Distributed Architectures for Event-Based Systems

Valentin Cristea, Florin Pop, Ciprian Dobre, and Alexandru Costan

Abstract. Event-driven distributed systems have two important characteristics, which differentiate them from other system types: the existence of several software or hardware components that run simultaneously on different inter-networked nodes, and the use of events as the main vehicle to organize component intercommunication. Clearly, both attributes influence event-driven distributed architectures, which are discussed in this chapter. We start with presenting the event-driven *software* architecture, which describes various logical components and their roles in events generation, transmission, processing, and consumption. This is used in early phases of distributed event-driven systems' development as a blueprint for the whole development process including concept, design, implementation, testing, and maintenance. It also grounds important architectural concepts and highlights the challenges faced by event-driven distributed system developers. The core part of the chapter presents several *system* architectures, which capture the physical realization of event-driven distributed systems, more specifically the ways logical components are instantiated and placed on real machines. Important characteristics such as performance, efficient use of resources, fault tolerance, security, and others are strongly determined by the adopted system architecture and the technologies behind it. The most important research results are organized along five themes: complex event processing, Event-Driven Service Oriented Architecture (ED-SOA), Grid architecture, Peer-to-Peer (P2P) architecture, and Agent architecture. For each topic, we present previous work, describe the most recent achievements, highlight their advantages and limitations, and indicate future research trends in event-driven distributed system architectures.

Valentin Cristea · Florin Pop · Ciprian Dobre · Alexandru Costan
University Politehnica of Bucharest, 313 Splaiul Independentei,
060042 Bucharest, Romania
e-mail: `valentin.cristea@cs.pub.ro`, `florin.pop@cs.pub.ro`,
`ciprian.dobre@cs.pub.ro`, `alexandru.costan@cs.pub.ro`

S. Helmer et al.: Reasoning in Event-Based Distributed Systems, SCI 347, pp. 11–45.
springerlink.com © Springer-Verlag Berlin Heidelberg 2011

1 Introduction and Motivation

Many distributed systems use the event-driven approach in support of monitoring and reactive applications. Examples include: supply chain management, transaction cost analysis, baggage management, traffic monitoring, environment monitoring, ambient intelligence and smart homes, threat / intrusion detection, and so forth. In e-commerce applications, the business process can be managed in real time by generated events that inform each business step about the status of previous steps, occurrence of exceptions, and others. For example, events could represent order placements, fall of the inventory below a specific optimal threshold, high value orders, goods leaving the warehouse, goods delivery, and so forth. An event-driven system detects different events generated by business processes and responds in real time by triggering specific actions.

Event-driven solutions satisfy also the requirements of large scale distributed platforms such as Web-based systems, collaborative working and learning systems, sensor networks, pervasive computing systems, Grids, per-to-peer systems, wireless networking systems, mobile information systems, and others, by providing support for fast reaction to system or environment state changes, and offering high quality services with autonomic behavior. For example, large scale wireless sensor networks can be used for environment monitoring in some areas. A large number of different events can be detected (such as heat, pressure, sound, light, and so forth) and are reported to the base stations that forward them to event processors for complex event detection and publication. Appropriate event subscriber processes are then activated to respond to event occurrences. They can set alarms, store the event data, start the computation of statistics, and others.

Event-driven distributed architectures help in solving interoperability, fault tolerance, and scalability problems in these systems. For example, Grid systems are known for their dynamic behavior: users can frequently join and leave their virtual organizations; resources can change their status and become unavailable due to failures or restrictions imposed by the owners. To cope with resource failure situations, a Grid monitoring service can log specific events and trigger appropriate controls that perform adaptive resource reallocation, task re-scheduling, and other similar activities. A large number of different event data are stored in high volume repositories for further processing to obtain information offered to Grid users or administrators. Alternatively, the data collected can be used in automatic processes for predictive resource management, or for optimization of scheduling .

Most important, event-driven distributed architectures simplify the design and development of systems that react faster to environment changes, learn from past experience and dynamically adapt their behavior, are pro-active and autonomous, and support the heterogeneity of large scale distributed systems.

The objective of this chapter is to give the reader an up-to-date overview of modern event-driven distributed architectures. We discuss the main challenges, and present the most recent research approaches and results adopted in distributed system architectures, with emphasis on incorporating intelligent and reasoning techniques that increase the quality of event processing and support higher efficiency of

the applications running on top of distributed platforms. Future research directions in the area of event-driven distributed architectures are highlighted as well.

2 Background

Events are fundamental elements of event-driven systems. An event is an occurrence or happening, which originates inside or outside a system, and is significant for, and consumed by, a system's component. Events are classified by their types and are characterized by the occurrence time, occurrence number, source (or producer), and possible other elements that are included in the event specification. Events can be **primitive**, which are atomic and occur at one point in time, or **composite**, which include several primitive events that occur over a time interval and have a specific pattern. A composite event has an **initiator** (primitive event that starts a composite event) and a **terminator** (primitive event that completes the composite event). The occurrence time can be that of the terminator (point-based semantics) or can be represented as a pair of times, one for the initiator event, and the other for the terminator event [43, 21]. The interval temporal logic [1] is used for deriving the semantics of interval based events when combining them by specific operators in a composite event structure.

Event streams are time-ordered sequences of events, usually append-only (events cannot be removed from a sequence). An event stream may be bounded by a time interval or by another conceptual dimension (content, space, source, certainty) or can be open-ended and unbounded. Event stream processing handles multiple streams, aiming at identifying the meaningful events and deriving relevant information from them. This is achieved by means of detecting complex event patterns, event correlation and abstraction, event hierarchies, and relationships between events such as causality, membership, and timing. So, event stream processing is focused on high speed querying of data in streams of events and applying transformations to the event data. Processing a stream of events in order of their arrival has some advantages: algorithms increase the system throughput since they process the events "on the fly"; more specifically, they process the events in the stream when they occur and send the results immediately to the next computation step. The main applications benefiting from event streams are algorithmic trading in financial services, RFID event processing applications, fraud detection, process monitoring, and location-based services in telecommunications.

Temporal and causal dependencies between events must be captured by specification languages and treated by event processors. The expressivity of the specification should handle different application types with various complexities, being able to capture common use patterns. Moreover, the system should allow complete process specification without imposing any limiting assumptions about the concrete event process architecture, requiring a certain abstraction of the modeling process. The pattern of interesting events may change during execution; hence the event processing should allow and capture these changes through a dynamic behavior. The usability of the specification language should be coupled with an efficient

implementation in terms of runtime performance: near real-time detection and non-intrusiveness [40]. Distributed implementations of event detectors and processors often achieve these goals. We observe that, by distributing the composite event detection, scalability is also achieved by decomposing complex event subscriptions into sub-expressions and detecting them at different nodes in the system [4]. We add to these requirements the fault tolerance constraints imposed on event composition, namely that correct execution in the presence of failures or exceptions should be guaranteed, based on formal semantics. One can notice that not all these requirements can be satisfied simultaneously: while a very expressive composite event service may not result in an efficient or usable system, a very efficient implementation of composite event detectors may lead to systems with low expressiveness. In this chapter, we describe the existing solutions that attempt to balance these trade-offs.

Composite events can be described as hierarchical combinations of events that are associated with the leaves of a tree and are combined by operators (specific to an event algebra) that reside in the other nodes. Another approach is continuous queries, which consists of applying queries to streams of incoming data [17]. A derived event is generated from other events and is frequently enriched with data from other sources. The representation of the event must completely describe the event in order to make this information usable to potential consumers without the need to go back to the source to find other information related to the event.

Event-driven systems include components that produce, transmit, process, and consume events. Events are generated by different sources — event producers, and are propagated to target applications — event consumers. Producers and consumers are loosely coupled by the asynchronous transmission and reception of events. They do not have to know and explicitly refer each other. In addition, the producers do not know if the transmitted events are ever consumed. What the system does is to offer the events to the interested consumers. To do this, other components are used such as the event channel and the event processing engine. To them must be added components for event development, event management, and for the integration of the event-driven system with the application (Figure 1). We next describe briefly the roles of these components.

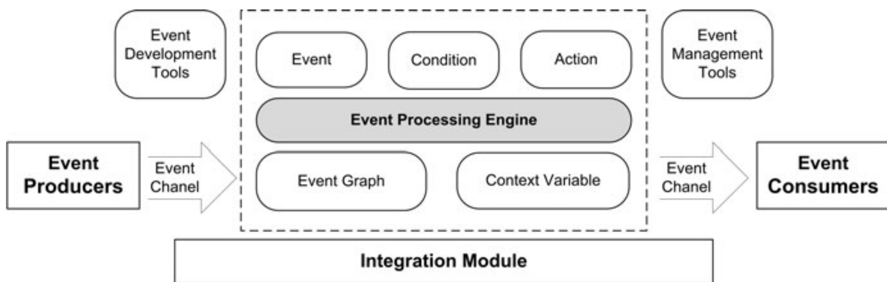

Fig. 1 Components of the event-driven systems

Event producers vary from one application to another. For example, mobile location sensors (GPS) or readers of mobile RFID tags are frequently used in location detection for context aware applications [18]. Web tracking services crawl the Internet to find new postings and generate events accordingly. RFID tag readings combined with information from a company's database can generate events for supply chain management. Mobile sensors can be used for health monitoring systems. Stationary sensors for detecting the presence of persons or sensing the ambient conditions are used in smart home applications. The event producer can also be an application, data store, service, business process, medical equipment, IC card, PC, phone, TV, PDA, notebook, smart phone, and so forth. It is not surprising that various event categories correspond to these sources: temporal, absolute or logical positioning events, change of status events, interval events, event strings, homogeneous or heterogeneous composite events, and others. In addition, events carry contextual information such as social context and group membership [27]. The producer can be a collaboration support service such as Instant Messaging or email. As we can see, there is a large variety of event producers, which generate events in different formats. Consequently, the events need to be converted, prior transmission, to the format accepted by the event channel. In distributed systems, the producer could also include a local event processor that has the role of selecting events for transmission over the event channel based on specific rules, or of detecting composite events. For example, if a temperature sensor generates an event each minute, a local event filter could select and send on the event channel only the events with temperatures greater than a threshold T. Since, in distributed systems, event producers might be spread over a large geographic area, filtering helps to reduce the traffic injected in the transport network that supports the event channels. Also, the producer can generate composite events out of workflows or message streams. For example, the application might want to know when the average temperature for each sequence of ten events "greater than T" events becomes larger than a limit T'. So, an event processor local to the producer calculates the average for each group of ten events and generates a new event when this average is greater than T'. This approach aims to reduce the network traffic and the load of the event processing engine by performing, at the place of the event source, some simple event processing operations.

Event consumers are system components such as software, services, humans, intelligent applications, business processes, performance dashboards, automated agents, actuators, or effectors. They receive events communicated by the event processing engine and react to these events. The reaction might include the generation of new events, so that consumers can be event producers as well.

The **event processing** engine receives events via defined channels, processes them according to specific rules, and initiates actions on behalf of event consumers. Processing can be event aggregation, classification, and abstraction, which are used for generating higher level complex events. For example, event combinations can be a disjunction or a conjunction of two events, a sequence of two events, occurrence of an aperiodic event in a time interval, periodic occurrence of an event, non-occurrence, and temporal [12]. To these operations, event monitoring, tracking, and

discovery can be added. Finally, event handling (for example event rating) can be used for situation detection and prediction purposes.

