Realizing Event-Driven SOA

Olga Levina
SOA and Public Services Research Group
Technische Universität Berlin
10578 Berlin, Germany
Email: olga.levina@sysedv.tu-berlin.de

Vladimir Stantchev, Member IEEE
SOA and Public Services Research Group, TU Berlin
Fachhochschule für Oekonomie und Management (FOM) Berlin
Email: stantchev@computer.org

Abstract

Event-driven architecture is gaining momentum in research and application areas as it promises enhanced responsiveness and asynchronous communication. The combination of event-driven and service-oriented architectural paradigms and web service technologies provide a viable possibility to achieve these promises. This paper outlines an architectural design and accompanying implementation technologies for its realization as a web services-based event-driven SOA.

Keywords: web services, event-driven architecture, service-oriented architecture, business events, business rules

1. Introduction

Physical systems supporting business processes are increasingly coping with the effects of external changes and inputs. This information is used to monitor and control the process flow but it also creates new requirements for underlying network and application system structure. Asynchronous and data-centric communication in a distributed system is an approach followed by designers promoting event-driven and service-oriented architectures. Ubiquity and functional independence are some of the value adding characteristics of Service-Oriented Architecture (SOA). Asynchronous communication, interest-based message delivery using the publish/subscribe principle and event orientation by providing event sensors and event processing components are the characteristics of an Event-Driven Architecture (EDA). This paper motivates the implementation of a holistic architecture: Event-driven service-oriented architecture (ED-SOA) for combining function-and data-centric views on IT systems and enterprise as a whole. The combination of the two approaches is an actively discussed topic among information systems researchers, IT architects and vendors. This paper provides needed definitions and structures to promote common understandings and terms. Furthermore, reference architecture of an ED-SOA is proposed. Web services are suggested as the realization technology. This decision is confronted with the ongoing research and development results for enterprise event-driven systems. The remainder of the paper is organized as follows: in Section 2 we provide the definitions of EDA, SOA and Web Services. Related work on technology for the implementation of enterprise event-driven systems is provided in Section 4. We introduce a reference architecture of an ED-SOA in Section 3 and present a realization approach based on web services and Quality-of-Service (QoS) assurance (Section 5). Discussion of our approach and outlook to future working areas complete the paper.

2. Definitions

2.1. Service-Oriented Architecture

Service-oriented architecture is one of the most discussed topics in the IT these days. Since there is no common SOA definition yet, the term is used as a combination of elements of software architecture and enterprise architecture. It is based on the interaction with autonomous and interoperable services that offer reusable business functionality via standardized interfaces. Services can exist on all layers of an application system (business process, presentation, business logic, data management). They may be composed of services from lower layers, wrap parts of legacy application systems or be implemented from scratch [1]. Service-orientation as a design paradigm roots in several already known approaches such as object-orientation, aspect-oriented programming (AOP), enterprise application integration (EAI) and business process man-
agement (BPM) [2]. Following service-orientation approach a system is decomposed in its functionalities. A service is hence an element that encapsulates a business function and cannot be further decomposed without harming its functionality. Services can be defined as autonomous, platform-independent entities that can be described, published, discovered and assembled [3]; they are technologically neutral, loosely coupled and support location transparency encapsulating business functionality [4]. There are different ways to implement distributed services into IT architecture. They can be implemented using data-based [5], object-oriented (e.g. CORBA and Java RMI) or service-oriented approaches. Since the data-oriented approach applies only to structured data [5] and object-oriented approaches do not necessarily enable loose coupling and ubiquitous services access [6], service implementation is done today mainly using web services. Service orientation and SOA can be used best, when processes or their parts are standardized, are often repeated without changes, or when multiple users need the same process component to complete their tasks. Service invocation (consumption) in an SOA is realized remotely using RPC-like procedure and on request of the service consumer. This approach allows an explicit request for a WSDL-defined service interface to be invoked using SOAP message exchange.

2.2. Event-Driven Architecture

An event-driven architecture is a structure in which elements are triggered by events. An event in the enterprise context is a change in the state of one of the business process elements that influences the process outcome. Being abstract constructions, events are captured as event objects that are a form that allows a machine to process, calculate and manipulate the event. Main components of an EDA are: event sources or generators, event recipients or consumers and event sensor and events processor. Event source(s) and event consumers are connected either directly (point-to-point) or via a middleware or broker (bus). Event source might be an application, business process, internal or external stakeholder or any other abstract data change [7]. Event recipients are all interested subscribers. Event capturing and delivery must be guaranteed by compatibility standards and can be processed in an extra component- the event agent. The logic of collecting and routing of events is captured in the event processor. Incoming event(s) are processed and forwarded to event consumer in (predefined and "soft") real-time. Event consumer reacts to received events by performing its functionality or publishing an alert. There are three types of events that need to be processed: single event, event stream and complex event(s). The difference between an event stream and a complex event can be described as event stream being a temporal sequence of event objects in the "first come-first-serve" manner [8] and complex events being a group of events that contains elements from different contexts or different time points. Processing events means performing operations on event objects like creating, transforming, reading or deleting. Algorithms for processing of multiple or interlaced events are summarized in complex event processing (CEP) technique. It allows identification and extraction of structured information from message-based systems. CEP includes event analysis and correlation delivering a decision triggering information. CEP uses business rules as well as patterns, maps and filters to specify relationship between events [9]. For event monitoring business activity monitoring (BAM) tools are used. These tools are often a part of a business process management suite and are currently more focused on detecting events and visualizing them on a dashboard than on automated decision making, therefore requiring less computational intelligence. Event-driven systems provide real-time visibility of the processes observed and allow almost real-time reaction. In this paper we show that a SOA can provide suitable conceptual structure for an EDA. Contrary to communication in SOA, EDA components interact asynchronously, event processor being a connector with high intelligence. In EDA event sources and event recipients do not know anything about each other, neither does event source know whether and what kind of reaction was caused by its appearance. Figure 1 shows an exemplary EDA architecture.

