Preface

THE COMMUNITY

Companies are cooperating in many different ways, from supply chains with tight inventory controls to
dynamic partnerships and virtual enterprises. This leads to an emerging class of enterprise wide appli-
cations in which monitoring of relevant situations and the timely response play an ever increasing role.
Parallel to this trend, we are seeing the massive deployment of sensors. The resulting cyber-physical
systems can monitor from low-level sensor signals to high-level abstract events, and react to them.
Taking the event processing paradigm a step further, novel applications combine sensor data with other
sources to infer threats and opportunities.

In (Hinze, Sachs, Buchmann, 2009) we surveyed 20 event-based applications and analyzed their
requirements and main features. In this book additional applications are addressed. The combined ap-
plications cover a broad spectrum:

- Financial applications, risk management and fraud detection (see chapters 2 and 9)
- Supply chain management, logistics and real time enterprises (see chapter 9)
- Environmental monitoring and warning systems (see chapters 2 and 7)
- Marine security and rescue systems (see chapters 8 and 19)
- Integrated health care systems and emergency response (see chapters 5 and 7)
- Smart environments ranging from smart cars and homes to smart cities (chapters 4, 6 and 17)
- Traffic control ranging from cars to air traffic control (see chapters 16 and 12)
- Information dissemination, alerter mechanisms and information dashboards (see chapter 18)
- Network management and service collaboration (see chapters 10, 11, 13 and 15)

The list of reactive systems, often called sense-and-respond systems, is continuously growing. The
reasons for this explosive growth is a combination of increasing demand for timely response to relevant
events (even in the face of a torrential flood of information), the ensuing need for scalability, and the
requirements to react fast to new threats and opportunities. At the same time, (complex) event process-
ing is maturing as a discipline and gaining widespread attention.

How vibrant the event processing community is can best be seen by its multiple activities: various
web sites and blogs serve as portals to relevant information and timely and informed commentary start
ups are gaining traction and are opening up new business domains; open source projects are making
event processing accessible to a broader academic clientele; early workshops, such as DEBS (Distributed
Event-based Systems) have come a long way from their early existence at the fringes of major conferences to a full-fledged conference with ACM sponsorship.

Last but not least, there are books that address a broad public such as the pioneering work by Luckham (Luckham, 2002), or the newest book by Chandy and Schulte (Chandy, Schulte, 2009) that gives practical advice to industrial users, as well as books directed more at the academic community and give a snapshot of the state of the art. Among the latter the book by Mühl, Fiege and Pietzuch (Mühl, Fiege, & Pietzuch, 2006) has a strong background in publish/subscribe technologies while the book by Chakravarthy and Jiang (Chakravarthy, Jiang, 2009) focuses more on quality of service in stream processing. The Event Processing Technical Society, which was founded in 2008, is making headway in standardization efforts.

Standardization and a common understanding are major issues in an emerging discipline that has its roots in so many areas of computer science, ranging from databases and data mining to artificial intelligence, from distributed and real-time systems to sensor fusion. Even in a book like this, the authors of the various chapters may have slightly different understanding of the terms we all use. In an attempt to further a common understanding we review here a few basic concepts and issues that will be addressed throughout the chapters of this book before addressing the individual contributions.

THE FOUNDATION

An event was defined by Chandy (Chandy, 2006) as a significant change in the state of the universe. Since time is an inherent dimension of the universe, two observations of the universe at different time constitute two distinct events, even if no other properties have changed. Chandy’s definition, however, refers to significant changes in the state of the universe, thereby limiting the infinite number of events to those that are relevant to an application. For an application it may be relevant that an object changed its position (change event), or that two different observations of a temperature sensor at different times yielded the same temperature (status event). By considering time as an integral part of the state of the universe, both change and status events can be modeled in a uniform manner.

Events must be observed, interpreted, reported and processed. An observation is a discrete instance of a (possibly continuous) signal. An observation of an event encapsulated in an event object carries a timestamp and descriptive parameters and is typically represented as a tuple of values. Parameters may be absolute values or deltas relative to older reference values. The type of event and the application domain determine the granularity of the timestamp. In distributed systems the timestamp may carry an imprecision due to clock synchronization problems. Depending on the type of event and application system, the timestamp may be just one point (point semantics of time) or an interval (interval semantics of time). Events may also be characterized by a validity interval.

Events may be simple events or compositions of simple and/or other composite events. Simple events could be individual sensor observations, method invocations or absolute temporal events. Composite events are aggregations of events that are produced from event representations and the operators of an event algebra.