Engines can process the events individually, one at a time, or in the context of former events. The second approach is known as Complex Event Processing (CEP) and aims at identifying event patterns by processing multiple events. Detecting event patterns can determine the generation of new events or trigger actions of event consumers. Intelligent event processing engines base their decisions on AI techniques and knowledge processing, in which case their behavior adapts continuously to the changing environment they monitor. Engine actions might include event logging, which keeps information about the time at which the event occurred. For composite events, the log preserves the constituent events in the order they were produced. The event processing can be centralized in a single engine or can be distributed to a network of cooperative event processing agents. Both solutions are used in distributed systems and have advantages and disadvantages that will be discussed further in this chapter.

Several models have been proposed for complex event detection. An early approach was extending finite state automata with support for temporal relationships between events [40]. A complex event detection automaton has a finite set of states, including an initial state in which the detection process is started, and *generative* states, each one corresponding to a complex event. Transitions between states are fired by events from an event input alphabet, which are consumed sequentially by the automaton. When the automaton reaches a generative state a complex event is detected. This approach provides efficiency through its direct support for event processing distribution since detectors can subscribe to composite events detected by other automata. Finite automata have several advantages: they are a well-understood model with simple implementations; their restricted expressive power has the benefit of limited, predictable resource usage; regular expression languages have operators that are tailored for pattern detection, which avoids redundancy and incompleteness when defining a new composite event language; complex expressions in a regular language may easily be decomposed for distributed detection.

Many event processing engines are built around the Event, Condition, Action (ECA) paradigm [13], which was firstly used in Data Base Management Systems (DBMS) and was then extended to many other categories of system. These elements are described as a rule that has three parts: the **event** that triggers the rule invocation; the condition that restricts the performance of the action; and the **action** executed as a consequence of the event occurrence. To fit this model, the event processing engine includes components for complex **event detection**, **condition evaluation**, and **rule management**. In this model, event processing means detecting complex events from primitive events that have occurred, evaluating the relevant context in which the events occurred, and triggering some actions if the evaluation result satisfies the specified condition. Event detection uses an event graph, which is a merge of several event trees [15]. Each tree corresponds to the expression that describes a composite event. A leaf node corresponds to a primitive event while intermediate nodes represent composite events. The event detection graph is obtained by merging common sub-graphs. When a primitive event occurs, it is sent to its corresponding

leaf node, which propagates it to its parents. When a composite event is detected, the associated condition is submitted for evaluation. The context, which can have different characteristics (e.g. temporal, spatial, state, and semantic) is preserved in variables and can be used not only for evaluation of the condition but also in the performance of the action.

In distributed systems, **channels** are used for event notification or communication. Events are communicated as messages by using negotiated protocols and languages such as the FIPA ACL (FIPA Agent Communication Language) used for inter-agent communication. Protocols can be push-based, in which producers distribute the events to consumers, or pull-based, in which consumers request the information. In the **publish-subscribe** paradigm [27], producers place notifications into a channel while consumers subscribe for notifications to specific channels. When an event is published, all entities that subscribed to that specific event category are automatically notified. As an alternative, content-based publish-subscribe [47] mechanisms use the body of the event description for routing the information from producers to consumers. Finally, concept-based publish-subscribe [19] uses ontologies and context information to decide on the routing. The publish-subscribe model has at least two advantages: first, it supports event delivery to multiple consumers; second, it decouples the event consumers from the rest of the system, with beneficial effects on scalability and on simplifying systems' design and implementation.

Event development tools are used for the specification of events and rules. **Event management** tools are used for the administration and monitoring of the processing infrastructure and of the event flows. The **integration** module includes interface modules for the invocation and publish-subscribe actions, access to application data, and adapters for event producers.

3 Event-Driven Distributed System Architectures

The concrete realization of a system requires the instantiation of the abstract architecture, placement of components on real machines, protocols to sustain the interactions, and so forth, using specific technologies and products [53]. Such an instance is called **system architecture**.

Event-driven distributed system architectures must respond to users' quality requirements, and to problems raised by the distributed nature of platforms and applications. One challenge is the scale in number of users and resources that are distributed over large geographic areas. They generate a huge number of events that must be efficiently processed. For example, in a transaction cost analysis application, hundreds of thousands of events per second of transaction records must be processed [27]. To support such and similar tasks, the distributed system is required to process composite events with the minimum delay and with minimum resource consumption, which might require a very careful placement of event services on available resources, based on load balancing and replication techniques [58]. Another challenge is presented by fault tolerance, which requires component replication combined with recovery techniques and the use of persistent memory for events.

Yet another challenge is the dynamic nature of the context in which the applications are running. Typical context aware applications are those based on wireless sensor networks or mobile ad-hoc networks for traffic monitoring, which require a greater adaptivity that can be supported by AI and knowledge-based techniques. Last but not least, systems must respond to the needs of large communities of users with different profiles and backgrounds, by offering expressive tools for specifying complex events, and using intelligent techniques for the manipulation of event patterns.

3.1 Complex Events Detection

Distributed event processing is based on decomposing complex event expressions into parts that can be detected separately in a distributed approach. An optimal strategy must be used for the distribution of event processing services between clients and servers or among peers of a distributed system. This is guided by optimization criteria, which can be a tradeoff between a low latency in event processing and low resource consumption. It takes into account the characteristics of systems' infrastructure and of the applications, which in most cases are dynamically changing and ask for flexible and adaptive mechanisms. For example, the policy could encourage the reuse of an existing complex event detector for several other complex events. In other cases, it could be more useful to replicate services to support fault tolerance and reliability or to gain in performance by placing the event processing closer to the consumers.

In many systems, the generation of new inferred events is based on other events and some mechanisms for predefined event pattern specifications. A widespread model that supports the dynamic modification of complex events is the **rule based** paradigm, which currently relies on expressive definition of the relevant events and the update of rules over time. The model uses a set of basic events along with their inter-relationships and event-associated parameters. A mechanism for automating the inference of new events [54] combines partial information provided by users with machine learning techniques, and is aimed at improving the accuracy of event specification and materialization. It consists of two main repetitive stages, namely rule parameter prediction and rule parameter correction. The former is performed by updating the parameters using available expert knowledge regarding the future changes of parameters. The latter stage utilizes expert feedback regarding the event occurrences and the events materialized by the complex events processing framework to tune rule parameters. There are some important directions which are worth exploring, for example casting the learning problem as an optimization one. This can be achieved by attaching a metric to the quality of the learned results and by using generic optimization methods to obtain the best values. For example, transforming rule trees into Bayesian Networks enables the application of learning algorithms based on the last model [56].

Event workflows are a natural choice for the composition of tasks and activities, and are used to orchestrate event interactions in distributed event driven systems. They can be based on associating components into groups or scopes, which induce

an hierarchical organization. Two components are visible to each other if there is a scope that contains them. When a component publishes an event, this is delivered to all visible subscribers. Each component relies on an interface specifying the types of events the component publishes (out-events), and the events the component subscribes to (in-events). Scopes are considered components as well. Each scope has an interface specifying the in- and out-events of the whole group. It regulates the interchange of events with the rest of the system. Exchange of events can be further controlled with selective and imposed interfaces that allow a scope to orchestrate the interactions between components within the scope. In event-driven workflow execution, ECA rules are fundamental for defining and enforcing workflow logic. Fault tolerance is supported by exception event notification mechanisms which, combined with rules, are used in reacting to workflow execution errors. However, further work remains to be done on how to specify and implement semantic recovery mechanisms based on rules. Also, the specific workflow life-cycle needs to be addressed, particularly with respect to long running workflows and organizational changes.

The advent of **stream** processing based on sensor and other data generated on a continuous basis enhances the role of events in critical ways. Currently, event stream technologies converge with classic publish/subscribe approaches for event detection and processing. The key applications of the event stream processing technologies rely on the detection of certain event patterns (usually corresponding to the applications domain). However, efficient evaluation of the pattern queries over streams requires new algorithms and optimizations since the conventional techniques for stream query processing have proved inadequate [9]. We note that applications may cope differently with performance requirements on event streams: while some applications require a strict notion of correctness that is robust relative to the arrival order of events, others are more concerned with high throughput. An illustration of such systems that integrate both event streams and publish/subscribe technologies and support different precise notions of consistency is CEDR, Complex Events Detection and Response [9]. The proposed model introduces a temporal data dimension with clear separation of different notions of time in streaming applications. This goal is supported by a declarative query language able to capture a wide range of event patterns under temporal and value correlation constraints. A set of consistency levels are defined to deal with inherent stream imperfections, like latency or out-of-order of delivery and to meet applications' quality of service demands. While most stream processing solutions rely on the notion of stream tuples seen as points, in CEDR each tuple has a validity interval, which indicates the range of time when the tuple is valid from the event provider's perspective. Hence, a data stream is modeled as a time varying relation with each tuple in the relation being an event. Stream processing faces some particular issues in the context of large-scale events processing. The high volume of streams reaches rates of thousands of detected events per second in large deployments of receptors. Also, extracting events from large windows is a difficult task when relevant events for a query are widely dispersed across the window.

In various event-driven systems, information is combined from different sources to produce events with a higher level of abstraction. When the sources are

heterogeneous the events must be meaningfully enriched, possibly by adding metadata that is often automatically extracted from semi-structured data [10, 26]. **Event enrichment** involves understanding the semantics of the events and of the external sources of information. Depending on the degree of abstract knowledge needed, the event-driven system might generate recommendations automatically, which in response might call for human involvement. Other approaches to addressing heterogeneity of event sources are based on the use of an intermediary ontology-based translation layer. Such systems include an additional layer of mediation that intelligently resolves semantic conflicts based on dynamic context information (history of events or state-context or any other information) and an ontology service [19]. The concept-based publish/subscribe mechanism is part of the event processing layer. If notifications are to be routed in the same context then no additional mediation is needed. If publisher and subscriber use different contexts, an additional mediation step is used. Therefore, concept-based publish/subscribe can be implemented on top of any other publish/subscribe methods (channel-based, content-based, subject-based).

3.2 Classes of Event-Driven Distributed Architectures

There are two large classes of event-driven distributed system architectures: **client-server** and **peer-to-peer**. Systems in the first category are based on the asymmetric relation between clients and servers that run on different machines. A client can actively invoke a service and waits for the response. The server, passively waits for invocations, executes the requested service and sends the results back to the client. Since the server can be client for another server, the possible client-server topologies can be very diverse. The client-server model has several sub-categories. Web-based applications use the browser as client and the Web server as broker for message exchanges with the application server. This sub-model has several advantages: there is no need to build special clients for application servers; new application services can be easily deployed without changing the client; the client's functionality can be enriched by downloading code from the server, and executing it in the browser. Both entities (client and server) can have an event-driven orientation as is illustrated in the sequel. A second sub-model is the event-driven Service Oriented Architecture (event-driven SOA) in which an event, possibly produced by a client's action, can trigger the invocation of one or several services. In turn, service execution can produce new events that are propagated to other services that contribute to solving the client's request. This sub-model is more flexible than the previous one since the client does not have to know the server in advance. Instead, the server publishes its interface, and a lookup service allows clients to retrieve the interface description and formulate service requests according to this description. In addition, the use of the publish-subscribe paradigm allows clients to be notified when new services are made available. Clearly, the use of the same paradigm for events and services simplifies the development of event-driven SOA systems, mainly in enterprise environments. Some authors [39] consider that event-driven SOA is nothing else than another style of SOA, two primary styles being "composite application" and "flow".

