SOA Worst Practices, Volume I

How to prevent your SOA from being DOA

PROGRESS SOFTWARE

Actional

sonic SOFTWARE

Table of Contents

For a Successful SOA, Just Do It! (Huh?)

How many times have you heard: "Security for your service-oriented architecture [SOA] should be tight enough to be secure, but not so rigid as to impact flexibility or performance"? Did you think: Hey, how do we measure this?

You're not alone—and we're not just saying that to make you feel better. These and other SOA best practices offer good advice and value, but applying them to your business can be tricky. After all, best practices don't always relate to your specific situation or business model.

At Actional we think that it's time for a new approach.

Your Best Idea Might Actually Be Your Worst SOA Practice

For more than six years, Actional has been at the forefront of SOA technology, working with early adopters and partnering with industry leaders. In that time, we've seen it all—and proudly penned a few of our own best practices.

We also realized that knowing what *not* to do when building an SOA can be just as valuable. That's why we created *SOA Worst Practices*, Volume I.

In this guide, you'll learn about real-life (and real lame) mistakes made by your peers.[1] We'll share with you the thinking behind these ideas, why they failed, and better approaches that can help you achieve a more successful outcome. (In other words, you will be smarter after reading this. Really.)

Perhaps you are considering adopting an SOA. Or maybe you are already far down the path of your first. Either way, you will want to keep this guide handy. There's bound to be a worst practice here that you (or a "friend") were tempted to try.

If Actional can save even one developer from the SOA worst practices hall of shame, then writing this guide was well worth the effort.

[1] Although we changed the names of all companies (except for ours) mentioned in these worst practices, their stories are true and painful.

The 8 Worst SOA Practices

We had a lot to choose from but narrowed it down to the top 8 worst practices to date. Something tells us that this is just the tip of the iceberg…

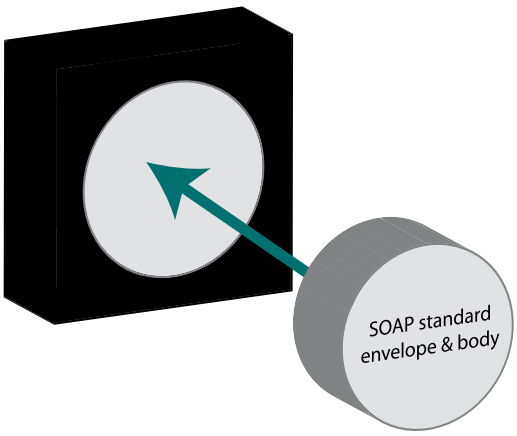
8. I Know That It's a Standard SOAP Stack. What's Wrong with a Little Change?

Performance is a concern for any IT environment, including Web services. Web service messages pass through multiple intermediaries, and each message must be processed in its entirety (parsing the whole message), all of which can put the brakes on message speed. But, as this worst practice illustrates, tinkering with a standard will only add to your problems.

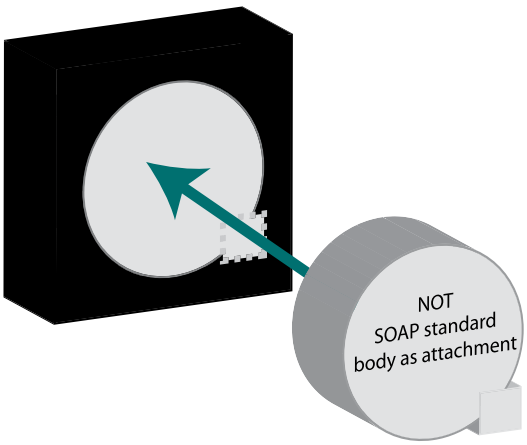
The Idea

Seeking to speed up its Web service messages, the IT team at Hutchings Healthcare decided to split SOAP messages into two parts: the envelope and the body. The envelope would be passed in the usual manner, but the body would always be passed as an attachment, instead of abiding by the SOAP standard and being passed in the message.

