TIBCO
The Power of Now

Realizing the Two-Second Advantage

Architecting Complex-Event Processing Solutions with TIBCO

Paul C. Brown

Architecting Complex-Event Processing Solutions with TIBCO®

Architecting Complex-Event Processing Solutions with TIBCO®

Paul C. Brown

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearsoned.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Brown, Paul C.
Architecting complex-event processing solutions with TIBCO / Paul C. Brown.
pages cm
Includes index.
ISBN 978-0-321-80198-2 (pbk. : alk. paper) — ISBN 0-321-80198-9 (pbk. : alk. paper)
1. Business logistics—Data processing. 2. Event processing (Computer science)
3. TIBCO Software Inc. I. Title.
HD38.5.B76 2014
658.50285'53—dc23 2013026369

ISBN-13: 978-0-321-80198-2
ISBN-10: 0-321-80198-9

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, September 2013

To Mugs and Willie

This page intentionally left blank

Contents

Preface

Complex-Event Processing

Complex-event processing is a nontraditional style of building solutions. This style makes it possible to address problems that do not yield well to traditional approaches such as real-time situation analysis. More broadly, complex-event processing enables the enterprise to sense, analyze, and respond to its business situations in new and innovative ways—ways that provide extreme value and competitive advantage.

In complex-event processing solutions, the word *complex* comes into play in two very different ways. The first refers to sensing, analyzing, and responding to what is going on. It's not just, "Oh, this event occurred, therefore I need to do <some activity>." It's more complex than that: It requires correlating that event with other events and with contextual information in order to understand whether a situation of business importance exists, and then deciding what, if anything, needs to be done. Complexity in sensing, complexity in analyzing, complexity in responding.

The other way that complexity applies is that complex-event processing involves a wide variety of computational techniques. There is no single approach to sensing, analyzing, and responding that is suitable for all types of situations. Each of the approaches has its own strengths and weaknesses, all of which need to be understood in order for you to craft your solution.

About This Book

This book provides an introduction to the complex-event processing space and the computational approaches enabled by TIBCO BusinessEvents®. It is divided into four parts: Getting Started, Technology, Design Patterns, and Deployment.

Part I, Getting Started, provides a conceptual overview of the complex-event processing space. It discusses how complex-event processing can be employed in a business context to provide competitive differentiation, covers the terminology of complex-event processing, and explores the ways in which complex-event processing is different from traditional computing. It also explores a number of business applications for complex-event processing.

Part II, Technology, covers the capabilities of the TIBCO Business Events® product suite. It covers the TIBCO Business Events suite of products and presents a life-cycle overview of solutions based on these products. The TIBCO Business Events executable, a Java virtual machine (JVM), can be configured with combinations of five functional components: inference agents, cache agents, query agents, process agents, and dashboard agents. Inference agents process rules, and cache agents provide the information-sharing mechanism within TIBCO BusinessEvents. Query agents provide both snapshot and continuous queries of cached information. Process agents provide orchestration capabilities, while dashboard agents provide real-time visualization capabilities. The architecture and functionality of each type of agent are explored.

Part III, Design Patterns, explores the building-block design patterns used in constructing complex-event processing solutions with TIBCO BusinessEvents. Patterns for recognizing situation changes, comparisons and changes to reference data, systems of record, handling duplicate inputs, run-time rule changes, and orchestrating actions are explored. Patterns for pattern recognition, integration, solution modularization, information sharing, locking, load distribution, and sequencing are covered.

Part IV, Deployment, covers the architecturally significant aspects of putting a solution into production. The Nouveau Health Care case study is a realistic design problem that illustrates many of the issues an architect needs to address. It is used as an example to explore performance, modularization for deployment, managing the cache and backing store, defining deployment patterns, and monitoring. Design patterns for solution fault tolerance, high availability, and site disaster recovery are discussed, along with best practices for the conduct of complex-event processing projects.

The organization of the book is shown in Figure P-1.

Online Examples

Many of the examples in this book are taken from actual TIBCO BusinessEvents projects that are available online. All of these projects begin with the prefix ACEPST and can be found at informit.com/title/9780321801982.

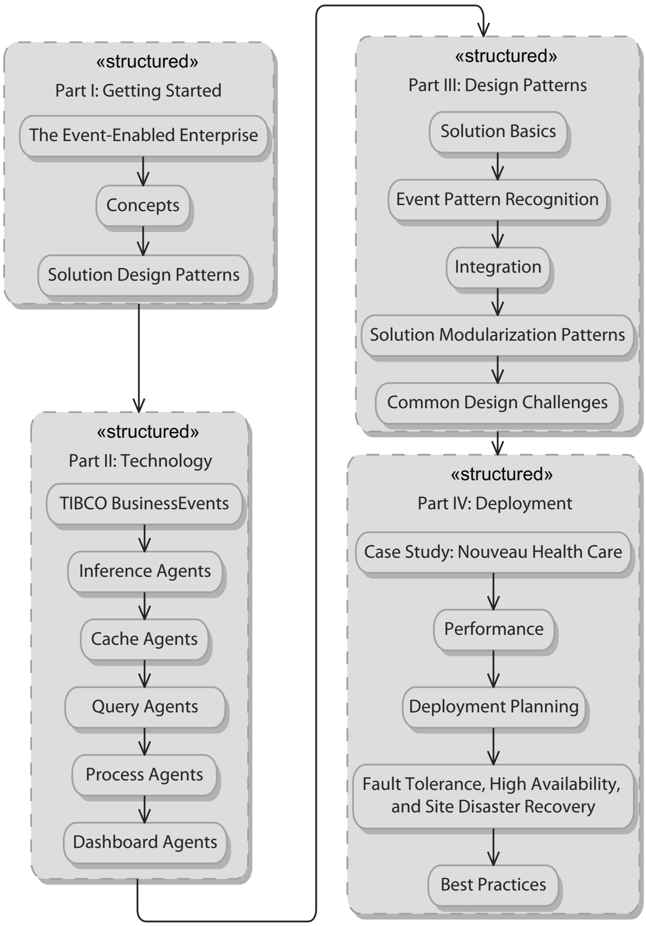

Figure P-1: *Organization of the Book*

TIBCO Architecture Book Series

Architecting Complex-Event Processing Solutions with TIBCO® is the third book in a series on architecting solutions with TIBCO products (Figure P-2). It builds upon the material covered in *TIBCO® Architecture Fundamentals*, which provides material common to all TIBCO-based designs. Each of the more advanced books, including this one, explores a different style of solution, all based on TIBCO technology. Each explores the additional TIBCO products that are relevant to that style of solution. Each defines larger and more specialized architecture patterns relevant to the style, all built on top of the foundational set of design patterns presented in *TIBCO® Architecture Fundamentals*.

