SYS-CON

THE WORLD'S LEADING MAGAZINE DEDICATED TO WEB SERVICES TECHNOLOGIES

SOA WORLD MAGAZINE

www.SOA.SYS-CON.com

JUNE 2008 / VOLUME: 8 ISSUE 6

Introducing SOA Design Patterns

THOMAS ERL PAGE 12

$6.99US $7.99CAN

06

0 71486 03420 9

_INFRASTRUCTURE LOG

_DAY 79: This is out of control! Our IT environment is rigid and inflexible. Our business needs are changing, but our environment isn't built to change with them. We can't adapt. Oh, no...I was afraid of this. We're so rigid we're stuck in time.

_*Infrastructurus prehistoricus*. I've read about this.

_DAY 80: I'm taking back control with IBM SOA solutions. Now we can align business goals with our IT. We have the hardware, software and services we need to respond to change. Strategy, planning and implementation are in tune with our specific business needs. Now we can deploy and update business processes faster and more efficiently.

_Goodbye, rigid past. Hello, flexible future.

Take the SOA business value assessment at:
IBM.COM/**TAKEBACKCONTROL**/SOA

SOA WORLD MAGAZINE
www.SOA.SYS-CON.com

Putting It All Together

WRITTEN BY **SEAN RHODY**

When I was a kid, which seems like just yesterday (and no comments from the peanut gallery), I loved playing with LEGO, making imaginary ray guns, space ships, and other things that amuse the average boy. LEGO's popularity and longevity have to be due in no small part to the ability to assemble a new creation from basic components.

Mashups are the IT version of LEGO (or at least the creations we used to make with all the little blocks). They are part of the breakthrough we've been waiting for in the user interface space for quite some time. Our interfaces had become boring, and just like the silos of application functionality that we are trying to dismantle using SOA, our interfaces have been isolated areas on our screen. They overlap, they cover one another, but they never intersect, no matter how much we'd like them to.

Recently it became apparent to me how much mashups are a part of our future, and, more important, how we can come to rely on them without even realizing it. I've been using some real estate sites and have gotten used to mapping and overlays and the ability to resize the maps. Nothing remarkable, really, but I also received an e-mail from a vendor that included a link to a map for the location. When I went to the site, I was disappointed – the map was simply a static image, rather than one I could adjust so I could see routes and other points of interest. I even muttered "How Web 1.0" under my breath. Clearly, I've become spoiled by the capabilities and possibilities that Web 2.0 and SOA offer.

Harvesting these capabilities is the true task of the IT organization, especially the CTO. Keeping the servers up and running the existing applications is definitely a vital part of the IT mission, but it's table stakes – it gets you into the game, or keeps you in it, but you don't win the game with excellence around data centers. You win by adopting new technologies and visioning the future of your business with them in your mix.

Innovation in business is vital to success – whether it is the initial success of a startup or the continued market dominance of a business leader. Technology plays a large role in many market-leading strategies for continued success. Seldom, however, is the differentiator purely or even mainly a technology – it's the business innovation that allows for differentiation that determines who succeeds. In a sense, that "IT Doesn't Matter" article was right – it's not about technology capabilities. But it is about technology application.

The real estate sites that I've been using are a good example. There's a common source of data – the multiple listing service – that is pretty much the table stakes of the game. Without it, you can't locate properties that are for sale. But that basic capability doesn't allow for any differentiation. Now add in the concept of a mashup using a map and you start to see how sites differentiate themselves. Some allow for mapping of properties onto the map, with brief pop-up descriptions of each property within the area. I found these to be particularly useful, because sometimes you want to see what's for sale in a neighborhood, to gauge the worth of a particular listing, or to explore just how volatile a neighborhood's ownership may be.

None of the sites were perfect, which is probably due to the nature of real estate purchases – very few transactions will occur online, so business differentiation in listings will probably never be the area where agencies concentrate permanently. Still, while they have an advantage, I'd be more tempted to go with a broker who had the better site – all things being equal, I tend to favor those who have their IT act together; it speaks to efficiency.

Knowing that a business advantage may dissolve at any time is also a key factor in innovation. You can't rest. You have to have a culture of innovation, of continually striving to be best. Whether that's with technology, business process, customer service, or just plain building a better castle out of LEGO, it's what sets the leaders apart from the rest of the herd. ■

About the Author

Sean Rhody is the editor-in-chief of **SOA World Magazine**. *He is a respected industry expert and a consultant with a leading consulting services company.* *sean@sys-con.com*

EDITORIAL OFFICES
SYS-CON MEDIA
577 CHESTNUT RIDGE ROAD, WOODCLIFF LAKE, NJ 07677
TELEPHONE: 201 802-3000 FAX: 201 782-9637
SOA World Magazine Digital Edition (ISSN# 1535-6906)
Is published monthly (12 times a year)
By SYS-CON Publications, Inc.
Periodicals postage pending
Woodcliff Lake, NJ 07677 and additional mailing offices
POSTMASTER: Send address changes to:
SOA World Magazine, SYS-CON Publications, Inc.
577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677

MAKE YOUR DATA CENTER A LEAN, GREEN IT MACHINE.

TODAY, EVERY DATA CENTER FACES THREE BIG CHALLENGES: SPACE, POWER, AND PERFORMANCE. And the pressure is on to deliver more for less. What's the answer? The Evergreen Data Center Framework from Intel and Tata Consultancy Services (TCS). This proven collection of today's newest technologies and best practices can help you design a data center infrastructure that doubles your performance—without expanding your power usage. It's built on techniques like virtualization and high-performance, power-efficient server platforms that can work together to transform your data center from power guzzler to lean, green IT machine.

For the whole story, attend "Evergreen Data Center from Concept to Completion" (see program guide for logistics) or contact Parviz Peiravi (parviz.peiravi@intel.com) or Sharad Isloor (sharad.isloor@tcs.com).

INSIDE SOAWORLD MAGAZINE

DOWNLOAD PRINT VERSION

SOAWORLD MAGAZINE
www.SOA.SYS-CON.com

You Can't Manage What You Don't Measure

10 Measures for Successful SOA Implementations

BY JERRY SMITH

The saying "you can't manage what you don't measure" has never been so applicable to software engineering as it currently is to SOA development. As organizations are struggling with the day-to-day development implications of SOA, business leadership is beginning to realize that success does not come without effective organizational measures. Cross-organizational measures are needed to bring transparency to those operational benefactors impacted by SOA's promise of agility and cost reduction. But what should you measure and are all measures equally important?

The answer to these questions is rooted in understanding the nature of the first part of the opening quote – you can't manage. We manage because we want to control and we need to control because we want to reduce risk to have outcome certainty. As such, every organization that implements SOA and every SOA implementation will need to identify and address the specific risks associated with the project. For example, if agility is an important driver for your SOA implementation, what does that mean to your organization and what specific metrics can be used to measure it?

There are several characteristics of measures and metrics, in general, that should be considered when identifying those specific to a SOA implementation. Are the measures predictive or reflective – that is, do they tell us about the future or the past? Are they theoretical or practical – considering theory tends to shed light on emerging areas, while practical tends to be more useful in narrowly defined situations? Are the measures internal or external – do they look internally into the services or externally towards their application? Are they objective or subjective – bearing in mind that objective measures tend to be less dependent on personal perception? Are they quantitative or qualitative? Quantitative measures lend themselves to classification, where qualification tends to be more descriptive. Who is the audience for the measure – are they meant for corporate, management, project, or developmental use? As you can see, there are lots of options and it is unlikely that any two organizations should have the same exact measurement system.

That said, there are a few measurement areas that should be looked into by any group and could be used as a starting point. I've broken those areas down into four core categories: Corporate Metrics, Management Metrics, Project Metrics, and Service Development Metrics. Across those categories are 10 measures that seem to get the most attention and relate directly to successful SOA implementations:

Corporate Metrics

1. Revenue Per Service

One of the most important measures for any company is revenue per employee. This number captures both the value an organization delivers, and the productivity achieved based on the value created by the employee base. Unfortunately, today there is no such universally accepted measure for services. Why? Mostly because SOA, up until now, has been dealt with as an engineering activity by technical people, which is normally tracked as cost in R&D. But this needs to change if SOA is going to be able to cross the business chasm.

Cost Effective SOA Management

small budget

Big Questions

mission-critical services

major Pressure

short timeline

reduced staff

A flexible Web Services Management solution for the enterprises that need the following capabilities

- Runtime Governance, Security & Policy Enforcement
- Visibility, Management and monitoring of the SOA environment
- Agentless visibility into SSL and non SSL Web service traffic
- Multiple Deployment options to fit Your environment
- Cost effective

Download a free trial at www.managedmethods.com

Managed Methods

www.managedmethods.com

For the same reasons that measuring revenue per employee is important, having a historical perspective on revenue per service will enable organizations to quantitatively assess both the value of the service and the productivity of the service oriented architecture through which it is delivered.