Standard SOAP



Standard SOAP used in unorthodox manner



Why It Wasn't So Smart

The IT team was using a standard SOAP stack, but in an unorthodox way. As a result, the messages and applications that used this stack were incompatible with other standards-based implementations. Although the IT team claimed that this was a standards-based SOA, they really had created a proprietary form of messaging.

The IT team mistakenly assumed that it could have an effective SOA while using mandatory proprietary "extensions." Here's why this cannot work:

1. Such extensions inhibit interoperability, which then drastically limits the scope of your SOA effort and dramatically increases the cost.
2. Due to the proprietary nature of this messaging, specific code must reside on each end of the wire. You therefore need to customize all of your applications and server software.

3. The extensions prevent you from efficiently using your developer tools, which are optimized around standard SOAP stacks.
4. Versioning is more difficult. When ISVs update their software, the messaging technology will likely have to be updated for the revised platform.

SOAP is an open language, so by modifying it, Hutchings Healthcare was removing its value. If SOAP was not working for its SOA, the company would have benefited more by having the standard updated than by creating a proprietary solution.

A Better Approach

Performance boils down to how you architect your SOA. A true SOA allows you to use open standards to facilitate reuse and flexibility. As a result, you are not locked in to a specific vendor or technology. Even the most heterogeneous systems can communicate quickly and easily across different software and hardware platforms. This all changes, however, once you introduce a proprietary solution, as Hutchings Healthcare did when it modified SOAP.

When designing your SOA, avoid the telephone game—that is, too many components and services passing the message and adding load. Instead, make sure that the components are loosely coupled and use a universal language. That way, they can avoid steps, say A to D, and go straight to step E to obtain the requested data or initiate the correct process. You then can simply flow the information you need.

7. Get Instant SOA—Grab Your Existing Apps and Wrap 'Em Up!

An SOA doesn't have to be complicated; however, it's important to have a clear SOA strategy and to understand the security risks and ways to mitigate them. Or, you could be looking at a recipe for disaster.

The Idea

Granger Machinery, a manufacturer of heavy and industrial machinery, thought that it had a great idea for starting an SOA. The company planned to use Web services in the form of a "wrapper" to make its existing inventory and customer database available to other lines of business. The SOA was initiated by allowing a client to send complete SQL statements in the request to the database. However, this protocol left the relational database susceptible to SQL injection attacks, whereby attackers bypass the SQL statements defined on a Web server in order to inject their own statements.

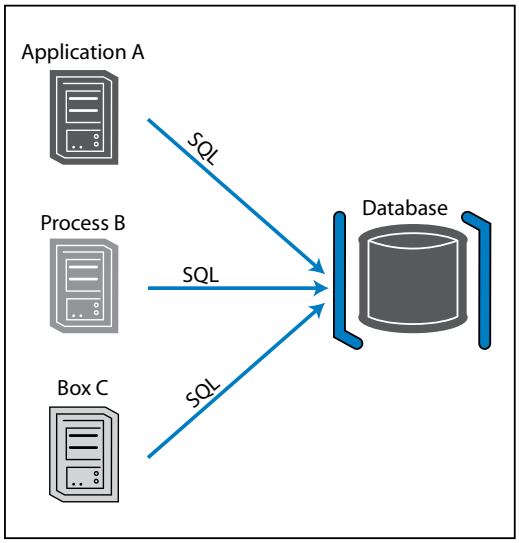
The IT team thought that it could reduce this risk by using an anti-injection attack feature in its existing security product. This feature would detect and sanitize SQL statements that were written to perform destructive operations, such as "Drop" or "Delete."