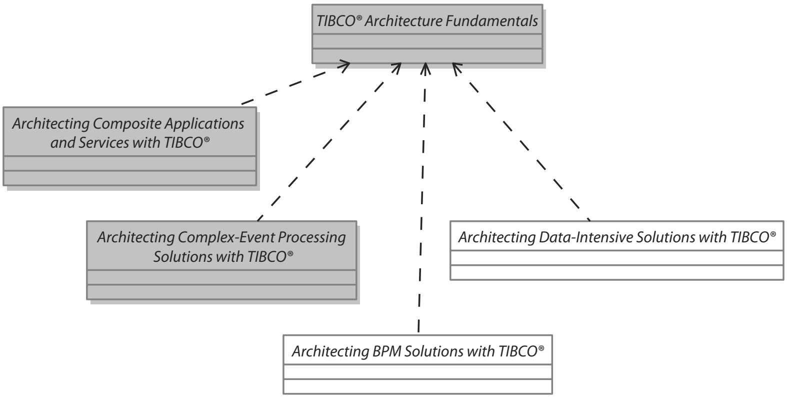

Figure P-2: *TIBCO Architecture Book Series*

Intended Audience

Project architects are the intended primary audience for this book. These are the individuals responsible for defining an overall complex-event processing solution and specifying the components and services required to support that solution. Experienced architects will find much of interest, but no specific prior knowledge of architecture is assumed in the writing. This is to ensure that the material is also accessible to novice architects and advanced designers. For this latter audience, however, a reading of *TIBCO® Architecture Fundamentals*[1] and *Architecting Composite Applications and Services with TIBCO®*[2] is highly recommended. These books explore integration and services along with the broader topics of solution architecture specification and documentation.

TIBCO specialists in a complex-event processing center of excellence will find material of interest, including background on TIBCO BusinessEvents product suite and related best-practice design patterns. The material on performance and tuning lays the foundation for building high-performance applications based on the product suite.

1. Paul C. Brown, *TIBCO® Architecture Fundamentals*, Boston: Addison-Wesley (2011).
2. Paul C. Brown, *Architecting Composite Applications and Services with TIBCO®*, Boston: Addison-Wesley (2013).

Enterprise architects will find content of interest as well. The collection of design patterns, in conjunction with those presented in *TIBCO® Architecture Fundamentals*, provides the basis for a baseline set of standard design patterns for the enterprise.

Detailed Learning Objectives

After reading this book, you will be able to

- Describe the characteristics of an event-enabled enterprise
- Explain the concepts related to complex-event processing
- List examples of complex-event processing solutions
- Describe the TIBCO BusinessEvents product suite
- Explain the operation and tuning of TIBCO BusinessEvents agents
- Explain how situations and changes in situations can be recognized
- Describe how rules can be changed at runtime
- Explain how activities can be orchestrated
- Describe how patterns of events can be recognized
- Modularize complex-event processing solutions to facilitate maintainability and scalability
- Describe how to share information among distributed components of a complex-event processing solution
- Select and apply appropriate patterns for load distribution, fault tolerance, high availability, and site disaster recovery
- Explain how design choices impact agent performance
- Define deployment patterns for complex-event processing solutions
- Describe the best practices for conducting complex-event processing projects

This page intentionally left blank

Acknowledgments

I must begin by acknowledging the extraordinary contribution of Rodrigo Abreu in formulating the content of this book and the accompanying TIBCO Education course. Not only was he instrumental in helping me fine-tune the scope of material to be covered, but he also clarified (often by experiment) many questions about actual product behavior. This was all in addition to his "day job," and often on his own time. It is fair to say that without his assistance, neither this book nor the accompanying course would exist. I am in his debt.

This book and accompanying course started out as a conversation with Paul Vincent in September 2010. The outline we put together at that time has stood the test of time and it can still be clearly recognized in the finished product. Wenyan Ma made many valuable contributions in defining the scope of material to be covered, and Michael Roeschter made significant contributions to the content.

Many others have contributed to the technical content: Pranab Dhar, Sergio Gonik, Ryan Hollom, Fatih Ildiz, Ali Nikkhah, Mitul Patel, Nicolas Prade, Patrick Sapinski, Rajarsi Sarkar, Rajesh Senapathy, Piotr Smolinski, Suresh Subramani, Piotr Szuszkiewicz, and Yueming Xu. I am grateful for their contributions.

My deepest thanks to the TIBCO Education team who worked with me on this project: Alan Brown, Mike Fallon, Michelle Jackson, and Madan Mashalkar. In more ways than I can mention, they made it all happen. Special thanks to Jan Plutzer, who strongly supported this effort from its inception.

Without the strong support of Eugene Coleman, Paul Asmar, and Murat Sonmez, I would not have been able to dedicate the time necessary for this effort. You have my deepest gratitude.

I would like to acknowledge those who took the time to review the manuscript: Abby Brown, Antonio Bruno, Benjamin Dorman, Jose Estefania, Lloyd Fischer, Alexandre Jeong, James Keegan, Lee Kleir, Edward Rayl, Michael Roeschter, and Mark Shelton. Your feedback has significantly improved the book.

I would like to thank the folks at Addison-Wesley for their continued support. Peter Gordon, my editor, has been a thoughtful guide through five books. Kim Boedigheimer continues to work behind-the-scenes magic to make things happen. The production team Julie Nahil, Megan Guiney, and Diane Freed did their usual fine work in making this book a reality.

Finally, I would like to thank my wife, Maria, and my children, Jessica and Philip, for their love and support.

About the Author

Dr. Paul C. Brown is a Principal Software Architect at TIBCO Software Inc. His work centers on enterprise and large-scale solution architectures, the roles of architects, and the organizational and management issues surrounding these roles. His total architecture approach, the concurrent design of both business processes and information systems, can reduce project duration by 25 percent. He has architected tools for designing distributed control systems, process control interfaces, internal combustion engines, and NASA satellite missions. Dr. Brown is the author of *Succeeding with SOA: Realizing Business Value Through Total Architecture* (2007), *Implementing SOA: Total Architecture In Practice* (2008), *TIBCO® Architecture Fundamentals* (2011), *Architecting Composite Applications and Services with TIBCO®* (2012), and *Architecting Complex-Event Processing Solutions with TIBCO®* (2014), all from Addison-Wesley, and he is a coauthor of the *SOA Manifesto* (soa-manifesto.org). He received his Ph.D. in computer science from Rensselaer Polytechnic Institute and his BSEE from Union College. He is a member of IEEE and ACM.

