-Trace

Most shipping, trucking, railroad, and other logistics companies use track-and-trace applications that allow the company and its customers to track the location of an item and trace its path from shipment to destination. Concerns about mad cow disease and bioterror attacks are leading toward a national farm identification system that tracks every farm animal with an identifier and possibly a tag or microchip from birth to death. Contaminations of milk products with melamine, peanut butter with salmonella, and spinach with *E. coli* have highlighted the importance of ensuring safety in food supply. Tracking food sources, both animal and vegetable, helps identify problems early and minimize risk.

Electronic pedigree systems record major events—location of manufacture, shipment, prior sales, and trades—that occurred over the lifetime of items such as pharmaceutical products. All these applications track events in the items' histories—be they packages, cows, tomatoes, medicines, or data. Some applications send alerts when histories deviate from norms: for example, when a food shipment that should have been kept at temperatures below a specified threshold is exposed to higher temperatures for an extended period, or when a package that should have arrived at a transshipment station by a specified time doesn't arrive.

Our expectations of track-and-trace applications and components are closer to expectations of EDA systems than to those of systems based on request-driven or time-driven interactions. We want to initiate the process of detecting a salmonella outbreak or tracing a lost package as soon as possible—not on a once-a-month or even

a once-a-day basis. And we expect these components to be continuously active carrying out tasks; we don't expect them to remain passive, waiting for requests.

Most track-and-trace problems have some of the A-E-I-O-U features. The consequences of poor track-and-trace systems are severe; in the food industry the consequences include deaths and loss of confidence in basic staples such as milk and peanut butter. An analysis of the REACTS measures shows that CEP is an appropriate technology for many applications dealing with track-and-trace. The detection of a salmonella or *E. coli* outbreak, for instance, requires analysis of time-varying data from multiple sources in different organizations. Pinpointing the outbreak to a specific peanut factory or spinach plant requires a great deal of analysis. A study of the cost/benefit measures suggests that public health agencies, fast-food chains, agribusinesses, package-handling companies, and indeed all enterprises that need to track-and-trace items benefit from EDA.

Infrastructure

Increasing amounts of funds in many countries are being allocated to public infrastructure such as roads, bridges, the electric power grid, and the water supply. Increasing power and decreasing costs of sensors, responders, and CEP agents makes "smart infrastructure"—a combination of traditional infrastructure with information technology—more cost-effective than traditional infrastructure. The term "cyber-physical systems" has been coined to describe systems that conjoin information systems with physical systems to provide powerful capabilities. Buildings with active controls that determine how they respond to earthquakes are examples of cyber-physical systems; the building without controls may collapse, but the building with sensors, processing agents, and responders is expected to be more resilient. A *smart* infrastructure is an infrastructure integrated with EDA applications. Cyber-physical systems have EDA applications tightly woven into the design and implementations of physical systems.

Infrastructures, such as bridges, have been traditionally inspected on a time-driven basis. As the infrastructure ages, the frequency of inspections has to increase because the mean time to failure decreases. EDA applications integrated with the infrastructure can improve reliability without requiring frequent inspections. Infrastructure such as buildings and bridges can be equipped with sensors such as accelerometers and strain gauges that transmit data to event processing networks that detect potential problems. The possible failure of transformers in the electric power grid can be predicted using data from sensors that measure parameters such as temperature, gas dissolved in transformer oil, and transformer vibration. Work crews are sent to inspect components that are identified by the EDA application as likely to fail. A combination of periodic inspections and sensor-based health monitoring of infrastructures reduces the likelihood that the public will be exposed to catastrophic failure.

All the A-E-I-O-U features that favor event processing appear in cyber infrastructures. The systems are required to be agile to deal with additions and replacement of components such as sensors and actuators. Early-warning mechanisms are compo-

nents of systems that manage by exception. Applications must monitor and respond rapidly to situations outside the enterprise, and they must deal with unanticipated situations. The expectations we have about components of smart infrastructure are closer to expectations in EDA than expectations for request-driven or time-driven systems.

