



A Primer for Service-Oriented Architecture (SOA)

June 2006

Progress.
OpenEdge

A Primer for Service-Oriented Architecture (SOA)

By Niel Powers, Vice President of Products for Progress OpenEdge Division

In the technology industry, it is all too common to define and describe a new methodology or approach by starting with its technical aspects. Invariably, this leads to deep descriptions of acronyms, standards, and technical attributes, which in turn leads to everything except understanding.

So let's try a different approach. Let's discuss Service Oriented Architecture (SOA) from the *business* angle. Let's look at the changing *business* requirements that result in changing *IT* requirements that result in new technologies, methodologies, and standards. This "outside-in" approach just might make it possible for everyone involved to reach a common understanding of SOA, regardless of their role in the business.

At its core, SOA is simply a response to a growing challenge: business requirements are changing at an accelerating rate, and those changes simply cannot be accommodated with traditional IT and application structures. Look inside a business today, any business, and you'll discover a pace of change that leaves most organizations in a constant state of flux. Each day, they are outsourcing processes to other providers, taking on processes that are new to the business, rearranging and re-defining traditional processes, and sharing information with a collection of stakeholders in a fashion unimaginable just a few years ago. Businesses today require a new level of agility, both in process definitions and in how various processes interact with each other. That agility, and its application to business processes, is at the heart of SOA.

Lessons from History: Mass Production & Interchangeable Parts

In the first half of the 19th century, the world of firearms was changed substantially by Eli Whitney and Samuel Colt. Their innovations were many, but among the most important was their specification that weapons must be manufactured to use interchangeable parts. It may seem obvious and normal to us now in this post-industrial age, but at the time it was quite a change. This specification recognized the value of each part, the value of the interaction of the parts, and value of the whole that it formed. Of course, it also allowed for easy field repair, model upgrades, and new model configurations. Believe it or not, this is the essence of SOA. If, somehow, software could be built and implemented as a collection of interchangeable, interoperable parts, then IT structures could be reconfigured, upgraded, and repurposed with the agility demanded by the business community.

To better understand how this concept applies to software and IT infrastructures, we need to break the problem down into two parts: the processes that will make up our interchangeable parts (or components), and the communication systems that will allow them to interoperate appropriately. Here's where it starts to get a little technical. Let's start with the components.

Business Applications = Opinions on Best Practices Expressed as Code

All applications (at least those aimed at business) are designed to take some business process and automate it. Or, as a friend of mine once put it, "Business applications are nothing more than someone's idea of how a business should run expressed as code." Simple enough, but the problem is that most applications are actually made up of any number of business processes, all expressed as code, and all mixed together in such a manner as to make the processes (and the software) inseparable. This results in the *opposite* of agility: an application that has rigid process definitions, rigid interactions between those processes, and pre-defined restrictions on how those processes can be implemented. SOA proposes to replace this structure with one in which business processes are clearly defined and implemented independently as a collection of services.

Roughly, each service corresponds to a business process with a well-defined and understood set of functionality and boundaries. This is the *service* part of SOA. Perhaps an example would help:

A sales order application actually represents a collection of automated business processes. These include processes to evaluate customer credit, check inventory availability, determine appropriate pricing and discounting, calculate taxes and shipping charges, arrange for provisioning, shipping, and billing, and prepare appropriate business ledger entries. But as constructed, most applications do not implement these processes as a collection of interchangeable parts. So a new customer credit process cannot be easily substituted, an outsourced service cannot easily be used to better calculate taxes, and the inventory availability capability cannot easily be reused by some other application.

SOA Doesn't Change the Opinion, Just the Way It's Delivered; BUT SOA Does Allow Businesses to Quickly Change Opinions in Rapidly Changing World

SOA proposes to replace this application with one that is designed, built, and implemented where each process is a service with all of the flexibility and agility that is missing in current implementations. The *service* may do the same thing that it did before, but it is implemented in a very new fashion. In this new world, an application consists of one or more services, each representing a distinct business process but all working together to solve business problems and automate business requirements.

Interactive Standards Are a Prerequisite for Complete SOA Vision

Of course, none of this works if services can't interoperate – if they can't talk to each other. Nobody would want a system where one part provided the credit information, another provided the inventory availability, and a third calculated taxes – but none of them worked together. Obviously, a communications system is needed, one that is robust, standardized, easily implemented, and flexible enough to keep up with evolving needs. So if the processes become the services, then the communication system needs to become the architecture that makes them all work together.

It is here where SOA takes advantage of many of the standards that have occurred over the last few years in the world of communications. While SOA is not as standardized as one might think, there are certain communication standards and assumptions that are shared in most SOA implementations. Internet standards and protocols are considered the standard for communication infrastructures, and XML forms the backbone of SOA standards such as SOAP and WSDL.

But communications is tricky. Think about all of the elements of simple human-to-human communications. Consideration must be given to language, lexicon, vocabulary, conversation topics, medium, and protocol. The same is true for service-to-service communications, but with greater precision and more pre-determination. SOA addresses some of these issues through standards such as XML, SOAP, WSDL, and UDDI, but other standards and protocols will need to be established at industry and application levels. To continue the example:

In our new services-based order processing system, the credit service is now separate from the pricing service, which is separate from the inventory availability service and all of the other processes. But now there must be a set of standards upon which all can agree for effective interoperability. What information is required for the credit service to do its job? What questions can be asked of it and what responses can be expected? Do all of the services have a common understanding of an inventory part number, and does the same part number mean the same thing to all of the services? What is the protocol to be followed if the order process asks the pricing and discounting service for an answer and it doesn't reply in a set time? If the business ledger service returns an error, how should the process (and the other services) react? All of these questions must be worked out ahead of time for these services to interact together.