This page intentionally left blank

This page intentionally left blank

Chapter 3

CEP Solution Design Patterns

Objectives

There are many different architectural patterns that arise in complex-event processing (CEP) solutions. While all add one or more sense-analyze-respond processes to the enterprise, the manner in which they do so varies widely. This chapter identifies the kinds of variation you can expect and presents a number of well-understood patterns, each of which addresses a common business challenge.

After reading this chapter you will be able to explain the variability in CEP architectures and describe the following patterns:

- Condition Detection
- Situation Recognition
- Track and Trace
- Business Process Timeliness Monitor
- Decision as a Service
- Situational Response
- Orchestrated Response

You will also be able to explain the challenges associated with pioneering projects that develop new solution patterns.

Variability in CEP Architectures

The core CEP process (Figure 3-1) is always the same: Some event is sensed, it is analyzed in the context of some reference data to determine whether something of business interest has occurred, and some decision is made about what the nature of the response ought to be. Yet despite the fact that the core process is always the same, there are many different architectures for complex-event processing. Why?

There are two dominant reasons for the variability in CEP architectures: the handling of reference data and the partitioning of functionality.

Handling Reference Data

The first area of variability centers around the relationship between the reference data and the events being analyzed: Does the stream of events alter the reference data that is used to interpret subsequent events? Applications in which the stream of events does not alter the reference data are relatively straightforward. The primary challenge in these applications is obtaining access to the reference data, which almost always originates elsewhere, and making access to the data efficient during the analysis and response activities.

On the other hand, applications in which the stream of events modifies the reference data are much more complicated. The portion of the reference data that represents the history of prior events does not have a system of record, at least without additional design. If it is unacceptable to lose this historical data when systems are shut down or fail, then the CEP solution must now include a system of record for the historical data. The system of record requires careful and clever design to ensure that it can handle the stream of data changes efficiently and robustly—and still make the data efficiently accessible to the analysis and response activities (Figure 3-2).

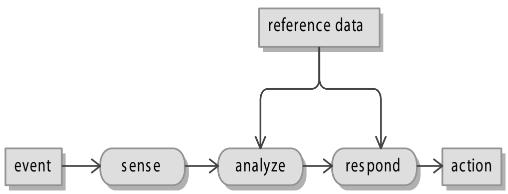

Figure 3-1: *Core CEP Process*

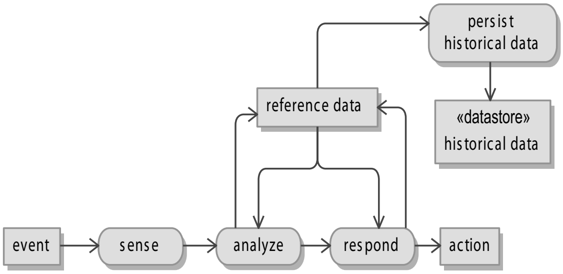

Figure 3-2: *Persisting Historical Data*

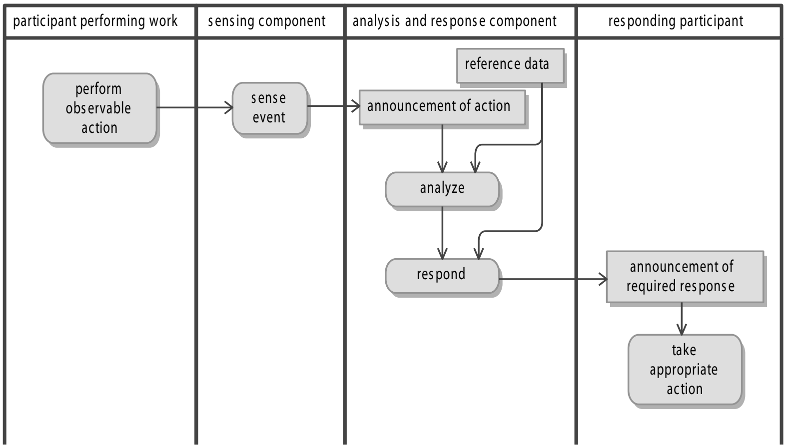

Figure 3-3: *Basic CEP Functional Partitioning*

Partitioning Functionality

The other area of variability lies in the many ways in which the CEP functionality can be partitioned and assigned to different components. The basic partitioning found in CEP solutions is shown in Figure 3-3.

Generally, the events driving the process are the observable actions of a participant (human or system) in some business process. Most of these participants do not announce their activities, at least to components not engaged in that business process. For this reason, CEP solutions generally have one or more components dedicated to sensing these actions and announcing their observations.

The techniques used for these observations are the same ones traditionally used in application integration. These techniques, and the products that support them, are detailed in *TIBCO™ Architecture*

Fundamentals.[1] The relevant observation here, however, is that the products used for sensing are, for the most part, not the products used for CEP analysis and response. Thus the participant that does the sensing is generally not the participant doing the analysis and response.

As a side note, one of the hallmarks of the event-enabled enterprise is that its architecture includes the types of components necessary to sense and announce actions and the types of components necessary to analyze and respond to those announcements.

In many cases, the volume of events handled by many CEP solutions makes it impractical to have one component handle all of the events and perform all of the analysis and response processing. Once this point is reached, there are a variety of ways in which performance can be increased. One is to simply deploy multiple instances of the component performing the analysis and response. This is a straightforward approach if the reference data is not updated when events occur. But when the reference data is updated by events, sharing the history across multiple instances of the analysis and response components requires additional design. The design patterns for this are discussed in Chapter 14.

Another approach to scalability is to begin to partition the functionality across additional components. Figure 3-4 shows one possible partitioning in which the analysis that leads up to situation recognition is performed by one component and the determination of the required responses is performed by another. Partitioning patterns also become more complex when the analysis and response computations also

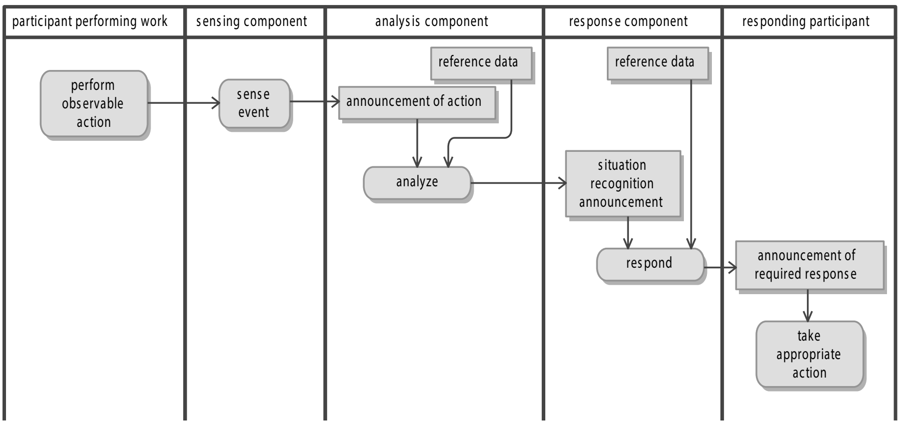

Figure 3-4: *Partitioning Situation Recognition from Response*

1. Paul C. Brown, *TIBCO® Architecture Fundamentals*, Boston: Addison-Wesley (2011).

update reference data. Chapter 13 discusses this and other partitionings as well as the tradeoffs that need to be considered.