2. Service Vitality Index

The Service Vitality Index is the amount of revenue from new services over the last 12 months as a proportion of total service revenue. You can't simply throw more money at service orientation and expect a proportional return on the investment. But the problem is there are no standardized metrics for measuring the investment value of SOA. And the three most commonly used means for measuring development cost and impact – 1) percent of sales, 2) R&D headcount, and 3) number of patents acquired – are flawed. None of these measures are owned by R&D. The percent of sales and headcount are cost-driven, and the number of patents does not give you an indication of future value.

A better way to track the service impact on the business is to track the revenue and return on investment directly related to innovation. For example, to do this, Symphony Services combines a Service Vitality Index with Service-oriented ROI measurements.

The Service-oriented Vitality Index, or SoVI, is the ratio of revenue generated from a service (or services) over the last 12 months as compared with all other existing SOA revenue. This is a revenue view of service orientation (as is the case with revenue per service) verses a spend view. For example, assume a company has four SOA product lines that have a combined return of $100 million in total revenue. Three of these four products have been earning revenue for more than a year. And the fourth SOA product was released just under a year ago and contributed $5 million to annual revenues. This company's SoVI would be 5/100 or 5%. Healthy organizations should strive for a SoVI of 10%-20%, which compounded over time, will result in a 100% turnover in revenue from new products every five years.

The second and supporting metric is called the Service-oriented ROI. SoROI accounts for the money invested in service-oriented development. The SoROI is the cumulative before tax profits over "N" number of years from SOA-driven products divided by the cumulative product expenditures for that same period. This can be further enhanced by discounting both revenue and cost as a function of prevailing and forecasted interest rates. The SoROI allows you to compare overall SOA values independent of their size.

Combining both the Service-oriented Vitality Index and the SoROI provides a much clearer picture of a company's SOA health. Using these direct measures of service-orientation, companies can foresee the revenue implications of unfocused R&D years earlier than with traditional measures. To foresee is to be forewarned, a metaphor realized through the Service-oriented Vitality Index.

Management Metrics

3. Number of New Services Generated and Used as a Percentage of Total Services

Organizations with non-existent or poor SOA governance often see out-of-control service proliferation (high ratio of new services as a percentage of total services). Uncontrolled development teams often look to create new service after new service, not thinking about re-choreographing existing implementations to achieve the desired business value. Not only does this drive the total cost of service development up, but it also reduces the average revenue per service, indicating poor service development productivity.

4. Mean Time to Service Development (MTTSD)

If one of the benefits of SOA is business agility, then how do you measure that? MTTSD provides a statistical measure, along with a range of certainty, of the average time to stand up a service. Organizations new to SOA development (those with a low SOA maturity level) can see development time 10 times longer than those that have a managed or optimized SOA organizations (high SOA maturity level). Reducing MTTSD is a key benefit of SOA governance; that is, those activities related to the control of services in a SOA environment.

5. Mean Time to Service Change (MTTSC)

Just as with MTTSD, it is equally important to understand how long it takes to change a service. Business agility is measured both by creation and change. Services that are created quickly often lack the commercial rigor that stands the test of time. MTTSC can point out those services that have been poorly created and costing the organization in terms of effort and lost opportunity costs.

6. Service Availability

Service availability is the percentage of time a service is usable by the end user. It is a measure of the total delivery system from having defect-free services to operable data centers. Low serviceability (less than 99.9%) need to be dealt with immediately since they impact customer satisfaction. Triaging service orchestration and choreography, service discovery through registries, as well as service load balancing and failure, are three activities performed by organizations when service availability is unacceptable.

Project Metrics

7. Service Reuse

Development organizations have a tendency to rebuild what has already been built, a continuance of the "not invented here" syndrome. If business agility is based on the ability to stand up services quickly, then creating services quickly is based on reusing what you have. As part of an overall SOA governance process, measuring the degree to which you reuse services is critical for keeping development costs low and business agility high.

"There are several characteristics of measures and metrics, in general, that should be considered when identifying those specific to a SOA implementation"

8. Cost of Not Using or Stopping a Service

One of the least understood business costs in a SOA environment is the cost of not using or stopping an existing service. Not only are there obvious lost opportunity costs that can be measured in terms of revenue, but development costs as well. The end value delivered to users is often composed of many choreographed services, each delivering unique value. A well-designed SOA implementation has low shutdown or switching costs.

Service Development Metrics

9. Service Complexity as Measured Through Cyclomatic Complexity

The cyclomatic complexity of a service is the single measure that will determine if your service is testable and maintainable. Studies have shown that services with cyclomatic complexity greater than 50 are not testable and often result in 10%-20% more maintenance efforts than those services whose cyclomatic complexity is less than 10.

10. Service Quality Assurance

Service Quality Assurance is based on systems-level tests that examine the behavior of service-oriented use cases across possible choreographies [derived through service code coverage]. As in the case of code coverage, we look to determine how much of the code was executed during the course of testing (which is still important for developers), but here we look to address how much of the service use case was executed. In this case we look to develop systems-level tests that execute all service use cases across all possible choreographies. In the case of code coverage, we know that the cyclomatic complexity number tells us how many test cases are needed (e.g., a CC of 10 implies a minimum of 10 test cases). However, this is not necessarily the case with service coverage, because of the emergent behavior implications. In this case, we need to apply specific design of experiment processes and tests to statistical outcomes (e.g., we are 95% confident that the system behaves within the specified requirements parameters).

Conclusion

Several of the industry's most well-known SOA pundits, like David Linthicum and Joe McKendrick, have debated SOA metrics – which ones work, which ones don't, and why SOA needs to be measured in the first place. That debate will continue to take place as SOA adoption matures, and executive management grapples for some tangible evidence that SOA is "working." While perhaps the 10 measure outlined here are not the traditional measures companies think about when discussing SOA implementations, they do provide some level of transparency into operational issues that impact SOA agility and therefore serve as effective starting points to translate SOA into a successful – and measurable – endeavor. ■

About the Author

Dr. Jerry Smith, CTO of Symphony Services (www.symphonyservices.com), is a technology innovator and IT strategist focused on helping the company and its clients derive business value from the successful adoption and use of critical technologies. Jerry was previously CTO, vice-president of engineering, and acting CIO for IPR International, a technology services company specializing in the protection and preservation of electronic information. Prior to IPR, he was the senior vice-president and CTO of Systems and Computers Technology (SCT) and CTO of the professional services firm Semaphore. He also worked in management roles at Xerox, Sales Technologies, and KPMG.

Introducing SOA Design Patterns

The SOA community collaborates to produce a master pattern catalog dedicated to SOA

BY THOMAS ERL

Originally inspired by techniques used to design buildings and cities, and popularized by the Gang of Four during the mainstream emergence of object-orientation, design patterns have seen us through the various shifts in architecture, technology, and, of course, design.

Pattern catalogs have periodically emerged, one building on the other, and each revealing a set of problem-solving techniques and providing invaluable insights as to how and when those techniques should be used to help us attain our design goals.

SOA has its own history, having risen out of a haze of ambiguity to establish itself as the basis of a distinct and maturing distributed computing platform with a distinct and ambitious design paradigm in its own right.

And now, finally, these worlds converge. SOA and service orientation (and surrounding technology platforms) have matured to the extent that proven design practices have surfaced for use by the masses. Subsequent to years of research, reviews, and validation, this body of work has been formally documented as a comprehensive collection of over 90 SOA design patterns.

One of the most intriguing aspects of the SOA design pattern catalog is its breadth. We have patterns providing design techniques that range from adjusting minute validation logic in a service contract to design strategies that help us structure pools of services across an entire enterprise.

This scope is indicative of the enterprise-centric focus of service-oriented computing in general. When carrying out an SOA initiative, we need to pay attention to many design details with every service we deliver, while always keeping the big picture in our sights. Design patterns support us in maintaining this balance by helping us overcome common obstacles that have historically inhibited or even derailed SOA project plans.

Each pattern is like a piece of wisdom resulting from the trials and errors of pioneers and the sweat and tears (and therapy) that sometimes accompanied those early SOA project experiences. So, please, a moment of silence for those who have suffered so that we can now benefit...

All right then, without further ado, let's introduce the SOA design patterns.

Patterns in a Service-Oriented World

Service-oriented computing has a specific set of strategic goals and benefits associated with it. Most of these goals, such as increasing agility and ROI, are well known, as is the fact that to attain these goals, you need to design your solutions by following service orientation, a distinct design approach tailored to support service-oriented computing.

