Fiorano
Enabling change at the speed of thought

www.fiorano.com

AMERICA'S
Fiorano Software, Inc.
718 University Avenue Suite 212, Los Gatos,
CA 95032 USA
Tel: +1 408 354 3210
Fax: +1 408 354 0846
Toll-Free: +1 800 663 3621
Email: info@fiorano.com

EMEA
Fiorano Software Ltd.
3000 Hillswood Drive Hillswood Business Park Chertsey Surrey
KT16 0RS UK
Tel: +44 (0) 1932 895005
Fax: +44 (0) 1932 325413
Email: info_uk@fiorano.com

APAC
Fiorano Software Pte. Ltd.
Level 42, Suntec Tower Three 8 Temasek Boulevard 038988
Singapore
Tel: +65 68292234
Fax: +65 68292235
Email: info_asiapac@fiorano.com

SCA unifying SOA and EDA

Distributed nature of complex business applications

--- Atul Saini

SCA UNIFYING SOA AND EDA

Distributed nature of complex business applications

Executive Summary

The concepts of SOA (Service-Oriented Architecture) and, to a lesser extent, EDA (Event-Driven Architecture) have evolved significantly as approaches to the development of a new generation of flexible, componentized business systems that can evolve rapidly in response to rapidly changing business conditions. Unfortunately, both SOA and EDA have more to do with the distributed nature of complex business applications and less with the modularity of the solutions themselves. While managing distributed modules of code is important in its own right, it is not possible to build agile, extensible business systems without primary focus on the componentization of the solution – whether or not it is distributed.

Service Component Architecture (SCA) refers to the development of business systems as a collection of reusable service components that interact either via request/reply (SOA) or via Event-Driven (EDA) interactions. SCA combines services and events to delivery business systems that drive reuse, ease of management, dynamic extensibility and configuration management and easier business process management.

Service Oriented Architecture

SOA refers to the design of applications via components (often referred to as *"services"*) that expose interfaces that can be called by other client applications. Multiple components are chained together via request/reply calls to create a larger *"composite application"* which can then be reused as a logical module within a larger business process. Figure 1 illustrates a typical SOA implementation.

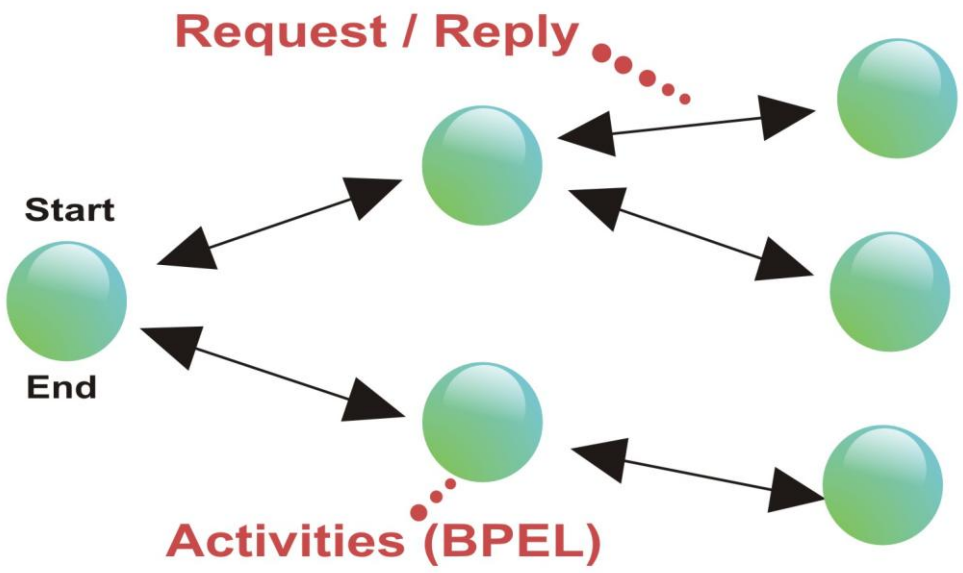

Figure 1: Service Oriented Architecture

Unfortunately, the primary focus of SOA has been the concept of accessing functions in remote components to create a distributed application based on request/reply semantics. SOA infrastructure typically does not mandate any particular component-model that guides developers to create software modules based on a coarser level of granularity that matches higher-level business functions rather than lower-level technical functions. In a typical SOA, the focus is more on the distributed nature of the interaction of the software modules than on the semantics and design of the modules themselves. While many SOA implementations exhibit efficient request/reply interactions between code-modules, the

modules themselves are poorly designed - being either *"too small"* or *"too large"* and often following technical rather than business-level semantics. The lack of focus on the creation of reusable, modular service components, often results in SOA systems that are almost as difficult to develop, deploy, modify and change as their previous legacy counterparts.