As should be obvious by now, there are many possible functional partitionings for CEP solutions. Some lead to simple and straightforward implementations. Others require clear architectural thinking to achieve the desired behavior in a robust and highly scalable fashion.

The following sections discuss a number of CEP solution design patterns, each focused on providing a commonly required business capability. For the most part, the patterns are arranged somewhat in order of increasing complexity. The chapter concludes with a brief discussion of problems for which there may not be well-established design patterns.

For simplicity, the sensing component is not shown in these design patterns: It is assumed to be always present.

Condition Detection

The simplest solution pattern you will encounter in complex-event processing is threshold detection (Figure 3-5). In this pattern, a component takes an action that can be observed and results in a technical event. The condition detector is listening for this event, whose arrival serves as the trigger for analysis. The analysis compares a value conveyed by the event to a threshold value and, if the event value exceeds the threshold value, generates a business event announcing this condition. Completing the pattern, another component is listening for these announcements and taking appropriate actions.

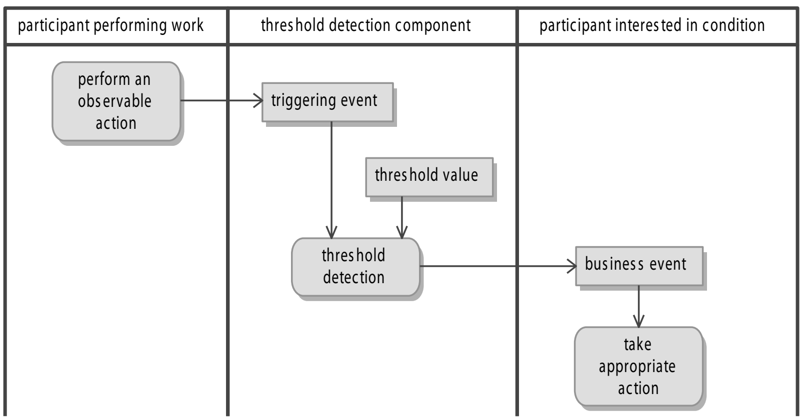

Figure 3-5: *Threshold Detection Pattern*

In using this pattern the location of the threshold value must be considered. One option is to permanently fix the threshold value in the analysis logic. Another option is to make it a piece of contextual information that is looked up by the condition detector, either when it starts or each time an event is analyzed. Yet another option is to use infrastructure that makes it possible to change the value at runtime. TIBCO BusinessEvents® rule templates provide this capability, as described in Chapter 10.

The more general form of this pattern is the Condition Detection pattern (Figure 3-6). In this pattern the detected condition is defined by a number of values that define the boundaries of the condition being recognized. The information considered in the analysis is generally a combination of event and contextual data. If the condition is detected, then a business event is generated announcing the existence of the condition.

When using this pattern the sources of the parameters defining the boundary conditions and the contextual data required to detect the condition must be considered, along with the possible need to change some of these values at runtime. The design effort required to provide access to information originating in other systems and make it efficiently available is often a major part of a CEP project.

In the Condition Detection pattern, the reference data that is used is not modified by the processing of events: It does not reflect prior history. The only state information being used is that conveyed by the

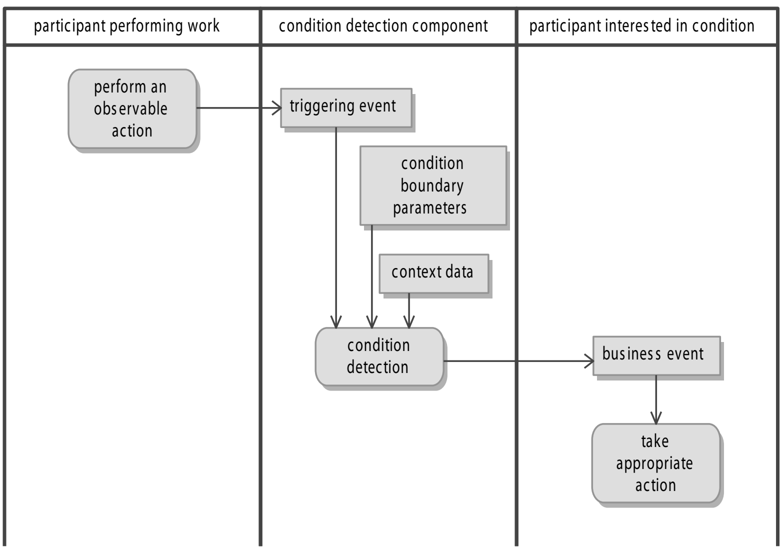

Figure 3-6: *Condition Detection Pattern*

triggering event. This makes the condition detector stateless, and therefore easy to scale and make highly available.

Situation Recognition

The Situation Recognition pattern (Figure 3-7), on the surface, looks a lot like the Condition Detection pattern. However, there is a major difference: In the Situation Recognition pattern, the context data used to recognize a situation when the triggering event arrives contains historical information. Many of the triggering events that arrive do not result in a business event, but their occurrence results in the modification of the context data. The updated context data then provides the context for evaluating the next event that arrives.

Since the context data in this pattern contains historical information, the ability of the pattern to recognize a situation may be compromised if the historical data is lost. Such a loss would occur if the situation recognition component is holding context data in memory and the component is restarted. For this reason, the use of this pattern almost always requires persisting the historical information and recovering this information when the component restarts. The object persistence discussion in Chapter 6 discusses techniques for doing this.

There are many variations on this pattern both in the manner in which the context data keeps track of prior history and the manner in which the historical information is used to interpret a current event. Chapter 10 discusses a number of design patterns that can be used for this purpose.

