Event-Driven Architecture: SOA Through the Looking Glass

by Udi Dahan

Summary

This article proposes that event-driven architecture (EDA) and service-oriented architecture (SOA) are really two sides of the same coin.

Introduction

While event-driven architecture (EDA) is a broadly known topic, both giving up ACID integrity guarantees and introducing eventual consistency make many architects uncomfortable. Yet it is exactly these properties that can direct architectural efforts toward identifying coarsely grained business-service boundaries—services that will result in true IT-business alignment.

ACID is an acronym for Atomicity, Consistency, Isolation, Durability: a set of properties that traditionally describe database transactions.

Business events create natural temporal boundaries across which there is no business expectation of immediate consistency or confirmation. When they are mapped to technical solutions, the loosely coupled business domains on either side of business events simply result in autonomous, loosely coupled services whose contracts explicitly reflect the inherent publish/subscribe nature of the business.

This article will describe how all of these concepts fit together, as well as how they solve thorny issues such as high availability and fault tolerance.

A *temporal boundary* is a boundary on the axis of time, where one side of the boundary exists in a time-frame disjoint from the other side.

Commands and Events

To understand the difference in nature between "classic" service-oriented architecture (SOA) and event-driven architecture (EDA), we must examine their building blocks: the command in SOA, and the event in EDA.

In the commonly used request/response communication pattern of service consumer to service provider in SOA, the request contains the action that the consumer wants to have performed (the command), and the response contains either the outcome of the action or some reaction to the expressed request, such as "action performed" and "not authorized."

Commands are often named in imperative, present-tense form—for example, "update customer" and "cancel order."

In EDA, the connection between event emitters and event consumers is reversed from the previously described SOA pattern. Consumers do not initiate communication in EDA; instead, they receive events that are produced by emitters. The communication is also inherently unidirectional; emitters do not depend on any response from consumers to continue performing their work.

Events are often named in passive, past-tense form—for example, "customer updated" and "order cancelled"—and can represent state changes *in the domain of the emitter.*

Events can be thought of as mirror images of the commands in a system. However, there might be cases in which the trigger for an event is not an explicit command, but something like a timeout.

Business Processes with Commands and Events

The difference between commands and events becomes even more pronounced as we look at each one as the building block in various business processes.

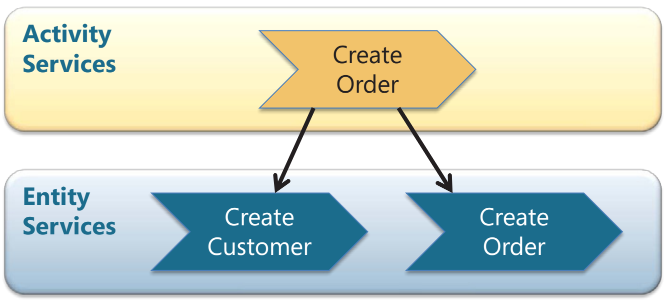
When we consider commands such as "create customer" and "create order," we can easily understand how these commands can be combined to create more involved scenarios, such as: "When creating an order, if a customer is not provided, create a new customer." This can be visualized as services that operate at different layers, as shown in Figure 1.

One can also understand the justification for having activity services perform all of their work transactionally, thus requiring one service to flow its transactional context into other lower-level services. This is especially important for commands that deal with the updating of data.

When working with commands, in each step of the business process, a higher-level service orchestrates the work of lower-level services.

When we try to translate this kind of orchestration behavior into events, we must consider the fact that events behave as mirror images of commands and represent our rules by using the past tense.

Figure 1: Commands and service orchestration



> Instead of: "When creating an order, if a customer is not provided, create a new customer."
>
> We have: "When an order *has been created*, if a customer was not provided, create a new customer."

It is clear that these rules are not equivalent. The first rule implies that an order should not be created unless a customer—whether provided or new—is associated with it. The second rule implies that an order can be created even if a customer has not been provided—stipulating the creation as a separate and additional activity.

To make use of EDA, it is becoming clear that we must think about our rules and processes in an event-driven way, as well as how that affects the way in which we structure and store our data.