Event Driven Architecture

Event-Driven Architecture refers to the design of applications as a collection of components that exchange events to perform business-functions. The major difference between SOA and EDA is that in an SOA, all intermediate service components suspend their operation until the relevant request/reply call returns, while in an EDA all service components continue to operate since their focus is on processing incoming messages and publishing outgoing messages; EDA is thus typically more efficient than an SOA approach due to pipelined, concurrent processing of events by multiple software modules chained together (since there is no waiting for blocked calls to return). Figure 2 illustrates an EDA interactions.

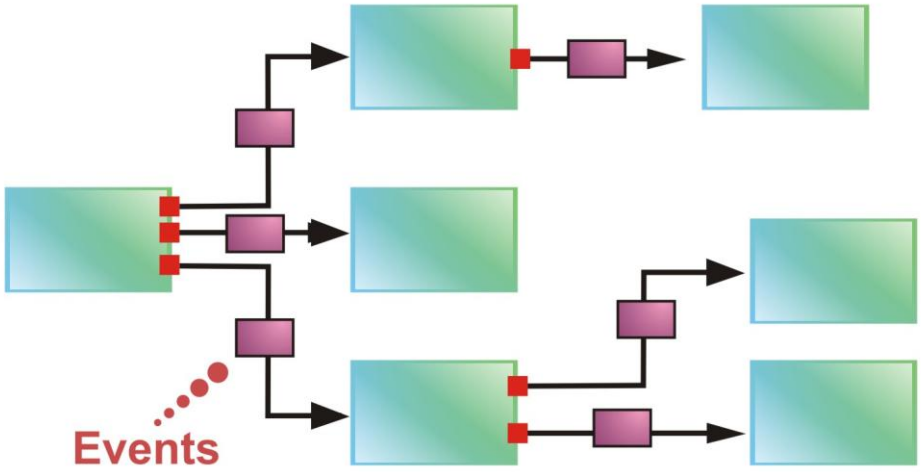

Figure 2: Event-Driven Architecture

Unfortunately, current EDA approaches suffer from the same problem as SOA since the focus is more on the event-exchanges between distributed software components rather than on the modularity and granularity of the components participating in the EDA process. Poor semantic design of the EDA modules result in applications that are difficult to maintain, debug and extend.

SCA unifying SOA and EDA

SCA is an architectural approach in which application developers decompose problems into smaller modules, each of which executes a well-defined business function and is implemented as an encapsulated component. The interactions between Service Components may be either request/reply (SOA) or via events (EDA). Service Component Architecture thus moves the focus of application design from the concept of distributed computing towards the intelligent design of modular service components. A single SCA application may involve multiple request/reply calls as well as multiple event-exchanges as illustrated in Figure 3.

SCA logically unifies SOA and EDA into a single framework, since the distributed nature of the interaction between service components in an application is now overshadowed by the notion of software modularity. Finding the right level of granularity at which to implement a service component now becomes more important than the request/reply or event-driven exchanges of information between the components themselves.

