For Net-Centric Operations, the Future Is Federated

John Michelsen
iTKO, Inc.

SOA (Service-Oriented Architecture) is an ideal strategy for enabling net-centricity across shared application environments. But before material providers can realize the flexibility and reuse advantages of migrating systems to a shared SOA model, how can they avoid the risk of missed end-customer requirements? Achieving a sense of trust in SOA requires more than the right development and testing technologies. Net-centricity requires constant validation and a shared certification process to ensure that applications are meeting the needs of the warfighter – even as the shared technology environment changes and evolves.

Leading institutions are moving toward net-centric systems: highly distributed and flexible methods for delivering required technology in a shared, reusable way to support operations. Achieving net-centricity in the military and other government agencies will require both organizational policies for sharing a common set of goals and technology strategies like SOA.

However, there are good reasons why divisions often build and maintain their own monolithic applications: a lack of trust. How can you be sure that a third party will deliver the required functionality if they are not on the line to make it work for you? Realizing an effective federated SOA strategy in mission-critical theatres, such as warfighter scenarios or air traffic control, requires establishing trust between peer units with services managed outside of the vertical chain of command.

Technology innovation around SOA is happening on a rapid scale within both defense and civilian government agencies. This change is being driven by economic and operational concerns that the business world at large may not fully comprehend. Indeed, a typical business technologist may comment on the operational and budgetary bloat of federal organizations, but reality demonstrates that there is fierce competition for investment dollars within the public sector. Programs must reach milestones of success, despite resource and time constraints, and within a potentially even more dynamic environment than the typical commercial enterprise.

By pushing for long-term goals, the public sector provides value to taxpayers by helping create a purpose-driven market for technology investments — opportunities for innovation that the stockholder-driven company can never realize.

However, there needs to be a realistic understanding of the issues at hand, and perhaps they need some new approaches other than building architectures that look like archeological layers of past IT contracts. I do think some of the more spectacular SOA successes will come from the government side. 

We can think of this system just like we think of the idea of multiple state governments operating within a federal government. Each state has its own laws, objectives, and budgets, but all of the states in the system are also governed through a federal authority. This model can also be applied to SOA application architectures, to define a Federated SOA.

The Federated SOA will provide a leading indicator for where net-centric software innovation is headed for the business world. The innovations that led up to DARPA or CMMI were not designed simply to make money or optimize cost. They were born from the idea that we have a specific, targeted outcome to reach. Let us take a step forward to prove why governmental approaches are going to produce leading-edge techniques for SOA governance.

Challenges

Federal agencies, just like any other global enterprise, are now at a crossroads for establishing an SOA strategy in a world where no single strategy can possibly cover every need. Countless siloed technologies currently exist as acronyms within each operational unit. Supporting and maintaining so many separate platforms becomes untenable over time, as any additional functionality or code adds to the existing technologies in a stovepipe fashion. Each new customization and every line of code written for one of these stovepipe technologies results in a long-term annuity that will have to be consumed through a central, shared authority and knowledge base while retaining the flexibility to meet operational and business needs.

Pertaining to the government as a model SOA example, Dave Linthicum of the Linthicum Group said the following:

I think that the government can benefit most from SOA, considering the nature of their business and the underlying need to have many systems interoperate.

* CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
paid back and supported over time.

**Organizational: Vertical Trust vs. Horizontal Competition**

For net-centric computing environments, *vertical trust* (i.e. trust up and down a common chain of command) is far easier to achieve than *horizontal trust* (i.e. across authority domains or organizational units).

**Vertical Trust**

In most organizations, vertical trust is relatively easy to come by and already exists. The line of reporting structure typically creates the sense of reuse standardization and the ability to leverage vertically within an organization.

As shown in Figure 1, in vertical hierarchies within organizations, there is an expected level of shared trust:

- The higher levels within can expect the underlying teams to *build to order* technology assets and maintain them according to defined policies.
- Supporting providers can expect the requestor to leverage the developed materiel capability services according to well-understood and defined requirements.

**Horizontal Trust**

At the highest levels, horizontally across peer groups, a lack of trust is usually not the case (see Figure 2). Ironically, a lot of organizations that share common goals tend to foster a *not invented here* mentality that inhibits collaboration between horizontal peer groups. That is the source of why horizontal governance is such a critical aspect to SOA adoption.