The event-driven SOA has proved to be very useful in Grid computing, for enhancing the performance of Grid services in several respects. One is for improving the collaborative activity in Virtual Organizations, in which partner processes can become more reactive to cooperation events. Another one is in monitoring and control of data Grid components engaged in performing complex processing on high volumes of data.

For the second architectural category, we mention multi-agent and peer-to-peer systems. Entities (agents or peers) have equal capabilities and develop symmetric relations. In multi-agent systems, both the event processing and business processing are distributed to agents that can be specialized to different functionalities and interact to perform the specific tasks that correspond to their roles. By definition, **agents** react to events that correspond to environment changes. So, agents are reactive, but they are also proactive, autonomous, adaptive, communicative, and mobile. Consequently, multi-agent systems are attractive for many applications in which these characteristics are important. **Peer-to-peer systems** can also base their collaboration, processing and content distribution activities on the event paradigm. They are very large scale systems capable of self-organizing in the presence of failures and fluctuations in the population of nodes. They have the advantage that ad hoc administration and maintenance are distributed among the users, which reduces the cost of collaboration, communication, and processing.

In the sequel, the discussion is focused on the architectures of intelligent event-driven distributed systems including service-oriented, Web-based, Grid, multi-agent, and P2P systems. It aims at analyzing research issues related to the development of these systems, and to the integration of intelligent and reasoning techniques in distributed platforms. The impact of these techniques on the efficiency, scalability and reliability of stand-alone and business integrated platforms is also presented.

3.3 Event-Driven SOA

The event-driven SOA architecture is an extension of the SOA architecture with event processing capabilities. Services are entities that encapsulate business functionalities, offered via described interfaces that are published, discovered and used by clients [57]. Complex distributed systems are built as collections of loosely coupled, technologically neutral, and location independent services that belong to middleware and application levels. Traditionally, enterprise distributed system components interact by sending invocations to other components and receiving responses. Complex interactions are controlled by orchestration (centralized coordination of services) and choreography (distributed collaboration among services that are aware of the business process). Events introduce a different interaction model in which event channels allow consumers to subscribe for specific events, and receive them when such events are published by producers. This mechanism is adopted in open standards (e.g. CORBA), and in products or platforms (such as .NET, websphere Business Events, Oracle CEP application server, and others) with the aim of simplifying the design of complex interactions and supporting interoperability.

Services can be event producers and consumers, but can also act as re-usable components for event processing, such as Rule service, Decision service, Invocation service, and Notification service (see Figure 2).

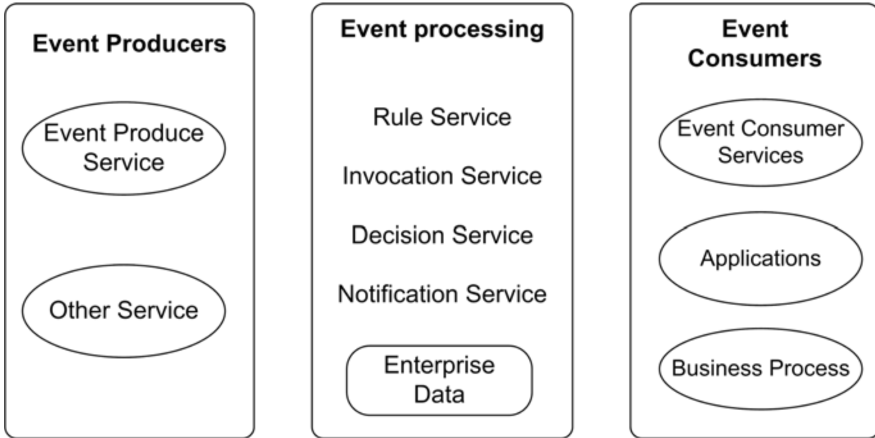

Fig. 2 Services of the event-driven SOA

The Enterprise Service Bus (ESB) architecture defines facilities for business events handling and complex event processing, along with message routing, transport protocol conversion, and message splitting / aggregation to support powerful, flexible, and real-time component interaction. ESB accommodates business rules, policy driven behavior (particularly at the service level), and advanced features such as pattern recognition. ESB has a hierarchical structure determined by horizontal causality of events that are produced and consumed by entities residing in the same architectural layer [23].

The ESB Core Engine is responsible for event processing and uses the Transformation, message Routing, and Exception Management modules depicted in Figure 3. Routing and addressing services provide location transparency by controlling service addressing and naming, and supporting several messaging paradigms (e.g. request / response or publish / subscribe).

The growing needs of the modern business environment have resulted in new standards, products and platforms. New emerging technologies are used in Event-Driven Business Process Management (EDBPM) as an enrichment of BPM with new concepts of Event Driven Architecture (EDA), Software as Service, Business Activity Monitoring, and Complex Event Processing (CEP). New standards for EDA have been defined or are under development by OASIS and OMG:

- Enterprise Collaboration Architecture is a model driven architecture approach for specifying Enterprise Distributed Object Computing systems; it supports event driven systems;

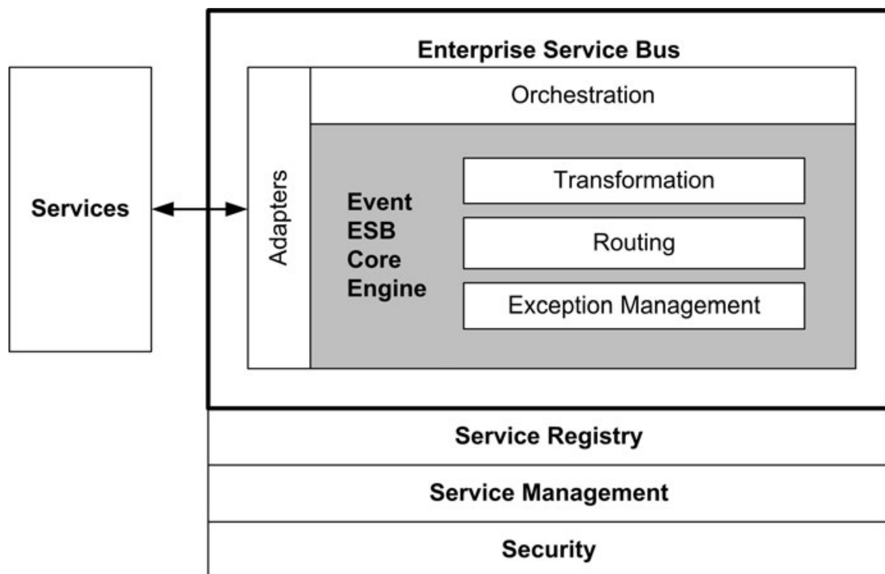

Fig. 3 ESB Components

- Common Alerting Protocol is a data interchange standard for alerting and event notification applications;
- WS-Notification is a family of three specifications (WS-BaseNotification, WS-BrokeredNotification, and WS-Topics) that define a Web services approach to notification using a topic-based publish/subscribe pattern;
- Notification / JMS Interworking refer to event message mapping, event and message filtering, automatic federation between a Notification Service channel concept and topic/queue concepts;
- Production Rules Representation relates to support for specifying Event - Condition - Action rule sets;
- Document Object Model Level 2 and 3 Events Specification refers to the registration of event handlers, describes event flows, and provides basic contextual information for events.

Specific products and platforms have been developed based on these standards. The Oracle Event-Driven Architecture Suite with Oracle Fusion Middleware products allow customers to sense, identify, analyze and respond to business events in real-time. Oracle®EDA is compliant with SOA 2.0, the next-generation of SOA that defines how events and services are linked together to deliver a truly flexible and responsive IT infrastructure [38]. Event-driven workload automation, added in IBM®Tivoli Workload Scheduler 8.4, performs on-demand workload automation and plan-based job scheduling [29]. This defines rules that can trigger on-demand workload automation.

Web services add to SOA their own set of event related standards, WS-Eventing [28] and WS-Addressing [7], targeting the implementation of event driven service-oriented ubiquitous computing systems. WS-Addressing offers endpoint descriptions of service partners for synchronous and asynchronous communication. WS-Eventing defines messaging protocols for supporting the publish/subscribe mechanism between web service applications. Event notifications are delivered via SOAP, and the content of the notifications can be described without restrictions for a specific application.

Much research is directed towards increasing the Web's (and Web applications') reactivity, which means disseminating information about data updates. This can be realized with events that are combined, transmitted, detected, and used by different Web servers. Events could be as simple as posting new discounts for flights that should be notified to interested customers or complex combinations of events that could happen in a more complex Web-based service. One solution to cope with the complexity and scale of the Web environment is the use of event-driven declarative approaches and languages. XChange [44] is a language and an associated runtime environment that supports the detection of complex events on the web and the separation between two data categories, namely *persistent data* (XML or HTML documents) and *volatile data* (event data communicated between XChange programs).

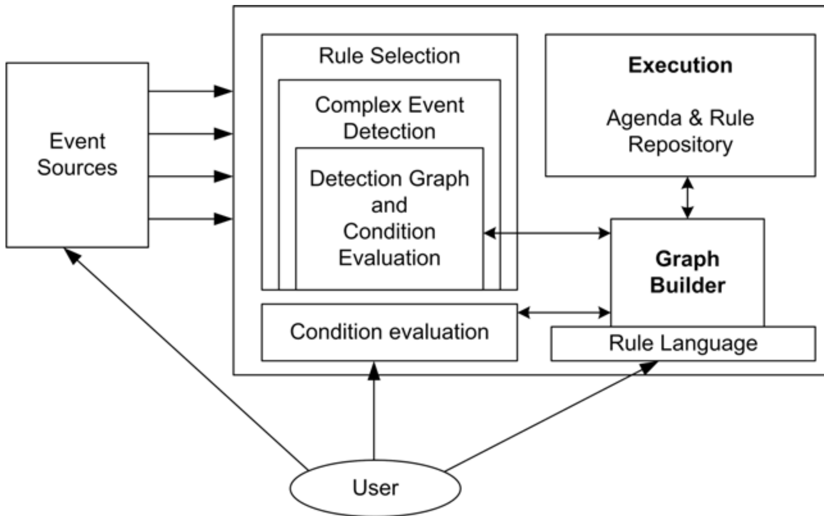

Fig. 4 RIA client architecture

Events can increase the reactivity of **Rich Internet Applications** (RIAs). These are the Web-based counterparts of many applications that are available on desktops. Clearly, to compete with local desktop environments, they have very high QoS requirements concerning the client - server interactivity. To respond to these requirements, RIAs adopt the fat client model, which implements in the browser the user

interface, and uses an asynchronous client-server interaction with reduced waiting times on both client and server.