The values of the REACTS cost/benefit measures depend on the specific application; however, an analysis of these measures for many businesses that manage access to fixed resources—roads, power grids, and bridges—suggests that EDA applications are cost-effective in managing infrastructure. The ratio of costs of information systems to costs of physical infrastructure—bricks, mortar, steel, and the costs of designing and building physical artifacts—continues to change in favor of information systems, so EDA will play an increasingly important role in physical infrastructure.

Healthcare

People are living longer. Older people who want to take care of themselves at home can do so better in smart homes equipped with sensors that detect whether doors are open or closed, whether gas ranges and other appliances are turned on or off, and whether people in homes need help. Increasing use of mobile phones in many countries allows medical measurements taken in remote villages to be sent accurately and rapidly to regional hospitals. Telemedicine and medical sense and response systems have all the A-E-I-O-U features that drive demand for event processing. They must be agile to deal with new types of devices; they must sense and respond to exceptional conditions; they must respond immediately; they must interact with patients who are outside the enterprise; and they must deal with unanticipated situations. Our expectations of telemedicine and medical sense and respond systems are similar to the expectations we have of EDA components. An analysis of the REACTS cost/benefit measures and the types of contracts appropriate for components of telemedicine systems shows that EDA is the architecture of choice for these applications.

Finance

Many applications in finance also have all the features that favor event processing. Financial applications must be agile to deal with rapidly changing global conditions, they must detect and respond to exceptional conditions, they are required to respond instantly, they have to monitor activities outside the enterprise, and they must respond to unanticipated situations. Our expectations of many financial applications match our expectations of EDA applications: for instance, users expect financial applications to monitor markets and respond when conditions indicate significant opportunities or threats. Analyses of cost/benefit measures show huge benefits from using event processing applications in finance, particularly in trading capital markets, but also in credit card processing, retail banking, and customer-relationship management applications such as cross-selling and up-selling. Many interactions in finance are not required to be transactional; and in many applications, businesses make more profits by keeping up with current conditions even if the applications don't process some earlier events. EDA and CEP are used in many aspects of trading, including cross-

trading, reducing errors such as fat-finger trades, algorithmic trading, order routing, market surveillance, and fraud detection. Indeed, finance is one of the "sweet spots" of event processing.

Conclusion

This chapter proposes a framework for determining whether EDA is the architecture of choice for an application. The framework is based on expectations of what components of applications do, a collection of features that favor event processing, and a comparison of the cost/benefit measures of implementations using event processing and other technologies. We used this framework to determine the appropriateness of event processing for different business domains. This framework, coupled with trends such as decreasing costs of sensors and responders and the increasing pace of business, suggests that event processing will be used for a greater variety of business applications in the future.

6

Positioning Event Processing in the IT World

We conclude this analysis by exploring what it means to implement event processing in a commercial IT environment. We'll look at how event processing relates to service-oriented architecture (SOA), business process management (BPM), business intelligence (BI), and rule engines. We'll then describe how IT projects that implement event processing differ from those that don't. Finally, we'll offer six specific action items for successful event processing initiatives.

SOA, EDA, and Event-Driven SOA

SOA is probably the most-talked-about architectural style for modern business applications. Unfortunately, the talk is often confusing because people use the term to mean a variety of things. For our purposes, we define SOA as an architectural style for application software in which five basic principles are implemented:

- The application must be modular, so that software components can be added, replaced, or modified individually.
- The components must be distributable—that is, they must be able to run on different computers and communicate with each other by sending messages over a network at run time.
- The interfaces and related externally visible characteristics of the software components must be clearly defined and documented in metadata. Metadata describes the input and output messages of each component and enough other information to enable developers to find and use the component as part of a new application.
- A software component that provides an SOA service can be readily swapped out for another component that offers the same service as long as the new one uses the same interface as the old one. The interface design (what to do) is separate from the internal service implementation (how it is done).
- Service provider components must be shareable (or reusable). This means that they are designed and deployed in a manner that enables them to be invoked successively by disparate application systems or multiple copies of the same application. The same code and data are available to users of any application that shares that component.

Any business application that implements these five principles is an SOA application. This concept of SOA doesn't require the use of any particular technology. Most Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP) applications are implemented using the SOA style, but good SOA applications can also be built without them.

