# EDSOA: An Event-Driven Service-Oriented Architecture Model For Enterprise Applications

Karina Hauser, Utah State University, USA
Helgi S. Sigurdsson, INFOR, USA
Katherine M. Chudoba, Utah State University, USA

**ABSTRACT**

*Enterprise Applications are difficult to implement and maintain because they require a monolith of code to incorporate required business processes. Service-oriented architecture is one solution, but challenges of dependency and software complexity remain. We propose Event-Driven Service-Oriented Architecture, which combines the benefits of component-based software development, event-driven architecture, and SOA.*

**Keywords:** Service Oriented Architecture (SOA), Event-Driven Architecture (EDA), enterprise application, design and development methodologies

## INTRODUCTION

As companies discover the importance and value of information for strategic decision-making, isolated, departmental, and non-integrated IT solutions are replaced by integrated enterprise applications (EAs) (Mitchell, 1996). EAs ensure that the same information is available throughout the whole company, between companies, and ultimately, throughout the entire supply chain for fast and reliable decision-making (El Sawy, Malhotra, Gosain, & Young, 1999). At the same time, researchers and practitioners alike have recognized the need for a flexible information technology infrastructure to gain and sustain a competitive advantage in today's fast-paced global marketplace (Brancheau, Janz, & Wetherbe, 1996; Byrd & Turner, 2001; Davenport & Linder, 1994).

Traditional EAs are constructed as a monolith of code that incorporates all the required business processes, making them difficult to implement, maintain, or extend (e.g., add new functionality). An additional challenge is that mergers and acquisitions create a diverse IT environment and the need to integrate different applications. IT organizations are embracing service-oriented architecture (SOA) to solve these problems associated with EA (Elfatatry, 2007; Marks, 2007). While there is much debate on an exact definition for SOA, a working definition is "*the architectural style that supports loosely coupled services to enable business flexibility in an interoperable, technology-agnostic manner"*(Knippel, 2005). A recent survey by the Yankee Group of Boston found that 75% of 473 enterprise buyers expect to invest in the infrastructure necessary to enable a service-oriented architecture (Mimoso, 2004), but there are few detailed reports on implementation.

Exceptions include Dietrich (2007) who describes a SOA-approach for mass customization and illustrates the approach as a case study example from the shoe industry. Another example is Vinci, a local-area SOA designed for rapid development and management of Web applications (Agrawal, Bayardo Jr., Gruhl, & Papadimitriou, 2002). A user-centric SOA, which allows users to publish their needs and let producers (software developers) provide services to meet the requirements, is described by Chang (2006). MIDAS (Kart, Shen, & Gerede, 2006) combines SOA and Web Services to provide a loosely-coupled distributed environment that enables all participants of the supply chain to manage their information over the Internet. However, all of these examples describe traditional SOA implementations, which have several problems that are described in the next section.

**CHALLENGES OF TRADITIONAL SOA**

The most widely accepted implementation of SOA uses standardized Web Services as illustrated in Figure 1.

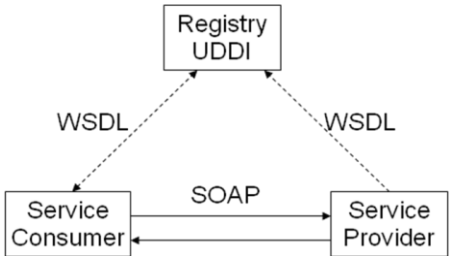


**Figure 1: Web Services**

The W3C (Booth et al., 2004) defines Web service as "..a software system designed to support interoperable machine-to-machine interaction over a network. …". The W3C has published standards on how the machine-to-machine interaction should be conducted. A service provider defines and describes its services using Web Services Description Language (WDSL) and then publishes them with the use of Universal Description, Discovery and Integration (UDDI) protocol in an information registry. The service consumer uses the UDDI registry to find and access information about available services. If the service consumer decides to use a service of the service provider, they exchange data via Simple Object Access Protocol (SOAP). The use of these web standards is by no means a prerequisite for the design and deployment of SOA, but they have enabled its widespread use. Once the use of service orientation became an accepted methodology in the software industry, enterprise architects and developers started to leverage this new technology to build enterprise applications.