Event-Driven Business Analysis and Database Design

When we analyze the "When an order has been created, if a customer was not provided, create a new customer" rule, we can see that a clear temporal boundary splits it up into two parts. In a system that has this rule, what we will see is that at a given point in time, an order might exist that does not have a corresponding customer. The rule also states the action that should be taken in such a scenario: the creation of a new customer. There might also be a nonfunctional requirement that states the maximum time that should be allowed for the action to be completed.

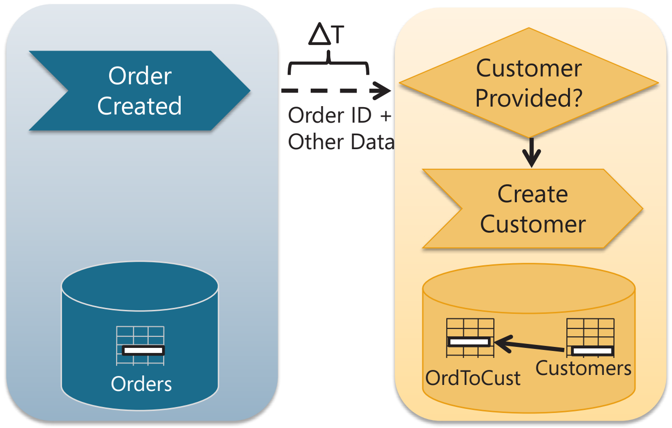
From a technical/database perspective, it might appear that we have allowed our data to get into an inconsistent state; however, that is only if we had modeled our database so that the Orders table had a non-nullable column that contained CustomerId—a foreign key to the Customers table. While such an entity-relationship design would be considered perfectly acceptable, we should consider how appropriate it really is, given the requirements of *business consistency*.

The rule itself indicates the business perspective of consistency; an order that has no connection to a customer is valid, for a certain period of time. Eventually, the business would like a customer to be affiliated with that order; however, the time frame around that can be strict (to a level of seconds) or quite lax (to a level of hours or days). It is also understandable that the business might want to change these time frames in cases in which it might provide a strategic advantage. An entity-relationship design that would reflect these realities would likely have a separate mapping table that connected Orders to Customers—leaving the Orders entity free of any constraint that relates to the Customers entity.

That is the important thing to understand about eventual consistency: It starts by identifying the business elements that do not have to be 100-percent, up-to-the-millisecond consistent, and then reflecting those relaxed constraints in the technical design.

In this case, we could even go so far as to have each of these transactions occur in its own database, as shown in Figure 2.

Figure 2: Event-driven data flows



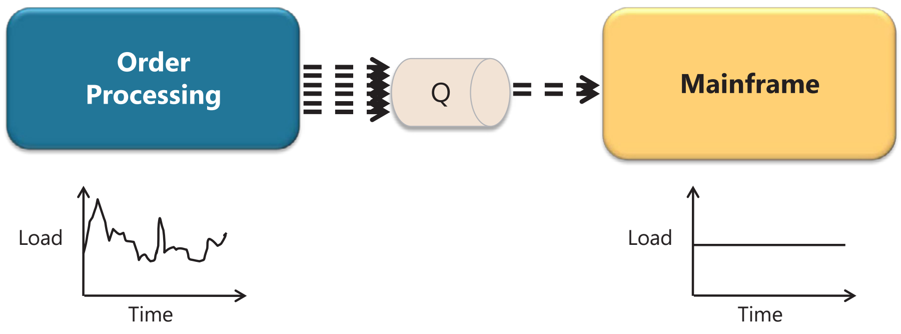
Benefits of Event-Driven Architecture

Given that EDA requires a rethinking of the core rules and processes of our business, the benefits of the approach must be quite substantial to make the effort worthwhile—and, indeed, they are. By looking at Figure 2, we can see very loose coupling between the two sides of the temporal boundary. Other than the structure of the event that passes from left to right, nothing is shared. Not only that, but after the event is published, the publisher no longer even needs to be online for the subscriber to process the event, so long as we use a durable transport (such as a queue).

These benefits become even more pronounced when we consider integration with other systems. Consider the case in which we want to integrate with a CRM, whether it is onsite or hosted in the cloud. In the EDA approach, if the CRM is unavailable (for whatever reason), the order will still be accepted. Contrasting this with the classic command-oriented service-composition approach, we would see there that the unavailability of the CRM would cause the entire transaction to time out and roll back. The same is true during integration of mainframes and other constrained resources: Even when they are online, they can process only N concurrent transactions (see Figure 3). Because the event publisher does not need to wait for confirmation from any subscriber, any transactions beyond those that are currently being processed by the mainframe wait patiently in the queue, without any adverse impact on the performance of order processing.