Adding the capabilities of event processing and declarative rule execution on the client side leads to intelligent RIA (IRIA) that benefits from increased reactivity, greater adaptability to complicated requirements, and higher scalability. The system's architecture (Figure 4) presented in [49] supports the event-condition-action (ECA) paradigm by including a complex event detector, condition evaluator, rule engine, together with adapters for the event sources and the rule language. A rule is implemented as an object in JSON, which contains the triple (event, condition, action). The event expression uses the operators defined in Snoop [14]. The condition part introduces restrictions on the set of composite events that are permitted by the event expression. It uses filters and joins similar to production systems. The action part includes one or more JavaScript code blocks. The system accepts events from local sources (resulting from user-browser interaction, Document Object Model events, and temporal events) and events coming from the network via servers (a stream of stock market events provided by a Comet server, and events resulting from polling RSS feeds). Incoming events are forwarded to the complex event detector, which uses an event detection graph. When a composite event is detected, the associated condition is evaluated. A Rete network [22] is used as the matching algorithm to find patterns in a set of objects contained in the Working memory. If a match for the condition is found, the action is triggered. The action may be the execution of JavaScript code, the triggering of a new event or the modification of the working memory used by the condition evaluator. The system has been tested and has shown that the use of declarative event patterns is able to process continuous event streams and makes RIAs more reactive and adaptive. Future work is needed to formalize the JSON ECA rule language. Also, the efficiency of using the active rules on the client side requires further experimentation in a larger application spectrum.

Event-driven SOA has moved into the sphere of ubiquitous computing. The first step in this direction was the integration of Web services in small devices and wireless network by the definition of a Universal Plug and Play (UPnP) architecture for direct device interconnections in home and office networks [42]. The second step was the addition of event-driven capabilities, which give support for context-based applications by using the sensing services offered by a multitude of ambient device types (sensors, mobile phones, PDAs, medical instruments, and so forth). Clearly, the highly heterogeneous devices handle various sets of data that are carried in the event parameters. The use of WS-Eventing for event notification in embedded services is shown in Figure 5 [30]. ECA rules are expressed in WS-ECA, an XML-based ECA rule description language for web service-enabled devices. Several event types are accepted: time, service, external, and internal. They can be combined in complex events with disjunction, conjunction, sequence, and negation operators. The condition is implemented by an XPath expression. The action part is a conjunction or disjunction of several primitive or composite actions. Primitive actions can be a service invocation, the creation and publishing of an external event or the creation of an internal event and triggering other rules on the same device. WS-ECA

suffers of possible static or dynamic conflicts (several rules triggered by an event may execute conflicting service actions). Some solutions for conflict detection and resolution have been proposed [34].

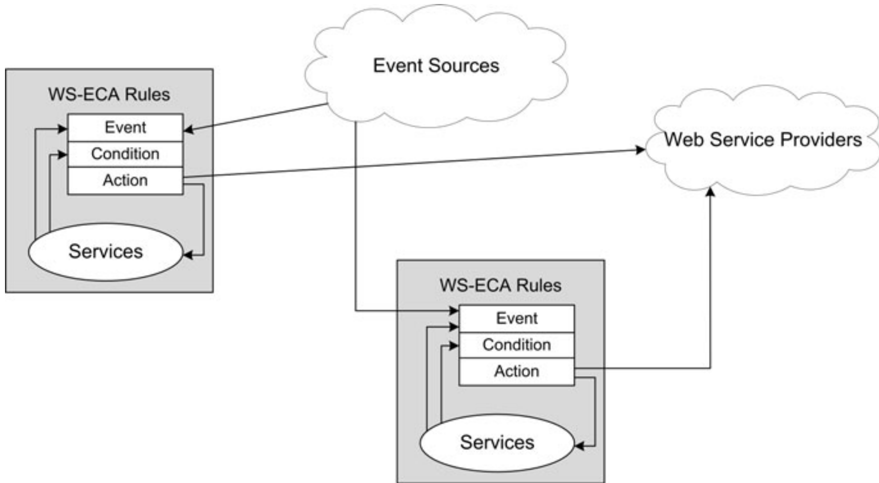

Fig. 5 The use of WS-ECA rules for embedded systems

When event processing is human-centered, the event description technique and the architecture supporting it must be carefully tailored to include context information in a readable form. In the architecture proposed in [31], a statement description, named Five W's and one H, includes context information which indicates: the device that created the statement (Who); the place in which the statement is valid (Where); the time or period in which the statement is valid (When); the name of the data, such as 'temperature' (What); the value domain of the data, for example the set of natural numbers (How); and the identifier of the previous statement on which the current statement causally depends. A primitive event is a sequence of one or more statements in which specific conditions are satisfied.

A composite event expression uses disjunction, conjunction, serialization, and negation operators. The framework architecture (Figure 6) includes the ubiquitous centralized server u-Server that receives reports from, and transmits commands to, several access points, APs, that connect to service nodes. Reports and commands are transmitted as statements. The event detector in the u-Server transmits detected events to a Context analyzer (the context is represented by statements received before the time of the event) which triggers the active rules. The Rule manager generates commands for the control part of the service nodes.

The statement descriptions can be mapped to the event-condition-action (ECA) model and can be integrated with WS-Eventing and WS-ECA event technologies for the implementation of event-driven SOA-based context-aware distributed platforms. While being focused on statement-driven event descriptions, this work opens

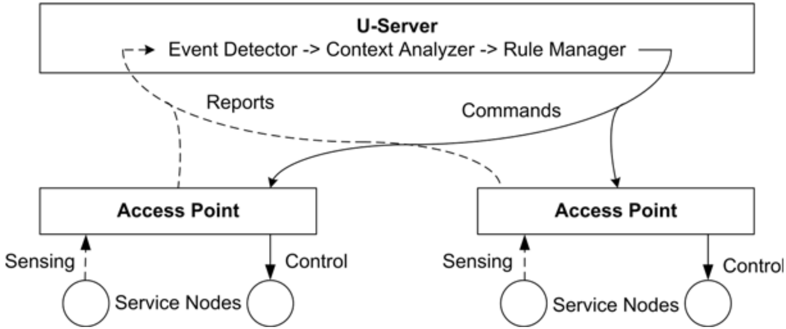

Fig. 6 Event processing framework architecture

new directions for further research towards an easier adaptation of the u-Server to dynamic changes of the context by adopting intelligent approaches in the u-Server functionality.

3.4 *Event-Driven Collaboration*

Events can be used also in Computer Supported Cooperative Work (CSCW) and Collaborative Working Environment (CWE), which have evolved from simple forms of groupware to more recent Virtual Organizations (VOs) used in both scientific and enterprise environments. A VO is composed of people from different organizations, working together on a common project, and sharing resources for this purpose. The collaboration between VO members can be supported by specific tools, which implement collaboration models adapted to the specific features of distributed systems, such as forums, chats, shared whiteboards, negotiation support, group building tools, and so forth. Such collaborations are routine and can take place according to well established **patterns**, which are recurring segments of collaboration that can be captured, encapsulated as distinct components, and used as solutions for further collaboration problems. Examples of such patterns could be "Team organization", "Project plan development", "Collaborative task execution", "Report elaboration", "Final result analysis" and others. Each pattern can be characterized by some triggering event (for example a specific time or the completion of a specific set of tasks), by the use of specific collaboration tools (forums, chats, videoconferences, shared repositories, and so forth), and by the nature and order of activities (one time, repetitive, scheduled, ad-hoc, and so forth). For example, when organizing a project team, the project leader might publish the number and skills of people needed, and then candidates make offers. The leader could interview the candidates, make a selection, notify selected people, and have a meeting with them. During the meeting, the leader could find out that some of the selected people are not available for the entire duration of the project. To replace them, the leader can restart the process from the selection activity.

Some collaborative activities are dynamic and cannot be captured in fixed predefined patterns. Instead, abstract high level patterns can be dynamically adapted to the continuous changes of the context they are used in by services that exploit collaboration knowledge bases and intercommunicate by events [55]. A Recommender service can provide the actions to be executed and the collaboration tools to be used. Awareness services process the events and give information about the collaboration work. Using the monitored events, Analytics services offer statistics about past and present collaborations. The collaboration patterns must be described in terms of the collaboration problems solved, the context they work in, the precondition and post condition of their use, the triggering event, and other relevant features.

The system architecture is presented in Figure 7. The Event and Service Bus links the event sensors to the Complex Event Processing Engine. Simple events are accepted by the Event Reasoner, which detects complex events using information from its Pattern Base. When a pattern is detected, the Event Reasoner notifies the Rule Engine. This engine uses facts about the collaboration, from the Collaboration Knowledge, and retrieves the collaboration pattern whose preconditions and triggers match these facts. Usually, collaboration patterns are combined in workflows and are mixed with user actions. Based on monitoring the collaboration, the system can make changes to the current collaboration pattern or recommend another pattern to replace it.

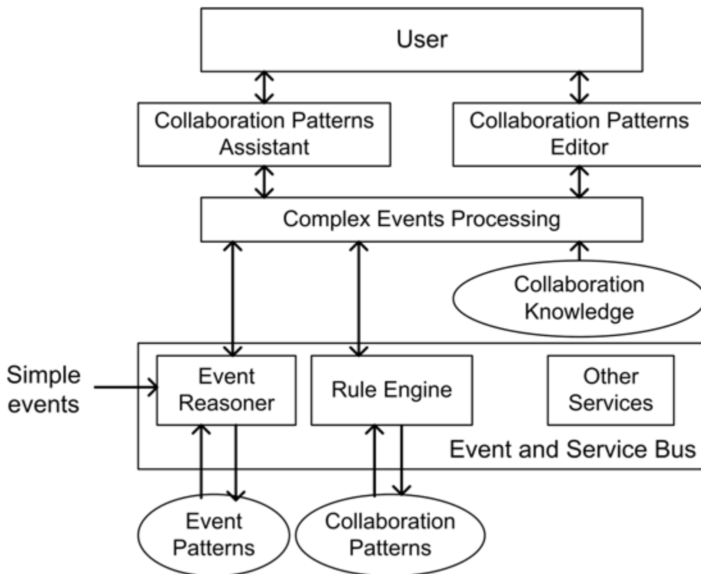

Fig. 7 Collaborative system architecture

The main contribution of this architecture [55] is the use of collaborative patterns in conjunction with knowledge-based event-driven architectures for coping with challenging problems of dynamic Virtual Organizations. Future experiments carried out on platforms that implement the above presented concepts will help to

determine the viability of the approach. Also, the development of an ontology [45] that will provide different levels of abstraction of collaboration patterns could have an impact on pattern integration in collaborative platforms. Another issue is the exploitation of the knowledge accumulated in knowledge bases for synthesizing new high performance collaboration patterns.

3.5 Event-Driven Grids

Event Processing in Policy Oriented Data Grids. Event processing is useful in Data Grids, which allow users to access and process large amounts of diverse data (files, databases, video streams, sensor streams, and so forth) stored in distributed repositories. Data Grids include services and infrastructure made available to user applications for executing different operations such as data discovery, access, transfer, analysis, visualization, transformation, and others. Several specific features influence the Data Grid architecture: users and resources are grouped in Virtual Organizations, large collections of data must be shared by VO members, access to data can be restricted, a unified namespace is used to identify each piece of data, different meanings can be associated with the same data set due to the use of different metadata schemas, and others. In order to support data sharing, protection, and fault tolerance, several services are offered for concurrency, data replication, placement and backup, resource management, scheduling of processing tasks, user authentication and authorization and so on. In addition, data consumers and data providers can specify requirements and constraints on data access and use. The contextual information about data, users, resources, and services is stored in persistent databases and is used in management activities related to the data life cycle.