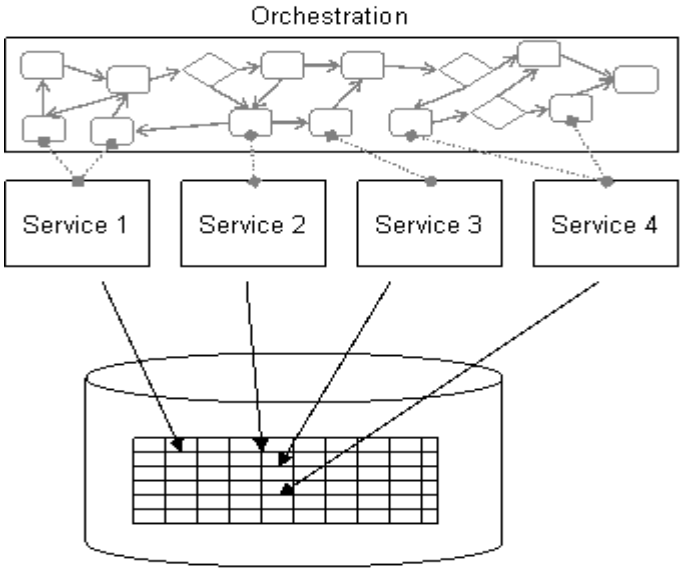


**Figure 2: Orchestration of Services**

The SOA and Web Service approach does, however, introduce another level of complexity; i.e., the services have to be complete, readily available and cannot be changed once in use. These issues have given rise to a new type of software category - Services Lifecycle Management (SLM). SLM software provides functionality for an enterprise to publish service definitions to a central repository, version them, and control their availability and lifecycle. Services can be tagged as under development, in production, depreciated, etc. This also puts a lot of requirements on the performance of the service.

The real power of the services model appears when orchestration is introduced, as shown in Figure 2. Orchestration allows a configuration to define where the services are, how and when they are consumed, other services that are executed, and how data are combined. The orchestration engine calls a service and passes the context to it. Once the service has been completed, the context is handed back to the orchestration engine that executes the process. This pattern has been called "command and control". The orchestration engine is commanding services to execute and control the overarching flow.

Building applications by using orchestrations to combine services provides several benefits. The use of higher level languages allows developers to combine existing services into a process. The use of transformation tools, state engines and workflow further enhances the application flow by allowing developers to leverage those to enhance functionality, re-use existing logic and provide a more configurable application. The use of these tools has moved the application from being data-centric table editors to applications that drive business processes in a organization. Languages like (BPEL) BPEL 2.0 and Business Process Execution Language for Web Services (BPEL4WS) offer some exciting possibilities where business analysts and developers can construct entirely new processes out of existing services.

While using orchestrations to call very discrete services over the Internet or to orchestrate trading partner relationships among disconnected trading partners has been very successful, its use in EA is far more problematic. Performance, control, life span, access control and several other issues need to be solved. EAs traditionally use a single database backing all services, and granularity of the services has usually been kept at an object/method level and the span of use has been unconstrained; hence, calling any service from any flow is allowed.

The use of SOA in building EAs was supposed to solve the problems of dependency and issues with software complexity (Marks, 2007). The very nature of a service forced developers to keep code better organized. With access to functionality readily available, integration should be a much simpler task, and applications could deliver substantial benefits by leveraging process management tools and the internal application code. Services, particularly Web Services, work very well in the distributed environment of the Internet where the amount of data to be transferred is relatively small and users are willing to wait a second or two for a response, and where the services have a very clear function and purpose (Elfatatry, 2007). Using services in EAs is much more complex and introduces challenges for developers.

**Challenge 1: Dependencies of Services**

In EAs, data are usually stored in a single database, which leads to conflicts if different services try to update the same table. This results in multiple dependencies and connections between services. For example, two services that could modify or ask questions about a customer create a dependency to each other through the customer data. Questions arise about who can change what data in what sequence, what service has priority and how to manage conflicting information. Change in the schema of the customer table or the lifecycle of a customer record might change the functionality of a service.