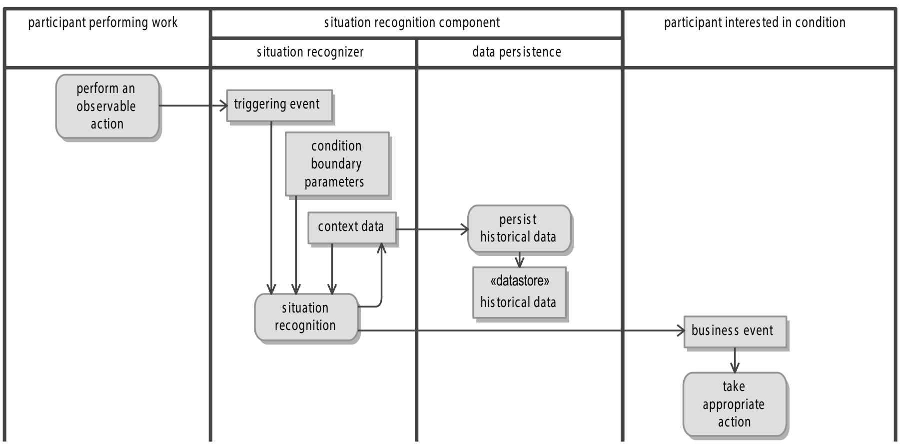

Figure 3-7: *Situation Recognition Pattern*

Track and Trace

The Track-and-Trace pattern (Figure 3-8) is a special case of the Situation Recognition pattern. This pattern involves two contextual elements: a model of the expected process and the state of an existing instance of that process. If the triggering event marks the beginning of a new process execution, an initial process state is created. For other events, information in the event is used to locate the state of the process already being executed (there may be many instances of the process being executed at any given point in time). Once the current state has been identified, the process model is then used to interpret the triggering event in the context of that state.

This simplified example omits a common challenge: the handling of out-of-sequence events. In many real-world situations, events may arrive out of sequence. In some cases, the first event that arrives may not be the initial event in the process. In a full solution, additional logic must be added to handle these situations. Chapter 14 discusses some of the design considerations.

The state machine approach provides for a rich and varied interpretation of the process execution. If the triggering event corresponds to

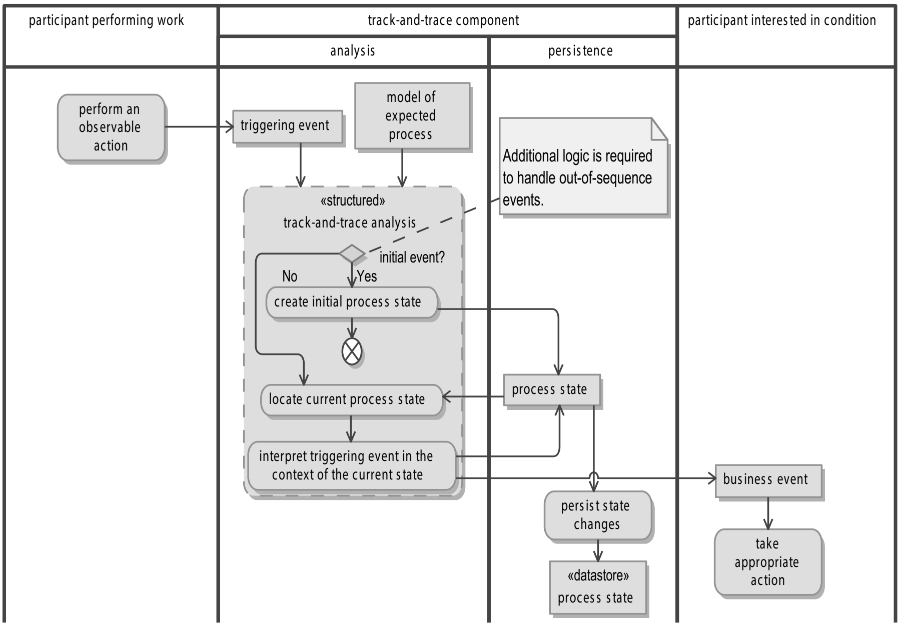

Figure 3-8: *Track-and-Trace Pattern*

an expected transition in the state machine (given the current state), the conclusion is that the process is executing in an expected manner—at least at this time. The analysis can be designed to announce business events when particular states have been achieved (i.e, announce that a milestone has been reached).

If the triggering event does not correspond to an expected transition, something unexpected has happened. Again, the analysis can be designed to emit business events announcing this unexpected situation.

This type of analysis is appropriate for monitoring any type of unmanaged process. Tracking of a package from initial pickup to final delivery is one example. Tracking your luggage from the time you drop it off at the departure airport ticket counter until the time you pick it up at the baggage carousel at your final destination is another.

In general, this approach is well suited for monitoring any process in which there is a hand-off of responsibility from one participant to another. You give your luggage to the counter agent—one hand-off of responsibility. The counter agent places the bag on the conveyer as a means of handing off responsibility to the baggage handlers. The process continues until the final hand-off, which begins when the baggage handler at your final destination places the bag on the conveyer leading to the baggage carousel and ends when you pick up your luggage.

The events being monitored in track-and-trace situations are the evidence that individual hand-offs have been successful. The challenge in most situations is finding the evidence. In the days before security requirements mandated scanning and tracking luggage on airplanes, the evidence was scanty: You got your receipt for your bag when you dropped it off (that is, when you handed it off to the airline) and you (hopefully) picked up your bag at its destination. There was little evidence available for any intermediate progress.

The security requirement that luggage not travel on a plane unless the associated passenger is also on board has resulted in better tracking—better evidence—of your luggage's progress. The luggage tracking tag is scanned when the luggage is loaded on the plane or placed in a bin that will subsequently be loaded on the plane. It is scanned again when it comes off. These scans provide intermediate evidence of progress.

Your challenge in designing a Track-and-Trace solution is going to be finding appropriate evidence of progress. It is not uncommon that the full set of evidence you would like to have is simply not available. When this occurs, you may want to implement the degree of tracking that is supported by the currently available evidence and independently begin an initiative that will eventually provide more detailed evidence of progress. This is exactly what happened in the telecommunications case study described back in Chapter 2.

Business Process Timeliness Monitor

The Business Process Timeliness Monitor (Figure 3-9) is an extension of the Track-and-Trace pattern. State machine models can be extended so that the absence of an expected event within some period of time can be recognized. While, of course, you can apply this approach to recognizing that an overall process did not complete on time, the greatest benefit comes from recognizing that some intermediate event did not occur on time, and thus the overall process is in jeopardy of being late. The recognition can be used to trigger an action that will correct the course of the overall process and get it back on track for an on-time completion. The telecommunications case study discussed back in Chapter 2 is an example of this pattern in action.