If all systems had to wait for confirmation from one another—as is common in the command-oriented approach—to bring one system to a level of 5 nines of availability, all of the systems that it calls would need to have the same level of availability (as would the systems that they call, recursively). While the investment in infrastructure might have business justification for one system (for example, order processing), it can be ruinous to have to multiply that level of investment across the board for nonstrategic systems (for example, shipping and billing).

Figure 3: Load-leveling effect of queues between publishers and subscribers



Service-Oriented Architecture and Its Implication in Business

by Manoj Manuja, Rajender Kalra, and Ruchi Malhotra

Service-oriented architecture (SOA) is one of the more buzzed architectures that are being adopted in business these days. It is meant to create a business model that is agile, flexible, and cost-effective. SOA is complemented by event-driven architecture (EDA). An EDA can be designed with systems that produce and transmit events among the loosely coupled service-oriented systems. These events fire certain triggers that, in turn, activate the services.

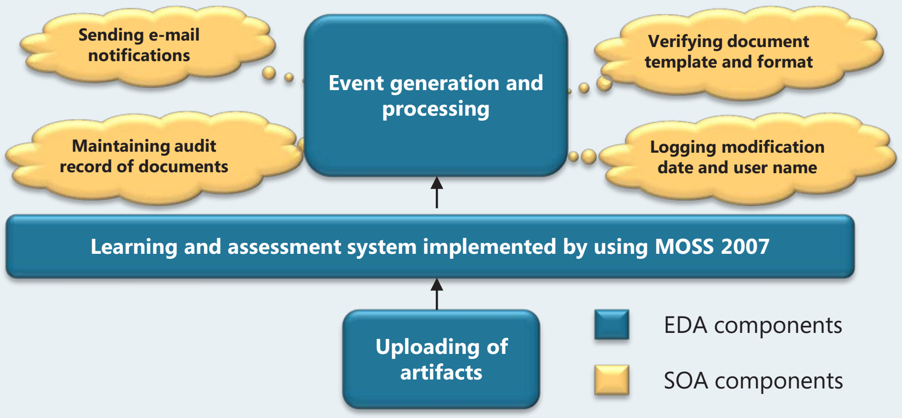
This can be depicted by a case study of a learning and assessment system that exists in a business enterprise (see Figure 1). The whole enterprise is driven by an SOA that is designed to handle and integrate copious subsystems. The learning infrastructure that is in place is meant to develop the competency of employees on various technologies that are prevalent in the software industry. There is a separate team of technology experts in place who run the complete infrastructure. The company relies on its own proprietary study material and other artifacts for training and assessment of its employees.

The whole infrastructure for the management and access of these artifacts by the technology-focus team and the employees of the company has been automated by using Microsoft Office SharePoint Server (MOSS) 2007, which is the latest portal-development product from Microsoft. The artifacts are updated on a monthly basis. This process is decentralized and is performed by the respective technology teams.

In this process, some events are produced that, in turn, fire triggers. The triggers activate services that exist in the complete organizational infrastructure that is based on SOA. The services carry out tasks such as sending e-mail notifications to the concerned persons, maintaining an audit record of the documents, verifying the document template and format, and logging the modification date and the name of the user who has performed the update. In this way, the SOA of the entire organization is supported by the event-based system that is implemented by using MOSS 2007, as shown in Figure 1.

