SOA: Challenges and Solutions

Overview The benefits of SOA (Service-Oriented Architecture) have been well documented, but the challenges associated with an enterprise wide SOA deployment have not. While the distributed nature of a SOA encourages reuse and provides a high level of agility for the business, it can also give rise to real challenges in the delivery of SOA-based applications.

XML is the core technology enabler of SOA-based applications. Its verbose nature, inherent lack of security, and the increase in connections between applications and services required result in a number of challenges that are well-understood, and that have proven solutions.

Challenge The challenge in delivering SOA-based applications lies in identifying where potential problems will arise and addressing them as early in the deployment cycle as possible. Application delivery controllers are well suited to addressing both the well-understood and unanticipated issues associated with delivering SOA-based applications. Incorporating an application delivery controller (ADC) into the deployment plans for your SOA can prevent unnecessary delays and even the need to re-architect portions of your SOA infrastructure—saving man hours, expense, and headaches.

Solution SOA is both an architectural design pattern and a deployment methodology. That's a nice way of saying that no one can define what any particular SOA implementation should look like. While there have been attempts by various vendors in a number of vertical SOA markets to create best practices for SOA environments, these have largely failed due to the rather dynamic and organizational-specific nature of SOA implementations. Because the services that comprise an SOA are business-focused and encapsulate business specific entities, there is no single agreed upon definition of what must exist and how components are deployed in order to bear the title SOA.

There are, however, a set of guiding principles underlying SOA that can be considered a framework around which discussions on common SOA attributes and challenges can be based. The distributed nature of services, for example, is a fundamental attribute of all SOA implementations and as such challenges associated with that deployment model can be applied to all SOA environments, regardless of their actual design or implementation.

1. An SOA is comprised of distributed business services and achieves business value through the reuse of those services.
2. An SOA is based upon industry accepted open standards.
3. An SOA reduces time to market through loose coupling of service interfaces and its underlying implementation, enabling agility.
4. An SOA is heterogeneous and applications comprise n-tiers that may vary widely based on the business process activity around which its composite services are built.

These common attributes result in a common set of challenges that all organizations face and that must be addressed at some point in the implementation process. SOA guiding principles suggest that consideration be given to the architecture of the deployment environment and infrastructure upon which services will be deployed and delivered before said services are put into production, in parallel with service definition and development efforts.

Fortunately, most of the challenges associated with deploying SOA are common to all web-based application deployments. The difference between SOA-based applications and traditional applications is that these challenges are often faced by implementers earlier in the application lifecycle due to the challenges associated with its core technology enabler, XML.

SOA Basics

In most cases, a SOA will be implemented using a variety of standards, the most common being HTTP, WSDL (Web Services Definition Language), SOAP (Simple Object Access Protocol), and

XML (eXtensible Markup Language).These latter three standards work together to deliver messages between services much in the same way as the post office.

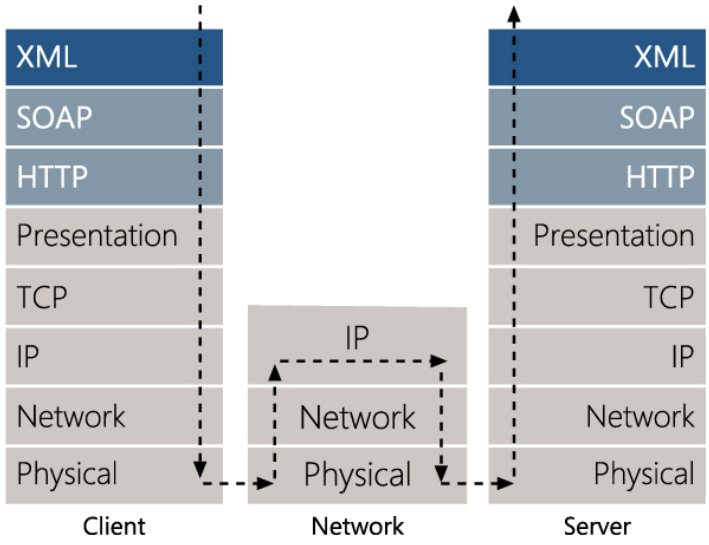
WSDL	→	Entry in an address book
HTTP	→	Postal carrier (transportation)
SOAP	→	Envelope (encapsulation)
XML	→	Letter (message)

If you're sending a letter then it is assumed you already know where to send it. In a SOA that information comes from the WSDL, which lists the transportation options (HTTP, FTP, SMTP) available, the possible addressees (remote functions available), and the format requirements for data sent to each addressee (XML).