Event contextualization implies the interpretation of observed events with respect to a mental model of the application. Event contextualization leads to derived events. Derived events are caused by other events and often are at a different level of abstraction. For example, five failed logins with the wrong password may cause an intrusion-attempt event to be signaled. Derived events involve semantic knowledge
and may be raised explicitly. Event contextualization often relies on external data and derived events are often enriched with data from external sources.

An event model consists of the types of events that are supported and the operators used for their composition and manipulation. Because of the wide variety of application domains and the different emphasis placed on the correlation criteria, such as time, location or other notion of context, no canonic event model is in sight. However, a tendency towards domain specific event models can be observed. The state of event description and manipulation languages parallels the state of the event models. (see chapters 3, 4, 5, 9, 15, 16, 17)

The reference architecture of an event-based system consists of event producers, event detectors, event composers, an event notification mechanism, and event consumers that trigger the application logic. The principles of an Event-Driven Architecture are postulated by Chandy and Schulte as: reporting current events; pushing notifications; responding immediately; communicating one-way; and being free of commands or requests from the consumer.

Event producers may be sensors, a clock or any component that produces events. Event composers can be both event consumers and event producers: they consume simple or composite events, operate on them and produce new events. Event detectors are a type of event producers; they may be software components that observe the environment and generate the corresponding events. For example, a software component in an RFID reader that recognizes the reading of a passive RFID tag, associates a position and timestamp with it and produces an event. Event detectors may implement a polling cycle, but they push the detected events to the consumers.

Events, or more precisely, their representation, must be reported to event consumers. An essential aspect of the event processing paradigm is that events are pushed from the event producers to the event consumers. One or more event representations are packaged into notifications. The header or envelope of a notification may contain routing information. Event notifications are routed from event producers to event consumers by a notification service. The notification service decouples producers and consumers and provides the routing from source to sink. Publish/subscribe middleware is the most flexible (but not the only) notification mechanism: event producers publish events and event consumers subscribe to events that are of interest to them. In the simplest form this may be a low level channel into which event notifications are placed by the publisher and from where they are retrieved by the subscriber (channel based pub/sub). In this case the envelope of the notification is minimal and streams of tuples are delivered over a fixed channel. However, the notification service may be a more sophisticated network of brokers routing the notifications based on type or content. Notifications consist of one or more event representations packaged in an envelope. Routing may occur on the envelope data (topic based pub/sub) or on the content of the notification (content based pub/sub). Content based pub/sub assumes a common name space used by publishers and subscribers. To deal with heterogeneous name spaces concept-based pub/sub introduced the use of ontologies to mediate and resolve ambiguities.

Event consumers are the reactive component in an event-driven architecture. They either implement the application logic in form of rules or provide a triggering mechanism to invoke other application code, e.g. services. Rules may be combined into policies. Rules may have different formats that result in different execution models. Procedural Event-Condition-Action rules are fired whenever the corresponding event (simple, composite or derived) is raised. The condition acts as a guard that can be used to express more complex application logic. Only if the condition is met, the action is executed. A missing condition is always considered to be true and results in an event-action rule. More powerful event expressions decrease the need for explicit conditions but require more powerful event algebras.
This also makes the event detection mechanism heavier and more difficult for users to use properly. The decision on the trade-off between expressiveness of the event language and the lightweight nature of the event system is domain dependent.

The functionality of event-based systems may be partitioned in different ways. In particular the event composition may be implemented at the monitoring component, in the notification service, or as part of the reactive component. The decision of where to realize event composition and contextualization depends on many application and environment specific factors, such as capabilities of the sensing devices, bandwidth of the communication channels, complexity of the composite events, source of the events that are to be composed, and additional information required for event enrichment and interpretation. (see chapters 4, 5, 9)

An additional dimension is introduced when the source and/or the consumer of events is mobile. Mobility may cause the loss or delay of events and requires special provisions for reliable event delivery and forwarding, staging of events, replay of events and makes the ordering of events more difficult. (see chapters 10, 15, 16, 17, 18, 19)

The reference architecture must be mapped to a specific platform. The platform is dictated by the requirements of the application. The capabilities of the hardware, the communication infrastructure, the source of events, and the application requirements will determine how the reference architecture is mapped to a specific platform. Four families of platforms are emerging for applications that are implemented according to the event processing paradigm:

**Stream processing engines** are typically used for processing very large volumes of homogeneous events that are provided in the form of continuous and high-volume streams. Filters and continuous queries are expressed in a SQL-based language. Scalability and non-blocking behavior are of paramount importance. Stream processing engines exhibit a high degree of centralization and often run on large mainframes or clusters. These platforms are often identified with Complex Event Processing.