There's a close relationship between the service-oriented architectural model and the service-orientation design paradigm. It is through the application of service-orientation design principles that you end up creating software programs that are legitimately "service-oriented." When you implement SOA as a technology architecture you establish an environment that is conducive not just to enabling the creation of effective service-oriented solutions, but also to

enabling the effective long-term governance and evolution of the individual services that can be composed and recomposed to comprise these solutions.

Design Patterns and Architecture Types

Each SOA design pattern provides a design solution in support of successfully applying service orientation and establishing a quality service-oriented architecture. Therefore, to better understand how and to what extent individual SOA design patterns can be applied, SOA as an architectural model itself needs to be broken down into the following types, each of which represent a common "scope of implementation":

- **Service Architecture** – The architecture of a single service.
- **Service Composition Architecture** – The architecture of a set of services assembled into a service composition.
- **Service Inventory Architecture** – The architecture that supports a collection of related services that are independently standardized and governed.
- **Service-Oriented Enterprise Architecture** – The architecture of the enterprise itself, to whatever extent it is service-oriented.

In a typical enterprise, these architecture types are very much interrelated, yet each requires individual design attention and documentation.

About the Patterns

This article is roughly organized according to these architecture types and related patterns.

SOA design patterns collectively form a master pattern language that allows patterns to be applied in different combinations and sequences. There are also compound patterns that are comprised of multiple individual design patterns. (For example, the Enterprise Service Bus and orchestration both represent compound design patterns.)

SOA design patterns are not specific to any particular vendor platform or business industry; they are simply design techniques that help overcome common obstacles to achieving the strategic goals and benefits associated with SOA and service-oriented computing.

The remainder of this article highlights key design patterns while referencing a cross-section of others. Not all mentioned patterns are explained, but descriptions are freely available at the SOA patterns community site (www.soapatterns.org).

Service Inventories

When building services as part of an SOA project, there is an emphasis on developing them as standalone programs that are expected to be flexible and robust so that they can be readily reused and composed. Service-oriented design has therefore been heavily influenced by commercial product design techniques to the extent that a service is delivered as a black box somewhat comparable to a software product. Inspired by commercial terminology, a collection of services for a given segment of an enterprise is referred to as a "service inventory." And, similarly, the technology architecture that supports this collection of services is referred to as the "service inventory architecture."

What's the difference between a service inventory and a service catalog? The same manner in which an inventory of products is documented with a product catalog, an inventory of services is documented with a service catalog. It's therefore still appropriate to refer to a collection of services as the service catalog; however, when applying design patterns and defining the actual concrete architecture, terms like "service inventory" (or even "service pools") tend to work better.

Patterns for Collections of Services

A service inventory represents a collection of independently standardized and governed services. As shown in Figure 1, the services you deliver for a given service inventory are standardized and designed according to service orientation so that they become intrinsically interoperable. This then allows you to draw from this pool of services to assemble and augment service compositions repeatedly.

Inventory Boundary Patterns

One of the biggest decisions a project team faces when starting an SOA initiative is determining the appropriate scope of a service inventory. The Enterprise Inventory and Domain Inventory design patterns help address this decision point by providing alternative approaches.

The goal of the Enterprise Inventory pattern is to establish an enterprise-wide service inventory. The end result of achieving this pattern is considered desirable because it enables you to build all of your services according to the same design conventions to ensure consistent and widespread inter-service compatibility. It further guarantees that all of the services will be owned and evolved by the same group or department, which is the ultimate in centralized governance.

Although ideal, this approach is often not realistic, especially for larger organizations. It can raise various issues, including time and budget constraints, cultural and political concerns, and order of magnitude considerations (especially in relation to the long-term growth and governance of the inventory). These issues can introduce risks and problems that outweigh the benefit potential of applying this pattern.

This is the reason the Domain Inventory pattern has become so popular. It advocates an approach whereby the enterprise is divided into segments (domains), each of which represents a meaningful cross-silo scope. Often, the boundary of a domain inventory architecture is aligned with a business domain (such as accounting or claims). Services delivered into this architectural boundary are subject to the same design standards and governance practices, allowing them to be evolved independently from neighboring domain inventories in the same enterprise.

Although the use of this pattern can introduce the need for cross-domain data model and protocol conversion (as per the Schema Transformation and Protocol Bridging patterns), there are additional patterns (such as Cross-Domain Utility Layer, Dual Protocols, and Inventory Endpoint) that help reduce this impact.

Inventory Structure Patterns

Regardless of its scope, within the boundary of a service inventory, certain design patterns are applied to ensure a consistent structure in support of service orientation. For example, Logic Centralization positions reusable services as the sole or primary contact points for the logic they represent. This is further supported by the Service Normalization pattern that fosters service autonomy by reducing the amount of functional overlap between individual service boundaries to establish more of a "normalized" inventory.

The Service Layers pattern (and related, specialized layer patterns) can be used to further organize a service inventory into a set of logical layers, each of which is based on a different classification of service.

Note: *Just a reminder that all of these structural patterns are only applied within the boundary of an inventory architecture. This means that, if you are working within the confines of a domain inventory, these patterns will not be applied on an enterprise-wide basis.*

To support and extend the structure of a service inventory archi-

tecture, various other design patterns can be applied. Some (like Canonical Schema and Canonical Transport Protocol) help standardize the services within the inventory boundary to foster native interoperability and composability, while others (like Process Centralization and Rules Centralization) can be selectively used to leverage established product platforms that support the centralized management of business process logic and business rules, respectively.

Yet another dimension to inventory architecture design is the centralization of service contract-related logic. The creation of redundant schema and policy content can be addressed by the Schema Centralization and Policy Centralization patterns, each of which establishes a separate data representation layer (one layer for data models, the other for global and domain-level policies) that supports the primary service contract layer.

There are many more specialized patterns that are applied to an inventory architecture to solve common problems related to resource management, state management, quality of service, security, and communication.

Patterns for Service Design

Each service exists as a standalone software program, autonomous yet still fully geared to participate in larger service aggregations. When designing a service architecture, numerous challenges can arise, especially when shaping this architecture according to service-orientation design principles, such as Service Statelessness and Service Loose Coupling. Figure 2 provides an abstract glimpse of service architecture design patterns that are applied at the service architecture level.

Agnostic Logic and Non-Agnostic Logic

The term "agnostic" originates from the Greek word "agnostos," which means "without knowledge." Therefore, logic that is sufficiently generic so that it is not specific to (has no knowledge of) a particular parent task is classified as agnostic logic. Because knowledge specific to single-purpose tasks is intentionally omitted, agnostic logic is considered multi-purpose. On the flipside, logic that is specific to (contains knowledge of) a single-purpose task is labeled as non-agnostic logic.

Another way of thinking about agnostic and non-agnostic logic is to focus on the extent to which the logic can be re-purposed. Because agnostic logic is expected to be multi-purpose it is subject to the Service Reusability principle with the intention of turning it into highly reusable logic. Once reusable, this logic is truly multi-purpose in that it, as a single software program (or service), can be used to automate multiple business processes.

Non-agnostic logic does not have these types of expectations. It is deliberately designed as a single-purpose software program (or service) and therefore has different design priorities.

Fundamental Service Design

At the most basic level of service design, the established "separation of concerns" theory needs to be applied as part of a process whereby a larger problem is decomposed in a way that we can clearly identify how corresponding solution logic should be partitioned into services. To accomplish this, a series of base patterns have emerged to form a fundamental pattern language that can serve as the basis of a primitive design process.

Examples of these basic patterns are Functional Decomposition and Service Encapsulation, both of which help determine what type of logic does and does not belong in a service. Additional patterns, like Agnostic Context and Non-Agnostic Context provide criteria that help determine whether certain kinds of logic are deemed sufficiently agnostic to be put into a reusable or multi-purpose service (see Figure 3).

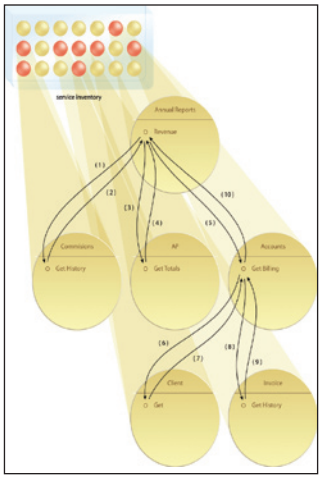


Figure 1

Service Implementation Design and Governance

A key pattern frequently applied to the initial design of service architecture is the Service Façade. Inspired by the Façade pattern created by the Gang of Four, this pattern wedges a component between the service contract and the core service logic to establish a point of abstraction. This is really nothing new, but it does establish an architectural foundation that can be leveraged by other more specialized SOA patterns.