The same way an envelope has an address, SOAP carries with it information about where the message should be delivered and who should open it. In the case of SOAP and SOA, the addressee is not a *who*, but a *what*, as the addressee is really the name of a remote function that should process the message inside. Just like a "real" envelope, SOAP has specific formatting requirements and failure to properly place information in the right place can cause delivery or processing issues.

With this information you can build a message based on the format requirements (XML-based), put it in an envelope address to the receiver (SOAP), and then drop it in the mailbox (HTTP) and wait for a response.

If you are the receiver of this message, the message must be delivered to you, you have to read (parse) the envelope and determine the correct addressee (remote function). The message then must be passed to the appropriate addressee (remote function) for processing. Once the correct addressee has the letter, they can then read (parse) the message and process it according to their designated function.



You'll note that this process requires a lot of reading (parsing) and formatting of data, from the actual data you want to transfer to the envelope to the transport layer. It also has the effect of adding two "layers" to the stack that must be processed. This is one of the challenges associated with SOA—the computational overhead associated with parsing and processing XML. Current analysis of the impact of XML parsing and processing on server utilization estimates that approximately 30% of a server's resources are consumed simply by the parsing and processing of XML. As a reference point, SSL processing was also estimated to consume 30% of a server's resources. Unlike SSL, XML currently has no effective server-based solution to alleviate this burden such as hardware

assisted processing, thus the issue of increasing performance of XML-based services and applications must necessarily lie outside the server and in the network.

Obviously a single SOAP message does not an SOA make. An SOA can be thought of as a distributed organization that makes use of the postal service (network) to deliver messages that assign tasks (remote functions) to individual business units (services).

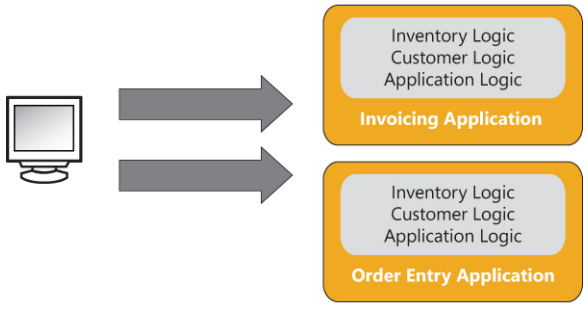
There is also nothing that disallows services from being built upon other technologies. REST (Representational State Transfer) is often used as an example of an alternative technology upon which an SOA can be built. REST services do not use SOAP, choosing instead to use the HTTP URI to specify the addressee of the message and putting the message right into the HTTP body. If SOAP is analogous to a letter, then REST messages are most like a postcard where the message is not encapsulated at all.

An SOA can mix and match both technologies, as well as others, though the few existing best practices documents and SOA consultants are quick to nudge—and even push—organizations toward standardizing upon one or the other. As many of the benefits of an SOA including interoperability and platform independence rely on an open-standards foundation, this push is not necessarily a bad one.

Connection Management

Architecting the distributed services upon which an SOA is built involves the identification of common business entities and their related functions that can be encapsulated in a service. A company's inventory, for example, is a common business entity that is likely shared across all business units. Querying and updating that inventory are two common functions performed by applications upon the inventory. Similarly, a customer entity likely exists that is common to all applications within an organization, with a common set of functions that can be performed on that customer such as querying, updating, or creating.

In the past these common functions and entities were often duplicated across all applications needing to apply application-specific logic to them. This necessarily meant that any changes to those entities resulted in the need to modify every application which used those entities. This is an inefficient model which leaves open the very real possibility that differences between applications will occur in the handling of these entities. These differences are replicated into the data stores, causing problems between disparate applications needing access to that data to perform specific tasks. The duplication of function across multiple applications also has a deleterious effect on the ability of an organization to affect rapid change due to the time and effort involved in deploying a change to any shared entity.



Traditional Application Design

SOA resolves these issues by removing common entities and their associated functions and establishing them as an atomic entity of their own (a service) that is shared by all applications.