Why It Wasn't So Smart

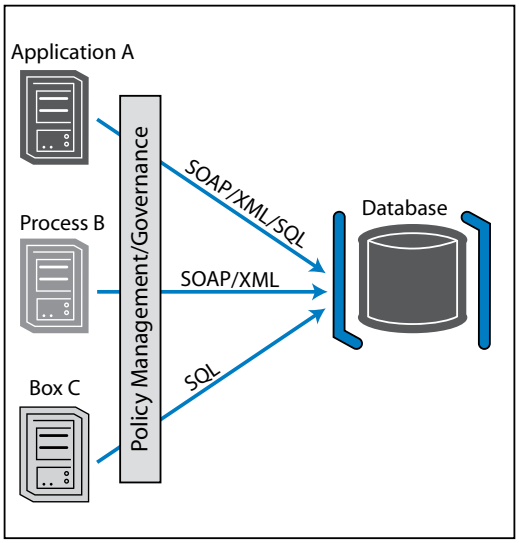
By itself, this Web service approach of putting a wrapper on an existing application or database is not dangerous. However, if you think about this case, it is actually a tightly coupled approach and requires that the requestor know the implementation details to make the SQL call. This approach suffers from two major problems:

1. It lacks business context. The IT team needs to associate a business context—or rules—around the Web services, instead of allowing "dumb" SQL requests. Such rules define who has a right to put in a request to the SQL database and in what situation they can access data. Individuals would then need to be authenticated before accessing the database. This would prevent the database from blindly giving out information, as well as prevent destructive requests from directly accessing the table.

2. The SQL injection problems are caused by two different issues: bad data that gets injected into SQL statements, and bad SQL statements themselves. The injection protection capabilities described earlier cannot solve these problems.



When communicating directly to the database, applications and processes have too much authority.



By offloading policy management and governance, rules and policies are easier to manage and update.

A Better Approach

By applying SOA and Web service standards, you can quickly achieve reuse and integration goals. However, this approach needs to be part of an overall SOA strategy, or you risk experiencing problems at the application level.

In the case of Granger Machinery, the company needed to address the following in its SOA strategy:

- Who should have access to the data?
- How should they access it?
- Can and should we prioritize the requests and transactions?

Bottom line? Wrapping a service is an excellent way to get reuse out of applications that are already delivering value to your business. It is therefore a good way to build an SOA, so long as you have a business strategy to dictate which services to wrap and the rules that govern their use.[2] Randomly wrapping services, however, can lead to security and performance problems—inside and outside the organization.

Example: Problems at the Application Level

If different services are allowed to talk to each other, an unauthorized individual can pull information or use more of the service functionality than you had intended. Imagine that Granger Machinery had a human resources portal and suddenly different lines of business wanted to use that information, even though they were not authorized to do so. Such a free-for-all could put the company at risk of violating security and privacy laws, as well as slow down performance.

Also, remember that building an SOA is an incremental process. You can start by wrapping services that are working well today and rebuild others over time. In addition, you can prioritize what to rebuild and what to build from scratch based on the new business opportunities that come your way.

[2] Governance should ensure that such issues are accounted for and built into the strategy. Policies then will perform the control, with management and run-time governance ensuring these polices are enforced and measured.

6. Hackers Can't Harm Us, We Have an XML Firewall!

Interoperability is key to communicating with your partners' and clients' systems. It's therefore important to remember that building an SOA can greatly impact these groups. As this worst practice demonstrates, sometimes the best of intentions can backfire on a business—and ignorance is not bliss in the SOA world.

The Idea

Dane Foods prided itself on delivering quality customer service, in addition to quality baked and frozen goods. Its customer service department was a pure Oracle shop, from backend databases, to Web and Oracle Forms front ends. The system had its quirks, but had been working well for years.

The customer service department was self-contained, except for limited interactions with external customers and partners to gather complaints. Complaints were accepted in virtually any format, since the department didn't want formatting requirements to prevent it from obtaining valuable data. No other business process in the department depended on interactions with external partners.

The CIO of Dane Foods decided that the company needed an SOA. He did not know a lot about SOA, but he had heard people talking about it. And it sounded like a good idea.

As a next step, the IT team intended to purchase an XML firewall to protect two services that would perform schema validation on dozens of transactions per second. The CIO then created an RFP, which listed improved performance and improved UI as the main goals of the SOA.

After installing the XML firewall, the IT team encountered communication problems, which frustrated customers and partners. They then had to turn off several firewall features, as they worked to resolve the problems.