**Wireless sensor networks** are typically used in small, well-contained applications in which homogeneous sensors are connected in the form of multi-hop networks to a single sink. These platforms at present do not scale and are typically used for low volumes of events. Processing is done mostly in the network with relatively simple filtering and event aggregation and some composition across heterogeneous sensors. Communication is wireless, low bandwidth, unreliable, and often the limiting factor.

**Messaging systems** are based on reliable and scalable message delivery systems that can connect stationary and/or mobile event publishers and subscribers. Event filtering, composition and routing occur in the broker network and the main goal is to decouple event sources and event consumers. Most information dissemination applications use messaging systems of different kinds as platform.

**Mixed-mode platforms** include a wide variety of nodes, ranging from simple tag readers and sensors to high end servers. Mixed-mode systems are typical for environments in which multiple smaller applications are integrated. Among the applications reviewed in (Hinze, Sachs, Buchmann, 2009), the infrastructure for smart cities, large scale health monitoring and care systems, integrated traffic monitoring and management systems all require mixed mode platforms. The event streams are heterogeneous, both in nature and volume. To scale, the communication must be based on messaging systems, enrichment and event derivation may occur at different nodes in the network, and stream processing engines may be required at selected nodes to detect complex patterns on many event streams. The high volume in mixed mode systems is often the result of many converging low volume streams rather than a few high volume streams. The combination of mobility, heterogeneity, and an extreme distribution compounds the problems these platforms must deal with.
The event based processing paradigm provides the necessary primitives and abstractions to support other architectures. For example, in an SOA, the changes to services are events that are used for notification of the users of these services. In general, event monitoring and processing ideally supports system management, from simple load balancing to control of redundancy and placement of replicas. In the case of self-configuring or adaptive systems, the detection of new components or the disappearance of a component are events that cause system reconfiguration. (see chapters 10, 13, 14)

Quality of service is a major issue in event-based systems. Quality of service (QoS) is a broad term that is often used to mean different things; here we name four aspects relevant to event-based systems. Firstly, quality of service often refers to the guarantees associated with the delivery of events, distinguishing whether events are delivered in order, at least once, at most once, exactly once or only on a best effort base. These delivery guarantees have an impact on the quality of service of the event composition and derivation. Secondly, QoS also refers to dependability, availability and scalability of the event system. A third dimension of QoS refers to performance (throughput) of the event system and the timely delivery of events (real-time behavior). Last but not least QoS often subsumes security. Because of the loose coupling that is inherent to event-based systems, quality of service aspects must be specified and enforced on consumer side based on the expectations of the event consumer. (see chapters 6, 7, 9, 12)

**THIS BOOK**

In its fledgling stages, the event-based research focused almost exclusively on performance of event delivery but in recent years, a number of new challenges have been identified. They are typically introduced by new application areas entering the scene: for example, health-care monitoring introduced security and privacy, traffic monitoring and gaming introduced mobility, avalanche warning sensors introduced memory limitations, e-commerce introduced transactions.

As described above, industry has given event-based systems increasing attention, which is creating and gaining momentum in the commercial world supported by industry-centered summits and working groups. A substantial number of jargon is newly created and used to such an extent that it has become hard to see the wood for the trees. Also the lack of a common knowledge base of real world applications has been repeatedly identified (Rosenblum, 2005) (Hinze, Sachs, Buchmann, 2009, July).

Different to other works recently published, this book does not present a single uniform view but presents a multidisciplinary approach. This book takes a practical approach to the subject, showcasing real-world applications. Including this introduction, the book contains 20 chapters that can be grouped into four parts. They move from the conceptual view to quality-of-service considerations, from novel architectures to mobile and ubiquitous applications of event-based concepts. The chapters bring together concepts and application examples from a wide basis of contributors.

The first four chapters discuss event models and generic system architectures for distributed event-based systems.

- Chapter 1 introduces an architecture for a generic system for performing complex event processing. The authors use a grid approach to achieve scalability and modularity: A system is represented by a grid of complex event processing components amongst which the processing of an event is divided. The fundamental components include event collection, event purification, event storage, event inference, and event situation management. The architecture is presented in two case studies.
on early warning systems in disaster management and on service availability management.

• Chapter 2 compares event models in various distributed event-based systems and in other implicit invocation systems. The event model is identified as a central part of each system as it determines each system’s event concept and how events are generated and notified to consumers. The chapter’s main focus lies in identifying common features between all systems that may need to be supported in the engineering of any event-based application. The chapter proposes a categorization for event models.

