Author: Jack van Hoof

The author has extensive practical experience and knowledge on all aspects and roles of application development since 1977. He is currently is employed as Enterprise Integration Architect at Dutch Railways where he advises on modern technologies, standards and patterns to increase IT-maturity with regard to business applications and application infrastructures. Topics are SOA, EDA, ESB and Portal.

Weblog: http://soa-eda.blogspot.com
E-mail: jack.vanhoof.soa.eda@gmail.com

How EDA extends SOA and why it is important

Many people think of SOA as synchronous RPC (mostly over Web services). Others say EDA is SOA. And there are people who say that the best of EDA and SOA is combined in SOA 2.0. But an architectural distinction can be made between a request-and-reply pattern and a publish-and-subscribe pattern. Both patterns are an inverse of each other. Because of the completely different nature and use of the two patterns, it is necessary to be able to distinguish between the both and to name them. You might say making such a distinction is a universal architectural principle. Combining both of the patterns into an increment of the version number of one of them is not a very clever act. It is appropriate and desirable to use the acronyms SOA and EDA to make this distinction, because SOA and EDA are both positioned in the same architectural domain; SOA focusing on (the decomposition of) business functions and EDA focusing on business events. This article explains the differences between the two patterns, when to use the one or the other and how to combine them.

Moving toward on-demand business

Organizations tend to change their structure frequently. The evolving focus on service-orientation and globalization will enforce this trend. 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. Departments and business units are transformed to service providers. These service providers no longer focus only internally on the organization, but they are seeking for external markets to offer their services. 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 IT-systems. This increasing degree of separation creates a need for loose coupling between application components to be able to have the supported business processes bend unimpeded with the continuously changing composition of the organization structure. To achieve this agility the supporting applications must be agnostic to organizational changes like reshuffling responsibilities and roles, outsourcing or insourcing, splitting up departments or the whole company, fusions and all kinds of other reorganizations. Business processes must not be limited by the supporting IT-systems to smoothly follow all of these organizational changes. If, for instance, part of the process will be outsourced to an external partner, a part of the supporting IT-system will be cut off. The remaining part of the system must start communicating with the external partner. The system must not collapse neither is it desirable that adaption to the new situation costs a lot of money or time. The same is applicable in case of changing partners or in case of insourcing external tasks.

SOA, false promise?

This requires loosely coupled application components to be able to easily put the scissors in the organization structure without disturbing the supporting IT-systems. However, the synchronous command-and-control nature of SOA is a way of tightly coupling application components which doesn't allow for this kind of flexibility. SOA may be loosely coupled in the technical domain, where common Web service technologies are used, but it certainly is not in the functional domain where SOA is associated with 'calling' foreign (reusable) services and eliminating data redundancy. The availability of services and stored data can be vanished after an act of outsourcing, which may lead to costly consequences and high risks. This has all to do with creating dependencies with SOA. The promise of SOA delivering loose coupling, which typically is asynchronous, could at the functional level be stated as a false promise.

IT-flexibility versus organization flexibility

Of course the use of SOA has benefits. Agility in application construction by using shareable components in a well defined functional decomposition and using standards based technology can have benefits in time to market with concern to the delivery of the application if adequate tools are used. And also restructuring the application in following business process redesign can be of much less effort than in the 'traditional' way. The IT-department may gain benefits, which will lead to lower IT-costs for the business and faster delivery.

At the moment SOA is positioned in the market mostly as a type command-and-control architecture at the higher granularity levels of functional decomposition. To achieve loose coupling and autonomy in the context of the previous mentioned organizational evolutions EDA (even-driven architecture) is a more appropriate style of architecture at this level of granularity. EDA provides flexibility directly to the organization itself. With EDA an organization can reorganize without affecting application constructions. Changing the structure of the organization without changing the applications is a promise of EDA. So we are talking about quite a different magnitude of agility.

Granularity

But why is SOA promoted at this level of granularity? The causality of four aspects is in charge. Firstly, SOA by most people is thought of in terms of Web services. Secondly, the performance of Web service technologies at the moment is not appropriate at the lower levels of granularity. Thirdly, Web services originate from the request-and-reply pattern, and so are associated with command-and-control solutions. Fourthly, the event-driven model is not well known; people tend to look for solutions in well known domains. Unfortunately the command-and-control pattern is not appropriate at this level. SOA based on synchronous Web services might be a good idea in the middle layers of functional decomposition. Here the command-and-control type of interaction may be required and may be in good balance with Web service performance. But you have to explicitly investigate it during design phase. So finding the correct context where an SOA might be appropriate is not a trivial job.