Across different operational or business units, coordinating the proper use of a service can be difficult:

- Materiel or service providers want to establish reuse of their services, but they are answerable to different stakeholders.
- Upstream consumers of services may not provide clear enough use cases or policies of how they will leverage the services.
- Therefore, teams often build and maintain redundant functionality in vertical silos.

Redundant efforts create inefficiencies when software functionality gets duplicated. Therefore, before we can realize the value of reusable services offered within a federated software strategy, agencies must learn to establish trust horizontally, across organizational and chain-of-command boundaries.

**Federated Technologies Are Heterogeneous**

The operational rules and behaviors of an SOA live in the middle tier (between the interface and the database layers), iterating within an *alphabet soup* of technology acronyms (for example, XML, SOAP, WSDL, ESBs, etc.), which are exposed as services. These services are technology assets that can be managed or published within your own authority domain or reside outside the department or even outside the organization.

**Services Need Not Be Web Services**

Many technology vendors simply equate Web services (WSDL/SOAP) with SOA; from a testing point of view, they equate the testing of Web services with the testing of SOA.

While it is true that a number of initiatives for doing SOA are very Web services-centric, the Aberdeen Group’s last research on this points out that only about 50 percent of the SOA initiatives at best-in-class companies are Web service-based [2]. There are a variety of technologies being used to create that commoditized middleware for SOA. While Web services can be a good integration strategy, other technologies are valid and possibly better for a given organization than a Web services stack, for instance, using an Enterprise Service Bus with little reliance on Web services.

The distinction between Web services and SOA testing in general is important. There is more than one way to build an SOA. Teams need to test the implementation and side-effects that occur across heterogeneous technologies, as opposed to just a selected middleware layer like SOAP.

As illustrated in Figure 3 (see page 26), there is inevitable complexity under the surface of any large-scale implementation. In an SOA, *more unique technology types*, multiplied by *more points of connection* equals an *exponential increase in possible failure points*.

Heterogeneous technologies will never go away and leave behind a totally homogenous platform. There are several reasons for this:

1. **Legacy systems cannot just be turned off.** Technologists are always attempting to provide more flexible application architecture at a lower cost to their constituents. As technologies evolve, we almost never have a cost justification to retire existing systems and replace them with the new technology. The Web services

   ![Figure 1: Vertical Governance Along a Chain of Command](image1)

   ![Figure 2: Horizontal Trust Across Organizational Boundaries Bears More Risk of Misunderstandings or Missed Requirements](image2)
stack now considered modern technology will soon be outdated, and an integration strategy to leverage these soon-outdated applications with tomorrow's better thinking will be needed.

2. **Resistance to vendor monopolization.** A *we do it all* vendor can promise uniform specifications and the benefits of increased scale, but for larger clients, tying the entire technology to the platform of a single vendor may inhibit specialization and become perceived as a long-term risk if development priorities (or pricing structures) change. Thus, even new applications are built on a variety of technology platforms.

3. **Distributed authority domains.** Federated organizations have multiple chains of command. Operational units have unique functional needs from technology assets, therefore, they naturally desire to keep some teams and service assets under direct control.

To trust SOA, a much deeper level of collaboration testing must occur as a continuous process, not an event. The services, and the expected uses of them, must be submitted and certifiable (functionally and at load) to the community relying on the SOA.

**Policies Are Hard to Follow**

A policy basically defines an expected behavior. Governance of SOA applications goes hand-in-hand with defining and enforcing policies in order to achieve control of, and trust in, the SOA application.

But what is policy? Often technologists mistakenly approach policy as a solely structural concern. For instance, *is the syntax of the XML message formed correctly? Do the components connect according to our selected standards? Structural concerns are just one aspect of policy that needs to be addressed.*

Once structural standards are in place, a behavioral definition of policy will become a determining factor in the success of any SOA endeavor. A behavioral policy basically defines what the system should operationally do to support its intended use.

We need to realize that the organization will not want to write policy concerned about technical standards. Rather, the organization's version of policy will be the following:

- I really need functional integrity on this particular transaction activity of the system.
- I really need the response on resource availability to be accurate within 30 minutes of my request.
- I cannot allow stale data to be reported as current activity in the field.

Net-centricity is about sharing operational functions and placing expectations upon the systems that are implementing those policies. That is meaningful policy. SOA policy must focus on the functional integrity of the application—the quality and reliability of the end customer experience, accuracy of data, and runtime performance of the application.

