

The SOA Magazine

Feature Article



SOA and EDA: Using Events to Bridge Decoupled Service Boundaries

by Jack van Hoof

Published: January 29, 2007 (SOA Magazine Issue IV: February 2007, Copyright © 2007)

[Download this article as a PDF document.](#)

Abstract: The distinction between service-oriented architecture (SOA) and event-driven architecture (EDA) can be traced to message patterns. Understanding the implications of common exchange patterns, such as request-and-reply and publish-and-subscribe, helps determine both fundamental differences and commonality in these two complementary architectural models.

It is appropriate and desirable to use the acronyms "SOA" and "EDA" to make this distinction, because both of these architectural styles are positioned in the same domain; SOA focusing on the decomposition of business functions and EDA focusing on business events. This article explores the differences between these models and specifically studies how EDA patterns can be used to connect decoupled service domains within SOA implementations.

Introduction: On-Demand Business and the Importance of Agility

Organizations tend to change their structure frequently. The increasing emphasis on service-orientation and globalization supports this trend. As a result, much of the world is preparing for network-oriented business structures with independent, autonomous service providers and service consumers. Parts of the business process will be outsourced to external partners, while departments and business units are transformed to service providers. Some of these service providers are no longer focused only internally on the organization, but are seeking a presence in external markets as well.

Everything is moving toward on-demand business where service providers react to impulses - events - from the environment. To excel in a competitive market, a high level of autonomy is required, including the freedom to select the appropriate supporting systems. This magnified degree of separation creates a need for agility; a loose coupling between services so as to support continuous, unimpeded augmentation of business processes in response to the changing composition of the organizational structure.

To achieve "true" agility the supporting applications must be agnostic to organizational changes like reshuffling responsibilities and roles, out-sourcing or in-sourcing, the splitting up or consolidation of departments, and all kinds of other reorganizations. Furthermore, the agility of business processes to follow organizational changes must not be limited by the IT systems that support them. All of these requirements are accommodated by loose coupling. Services with reduced coupling are more easily able to "take the scissors" to the organizational structure without disturbing the continuity of the supporting IT systems. This is the basis of *organizational agility*.

Agility in application construction can be achieved by using shareable services in a well-defined functional boundary. Standards-based technology can also contribute to agility by addressing time-to-market concerns for the delivery of applications (as long as the standards are mature and adequate tools are used). Much of this ties into the ability to carry out a more agile restructuring of applications in support of business process redesign. The ultimate result is lower IT costs for the business and faster project delivery cycles.

Both EDA and SOA promise to increase agility on an organization level, but each goes about it in a in different way.

EDA in Relation to SOA

Though EDA and SOA share common goals, how do we know when to utilize event-driven and service-oriented approaches? Let's take a closer look at when each architectural style is most appropriate.

Unlike traditional distributed architectures, EDA does not advocate the synchronous command-and-control type of exchange pattern. Instead, it is based on the asynchronous publish-and-subscribe pattern, where the publisher is completely unaware of the subscriber and vice versa; services are loosely coupled in the sense that they only share the semantics of the message.

If you are seeking to support independence between business process steps, EDA can provide significant benefit, especially in federated and autonomous processing environments. Recognizable situations where EDA might be applicable are:

- *Horizontal* communication between tiers in a process chain.
- Processes that cross recognizable functional organization borders (internal and external).
- Processes that cross physical organizational boundaries.

However, when looking to apply strong cohesion in business processes, SOA can provide significant benefit. This is generally suitable in the following situations:

- *Vertical* interaction between hierarchical layers of functional decomposition.
- Functional request-and-response processes, such as man-machine dialogues where the user waits for an answer.
- Processes with a transactional nature which require commit and rollback facilities.

Like EDA, SOA is frequently implemented with a messaging framework that can therefore also leverage asynchronous message patterns to implement request-and-reply communication. This allows for loose coupling by separating the issuer of the request from the consumer of the reply as well as separating the consumer of the request and issuer of the reply. However, due to the common misperception that an SOA is synonymous with a (tightly-coupled) RPC-style architecture that happens to use Web services, there are many implementations in which traditional, synchronous exchange patterns are still deeply engrained.