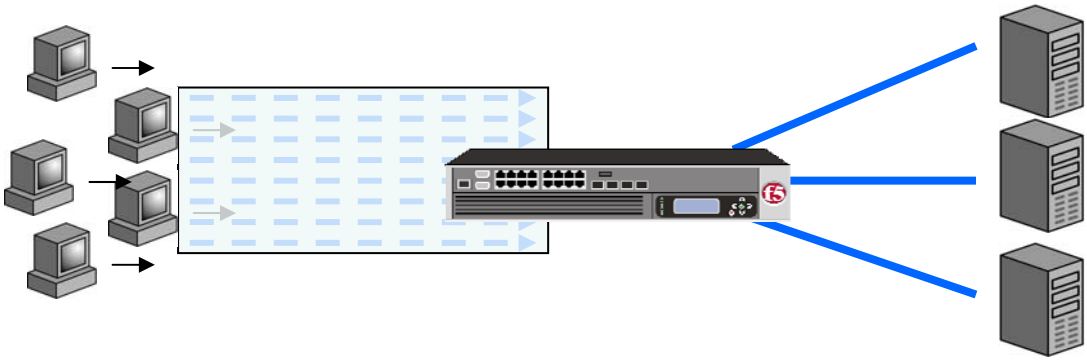
Changes to the logic or data structure associated with that entity are now reduced to a single code-set, and assures consistency across the enterprise.

SOA Application Design

While an SOA certainly improves consistency of application logic and reduces the time required to introduce a change to business entities or their underlying data structure, you will note that it has introduced complexity in terms of the number of connections required.

This increase in connections required can have a negative impact on performance if services are physically distributed across the network. The overhead associated with TCP connection management is applied directly to the computational cost of the application, usually to the detriment the application's overall performance. Also note that each intermediary (i.e., device or application that touches an XML message) incurs both TCP session and XML parsing performance penalties. As the number of services associated with a single application increase, so also does the total application response time increase.

There are essentially two ways in which ADCs can address this challenge. The first is based on accepted and proven connection management technologies. By allowing an ADC to manage the myriad interconnects between services, it can offload the burden of managing TCP sessions on servers and increase their overall capacity.

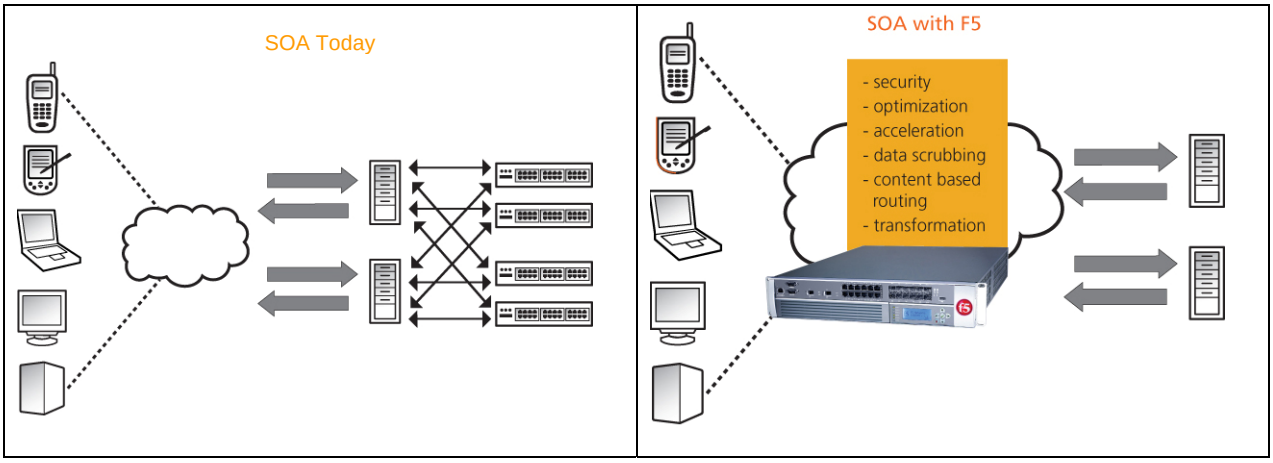


Connection management technology reduces the burden of TCP on servers

The second way in which ADCs address the challenges of distributed services is by providing a network platform on which shared network services can be hosted.

Many of these functions—security, data scrubbing and transformation, authentication, and caching are application specific, and therefore network-focused point solutions can't host these shared services. They can only be supported by an ADC or on an application platform such as those offered

by BEA, IBM, Oracle, and Microsoft.



By moving applicable network-focused shared services to an ADC, it reduces the complexity of the SOA by consolidating services into a single network-hosted platform without sacrificing performance. Hosting shared network services on an ADC also has the added benefit of alleviating unnecessary parsing of XML from the services in your SOA. By validating the credentials of an incoming message before it reaches the service, the service only parses those messages which are authorized to invoke its operations. This principle applies equally as well to XML and web-application threat defense measures. Rather than impede performance by duplicating scans for common threats such as SQL injection, overly large message sizes, cookie tampering, and XML-specific attacks, it makes sense to provide a single, shared service through which the message is initially validated and verified as being free of malicious content.

