



12 Steps towards SOA

David S. Linthicum

While many may understand the notion of SOA by now, very few have any idea how to get there. Truth-be-told there is no hard and fast rule as to how one builds an SOA in their organization. Clearly, SOA is a situational thing and your mileage may vary.

However, some common patterns are emerging which may assist you in understanding how to implement SOA. These patterns may also provide a step-by-step guide toward implementing your SOA, either in the fast track (revolutionary) or the slow track (evolutionary). As long as you're on the right track.

In an attempt to supply some practical information here, I would like to provide you with 12 steps you should consider when planning for and implementing an SOA...a plan of attack, if you will, that provides you with a sure-fire way to get your SOA up-and-running.

The steps are:

1. Understand your business objectives and define success.
2. Define your problem domain.
3. Understand all application semantics in your domain.
4. Understand all services available in your domain.
5. Understand all information sources and sinks available in your domain.
6. Understand all processes in your domain.
7. Identify and catalog all interfaces outside of the domain you must

leverage (services and simple information).

8. Define new services and information bound to those services.
9. Define new processes, as well as services and information bound to those processes.
10. Select your technology set.
11. Deploy SOA technology.
12. Test and evaluate.

The Drill Down

Understand your business objectives, and define success. Let's face it; we're running a business, and the technology you layer into that business should add value. In other words, make for a better bottom line. Thus, it's very important to define these objectives up front, including what defines success. Pretty scary stuff for technologists, but absolutely essential if you're to create an SOA that benefits a business. If you find this difficult to define, perhaps you need to evaluate the need.

This process is a basic requirements-gathering problem. It requires interfacing with paper, people, and systems to determine the information that will allow the application integration problem to be defined correctly so that it can be analyzed, modeled, and refined. Only then can the appropriate solution set be employed.

Keep in mind there are both hard and soft benefits from implementing any type of technology. Hard benefits include immediate cost savings, such as automating a sales order system that used to be manual. Soft benefits

are a bit more difficult to define, such as customer satisfaction which leads to additional sales.

Define your problem domain. You can't boil the ocean, thus you must define the scope of your SOA, within an enterprise. Most SOAs are best implemented in small steps, such as moving a single division, or portion of a division, to SOA, if needed, instead of an entire enterprise all at once. Small successes lead to larger more strategic successes over time, and you need to establish the demarcation lines at the beginning of the project to provide better focus and understanding.

Understand all application semantics in your domain. You can't deal with information you don't understand, including information bound to behavior (services). Thus, it is extremely important for you to identify all application semantics—metadata, if you will—that exist in your domain, thus allowing you to properly deal with that data.

The understanding of application semantics establish the way and form in which the particular application refers to properties of the business process. For example, the very same customer number for one application may have a completely different value and meaning in another application. Understanding the semantics of an application guarantees that there will be no contradictory information when the application is integrated with other applications at the information or service levels. Achieving consistent application semantics requires an application integration “Rosetta Stone” and, as such, represents one of the major challenges to creating your SOA.

Defining application semantics is a tough job since many of the existing systems you'll be dealing with are older, proprietary, or perhaps both. The first step in identifying and locating semantics is to create a list of candidate systems. This list will make it possible to determine which data repositories exist in support of those candidate systems.

Any technology that can reverse-engineer existing physical and logical database schemas will prove helpful in identifying data within the problem domains. However, while the schema and database model may give insight into the structure of the database or databases, they

cannot determine how that information is used within the context of the application or service, that's why we need the next several steps.

Understand all services available in your domain. Service interfaces are quirky. They differ greatly from application to application, custom or proprietary. What's more, many interfaces, despite what the application vendors or developers may claim, are not really service interfaces at all, and you need to know the difference. Services provide behavior as well as information, thus they are service-oriented. There are some services that just produce information; those are information-oriented and should not be included in this step. We are only interested in the former at this point.

It is important to devote time to validating assumptions about services, including:

- Where they exist.
- The purpose of the service.
- Information bound to the service.
- Dependencies (e.g., if it's a composite service).
- Security issues.

The best place to begin with service is with the creation of a services directory. As with other directories, this is a repository for gathered information about available services, along with the documentation for each service, including what it does, information passed to a service, information coming from a service, etc.. This directory is used--along with the now-understood application semantics--to define the points of integration within all systems in the domain.

Understand all information sources and sinks available in your domain. Next, it's important to define those interfaces that just deal with simple information. They can either do two things: Consume information (sink) or produce information (source).

You need to understand a few things here, such as:

- Where they exist.

- The structure of the information flowing in and out.
- Integrity constraints.
- Dependencies (other sources and sinks, perhaps services in some instances).
- Security issues.

Understand all processes in your domain. You need to define and list all business processes that exist within your domain, either automated or not. This is important because, now that we know which services and information sources and sinks are available, we must define higher level mechanisms for interaction, including all high-level, mid-level, and low level processes. In many instance, these processes have yet to become automated or are only partially automated.