Detecting the absence of an event requires the establishment of a service-level agreement specifying the maximum amount of time it should take for the process to complete or remain in each intermediate state. When the state machine monitoring the process is started or a particular intermediate state is entered, a timer is started. When the overall process completes, or the intermediate state is exited, the corresponding timer is stopped. However, if the timer expires before the

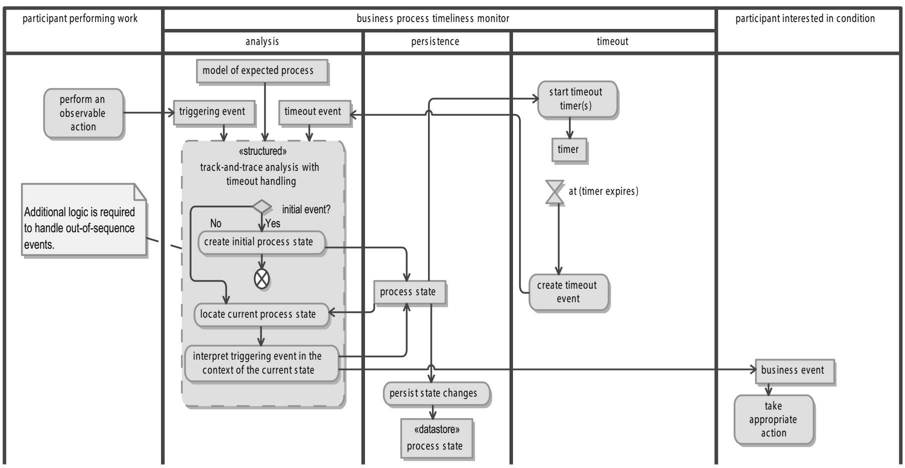

Figure 3-9: *Business Process Timeliness Monitor*

process completes or the intermediate state is exited, a timeout event is generated. This is an indication that some expected event did not occur.

In recognizing this situation, it is the expiration of the timer that serves as the trigger for the analysis. Some introspection of the state machine may be required to identify which events did not occur, but the larger design requirement is to determine which parties should be notified when this situation arises and what actions those parties are going to take to get the overall process back on track.

Situational Response

All the patterns in this chapter up to this point have had one characteristic in common: They simply recognize that some condition exists and announce that fact with an event. Other independent participants receive these notifications and decide what action to take.

In some situations there is an additional challenge in determining what the appropriate response ought to be (Figure 3-10). Further analysis is required, generally to focus the actions on achieving specific business objectives. Reference data, often containing historical information, is required for the analysis. The result of the analysis is generally one or more directives to actually perform the identified actions.

Consider the case in which there is some form of perishable commodity being sold: fresh produce and meat, seats on a plane, or hotel rooms—anything that becomes worthless if not sold by some point in

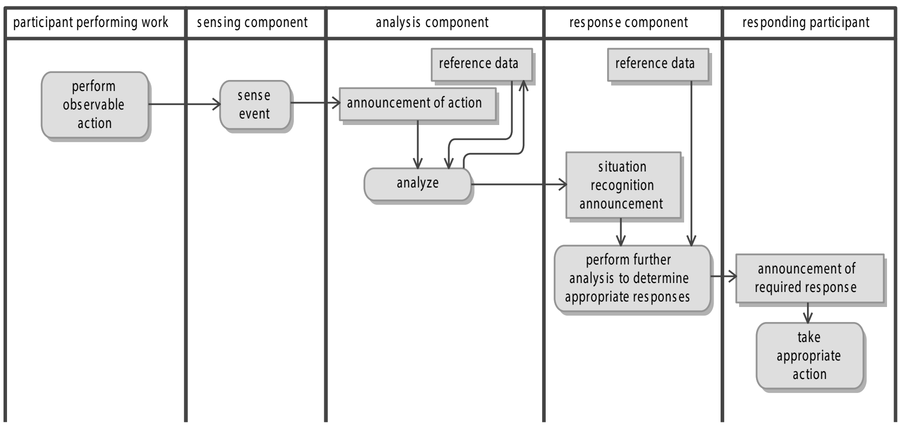

Figure 3-10: *Situational Response Pattern*

time. The desired business strategy is to dynamically set the price of the commodity based on the remaining inventory and the time remaining before the commodity becomes worthless. The situation being responded to in these cases is the presence of a potential consumer for the perishable commodity.

The simplistic approach to pricing the commodity is to fix a point in time at which it will be put on sale. The idea is that this will raise demand and ensure that the commodity does not go to waste. The problem with this approach is that it neither maximizes revenue nor minimizes the likelihood that the commodity will go to waste. If the commodity is selling well and will likely sell out, putting it on sale will result in lost revenue. On the other hand, if the commodity is selling very poorly, lowering the price by a set amount at a fixed point in time might not ensure that the commodity actually sells out.

A more sophisticated approach is to track the rate at which the commodity is selling versus the price of the commodity. With this approach, the offering price for the commodity can be adjusted dynamically. This approach is often applied to online product sales. It requires complex-event processing to do the dynamic price adjustments as consumers shop and as commodity inventories change. Note that the rate of sales and the current inventory become part of the reference data—a dynamic part whose currency must be maintained in a timely manner—most likely via events!

Decision as a Service

In the Decision-as-a-Service pattern (Figure 3-11), the logic necessary to make a decision is factored into a separate component. The service consumer gathers all relevant current-state input data for the decision and passes it to the service. This is typically a synchronous request-reply interaction, but it may be asynchronous. In either case, the decision service computes output data from the input data, using static reference data as appropriate. The output data reflects the decision results.

The value of this pattern is that it encapsulates the logic of the decision as a service. This simplifies the maintenance of both the service consumer and the decision service. In particular, it allows the implementation of the service (that is, the business rules) to be updated without requiring a modification to the service consumer.

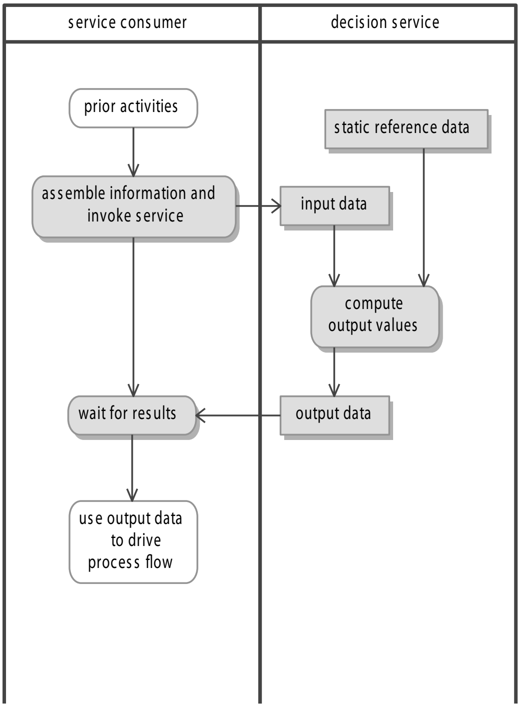

Figure 3-11: *Decision-as-a-Service Pattern*

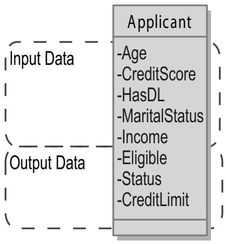

Figure 3-12: *Credit Card Decision Service Data Structure*

To make this possible, however, both the input and output data structures have to remain fixed.