**Challenge 2: Response Time**

In EAs, the amount of data that need to be transferred is usually very high. A clerk who needs to enter hundreds of transactions is not willing to wait 10 or 20 seconds between transactions. To keep the response time of any services request to a minimum, systems have to be highly scalable, which leads to increased expense in hardware and the addition of technology to do performance monitoring and management. An EA that was optimized on database performance and the movement of data in and out of the database now has a whole new services layer embedded. Performance profiling becomes much more difficult.

**Challenge 3: Linking of Implemetation and Services**

The most common problem with SOA is linking internal application code (the implementation) of the services to the message payload of the service (Marks, 2007). SOAP, one of the most popular services protocols, is based on a request respond pattern to a back-end object and is used to give access to remote objects. The direct connection between the application objects and methods to services means the shape of the object has become as rigid as the database schemas. Once the service is in use, it cannot simply be refactored and changed.

The granularity of back-end object methods is another complexity of this tight binding. Most back-end object designs deal with data at a finer level than services require or optimally want. Create order header, create order line, etc. is the norm of the OO design, but building services at this granularity suggests the need for distributed transaction control.

**Challenge 4: Rigidity of Services**

As SOA increases flexibility and extensibility of EAs, it also increases complexity by adding an additional layer. Once several applications rely on a service to behave in a specific way, changing that behaviour can have widespread ramifications, making changes or enhancements to the service nearly impossible. Making service activities more adaptive is an increasingly important objective in SOA development (Papazoglou & Van Den Heuvel, 2006) .

All of these challenges need to be addressed to drive the next level of flexibility and functionality into EA development. The development of our event-driven SOA model was guided by the following objectives:

- Decouple processes even further, generating a true services environment where the services have minimal interaction with each other.
- Build a new application code so that it can augment current applications.
- Provide a more scalable architecture that can provide services at every level, at the performance EA users have come to expect.
- Allow versioning of services to be independent from each other.
- Build applications that enable provisioning of services and functionality independent from geographic location or deployment model, and support on-site, hosted and a "Software as a Service" model.

**EVENT-DRIVEN SOA**

The proposed event-driven SOA (EDSOA) model, shown in Figure 3, consists of five parts: Components (including service and user interface), a messaging BUS and event orchestration engine, a composite user interface, single sign-on, and security features.

**Components**

The components are the heart of the model. Component-based software development provides a variety of advantages, it manages complexity more effectively, decreases development time, improves software quality, and increases reusability (Brown, 2000; Vitharana, 2003); but it also has some challenges (Brereton and Budgen, 2000; Stafford and Wallnau, 2001), especially in the area of reusability. Szyperski (1998) defines a software component as a unit of composition with a contractually specified interface and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. In component-based software development, business processes are split into a number of domains, such as customer or supplier management. Within each major domain, there are a number of components. Those components, along with other parts of the system, then complete a business process. The components are built to be as independent from each other as possible. Each component has an "event" or completes a workflow that has a result. For example, the result of the order management component is an order event. This result is communicated to the rest of the EA using the Messaging BUS and Event Orchestration Engine.

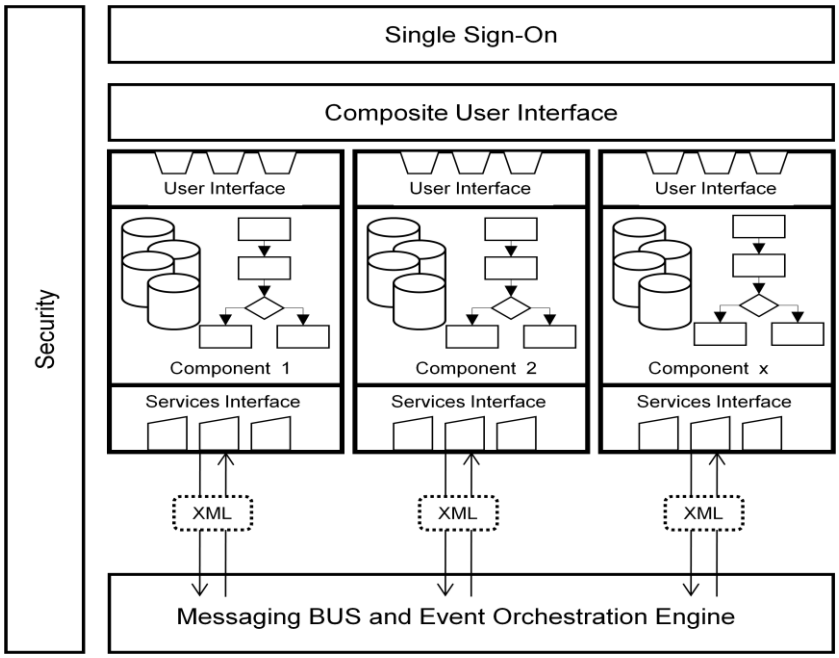


**Figure 3: The Five Parts of the EDSOA**

Each component has the following characteristics:

- Publishes business events in asynchronous fashion.
- Subscribes to business events in asynchronous fashion.
- Has service interfaces for the business services it provides.
- Has local data store for information needed to execute logic.
- Is a system of record, or authoritative data source, for information.
- Is deployable in a "stand-alone" mode.
- Can have a proxy component that is deployed within other components.
- Has a user interface that can be integrated in the composite user interface.

Each component consists of three building blocks, as shown in Figure. 4: a User Interface, the actual application, and a service interface.

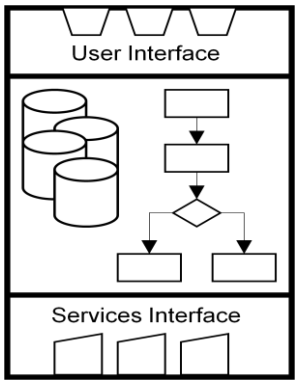


**Figure 4: Basic Building Blocks of a Component**

When building the software components, each component is a set of codes that has the following properties:

- It has its own database and process. It receives business events that it is interested in, stores the data locally, and executes its own process.
- It is the minimal set of code that can be deployed by itself.
- It is capable of running for a limited amount of time without having a connection to other components.
- It implements its own services and interfaces that can be used to communicate with it.
- It is the System Of Record (SOR); i.e., authorative data source, for specific information within the enterprise, possibly more than one.
- It provides services for the information that it is a SOR for.
- It sends out business events when a process goes through states.
- It can consume one or more services.

**Special Component: Master Data Management**

In a component-based system that uses business events to communicate between the components, integrity of master data is extremely important. In general, there are two major approaches to managing master data.

1. Distribute the data elements to the components that need them. Each component will then send out a message whenever its data changes. This will be the most loosely coupled system, but multiple components could maintain the same or similar data.
2. Using Master Data Management, a single component is responsible for the storage and management of the master data. Once the data changes, a message is sent to the rest of the components that need the changed information. While this approach creates a little more dependency, it reduces the number of times a specific type of data has to be maintained.

Regardless of which approach is used, information, such as item, location, codes, currency, calendar and party, need to be managed in a cohesive way.

**Special Component: Component Proxy**

The overall purpose of component proxies is to reduce latency and ensure availability of a service. One of the challenges with EDSOA is the focus on asynchronous messaging. Complex processes, like pricing services, need to be highly scalable. For example, having one Web Service (one component) answer all pricing requests might seem like an obvious choice, but the consumer of the service has to be considered. Pricing requests can come from different entities – web-stores, brick and mortar stores, call centers or some other channel. In high speed order entry, the optimization of pricing is paramount, and users of a web-store will not be satisfied if the pricing and discounts of products take seconds (or minutes) to calculate. In EDSOA, the proposed solution is through the component proxy design pattern, which is incorporated in the component and illustrated in Figure 5.

In a component proxy, business rules are maintained in a single component, such as a pricing component, which sends out messages containing the rules that provide local answers when executed by local engines (component proxy). Multiple identical component proxies enable disconnected evaluation where an engine provides results based on cached rules, resulting in high performance (scale-out) and 24/7 availability, even if the pricing component is not available because of connectivity problems or scheduled maintenance. The pricing component can be "off-line" while the local engines execute the rules as they are published. In this case, the engines are accessed locally by different applications, using either an Application Programming Interface (API) or traditional SOA.

