



Accelerate your SOA Projects through Service Simulation

Overview

Modern web services-based Service Oriented Architecture (SOA) enables service consumers and producers to exchange messages over ubiquitous standards such as XML and SOAP. Typically, companies embark on SOA projects for system-to-system integration within their corporate domains as well as with external trading partners. Re-usable services are the cornerstone of a successful SOA implementation. In a SOA Project, once a producer service is built, developers can subsequently start implementing consumers that can invoke the producer service. If the producer service is under development and is not available for request-response interaction, a consumer development team is severely hampered in its ability to develop consumer code. In addition to such linear task dependencies that dilate SOA project timelines, corporations have to build expensive “reference” infrastructure for their trading partners to use during their development process – developing against a production system in neither a recommended best practice nor a practical approach. Finally, governing what kind of messages are acceptable, beyond what is communicated through a WSDL – the prevalent web services description language – is crucial to ensure seamless message exchange.

Service simulation – the ability to mimic producer services even before they are implemented – enables consumer developers and QA professionals to parallelize their efforts without having to wait for service implementation to complete. Service simulation also enables corporations to provide a portable alternative to building an expensive reference environment. By using a services simulator, such as SOAPSimulator™ developed by Crosscheck Networks, corporations can also provide SOA governance through policies that consumer development teams have to meet before their client-side code is put into production. In this paper, we will cover SOA Project Life cycle issues and how best to address them through service simulation.

I. Introduction to Service Simulation

Web services – the foundation of modern Service Oriented Architecture (SOA) – are self-contained, modular applications that one can describe, publish, locate, and invoke over a network. Web services are agnostic to operating system, hardware platform, communication protocol or programming language. Most IT assets such as application servers, ESBs, RDBMS, CRM/ERP applications, and SaaS products now advertise their interfaces as a Web Services Definition Language (WSDL) interface ready for SOAP/XML messaging. The fundamental advantage of deploying a Service Oriented Architecture is reuse of services exposed via standards-based WSDL interfaces in a service consumer (client) -producer (server) interaction model.

Once a producer service is built, consumer developers are handed a WSDL file that describes the service interface. This WSDL file is used to generate SOAP messages to the target producer service and receive SOAP responses from the service. If the producer service is under development and is not available for such request-response interaction, the consumer development team is severely hampered in its ability to develop consumer code and validate the SOAP messages. Service simulation plugs this gap by mimicking a service before it is ready and available. Service simulation provides a mimicked image of a service that can be invoked by service consumers without having to wait for the services to be built or accessible. As show in Figure 1 below, a producer service may not be available because it is still being developed. In some instances, it may not be accessible to the consumer because the consumer may not have been granted access rights through a corporate firewall. In both scenarios, significant hurdles and delays can be eliminated by deploying a services simulator such as SOAPSimulator™ from Crosscheck Networks.

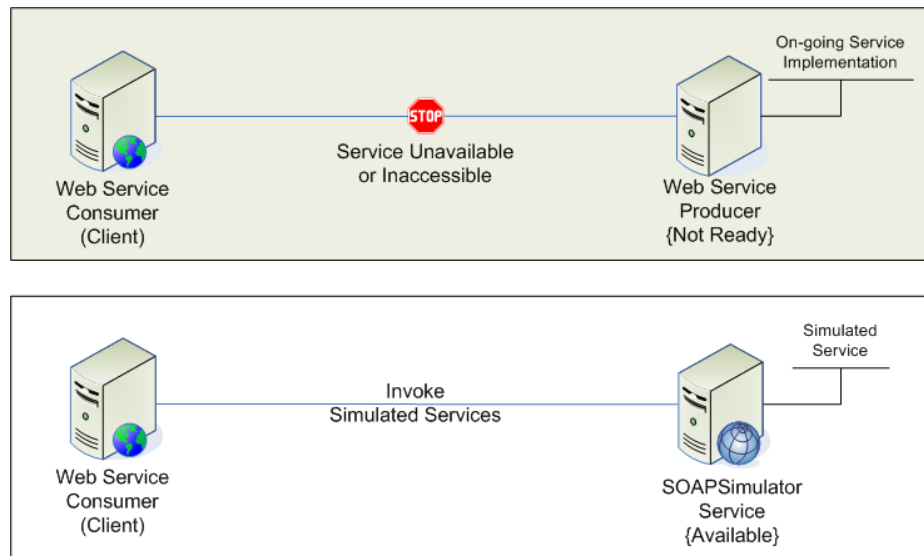


Figure 1: Service Simulator representing alternate endpoints for service invocation and testing.

In this paper, we will examine the timeline, costs and best-practices adherence associated with a SOA project and techniques of addressing such issues by using services simulation. The three areas that we will focus on in this paper are as follows:

1. How to compress a SOA Project Lifecycle.
2. How to reduce costs of a SOA Project Lifecycle.
3. How to build a set of best-practice governance policies for seamless integration.