Let's consider an example from the banking world. A bank needs to evaluate applications for credit cards to determine whether a credit card should be issued and what the credit limit should be on the account. In this case, the same data structure is used for both the input and output, with the difference being that some of the field values are computed by the Credit Card Decision service. Figure 3-12 shows this data structure. The input data includes the applicant's age, credit score, Declarations | Decision Table | Exception Table

Add Remove | Custom | Fit Content Merge Rows Duplicate Row Show Text

ID	applicant.Age	applicant.CreditScore	applicant.HasDL	applicant.maritalStatus	applicant.Income	applicant.Eligible	applicant.Status	applicant.CreditLimit
	– Condition Area					– Action Area		
1	0					false	invalidData	0.0
2	>= 16 && < 18	> 0 && < 500	true			true	rejected	0.0
3	>= 16 && < 18	>= 500 && < 600	true	single	> 0 && < 100000	true	rejected	0.0
4	>= 16 && < 18	>= 500 && < 600	true	married	> 0 && < 100000	true	pending	0.0
5	>= 16 && < 18	>= 500 && < 600	true		>=100000	true	accepted	500.0
6	>= 16 && < 18		false			false	notEligible	0.0
7	>= 16 && < 18	>= 600 && < 700	true		> 0 && < 100000	true	accepted	2000.0
8	>= 16 && < 18	>= 600 && < 700	true		>100000	true	accepted	3000.0
9	>= 16 && < 18	>= 700 && <= 850	true		> 0 && < 100000	true	accepted	4000.0
10	>= 16 && < 18	>= 700 && <= 850	true		>=100000	true	accepted	5000.0
11	>= 16 && < 18			divorced		true	rejected	0.0
12	>= 16 && < 18	0				false	invalidData	0.0
13	>= 16 && < 18				0	false	invalidData	0.0
14	>= 18 && < 26	> 0 && < 500				true	rejected	0.0
15	>= 18 && < 26	>= 500 && < 600		single	> 0 && < 100000	true	rejected	0.0
16	>= 18 && < 26	>= 500 && < 600		married	> 0 && < 100000	true	pending	0.0
17	>= 18 && < 26	>= 500 && < 600			>=100000	true	accepted	2000.0
18	>= 18 && < 26	>= 600 && < 700			> 0 && < 100000	true	accepted	5000.0
19	>= 18 && < 26	>= 600 && < 700			>=100000	true	accepted	7000.0
20	>= 18 && < 26	>= 700 && <= 850			> 0 && < 100000	true	accepted	8000.0
22	>= 18 && < 26	0				false	invalidData	0.0
23	>= 18 && < 26				0	false	invalidData	0.0
24	>= 26 && < 75	> 0 && < 500			> 0 && < 100000	true	rejected	0.0
25	>= 26 && < 75	> 0 && < 500		single	>=100000	true	rejected	0.0
26	>= 26 && < 75	> 0 && < 500		married	>=100000	true	pending	0.0
27	>= 26 && < 75	>= 500 && < 600		single	> 0 && < 100000	true	rejected	0.0
28	>= 26 && < 75	>= 500 && < 600		married	> 0 && < 100000	true	pending	0.0
29	>= 26 && < 75	>= 500 && < 600			>=100000	true	accepted	2000.0
30	>= 26 && < 75	>= 600 && < 700			> 0 && < 100000	true	accepted	5000.0
31	>= 26 && < 75	>= 600 && < 700			>=100000	true	accepted	7000.0
32	>= 26 && < 75	>= 700 && <= 850			> 0 && < 100000	true	accepted	11000.0
33	>= 26 && < 75	>= 700 && <= 850			>=100000	true	accepted	20000.0
34	>= 26 && < 75	0				false	invalidData	0.0
35	>= 26 && < 75				0	false	invalidData	0.0

Figure 3-13: *Decision Table for the Credit Card Decision Service*

a flag indicating whether or not the applicant has a driver's license, another flag indicating whether they are married, and the applicant's income. The computed output values comprise a Boolean indicating whether the applicant is eligible, a field indicating the current status, and another field indicating the credit limit should the status be accepted.

A decision table describing the logic for this service is shown in Figure 3-13. This example is developed using the TIBCO BusinessEvents® Decision Manager, which is described in Chapter 10. Each line of the table defines a set of conditions for the input values (the Condition Area) and the corresponding computed output values (the Action Area).

The Decision-as-a-Service pattern is useful when the business rules change frequently but the data used to drive the decision and the outputs of the decision can be fixed.

Orchestrated Response

While process orchestration is not a traditional focus of complex-event processing, the need to orchestrate portions of CEP solution activity is increasing in importance (Figure 3-14). In this relatively common