• Chapter 3 suggests methods for temporal and spatial correlation between events as needed, for example, in ubiquitous computing. The chapter introduces an event model that uses interval-semantics for event correlation considering spatio-temporal event information. The model supports stateful subscriptions, parameterization, and aggregation. The authors show how controlled event consumption mechanisms (e.g., time restriction, subset policies, and use of durative events) can reduce the event buffer for composite event detection.

• Chapter 4 explores the temporal aspects of event processing. It introduces the notion of an event processing network as the underlying model behind event processing. This event processing network is being executed on a collection of processors that span different time zones. The author identifies and discusses temporal aspects of the execution and within the underlying event model. Temporal aspects that are discussed in the chapter are: temporal dimensions of events, time granularities, temporal context, temporal patterns, event order, and retrospective and proactive operations.

In the second part, five chapters explore detailed quality-of-service aspects of publish-subscribe middleware and event-based messaging systems.

• Chapter 5 addresses the problem of load balancing for event-based messaging. The authors propose the use of a structured p2p overlay network to ensure that the participating peers perform only limited work and store limited state information. The overlay network ICE uses hierarchical tesseral addressing (using regions of space instead of Cartesian coordinates) and an amortized multicast routing algorithm. The suggested load balancing is shown to be independent of the number of peers, popularity of subscriptions, or frequency of messages.

• Chapter 6 examines several security techniques for various types of event-based systems. The chapter first discusses methods for application-level security. Then security measures within a large-scale distributed broker network are introduced, including distributed cryptographic key management. Thirdly, broker-specific policy enforcement mechanisms are evaluated, where context-sensitive event processing allows more than binary access control. The authors identify a number of open issues in security for event-based systems.

• Chapter 7 discusses testing of distributed publish/subscribe systems. In addition to testing separate components, the chapter proposes two techniques for testing the component integration. One is based on large numbers of random event sequences to identify malfunctioning system states. The second one uses predefined data-flows to be tested together with unit-testing. The authors also discuss runtime testing of component integration. The suggested techniques are exemplified in a small case study on testing in Maritime Safety and Security (MSS) systems.

• Chapter 8 gives a detailed overview of the Padres publish/subscribe system. The authors describe Padres’ content-based routing mechanism and discuss techniques to ensure system robustness. The Padres event model supports filtering, aggregation, and correlation of historic and future event
data. The system’s architecture consists of distributed and replicated data repositories that can be configured according to criteria such as availability, storage and query overhead, query delay, load distribution, parallelism, redundancy and locality. The chapter describes Padres’ techniques to ensure robustness, in particular the use of alternate message routing paths, load balancing, and fault resilience techniques in case of broker failures.

• Chapter 9 introduces a publish/subscribe middleware for wireless sensor networks (WSN). The authors address two issues in traditional WSN: lack of interoperability with access networks and lack of flexibility to customize non-functional properties. For example, an event detection application may require in-network event correlation and filtering as its non-functional properties in order to eliminate false positive sensor data in the network. The chapter gives a detailed description of the implementation of the proposed middleware TinyDDS. TinyDDS allows applications to interoperate regardless of programming languages and protocols across the boundary of WSNs and access networks. It also gives WSN applications control over application-level and middleware-level non-functional properties such as event filtering, data aggregation and routing.

The next four chapters in the third part of the book describe novel system architectures that can be realized with event-based communication: knowledge-based networking, processing in parallel clusters, service-oriented architectures and dynamic adaptive systems.

• Chapter 10 discusses Knowledge-Based Networking: the routing of events based on filters applied to some semantics of the event data and associated metadata. The chapter introduces techniques of filtered dissemination of semantically enriched knowledge over a loosely coupled network of distributed heterogeneous agents. Ontological semantics is incorporated into event messages to allow subscribers to define semantic filters. The ontological concepts are encoded as additional message attribute type, onto which subsumption relationships, equivalence, type queries and arbitrary ontological relationships can be applied. The chapter authors explore these ontological operators and their application to a publish/subscribe application.

• Chapter 11 introduces a distributed event processing system for large clusters. Instead of traditional event stream pipes, a content-based publish/subscribe middleware is used for the communication between components. To parallelize processing, the system supports speculative execution of events. The execution of CPU intensive components is spread across multiple cores. Resulting ordering conflicts are detected dynamically and rolled back using a Software Transactional Memory. The authors suggest an extension of the pub/sub mechanism to support load balancing across multiple machines.

• Chapter 12 proposed an event-based approach to Service-oriented Architectures (SOA) and Web services. The services in an SOA are changing regularly, which creates a challenge to loosely coupled service providers and consumers. Using the publish/subscribe style of communication service consumers can be notified when such changes occur. This chapter describes an approach of an event-based Web service runtime environment based on a rich event model and different event visibilities. The authors describe the full service lifecycle, including runtime information concerning service discovery and service invocation, as well as Quality of Service attributes.