In the following sections, we will examine time, cost and integration challenges on a SOA Project in detail and look at service simulation techniques to reduce their impact.

II. SOA Lifecycle Challenges

Well known challenges of legacy distributed computing are exacerbated in the modern, web services-based world where distributed interactions are no longer contained within the confines of a corporation but span across corporate suppliers, buyers, and 3rd party service providers. Because of its highly distributed nature, a SOA Project can suffer extensive Project Timeline Dilation, increasing Project Costs, and significant Integration hurdles. Building a web services-based SOA provides a unique set of challenges beyond the challenges of legacy distributed computing architectures. These challenges include:

1) *Project Timeline Dilation*

In a SOA deployment with highly reusable services distributed across internal and external networks, the consumer-producer model complicates project dependencies between consumer development and producer development, and between QA test case authoring and producer development. The producer services are a gating factor and have to be complete and available before consumer-side development and test case authoring can commence. Delays in developing producer components have a domino effect on testing and development tasks that lead to significant project delays. If producer services are not ready or available to developers and testers then they have to wait for the services before they can become productive. Productivity can also come to a standstill if a test producer service abruptly becomes unavailable. In large corporations with extensive perimeter security enabled through firewalls, getting access to internally deployed services requires significant IT resource coordination and legal contract paperwork that can further add delays to the project lifecycle. Regardless of whether services are inaccessible because of firewall restrictions or unavailable because they are still being developed, SOA Projects face such hurdles that dilate timelines and contribute project delays.

2) *Project Cost Increase*

In a SOA Project Lifecycle, once a WSDL file is available, consumer development teams and QA teams require an endpoint for a SOAP request-response interaction. Typical enterprise-class SOA efforts deploy a "reference architecture" that acts as a fully functional replica of the production environment. Developers and testers use this reference architecture to build their consumer code and test suites against. Once the consumer application is code-complete and tested against the reference architecture, the consumer invocation path is switched from reference architecture to the production architecture. Reference architectures have to be fully functional replicas of the production environments. This requires duplicated instances of expensive SOA components such as application servers, databases, ESBs and SOA gateways, and IT infrastructure such as routers, switches and load balancers. These components add significant hard cost in software licensing fees, hardware, and support costs to a SOA project.

3) *Integration Challenges*

In SOA deployments, complex message- and protocol-based standards are used for enabling system-to-system integration. Established standards such as XML, SOAP, and WSDL provide ample richness and flexibility for message exchange. Higher level WS-* standards address areas such as message-level privacy, integrity and identity. With such wealth of flexible standards, SOA deployments tend to cover a large spectrum of complexity starting with the simple SOAP request-response over HTTP deployment without any authentication to complex asynchronous SOAP messaging that uses message-level security and authentication through standards such as WS-Security and SAML. With distributed corporate development teams, offshore development shops, and a variety of internal platforms that have differing implementations of web services specifications, seamless integration can become a daunting task. For corporations trying to realize the benefits for SOA by rapidly integrating with their trading partners who they may have little or no control over, the promise of seamless and rapid integration seems unreachable.

As daunting as these challenges may seem, through service simulation, these challenges on SOA Projects can be addressed directly and their effects mitigated. We will now explore how to address these challenges by using service simulation techniques.

III. Compressing SOA Life Cycle through Service Simulation

SOA Project Timeline Dilation arises from a number of factors such as inaccessibility or unavailability of producer services. Even when an elaborate reference-architecture has been built for developers and QA teams to work with, time-slots have to be allocated to team members to prevent collisions. An intrinsic delay exists in a SOA Project caused by the dependency between consumer and producer in a web services-based SOA Project. Using service simulation – mimicking a service before it is ready or accessible – can decouple this development dependency. A sample SOA project, developed with and without services simulation is shown in Figure 2 below. Without services simulation, this hypothetical SOA Project works in a “Linear Mode” and takes 12 weeks for design-to-rollout. It follows a classic waterfall model with project design followed by producer development, producer unit testing, consumer development, system-testing and project rollout.

With service simulation introduced, this sample SOA project can be compressed by removing two critical dependencies:

- 1) Consumer-side developers can start developing at the same time as producer development teams.
- 2) Producer test team can start authoring test suites in parallel with the development teams.

Services Simulation, therefore, provides a “Parallel Mode” for SOA development that helps compress SOA deployment timelines.