In the Integrated Rule-Oriented Data System, iRODS [46], a Data Grid complex operation is an event that triggers a sequence of actions and other events. An event has a name and is represented as an extended ECA-style rule:

$$A : -C|M_1,\dots,M_n|R_1,\dots,R_n,$$

in which A is the triggering event, C is the condition for activating the rule, M_i is an action (named a micro-service) or a "sub"-rule, and R_i is a recovery micro-service.

More than one rule can be defined for an event, in which case the rules are tried in a priority order. If the condition evaluation (based on the context) is successful, the sequence of actions is executed atomically. Subsequently, if the execution of an action fails, the recovery micro-services are executed to roll back the effect of the performed actions, and another rule is considered for activation.

These extended ECA rules give more flexibility to the system. Even if the set of rules that applies in a user's session is fixed, different users and groups can use different rules. In addition, users and administrators are permitted to define rules and publish them for use by other users or administrators. The conditions, which are part of the rules, adapt rule execution to the context. The following example, reproduced from [46], explains the role of contextual information for a rule that refers to an ingestion event for uploading a data set into the iRODS data grid:

```
(a) OnIngest :- userGroup == astro
    | findResource, storeFile, regInIcat, replFile
    | nop, removeFile, rollback, unReplicate.
(b) OnIngest :- userGroup == seismic && size > 1GB
    | findTapeResource, storeFile, regInIcat, seisEv1
    | nop, removeFile, rollback.
(c) OnIngest :- userGroup == seismic && size <= 1GB
    | findTinyResource, storeFile, regInIcat, seisEv2
    | nop, removeFile, rollback.
```

The format respects the rule structure previously described and includes the event and condition (on the first line), the action (the second line), and the recovery services (the third line). In the three rules, the context is represented by the "user group" and by the "size" of the data set being processed.

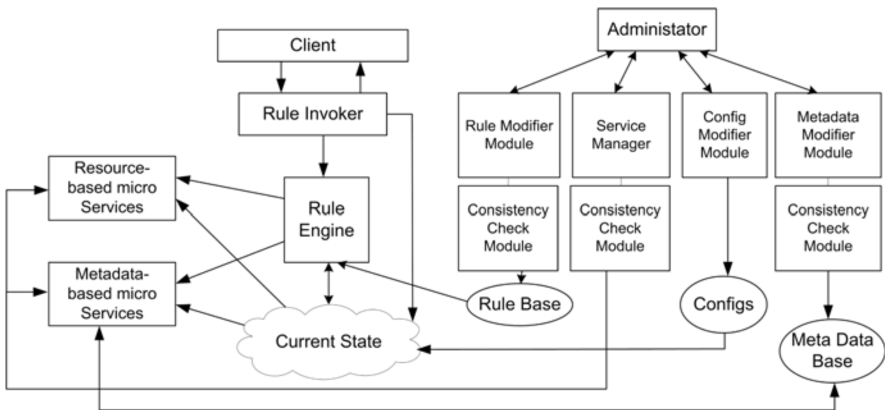

Fig. 8 iRODS architecture

The iRODS architecture is shown in Figure 8. When a user invokes a service, a rule is fired that activates micro-services. To do this, the rule engine uses information from the rule base and Current State. The micro-services can run in parallel or at different times and can intercommunicate by using a Messaging Server. They check the conditions and execute operations on data resources (for example copying a file) or on the MetaData Base. The modifications of the MetaData Base are persistent and can be viewed by other services and by other subsequently executed rules. Micro-services can also intercommunicate by means of a white-board that keeps the local context information. Micro-services can have side-effects outside the iRODS system. For example, the creation of a file can be such a side-effect. Sending an e-mail is another example. The two mentioned operations behave differently with respect to recovery: while a created file can be destroyed, a sent mail cannot be cancelled although a separate mail could be sent to ask the receiver to discard it.

While the intelligent event-driven paradigm can be found in other works related to processing high data volumes [51], iRODS is, to our knowledge, the first attempt to use this paradigm in a Data Grid. The rule-based event processing engine has been successfully integrated with the Data Grid at San Diego Supercomputer Center (SDSC), and it is expected that further experiments will help to improve its performance and adding new features to the platform.

Grid Event Driven Monitoring. The Global Grid Forum elaborated a Grid Monitoring Architecture (GMA) model [5] as a reference to encourage monitoring systems implementations in Grid environments. GMA has several components: a producer, which implements at least one Application Programming Interface (API) for providing events; a consumer that uses an implementation of at least one consumer API; a registry (or lookup service). After discovering each other through the registry, producers and consumers communicate directly. GMA defines several types of interactions between producers and consumers: publish/subscribe, notification, and query/response. It also defines a republisher and a schema repository. The republisher implements producer and consumer interfaces for filtering, aggregating, summarizing, broadcasting, and caching, which correspond to the reactive and processing component of composite event driven models. The schema repository holds the event schema, as a collection of defined event types. A system that supports an extensible event schema must have an interface for dynamic and controlled addition, modification and removal of event types.

A relevant implementation of this model is MonALISA [35], a system able to monitor and control large-scale distributed systems. MonALISA is designed as an ensemble of autonomous self-describing agent-based dynamic services. These services are able to collaborate and cooperate in performing a wide range of distributed event detection, filtering and processing. The system's architecture is based on four layers of services, closely coupled with the GMA model and the abstract model of composite event-driven systems. The first layer is the lookup services network, which provides dynamic registration and discovery for all other services and agents. The second layer represents the event producers. They provide the execution engine that accommodates many monitoring modules, event detectors and a variety of loosely coupled agents that analyze the collected information in real-time. Dynamically loadable agents and filters are able to process the events locally and communicate with other services or agents in order to perform global optimization tasks according to some sets of specified rules. The use of dynamic remote event subscription allows a service to register an interest in a selected set of event types, even in the absence of a notification provider at registration time. Proxy services make up the third layer of the MonALISA framework (Figure 9). They provide intelligent multiplexing of the events requested by the clients or other services and are used for reliable communication among agents. Higher-level services and clients (the event consumers) access the detected events using the proxy layer and thus can obtain real-time or historical data by using a predicate mechanism for requesting or subscribing to selected events and for imposing additional conditions or constraints for interesting events. Once subscribed, consumers receive a stream of relevant events that

are stored and processed. The high level services allow filtering of these events and implement a custom aggregation mechanism to support complex composite events and present global views.

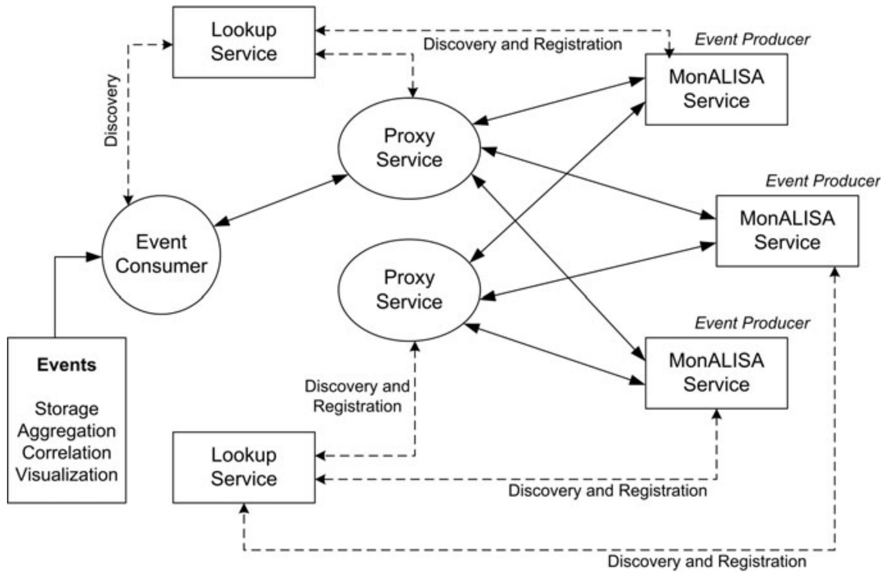

Fig. 9 MonALISA architecture

MonALISA further supports the reactive component of the abstract model, using the detected and processed events to improve the monitored system. The automated management framework implemented within MonALISA represents the first step toward the automation of decisions that can be made based on the monitored events. Actions can be performed at two key points: locally, close to the event producers (in the MonALISA service) where simple actions can be executed; and globally, in a central event consumer (client) where the logic for triggering the actions can be more sophisticated, as it can depend on several flows of events. Hence, the central consumer is equipped with several decision-making agents that help in operating complex systems: restarting remote services when they do not pass functional tests, sending alerts when automatic restart procedures do not fix problems, managing the DNS-based load balancing of the central machines, automatically executing standard applications when CPU resources are idle, and supporting scheduling decisions based on real-time events.

3.6 P2P Systems

Peer-to-peer (P2P) systems consist of interconnected nodes that have similar functions and execute similar tasks. Peers directly share resources such as content, CPU

cycles, storage and bandwidth, without requiring the support of a global centralized server. Instead, they cooperate by means of events that take the form of messages exchanged between peers. P2P systems are capable of adapting to failures and dynamic populations of nodes while maintaining acceptable performance. P2P systems are used to support application services for communication and collaboration, distributed computation, content distribution, and so forth, and middleware services like routing and location, anonymity, and privacy.

While resource sharing is based on direct communication between peers, lookup and locating the peer to communicate with are supported by different mechanisms, which use an overlay network that connects all peers and supports the exchange of events between peers. The overlay is built on top of a physical computer network such as the Internet but has a different topology. Also, the mechanisms depend on the category of P2P network. In unstructured networks, the placement of resources is not related to the overlay topology. By contrast, structured networks map keys that reflect resource characteristics (or content) to node addresses where the resource is located [3].

In unstructured P2P networks, a flooding mechanism is used for event transmission. Each event is transmitted by a peer to all neighboring peers in the overlay. Each receiving peer processes the event (for example, discards the event if it is not interested in it or stores it for further tracking). If the event is addressed to the receiving peer, the event detector processes it, decides on the rule to be executed, and performs the corresponding actions. If the event should be made known to other peers then the current peer forwards it to its neighbors in the overlay network. This approach is used in Gnutella [60] and other similar P2P systems. Routes can be computed by a central event dispatcher (ED). Peers are autonomous computational units that interact with other peers by explicitly producing and consuming events. An event is generated by a peer and sent to the ED, which computes the route that includes all subscriber nodes. The event then traverses this predetermined path. Since the solution is based on a central ED node, is not scalable and does not tolerate faults [20].

Better approaches are offered by structured P2P networks, consisting of transmitting an event only to those neighbors situated on the path towards its subscribers. Traffic reduction is particularly important when events need to be transmitted to a small number of subscriber peers. One solution is to arrange the subscribers into logical clusters such that the event routing is performed by a small number of nodes.

When an event is produced by node A (see Figure 10), it is transmitted to one node in the cluster of subscriber peers (peer B in the figure), which in turn transmits it further to other subscribers (nodes C and D in the figure).