Why It Wasn't So Smart

Dane Foods' idea failed for three reasons:

1. The RFP confused performance and UI requirements of the firewall with the objectives of the SOA. Not only are these requirements not typical SOA objectives, but they also are not considered to be strengths of firewalls.
2. The department needed to secure the interoperability it had with external users.
3. The IT team permitted different data schemas from various sources, which created issues for the IT department and later forced partners and customers to change their schema. This generated unexpected work and costs for those organizations.

A Better Approach

First, make sure that you are building an SOA for the right reasons. These reasons should be based on your business as well as your IT objectives.

Second, devise a strategy for your SOA, plus a way to communicate your SOA plans to partners, customers, and anyone else that it impacts.

Third, be aware that interoperability is a requirement of an SOA, but it does not happen magically. Web services often have numerous suppliers and consumers. Before you patch or update a service, you will want to know all the interdependencies—the consumers downstream and the providers upstream—as well as the potential business impact.[3]

[3] You can automatically detect and discover these with Actional solutions.

5. Extreme Makeover SOA Edition: Let's Rip and Replace Everything!

Consolidating systems can be a good way for companies to improve efficiency and cut costs, particularly in the case of acquisitions. But, as this worst practice proves, migrating and integrating systems into an SOA should not be done overnight, or all at once. And you should never select a consulting firm without first doing your homework.

The Idea

Carson Office Supplies was growing at a phenomenal rate, acquiring 20 previously independent franchises in less than a year. Soon the company noticed a significant amount of overlap among its IT systems and business processes. As a result, it decided to integrate some of these systems by building an SOA. Carson Office Supplies dubbed this effort Project TROY.

The company then hired Wecan Consulting, which devised a comprehensive plan, involving a team of 160 to 200 people (i.e., burn rate like a dot-com startup), and software from 30 different vendors, including:

- Plumtree Corporate Portal
- BEA WebLogic
- RSA ClearTrust
- IONA Artix
- Oracle Application Server
- Sunflower Assets
- Adobe Form Server (now called Adobe LifeCycle Forms)
- Oracle eBusiness Suite
- ILOG JRules
- Software AG Tamino XML Server
- Network Inference Cerebra Server
- Informatica PowerCenter and SuperGlue
- Microsoft SharePoint Server
- IBM Rational Suite
- Actional Looking Glass and SOAPstation

Wecan Consulting fell consistently behind schedule, failing to deliver on its promised milestones. After massive team cutbacks, re-planning attempts, and scaling back the requirements, Wecan Consulting had to start over and completely redesign the SOA.

Wecan Consulting's new plan was to go completely COTS (commercial off-the-shelf) with point-to-point integrations. By this time, however, morale was low among Carson Office Supplies' IT team. After such a botched start, the IT team had little interest in pursuing an SOA and was dragging its heels. As a result, Carson Office Supplies had to wait and experience more pain before realizing any benefits of an SOA.

Why It Wasn't So Smart

SOA gives you the opportunity to use a best-of-breed approach, as well as leverage open standards. So, mixing and matching vendors that use open standards should work well. However, Project TROY was trying to do too much, too soon.

Although Project TROY did many things correctly, the complexity of the project was overwhelming—from the scope of the project to the number of vendors involved. Carson Office Supplies needed a more simplified, incremental approach.

To compound matters, Wecan Consulting, while a noteworthy firm, did not have much SOA experience and was learning along the way.

Finally, Carson Office Supplies was duped into forgoing a true SOA in favor of a point- to-point model, after being told that the point-to-point model would deliver the same benefits. The company ended up with a tight coupling and a second-rate integration solution, which is not the point of an SOA. Although this alternate strategy enabled all the systems to "talk," it did not deliver the same level (and therefore value) of reuse and flexibility as an SOA.

A Better Approach

Fortunately, SOA allows you to avoid a "big bang" or "rip and replace" approach. When thinking about an SOA, remember to ask yourself:

- What is working?
- What is not working?
- What is delivering value?
- What new business opportunity should we be pursuing?