To establish an environment wherein SOA and EDA can harmoniously co-exist, up-front analysis is required. Here are some suggested, preparatory steps:

1. Model business requirements into functions at the granularity level of the desired autonomy.
2. Outline the application landscape to identify all affected systems.
3. Map the application landscape to the business function model.
4. Identify applications that cross functional borders as potential "agility bottlenecks" (assign a special high priority to those applications that are required to cross external organization borders).

Step 4 forms the basis for the decoupled service boundaries further described in the upcoming section. It is this cross-boundary communication that we are very interested in when looking to position the EDA publish-and-subscribe patterns as a means of connecting these boundaries while allowing them to maintain their decoupled state.

Carrying out a simple process, such as this, can lay the groundwork for introducing an extent of loose coupling and establishes a good starting point for moving a technology landscape toward a modern, flexible and adaptable environment capable of leveraging both SOA and EDA together.

Loose Coupling and "Points of Decoupling"

Loose coupling means independence. Loosely coupled services quite simply have less dependency on each other than tightly coupled ones. This applies to both functionality and data. The level of coupling between services can be potentially increased (tightened) if data associated with a given service is isolated to that service's implementation boundary only. This design approach essentially increases the chances that other services will form dependencies on each other in order to access the isolated data. Coupling levels between services can be decreased (loosened) if data redundancy is allowed via established mechanisms such as replication, avoiding any kind of shared layers other than shared layers for transport and semantic harmonization.

Maintaining data redundancy across decoupled borders makes loose coupling more robust. EDA is suitable in this type of situation, as it can support automatic data synchronization mechanisms in redundant environments.

When working with SOA and EDA, it is furthermore helpful to find "points of decoupling" by looking for parts of the business process where you are sure they always will stay together in one organizational unit (strong cohesion and atomic business functions). In this way a functional composition of the business will become visible. The borderlines between the business functions are the points of decoupling. If atomic transactions cross decoupling borders, then compensating rollback transactions can be implemented at these decoupling points.

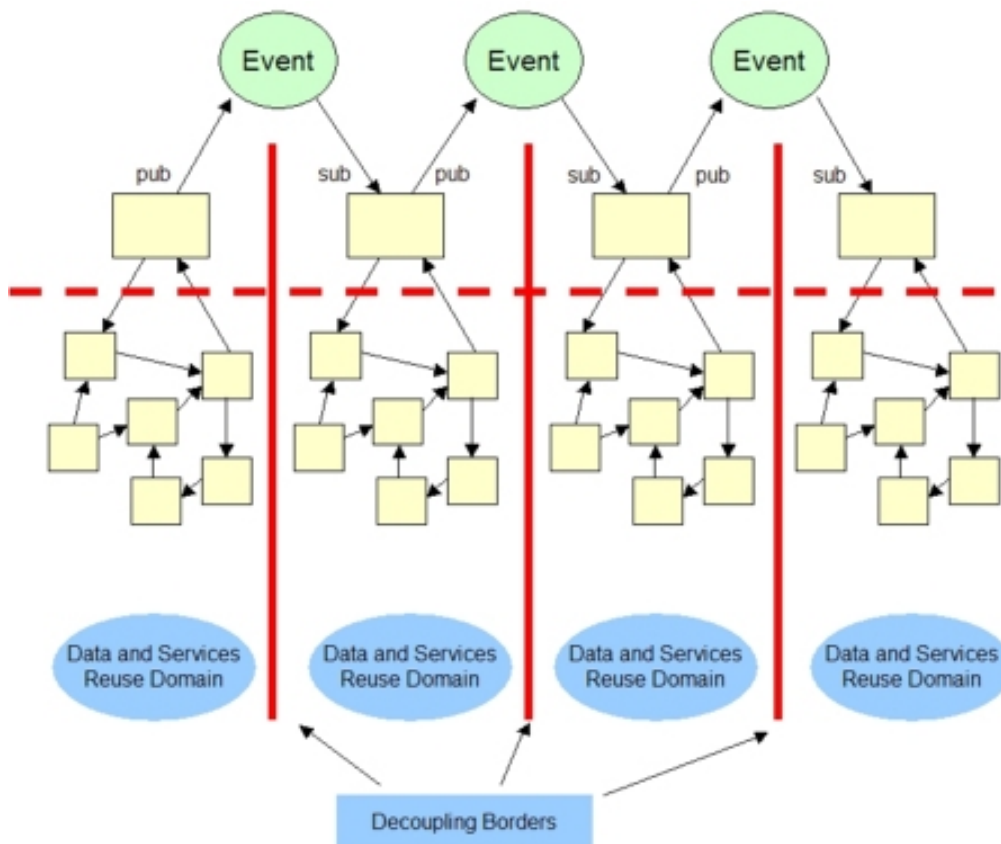


Figure 1: How events can be positioned to cross decoupled service boundaries.