**Service Consumption Is Not Free**

There is a commonly held (and somewhat sentimental) notion that once the SOA architecture is in place, it will provide an environment of published services that multiple consumers can basically leverage within their own workflows at little or no cost. This model is valid for very non-differentiated or commoditized services such as news feeds, simple calculators for unit conversions, and the like.

Take for instance a small application, developed specifically for a military unit's internal use. The certification level, and the level of structural, behavioral, and performance policy will not be nearly as high. The maintenance cost for that service will be much lower and there will be much less risk and testing rigor in changing that service when a finite set of consumers are impacted by that change.

Now consider a mission-critical, broadly reused service that will have a widely distributed use among all military divisions, including some that the development team may not even yet be aware of. That creates an increased cost of producing a service that is robust over time and reusable. If consumption creates cost and effort for the producer, then the consumption itself should not be free.

A reckoning must occur when a consuming project team gets benefit from an existing reusable service that is properly maintained. Otherwise, the production of robust, quality services is penalized. An increased cost burden, without an associated increase in the budget for that service to be reused, will actually threaten that service's long-term quality and adherence to the policy the consumer expected.

Consumption of existing services can reduce the cost and effort of producing a solution. Over time, the service consumer must bear some accountability to the publisher of those services. If all parties involved in SOA realize that there is no *free beer*, the end result will be a more sustainable marketplace of services.

**Solutions**

**Establishing SOA Governance Enhances Capabilities**

The primary goal of net-centric computing is to enhance the ability of each organization to efficiently meet the needs of the warfighter. If the level of trust is high, the organization can rely on both the historical and runtime validation of every service it depends on. Without SOA governance, SOA remains a chaotic, free-form exercise.

If the extended organization plans to overcome the *vertical silos* and rely on an application made from separately managed services, each being developed and maintained on their own life cycle, what are the rules of the road?
According to Gartner analyst Frank Kenney [3], SOA governance is made up of three components: the Registry (or Repository) where the assets of SOA are stored and catalogued, Policy which is meant to keep track of the rules of engagement and service levels expected in SOA, and SOA Testing that is needed to ensure SOA life-cycle quality.

What good is a registry if it contains assets that are not sufficiently tested at both the service and the implementation layer? Strong testing is required to ensure that the SOA application continually meets the business needs – in development, integration, and deployment. The longer testing is delayed as an aspect of SOA governance, the wider the deviation becomes between expected and delivered results.

Defining a Big Policy
As we continue to mature the SOA governance space, the policy area appears to be the one that is the most immature. In the near future, governance will become synonymous with policy. Each type of SOA policy is vitally important to achieving reliability and trust, as shown in Table 1.

Currently, most technologists focus on testing the structural policy type mentioned in Table 1. True, integration standards are important, but once those types of problems are solved, behavioral and performance level validation will gain prominence. After all, what good is certifying structural policy if the SOA application does not perform its function correctly and at the expected scale, design time, run time, and change time?

The Certification Environment: Two Sides of the Coin
There are two critical certification flows to a robust, federated test and policy validation strategy: a publish cycle and a consume cycle. These can be considered federal-level processes that form a certification environment, a central authority offering the rules of the road for creating and offering services, then leveraging these created services in an expected fashion. The need for a publish cycle is evident: A standards body must establish and enforce criteria that provide for an environment of trust to encourage and enable reuse. The consume cycle is equally as important, as it must properly lay out the expected use cases for published services in a realistic and enforceable way.

The Publish Cycle
A development team that wants to offer up a service to the community submits their asset as a proposed service for reuse (see Figure 4). This service is reviewed by a cross-domain group called a certification group that must verify that the offered service conforms to expected policies before making it available to the community. Next, the group needs to continuously monitor and test those services, as they may change or fall out of expected policy guidelines.

How the Publish Cycle Works
1. **Offer.** A developer or development team creates a new, uncertified version of a Service Component (SC).

2. **Certify.** An independent third party certifies the service component after it passes the necessary tests.

3. **Publish.** The service component is published to the registry and other SOA environments.

They can offer the working SC along with documentation to a certification group for testing and approval. Optionally, they can include working test cases as part of the documentation process to aid certification and store those in the registry.