SOA and EDA are compatible and complementary. SOA encompasses request-driven and event-driven relationships. In Chapter 2, we listed five principles that an application must implement to qualify as an instance of EDA. When the relationship between software components adheres to the principles of SOA and EDA simultaneously, the overall system can be described as event-driven SOA (ED-SOA). Later, we explain why SOA and EDA should be implemented together from an organizational and a governance viewpoint.

How Business Process Management Uses Events

The term *business process management* describes two related concepts:

- In some contexts, it is a discipline for designing and managing systems in a thoughtful, systematic, and flexible way that takes the whole, end-to-end business process into account.
- Elsewhere, it is the use of BPM software, such as orchestration engines and workflow tools, at run time to direct the sequence of execution of software components and human activity steps in a process. BPM software controls the conditional execution of activities based on rules.

The BPM discipline is a collection of methods, policies, metrics, management practices, and tools used to design, run, and manage systems that support a company's business processes. Business managers, end users, business analysts, and software engineers all have roles to play when using a BPM approach. The maxims that system design should begin at a business level and that should reflect a broad, end-to-end view of the process have been part of good IT practices for many decades. However, modern BPM puts more emphasis on continuous change, paying homage to the idea that a business process is "a journey, not a destination."

BPM is naturally complementary to SOA. SOA's modular nature and well-documented interfaces can reduce the effort required to modify or add new activities in a business process. Compared with traditional applications, SOA applications are more likely to use BPM engines at run time to orchestrate the flow of work through a sequence of activities. SOA is sometimes successful without a formal BPM program or process modeling tools, but you should never develop an SOA application without at least applying the spirit of the BPM approach.

Business events are central to both concepts of BPM. Every business process starts with a business event such as receiving a customer order, a request for a bank payment, or a notice of a customer address change. Each activity within the process is also triggered by an event, such as the end of the previous activity. However, a BPM engine or an application software component that implements a business process or a step

within a business process need not be event-driven—many processes and steps within processes are request-driven or time-driven.

Run-time BPM software manages the flow of business processes and keeps track of the status of each instance of a business process. It uses rules to evaluate events as they occur and programmatically activates the next activity at the appropriate time. Vendors offer several types of BPM software products, including orchestration engines, workflow products, composite application tools, and other tools. Orchestration tools primarily control activities that are executed by software components, and workflow tools primarily control activities that are executed by people.

Figure 6-1 shows four ways that run-time BPM software uses event processing:

- **To initiate a new process instance** Each instance of a business process is triggered by a business event. Sometimes an EDA-style notification has been sent to the BPM software. BPM software can also be triggered in other ways, such as through a request-driven service call.
- **To trigger an activity step within a process** A software component that implements a step within a process can be triggered by an EDA-style notification—although, again, other mechanisms, such as a request-driven call (for example, a web service call), may be used instead.
- **To make flow decisions** Traditional BPM engines determine how to control the process flow using rules that are not affected by complex patterns in recent events. A recent advance in BPM technology involves the use of complex event processing (CEP) software to augment the rule processing capability of run-time BPM software.