Figure 1 illustrates a potential relationship between SOA and EDA. The circles at the top denote decoupling points (events) linking individually loosely coupled systems. At these decoupling points services can be connected and disconnected without any change to the connected systems. All data exchange between the domains takes place only at this point and not at the lower levels.

Within a reuse domain (as represented by the bottom layer in Figure 1), finer grained EDAs may be further implemented. The more fine-grained the EDA, the more flexible the systems are, but also the smaller the reuse domains will be.

If using SOAP-based Web services technologies at the points of decoupling combined with a common infrastructure (such as an Enterprise Service Bus), it becomes possible to easily connect heterogeneous systems, SOAP-wrapped legacy systems, commercial off-the-shelf software (COTS), ERP systems, and gateways to external systems (as shown in Figure 2).

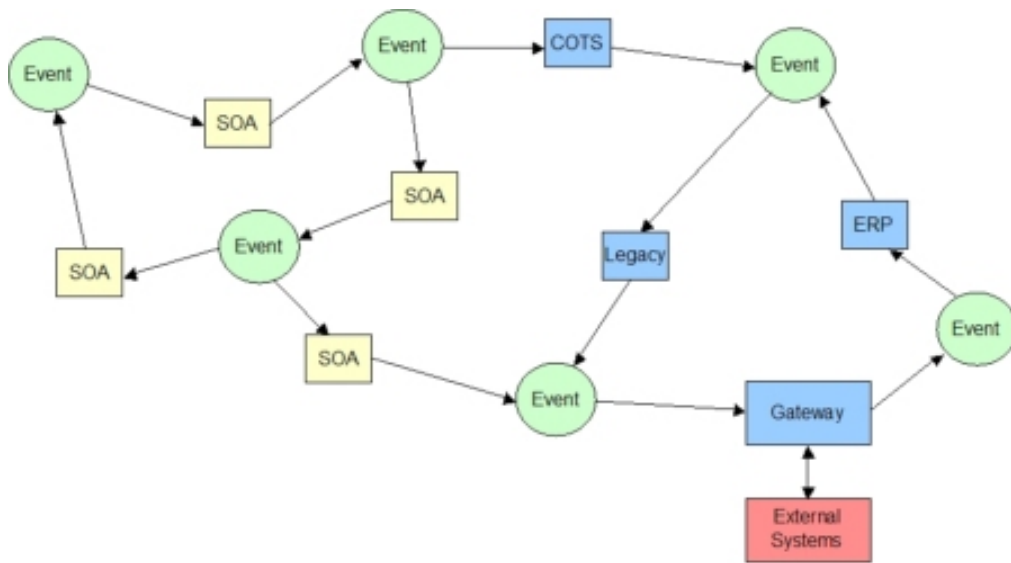


Figure 2: Services are no longer directly coupled, but instead connected via decoupling points represented by events.

Striving for loose coupling - and so for flexibility and agility - is always a good idea at all levels of granularity with respect to both EDA and SOA. Of course loose coupling must always be approached with a dose of reality, especially when it comes to runtime performance.

ESB Considerations and the Global Dataspace

To implement EDA with Web services technologies today, an additional SOAP-aware message queuing infrastructure is required. This is not the case for a Web services-based SOA implementation, because SOA in its basic form can be established solely by Web services over existing network infrastructures like the HTTP layer.

Current Enterprise Service Bus (ESB) infrastructures provide a way for message queuing to be combined with Web service technologies. That is why the use of an ESB is very appropriate to implement various combinations of EDA and SOA-based solutions.