Your SOA plan should cover migrating your entire system to an SOA, but remember that this process can take years. It's better to start instead with applications that are not working. Migrating or rebuilding these applications will deliver immediate benefits.

In addition, avoid rebuilding applications that are working and delivering value. You can incorporate their value in your SOA immediately by simply putting a wrapper around them. This can be achieved by using an adapter or broker.

Moving forward, let your business needs dictate when and how to build new applications or new features.

Bottom line? Try to avoid any effort that does not add business value, such as redesigning or redeveloping an application that is already working well. Instead, make sure that your SOA strategy focuses on leveraging the business processes that deliver the most benefit to the company. Then look at the underlying components and applications for each business process. If some of the applications work well, just put a wrapper around them. If they are broken, or if you want to enhance the process, then that is a compelling reason to create a new service to provide that functionality.

Finally, do your homework when choosing a consultancy. Just because a firm is large and well known, does not mean that it has all the key SOA experience you require.

4. Forget About Cloning Humans, Clone Your Apps!

Cloning and reconfiguring existing applications might sound like a good way to save on maintenance and development costs. But in practice it is far from a perfect science.

The Idea

The IT team at Jesper Travel, an online travel service, had a novel approach to systems management. It created new applications and new business processes by reconfiguring and cloning existing applications. One application, for example, had 30 "separate" instances in production, all customized to a certain degree with 50% to 60% of the application core residing in each instance.

The IT team thought that this approach would reduce development time, make maintenance easier, and reduce problem resolution time. They reasoned that:

- More developers would be familiar with a larger percentage of applications.
- The architecture was simple: the two points being the cloned application and the customer.
- It would be easy to plan and compute future costs, since a new customer = new hardware + cloned application + a few modifications specific to that customer.
- A new customer would be a self-contained "business model," with all the costs associated with it and no shared infrastructure to depreciate.

Why It Wasn't So Smart

This approach was flawed in four major ways. First, if there was a bug in the core code, it was replicated. Second, some of the core code was altered during the customization process, but these changes were rarely documented. So, when the triage team needed to determine why something was not working, it spent far more time doing detective work than actually fixing the problem.

Third, the result of this cloning was redundant infrastructure, maintenance, and development—not only at a business area level, but also at the IT level. The IT team had no way to share capacity between customers, lower the TCO, or manage SLAs. (But the hardware and software vendors were quite happy with this model, as every new customer for Jesper Travel meant additional hardware and software purchases.)

Example: Embedding vs. Abstracting

Instead of abstracting one set of security rules that applied to all customers, such as "all customer data must be encrypted," Jesper Travel had this rule embedded in each application. This greatly complicated systems management. Abstracting would have allowed Jesper Travel to change the rules simply by changing the configuration of the core infrastructure functionality—instead of the code of each application.

Fourth, core code changes impacted every customer who relied on these services. Following every change, the core and related customized modules had to be tested before the company could redeploy these services. In addition, the IT team could not simply patch applications, but had to go through more steps, all of which translated into a longer release cycle. Plus, this approach embedded rules, or policies, as code into the application—rather than abstracting them. This added to the level of redundancy and made change even more difficult.

A Better Approach

If you want to save on maintenance and development costs, focus on reuse. Unlike cloning and reconfiguring, reuse enables growth within your business, while minimizing complexity within your IT systems.

By building an SOA, you can attack redundancy at the application and infrastructure levels. For instance, you can distill from hundreds of duplicated applications a small set of core application feature sets. These reusable services can be used globally, across applications and business processes, resulting in greater operating efficiency and reduced maintenance costs. In addition, you can standardize, break out (abstract), and control the core infrastructure functionality of key business processes, including security, identity, profile, message queuing and brokering, translation, directory service, etc.