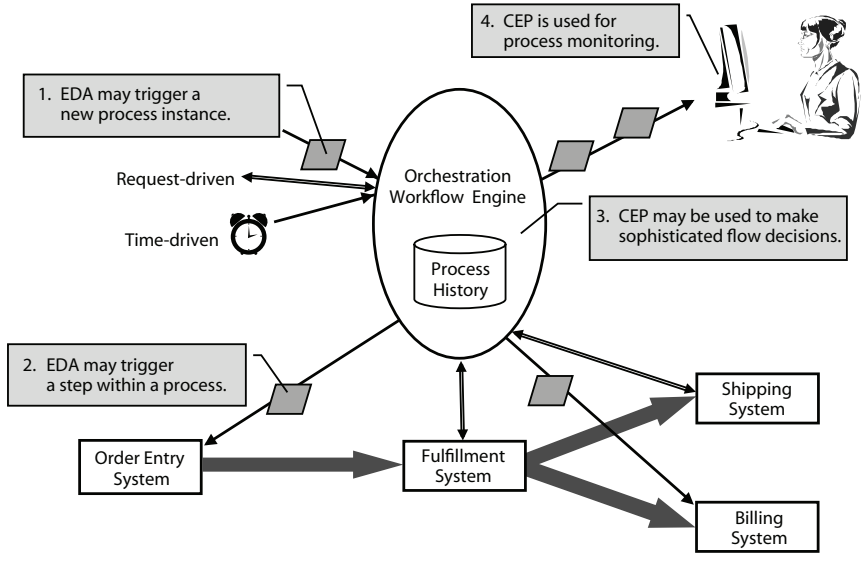


Figure 6-1: Using events in BPM at run time.

- **To monitor the process** Most commercial run-time BPM software products have a business activity monitoring (BAM) dashboard feature that can be used for process monitoring (process intelligence).

Business Intelligence and Business Activity Monitoring

The purpose of BI systems is to improve decision-making—indeed, such systems were previously called *decision-support systems* (DSSs) and sometimes still are. BI systems can be sorted into three main categories: analyst-oriented systems, BAM systems, and performance-management systems. Event-driven CEP is not used in most analyst-oriented systems, which are the most common type of BI. However, it is central to BAM and plays a small role in performance-management systems.

Analyst-Oriented BI

The most common BI applications are those aimed at analysts and knowledge workers. These provide in-depth, domain-specific analysis and delivery of information using ad hoc queries, data mining, and statistical techniques. This style of BI answers questions such as "Do we have enough account executives on the street to meet our sales quota?" or "Should we retire a product line because it is underperforming?"

BAM as a Type of BI

BAM is the second style of BI. It provides near–real-time access to critical business performance indicators to improve the speed and effectiveness of business operations. Many of the examples of CEP applications provided earlier in this book are BAM. Business process monitoring is one of the most common types of BAM.

BAM differs from analyst-oriented BI in several ways:

- The user of a BAM system is typically a line manager or other business person charged with making operational decisions. By contrast, analyst-oriented BI systems are used by staff people preparing recommendations for strategic or tactical decisions.
- Most of the input for BAM is event data that has arrived within the past few seconds or minutes, although historical data is often used to put the new data in context and to enrich the data before it is passed on to end users. By contrast, traditional BI systems rely mostly on historical data from previous days, weeks, months, or even years.
- Most BAM systems run continuously throughout the day, listening to incoming events and communicating with business people through dashboards, e-mail, or other channels. By contrast, most analyst-oriented BI systems are request-driven where the end user submits ad hoc queries or asks "what-if" questions through a spreadsheet that sits in front of a database.

Performance-Management BI

Performance-management systems (including corporate performance management, or CPM, systems) provide a high-level, slow-motion form of BAM. They are used to measure and manage business performance against plans and objectives. They provide visibility into issues such as "Are we on track to meet our monthly sales targets?" or "Will any of our operating divisions overspend their capital budgets this quarter?" Like BAM systems, performance-management systems often use dashboards, and most of the data that they use is event data. However, performance management differs from operational BAM in certain ways:

- Performance-management dashboards typically display key performance indicators (KPIs) and other metrics that are calculated from totals, averages, or other summaries of event data that have accumulated over days or weeks. By contrast, most event data used in BAM systems is a only few seconds or minutes old and it is often narrower in scope.
- The end user of a performance-management system is typically higher level than the line managers or functional decision-makers that use most operational BAM systems.
- Decisions made on the basis of a performance-management system are generally tactical in nature and have a medium-term time horizon. By contrast, BAM systems are generally targeted as more urgent, although often less consequential, decisions.
- Performance-management systems virtually always have a person in the loop. By contrast, some BAM systems have automated decision-making and automated responses because the decisions are simpler and the threat or opportunity may be more urgent.