For example, if an application integration architect needs to understand all the processes that exist within an inventory application, he or she will either read the documentation or the code to determine which processes are present. Then, the architect will enter the business processes into the catalog and determine the purpose of the process, who owns it, what exactly it does, and the technology it employs (e.g., Java or C++). These processes are later bound to new processes—or, meta processes--providing orchestration of encapsulated processes or services to meet some business need.

You should also consider the notion of shared versus private processes. Some processes are private, and thus not shared with outside entities (or, in some cases, they are not even shared with other parts of the organization). Other processes are shared, meaning that others leverage the same processes in order to automate things inter-enterprise. Private and shared processes can exist in the same process space with the process integration technology managing security among the users.

Other information may be maintained in the catalog; information that may include variables used within the processes, object schemas, security requirements, and/or performance characteristics. Each process catalog must maintain its own set of properties, custom-built

for each specific application integration problem domain.

Identify and catalog all interfaces outside of the domain you must leverage (services and simple information). In today's real time world, automation typically does not stop at the corporate firewall. You need to identify all of the outside interfaces that systems in your problem domains interact with, or, should interact with, to provide the maximum value. These can range from EDI interactions to modern Web services connections. We need to define these interfaces with the same amount of detail as defined above.

What's important here is to make sure that all desired interfaces are also defined, including the ability to expose services outside of your problem domain to partners, as well as your ability to see and leverage their services. Their systems and your systems should work together to support common shared processes.

Define new services, composite services, and information bound to those services. This is self explanatory. You must define all new services that are to make up your SOA, these will fall into one of three categories.

First are services exposed out of existing systems, or, **legacy services**, such as ERP, CRM, legacy, etc.. These types of services really are defined for you, since the services are one-to-one representations of internal application functions or interfaces exposed as Web services (typically), to facilitate integration. You should note that we are calling these 'new services,' even though many are preexisting, because now they are accessible by your SOA, exposing true services and not proprietary interfaces. Some of these may be established through a simple upgrade of an enterprise application (ERP, CRM, ERP, etc.) to a service-oriented version (e.g., exposing existing behaviors as Web services).

The second type of services are **composite services**, which are services unto themselves that are made up of many different services. In many instances, these services are mere interfaces to many other services and don't add much if any additional functionality. These are complex services, since there are so many dependencies as well as information bound to composite services that you must understand before creating your SOA.

Finally, **scratch built services** are services that are built from the ground up to be a true service. These are typically newer applications and are more under your control and useful since you're building them with an SOA in mind, and thus designing them to provide services.

Define new processes, as well as services and information bound to those processes. At this point we should understand most of what is needed to define new processes, as well as bind them to existing processes, and automate processes previously not automated. New processes should be defined that automate the interactions of services as well as information flows to automate a particular business event or sets of events.

While you can define some very complex logic within new processes using today's tools, the theme here is to orchestrate existing services and information flows rather than create new functionality. In essence, it's a meta-application that sits on top of many smaller applications, defining interactions from lower to higher levels.

Select your technology set. Many technologies are available, including application servers, distributed objects, and integration servers. The choice of technology will likely be a mix of products and vendors that, together, meet the needs of your SOA. It is very rare for a single vendor to be able to solve all problems--not that that reality has ever kept vendors from making the claim that they can.

Technology selection is a difficult process which requires a great deal of time and effort. Creating the criteria for technology and products, understanding available solutions, and then matching the criteria to those products is hardly a piece of cake. To be successful, this "marriage" of criteria and products often requires a pilot project to prove that the technology will work. The time it takes to select the right technologies

could be as long as the actual development of the SOA. While this might seem daunting, consider the alternative--picking the wrong technology for the problem domain. A bad choice practically ensures the failure of your SOA.

Deploy SOA technology. This is the "just do it" step, meaning that you've understood everything that needs to be understood, defined new services and processes, selected the proper technology set, and now it's time to build the thing.

Test and evaluate. To insure proper testing, a test plan will have to be put in place. While a detailed discussion of a test plan is beyond the scope of this article, it is really just a step-by-step procedure detailing how the SOA will be tested when completed. A test plan is particularly important because of the difficulty in testing an SOA solution. Most source and target systems are business-critical and therefore cannot be taken offline. As a result, testing these systems can be a bit tricky.

Steps to Success

One thing to understand is that I have not (and could not) define everything you must do to create a successful SOA project. My goal has been to outline most of the activities necessary for your SOA project, in some case you'll need to add some steps to accommodate your particular needs. I'm not sure anyone should ever delete steps.

Unlike traditional application development, where the database and application are designed, SOAs are as unique as snowflakes. When all is said and done, no two will be alike. However, as time goes on, common patterns emerge that allow us to share best practices when creating an SOA. We still need to travel farther down the road before we can see the entire picture. But we're getting there.