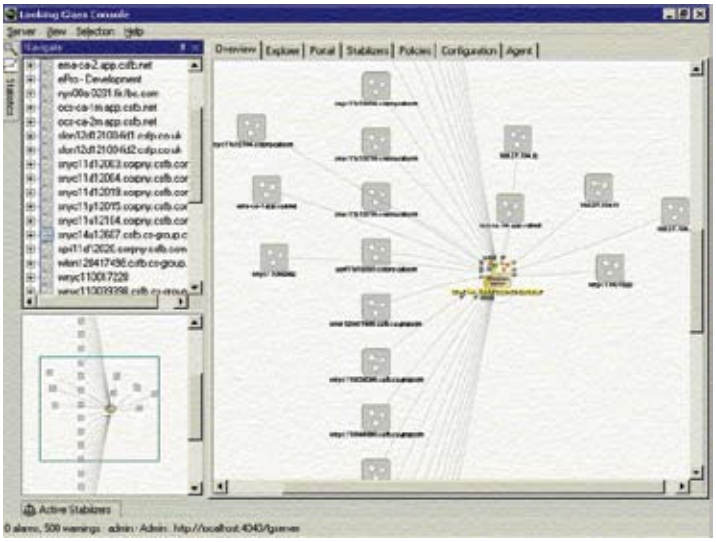
With an SOA, you also can construct new business solutions far faster than by cloning applications. As a result, you can spend more time customizing solutions for each customer, while still leveraging your core set of services. These smaller services actually can help reduce your hardware and software costs as well, since they allow your company to better utilize what it already has. In addition, you can measure, manage, and secure these services or change them easily with policy changes.

3. We Even Let Grandma Use Our WSDL

Reusability is a key benefit of SOA, but context must dictate how services can and should be reused. As this worst practice illustrates, security, privacy, and compliance issues can surface when the use of a Web services description language (WSDL) is not governed.

The Idea

Venton Automotive, an OEM of sunroofs, had provided its human resources (HR) team with a great new tool. The company had designed its internal HR portal so that employees could enter and manage their own information, including skill sets, phone numbers and locations, managers' names, etc. Likewise, HR validated this information and manually removed records for any individuals who were no longer with the company.



The portal functioned largely as a trust-based system, performing only minor data validation, such as validating reporting structure, titles, etc.

Soon the portal was the only place where up-to-date and complete employee profile information could be found. In fact, it was the only place where HR could locate a current organizational chart. Over time, other departments discovered the portal—and began to use it. The audit team, for example, used the information to ensure that everyone in a group had been trained on compliance and privacy issues. And the company telephone operator was able to ensure calls were routed to the appropriate individuals.

When other developers requested access to the HR portal information, the IT team shared the WSDL used to define the service's formats and protocols. The IT team was comfortable sharing the WSDL, for it reasoned that the point of an SOA was to have a reusable service. The main developer shared the WSDL with three or four people, who then shared it with other developers. After a few months, no one was certain how many people were using the service.

Why It Wasn't So Smart

The IT team soon learned that someone had put the WSDL into a library and then shared the library with all the development teams. They soon learned that the Web service had more than 30 consumers.[4]

The IT team had no idea that so many users and lines of business were being supported. They also didn't know:

- Were the right departments being supported?
- Was the data protected or encrypted?
- What was the capacity of the service?
- If it failed, what would be the ramifications?
- What if the IT team needed to version the service?

To make matters worse, people were using the library in production, so the IT team had a development server supporting production applications. This arrangement posed serious security risks and jeopardized adherence to compliance and privacy laws.

In sum, IT management was exposing its systems to substantial risk, while missing the opportunity to show measurable value of an SOA, as demonstrated by reuse.

A Better Approach

To achieve secure reuse of services via an SOA, you want to have a detailed process in place. Such a process must:

- Identify the consumers and providers that depend on the service
- Ensure that the right security measures are in place, IT policy is enforced, and business rules are applied
- Set up management and runtime governance technology
- Validate that the service is running as designed and meeting the business objectives for which it was designed
- Include a warning mechanism and discovery capability, so you can know immediately when production operations go awry or something slips by the process—such as the inappropriate sharing of a service or WSDL

Bottom line? Sharing a WSDL may seem like an innocent act, but, if not controlled, countless individuals can gain access to a valuable service, putting your data at risk.