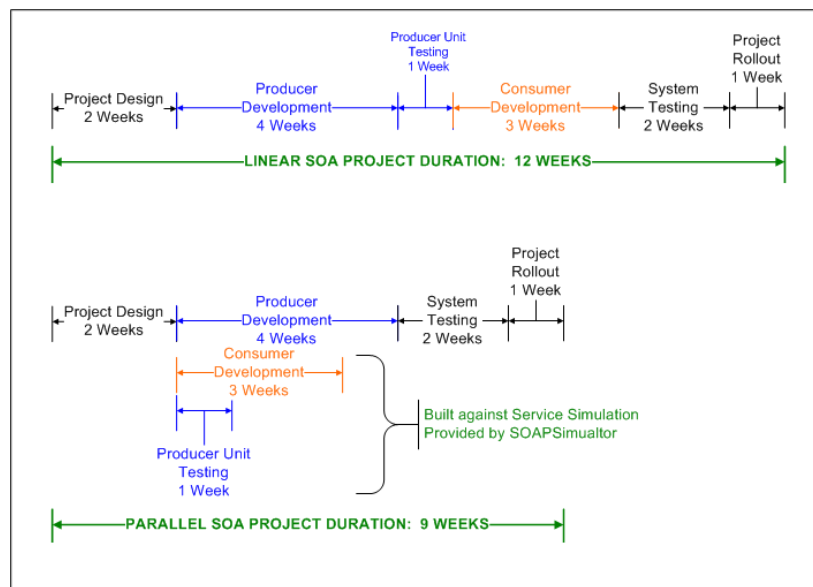


Figure 2: Compressing SOA Project Duration through Service Simulation.

In the hypothetical SOA Project shown in Figure 2 above, the results of parallelizing the consumer development, producer development and test authoring efforts through a services simulator provides quantifiable time compression – 9 weeks from 12 weeks in this case. This illustrates the efficiencies gained through parallelizing tasks in a SOA project. One of the key assumptions for realizing such gains through service simulation is the ability to rapidly and easily install and configure a service simulator. Without such ease-of-use and configuration of a simulator, the gains from using a service simulator are marginalized by the overhead of the simulator solution itself.

In SOA projects that require access to producer services behind a firewall, the project timeline can experience significant delays because of service inaccessibility. Obtaining permissions for producer services may require a lengthy legal process and agreements to be signed as well as the operational overhead of the IT resources and network policy changes. Such delays can be avoided by providing consumers a portable simulator that contains all the request-response pairs required to start the consumer development process. While the legal and business relationships are being worked out between the service producers and its consumer, the development and testing teams can start their tasks without requiring access to the producer services deployed behind the corporate firewall.

V. Reduce Integration Costs with Service Simulation

SOA projects require consumer development teams to develop against a producer service. Typical SOA projects mandate that such consumer development does not occur against production systems. They tend to build out a fully replicated, full-scale “reference” image of the production systems for the developers to develop against. Once the development is complete, the consumer components are pointed away from the reference architecture to the production instance of the producer services.

With service simulation, using products such as SOAPSimulator, a SOA project’s cost can be significantly decreased by:

- 1) Realizing that, to a consumer, a producer is an independent façade. The consumer does not care about what is being invoked by the producer service as long as the service consuming and returning messages as expected.
- 2) Eliminating hardware and software costs required for building full-scale reference architecture that is a replica of the production environment with application servers, ESBs, RDBMs, and SOA gateways.
- 3) Eliminating the configuration, management and on-going support costs in maintaining reference architecture that is a replica of the production environment.

Service simulation provides a mechanism for building a “light-weight” reference architecture by eliminating expensive IT components and without sacrificing the functionality required by consumer developers to build their applications hooks into the producer systems. Full-scale reference architecture adds significant expense to a SOA project without guaranteeing timely access to consumer developers who may need to jump through legal hoops before being granted access to the producer services within the reference architecture.

A self contained, desktop-installable and self-contained service simulator – such as SOAPSimulator – eliminates the need for a full-scale image of a services production environment. It can run on a developer’s desktop mimicking the request-response pairs required for developing consumer code. SOAPSimulator serves as a “reference architecture in-a-box” by providing cost-effective, portable and readily accessible simulation of producer services.

VI. Reduce Integration Challenges through Best Practice Governance

Modern web services-based SOA projects rely on ubiquitous base standards such as SOAP/XML and WSDL to transmit messages and describe service interfaces. Beyond these base standards, a number of standards have evolved for addressing message integrity, privacy, and identity and transmission reliability. Collectively, these web service standards, expressed as WS-*, provide high degree of flexibility for system-to-system integration. However, this flexibility creates a management nightmare for corporations with distributed development teams and external partners using a variety of programming languages and platforms with differing degrees of standards implementations.