A publish-subscribe architecture based on node clustering (see Figure 11) could include two layers. One is concerned with the management of subscription groups; the other deals with routing events within the network. Subscription groups are formed based on the events' content (the content-based model) or the category they should belong to (the topic-based model). In each node, an Event Handler transmits events, publishes them and notifies subscribers. A Subscription Lookup &

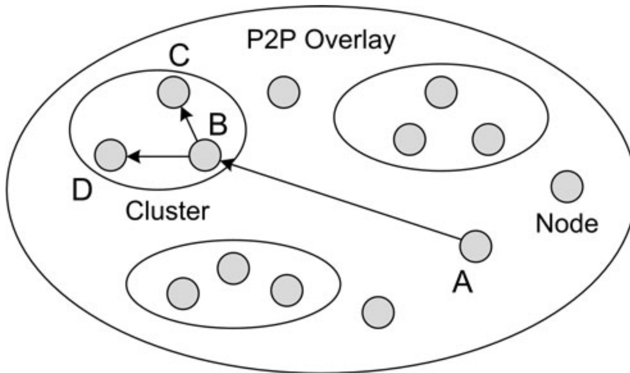

Fig. 10 The topology of the event-driven overlay network systems

Partition Merging component uses a catalogue to map event topics to node addresses for transmitting events. A Subscription Handler performs the node clustering mentioned earlier. The Multicast component connects the event clustering logic to the P2P network overlay underneath.

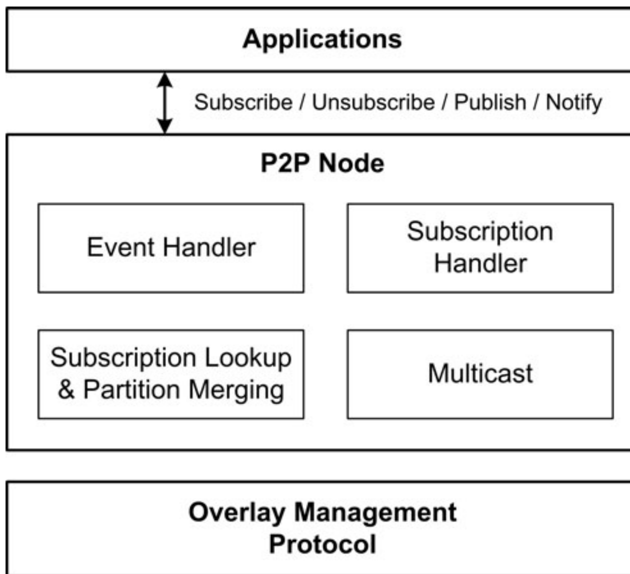

Fig. 11 A general node architecture

Scribe [11] implements this publish/subscribe architecture for managing subscription groups and the multicast communication necessary to send an event to its subscribers. It is constructed on top of Pastry, a P2P location and routing platform

that achieves peer clustering based on the similarity of 128-bit keys used as node identifiers. The routing is performed by always sending an event message to the neighbor identified by a key being numerically closest to the key of the destination subscriber. In Scribe each subscription group has a unique *groupID*. The node with the ID numerically closest to the *groupID* acts as the rendezvous point for the associated group. Each node having an interest in receiving a particular flow of events joins a corresponding group. Each event is routed to the rendezvous node, which further sends it in the form of a multicast message to all members of that group. A similar approach is used in Bayeux [61]. These systems suffer from high cost of maintaining the publish/subscribe groups. Also, each group is accessed through one rendezvous node, which is a communication bottleneck and a single-point-of-failure. TERA [6] avoids this disadvantage by introducing several Access Point Lookup components per group, which are able to receive and route events to the appropriate subscribers.

In mobile environments, producers must be able to send events even to subscribers that are permanently on the move. A notification service has to store all events while subscribers are offline. Once a peer is reachable again, large amounts of data associated with the saved events have to be delivered to it. The peer would have to process them locally in order to extract relevant information [50]. However, in some mobile environments, such as Intelligent Transportation Systems, peers have only a few seconds of connectivity and very limited bandwidth [59]. This may lead to loss of high priority events, such as safety-critical driver warnings, with effects on the system's effectiveness and efficiency. Solutions have been proposed such as using a combination of Distributed Hash Tables (DHTs) with Aspect-oriented Space Containers [33]. The Space Container is a storage and retrieval component for structured, spatial-temporal distributed data. Aspects are components with customizable application logic executed either before or after the operation on the Space Container for event processing. Aspects are executed on the peer where the Space Container is located and can be triggered by operations on the Space Container. A peer (e.g. a vehicle) subscribes for events by deploying a Space Container, installing an Aspect, and publishing it in the DHT network. The Aspect registers itself as a subscriber and, independently of the connectivity mode of the original subscriber, receives events, processes them and stores the results in the Space Container for the use of the original peer.

Most of the available P2P systems are research prototypes, which concentrate on scalability and reliability rather than on durability in P2P environments. Durability refers to the capability to correctly send the events to *all* subscribers, even if nodes or links in the underlying communication layer *fail*. In these P2P systems, some nodes may be involved in subscription groups receiving many events, while others may rarely receive any event. So, load balancing of subscriber groups among all peers in the network is also an issue. The separation of the communication layer from the subscription management layer presents another problem: in these systems events are sent to nodes that are not necessarily interested in receiving them.

3.7 Agent Systems

Software agents react in response to other agents and to environment changes, and can act independently (are autonomic). In addition, agents initiate actions that affect the environment (are pro-active), are flexible (able to learn) and cooperate with other agents in multi-agent systems. Some agents are mobile (can migrate from one place to another). Agent architectures are distributed, robust, and fault tolerant.

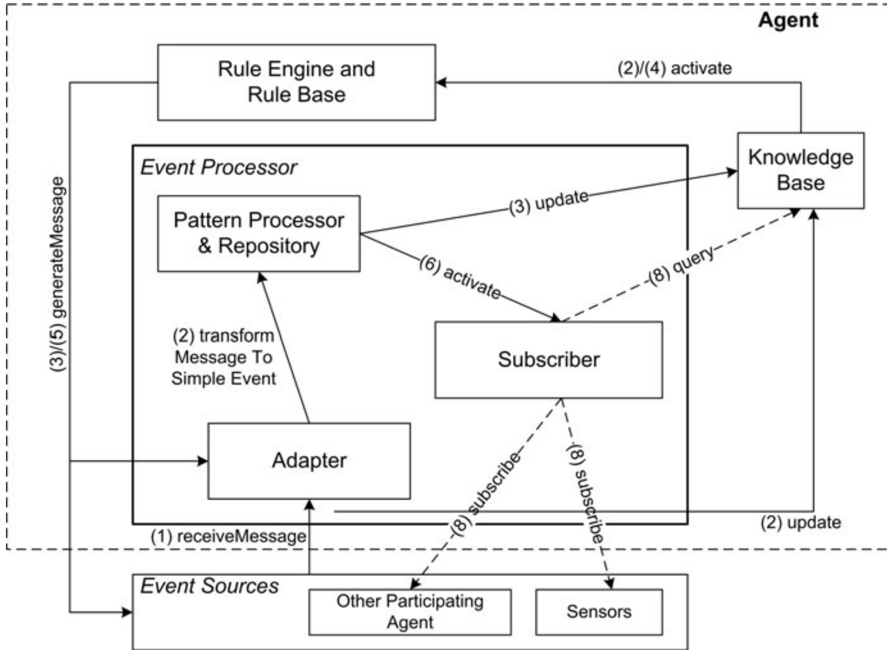

Fig. 12 Event-driven Agent architecture

These characteristics make agents suitable for distributed system middleware, more specifically for parallel execution of tasks, event monitoring, load balancing, trust evaluation, intrusion detection, routing, and other tasks that benefit from the combination of agent distribution and intelligent techniques to obtain optimal solutions.

This section presents relevant architectural features of agents and multi-agent platforms in relation to their role in event-driven distributed systems. The general event-driven agent architecture is presented in Figure 12. The main components are the event processor (EP), a rule base (RB), and a knowledge base (KB). The occurrence of an event is detected by the event processor and is stored in the knowledge base, which then activates a specific rule or set of rules from the rule base. The actions that correspond to the activated business rules are then executed. Complex control functions or computations result from cooperation among agents in the multi-agent system.

In **reactive agent** architectures, the event processor includes modules that are also typically found in other distributed architectures: The **adapter** transforms messages, coming from the environment or other agents, to the format required by the event processor. The **pattern matcher** receives events from the adapter, combines them and detects complex events with the help of a pattern repository. To ease this process, the subscriber inspects the pattern repository, determines the set of interesting possible future events, and subscribes for them. Other features might be present in specific contexts, for example those related to **exception patterns** [16]. An exception pattern E_{ex} for E is obtained by complementing E and eliminating events that cannot happen or are not relevant for the complemented pattern. Thus, E_{ex} represents all the conditions that make E not happen. The agent must include internal policies that are triggered when exceptions are detected. Pattern matching is performed by residuating all event patterns found in the pattern repository with the event that has occurred. For example, for the event pattern $E = a \cdot b \cdot c$ (meaning that event a must be followed by b and then by c), the residual with a is $E/a = b \cdot c$, which is the pattern left to be satisfied after the occurrence of the event a.

Intelligent **pro-active** agents have additional capabilities for interpreting perceptions, drawing inferences, taking decisions, planning actions, and scheduling their execution. They act upon the environment to fulfill the goal for which they were designed. This goal-oriented reasoning allows the agent to commit to the course of action that best accomplishes its task. Intelligent agents have additional capabilities such as supporting contradictory rules, learning, social abilities, natural language processing capabilities and others.

In multi-agent architectures, capabilities are distributed, giving rise to **different agent types**. The authors of [32] present a design in which three primary types of intelligent agents are used: reasoning, learning and evolving agents. Reasoning agents have the ability to make inferences by following a chain of predefined rules, and can be proactive in their behavior. Learning agents are capable of following a set of rules, and also improving their responses by learning from their experience (for example, by dynamically weighting their decisions). Evolving agents improve their behavior with each successive generation. Agents can play other roles too in event-driven architectures [25]. For example, event processing agents [36, 37] act as event detectors, while routing agents are used to construct different types of event channels [24].

Learning is at the base of agent adaptation as response to unexpected events or to dynamic environments. For example, multi-agent systems should adapt to agent failures. They should also support events that occur randomly, events with fluctuating priorities (importance), the inclusion of new information sources and agents, and so forth. The architecture of CyberARIES [48] has an agent part, and a distribution layer. The distribution layer manages the event flows between agents and determines which agents should assist other agents during perception. Agents receive continuous flows of images from one or more camera, filter images and select motion events. They use a motion detection algorithm to acquire a model of the environment. The first step is the construction of a background model from successive frames, using an Auto Regressive filter. In addition, the motion detection algorithm

classifies moving objects either as people (single or in groups), vehicles or unknown objects. The agents act together as a neural network, permanently adapting the classification algorithm based on their previous experience. The final result can be configured dynamically, for example calling a service if a person is noticed or adjusting the movement trajectory based on obstacles noticed. Improvements have been added to this work in [41] who include a surveillance system composed of mobile path-planning robots. The system, called CyberScout, produces a timely interpretation of the environment using feedback from perception processes.