The Two Essential Elements of SOA

So here we have the two critical elements of the SOA:

- 1) Application components expressed as a collection of well-defined services, and
- 2) A communications infrastructure (built upon standards) that supports the interoperability of all of those services into a single, cohesive unit.

Of course, all of that is theoretical, and we all know that theory and reality are often miles apart. So it is worth discussing a few common myths about SOA, and its cousins Service Oriented Business Applications (SOBA) and Service Oriented Development of Applications (SODA).

Advantages of SOA

SOA also brings the following real-world benefits to IT and application development:

- Incremental development and deployment of business software (*i.e. you can change an opinion about how to run a small piece of your business, without having to dump and replace an entire application*)
- Reuse of business components in multiple business experiences
- Low-cost assembly of some new business processes
- Clarity of application definition and structure
- Agility in the ability to upgrade components individually, without having to overhaul the whole application
- Greater flexibility in arranging components into workflows [Example: If one business wants to check credit before inventory, and another the opposite order, SOA allows each enterprise to organize this type of workflow.]

SOA does not necessarily bring these mistakenly attributed benefits:

- Simple software engineering [Example: Sometimes it is easier and faster to create a custom piece of code to solve an immediate problem, but this piece of code ends up incompatible with other services. Although thinking things through all scenarios outside of the immediate situation takes time, it does save time in the long run.]
- Free integration or interoperability [There usually will be a cost associated with services, either via subscription, per transaction or sold outright.]
- Automatic technology independence and/or vendor independence [Platform independence cannot be assumed; planning still counts.]
- The ultimate architecture for the modern enterprise [SOA may not be the ultimate or most efficient solution for every problem. Prioritization is critical to cost justify SOA, especially in SMBs.]
- Fully defined standards for business process interpretabilities [Americans all speak English, but that does not mean we can all hold intelligent conversations about quantum physics. Specialized topics require specialized conversations, vocabulary and – of course – acronyms.]

As is the usual case, SOA describes the practices and methodologies of a perfect world. Of course, none of us live in a perfect world, and neither do our IT environments. So we are faced with the time honored question, "That's great, but how do we get there from here?" How do we take whatever current applications, integration methods, communication systems, and IT infrastructures we have in an organization, and re-purpose them to SOA? This is where the first mistake is often made: SOA is not a technology, a product, or even a set of standards. You can't buy a SOA, either in kit or finished form.

To dust off and re-purpose an old saying, it might be better to think of SOA as a journey, not a destination. In other words, it is a methodology and a set of best practices that you can implement over time and as needed to achieve the benefits described above. And, as usual, there are a number of steps you can take to determine if and how SOA can benefit your business or business application:

1. Spend some time learning and understanding SOA methodology, benefits, practices, and standards. Most of the standards are technology based, but their structures help you to understand how to implement some of the ideas of SOA.
2. Take an inventory of the business processes that compose the scope of your enterprise/business. Note which processes are set in stone, which are currently in flux, which are most likely to be in flux, and which areas of the business require the most immediate attention. Pay close attention to processes that are likely to be subject to out-sourcing to outside services or supply-chain participants.
3. Take a similar inventory of applications and other IT assets, and cross-reference these to the business processes from the previous steps. Which applications automate which processes, either in whole or in part? Which applications have the most flexible architectures, and which the most rigid? Large, far-ranging applications will represent the greatest challenge; smaller, single-purposed applications may be easier targets.
4. Assess the current communications infrastructure for the business. Organizations are often surprised to discover the low reliability and/or scalability of their systems once SOA principles are put into practice. Seriously consider commercial communication infrastructures designed specifically for SOA support.
5. Make a plan that matches the business requirements to the implementation priorities. This may seem obvious, but IT departments tend to concentrate on the technologically issues that are most interesting to them, instead of making "groovy" or preferred technology secondary to what must be the highest priority of any enterprise/business -- to align technology with the business processes that require the most agility and interoperability.

About now, the astute reader will be thinking, "Wait a minute, this is all sounding a bit familiar." Of course it is. The entire history of software has been a process of further specialization and componentization. The evolution from monolithic to services is entirely predictable and expected, but the issue of interoperability and communications has always been a problem. We tried CORBA, RMI, COM and DCOM – is this just another **trend** designed for some future dustbin? Anything is possible, but the SOA direction is a good evolutionary step. It is definitely a plus in that its initial focus is the business process, not the technology. It is built on a set of practices, models, and standards that are understood, flexible, and well accepted.

Most importantly, it is a methodology that opens more doors than it shuts, allowing both agility and flexibility. Perhaps this is the factor that has led to wide industry acceptance and adoption, even if some of the technical details are not as explicit as earlier efforts. In fact, Gartner, Inc. says that by 2008, more than 75 percent of then-current application packages either will be natively SOA or will expose SOA interfaces through a wrapping layer of interfaces ¹.

So where does this leave us? Hopefully, it leaves us in a much better spot over the next few years. We can get back to a point where our IT environments and applications can deliver the agility, functionality, and interoperability that our business owners expect and deserve. But nothing is ever gained for free. Communication structures will need examination and appropriate upgrades. Additional standards will need to be defined and agreed upon. And applications, purchased or built in-house, will need to evolve into more componentized structures to release and expose the business processes that will comprise the services of the future. This is the promise of SOA, and it can, over time, become the reality of SOA.

¹ Gartner article dated 16 April 2003 by Yefim V. Natis "Service-Oriented Architecture Scenario".