For example, Service Decomposition (one of the service governance patterns) allows a coarse-grained service to be split up subsequent to its deployment. Additional governance patterns, such as Proxy Capability and Distributed Capability (Figure 4), help ensure that this decomposition of one service into two or more does not impact the contract (technical interface) of the original service, thereby also avoiding impact to that service's consumer programs.

Other patterns, such as Legacy Wrapper, Redundant Implementation, Service Refactoring, and Service Data Replication, can be selectively used to address various implementation and scalability-related requirements, while maintaining the overall flexibility required for services to be repeatedly composed and extended, as required.

"Capability" v "Operation" v "Method"

Service contract design patterns avoid the use of terms like "operation" and "method" because they are generally associated with a specific implementation platform (like components or Web Services, respectively). The term service capability is used to represent a specific function of a service that it can perform and through which it can be invoked. Service capabilities are generally expressed by the service contract. This term is also common during early service modeling stages, when service candidates are conceptualized prior to determining how they will be physically designed and developed.

Service Contract Design and Governance

A key design pattern that helps increase the longevity of service contracts (to postpone versioning requirements) is Contract Denormalization. This pattern essentially explains how capabilities with overlapping functional scopes can be added to a service contract without negatively impacting service or inventory architectures. This results in the contract's technical interface exposing similar functions at different levels of granularity, allowing the same contract to facilitate the requirements of different types of consumers.

Alternatively, multiple groups of consumers can be accommodated by the Concurrent Contracts pattern that describes how entirely separate contracts can be created for the same underlying service logic. Security restrictions or a need to split up policy alternatives for reasons of governance can also result in the requirement to employ multiple service contracts. This pattern also benefits from the existence of a service façade that can be positioned to coordinate

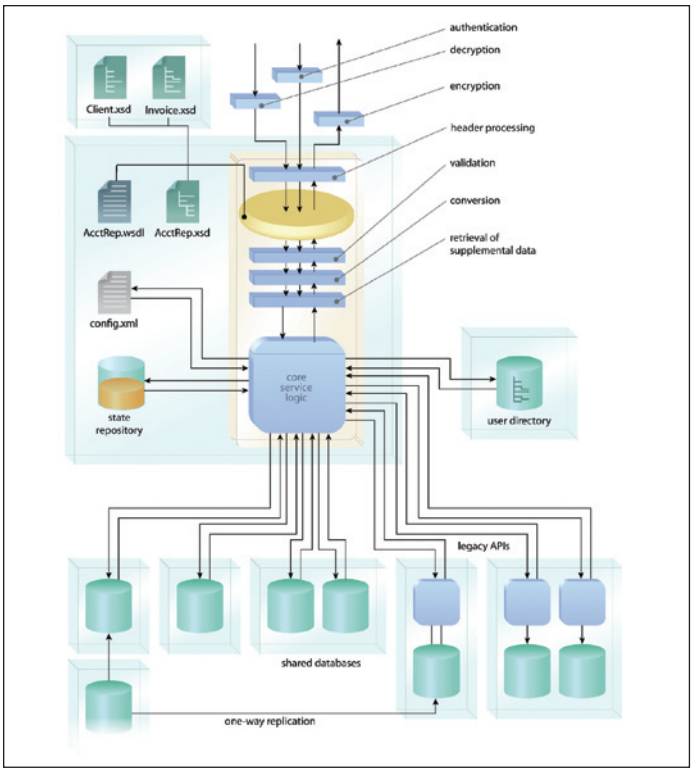


Figure 2

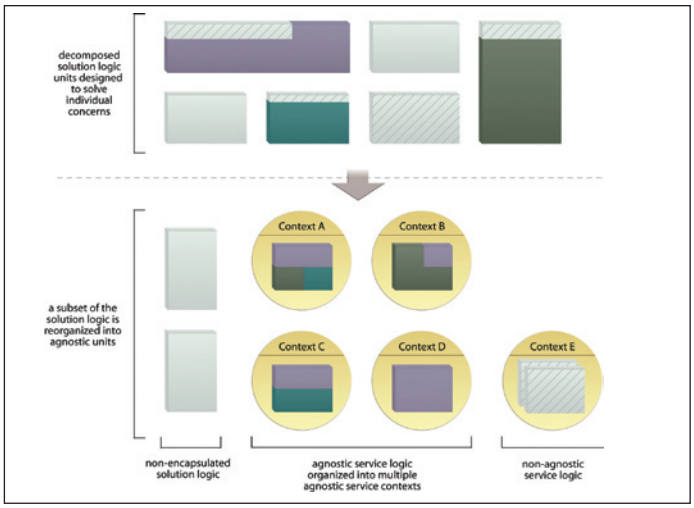


Figure 3

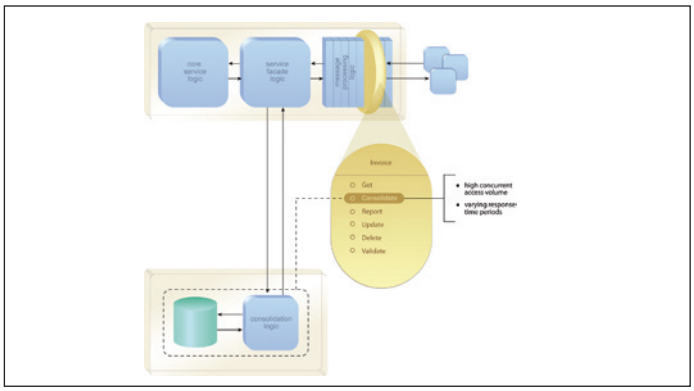


Figure 4

incoming and outgoing data exchanges across multiple contracts but with a single body of core service logic.

Part of these governance patterns includes various contract versioning design techniques to minimize the impact of having to introduce new contract versions or support multiple versions of the same contract. Some governance patterns are reactive because they help solve unexpected governance-related design issues, while others are preventative in that they can be applied prior to the initial deployment of a service in order to build in additional flexibility that allows the service to be more easily governed and extended over time.

Patterns for Service Composition and Communication

As an aggregate set of services, a service composition establishes its own unique architecture encompassing the individual service architectures of the composition members and introducing additional design requirements focused on inter-service communication and runtime activity management. It's therefore no surprise that many design patterns have emerged to address composition-related design issues.

There are several key design patterns that establish the mechanics behind inter-service communication. Due to the popularity of building services as Web Services, several of these design patterns are based on messaging, and some are focused solely on asynchronous messaging. For example, the Asynchronous Queuing design pattern establishes a central queue to allow services to overcome availability issues and increase the overall robustness of asynchronous data exchange.

The marriage of SOA and EDA has resulted in the Event-Driven Messaging pattern that enables publish-and-subscribe functionality between services over extended periods. This can go hand-in-hand with the use of the Service Agent pattern that introduces an additional event-driven dimension into composition architecture by allowing you to defer various types of cross-cutting logic to transparent agents that intercept and forward messages at runtime (Figure 5).

The Intermediate Routing pattern takes this a step further by providing intelligent agent-based message routing that can also help increase the overall scalability of services and compositions.

Very much related to supporting service messaging and the hosting and execution of service compositions as a whole is the Enterprise Service Bus compound pattern. As shown in Figure 6, this pattern establishes an environment comprised of multiple other patterns, such as the aforementioned Intermediate Routing and Asynchronous Queuing patterns in addition to the Broker pattern (that in itself is also a compound pattern that represents a set of individual transformation patterns).

Note that there are additional design patterns associated with the Enterprise Service Bus compound pattern, several of which are classified as optional extensions to a core model. Other design patterns that tie into service composition architecture include Cross-Service Transaction, Composition Autonomy, Reliable Messaging, and Agnostic Sub-Controller.

Working with SOA Design Patterns

Before we conclude, here are just some guidelines for getting the most out of SOA design patterns.

View Patterns with a Strategic Context

One of the greatest expectations of SOA initiatives is for the IT enterprise as a whole to become a more streamlined, responsive,

and efficient part of the overall organization. Regardless of whether you are applying service-orientation design principles or SOA design patterns, it's important to keep these strategic goals in mind because they help you understand why there are certain priorities, preferences, and sacrifices that need to be made in order to deliver truly valuable service-oriented logic.

Take Governance into Consideration

Many IT professionals focus on governance as a project lifecycle phase that begins after services and service-oriented solutions have been deployed and are in use. However, governance itself is also very much a design-time consideration. Just about every decision point you face when designing services, service compositions, and service inventories will have governance-related implications. Understanding the long-term consequences of design decisions in advance will help you choose and combine design patterns wisely to minimize the governance impact of anything you end up building. This is a critical consideration because it's through reduced governance burden that you can truly achieve the strategic goals of service-oriented computing.