BAM in Action

BAM is a broad term that covers a wide range of applications. Most BAM systems are not actually called BAM; instead, they are known by their business purpose. Most BAM implementations are "stove-piped"—that is, they monitor one or a few things within a single functional area, one process, or one application system. Piecemeal, stove-piped BAM can be quite valuable because it provides visibility into the key metrics that matter to a particular task. However, it falls short of providing the comprehensive situation awareness that is helpful for certain decisions in certain jobs. Over time, we expect that BAM stove pipes will become more integrated. A single dashboard (or "cockpit") will provide cross-business unit, and cross-business process, monitoring capabilities.

Rule Engines and Event Processing

Although CEP software is a type of rule engine, it differs from business rule engines (BREs) as commonly understood. A BRE is a software component that executes busi-

ness rules that have been segregated from the rest of the application software. Most rules are expressed declaratively, rather than procedurally, which implies that the developer has provided a description of what is supposed to be done rather than a step-by-step sequential algorithm for how to perform the computation. The BRE translates the declarative rules into the actual computer instructions that will implement the logic.

BREs are typically packaged into a comprehensive business rule management system (BRMS) product that incorporates a variety of features that complement the core rule engine. A BRMS stores rules and checks them for logical consistency. A BRMS is based on the premise that business rules usually change more often than the rest of the application, so the application will be easier to modify if the rules are managed and stored separately from the other logic.

CEP (rule) engines have many of the same characteristics as BRMS BREs (rule engines). Both externalize business rules and are commonly packaged as discrete software components. Much of the data that is processed by a BRE and all of the data processed by a CEP engine is event data. Both types of engines can be used to support human decision-making or to compute fully automated decisions. However, they are constructed differently because they serve different purposes.

BREs are typically request-driven. The general model for a rule is If <some condition> then <do action X> or else <do action Y>. By contrast, CEP engines are typically event-driven. They usually run continuously, processing notification messages as soon as they arrive, in accordance with the principles of EDA. The general model for a CEP rule is When <something happens or some condition is detected> then <do action X>.

The internal design of a BRE is optimized for its request-driven mode of operation. By contrast, CEP systems directly handle the I/O to and from messaging systems, which makes them faster and more efficient at receiving and preparing event data that is delivered through such channels. Time is fundamental to a CEP system, so CEP systems are naturally well suited to work with event data that is grouped according toe time windows when the events occurred. BREs can also apply rules to data that is selected according to time windows, but this type of operation is generally cumbersome to specify at development time compared to performing the same calculation with a CEP system.

BREs are designed around the concept of a working set of data. However, each transaction is logically independent so the work can be spread across dozens of copies of the application on dozens of systems. CEP engines are appropriate when large amounts of potentially related data must be manipulated quickly as a set. The technology that is used in typical BRMSs is inherently different from that used in most CEP engines. Nothing would prevent a vendor from creating a single BRE/CEP product that is good at both kinds of rule processing, although it would need to implement both technical architectures. In general, BRMSs and CEP systems are complementary notions. Together, they are the core technology needed to implement intelligent decision management (IDM) programs.

Action Items

To succeed at event processing, an enterprise has to do six things well:

Action Items for Event Processing Success

- Determine where to use event processing and which type of event processing (simple EDA or CEP) to employ.
- Acquire event processing skills by training staff or hiring outside consultants.
- Incorporate event processing into the company's IT architecture.
- Buy event-enabled packaged applications.
- Implement an infrastructure that facilitates event processing.
- Integrate event processing into SOA initiatives.

Determine Where to Use Event Processing

There are no event processing projects per se, but only application development projects that use event processing in some parts of a system. For every major project that involves designing a new or modified system, architects must determine whether simple EDA, CEP, both, or neither should be used. The CEP value proposition differs from that of simple EDA. Simple EDA processes one event at a time and is intended to improve timeliness, agility, and information availability. By contrast, CEP systems operate on multiple events at a time to generate insights into what is happening. EDA helps people and systems react more quickly, whereas CEP helps people acquire situation awareness and make better decisions. When CEP is implemented in an EDA mode, both sets of goals can be achieved.

Educate Your IT Staff and Leverage Outside Expertise

Business and system analysts, architects, project leaders, and anyone else involved in collecting application requirements, modeling business processes, and developing high-level system designs must understand when and how to use EDA and CEP.