With service simulation, using products such as SOAPSimulator, a corporation can ensure seamless integration by:

- 1) Providing quantifiable metrics on corporate-wide service interfaces (WSDLs) regarding their adherence to corporate rules such as service naming conventions, data types, data structures, and protocol bindings.
- 2) Ensuring that both consumer and producer development teams adhere to corporate mandates on message security and interoperability, for example, WS-Encryption and SAML must be used for message-level privacy and identity.
- 3) Governing use-cases within their enterprise as well as with their trading partners, for example, messages with suppliers must have WS-Signatures enabled for message integrity.

Development teams are typically more focused on building business logic rather than being concerned about security, integrity and access policies. Usually such policies are an afterthought and are not baked into the SOA project plan. To a developer’s surprise, because of the consumer-producer coupling, either end of the message flow can mandate security; identity or message transmission reliability standards much later in the SOA project. Such surprises are costly and can result in failed integrations and delayed SOA projects.

With a service simulator – such as SOAPSimulator – granular decisions on message structure, interface definition and associated security, identity and reliability standards can be made and mandated much earlier in the SOA development process. Both ends of the consumer-producer development teams can share a common service simulator configuration that ensures seamless integration. By sharing such configuration and using a simulated environment, integration issues can be identified and solved early in the SOA project lifecycle with significant cost savings.

IV. Introducing SOAPSimulator for Service Simulation

With complex consumer-producer dependencies, SOA Developers and Testers face unique challenges in managing the SOA Lifecycle. SOA Architects, Developers and Testers are responsible for adapting their testing techniques, selecting appropriate testing tools and developing web services domain expertise to make their SOA deployments deliver business value reliably and securely. SOAPSimulator™, shown in Figure 3 below, provides an easy-to-use, point-and-click service simulation environment that can be installed on a developer's desktop machine. Once the development and test teams have SOAPSimulator, they are no longer gated by the actual web services implementation timelines and can commence test authoring and client side development. Using SOAPSimulator is a simple 2-step process. The WSDL file from yet-to-be-implement services are loaded in SOAPSimulator followed by setting SOAP Request and Response values in a simple tree-structure as shown below.

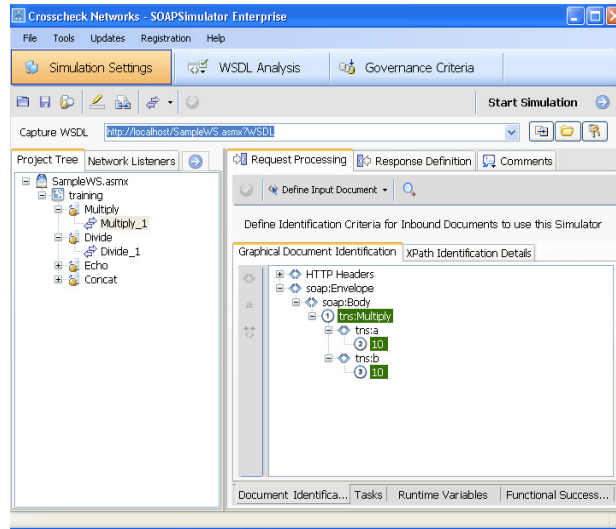


Figure 3: SOAPSimulator for SOA Service Simulation

SOAPSimulator™ also provides extensive support for WS-Security standards, SAML, schema validation, WS-Addressing and a variety of identity tokens such as SAML, Kerberos, X.509 and User Name tokens. Policies can be set for both incoming-request as well as out-going responses. SOAPSimulator also provides an industry-first, interface-scoring module which generates a WSDL Report Card. Based on extensible rules, the WSDL Report Card provides a detailed score for the WSDL services based on best-practice rules established by the corporation. Using such quantitative techniques ensures that there is measurable convergence towards corporate SOA mandates thereby improving the quality, security and interoperability of a SOA deployment.

VI. Conclusion

The promise of web service-based SOA lies in re-usability across distributed environments. Intense time pressures to build and deploy services leaves very little room for error in meeting business goals set forth for SOA projects. With IT budgets under constant pressure, tight delivery deadlines, and the drive to integrate systems within and across trading partners, SOA projects need to introduce service simulation within their environments. Service simulation decouples consumers and producer dependencies and enables them to implement independently. Service simulators such as SOAPSimulator by Crosscheck Networks provides a “reference system in-a-box” and eliminates the expense associated with building a full-scale replica of the production system for the benefit of developers. Additionally, SOAPSimulator provides point-and-click service simulation that allows for rapid simulation development. And then there is the measureable advantage of using service simulation to provide a design-time and run-time analysis on the adherence of a consumer to corporate best practices. Those services that do not meet the governance criteria are quarantined until they meet a minimum governance score at which point they are put in production for consumers to invoke. Service simulation provides a powerful mechanism for corporations to accelerate their SOA project by parallelizing development tasks and measuring governance without building expensive full-scale replicas of their SOA production environments.