When to use SOA and when to use EDA?

In contrast to SOA, EDA provides a way of loose coupling. EDA is not a synchronous command-and-control type of pattern, but just the contrary: an asynchronous publish-and-subscribe type of pattern. The publisher is completely unaware of the subscriber and vice versa; components are loosely coupled in the sense that they only share the semantics of the message.

If you are seeking to support strong cohesion in the business processes, situations where all process steps are under one control, SOA is the way to go. The command-and-control style of SOA - in general – is applicable to:

- **Vertical** interaction between the hierarchical layers of functional decomposition
- Functional request-and-reply processes such as man-machine dialogues; the user waits for an answer
- Processes with a transactional nature which require commit and rollback facilities
- Data enrichment in a message to be published to bring the message to its full content in a formal format

If you are seeking to support independency between business process steps, EDA is the way to go. This style of architecture is appropriate in federated and autonomous processing environments. Recognizable situations where EDA might be applicable are:

- **Horizontal** communication between tiers in a process chain
- Workflow type of processes
- Processes that cross recognizable functional organization borders, external (B2B) as well as internal

To find points of decoupling look for parts of the business process of which you are sure they always will stay together in one organizational unit (strong cohesion, atomic business function). In this way a functional composition of the business will arise. The borderlines between the business functions are the points of decoupling. If atomic transactions cross the decoupling borders, then implement compensating rollback transactions at these points.

Striving for loose coupling – and so for flexibility and agility - always is a good idea at all levels of granularity. So a rule of thumb might be: use loose coupling whenever possible and only use command-and-control if required.

All of this with respect to the functional dimension for Both EDA and SOA. Of course these principles always must be challenged with performance aspects like required and feasible response times.

Redundancy: strong design

Loose coupling means independency. Loosely coupled components do not rely on each other. Not even on each others stored data. Each loosely coupled environment maintains its own copy of (a relevant subset of) the data and services. In loosely coupled environments redundancy must not be seen as poor design, but as strong design. Banning redundancy across decoupling borders makes the coupling more tightly. Maintaining redundancy across the decoupling borders makes the loose coupling more robust. EDA is perfectly suited by its nature to support automatic data synchronization mechanisms in redundant environments while maintaining loose coupling.

Visualization

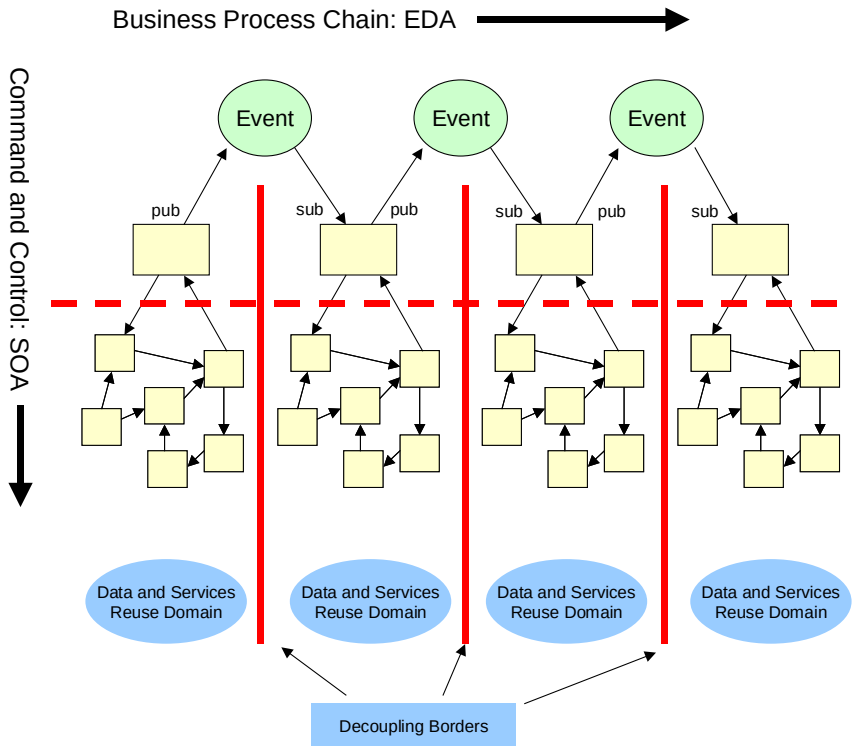

Figure 1: SOA versus EDA

Figure 1 visualizes the relationship between SOA and EDA.