![Figure 1. Event-Driven Architecture](image)
3. Combining EDA and SOA

Both SOA and EDA have characteristics that complement each other. Both use services but differ in the way a service to be invoked is addressed. SOA provides loosely coupled techniques like web services but its functionality is tightly coupled to the Request/Respond mechanism while EDA provides an asynchronous communication and loose coupling [10].

While SOA offers EDA a suitable design approach by providing a distributed environment for separating business logic, processes and technical functions, it benefits from another service invocation technique that loosens the rigor of the RPC-style calls. When observing these characteristics the merged structure of the three concepts provides, one can realize multiple synergy aspects. Service-orientation allows to capture and store events as services. Integration of legacy systems into service-oriented architecture may be done using the derived business rules the systems are using, or by using event-driven architecture. SOA is based on a remote access principle allowing a distributed environment, necessary for both event-driven architecture and business rules. EDA has a decoupled, asynchronous structure that complements loose coupling and synchronous communication of SOA [11]. Implementing SOA-suitable environment means implementing an environment where events can operate on their best and many architectural interactions are already standardized. Further synergies come up with communication and process management in a distributed system, which can be assured by adopting a business rules oriented ED-SOA. Often having a highly distributed architecture, enterprises create benefits from the real-time information availability. EDA provides a structure that allows a fast reorganization of business processes without affecting application or technical structures. Fast reaction to environmental changes in is possible without the need to adapt technical infrastructure. Functional decomposition on a high-granularity level, that is crucial for robustness to change of a system, is provided by SOA. Merging these concepts results in an enterprise architecture that is more flexible while being robust to changes. Its components are loosely coupled and can be accessed in any business situation.

The major aim of enterprise architecture is realized in the ED-SOA concept by SOA combining business functions and IT, and EDA focusing on data as well as business relevant event orientation; both SOA and EDA concepts can be used for application and legacy systems integration [12]. Covering the aim and component spectrum of enterprise architecture as described above, ED-SOA can be regarded as its evolution. Figure 2 shows a proposed ED-SOA reference architecture including security aspects, business rules processing and business data integration. Components that can be encapsulated as services are named. They were identified according to the main principles of service-orientation: their granularity can be easily identified and discovered while being reusable by different components in different points of time. The concrete integration infrastructure into the application systems landscape depends on the technology used to realize ED-SOA. Here an enterprise service bus (ESB) is a suitable solution as the architecture is to be realized using web services.

4. Related Work

Distributed event processing and event-driven systems became popular in recent years as the technology needed to provide and support these systems is rapidly evolving. In the 1980s and 1990s message-oriented middleware was used to facilitate integration of various application systems within an enterprise. Basic event-processing can be regularized by inclusion of Java Message Service and message-driven beans in Java Enterprise Edition (J2EE) [13]. Message-oriented middleware allows a push-based, publish-subscribe data-centric communication through message brokers or queued messages. As for the embedded, real-time systems based on event-orientation, they are often written in languages such as C or C++, with real-time services provided by CORBA (Common Object Request Broker Architecture) [13], [14]. CORBA also provides a publish-subscribe mechanism by the CORBA/IIOP (Internet Inter-ORB Protocol) [15]. Such
solutions typically require direct involvement of IT and computer professionals for their implementation and deployment. Service-oriented approaches have on the other side a higher potential for individual and cost-effective integration of an event-driven architecture.

Web services are currently the most promising service-oriented technology [16]. They use the Internet as the communication medium and open Internet-based standards, including the Simple Object Access Protocol (SOAP) for transmitting data, the Web Services Description Language (WSDL) for defining services, and the Business Process Execution Language for Web Services (BPEL4WS) for orchestrating services.

5. Enabling ED-SOA

This section provides an overview of implementation technologies that we used in our proof-of-concept and is structured according to the elements presented in Figure 2.