It is also worth noting that there are several key emerging Web service standards relevant to both EDA and SOA. Some of the more prominent ones include WS-Eventing, WS-Notification, WS-MetadataExchange, WS-ReliableMessaging, WS-Security, and WS-CDL. Once these become widely supported and are then combined with emerging SOAP-aware infrastructure components (as they will be made available in new generation network devices and operating systems), they will naturally provide much of the ESB functionality which at the moment is only available within ESB vendor products.

From an EDA perspective, an ESB is very well suited to function as a container for published business events, because it allows the events to become widely available for subscription. As a result, the ESB can be positioned as the enterprise's global dataspace, making event intelligence uniformly accessible by all applications, regardless of location, time, and back-end technology (as explained in Figure 3).

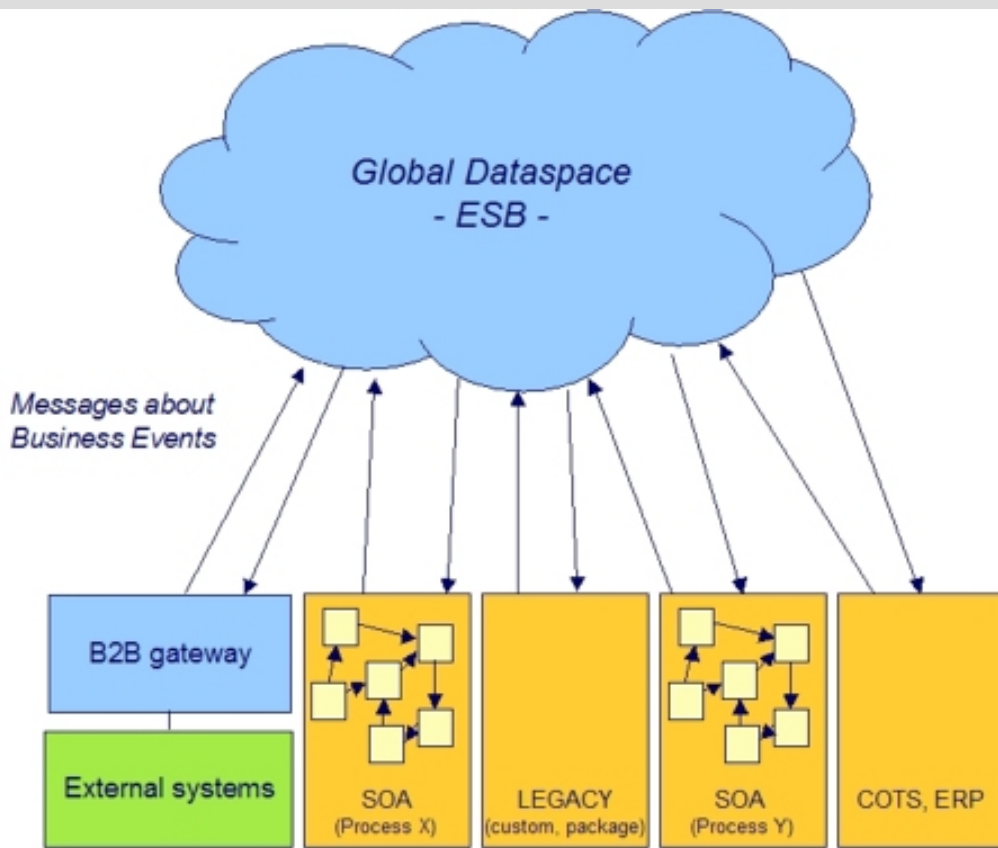


Figure 3: The global dataspace implemented by an ESB.

An ESB needn't be a one-vendor product. In a federated company multiple service bus products can be combined to provide ESB features, allowing for local autonomy. Figure 4 shows how an enterprise-wide service bus can serve as the global dataspace for an entire organization by hosting multiple service bus domains. Using Web service technologies further makes propagation of the messages across multiple vendor products relatively easy to implement.