[4] Actional's technology, which automatically discovers all services in use, revealed the number of consumers of this Web service.

2. Hackers Will Never Touch Our Data, We Have External Schema Validation!

Securing your systems against hackers is a constant battle. Invoking external schema validation may sound like added protection; however, this approach can actually leave you more vulnerable to attacks.

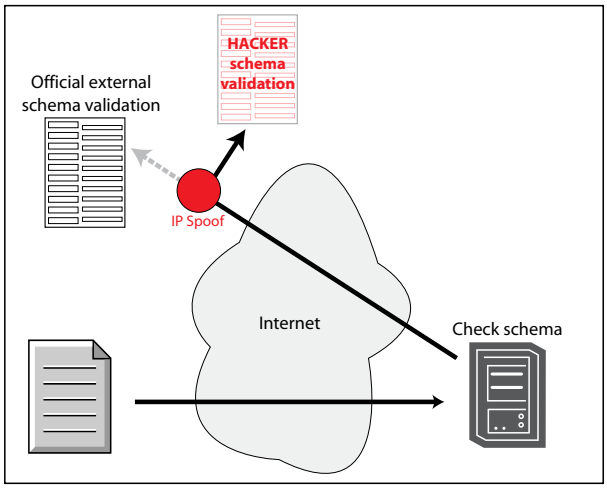
The Idea

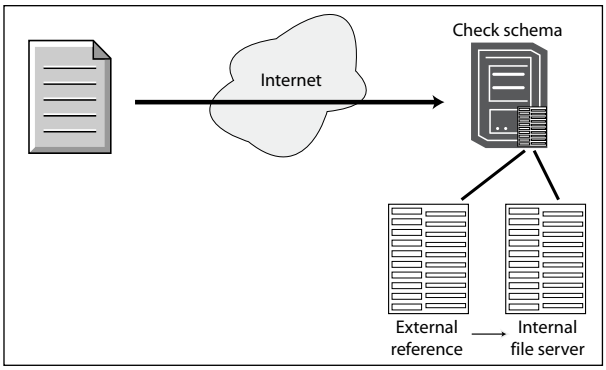
Academic Business Services (ABS), a chain of schools specializing in academic degrees for working adults, wanted to increase security for its Web services. It decided to invoke external schema validation. This setting allowed the schema "check" to reside in an alternate location, thus reducing the risk that hackers would be able to decipher the schema using carefully constructed "probing" transactions.

Why It Wasn't So Smart

Often the location of the external validation can be deciphered from the transaction response. A hacker quickly discovered that he could spoof ABS' IP and change the location of the company's schema check. He then initiated transactions with an alternate schema and had it verified by the faux schema validation at the spoofed IP address. This caused legitimate transactions to fail.

So, although hackers could not obtain important data from ABS, they still succeeded in impacting its business.





A Better Approach

You can safeguard your schema by following a few simple steps. First, give the schema to your partners, since you can directly contact them. Then put schema validation on an internal table that is secured by both traditional perimeter defenses and end-point security.

1. Our SOA Initiative Is a Success—Just Look at the Number of Services We Have!

In this era of ROI, most companies want metrics to demonstrate that their SOA initiative is progressing and delivering value. So, how do you accurately quantify SOA value? Hint: Don't simply count the number of Web services.

The Idea

Rimstar Bank's executive board had requested a report on the value of its SOA initiative. The bank's CTO, however, was unsure how to quantify this value. Ultimately, he decided to base this figure on the total number of services that the IT team had developed and that were in production. The company had 25 services in production, with five being reused for multiple business processes.