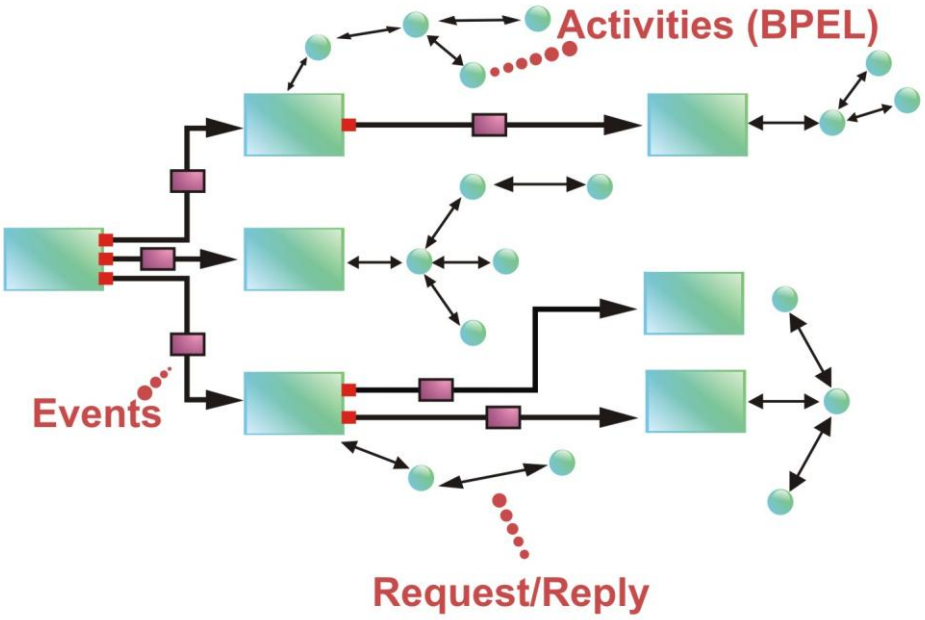

Figure 3: Service Component Architecture: Unifying SOA and EDA

Service Components: Building blocks for SOA and EDA

Service components are software modules that follow the semantics of business-functions, as distinct from technical components that typically implement a single technical function. For instance, *"update customer address"* is a service component since it has a meaning relevant in a business context, while *"update database table"* is a technical component since it implements a technical function that has no direct business relevance. Service Component Architecture (SCA) is the process of developing applications as a collection of service components that exchange information via request/reply (SOA) or events (EDA). SCA applications can be modified, managed and changed with little or no programmer intervention, making the SCA approach significantly more attractive for the deployment of business systems in comparison with traditional monolithic application design.

Characteristics of SCA

Service components have two important characteristics: **Encapsulation** and **Modularity**.

A) **Encapsulation** - a well-known feature from the days of object-oriented programming in the late 70s - refers to the separation of the interface and implementation of a software component. The external interfaces of a service component expose the relevant functions offered by the component, while the internal implementation details are hidden from unauthorized external access. Service components can be *"client-based"*, in which case they have service-interfaces that can be called by external programs to invoke functions, or event-driven, in which case they have event-descriptors that have to be matched at runtime for an event to be consumed at an input port. Since the only access to a service component is via its external interfaces, the internal implementation of the component can be changed without affecting any applications using the component. As such, it is possible to replace an existing service component in a SCA application with a new component that shares the same interfaces but has a different internal implementation. The separation of interface from implementation allows service components to be designed, developed and tested by independent teams of developers in different geographical locations provided that the interface contracts between the components remain fixed. Encapsulation also supports versioning, configuration management, dynamic deployment and a host of other useful features for the development of modern distributed business applications.

B) Modularity refers to the process of decomposing a problem into a set of smaller problems. Service components are modular in the sense that each service component implements a relevant business function (which can be reused across multiple business solutions). The internal implementation of single Service Component module typically consists of a series of steps or *"activities"* which are chained together to implement the information flow required by the component.

Benefits of SCA

In contrast to traditional monolithic applications that are designed as a single whole, SCA applications consist of a coalition of Service Components that communicate either via events (EDA) or via request/reply calls (SOA). SCA offers several key advantages over the traditional approach of monolithic application design:

- **Flexibility of Development:** Service Components are easier to develop because the semantics of each independent service component are significantly less complex than the overall of a single, (relatively large) monolithic application; each Service Component can be developed by a different team of developers, each of whom focus only on their component without having to know the details of work done by others.
- **Reuse:** Since each Service Component has well-defined interfaces, each component can be developed, tested and debugged independent of the other components. This not only speeds up project implementations but, in the case of well-designed service components, also leads to significantly enhanced reuse.
- **Dynamic Deployment and Runtime Modification/Replacement:** Service components can be dynamically deployed to remote nodes at runtime, and components within a process can be easily replaced by new or updated components, further reducing the time taken to modify or change an existing process in response to business requirements.
- **Configuration Management and Version Control:** Service components facilitate version control and dynamic configuration management, allowing fine-grained control over deployments across the enterprise.

Summary

While SOA and EDA have become popular as approaches for building distributed systems, neither approach can lead to truly extensible business systems without first focusing on the design of modular, encapsulated service components. Service components can interact via any combination of synchronous request/reply or asynchronous event-exchange. Service Component Architecture thus unifies services and events to create a single framework based on modular service components leading to applications that can be modified, managed and changed with little or no programmer intervention to create the next generation of agile business systems.

About Fiorano Software

Fiorano Software (www.fiorano.com) is a leading provider of enterprise class business process integration and messaging infrastructure technology. Fiorano's network-centric solutions set a new paradigm in ROI, performance, interoperability and scalability. Global leaders including Fortune 500 companies such as Boeing, British Telecom, Credit Agricole Titres, Lockheed Martin, NASA, POSCO, Qwest Communications, Schlumberger and Vodafone among others have used Fiorano technology to deploy their enterprise nervous systems.