Cooperation abilities (mentioned previously) of agents are important in many other systems, such as those for tracking moving targets. In a sensor-based agent infrastructure for tracking [52], each sensor, which is fixed at a specific physical location, collaborates with neighboring sensors to triangulate their measurements and obtain an accurate estimate of the position and velocity of the mobile targets passing through their coverage area. As more targets appear in the area, the sensors need to decide which ones to track and when to track them, always being aware of the status and usage of sensor resources. Since accurate target tracking requires triangulation, an agent that finds a potential target must contact other agents to ask for their help. AI methods are used to optimize the selection of objects to track and also of the neighbors to help triangulating positions while minimizing resource consumption.

Scalability is an important requirement for interactive intelligent multi-agent systems. In EVA (Evolutionary Virtual Agent) conversational system, a virtual assistant, also called a conversational creature [2], undertakes a dialogue in natural language. EVA has a 3D face with real-time animations and is able to support natural language interactions. EVA's goals are very ambitious: to correctly answer the user's questions in minimal time, avoid some inappropriate questions, achieve tasks that the user seems interested in, and build a user profile. The cognitive part of EVA consists of a natural language processing module, a reasoning module, and a learning module. The reasoning module is a multi-agent system with a pro-active architecture based on a combination of an active layer and a reactive layer. The active layer includes a plan agent (which creates sequences of actions for achieving a goal) and a strategy agent (which adapts the layer's behavior to the environment). To ensure pro-activeness, the active layer reconfigures the priorities of the agents in the reactive layer when specific events occur. The decisions are based on measuring the values of some parameters that characterize the environment. For learning, a classifier system is introduced to express agent behaviors and activate specific procedures accordingly.

Integrating the event-driven component with the business part of the system can be a challenge especially when both parts have high performance requirements. This is the case, for example, in a virtual reality environment for training or computer games, in which AI and Virtual Reality techniques are used to simulate the real world inhabited by autonomous intelligent entities [8]. The system incorporates capabilities for simulating intelligent autonomous entities and, at the same time, for responding to high performance demands of visualizing the virtual world. In a high performance implementation, each agent can have its own computer to run on

independently of the visualization module. This is made possible due to a framework that includes three modules: a FIPA compliant multi-agent platform that acts as middleware, a multi-agent system (MAS) that runs on top of the multi-agent platform, and a visualization module. MAS uses two classes of agents: inhabitant agents simulate beings in the virtual environment and execute actions that change the state of the virtual world; and a simulation controller maintains consistency and synchronization between the inhabitant agents and the virtual world.

The simulation controller has a 3-layer architecture. The simulation layer contains the *world's knowledge base* that maintains the data representing the virtual world state, and the simulator's logic manager module that controls the simulation. The reactive layer contains the sensory responder module that captures events from the environment, performs agents' actions, modifies the *world's knowledge base* and sends the changes to other agents involved; it also has a second component which sends information to several graphic viewers that are connected to the simulation. The social layer supports interaction with other agents.

Inhabitant agents have also a 3-layer architecture. The physical environment layer connects to the virtual world through sensors (that capture events in the virtual world) and effectors (that send actions to the simulation controller). The *cognitive* layer has a memory module (knowledge base), a decision module (with a reactive sub-module for immediate reactions and a deliberative sub-module for better solutions based on the use of the knowledge base), and a perform module with a list of tasks, a scheduler and a dispatcher. The social layer supports interaction with other agents.

Since many results presented previously refer to pilot implementations of event-driven distributed systems, further work is needed to reinforce the results obtained so far. Important research issues include: ensuring platform independence of event-based multi-agent systems, supporting high scalability, resolving issues related to uncertain environments by new facilities for trust estimation, increasing reliability, confidence support, resistance to security attacks, and others. More efforts are needed in understanding the role of agent mobility and self-replication for ensuring multi-agent systems resistance to external attacks. Also, improved collaborative methods leading to better perception of environment changes are needed.

4 Conclusions and Future Work

This chapter focuses on distributed intelligent event processing in Web, Grid, P2P, and agent-based systems. Previous work and research for solving scalability, interoperability, and fault tolerance problems are discussed, with emphasis on those solutions that ensure high reactivity and adaptability to environment changes, proactive and autonomous behavior, learning and social abilities. For each major topic the impact, strengths, weaknesses, and possible improvements are presented.

Adopting event processing in distributed systems is supported by specific models such as the ECA rules paradigm. The capacity to describe the composition and derivation of complex events supports reasoning over event relationships and

distribution of the event detection functionality. In addition, the declarative nature of the rules facilitates adaptation to new and evolving situations.

Event-driven capabilities have been added to distributed systems by extensions to traditional architectures. The extension of SOA has benefited from the publish / subscribe mechanism included in the original SOA model. In addition, the SOA orientation towards open standards has stimulated the development of standards related to events and event services, as well.

The Web has been extended by adding event processing capabilities to servers and clients (browsers) with the aim of making them more responsive to dynamic changes of Web resources (data) and increasing interactivity in user dialogues. Web monitoring services allow users to express their interests and respond by sending alerts or executing other activities. For example, Google Alerts sends users email notifications of events related to their interests.

Event driven capabilities are used in collaborative VOs to help users cooperate for achieving common tasks, or for Grid performance optimization. P2P networks have adopted new models for distributed event transmission and routing, and for event detection that exploit the collaboration of nodes with similar capabilities. Last but not least, multi-agent systems have innate capabilities (pro-activeness, learning, social abilities, and so forth) that make them suitable for perceiving and processing environment events.

Event-based distributed systems are an active research field with many contributors. New ideas have been recently proposed and tested, which show the feasibility of solutions based on the new concepts. Nevertheless, these proposals need further evaluation studies to confirm their validity and performance in more significant, larger scale environments, and for a larger application spectrum. This will require the development of specific evaluation models and metrics for event-based distributed systems. While some results have been reported in the literature [27], more efforts and collaboration with neutral benchmark organizations, like TPC and SPECS, will be needed.

Clearly, testing and evaluation are just two steps of the complex software and system development process for event-driven distributed systems. An important trend will be moving the interest from the development of individual pilot systems to methodologies, software engineering methods, models, and frameworks for the whole software process, which includes requirements specification, design, implementation, deployment, maintenance, policy statement, and system administration. Techniques and methods to develop high performance distributed event detectors and rule-based systems are important subjects for future research. Also, since the design patterns approach has been successfully used in different domains, it is expected that more effort will be directed towards understanding and formalizing the architectural features of the event-driven distributed systems developed so far, and deriving design patterns for different application domains.

Further work on event formalization is also needed, including formal specification and verification of models used in complex event processing. Research and development in several other directions could also be of interest. One is related to information produced by heterogeneous sources. In order to combine them for

deriving meaningful complex events, context information (metadata) might be added to better understand the semantics of events and also of the event sources. Context information is also needed in adaptive pervasive systems in which event and context semantics play an important role for the discovery and composition of services. More work is needed in the development of event, context, and service ontologies. Also, future research will be focused on classifying events and developing discriminant functions for event classes. Developing new methods for exploiting knowledge bases and learning processes could help in improved event detection and replace the human intervention that is used in some systems. Another issue is related to enhancing the event life cycle model with new approaches for event replication, logging, disregarding, consumption, and others to develop a common consistent framework for the operational semantics of event-driven systems [27].

Since wireless and mobile event-driven systems are expected to cover large-area applications, issues related to high variations of connectivity and unreliable data communication will be an important research subject. New policies, event semantics, state synchronization methods on reconnection, late event delivery, security, and others will have to be considered in the design of event-driven systems based on wireless and low capability devices.

Another topic of interest, which goes beyond the borders of event-based systems, will be security such as privacy and protection of producers and consumers. These issues are augmented by the use of profiling techniques for enhancing the performance and precision of event processing engines, and of portable devices used in tracking services. More research will be focused on techniques and methods for ensuring anonymity and for controlling access to sensitive information. Also, more work will be required in finding solutions for reducing the vulnerabilities due to the distribution of system components over large geographic areas, the broadcast communication, and the reduced capabilities of low-end equipment used frequently as event producers or consumers.

References

1. Allen, J., Gerguson, G.: Action and Events in Interval Temporal Logic. Journal of Logic and Computation 4(5), 31–79 (1994)
2. Ameur, R., Heudin, J.-C.: Interactive Intelligent Agent Architecture. In: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IATW 2006), pp. 331–334. IEEE Computer Society, Washington (2006)
3. Androutsellis-Theotokis, S., Spinellis, D.: A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys 36(4), 335–371 (2004)
4. Anicic, D., Fodor, P., Stojanovic, N., Stühmer, R.: Computing complex events in an event-driven and logic-based approach. In: Proceedings of the Third ACM international Conference on Distributed Event-Based Systems (DEBS 2009), Nashville, Tennessee, USA, pp. 1–2 (2009)

5. Aydt, R., Smith, W., Swany, M., Taylor, V., Tierney, B., Wolski, R.: A Grid Monitoring Architecture. GWDPerf-16-3, Global Grid Forum (2001), http://wwwdidc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf (retrieved on February 02, 2010)
6. Baldoni, R., Beraldi, R., Quema, V., Querzoni, L., Tucci Piergiovanni, S.: A Scalable p2p Architecture for Topic-Based Event Dissemination. Technical report, Universita di Roma "La Sapienza" (2007)
7. Bank, D.: Web Services Eventing, W3C Member Submission (2006), http://www.w3.org/Submission/WS-Eventing (retrieved February 26, 2010)
8. Barella, A., Carrascosa, C., Botti, V.: Agent Architectures for Intelligent Virtual Environments. In: 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007), pp. 532–535 (November 2007)
9. Barga, R.S., Goldstein, J., Ali, M., Hong, M.: Consistent Streaming Through Time: A Vision for Event Stream Processing. In: Proc. of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, California, USA, pp. 363–374 (2007)
10. Blanco, R., Wang, J., Alencar, P.: A metamodel for distributed event based systems. In: Proceedings of the Second international Conference on Distributed Event-Based Systems (DEBS 2008), vol. 332, pp. 221–232. ACM, New York (2008)
11. Castro, M., Druschel, P., Kermarrec, A., Rowstron, A.: SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE JSAC 20(8), 1489–1499 (2002)
12. Chakravarthy, S., Adaikkalavan, R.: Provenance and Impact of Complex Event Processing (CEP): A Retrospective View. In: Buchmann, A., Koldehofe, B. (eds.) Special Issue of IT - Complex Event Processing, vol. 51(5), pp. 243–249. Oldenbourg Publications (September 2009)
13. Chakravarthy, S., Adaikkalavan, R.: Ubiquitous Nature of Event-Driven Approaches: A Retrospective View (Position Paper). In: Proceedings of the Dagstuhl Seminar 07191 (2007), http://drops.dagstuhl.de/volltexte/2007/1150/pdf/07191.ChakravarthySharma.Paper.1150.pdf (retreived January 10, 2010)
14. Chakravarthy, S., Mishra, D.: Snoop: An expressive event specification language for active databases. Data Knowledge Engineering 14(1), 1–26 (1994)
15. Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.-K.: Composite Events for Active Databases: Semantics, Contexts and Detection. In: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 606–617. Morgan Kaufmann Publishers Inc., San Francisco (1994)
16. Chakravarty, P., Singh, M.P.: An event-driven approach for agent-based business process enactment. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Article No.: 214, Honolulu, Hawaii, pp. 1261–1263 (May 2007)
17. Chandrasekaran, S., Franklin, M.: Streaming queries over streaming data. In: Proc. of the 28th Int. Conference on Very Large Data Bases (VLDB 2002), pp. 203–214 (2002)
18. Cheng, S., Jih, W., Hsu, J.Y.: Context-aware Policy Matching in Event-driven Architecture. In: AAAI 2005 Workshop: Contexts and Ontologies: Theory, Practice and Applications, Pittsburgh, Pennsylvania, USA, pp. 140–141 (2005)
19. Cilia, M., Antollini, M., Bornovd, C., Buchman, A.: Dealing with heterogeneous data in pub/sub systems: The Concept-Based approach. In: International Workshop on Distributed Event-Based Systems (DEBS 2004), Edinburgh, Scotland (2004), http://www.dvs.tu-darmstadt.de/publications/pdf/Concept-based04.pdf (retrieved 10 January, 2010)