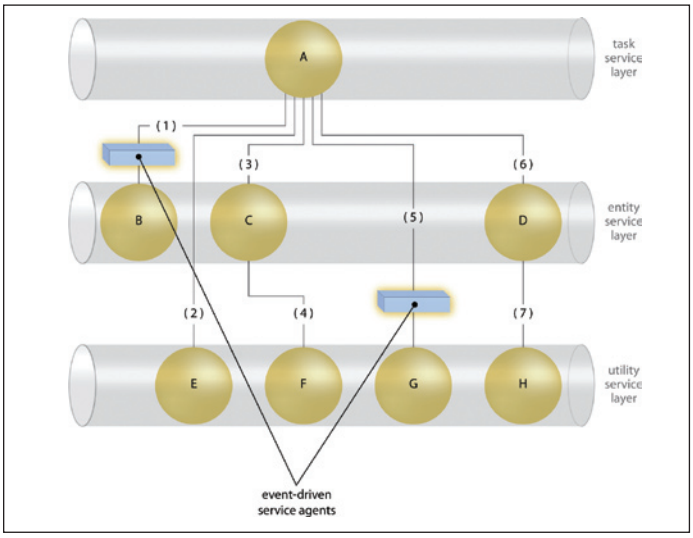


Figure 5

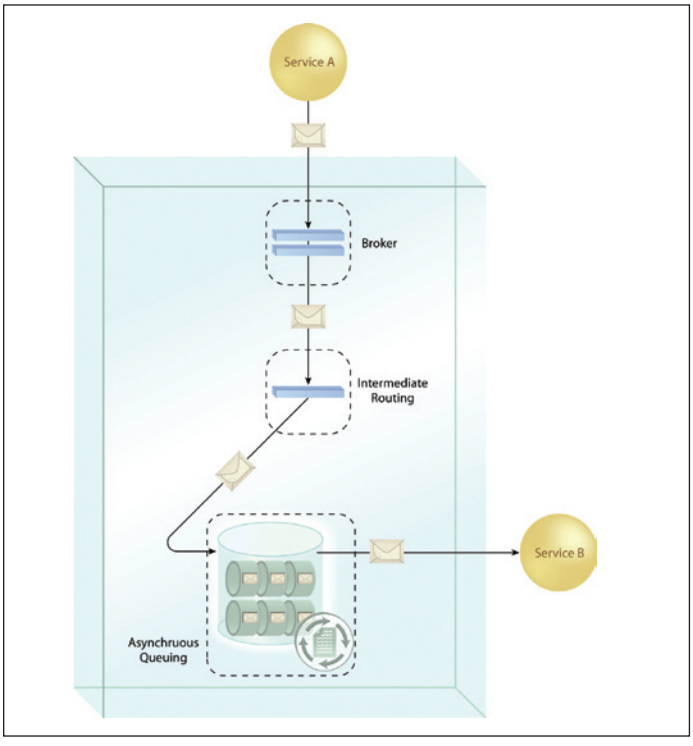


Figure 6

Patterns are Applied in Measures

As with service-orientation design principles, SOA design patterns can (and often must) be applied to various extents. This means that for any given pattern, you are not faced with an all-or-nothing proposition. Some patterns need to be applied in a limited capacity, while with others it makes a whole lot of sense to maximize their application to whatever degree possible. Either way, it's always good to keep in mind that the benefit promised by a given pattern is generally tied by the measure to which it is applied.

Acknowledge that Some Patterns Are Evolutionary

For those of you with an OOD or EAI background, several of the design patterns mentioned here probably looked familiar. Service orientation owes much of its existence to past design paradigms. What makes it distinct are the parts of those paradigms that were not included, combined with new design considerations and techniques that are particular to the enterprise-centric focus of service-oriented computing.

The same goes for SOA design patterns. While some of the patterns in this catalog are based on (or even derived from) patterns in previously published pattern catalogs, they are documented in the specific context of SOA and service orientation. Other SOA design patterns introduce brand new design techniques that complement and build on the derived patterns.

Patterns Relate to Patterns

Most SOA design patterns have numerous relationships with each other (see Figure 7). For example, one pattern may have a dependency on another, or its application may influence another, or perhaps it is simply part of a larger compound pattern. With an understanding of inter-pattern relationships, you can avoid potential conflicts (some of which don't reveal themselves until later). Additionally, you can fully leverage the SOA design pattern catalog as a full-fledged pattern language that supports the application of patterns into various creative pattern sequences.

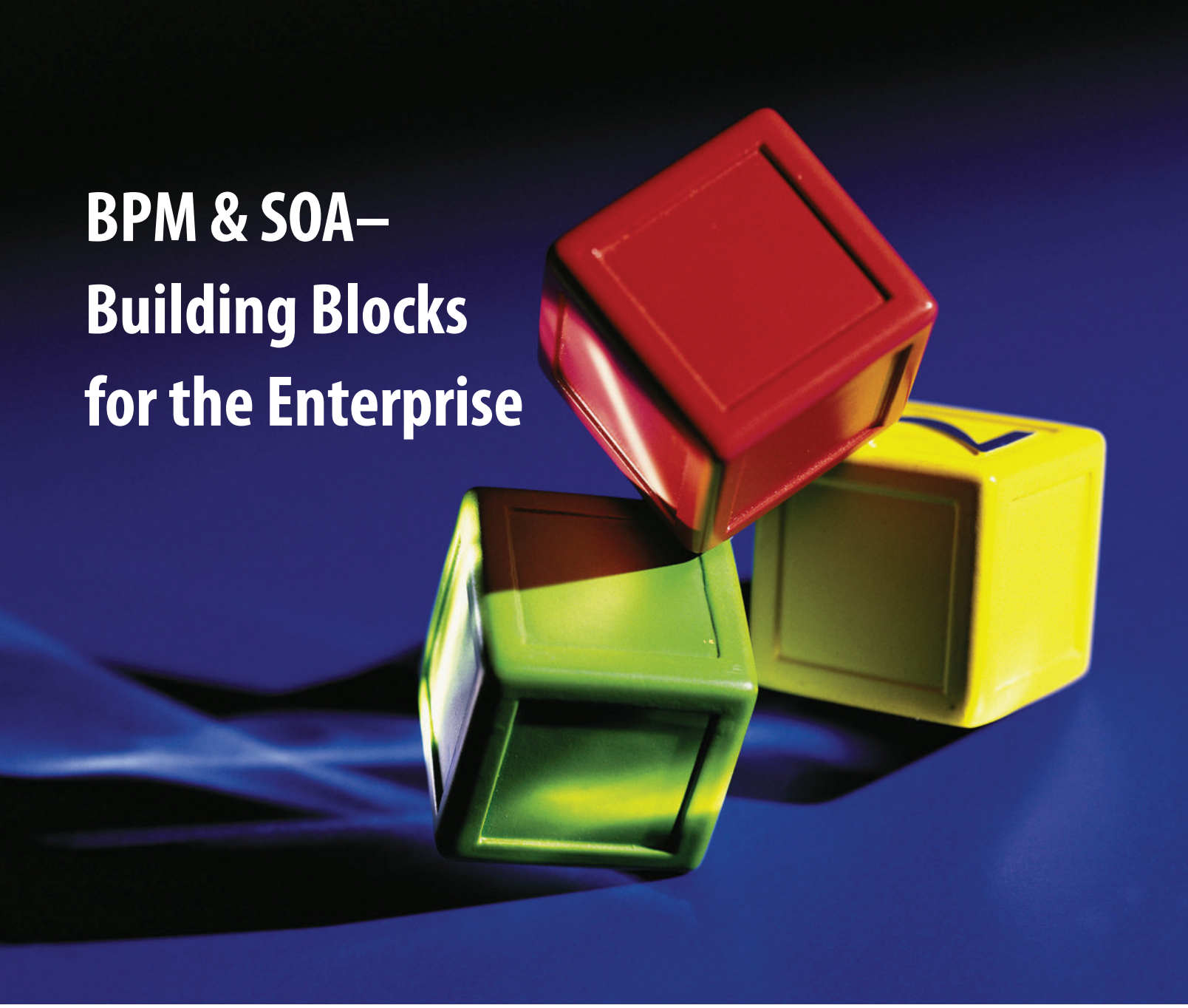
Patterns Relate to Principles

Each SOA design pattern affects and influences the application of one or more service-orientation design principles. By identifying these relationships, you can leverage specific patterns when working with the design principles individually. There are also adverse relationships, where the results and trade-offs of some patterns negatively impact the goals of a design principle. Again, knowing these relationships in advance will help you make educated design decisions. (For more details regarding service-orientation design principles, visit www.soaprinciples.com.)