- Developers self-assess quality by creating tests against their own SCs (whether it is a technology component or Web service) before submitting them.
- Developers offer the proposed SC to a centralized registry for certification. The development team can

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Definition</th>
<th>Examples of Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural</td>
<td>The services components are compliant with chosen integration standards and reusable with the current development, deployment and governance platforms.</td>
<td>- Do the pin outs line up so that the components can technically communicate with each other? - Are the services following correct authentication protocols? - Is the XML syntax compliant?</td>
</tr>
<tr>
<td>Behavioral</td>
<td>The service interacts and provides correct results within the context of the workflow or task that needs to be accomplished.</td>
<td>- Are the results I expected actually being produced? - Does the operational logic of this service properly support the process it is being used for?</td>
</tr>
<tr>
<td>Performance</td>
<td>The service can sustain the performance, scalability, and reliability levels required over time.</td>
<td>- Can this component produce the results I need with the number of users I need, within the time constraints, and infrastructure that I need it?</td>
</tr>
<tr>
<td>Runtime</td>
<td>Expectations around the service level of the component in the live production environment.</td>
<td>- Expected response time for this service is one second for our top 20 constituents and three seconds for all others. - Uptime must be &gt; 99.999 percent.</td>
</tr>
</tbody>
</table>

Table 1: Defining Policy Types

| Service Components are compliant with chosen integration standards and reusable with the current development, deployment and governance platforms. | - Do the pin outs line up so that the components can technically communicate with each other? - Are the services following correct authentication protocols? - Is the XML syntax compliant? |
| The service interacts and provides correct results within the context of the workflow or task that needs to be accomplished. | - Are the results I expected actually being produced? - Does the operational logic of this service properly support the process it is being used for? |
| The service can sustain the performance, scalability, and reliability levels required over time. | - Can this component produce the results I need with the number of users I need, within the time constraints, and infrastructure that I need it? |
| Expectations around the service level of the component in the live production environment. | - Expected response time for this service is one second for our top 20 constituents and three seconds for all others. - Uptime must be > 99.999 percent. |

Figure 4: Publish Cycle for Service Providers to Submit and Certify That Their Service Assets Will Meet the Needs of the Net-Centric Community

SOA Publish Cycle

1. **Offer.** Candidate Services
2. **Certify.** Component Testing
3. **Publish.** Structural Behavior Performance
4. **Review.** Metrics and Alert Monitoring
5. **Check-in.** Continuous Policy Tests
6. **Verify.** Structural Behavior Performance

September 2007
accelerate the process by submitting functional, performance, and other tests to validate the policy out of the box.

2. **Certify.** Certifiers use existing tests and iterate on those tests to validate that the SC meets all required policies at every stage of the service’s life cycle. Tests are checked into the test registry/repository alongside the services, which are rated according to their level of certification (each organization defines its own certification levels, for example, from trial level of certification to partially certified to fully certified).

3. **Verify.** Initial certifications are not much good if they are not enforced later in production. Certifiers register the test cases, and they are run continuously to validate that expectations are met as the application environment evolves. Continuous verification happens on a regular time interval or based on any system level event.

4. **Review.** All metrics and test information is published to both certification and publishing teams for reporting and alerting on SC issues. Alerting mechanisms can inform any development or deployment team of exceptions or errors within the certification environment.

**The Consume Cycle**
The Consume cycle is equally important; those who plan to leverage published services must establish a model workflow outlining their expected behavior so that ongoing change does not cause the system to fail in unexpected ways (Figure 5).

### How the Consume Cycle Works
Consumers browse the registry of available services and use them to define workflows that consume one or more SCs as steps needed to complete an objective.

1. **Discovery.** Development consumers browse available SCs and their associated published policies and test cases in order to determine applicability to their proposed workflows. Development teams can test them using existing tests or combined with a target workflow test (i.e., testing the validity of the workflow in absence of the underlying services).

2. **Confirm workflow and set policy.** Certifiers verify that the intended uses defined within the consumer’s workflow are achievable, and once the workflow is certified, it is published to the registry as a set of expected behaviors – a policy – that can be certified via suites of test cases that accompany the workflow.

3. **Workflow validation.** Test suites for workflows are checked into a continuous testing process as policies for continuous monitoring of required quality of service – performance, scalability, and reliability. These tests set the bar for candidate and certified services that are accountable to support the workflow.