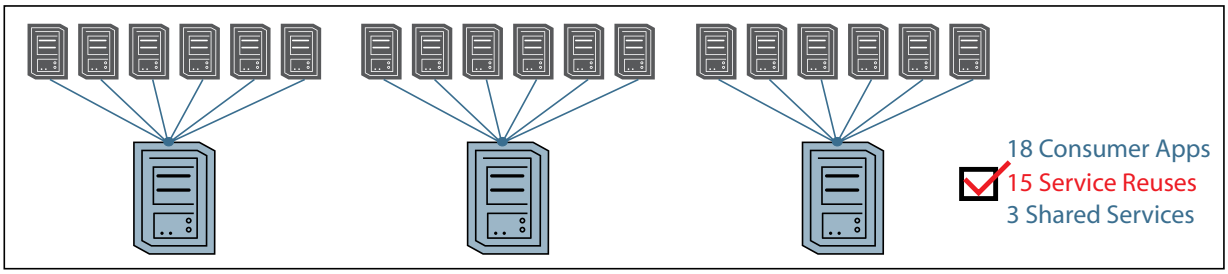
Why It Wasn't So Smart

The primary business value provided by an SOA is reuse. The number of Web services therefore has nothing to do with the success of the SOA initiative. In fact, a high number of services can be a leading indicator that the initiative is failing, since it's likely that reuse is not happening.

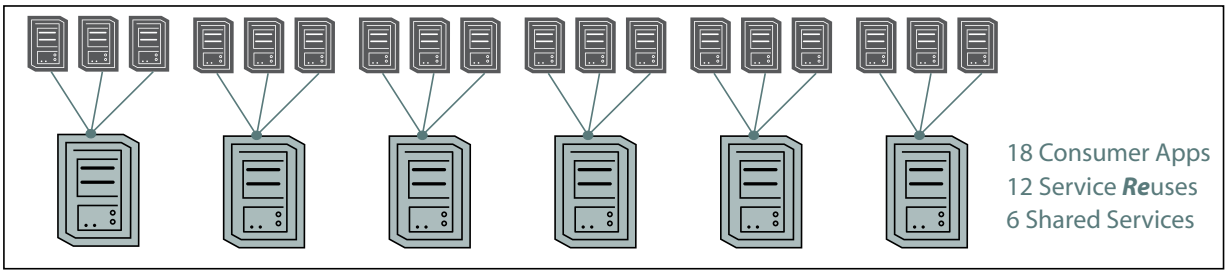
At Rimstar Bank reuse was low, with 20 of the 25 services in production still being used as individual applications. Thus, existing applications being converted to services, or newly created services, were not the right ones to tally for an ROI report. After all, they were not being reused by any other applications and processes, nor were they being reused by other lines of business.

Quantifying SOA - Which Initiative is more successful?

#1



#2



A Better Approach

When calculating the value of an SOA, it's best to focus on reuse among existing Web services, rather than the number of newly developed Web services. Reuse is not only a key benefit of SOA, but also something that you can quantify.

You can measure how many times a service is being used and how many processes it is supporting, thus the number of items being reused. This enables you to measure the value of the service. With a little work, you can calculate the cost savings for each instance of reuse, including saved architecture and design time, saved development time, and saved testing time.

Moreover, reuse means fewer services to maintain and triage. So reuse generates savings, and frequency of use drives value.

Parting Thoughts

Building an SOA can be challenging, even agonizing. Sometimes ideas that look good on paper are not so good in practice. Many actually turn out to be worst practices. But, as this guide illustrates, you can learn a lot from others' mistakes, including:

- Why reuse, not the number of Web services, is the true indicator of SOA value
- Why standard SOAP stacks are key to performance
- When to build an SOA and the business benefits
- How to build an SOA by adding a wrapper to existing applications
- Why an XML firewall an SOA does not make
- Why it's important to understand the purpose and true benefits of an SOA before moving forward
- What key business rules an SOA strategy should address
- Why SOA does not require a "rip and replace" approach
- Why cloning and reconfiguring applications is not the way to go
- Why you should be careful about sharing your WSDL
- Why your partners can be your best allies when it comes to securing your schema validation

More important, you can gain the wisdom needed to make better decisions as you embark on an SOA initiative.

For More Information...

To learn more about building an SOA, contact the experts at Actional at 800-808-2271 (US Only) or +1 650-210-0700 (Outside US) or send an email to sales@actional.com.