Companies that want to accelerate the development or lower the risk of a project that involves event processing may also hire architects, developers, or project managers from a system integrator or software vendor. Consultants at a system integrator with a background in EDA and CEP are typically associated with a BPM or SOA team. Most CEP software vendors have consulting practices; these are usually good sources of product-specific advice and general help on CEP implementation. Virtually all vendors of message oriented middleware (MOM), enterprise service buses (ESBs), and other SOA infrastructure have consultants who understand simple EDA and its role in modern application development. Some software vendors and system integrators offer application templates or industry frameworks that include software products, data models, best practices, and sample application flows that incorporate event-

processing features. Outside experts may help your staff quickly learn how to use EDA and CEP, especially if your people actively work with them.

Incorporate Event Processing into IT Architecture

Some leading-edge corporate IT architecture programs are beginning to pay explicit attention to event processing, although most architecture initiatives historically did little or nothing to address it. Companies that have formal architecture programs should document their policies on when to use EDA and CEP and where to apply event-driven, request-driven, and time-driven patterns. Companies that have a review process that verifies system design as part of the development cycle should examine the conceptual design of every new system to confirm that EDA and CEP have been utilized where appropriate.

Buy Event-Enabled Packaged Applications

The presence of event processing features should be a factor when selecting commercial off-the-shelf–packaged applications, for the same reasons that it is a factor in the design of good custom applications. Many packaged applications can now emit event notifications for many types of business events. This is usually a configuration option so developers who are installing and tailoring the package must study the business requirements to identify what type of notifications should be sent. Many packaged applications also have business dashboards or other monitoring and alerting features as part of the product. The vendor will rarely refer to these as "CEP" capabilities, although they technically qualify as a limited type of CEP.

Deploy Event Processing Infrastructure

Event processing is a design concept that doesn't necessarily require any particular type of middleware or development tool. However, many event processing applications would be impractical unless the appropriate commercial middleware infrastructure, CEP, development, or monitoring tools are used. EDA and CEP systems need some messaging infrastructure to provide the channels and dissemination network that will convey the notifications from event producers to event consumers. All large companies already use the Web, e-mail, and other basic message-capable communication protocols, and most already use MOM, ESBs, and SOAP in some locations. Therefore, more than half of all event processing projects don't need any new messaging software because they can use the infrastructure that is already in place. However, projects that are implementing demanding new applications that have high volume, low latency, high integrity, or other requirements may need to acquire MOM or other infrastructure products if they are not already present.

Some demanding CEP applications and most dashboards are built with standard development tools or off-the-shelf utilities. For low volume or simple applications, this is usually still the best strategy. However, more-demanding CEP applications require sophisticated algorithms and specially designed internal architectures, so companies

will be better served by commercial CEP technology for some applications. Commercial CEP, dashboard, and monitoring technology can be acquired as part of a business event processing (BEP) platform, a CEP-enabled application, such as a financial trading platform or supply-chain management product, or as a dedicated CEP engine, appliance, or other point product.

Integrate Event Processing into SOA Initiatives

Many development projects that implement simple EDA, CEP, or both are promoted under the aegis of an SOA or BPM strategy. EDA, SOA, BPM, and CEP are compatible concepts that are often used together. In view of the overlaps, companies should fold much of their EDA and CEP work into their SOA programs. They should use the same competency center and governance practices for event-driven SOA used for other forms of SOA. The SOA competency center may have one or two architects or analysts who have more training or experience in EDA or CEP than other members of the team, but they should all be part of one team.