4. **Alerts and exceptions.** A test dashboard provides key metrics to development, certification, and administration teams. Workflows are monitored for issues as SC development life cycles and demands on deployment evolve. If an exception, error, or boundary condition event occurs that violates one or more workflows, stakeholders can be alerted with root cause test cases provided.

Most organizations have not even considered managing policies or tests for how they consume services. There is no free beer in SOA. If there are no expectations placed upon the consumer of services, total chaos ensues, and there is no governance. By defining the expected behaviors of the consumer, service providers in the network can supply provisions for this, and validate that all the critical workflows are supported both now and in the future when each new release of the service component is proposed.

### A Center of Excellence for SOA Life-Cycle Quality
**Go Horizontal for SOA Excellence**
One way to instill a sense of horizontal trust across the organization is through an SOA Center of Excellence (COE). For example, consider an analogy of states’ rights vs. federal rights. The individual divisions (or states) need to consider themselves autonomous on certain levels and owe certain rights at the federal level; but they also have the opportunity to participate in how policies are set at that federal level, so that it is not just a down from above edict that immediately creates a defensive reaction instead of a unified purpose.

Booz Allen Hamilton Vice President Art Fritzson noted the following:

I tend to reduce net-centricity to two questions that are both addressed by community. *Have you asked your community for help? and Are you helping your community?* If you get positive answers to both those questions, you’ve got 80-90 percent of what net-centricity promises just through behavioral changes. [4]

Which policies need to be tested as so-called federal policies and which can safely be handled at the state level? Policy testing can be automated along four domains: structural, behavioral, performance, and runtime. At the federal level, structural policy (or compliance) might be the most important aspect, while from a state-to-state, horizontal kind of policy, behavioral and performance aspects will perhaps be most important to define and test at those levels.

---

**Figure 5:** Consume Cycle for Sharing Testing and Certification Processes in a Net-Centric Environment
Publication and consumption of shared services will only happen in an environment where trust is fostered. Trust can never exist unless empirical data (test data) that supports assertions against expected policies about a component (service, component module, etc.) can be quantifiably and continuously obtained. The levels of certification, and what is defined as important to test, will differ for different communities of interest (COIs). Each federated consumer may reuse, or not reuse, those assets that will deliver efficiently for them.

**Life-Cycle Quality and Testing**

Software testing in the standard waterfall development cycle has long been relegated to a project milestone somewhere after development and integration happens. But in SOA, life-cycle quality is a continuous part of SOA governance and not an event that unit tests a specific component (a form of XML) for transmitting data objects among Web services. SOAP stands for Simple Object Access Protocol, which is the common method (form of XML) for transmitting data objects among Web Services and other technologies. For more, see <www.w3.org/TR/wsdl>.

**Conclusion**

This article has outlined several key strategies for leading the way to a true net-centric approach to SOA life-cycle quality and testing, which is one of the three primary components of SOA governance.

Success in SOA is not something you can buy as a software package; it is something you must do. In fact, the quality of your policy and the relevance of your certification efforts depends entirely upon the skill and discipline level of all participants in the SOA strategy. The architect, the developer, the tester, and the requirements owner must work to establish trust, whether from a development perspective or a quality perspective.

The entire extended organization needs to adopt an SOA COE – the federal authority that helps the underlying states align around common goals. If we think about what the SOA COE must look like, then certainly there needs to be a set of participants that are not beholden to any particular one of the divisions involved, but there also needs to be significant membership from all of those who will be involved so that we get the participation. After all, this is not us vs. them – net-centricity is all of us in the same boat together.

**References**


**About the Author**

John Michelsen, is the founder and chief architect of iTKO, Inc. He has more than 15 years of experience as a technical lead at organizations managing large-scale, object-oriented solutions in traditional and network architectures. Michelsen is the chief architect of iTKO's LISA automated testing product and a leading industry advocate for software quality. Before forming iTKO, he was director of development at Trilogy Inc. and vice president of development at AGENCY.COM. Through work with clients such as Cendant Financial, Microsoft, American Airlines, Union Pacific, and Nielsen Market Research, Michelsen has deployed solutions using technologies from the mainframe to the handheld device.

iTKO Inc.
1505 LBJ FWY
STE 250
Dallas, TX 75234
Phone: (877) 289-4856
Fax: (817) 281-2458
E-mail: info@itko.com