Not Every Pattern Is Suitable for Everyone

As mentioned at the beginning of this article, this collection of SOA design patterns was produced as a result of numerous (successful and failed) projects, on-going research, and documented lessons learned. The focus of each pattern is on providing a solution to a common problem. Just because a given problem was common

BPM & SOA– Building Blocks for the Enterprise



Contrary to popular perception, business and IT can work together. And BPM and SOA are the building blocks that make this happen. SOA enables asset re-use and BPM accelerates SOA implementations with greater business-IT synergies.

Interstage® Business Process Manager is an award-winning BPM Suite. CentraSite™ is a completely standards-based Registry-Repository.

For more information on Fujitsu's BPM and SOA offerings, please visit us at

www.fujitsu.com/interstage

or contact

Interstage@us.fujitsu.com

FUJITSU

elsewhere does not mean it will apply to your environment. Also, you always have the option of taking ideas from these design patterns and then deriving your own distinct design techniques.

A Final Word

We've really just skimmed the surface with this exploration of SOA design patterns. The pattern catalog has nearly 100 design patterns, the majority of which we haven't been able to mention in this article. Examples of other types of patterns that have been documented include those that address grid computing and balanced stateful service design, REST-based communication and service design, security technology, attack prevention and perimeter protection, versioning of services contracts and related schemas and policies, dynamic messaging, runtime compensation, the use of binary attachment technologies, various transformation requirements, and others as it evolves.

All of these patterns are intended to help you make a success out of an SOA initiative. How success is measured in the SOA world often comes down to how well the service-oriented environment you create represents the business of your organization – and how well it continues to stay in alignment with the business.

Figure 8 illustrates the never-ending progress cycle that continually transpires between business and IT communities. The result of this dynamic is constant change, and if we had to choose the one thing responsible for inspiring service-oriented computing in the first place, it would be that word "change."

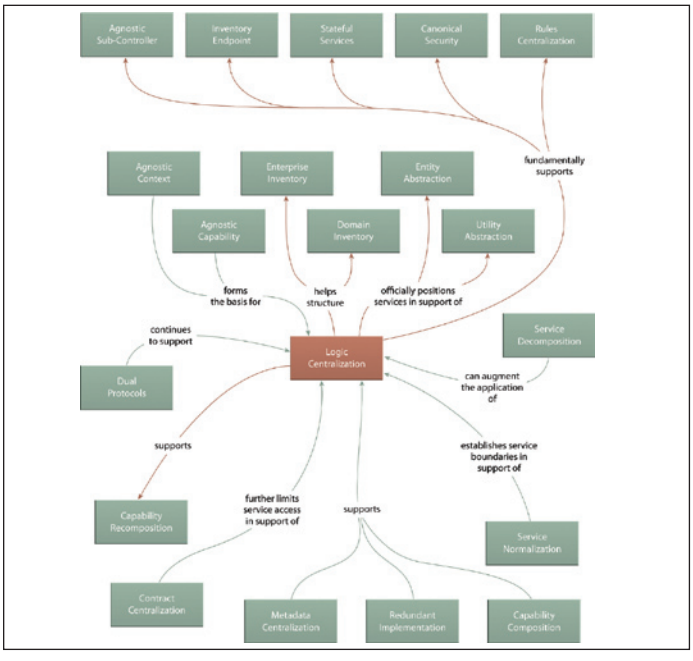


Figure 7

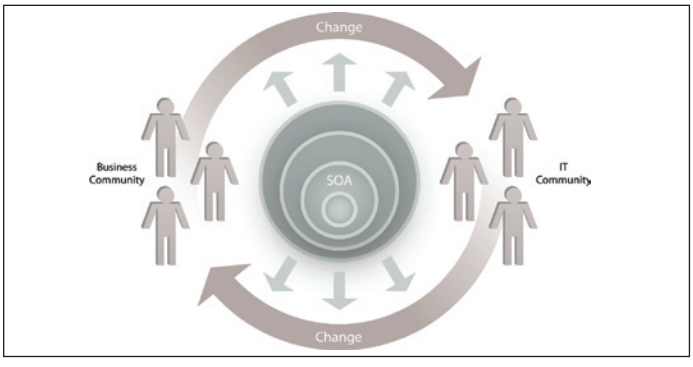


Figure 8

Ultimately, service orientation and the implementation of the different service-oriented technology architecture types (inner circles in Figure 8) support the two-way dynamic between business and IT communities (outer circle in Figure 8), allowing each to repeatedly introduce or accommodate change. SOA design patterns help establish and strengthen SOA architectures in support of service-orientation, a paradigm specifically geared toward facilitating this cycle of constant change.

From the minute you design them to the months and years later when you extend or build on them, your services, your compositions, and the inventory architectures in which they all reside must be equipped to deal with change. Only this way can your technology enterprise evolve in tandem with the business.

To establish such an environment requires a different approach to design. Service-orientation provides an overarching design paradigm that defines this approach, but it's the SOA design patterns that help you deal with the nitty-gritty. Regardless of whether you're applying design principles or patterns, be sure to keep the strategic vision of SOA in your sights, because it is only by the extent that you attain and maintain this vision that the success of your SOA initiative can be truly measured.

Acknowledgements

This design pattern catalog is very much the result of an international community effort. Experts throughout the industry have contributed patterns to this catalog, and patterns were subjected to an open community review at soapatterns.org in which hundreds of members from standards organizations, major SOA vendors, and the patterns community itself participated.

The author would like to especially thank (in alphabetical order): Mohamad Afshar, Raj Balasubramanian, Frank Buschmann, David Chappell, Martin Fowler, Richard Helm, Kelvin Henney, Jason Hogg, Gregor Hohpe, Ralph Johnson, Mark Little, Brian Loesgen, David Orchard, Chris Riley, Thomas Rischbeck, Robert Schneider, Arnaud Simon, Bobby Woolf, and Olaf Zimmermann. The author would also like to thank Prentice Hall for making advance copies of the book manuscript available for distribution during the review phase.

This article contains diagrams from the upcoming book "SOA Design Patterns", a title in the "Prentice Hall Service-Oriented Computing Series from Thomas Erl" to publish in Fall, 2008 (ISBN: 0136135161, Prentice Hall, Copyright 2008 SOA Systems, Inc.). The book is currently available as part of the Safari Books Online "Rough Cut" program. For more information, visit www.soabooks.com and www.soapatterns.com.

About the Author

Thomas Erl is the world's top-selling SOA author and Series Editor of the "Prentice Hall Service-Oriented Computing Series from Thomas Erl" (www.soabooks.com). With over 90,000 copies in print worldwide, his books have become international bestsellers and have been formally endorsed by senior members of major software organizations, such as IBM, Microsoft, Oracle, BEA, Sun, Intel, SAP, Cisco and HP. His most recent title ("SOA Principles of Service Design") was released in 2007, and his fourth and fifth titles ("Web Service Contract Design & Versioning for SOA" and "SOA Design Patterns") were jointly authored with industry experts and are scheduled for publication this year. Thomas is also the founder of SOASchool.com, an organization providing industry-recognized SOA training and certification.

The Number One SOA Event
Comes to New York Colocated with the Virtualization Conference & Expo
June 23-24, 2008 at The Roosevelt Hotel

Service-oriented architectures (SOAs) have evolved over the past few years out of the original vision of loosely coupled web services replacing constrained, stovepiped applications throughout enterprise IT. Every major enterprise technology vendor today has developed its own SOA strategy, supported by innumerable mid-size companies and start-ups offering specific SOA aspects or entire solutions. This explosive growth in SOA technology is in response to a global demand--IDC estimates that spending on SOA services alone will grow from $8.6 billion to more than $33 billion by 2010.

SOA World Conference & Expo 2008 East brings together the best minds in the business to New York for a two-day conference that offers comprehensive coverage of SOA and what it means to enterprise IT today. As Zapthink analyst Jason Bloomberg has noted, "SOA involves rethinking how the business leverages IT in many various ways." Attend SOA World Conference & Expo 2008 East and learn from more than 100 speakers about how SOA is transforming business--and the way IT and business managers think about their businesses, processes, and technology.

The colocation of SOA World Conference & Expo 2008 East and Virtualization Conference & Expo delivers the most relevant content to IT and business. Conference attendees will be able to choose from four great tracks at this year's event. Mix and match all you want, or slot into your favorite topic for the duration! The tracks include:

- Web 2.0/AJAX and SOA
- Interop Standards and Services
- Real-World SOA
- SOA Technology
- Virtualization
- Specially Selected Hot Topics

Who Should Attend?

CEOs and CTOs, senior architects, project managers, Web programmers, Web designers, technology evangelists, user interface architects, consultants, and anyone looking to stay in front of the latest Web technology.

REGISTER TODAY AND SAVE!

www.soaworld2008.com

Does Your Business See Value in SOA?