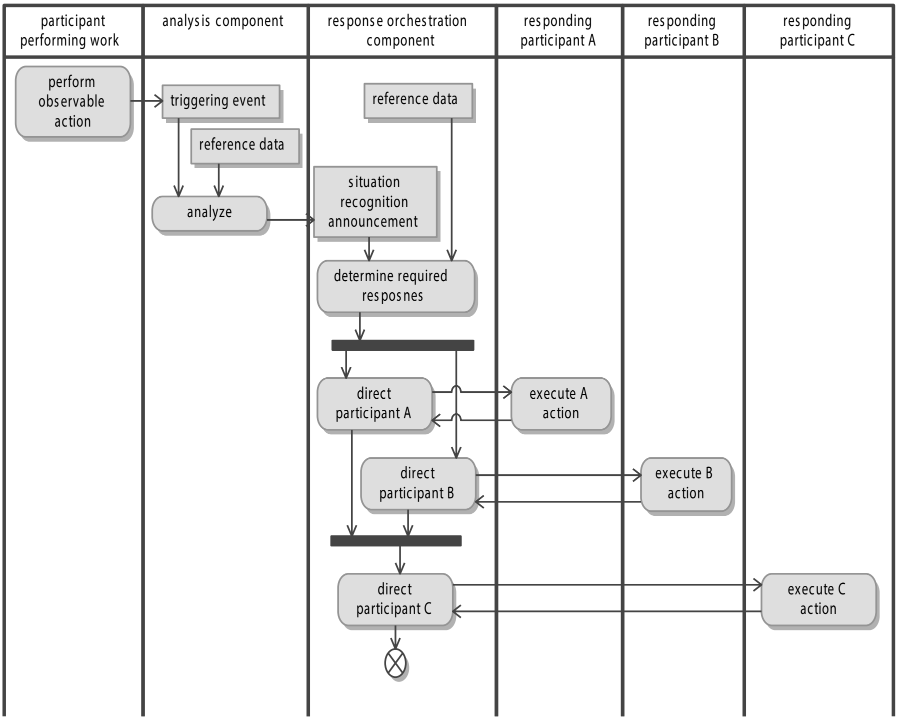

Figure 3-14: *Response Orchestration Pattern*

pattern, process orchestration is used to coordinate multiple participants in responding to a situation. The reason for the orchestration is twofold: to control the order in which the actions are performed and to confirm the successful completion of the actions. Less common is a situation in which process coordination is required for situation recognition.

This pattern is a hybrid of event-driven and request-driven interactions. All of the interactions up to the receipt of the situation recognition announcement are event driven. The response orchestration component, however, uses request-driven interactions to not only request that each participant perform its work but also to confirm the successful completion of that work.

When this pattern is used, a choice must be made regarding the type of technology to be used for the response orchestration. Traditionally, this would be a component designed specifically for process orchestration, such as TIBCO ActiveMatrix BusinessWorks™ or TIBCO ActiveMatrix® BPM. With this approach, if rule-based reasoning is required in the orchestration, the Decision-as-a-Service pattern is used. The service returns values that then guide the subsequent process execution.

However, separating process orchestration from complex-event processing may become a performance barrier, particularly if a significant amount of repetitive information must be passed to the decision service on each invocation. In such cases, it is better to have the process orchestration performed directly by a CEP component. This is the purpose of the TIBCO BusinessEvents® Process Orchestration product. It adds process orchestration capabilities to TIBCO BusinessEvents®.

Pioneering Solutions

We close this chapter on a cautionary note. Early explorers drew maps of the territories they became familiar with and drew dragons in the unexplored corners of these maps, warning those later map readers to beware of those unexplored spaces. Even worse, many explorers never even reached their goals: Columbus was seeking Asia when he found the Americas, and numerous explorers sought unsuccessfully for the Northwest Passage that would provide a North American route from the Atlantic to the Pacific.

The relevance here is that there are many types of applications for complex-event processing that have been well explored. If you are working in one of these areas, the problem is well defined, and implementing your solution will be a straightforward engineering exercise. If, however, you are working in an area that is not well defined, one in which the analytical approach for either situation recognition or action determination has not yet been established, proceed with caution. Some (but not all) of these areas are true research topics—you need to invest a little time in determining whether or not your particular problem is well defined before you commit to building a solution. Remember, it took more than 400 years to find the Northwest Passage!

How can you tell when you are on safe ground? Ask yourself the following questions:

- Is the information related to the problem understood well enough to create a quality information model (including relevant state information)?
- Is there a well-defined (i.e., measurable) set of criteria that defines the situation that needs to be recognized?
- Are there well-defined triggers that identify the points in time at which the situation recognition analysis will be performed?

- Is the information necessary for this recognition analysis readily accessible?
- Is there a clearly articulated approach for using the available information to recognize the situation?
- Is there a well-defined (i.e., measurable) approach for responding to the situation once it has been recognized?
- Is the reference information needed for determining the response readily accessible?
- Does the business value of the resulting situation recognition and response capabilities warrant the investment in the solution?

If you answered yes to all of these questions, you are on solid ground. If you answered no to any of them, you may be plowing new ground. You need to eliminate this uncertainty before you commit to producing a solution. Focus your initial efforts on developing the answers to these questions, with particular attention to the last one: Is the result worth the effort? Then, and only then, should you commit to building a solution.

The riskiest question in the list is the first: What is it that you are trying to recognize? Define your goals based on solid analytical results and beware of open-ended criteria. For example, you are never going to recognize all forms of financial fraud: The bad guys are constantly inventing new ways to scam the financial system and circumvent the checks currently in place. Identifying fraud, in general, is not an achievable goal.

On the other hand, there are specific behavior patterns that fairly reliably indicate that there might be fraud in progress. An analysis of login patterns might identify these behavior patterns, and the recognition of these patterns as they occur is definitely a well-defined and measurable goal.

If you find yourself waving your hands as you attempt to get specific about defining your recognition goals—stop! You are treading on thin ice. Do your analytical homework and convince yourself that you can be precise about what is to be recognized.

Summary

There are two factors that contribute to the variability in complex-event processing architectures. One is the handling of reference data and the extent to which the stream of events modifies the reference data used to interpret subsequent events. The other is the myriad ways in which the necessary sense, analyze, and respond activities can be partitioned and assigned to components. There is no one-size-fits-all architecture for complex event processing.

The simplest architectures are those in which the reference data is not impacted by the stream of events. The Threshold Detection and Condition Detection patterns are examples of these.

When the event stream can alter the reference data, the architecture gets a bit more complicated. The reference data now contains some historical information. If this information is essential for analysis, the solution must now become a system of record for this information. This requires persisting the information.

The Situation Recognition pattern uses historical data in its analysis. Some of the events that arrive simply result in updates to the historical data. Others, when analyzed, signify the recognition of a business-significant condition that must be announced. Track-and-Trace is a specialization of this pattern that does milestone-level tracking of a process. The Business Process Timeliness Monitor extends Track-and-Trace to determine whether the milestones are achieved on time.

Some applications require more than simply announcing that a condition exists. The Situational Response pattern applies contextual analysis to determine the actions that are required in a specific situation. The Decision-as-a-Service pattern makes these analytical capabilities available to non-CEP components. Sometimes the requirement extends beyond simply identifying the required actions to include the management of their execution. The result is the Orchestrated Response pattern.

Building a solution in which the situations to be recognized, the desired responses, and the analytical techniques to be used are all well defined is a straightforward (though sometimes complex) engineering exercise. Building a solution when any of these is not well defined has a significant degree of uncertainty. In these situations, before a commitment is made to produce a solution, preliminary work should be undertaken to clarify the approach to recognition and response. Once this preliminary work has been completed, an estimate of the effort required to implement the solution should be made to ensure that it is warranted by the expected business benefit.

This page intentionally left blank

Index

I