The circles at the top of the picture denote decoupling points (events), the linking pins between loosely coupled systems. At these decoupling points components can be connected and disconnected without any change of the connected systems (peers). All data exchange between the domains takes place only at this point and not at the lower levels.
Within a reuse domain (see figure 1) a finer grained EDA may be implemented. The more fine-grained EDA, the more flexible the IT-systems are, but also to smaller the reuse domains will be.

If using Web service technologies (SOAP) at those points of decoupling combined with a common infrastructure (Enterprise Service Bus), it is possible to easily connect heterogeneous systems. Downstream systems are not only SOAs, but also SOAP-wrapped legacy systems, commercial of the shelf software (COTS), ERP systems and gate-ways to external systems. Figure 2 visualizes an EDA.

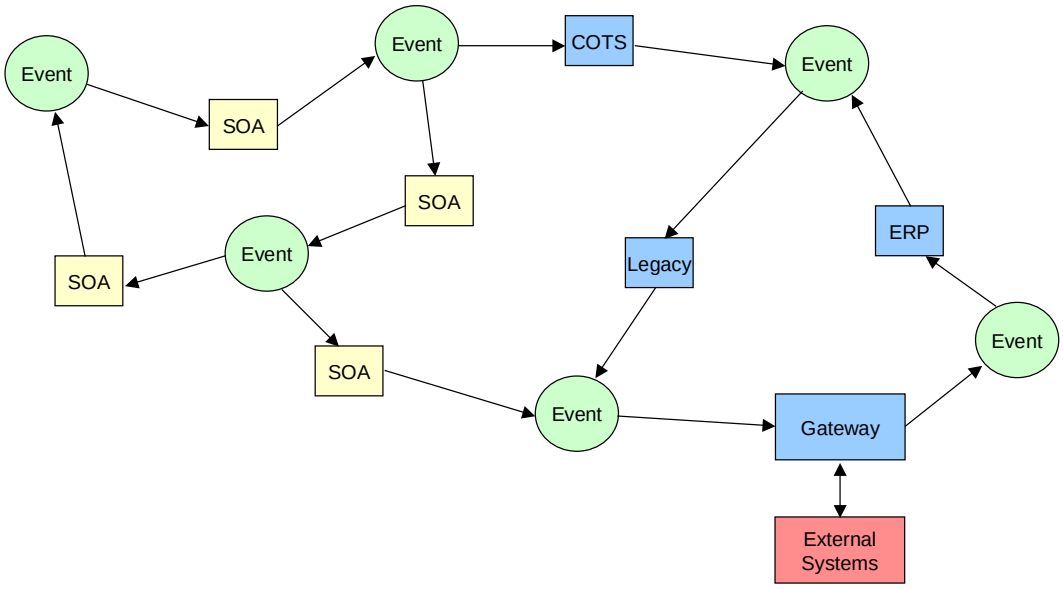

Figure 2: EDA

Components are no longer directly coupled, but connected via decoupling points (events).

Implementation

To implement EDA with Web service technologies today, an additional SOAP-aware message queuing infrastructure is required, whereas this is not the case for a Web service based SOA implementation. SOA in its basic form can be implemented solely by Web services over existing network infrastructures like the HTTP-layer. This is where the current 'SOA-hype' originated from and this also might be the reason why SOA is overwhelming EDA. Current ESB (Enterprise Service Bus) infrastructures provide a way of message queuing combined with Web service technologies. That is why the use of an ESB infrastructure is very appropriate to implement EDA and SOA solutions and to combine both styles of architecture.
Evolving Web service standards like WS-Eventing, WS-Notification, WS-MetadataExchange, WS-ReliableMessaging, WS-Security, WS-Choreography, and lots more, combined with the emerging SOAP-aware infrastructure components like network devices and operating systems will in future provide part of the ESB-functionality which at the moment we have to obtain from ESB-vendors.