The value of SOA is difficult for the business functions within organizations to understand

BY NICOLAS ROBBE AND BRETT STINEMAN

SOA – a simple three-letter acronym that causes a lot of confusion. What exactly is SOA and why is it at the forefront of IT initiatives around the world? Interestingly, if you ask your peers what SOA means, you will get a wide variety of responses. Some people will talk about modular applications or reusable components, while others will describe loose coupling or standards-based integration. Other people will describe it as the next generation of IT infrastructure, and invariably you will get the most stripped-down description, "Web Services." Of course, SOA incorporates all of these concepts, but what is missing in these definitions is the value that SOA brings to organizations. In other words, the translation of IT benefits to business benefits has been lost.

The value of SOA is difficult for the business functions within organizations to understand. Why? Because IT tends to overly focus on the underlying infrastructure aspects without exposing the value in ways that those outside of IT can easily see and experience. Think of an older house with knob-and-tube electrical wiring. Replacing the wiring can provide a number of advantages in terms of capacity, grounding, and safety, but it makes the most sense to replace it when there is an associated tangible benefit. It is at this point where the value of the infrastructure improvement can be easily understood. In the same way, SOA initiatives need to tie a business benefit to the important work that is taking place. The best way to show this value is to demonstrate business agility – the ability to respond more quickly to changes in the business environment.

Business Agility – The Value Driver for SOA

"Agility" is a concept that is often used to describe the benefits of SOA, so it is important to step back and look at why and how organizations are seeking to accelerate and easily adapt to changing conditions. First off, in many industries the frequency of change is increasing. The reality of the world today is that technology makes it easier to compete across multiple geographies, while at the same time it creates more complexity due to varying market conditions, regulations, consumer preferences, and competitors. So, from a systems standpoint, there is a need to be able to manage changes that are constantly taking place more quickly and easily. While some of these changes are related to integrating disparate applications and data sources, the most important benefits can be derived from having applications that are smarter and easily adaptable to changing business needs. In fact, applications must be able to deal with the following agility pressure points:

- Making more decisions in real-time, closer to the point of sale, with the maximum degree of decision automation when there is sufficient information about the customer, transaction, or process
- Making more complex decisions based on more data points and interrelated conditions
- Providing customers with more personalized service or offers based on the context of the current transaction and prior interactions
- Enforcing decision consistency, including across distribution channels, geographies, or business units, when regulations or policies require that people or processes follow specified standard operating procedures
- Enforcing decision compliance based on local regulatory environments for a specific customer or transaction

Looking at these requirements, it is easy to see that there are a number of opposing forces working (more consistency versus increased personalization, more automation versus the need to handle more complexity). But at the core of all of these various pressure points is the need to manage the business decision variations that are at the heart of how applications operate. If you want to improve business agility, you need to be able to build services that allow decisions to be defined more easily, controlled and maintained based on the ever-changing business environment.

Making Decisions a Central Part of a SOA

If we accept the premise that improving the management of decision logic in applications is a key factor to enabling business agility then decisions need to be handled in ways that allow for the SOA principles of modularity, reusability, and ease of integration. Traditionally, decisions within applications are defined as rules that are hard-coded directly in the application – rules are defined as "if-then-else" conditions, and these can become very long and difficult to define for complex decisions. Also, as decisions change over time, the embedded rules must be located in the code and modified. It is easy to see how this approach impedes business agility, especially as the frequency of change increases.

At this point, let's introduce a different way of dealing with decisions. Business Rules Management Systems (BRMS) allow decision logic to be separated from application code so that decisions can be managed and maintained more easily and multiple methods of defining complex decision logic will be allowed. BRMS allows decision logic to be defined and understood in three ways:

- The set of underlying conditions (expressed as if-then-else, decision tables, decision trees, or scorecards)
- The associated decision metadata (input/output parameters, rule service packaging, effective/expiration dates, etc.)
- The organizational hierarchy of where a decision is used and its interrelationship with other decisions (aka rule flow)

In this way, decisions can be managed as information assets, allowing them to be understood in the context of the overall business objectives, not just as isolated conditional code.

BRMS is essential to SOA because it enables business agility and provides demonstrable value to IT modernization initiatives. Business policy is separated from core application code so that it can be easily understood in terms of its context and usage and it can be easily changed and deployed as needed, so that rules can be shared and reused in and across applications.

Let's take the example of an insurance carrier. Underwriting insurance policies is typically handled by multiple systems that manage various types of policies, such as auto, home, commercial, and others. But in these systems, there may be common rules for handling corporate procedures, governmental regulations, fraud analysis, and more. At the same time, each business unit has unique rules such as eligibility restrictions, pricing, etc.

By using BRMS, the decisions that define how policy submissions are evaluated and approved can be defined and stored in a rules repository containing both the instructions and context of the rules. Individual rules or associated sets of rules can be packaged as services (known as transparent decision services or TDS), which are invoked by the underwriting system at specified points. The underwriting system passes the TDS a set of input parameters that are then processed against the rules in runtime through a rule execution server. The output of the TDS execution is sent back to the underwriting system so it can proceed to the appropriate next action, event, or process activity.

Now, let's consider that a governmental regulation is added in one of the localities where this insurance carrier operates. By running a simple set of queries, any affected rules can be identified and updated, and the associated services can also be identified, tested, and redeployed if needed (e.g., if a new input parameter is required). If the rule requires a change to take place on a certain date then this condition can be easily defined as part of the rule metadata and the same TDS can be used both before and after the effective date.

A New Model for Business and IT Collaboration

Today, companies across many industries have successfully enabled actuaries, product marketers, and other business groups to manage decision logic with minimal IT involvement using BRMS. How did they do it? What can we learn from their experience?

At the heart of successful BRMS projects is a set of work and communication practices that defines a new form of collaboration between business and IT. This collaboration relies on three key infrastructure components: a shared language between business and IT, a set of tools for modeling and communicating decision logic, and a governance process that ensures safe collaboration.

Building a Common Language Between Business and IT

The ability for business and IT to communicate clearly with each other on how business decisions should be automated in operational systems is paramount to improving agility. Over the years, decision logic has been overlooked as a business asset and, as a result, it has been underserved from a technology perspective. Too often, the language of decisions is COBOL, C++, or Java code. As companies grow into new regions, diversify their channels or launch new products, the code has evolved to follow new regulations, tax laws, or price adjustments. Eventually, this code becomes so complex that the business rules are hard to find and comprehend, making changes slow and error prone. More importantly, the participation of business users is nearly impossible.

With BRMS, the logic of a decision service is modeled and maintained using a combination of structured English-like statements and high-level modeling abstractions based on a vocabulary that the business can understand. The structure of a pricing policy is most effectively expressed and communicated as a decision table rather than as nested "if-then" Java statements. The complexity of risk scoring decisions is best understood as a set of scorecards. Once defined, those artifacts can then be automatically parsed and executed by a runtime engine. Now business stakeholders can interact and contribute to the definition, modeling or validation of application logic. Changes can be turned around quickly. Risk of misinterpreting the company policies and strategies is greatly reduced.

The Right Tool for Each Stakeholder

Each stakeholder involved in maintaining business rules needs tools that correspond to their skill level and the context in which they perform this maintenance task. Too often, subject matter experts are expected to interact and model their knowledge-using tools, which are primarily designed with IT users in mind. For business stakeholders to be involved in maintaining business rules, their management environment must be more focused than traditional development tools. They also need to provide a safe environment for experimentation and anticipation of the impact of a business rule change. For instance, providing a set of common business rules templates, which restrict the scope of behavior changes to common business contexts, is a safe and easy way for business users to introduce new logic into a decision service. In addition, templates enable QA to rapidly generate new test cases for any given rule changes. Typical rule authoring tools in BRMS are based on a set of Web-based graphical editors or, even better, based on Microsoft Office.

Conversely, developers, QA, and system administrators need to understand the effect of decision logic changes on the service itself

and its service-level agreements. In particular, IT needs the proper tools to understand and manage dependencies with the underlying data models, the testing and deployment processes, and the monitoring of the service performance. One of the keys to ensuring proper integration of a business-lead decision management process into the software application maintenance process is for the IT tools to be based on standard development environments (e.g., Eclipse and MS Visual Studio) and administration tools.

The level of involvement of business experts in the maintenance of decision logic depends, of course, on many factors, such as the frequency of changes, expected turnaround time, and engagement procedures between IT and business groups. While some organizations give business experts a read-only access to decision logic for review and validation, others have successfully transitioned the full ownership of business rule lifecycles to the business. Simple but efficient rule governance principles can help turn that vision into reality.

Rule Governance: Striking the Right Balance Between Agility & Control

Modifying business logic is inherently part of an application maintenance process. Moving to a business-driven process rather than an IT-driven development process requires adopting basic governance principles that put appropriate controls in place while allowing for agility.