By verifying the message at the point of ingress, it alleviates the need for each service to perform that same verification and thus reduces the processing burden on their respective servers and increasing overall capacity.

Scalability

The high computational cost of parsing and processing XML reduces the overall capacity of already burdened application servers. The issue with XML arises from its text-based formatting, which must be parsed and converted into objects that the application server can understand and manipulate. This process is called marshalling, and the performance of this process is heavily dependent upon the complexity of the data. Each element in the data must be created and many functions called to assign values to its attributes. In most environments this process is accomplished via serialization, which is treated as a form of I/O within the system. I/O in general is resource-intensive, requiring a lot of memory and CPU cycles on the server regardless of platform.

This process occurs twice; once when the message is received to transform the XML into a machine readable format, and again when the response is transformed back into XML to be returned.

Because of the drain on resources caused by this process, and any additional processing required such as connections and queries to databases, application servers on which services are deployed are rapidly driven to high CPU utilization states and a dwindling pool of available memory. This reduces the capability of the application server to process requests in a timely fashion and introduces delay as the system tries to judiciously process each request by sharing its resources amongst them.

The effect of heavy load upon application servers is a well-understood problem, with existing solutions such as ADCs assisting in the horizontal scaling of these servers to distribute requests.

Scaling application servers in an SOA environment becomes important earlier in the deployment process because of the additional burden placed on application servers by the parsing and processing of XML.

It is not only the application servers in an SOA environment that are in need of scalability services. Many organizations have turned to XML gateway appliances such as those from Reactivity and IBM DataPower to offload the compute heavy XML processing and to provide additional XML-specific functionality such as WS-Security, schema validation, virtualization of services, and transformation via XSLT (eXtensible Stylesheet Language Transformation). These devices are capable of performing these functions on XML-based messages much more efficiently than an application server, but lack many of the enterprise-class features required of a network device such as advanced load balancing algorithms, proven failover methods, and session management.

In order to scale these devices an ADC is still a necessity. Care should be taken when considering when the issue of load balancing and XML appliances arises. While these devices are capable of load balancing services, their implementations are rudimentary and inflexible in their deployment options. These devices do not provide the advanced health-checking capabilities of ADCs, and can therefore fail to properly route traffic as they have no awareness of the state of the application as is the case with an ADC.

The traditional round-robin or least-connection algorithms used by both XML appliances and application server clustering capabilities fail to recognize the resource intensive nature of XML and are therefore incapable of truly distributing requests within a pool or cluster of servers in the most efficient manner. The capability of an application server to handle the load of SOA messages is dependent not only on the number of requests it is currently attempting to handle, but on the resources currently available. This means that advanced algorithms and capabilities are required to determine the current state of the application and the server, which is one of the domains of expertise of ADCs. A server may have only one connection or be “next” in line to receive a request, but if it is currently parsing and processing a very large XML message it may not have the memory or CPU cycles to process another message and still meet SLA goals for both messages.

Security

XML is text-based and therefore human-readable. It is also web-based and hosted on the same application infrastructure as “traditional” web-based applications, making it subject to traditional web attacks.

Many of the same vulnerabilities that have plagued web-based applications are applicable to XML-based applications. SQL injection is a common method of attempting to extract unauthorized information from corporate databases or to wreak general havoc by deleting important data. SQL injection can be as easily accomplished through an XML message as it is a traditional HTTP GET or POST formatted message.

Because this type of attack is well-understood by existing solutions designed to protect web-applications, these same solutions provide a measure of security for XML-based applications. So, too, can other security techniques such as signature scanning mechanisms that seek to discover hidden viruses within web-borne messages.

While it is always preferred that developers produce secure code, the reality is that if they responded in code to every threat that they would spend most of their time coding protection against new attacks, testing those solutions and then deploying them, only to start again the next day. Additionally, such security-in-code techniques require duplication of code throughout an application, which can negatively impact performance and increases the possibility of other errors being introduced into the code. The duplication of code is an anathema to the guiding principles of SOA, which seeks to reduce the costs and time to market for software solutions through the identification of such shared services.

By employing an application firewall to mitigate the risks associated with SOA and distributed service environments, the principles of an SOA are upheld while protecting applications from these types of well-understood threats. Moving as much application security to the point of ingress has the added benefit of improving performance, as messages that are clearly malicious never reach the application server, thus reducing the burden on application infrastructure.

Bandwidth

A concern of many regarding XML is the increased size of messages flowing between systems. Indeed, the SOAP envelope used by web services to carry the XML messages itself adds a minimum of 256 bytes to every message, though usually the increase is much higher.