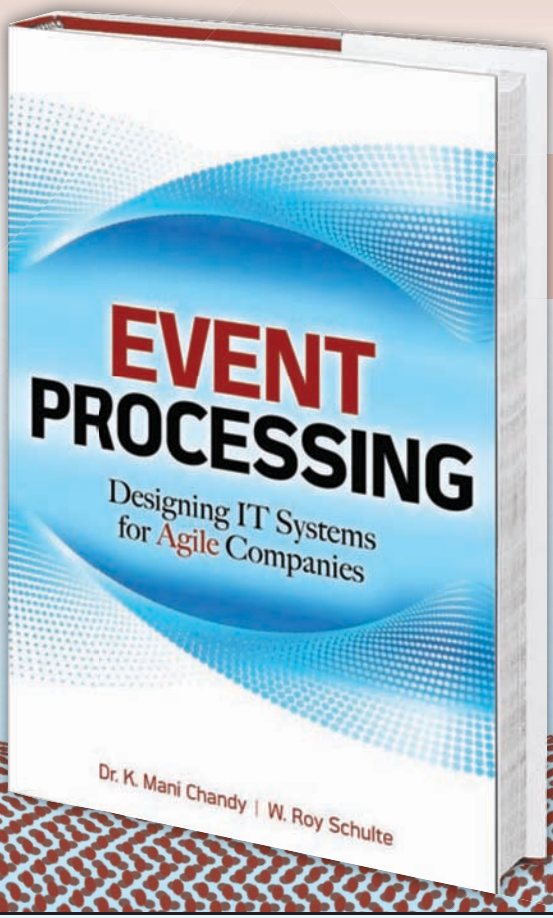
It is a mistake to implement an SOA program that can't support event-driven SOA and some type of business monitoring from its inception. Some companies have shied away from event-driven SOA, consciously or not, to focus exclusively on traditional request-driven SOA. However, this results in over-using batch processing and request-driven SOA in new SOA systems. There is no benefit to deferring adoption of event-driven SOA—it's not a major burden to undertake and its advantages should be tapped even by the initial SOA applications.

However, some EDA and CEP projects don't belong to an SOA or BPM project for organizational and technical reasons. They may have specialized event sources (such as third-party event data feeds), their own ways to define event types, their own special communication protocols and data formats, and alternative monitoring tools and metadata management technology.

Conclusion

The discipline of event processing makes SOA more effective by adding the notion of event-driven services (ED-SOA). Business events are inherent in BPM, although event objects are not always used when BPM tools deal with business events. CEP technology is used for process monitoring and for some sophisticated run-time BPM flow management. Event processing is not implemented as a separate application system or IT project; it is used as part of IT systems that may be conventional in other ways. Companies should modify their IT architectures and packaged application selection processes to include event processing concepts. Companies should incorporate EDA and CEP-based monitoring capabilities in all SOA and BPM programs.

IMPLEMENT AN EFFECTIVE EVENT-PROCESSING SOLUTION



Available September 2009 in print and e-book format

Dr. K. Mani Chandy, professor of computer science at California Institute of Technology

W. Roy. Schulte, vice president and distinguished analyst at Gartner, Inc.

Discover how to design, deploy, and use event-processing (EP) systems to detect and respond to business anomalies, threats, and opportunities. *Event Processing* first explains how this methodology compares and contrasts to the IT architectural styles used in conventional business applications. The book then discusses the types of software needed to develop and run EP applications, helping you form a deployment plan and purchase the appropriate tools. Real-world examples illustrate successful EP implementations.

Learn more. Mc Graw Hill Do more.

MHPROFESSIONAL.COM

EVENT PROCESSING

Designing IT Systems for Agile Companies

Enterprises must respond ever more quickly to unpredictable events and new threats and opportunities in their environment. Event processing has emerged as the key enabler for situation awareness and a set of guiding principles for systems that can adapt quickly to shifts in customer demand and market conditions. Written by experts in the field, this book explains how to use event processing in the design of business processes and the systems that support them. *Event Processing: Designing IT Systems for Agile Companies* covers:

- The role of event processing in enabling business dashboards and situation awareness
- Types of event processing applications and their costs and benefits
- How event-driven SOA complements conventional request-driven SOA
- How to implement event processing without disrupting existing applications

K. Mani Chandy, Ph.D., is a professor of computer science at the California Institute of Technology. He is a member of the National Academy of Engineering.

W. Roy Schulte, M.S., is a Vice President and Distinguished Analyst at Gartner, Inc. He has more than 25 years of experience spanning user enterprises, IT vendors, and analyst firms.

ISBN 978-0-07-163711-4
MHID 0-07-163711-7

9 780071 637114 51495

$14.95 USA