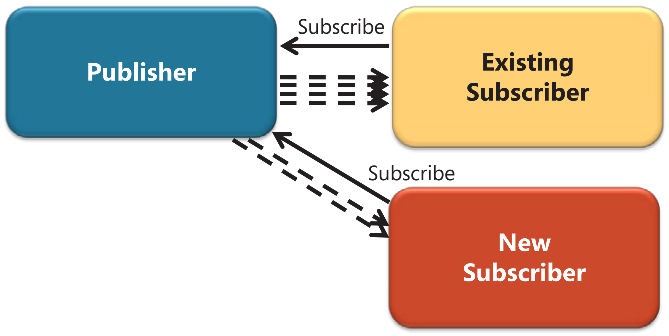
The role of EDA in implementing SOA is very significant. The functioning of the various services that form part of a SOA can depend on the occurrence of certain events. In the case study that is discussed in this article, the use of MOSS 2007 for management of the company's learning and assessment system largely contributed to the management of documents, as well as the running of the SOA infrastructure within the enterprise. The use of MOSS 2007 decentralizes the whole process, and the events that are produced in it also drive the SOA services. Along with EDA, SOA implementation in an enterprise leads to better productivity and improved customer satisfaction, as it provides increased control over business.

Figure 1: SOA supported by EDA implemented by using MOSS 2007


Sending e-mail notifications
Event generation and processing
Verifying document template and format
Maintaining audit record of documents
Logging modification date and user name
Learning and assessment system implemented by using MOSS 2007
Uploading of artifacts
EDA components
SOA components

Manoj Manuja (Manoj_Manuja@infosys.com), **Rajender Kalra** (Rajender_Kalra01@infosys.com), and **Ruchi Malhotra** (Ruchi_Malhotra@infosys.com) belong to the Education and Research Department, Infosys Technologies Limited, in Chandigarh, India.

Figure 4: Adding new subscriber to existing publisher



In companies that are undergoing mergers or acquisitions, the ability to add a new subscriber quickly to any number of events from multiple publishers without having to change any code in those publishers is a big win (see Figure 4). This helps maintain stability of the core environment, while iteratively rolling out bridges between the systems of the two companies. When we look practically at bringing the new subscriber online, we can take the recording of all published events from the audit log and play them to the new subscriber, or perform the regular ETL style of data migration from one subscriber to another.

IT-Business Alignment, SOA, and EDA

One of the more profound benefits that SOA was supposed to bring was an improved alignment between IT and business. While the industry does not appear to have settled on how this exactly is supposed to occur, there is broad agreement that IT is currently not aligned with business. Often, this is described under the title of application "silos."

To understand the core problem, let us try to visualize this lack of alignment, as shown in Figure 5.

What we see in this lack of alignment is that IT boundaries are different from business boundaries, so that it is understandable that the focus of SOA on explicit boundaries (from the four tenets of service orientation) would lead many to believe that it is the solution. Yet the problem that we see here is while there *are* explicit technical boundaries between App 1 and App 2, the *mapping* to business boundaries is wrong.

Figure 5: Lack of IT-business alignment

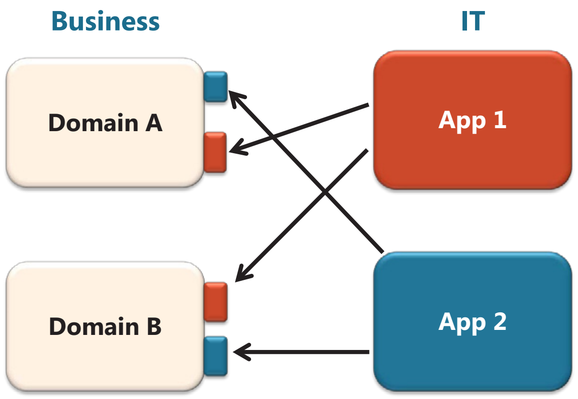
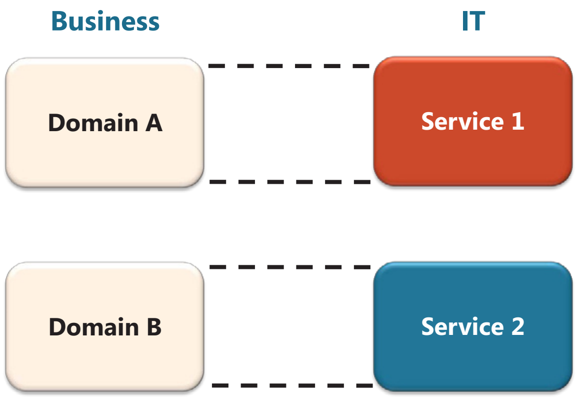


Figure 6: Services aligned with business boundaries



If SOA is to have any chance of improving IT-business alignment, the connection between the two needs to look more like the one that is shown in Figure 6.