Software components that call (consume) services can be developed in a variety of languages on a variety of platforms. Typical integrated development environments (IDEs) allow this interaction without the need to code SOAP messages. They generate a proxy stub object on the local machine that marshals calls to the actual web service. Therefore, from a software engineering point of view a single service interaction is not much different from the interaction of COM (Component Object Model) or CORBA components. Important new aspect of web services is the promise of automatic composition going beyond the binary integration of COM and CORBA. Such flexible processing infrastructure can adapt more easily to changes in the functional requirements of an event-driven business process.

5.1. Platforms

The complexity involved in providing a single web service is often underestimated. A look at hardware platforms, even commodity hardware, reveals complex microprocessors and processing architecture. Standard OSs are far away from microkernel designs and contain a large number of OS extensions. These are called modules in a Linux system and drivers in a Windows system. Beside typical device drivers, extensions include network protocol implementations, file systems and virus detectors. Typical component frameworks such as .NET and J2EE often serve as the middleware for providing web services [17]. Therefore, we selected the .NET Framework as platform.

5.2. Quality of Service and Nonfunctional Properties

The nonfunctional properties (NFPs) of a software system are those properties that do not describe or influence the principal task / functionality of the software, but are expected and can be observed by end users in its runtime behavior [18].

QoS encompasses important NFPs such as performance metrics (for example, response time), security attributes, transactional integrity, reliability, scalability, and availability. Traditionally, QoS is a metric that quantifies the degree to which applications, systems, networks, and other IT infrastructure support availability of services at a required performance level [3]. Web services environments are based on flexible composition of services and therefore demand greater availability of applications. Furthermore, they introduce increased complexity in terms of delivering, accessing and managing services.

The existing standards for specification of QoS characteristics in a service-oriented environment can be grouped according to their main focus: software design/process description (e.g. UML Profile for QoS and QML - QoS Modeling Language [19], service/component description (e.g. WS-Policy) and SLA-centric approaches (e.g. WSLA - Web Service Level Agreements [20], WSOL - Web Service Offerings Language [21], SLAng - Service Level Agreement definition language [22] and WS-Agreement [23].

Extensive research concerning NFPs also exists in the field of CORBA.

5.3. Implementing Rule and Decision Services

Our sample implementation uses the .NET Framework as a serviceware and the Microsoft Workflow Foundation (included in .NET 3.0) as basis for the rule and decision services. The workflow foundation supports different types of workflows (see Figure 3) and facilitates particularly the implementation of rule-based activities. Using it, we can map rules defined at the business level to any .NET programming language in a straightforward way.

5.4. Implementing Invocation and Notification Services

Any step in our workflow (as implemented in the Microsoft Workflow Foundation) can call operations on other objects on the same machine, invoke other workflows or directly invoke web services. Events that
trigger a state change (next step) of a workflow range from sensor information (e.g., RFID) through changes in data sources (e.g., relational databases) to web service outputs or fault messages. There are several integration depths that we regard as relevant for events:

- Events at the data level – here we differentiate between events originating from database management systems (DBMS), e.g., relational databases, and events originating from sensors, e.g., RFID readers and scanners.
- Events at the object level – these are typically state changes in class instances which we regard only if they are manifested by public methods.
- Events at the service level – a call, or a response of a service.

Generally, we can map the "lower level" events (data and object) to the service level using web service wrappers. Furthermore, we can combine events to complex events (e.g., a delivery has arrived and a warehouse is full) by using composite services. In the context of this composition we particularly regard the NFPs of the composed service, as described in [18].

5.5. Integration Aspects

An already agreed-upon SOA strategy greatly facilitates our approach as we can then expect that critical software functionality will be provided as web services in the specified timeframe. If our approach has to be integrated in more heterogeneous environments we can benefit from the capabilities of .NET 3.0 to interact with diverse remote components, such as other .NET objects, SQL servers and web technologies.

5.6. Application Scenario

One application scenario that can greatly benefit from ED-SOA is logistics. Our demo application in this domain (more particularly contract logistics) differentiates between several states of a shipment that is being transported (see Figure 4). It begins with an initial event (Container sent) and goes through the following statuses: Fetched, Accepted, Loaded, Unloaded, in Delivery, and Delivered. Business users can define rules related to these statuses and corresponding events (e.g., a longer delay or a missed deadline) using a web-based user interface. We then use these rule specifications in our implementation to trigger next (or additional) steps in the workflow according to incoming events. Events can be propagated in a variety of ways: RFID-based communication in a warehouse system, e-mail notifications, changes in inventory databases, as well as other components or web service calls and responses. This makes our approach highly flexible to changing business requirements – they can be submitted to our system as a new rule set via the user interface.

6. Discussion and Outlook

In this paper we introduced the concept of an event-driven service-oriented architecture (ED-SOA) and its reference structure. Furthermore, we provided technology and element definitions and outlined possible advantages of combining service-oriented and event-driven approaches for which we proposed a reference architecture. Our application scenario in contract logistics used web services and the .NET Framework as enabling technologies and demonstrated major benefits of the approach. Our future work lies in the areas of incorporation of pre-defined rule sets for specific domains (e.g., environmental conservation, privacy and security, healthcare applications) in the approach. We
are also working on the incorporation of various high-assurance techniques [24], [25] into the approach.

References