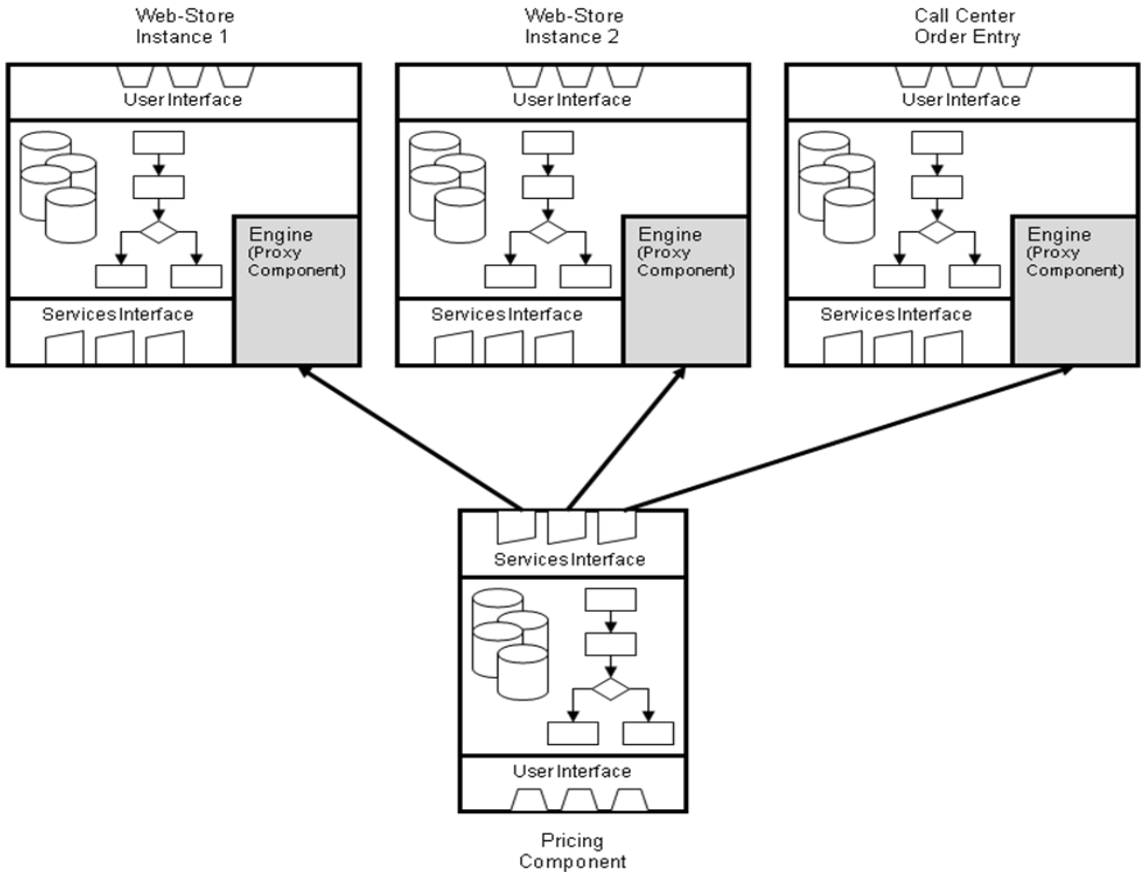


**Figure 5: Example of a Pricing Component incorporated as Proxy Components into a Variety of Order Components**

**Messaging BUS and Event Orchestration Engine**

Each component is part of a system that subscribes and publishes information. To ensure performance and simplicity, EDSOA uses asynchronous messaging to deliver and consume the messages. Each message is related to the business event that a component creates. As the messages (events) are sent using asynchronous methods, it is important to assure delivery of those messages to all components that subscribe to it. Assured delivery can be implemented over HTTP by using standards like Web Services (WS)-ReliableMessaging, Message Queues (MQ) or Java Messaging Services (JMS) message queuing systems. When implemented correctly, these can all work together in harmony, delivering data to components using any type of protocol.

With multiple different components subscribing to the same message, the distribution of the messages is handled by an Event Orchestration Engine (EOE). The EOE is a router that contains the business event ontology; i.e., it contains and manages all publish/subscribe rules, with the actual flow of information coded into the rules. These routing rules are normally very simple. Component 1 sends out a message based on an event. This message is then routed to every subscribing component. The EOE manages the copying and forwarding of the message to all subscribers. The messages that are transmitted could be in any format – binary, flat, EDI or XML. A good design choice for these are standards like the Open Application Group (OAGi) XML standards.