One could describe such a connection as a service "owning" or being responsible for a single business domain, so that anything outside the service could not perform any actions that relate to that domain. Also, any and all data that relates to that domain also would be accessible only within the service. The EDA model that we saw earlier enabled exactly that kind of strict separation and ownership—all the while, providing mechanisms for interaction and collaboration.

We should consider this strong connection when we look at rules such as: "When an order has been created, if a customer was not provided, create a new customer." The *creation* of the order as an object or a row in a database has no significance in the business domain. From a business perspective, it could be the *acceptance* or the *authorization* of an order that matters.

What SOA brings to EDA in terms of IT-business alignment is the necessity of events to represent meaningful business occurrences.

For example, instead of thinking of an entity that is being deleted as an event, you should look for the business scenario around it—for example, a product that is being discontinued, a discount that is being revoked, or a shipment that is being canceled. Consider introducing a meaningful business status to your entities, instead of the technically common "deleted" column. While the business domain of sales will probably not be very interested in discontinued products and might treat them as deleted, the support domain might need to continue troubleshooting the problems that clients have with those products—for a while, at least. Modern-day collaborative business-analysis methodologies such as value networks can help identify these domains and the event flows between them.

What an EDA/SOA Service Looks Like

In the context of combined EDA and SOA, the word "service" is equivalent to a logical "thing" that can have a database schema, Web Services, and even user-interface (UI) code inside it. This is a very different perspective from the classic approach that considers services as just another layer of the architecture. In this context, services cut across multiple layers, as shown in Figure 7.

In this model, the processes that are running on various computers serve as generic, composite hosts of service code and have no real logical "meat" to them.

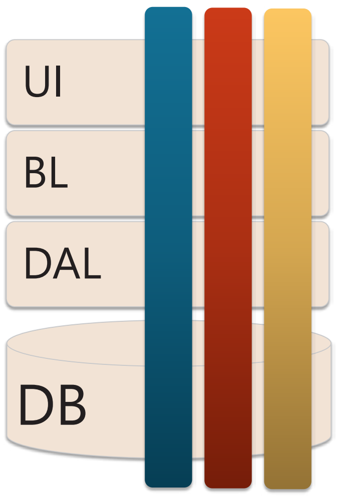
When we look at the code in each of the layers in light of the business domain that it addresses, we tend to see fairly tight coupling between a screen, its logic, and the data that it shows. The places in which we see loose coupling is between screens, logic, and data from different business domains; there is hardly any coupling (if at all) between the screen that shows employee details and the one that is used to cancel an order. The fact that both are screens and are categorized in the UI "layer" appears not to have *much* technical significance (if any business significance). Much the same can be said for the code that hooks those screens to the data, as well as the data structures themselves.

Any consistency concerns that might have arisen by this separation have already been addressed by the business acceptance of eventual consistency. If there are business demands that two pieces of data that have been allocated to different services always be consistent, this indicates that service boundaries are not aligned with business boundaries and must be changed.

This is extremely valuable. Architects can explain to the business the ramifications of their architectural decisions in ways that the business can understand—"There might be a couple of seconds during which these two bits of data are not in sync. Is that a problem?"—and the answer to those kinds of question is used to iterate the architecture, so as to bring it into better alignment with the business.

As soon as service boundaries reflect business boundaries, there is great flexibility within each service; each can change its own database schema without having to worry about breaking other services, or even choose to change vendors and technology to

Figure 7: Services logically connecting code from different layers



such things as object or XML databases. Interoperability between services is a question of how event structures are represented, as well as how publish/subscribe is handled. This can be done by using basic enterprise service bus (ESB) functionality, such things as the Atom Publishing Protocol, or a mix.

Integration of legacy applications in this environment occurs *within* the context of a service, instead of identifying them as services in their own right. Use of Web Services to ease the cost of integration continues to make sense; however, from the perspective of a business domain, it really is nothing more than an implementation detail.

Conclusion

EDA is not a technical panacea to Web Services–centric architectures. In fact, attempting to employ EDA principles on purely technical domains that implement command-centric business analysis will almost certainly fail. The introduction of eventual consistency without the ratification of business stakeholders is poorly advised.