The verbose nature of XML means that even the smallest of messages will become approximately twice as large as the same message formatting as HTTP POST/GET name-value pairs. When these messages are being transferred between servers in the data center this increase oft has a negligible effect on bandwidth and transfer times, owing to the fat nature of data center backbones. But with increased use of Asynchronous JavaScript and XML (AJAX) as the basis for user-interfaces that interact with SOA-based back end systems, it is important to consider the effect of these increasingly large messages on delivery times to the client.

In addition to the potential use of AJAX clients to access SOA services, there is the very real possibility that external partners and customers may be accessing services via the public Internet, over an uncontrollable and often unpredictable public network. In any case in which clients are accessing services via a public network there is the possibility that a disparity exists between the client's ability to receive responses as quickly as the server can send them. This has the effect of forcing servers to segment data into small chunks that the client can easily consume, but increases the total time required to deliver the content.

This ties up the application and web server, reducing the total number of clients it can serve at any given time. Application delivery controllers address this issue by employing a technique often called "content spooling." Content spooling allows an ADC to mediate between the client and server and allows the server to send content as fast as possible to the mediating device. The ADC then takes on the responsibility of spoon-feeding that content to the client, allowing the application or web server to handle more requests.

Another challenge arising from the use of XML to transport data is, however, that messages are all too often exceedingly fat. Message sizes in the 100's of kilobytes are not uncommon, with messages reaching into the gigabytes in size less common, but not unheard of.

As with the challenges related to the size of HTML and other text-based message formats, the issues associated with the size of messages are easily addressed by the right solution. A number of acceleration technologies exist today that are capable of employing a number of compression techniques to reduce the overall size of messages, thereby decreasing the time to transfer between server and client. Industry standard compression technologies, stripping of white space, and other data reduction technologies exist solely to respond to this challenge and reduce the impact of large messages on the application infrastructure and delivery network.

Application acceleration-specific products as well as many of the features associated with ADCs can provide many of these capabilities, as well as advanced mechanisms for assisting with issues only found in browser-based applications such as artificially imposed connection limitations that inhibit AJAX and browser-based XML technologies from performing at full capacity. Dynamic content caching is as applicable to XML as it is to HTML and related text-based formats, and can increase the performance and capacity of SOA-based deployments dramatically. WAN optimization controllers, as well, can provide additional improvements in remote-office scenarios by taking advantage of symmetric data reduction capabilities.

Conclusion

Many of the challenges associated with SOA deployments are the same challenges we've encountered when deploying traditional web-based applications. The underlying causes of these challenges may differ in a SOA, but the solutions to address those challenges remain the same. The key to addressing SOA-related challenges is to identify them as early as possible and include the appropriate solution in the architecture to reduce the possibility of re-architecture being required in a post-deployment environment.

Challenge	Impact	ADC Solution	Benefit
Exponential increase in connections	• Overhead of TCP connection management adds to burden on servers • Additional hops add additional latency	• Connection management • Hosting shared services	• Reduces number of servers required, lowering costs • Reduces complexity and time to manage • Maintains availability and increases performance • Reduces requirement for duplication of functionality in deployed services
XML Message size	• More bandwidth consumed • More resources consumed to parse/process messages	• Content spooling • Compression • Caching • Hosting shared services • QoS	• Increases capacity of servers, reducing costs and increasing operational efficiency • Mitigates risk of attack • Prioritizes messages to ensure SLA compliance
Scalability	• XML parsing is compute intensive and requires horizontally scalable infrastructure • XML appliances that offload XML specific duties do not support persistence, failover, or advanced load balancing capabilities	• ADCs support persistence, advanced load balancing capabilities and failover scenarios	• Maintains availability of services • Increases performance • Protects investments in technology by ensuring horizontal scalability • Supports necessary capabilities lacking in other solutions

Security	• Exploitation of vulnerabilities in application infrastructure • Theft of sensitive data	• Application firewall capabilities • Content scrubbing • Data validation	• Prevents attacks from reaching application infrastructure • Improves performance of application infrastructure by reducing duplication of code

About F5

F5 Networks is the global leader in Application Delivery Networking. F5 provides solutions that make applications secure, fast and available for everyone, helping organizations get the most out of their investment. By adding intelligence and manageability into the network to offload applications, F5 optimizes applications and allows them to work faster and consume fewer resources. F5's extensible architecture intelligently integrates application optimization, protects the application and the network, and delivers application reliability—all on one universal platform. Over 10,000 organizations and service providers worldwide trust F5 to keep their applications running. The company is headquartered in Seattle, Washington with offices worldwide. For more information, go to www.f5.com.