Each component is a System Of Record (SOR) for a particular business object or content and sends out a Sync message. For example, the Order Management component sends out SyncSalesOrder that is then subscribed to by multiple other components, like the shipping and invoicing components. By using a canonical format for these messages, the effort to transform and get a content understanding is reduced significantly.

**Composite User Interface Technology**

The objective of composite user interface (CUI) technology in EDSOA is to provide one single point of access for all information for every user. The access is role-based. Users are grouped into roles (e.g. manager or sales clerk) and authorization for specific information is given by role. Each component has rules associated with it about how best to display the information in the UI. An example of a CUI is shown in Figure 6. Each component is visible and accessible, regardless of its source.

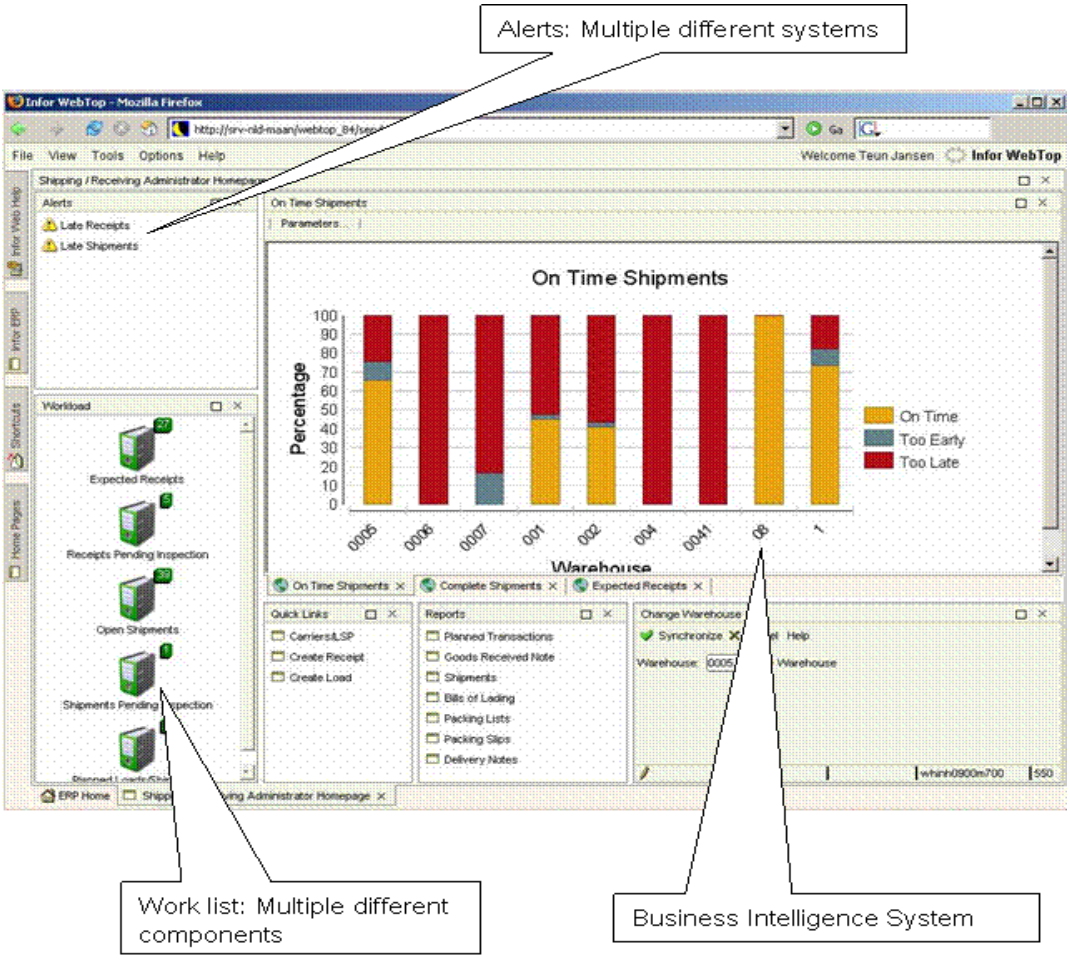


**Figure 6: Composite User Interface (CUI) - Data from Multiple Components is aggregated into one Screen.**

One component can store the essence of a business object and then provide a way to navigate to the SOR for that information. For example, a shipping component that has the order number is shipped along with customer information, shipping quantity and delivery information. For more detail on the order, it provides a link to the SOR of the order and passes the order number, along with the link that allows the end-user to navigate through the system in a systematic fashion.