However, if in the process of architecture we work collaboratively with the business, map out the natural boundaries that are inherent in the organization and the way in which it works, and align the boundaries of our services to them, we will find that the benefits of EDA bring substantial gains to the business in terms of greater flexibility and shorter times to market, while its apparent disadvantages become addressed in terms of additional entity statuses and finer-grained events.

By itself, EDA ignores the IT-business alignment of SOA—so critical to getting boundaries and events right. Classic SOA has largely ignored the rock-solid foundation of publish/subscribe events—dead Web Services eventing and notification standards notwithstanding. It is only in the fusing of these two approaches that they overcome the weaknesses of each other and create a whole that is greater than the sum of its parts.

Interestingly enough, even though we have almost literally turned the classic command-driven services on their heads, the service-oriented tenets of autonomy and explicit boundaries have only become more pronounced, and the goal of IT-business alignment is now within our grasp.

Beyond just being a sound theoretical foundation, this architecture has weathered the trials of production in domains such as finance, travel and hospitality, aerospace, and many others—each with its own challenging constraints and nonfunctional demands. Organizations have maximized the effectiveness of their development teams by structuring them in accordance with these same service boundaries, instead of the more common technical specialization that corresponds to layered architectures. These loosely coupled service teams were able to wring the most out of their agile methodologies, as competition for specialized shared resources was eliminated.

Oracle once named this approach SOA 2.0. Maybe it really is the next evolutionary step.

Toward Web-Scale SOA

by Carlos Pedrinaci, Elena Simperl, Reto Krummenacher, and Barry Norton

Opening SOA technologies to the Web has important implications from an engineering perspective, as well as with respect to the usage of technology that one should expect and accommodate. Typically, these implications have been overlooked; as a consequence, SOA remains mostly an enterprise-specific solution, and its adoption for supporting the creation of distributed systems on the Web has largely fallen behind initial expectations.

The Web is built upon the essential principle of *openness*, which has defined its character and led to its growth to become the world's richest source of information in its 20 years of existence. Anyone can provide information, and anyone else can use this information for whatever purpose is deemed appropriate. However, this lack of centralized control and the scale that characterizes the Web clash with current Web Services and SOA-based solutions, in which services are typically known in advance and stored in some centralized repository, and flexibility in consumption is not contemplated.

A second Web principle of particular relevance is the basis of communication through *persistent publication*. This style of communication allows efficient asynchronous communication between one provider and many (possibly unknown) consumers. This simple mechanism supports an unprecedented level of creation, flow, and recombination of information, which contributes significantly to the enrichment of the Web. In contrast, service-oriented systems are most often limited to communication on the basis of one-to-one synchronous messaging. This fails to take into account that systems might suddenly disappear and new consumers who have different requirements might appear, and it maintains unnecessarily a strict separation between procedures and data.

The E.U. project that is known as Service Oriented Architectures for All (SOA4All) is working toward an architecture and language stack for a service-delivery platform that fosters the Web-scale adoption of service technologies. This architecture extends SOA with essential principles upon which the Web builds, such as openness and the support for communication that is based on persistent publication. Additionally, we adopt semantic technologies as a means to lift services and their descriptions to a level of abstraction that deals with computer-understandable conceptualizations, so as to increase the level of automation that can be achieved while carrying out common tasks during the life cycle of services, such as their discovery, composition, and invocation.

Further extensions come from the adoption of Web 2.0 principles—notably, the tight integration of people as service prosumers and RESTful services as a technology that complements traditional Web services. Finally, automated context-adaptation capabilities are embedded within the architecture to support the use of services in unforeseen contexts—thus, increasing the versatility of services while retaining their manageability.

For more information, visit the Service Oriented Architectures for All (SOA4All) Web site.

Dr. Carlos Pedrinaci is a research fellow at the Knowledge Media Institute (KMi) at the Open University, United Kingdom.

Dr. Elena Simperl works as senior researcher and vice-director of the Semantic Technology Institute at the University of Innsbruck.

Reto Krummenacher is researcher and project assistant for the Semantic Technology Institute at the University of Innsbruck.

Barry Norton is a senior researcher for STI Innsbruck at the University of Innsbruck.