• Chapter 13 describes how event-based interaction could be used in Dynamic Adaptive Systems. These systems consist of interacting components from different vendors; they have to be automatically composed at runtime to deal with components joining or leaving the system. Traditional
event-based interaction is lacking explicit directed dependencies that are needed to derive a system configuration. This chapter proposes an approach for realizing Dynamic Adaptive Systems using event-based interaction among the components while maintaining automatic system composition. The system use is demonstrated for a smart city application.

Part four introduces five approaches to mobile and ubiquitous systems. Chapter 15 and 16 propose solutions for mobile ad-hoc networks.

- Chapter 14 argues that event-driven approaches are particularly suitable to address the challenges of mobile and ubiquitous computing, and mobile ad hoc networks. The authors observe that since most programming languages do not provide built-in support for event-driven programming, these concepts are often implemented using different paradigms. This chapter studies the difficulties of combining events with the object-oriented paradigm. Various issues at the software engineering level are highlighted and as a solution, the new object-oriented programming language AmbientTalk is proposed. The language is intended for composing software components across a mobile ad hoc network and supports first-class abstractions to represent event-driven communication.

- Chapter 15 also addresses publish/subscribe communication in mobile ad-hoc networks. In particular, this chapter focuses on ensuring quality-of-service characteristics such as end-to-end timeliness, reliability properties, and limited message overhead. The chapter provides an overview of interconnection topologies, event routing schemes and architectural support of publish/subscribe systems in MANETs. The authors propose the use of event routing in a two-layer scheme using hierarchical tree-based connections in the Manet. They additionally introduce a dynamic system self-reconfiguring ability for event causal dependencies.

- Chapter 16 introduces location-based triggers to support location-aware mobile applications. A location-based trigger is a standing spatial trigger with reference to a spatial region, actions to be taken, and a list of recipients. The author describes the mTrigger framework in three alternative architectures for location triggers: a client-server architecture, a client-centric architecture for management within mobile environments, and a decentralized peer-to-peer architecture for collaboration between mobile users. Performance optimizations are proposed for each of the architectures, including a number of energy-efficient spatial trigger grouping techniques for optimizing both wake-up times and check times of location trigger evaluations.

- Chapter 17 discusses an event-based infrastructure that combines the Session Initiation Protocol (SIP) and Web technologies to realize secure converged mobile services. The chapter uses the scenario of proactive information delivery to airline customers, which has been developed in collaboration with an airline company. The distributed system architecture is outlined and the use of the two key technologies is evaluated for the given scenario. The chapter reports about the authors’ experience in combining the various asynchronous communication mechanisms that are commonly used in today’s Internet in a realistic mobile Internet context.

- Chapter 18 is motivated by the need to provide event-based notifications (ENS) in rescue and emergency applications. The authors focused on sparse Mobile Ad-hoc Networks (MANETs), since emergency applications have to run, for example, in places lacking infrastructure where the density of nodes may be low, or where physical obstacles may limit the transmission range. The chapter analyzes characteristics of rescue operations to discuss ENS design choices such as sub-
scription language, architecture and routing. The authors also present their own Distributed Event Notification Service (DENS), which is specifically tailored for this application domain.

- Chapter 19 is a panel discussion; the chapter tries to capture the opinions of some of the leading researchers from academia and industry about the future of event-based systems. Seven panelists were interviewed; each of them is introduced with a short biography highlighting their professional background from which they approach the field of event-based systems. This allows one to see how different panelists set their individual focus, mirroring their community of origin. The chapter presents their statements and a discussion of the opinions voiced.

REFERENCES


ENDNOTES


2 Examples are Aleri (http://www.aleri.com/), Coral8 (http://www.coral8.com/), Steambase (http://www.streambase.com/), Tibco (http://www.tibco.com/)
Software is openly available for Esper (http://esper.codehaus.org), Padres (http://padres.msrg.utoronto.ca), Siena (http://www.inf.usi.ch/carzaniga/siena/software/index.html)

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

The pushing of events from producers to consumers is a conceptual view. On implementation level, either push, pull, or combinations may be used at different abstraction levels.

The EPTS Event Processing Symposia are now in their 5th instalment: 5th EPTS Event Processing Symposium September 2009, Trento, Italy, and the Gartner Event Processing Summit is a regular annual event since 2007.

Several industry presentations at the Dagstuhl seminar on event processing 2007 focused solely on the comparison of acronyms and concepts (e.g., EDA, ESP, BAM, BPM, SOA, CEP).