**Single Sign-on and Security**

A centralized user management, enterprise-wide security model with single sign-on is needed with composite user interfaces and multiple components. The security context connects the components to the messaging infra-

structure and the users to the back-end components. By providing a single sign-on infrastructure, each user can seamlessly access data in each component that he/she is authorized to. By leveraging a single security model, the information can be secured in each component individually. While both the single sign-on and security model provide a certain level of coupling between applications, it is needed to provide a holistic experience for end-users.

**APPLICATION AND BENEFITS OF THE MODEL**

The EDSOA model can be used to develop new EAs or extend existing ones. INFOR, a developer of EA software, used EDSOA to integrate an Internet store into an existing EA that supported the clients' bricks and mortar operation. The following example shows how they used EDSOA to solve the four challenges associated with traditional SOA described earlier.

In traditional SOA, when a customer makes any request on the website (e.g. product search, login, credit card verification) a service request goes to the central database that stores all core business data. Since other components are making requests at the same time the number of messages that have to be pushed over to the central data center is extremely high, which leads to potential dependency, latency and load conflicts. The problem is amplified with multiple hosted web front-ends in many locations.

The EDSOA solution handles messages in an asynchronous way, based on business events as they happen. Hence the architecture is not engineered around the maximum number of messages in a pre-defined response period, but rather over a much longer time, several minutes versus seconds. For example, whenever a customer is added to the back-end system it sends a message (SyncCustomerPartyMaster) over to the Web-Store. The Web-Store keeps a copy that is used for local lookups. Pricing rules are sent per message to the Web-Store where they are put into a pricing engine that can execute the pricing and promotion requests on demand with the help of a local proxy component. The allocation of inventory to an instance of a Web-store allows for the sale of items without having to validate with the inventory management component that the item is being sold. Web-Store creates an order and sends the order to the back-end system. Since the inventory has already been allocated to this instance and pricing has taken place, the order can be sent whenever the channel is open. As far as the user is concerned, the order has been taken.

While building Web-Store using synchronous services based messages in an SOA framework will give a good shopping experience under low load, it suffers from several problems that EDSOA resolves. The performance of the EDSOA model is much simpler to improve by installing multiple, independent instances of the Web-Store component. The Web-Store can be continuously operating while the back-end EA can be turned off for necessary maintenance. The introduction of new features on either end is less likely to disrupt the operation, and it is easier to troubleshoot in an environment where all the messages are easy to read and the flow can be interrupted to validate content.

The EDSOA model combines the benefits of component-based software development, event-driven architecture (EDA) and SOA, and at the same time overcomes the challenges of traditional SOA, described earlier. The introduction of a Master Data Management component and the use of asynchronous messaging gether with the EOE reduce dependency conflicts. The EOE knows which component is the SOR for the table that needs to be updated and sends a message to that specific component so that no two services try to update the same table at the same time.The use of components and proxy components assures quick response time even in high load situations. In addition, it allows for high scalability, since components can be easily duplicated on a variety of hardware environments.

EDSOA also overcomes the challenges of linkage of implementation and services and rigidity of services. In EDSOA, individual components can easily be maintained, upgraded and even exchanged without affecting the entire EA or interfering with day-to-day operations. For example, a message to update the inventory could be sent to the "old" inventory management component, while the next message to update the inventory could be sent to the "new" inventory management component.

The EDSOA model is especially useful for software development companies that provide solutions for a wide variety of customers and industries. Whereas most order processing components will be applicable for a wide variety of customers, customer relationship components can be written to the specifications of the individual cus-

tomer. A company that wants an EA can select and combine standard components to fit its individual business processes.

**ACKNOWLEDGEMENT**

We appreciate insightful feedback from Chuck Kacmar.