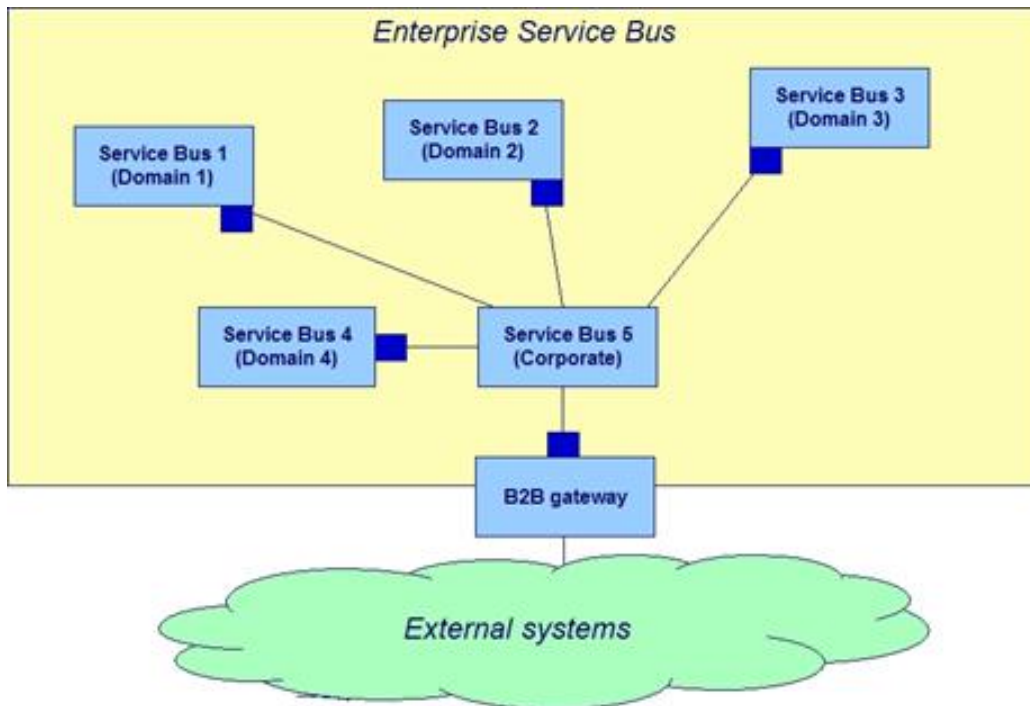


Figure 4: ESB as a composite structure resulting in a federated ESB architecture.

BPEL Considerations

It is also important to study SOA and EDA implementations within the context of Business Process Management

(BPM) and the Business Process Execution Language (BPEL). Current BPEL implementations focus strongly on the command-and-control model - essentially the orchestration and composition of services defined via a procedural syntax that requires a runtime BPEL engine.

The use of BPEL does not cause a problem for SOA because it can establish a parent service composition layer that fully abstracts business process logic. Any one business process is not limited to that layer because BPEL allows a process to be further encapsulated by a service and composed within other business processes.

However, BPEL is not as clean of a fit with EDA. A more suitable model for EDA would be declarative rather than procedural in nature. A model where the designer can simply connect events to publishers and subscribers via a point-and-click mechanism. Runtime implementations would be independent of a controlling engine and based on the earlier mentioned Web services standards. While some platforms are evolving in this direction, currently we need to rely on the use of SOAP over JMS or other SOAP-based alternatives provided by ESB products.

Combining EDA and SOA

The foremost challenge to working with both SOA and EDA is changing the mindset of those responsible for designing the solutions. New EDA-related principles, such as targeted data redundancy and the explicit incorporation of events, often need to be fully understood so that they can be appropriately positioned.

For example, the dynamics behind the publish-and-subscribe pattern and how it exists as an inverse of the request-and-reply pattern should be clear to service designers and architects. This way they can recognize opportunities to combine them into complex message exchange patterns within the same overall architecture. This may not be as easy as it sounds, because asynchronicity is still not a fully accepted design approach for many project teams.

Although publish-and-subscribe is by no means a new pattern, it is new to many who only think in terms of call stacks. Writing records to a batch file or to a database, for example, is similar to publishing events. Reading records from a file is like consuming from a subscription topic. These are all examples of asynchronous design.

Conclusion

Business events provide a powerful means of bridging previously isolated environments and driving service interaction reactively instead of explicitly. What EDA has to offer is most effectively leveraged when the messaging patterns it represents are clearly distinguished. This allows event-driven design approaches to be applied where appropriate, strengthening and streamlining the quality of communication between services.

