

What is Different About Event Driven Architecture?

K. Mani Chandy
California Institute of Technology,

Introduction

An event driven architecture (EDA) is a software architecture for applications that detect and respond to events. An event is a significant change in the state of a system or its environment.

EDA can be viewed primarily in terms of application integration — pushing loose coupling to its limit by having components generate information independent of consumers of the information, or EDA can be viewed as an architecture for sensing and responding to the environment. This note deals with the latter viewpoint.

Event driven systems have been used in military command and control systems, and in SCADA (supervisory control and data acquisition) systems for decades. Recently there has been a surge of interest in event-driven architectures for designing cyber-physical systems that integrate physical infrastructure, such as bridges and the electrical power grid, with computation and communication systems.

Platforms for event driven applications have been developed by several companies in the last decade; these general-purpose platforms can be tailored to obtain special-purpose event-driven applications. The development of event-driven platforms has evolved from several fields including databases, message-driven middleware, rule-based systems, business intelligence, statistics, sensor networks, and command and control systems.

A question that appears repeatedly in the literature is: What is different about event driven architecture compared to other architectures? One way to understand the differences is to compare the metrics used in measuring the benefits of EDA with metrics used for more common architectures. Architectures define how components are composed to form structures; so another way of studying differences between EDA and other architectures is to contrast the types of components used.

Measures for Evaluating Event Driven Applications

Event-driven applications detect and respond to events. Examples of such applications include those that: take action when earthquakes, tsunamis, or hur-

ricanes are imminent; help manage investment portfolios by sensing risks and opportunities in markets; help disabled or aging people live independently by detecting and responding to potential problems; and prevent crime by identifying and responding to threats.

Measures by which event-driven applications are evaluated include the usual measures of total cost of ownership and measures, called the *REACTS* metrics (after the first letters of the measures), which are more specific to event driven architectures.

- **Relevance:** This measures the relevance of actions taken by the system. A tsunami warning is highly relevant to a tourist on a beach where the tsunami is about to strike but less relevant to the same person a day later when she is hundreds of miles away. Systems that send a great deal of irrelevant information (spam) are likely to be ignored.
- **Effort:** This measures the effort required by operational people to define specifications of events and responses and by IT staff to integrate event applications with existing systems. The time taken by end users, such as stock traders, to tune systems to meet their specific needs, are major costs.
- **Accuracy:** This measures the costs of inaccurate information, such as false positives. A false tsunami warning gets beaches to be evacuated unnecessarily. High frequencies of inaccurate information result in information being ignored.
- **Completeness:** This measures the costs of incomplete information. For example, the absence of a hurricane warning can result in thousands of deaths. False negatives have higher costs than false positives in most situations.
- **Timeliness:** This measures the costs of detecting events too late. The benefit of detecting an event depends on the time available to execute an appropriate response. There is little added benefit in having millisecond responses when responses in minutes will do. There are, however, significant costs in responses initiated too late: a tsunami warning received after a tsunami strikes offers no value.
- **Security:** This measures the costs of reducing and managing intrusions and other attacks. Event driven applications often run for extended periods and manage critical infrastructure. For example, new infrastructure in the electrical power grid enables utilities to control airconditioners and other devices in homes and offices. Attacks that shut off power to homes and factories have severe consequences.

The design of event driven applications is a process of evaluating, comparing, and trading off REACTS measures for different design options.

Components of Event Driven Architectures

Event driven architectures consist of data acquisition components such as sensors; event processors that integrate data from multiple sensors and databases; responders that initiate appropriate actions after events are detected; communication links that transmit information between components and administrative services that help in specifying and managing the operation of event driven applications. Next, each of these components is discussed briefly.

- **Sensors:** Data acquisition components acquire data from the environment and from within the enterprise; this data is analyzed to detect patterns that signify events. Data may be pushed to these components or the components may acquire data by polling services. For example, stock ticker feeds are data acquisition components that push stock ticker information to the application without having the application make an explicit request for each stock price update. Other data acquisition components poll Web sites or other services according to fixed schedules or when requested to do so by event processors.
- **Event Processors:** The role of event processors is to integrate data from multiple sensors, add value to event objects by incorporating data from databases and business intelligence repositories, find patterns that signal events, and initiate actions by responders. Event processors may also request sensors to obtain additional data, providing closed-loop coupling between data acquisition and processing.
- **Responders:** A typical response is the initiation of a business process. Example responses include updating dashboards that display key performance indicators, sending alerts, setting up temporary task forces to respond to an event, and updating databases and logs for later analysis. The execution of the business process responding to an event may, in turn, generate new events.
- **Communication Layer:** Communication among sensors and event processors, in some applications, is by means of wireless. In some applications, wireless bandwidth may be low because sensors may be required to be inexpensive, power may have to be conserved, or the environment may be noisy. The geographical layout of components communicating by wireless is critical in such applications.

Administration Layer In addition to the basic component types and connectors, EDA includes an administration layer. The administration layer is used for several purposes including the following: initially to tailor a general-purpose event driven architecture to suit a specific application; at run time to monitor an application; and, at redesign time to replay scenarios for forensics and for adapting systems to changing conditions. Since event driven applications are often used to control physical infrastructures over long periods, monitoring sensors and responders is an essential aspect of these applications. Thus,

the administrative layer of an event driven application is itself an event driven application.

System Specification The process of implementing an event driven application is, in essence, an application integration activity: sensors, processors and responders have to be integrated into operations. The specification of an event driven application consists of specifications of the sensing, processing and responding subsystems and their integration. The subsystems are quite different from each other; therefore, the notations used in specifying them are different.

Specifications of sensing subsystems include identification of what data sources and sensors to use, where they should be placed, how they should communicate, activation and polling schedules, and mappings from data models used by the external environment to data models used within the application. Specifications of event processing subsystems deal with functionalities such as fusing, filtering and splitting that also occur in database systems, rule-based systems and business intelligence systems; therefore some notations for specifying event processors use extensions of SQL, or rules, or statistical packages. Specifications of responder subsystems identify activities carried out when an event is detected; these activities are often specified using notations for business process management (BPM) and business application monitoring (BAM).

Differentiating Features of EDA

Comparisons of the metrics used in evaluating event driven applications with metrics used in other domains highlight the differentiating features of EDA. One difference is the increased explicit emphasis in EDA metrics on the costs of error: the costs of false positives, false negatives and delays. Comparisons of the types of components used in EDA and other architectures also helps in understanding the differentiating features. EDA deals explicitly with sensors and responders as well as information processing; the architecture deals, in a fundamental way, with interactions between an enterprise and its environment including sensing activity in the environment and influencing the environment.

Service oriented architecture (SOA), viewed in terms of loose coupling and systematic interfaces, encompasses EDA; indeed one could use the term EDSOA rather than EDA. If SOA is viewed narrowly in terms of request-reply interactions between clients and servers then SOA does not encompass EDA because the metrics and components of EDA are different from those of narrowly-viewed SOA systems.

In conclusion, this note makes the point that differences in software architectures can be understood by comparing metrics used in evaluating applications for which the architectures are used and by comparing the types of components used.