SOA and EDA implementations must be regarded in the context of Business Process Management (BPM). Modern BPM-tools are based on BPEL (Business Process Execution Language). The current BPEL implementation focuses strongly on the command-and-control model, the orchestration of services, and so on SOA. Beside orchestration BPEL - to a certain extend - also supports workflow, a kind of choreography, which goes in the direction of EDA. BPEL, however, has a procedural nature and runtime implementations need a controlling BPEL-engine. This is not a problem in case of SOA, but to achieve the aimed loose coupling of EDA it is. Good support for EDA would rather be a declarative model than a procedural model. A model where the designer – simply said – can connect events to publishers and subscribers by a point-and-click mechanism. Runtime implementations should be independent of a controlling engine, but rather be based on the earlier mentioned Web service standards. It is obvious that solutions evolve in this direction. At this moment in time it is appropriate to use SOAP over JMS or other SOAP-based alternatives provided by the ESB. By creating current solutions based on commonly recognized, understood and implemented Web service technologies the systems will be robust in following technological, economic and organizational (r)evolutions in the future.

Global dataspace

An ESB infrastructure is very well suited to function as the container of the business events to be published. This makes the published business events widely available for subscription. The ESB infrastructure behaves as the enterprise's *global dataspace* uniformly accessible by all applications, regardless of location, time and back-end technology. Figure 3 represents a global dataspace, implemented by an ESB.

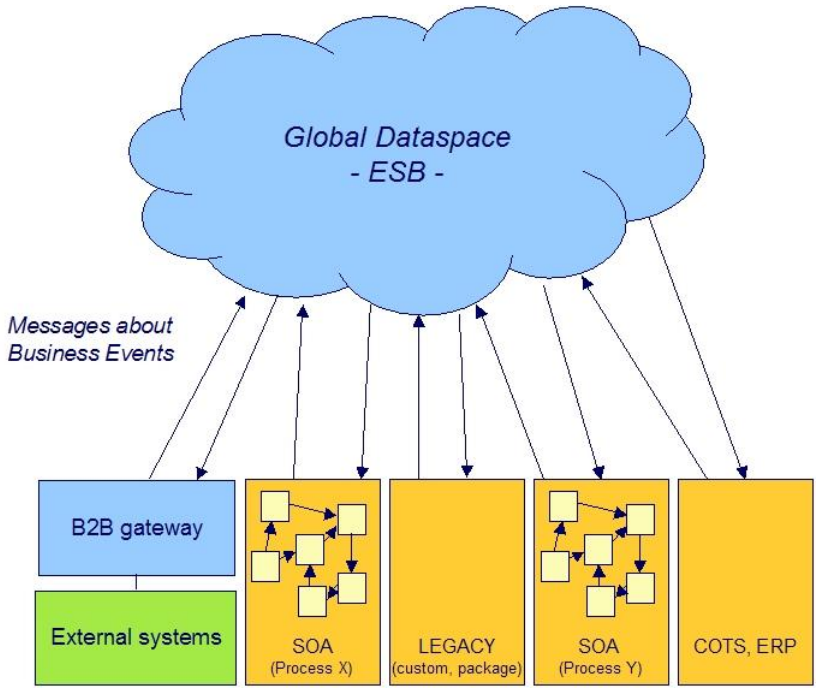

Figure 3: Global dataspace

An ESB needn't be a one-vendor product. As technology homogeneity increases, interoperability increases and the need for product homogeneity decreases. In a federated company multiple service bus products may be in use. This allows for local autonomy. A corporate service bus serves as the global dataspace and supports semantic harmonization for the entire organization. All business events are published in this global dataspace to make them company wide available in a canonical format; of course in a secured way. Figure 4 illustrates this federated ESB architecture.