**AUTHOR INFORMATION**

**Helgi S. Sigurdsson** is the VP of Research, Chief Architect at Infor Global Solutions. He has a degree from the University of Iceland in Electrical Engineering, where his thesis was on optimizing neural networks using genetic algorithm. Currently, Helgi focuses on building application frameworks and core technologies for next generation enterprise applications, that provide the flexibility, ease of use and speed that is needed.

**Karina Hauser** is an associate professor in the Management Information Systems department at Utah State University. She received her PhD in Decision Science and Information Technology at the University of Kentucky on a Toyota Fellowship. Her research interests are in Web Development, Web Design and Lean Manufacturing. Before going into academia, Karina spent 16 years in industry, first as a programmer and later as a consultant and project manager for Enterprise Resource Planning systems, mainly in the automotive sector. Her research appeared in journals such as the *Journal of Information Systems Education*, *Journal of Computer Information Systems*, and *International Journal of Production Research.*

**Katherine M. Chudoba** is Associate Professor of MIS in the Jon M. Huntsman School of Business at Utah State University. Her research focuses on the nature of work in distributed environments, and how Information and Communication Technologies (ICTs) are used and integrated into work practices. She has published in journals such as *MIS Quarterly*, *Organization Science*, and *Information Systems Journal*. She earned her Ph.D. at the University of Arizona, and her bachelor's degree and MBA at the College of William and Mary. Before joining academe, she worked as an analyst and manager with IBM.

**REFERENCES**

1. Agrawal, R., Bayardo Jr., R., Gruhl, D., & Papadimitriou, S. (2002). Vinci: a service-oriented archtecture for rapid develpment of Web applications. *Computer Networks, 39*, 523-539.
2. Booth, D., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004). Web Service Architecture. May, from http://www.w3.org/TR/ws-arch/#whatis
3. Brancheau, J. C., Janz, B. D., & Wetherbe, J. C. (1996). Key Issues in Information Systems Management: 1994-95 SIM Delphi results. *MIS Quarterly, 20*(2), 225-242.
4. Byrd, T.A., & Turner, D. E. (2001). An Exploratory Examination of the Relationship Between Flexible IT Infrastructure and Competitive Advantage. *Information & Management, 39*(1), 41-52.
5. Chang, M., He, J., Tsai, W. T., Xiao, B., & Chen, Y. (2006). UCSOA: User-Centric Service-Oriented Architecture. Paper presented at the IEEE International conference on 2-business engineering, Shanghai, China.
6. Davenport, T. H., & Linder, J. (1994). Information Management Infrastructure: The New Competitive Weapon. Paper presented at the Proceedings of the 27th Annual Hawaii International Conference on Systems Sciences, Hawaii, USA.
7. Dietrich, A. J., Kirn, S., & Sugumaran, V. (2007). A service-oriented architecture for mass customization - A shoe industry case study. *IEEE Transactions on Engineering Management, 54*(1), 190-204.
8. El Sawy, O., Malhotra, A., Gosain, S., & Young, K. (1999). IT-Intensive Value Innovation in the Electronic Economy: Insights from Marshall Industries. *MIS Quarterly, 23*(3), 305-335.
9. Elfatatry, A. (2007). Dealing with Change: Components versus Services. *Communications of the ACM, 50*(8), 35-39.
10. Kart, F., Shen, Z., & Gerede, C. E. (2006). The MIDAS System: A Service Oriented Architecture for Automated Supply Chain Management. Paper presented at the IEEE International Conference on Services Computing.
11. Knippel, R. (2005). Service-Oriented Enterprise Architecture.

12. Marks, E. (2007). Executive Bulletin: Sold on SOA. Retrieved October, from http://www.computerworld.com/action/executivebriefings.do?command=viewExecutiveBriefingDetail&contentId=9000103
13. Mimoso, M. S. (2004). Survey: SOA Prominent on 2005 Budgets. Retrieved July, from http://searchsoa.techtarget.com/news/article/0,289142,sid26_gci1010622,00.html?Offer=TMIRsept04
14. Mitchell, V. (1996). Integration and Information Technology Project Performance. *MIS Quarterly, 30*(4), 919-939.
15. Papazoglou, M. P., & Van Den Heuvel, W.-J. (2006). Service-oriented design and development methodology. *International Journal of Web Engineering and Technology, 2*(4), 412-442.

**NOTES**