References

ACID

Four Tenets of Service Orientation

Value Networks

The Atom Publishing Protocol

About the Author

Udi Dahan is an internationally renowned expert on software architecture and design. Recognized four years in a row with the coveted Most Valuable Professional (MVP) award by Microsoft Corporation for solutions architecture and connected systems, he is also on the advisory board of Microsoft's next-generation technology platforms: WCF/WF/OSLO, the Software Factories Initiative, and the Composite Application Library & Guidance. He provides clients all over the world with training, mentoring, and high end–architecture consulting services—specializing in service-oriented, scalable, and secure enterprise architecture and design.

Dahan is one of 33 experts in Europe who are recognized by the International .NET Association (INETA); an author and trainer for the International Association of Software Architects on Reliability, Availability, and Scalability; and an SOA, Web Services, and XML guru who is recommended by *Dr. Dobb's*—the world's largest software magazine.

Follow up on this topic

- Enterprise Service Bus Toolkit 2.0

How Managed Is Your SOA?

by Aarti Kaur

The dynamic, collaborative nature of today's business processes makes it necessary to have a scalable and flexible integration approach. This is when the paradigm shift occurred and the IT industry came up with SOA as one of the most viable solutions for EAI. The big question is whether implementing a pure SOA will address the real-time business issues in the long term.

Business Problem

The basic services that are provided by telecom providers are commoditized, which forces telecom operators to come up with innovative value-added services that will distinguish them from their competitors and help them retain their existing subscriber base. Introduction of a new service means the ability to test the service quickly in the market and, if it is good enough, the ability to scale as quickly as possible. On the other hand, if the service is not very popular, telecom operators should be able to decommission the service and replace it with another, without a lot of integration effort. Introduction of a value-added service is marred by various issues—such as time-to-market delay, fragmented OSS/BSS platforms, and high operational and development costs—which results in a lack of flexibility to introduce new business models and products rapidly.

SOA Solution

Business-Process Centralization

One of the key issues with the current system is business-process redundancy; that is, based on the type of service that is requested, the appropriate provisioning and billing subsystems are invoked. This process can be automated as an orchestration that consists of all the steps (validation, provisioning, and billing) in a sequential arrangement. The orchestration invokes the respective generic service at each step.

Service Identification and Design

In the current context, one of business pain areas is integration with legacy systems, which leads to high integration and maintenance costs. Introduction of a level of service abstraction that is composed of a set of generic services (which will implement the major service-processing steps, such as validation, provisioning, and billing) can address the aforementioned issue. These, in turn, will call core services that are based on the requested service type, line type, and other parameters. Core services will interact with the BSS/OSS systems, whenever it is required. Some examples of core services are the SMS Validation Service, SMS Provisioning Service, SMS Billing service, and MMS Provisioning Service.

Event-Driven SOA

Events can be of two types: internal system events and business events. A *business event* is any meaningful activity that alters the flow of a business process or triggers a new process. The subscription to a service by an end user is an example of a business event, which will invoke the orchestration. Usually, services follow a simple request-reply communication, which might not be appropriate in the current scenario. For instance, the generic provisioning service will interact with the appropriate provisioning system; the time that is taken to provision a service can vary, depending on the type of service that is requested. In this case, the provisioning service should be able to signal the orchestration when the provisioning is complete. This is an example of an internal event and can be addressed by using asynchronous Web services, such as "WaitHandlers" or the WCF Callback mechanism.

Managing SOA with ESB

ESB is the infrastructure for managing an SOA. The Microsoft ESB Toolkit 2.0 itinerary Web service can be used for service discovery and invocation. The resolver mechanism can provide a generic service resolution. An itinerary service can be an entry point to this system; it will read the message and use the ESB resolver mechanism internally to invoke the business-process orchestration. Addition of a new service in the system means registering the service in a service registry and using the appropriate discovery mechanism. Message transformations between services can be performed by using Microsoft BizTalk maps—thus, each service will complete its processing, and the ESB will take care of the message transformations and routing.

Conclusion

SOA is a very powerful architectural concept; but, all by itself, it might not address the dynamics of a real-time business. Managing SOA with ESB offers many potential benefits; also, events and services complement each other and cannot be looked at in isolation from one other. In combination with event processing and ESB, SOA can provide an optimal solution that not only addresses the business complexities, but also adds value to it.

Aarti Kaur is a solution architect with Mahindra Satyam, India. The complete article is available on her blog.