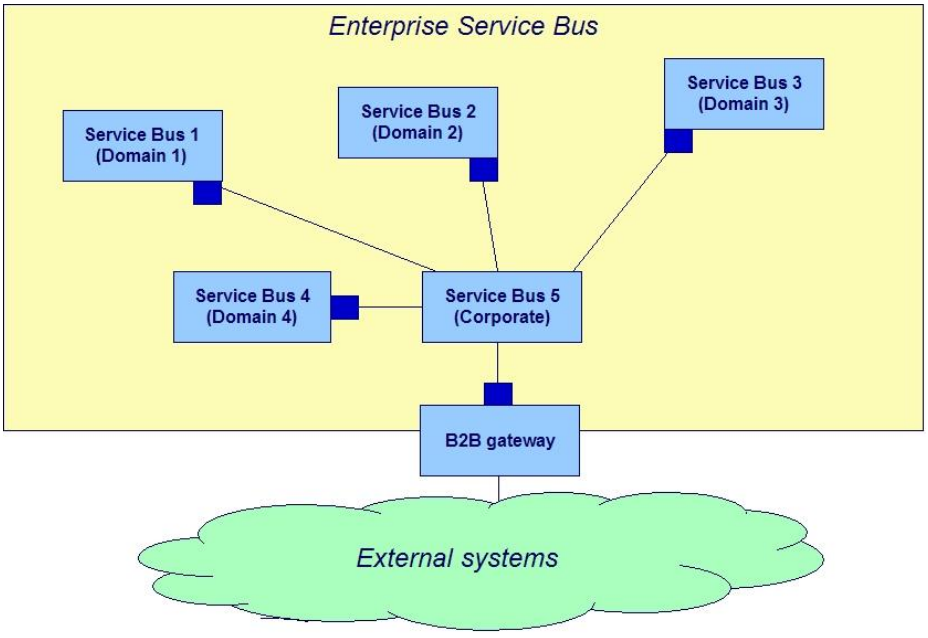

Figure 4: ESB as a composite structure

Using common Web service technologies makes propagation of the messages across multiple vendor products relatively easy to implement. See figure 5.

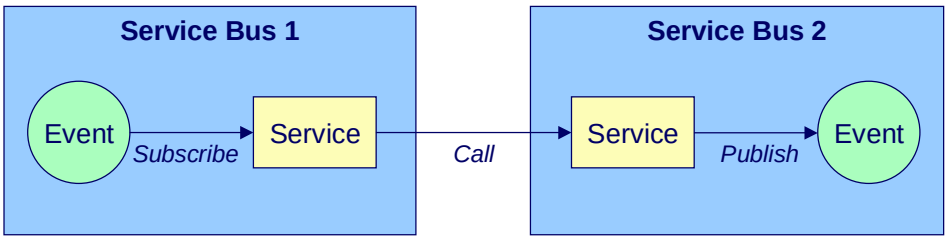

Figure 5: Propagation of events

Getting started

The first thing to do to get started is to change the mindset of your designers and learn them to stop thinking that calling foreign (reusable) services is the only best design pattern. Make them aware of the usefulness of data redundancy as an architectural principle. Have them start thinking event-driven in order to be able to decide when to choose which pattern. Learn them to view the publish-and-subscribe pattern as an inverse of request-and-reply pattern. And learn them to recognize that both patterns can be combined smoothly into one single overall architecture. It won't be easy, because the asynchronous mindset is not the common way of thinking for most of our developers. Publish-and-subscribe is not a new pattern, however it is a new world for the many who only think in terms of call stacks; just read the publications and blogs on the subject of SOA to recognize this limited perspective of the authors. Young developers may learn from the older developers who used to design job flows coded in JCL (Job Control Language), which was mainstream two or three decades ago. Writing records to a batch file or to a database is similar to publishing events. Reading records from a file is like consuming from a subscription topic. This is asynchronous design.

As mentioned before, the SOA and EDA pattern principles always must be challenged with practical feasibility when applied to specific situations, as always must be with architectural principles. Most important is to distinguish between the two patterns and think over subtle and skillful what is most appropriate in a specific context. Be aware of being able to choose and know very well what pitfalls to avoid. If you don't feel confident, start with simple projects.

The following steps may guide you to create an initial roadmap for innovation of your application landscape:

1. Model your business into functions at the granularity level of the desired autonomy
2. Outline your application landscape
3. Map the application landscape to the business function model
4. Applications that cross functional borders are potential bottlenecks
    - Plan to create loose coupling (if not already present)
    - Assign high priority to applications that cross external organization borders

These steps may be your starting point in moving your current application landscape toward a modern, flexible and adaptable application environment leveraging your legacy systems as well as your newly build SOAs.