20. Cugola, G., Di Nitto, E., Fuggetta, A.: The jedi event-based infrastructure and its application to the development of the OPSS WFMS. IEEE Trans. Softw. Eng. 27(9), 827–850 (2001)
21. Dasgupta, S., Bhat, S., Lee, Y.: Event Semantics for Service Composition in Pervasive Computing. In: Intelligent Event processing - AAAI Spring Symposium 2009, pp. 27–37. AAAI Press, Menlo Park (2009)
22. Doorenbos, R.B.: Production Matching for Large Learning Systems, PhD Thesis (1995), http://reports-archive.adm.cs.cmu.edu/anon/1995/CMU-CS-95-113.pdf (retrieved March 11, 2010)
23. Ermagan, V., Krüger, I.H., Menarini, M.: Aspect-oriented modeling approach to define routing in enterprise service bus architectures. In: Proceedings of the 2008 International Workshop on Models in Software Engineering (MiSE 2008), Leipzig, Germany, pp. 15–20 (2008)
24. Etzion, O.: Event Cloud. Encyclopedia of Database Systems, 1034–1035 (2009)
25. Fortino, G., Garro, A., Mascillaro, S., Russo, W.: Using event-driven lightweight DSC-based agents for MAS modelling. International Journal on Agent Oriented Software Engineering (IJAOSE) 4(2), 113–140 (2010)
26. Hinze, A., Michel, Y., Schlieder, T.: Approximative filtering of XML documents in a publish/subscribe system. In: 29th Australasian Computer Science Conference, ACSC 2006, pp. 177–185 (2006)
27. Hinze, A., Sachs, K., Buchmann, A.: Event-Based Applications and Enabling Technologies. In: Proc. of the 3rd ACM International Conference on Distributed Event-Based Systems (DEBS 2009), Nashville, TN, USA (2009), Session Keynote papers, Article No.: 1. http://delivery.acm.org/10.1145/1620000/1619260/a1-buchmann.pdf?key1=1619260&key2=9544530821&coll=GUIDE&dl=GUIDE&CFID=98611992&CFTOKEN=93216417 (retrieved January 15, 2010)
28. Huang, Y., Gannon, D.: A Comparative Study of Web Services-based Event Notification Specifications. In: Proceedings of the 2006 international Conference Workshops on Parallel Processing (ICPPW), pp. 7–14. IEEE Computer Society, Washington (2006)
29. IBM. IBM Tivoli Workload Scheduler Version 8.2: New Features and Best Practices. IBM Press (2004)
30. Jung, J., Park, J., Han, S., Lee, K.: An ECA-based framework for decentralized coordination of ubiquitous web services. Inf. Softw. Technol. 49(11-12), 1141–1161 (2007)
31. Jung, J.-Y., Hong, Y.-S., Kim, T.-W., Park, J.: Human-Centered Event Description for Ubiquitous Service Computing. In: Proc. of International Conference on Multimedia and Ubiquitous Engineering, International Conference on Multimedia and Ubiquitous Engineering (MUE 2007), Seoul, Korea, pp. 1153–1157 (2007)
32. Khalifa, Y.M.A., Okoene, E., Al-Mourad, M.B.: Autonomous Intelligent Agent-Based Tracking Systems, Recent Developments. ICGST-ACSE Journal 7(1), 21–31 (May 2007)
33. Kühn, E., Mordinyi, R., Keszthelyi, L., Schreiber, C., Bessler, S., Tomic, S.: Aspect-Oriented Space Containers for Efficient Publish/Subscribe Scenarios in Intelligent Transportation Systems. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2009. LNCS, vol. 5870, pp. 432–448. Springer, Heidelberg (2009)
34. Lee, W.-s., Lee, S.-y., Lee, K.-c.: Conflict Detection and Resolution method in WS-ECA framework. In: Proc. of The 9th International Conference on Advanced Communication Technology, vol. 1, pp. 786–791 (2007)

35. Legrand, I.C., Cirstoiu, C., Grigoras, C., Betev, L., Costan, A.: Monitoring, accounting and automated decision support for the alice experiment based on the MonALISA framework. In: Proceedings of the 2007 Workshop on Grid Monitoring (GMW 2007), Monterey, California, USA, pp. 39–44 (2007)
36. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, May 18. Addison-Wesley Professional, Reading (2002)
37. Luckham, D., Schulte, R. (eds.): Event Processing Glossary - Version 1.1, Event Processing Technical Society (July 2008), http://www.ep-ts.com/ (retrieved January 10, 2010)
38. Memon, A., Xie, Q.: Using Transient/Persistent Errors to Develop Automated Test Oracles for Event-Driven Software. In: Proceedings of the 19th IEEE international Conference on Automated Software Engineering. ASE, pp. 186–195. IEEE Computer Society, Washington (2004)
39. Michelson, B.M.: Event-Driven Architecture Overview. Patricia Seybold Group / Business-Driven ArchitectureSM, February 2, pp. 1–8 (2006), http://soa.omg.org/Uploaded%20Docs/EDA/bda2-2-06cc.pdf (Retrieved January 10, 2010)
40. Mühl, G., Fiege, L., Pietzuch, P.: Distributed Event-Based Systems. Springer, Heidelberg (2006)
41. Oliver, C.S.: Autonomous Mission Planning for a Distributed Surveillance System. Master Thesis: Department of Electrical and Computer Engineering. Carnegie Mellon University, USA (2000)
42. OMA. OMA Web Services Enabler (OWSER): Overview. OMA-AD-OWSER Overview-V1 1-20060328-A (2006), http://www.openmobilealliance.org/releaseprogram/owserv11.html (retrieved March 20, 2010)
43. Paschke, A.: Design Patterns for Complex Event Processing. In: Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS 2008), Rome, Italy (2008), http://arxiv.org/ftp/arxiv/papers/0806/0806.1100.pdf (retrieved Januaruy 15, 2010)
44. Pătrânjan, P.L.: The Language XChange: A Declarative Approach to Reactivity on the Web. PhD thesis. University of Munich, Germany (September 2005)
45. Pattberg, J., Fluegge, M.: Towards an ontology of collaboration patterns. Lecture Notes in Informatics, vol. 120 (2007), pp. 85–96 (2009), http://subs.emis.de/LNI/Proceedings/Proceedings120/gi-proc-120-007.pdf (retrieved February 1, 2010)
46. Rajasekar, A., Moore, R., Wan, M.: Event Processing in Policy Oriented Data Grids. In: Proc. of Intelligent Event Processing AAAI Spring Symposium, Stanford, California, USA, pp. 61–66 (2009)
47. Rosenblum, D., Wolf, A.: A design framework for internet-scale event observation and notification. ACM SIGSOFT Software Engineering Notes 22(6), 344–360 (1997)
48. Saptharishi, M., Bhat, K., Diehl, C., Oliver, C., Savvides, M., Soto, A., Dolan, J., Khosla, P.: Recent Advances in Distributed Collaborative Surveillance. In: Proceedings of SPIE's 14 Annual Conference on Aerospace-Defense Sensing, Simulation and Controls, AeroSense, Orlando, USA, pp. 129–208 (2000)
49. Schmidt, K.-U., Stühmer, R., Stojanovic, L.: Gaining Reactivity for Rich Internet Applications by Introducing Client-side Complex Event Processing and Declarative Rules. In: Proc. of the Intelligent Event Processing - AAAI Spring Symposium, pp. 67–72. Stanford University, USA (2009)

50. Schwiderski-Grosche, S., Moody, K.: The SpaTeC composite event language for spatio-temporal reasoning in mobile systems. In: Proceedings of the Third ACM international Conference on Distributed Event-Based Systems (DEBS 2009), Nashville, Tennessee, USA, pp. 1–12 (2009)
51. Seufert, A., Schiefer, J.: Enhanced Business Intelligence - Supporting Business Processes with Real-Time Business Analytics. In: Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA 2005), pp. 919–925 (2005)
52. Soh, L., Tsatsoulis, C.: Reflective Negotiating Agents for Real-Time Multisensor Target Tracking. International Journal Conference on Artificial Intelligence, 1121–1127 (2001)
53. Tanenbaum, A.S., van Steen, M.: Distributed Systems. Principles and paradigms, 2nd edn. Prentice-Hall, Englewood Cliffs (2007)
54. Turchin, Y., Gal, A., Wasserkrug, S.: Tuning complex event processing rules using the prediction-correction paradigm. In: Proceedings of the Third ACM international Conference on Distributed Event-Based Systems (DEBS 2009), Nashville, Tennessee, USA, pp. 1–12 (2009)
55. Verginadis, Y., Apostolou, D., Papageorgiou, N., Mentzas, G.: Collaboration Patterns in event-driven environments for Virtual Organizations. In: Intelligent Event Processing - AAAI Spring Symposium 2009, Atlanta, US, pp. 92–97 (2009)
56. Vijayakumar, N., Plale, B.: Missing Event Prediction in Sensor Data Streams Using Kalman Filters. In: Ganguly, A.R., Gama, J., Omitaomu, O.A., Gaber, M.M., Vatsavai, R.R. (eds.) Knowledge Discovery From Sensor Data, pp. 149–170. CRC Press, Boca Raton (2009)
57. von Ammon, R., Emmersberger, C., Ertlmaier, T., Etzion, O., Paulus, T., Springer, F.: Existing and future standards for event-driven business process management. In: Gokhale, A., Schmidt, D.C. (eds.) Proceedings of the Third ACM International Conference on Distributed Event-Based Systems 2009, pp. 1–5. ACM, New York (2009)
58. Xhafa, F., Paniagua, C., Barolli, L., Caballé, S.: A Parallel Grid-based Implementation for Real Time Processing of Event Log Data in Collaborative Applications. Int. J. Web and Grid Services, IJWGS 6(2) (2010) (in press)
59. Zaera, M.: Wave-based communication in vehicle to infrastructure real-time safety-related traffic telematics. Master's thesis, Telecommunication Engineering. University of Zaragoza (August 2008)
60. Zhao, S., Stutzbach, D., Rejaie, R.: Characterizing files in the modern Gnutella network: A measurement study. In: Proc. Multi-media Computing and Networking Conf., San Jose, CA, USA, pp. 267–280 (2006)
61. Zhuang, S.Q., Zhao, B.Y., Joseph, A.D., Katz, R.H., Kubiatowicz, J.D.: Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In: Proc. of the 11th International Workshop on. Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001), Danfords on the Sound, Port Jefferson, New York, USA, pp. 11–20 (2001)