First, the map of stakeholders needs to be clearly established. This is an important step in making sure everyone understands their roles and responsibilities in maintaining, reviewing, testing or deploying rules. Most importantly, the stakeholders' map gives a greater sense of empowerment and accountability for all the people involved in maintaining rules.

Then the rule management process must be defined, specifying how a change to the decision logic will be authored, approved, put in production, and eventually retired. For instance, a branch manager might have the authority to update surcharge and discount rules that apply to his region, while his regional manager will have overall responsibility for reviewing all pricing rules nationwide and approving them for deployment into production. Those processes set expectations among all stakeholders about what changes are allowed and when they will take effect.

Finally, the last step is to establish a clear set of controls that will provide the ability to monitor each business rule lifecycle from authoring to testing, deployment, and monitoring. As business rules embody the business decisions of an organization, understanding who changes them, when, and why is critical to proper governance. Equally important is the ability to provide execution reports that explain how decisions were reached for a given transaction.

Depending on the experience of the organization and the frequency and magnitude of expected rule changes, the best approach can be to set up a dedicated rule management team. Such a team would typically be a liaison between business and IT, and it would ensure that the rule development process is followed and evolves as needed.

Establishing such a collaborative, business-driven maintenance process for your decision services has powerful consequences for the organization and positive psychological consequences for all participants. Those consequences are a significant improvement in agility, an increased trust from the business in the IT systems, and a high level of support for the SOA vision across the organization.

Case Study – Using BRMS as an Essential Part of SOA Strategy

When organizations decide to adopt BRMS, they quickly realize the business value it can bring to SOA. A major financial services group with operations across Europe and Latin America is a good example. Due to its rapid growth through acquisition, this organization realized that it needed to invest heavily in IT modernization to align its various financial institutions and enhance its competitiveness across the markets in which it operates. It decided to adopt an SOA strategy to streamline its IT infrastructure from the front to the back office, with the goals of continuously improving customer service and reducing costs. The objective was to have a single platform across all banks in the group, standardizing processes and creating product factories (back-office systems able to process the products for various distribution channels, under different brands or for different packages).

As part of this SOA initiative, the company realized that management and automation of decisions within its business systems was a critical element to realizing the value of its overall IT investment. By adopting and implementing BRMS, it is working to maximize the reusability and adaptability of its platform across all banks, ensuring that its banks can respond quickly to changes in regulations and market conditions.

In a variety of systems that are subject to frequent business changes, the use of BRMS has separated the management of business policy from application code, allowing business experts to maintain these policies and to easily make changes to them based on the needs of the business. This enables both an increase in the speed with which it can adapt its systems to changes in the business environment (due to regulations, competitors, or internal factors) and a flexible approach to the variability that exists in handling unique requirements across geographies, customers, and processes. One of its most important requirements from a technology perspective is the separation of process and policy logic, since policies tend to change much more frequently and require rapid implementation and deployment. The use of BRMS technology provides the standard SOA benefits of decoupling, reusability, and standardization with the business benefits of reduced time to market and improved responsiveness to changing market conditions.

Conclusion

If you want to build business agility and value as part of your SOA initiative, where do you start? In terms of BRMS adoption, it is important to look for a product that is designed for SOA, both in terms of decision services composition and in terms of deployment, management, and execution of these services. Next, it makes sense to identify a starting point for the use of BRMS. The best place to begin is with the most dynamic business systems, where the frequency of changes to decision logic and the need for fast response to ongoing change are creating a barrier to agility. It is in these conditions where BRMS will allow your organization to make business change work to your advantage and to demonstrate the value that SOA promises. ■

About the Authors

Nicolas Robbe, vice president of product marketing, heads strategic marketing for ILOG. Nicolas has attended University of Colorado, U.S., Universite de Technologie de Compiegne, France, and Stanford Graduate School of Business.

Brett Stineman is the director of Business Rules Management Systems at ILOG. Previously, he was a senior product marketing manager at EMC, focusing on the Documentum Process Suite. He was responsible for external marketing efforts for the EMC Process Suite, including messaging, positioning and promotion.

2008 VIRTUALIZATION CONFERENCE + EXPO

www.virtualizationconference.com

The First Major Virtualization Event in the World!

June 23-24, 2008 New York City

Reaching Beyond Physical IT: Explore the Power of Virtualization in Enterprise Computing

Virtualization, the fundamental technological innovation that allows skilled IT managers to deploy creative solutions to such business challenges, is rapidly evolving from taking one computer and making it appear to be many computers, to virtualizing almost any information technology resource.

Virtualization of distributed computing, of IT resources such as storage, bandwidth, and CPU cycles are all increasing significantly in importance within enterprise IT. Virtualization can also apply to a range of system layers, including hardware-level virtualization, operating system level virtualization, and high-level language virtual machines.

In short, Virtualization is being adopted by IT organizations around the world so rapidly right now that, according to research firm IDC, spending on virtualization software and services is expected to exceed $15 billion worldwide by 2011 - up from $6.5 billion in 2006.

The Virtualization Conference & Expo 2008 East, taking place on June 23-24, 2008 in New York City, is the leading event covering the booming market of Virtualization for the enterprise.

TOPICS WILL INCLUDE:

- I/O Virtualization
- Application Virtualization
- Desktop Virtualization
- Network Virtualization
- Physical to Virtual (P2V) Migration
- Server Virtualization
- Storage Virtualization
- Virtual Machine Automation
- File Virtualization
- Management Applications, Tools and Utilities
- Virtualization Scripts and Procedures
- Paravirtualization
- Real-time Virtualization
- Device Virtualization
- Thin Client Solutions
- Virtual Desktop Infrastructure
- Greening of IT
- Vista & Virtualization
- Microsoft's Remote Desktop Protocol
- Virtual Machine Desktops
- Security & Performance Issues
- Virtualization Security Audits
- Utilization and Performance
- Virtual Backup
- Systems Integration
- Cloud Computing
- High Availability (HA)
- KVM (Kernel-based Virtual Machine)
- Virtual Appliances
- and more...

REGISTER TODAY AND SAVE!

www.virtualizationconference.com

Rationale for a SOA Shared Service Center

BY THADDEUS MARCELLI

Over the past several years, there has been much fanfare regarding the promise of SOA. However, as with any IT-related solution, it is not a magic bullet or cure-all for IT integration. In fact, SOA does not solve business process problems, but rather identifies both the good and bad organizational processes. In most instances, SOA will require additional front-loaded costs until a critical mass of services are developed for reuse. It will require organizational and even process changes that will demand high levels of training, funding, and organizational governance. Even with these challenges, adopting an SOA methodology is recommended.

Building a successful SOA Governance organization requires the attention to assets and capabilities across all service domains or high-level categories of services. Domains typically can focus on a major entity, e.g., customer or employee, the intersection of two entities, or smaller partitions like product pricing. SOAG is an extension of IT Governance, which focuses on the decision rights and accountability framework to encourage desirable IT behavior.

SOAG also balances enterprise needs and departmental goals to create a framework for delivering service-oriented business solutions. It defines individual and group responsibility, accountability, and a structure for identifying, amending, and enforcing governance policies. Conversely, developing SOA without a defined governance model could lead to less-than-desired outcomes. There are the obvious sunk costs associated with the resources, hardware, and software used during the trial and implementation phases, but it goes beyond financial loss.

One of the most efficient means to execute against an SOA Governance plan is to institute an SOA Shared Service Center (SSSC). Generally speaking, a Shared Service Center provides a centralized way to coordinate all SOA-related activities among team members efficiently. It also provides a way to enforce governance processes very similar to the way police enforce state and city laws.

SOA Shared Service Center Rationale

The argument could be made that creating a separate organization to support the SOA Governance program is too much. Those making this argument believe that resources spread across existing organizations could be as effective. While this can sometimes be true, it greatly depends on the scope and goals of the SOA program. The approach still requires a strong central coordination point with managerial power over a variety of resources with differing levels of responsibilities, priorities, and time. It has been found that in most organizations there are many good reasons to establish an SSSC.

Fulfill Customer Needs

A customer can be defined as any individual, group, or organization with SOA environment dependencies as either a consumer or provider. It is significantly easier for customers to contact one entity or unit that can consult, develop, guide, and support their SOA needs rather than having to communicate with a variety of individuals with no set guidelines or centralized, consistent communication back to the constituent. A single point of contact simplifies communications as the SOA Program matures.

A simple analogy is a call center where there is usually one phone number and a set of options to choose from once connected. This model is significantly easier than having to dial a different phone number for each individual problem.

One thing to always remember is that success sells. Success for SOA comes from being overly attentive to constituents and ensuring that their concerns are addressed, deadlines are met, and expectations are exceeded. It is also important to document the successes, issues that were overcome, and the expected returns in order to communicate results back to the enterprise. This communication with customers and prospects fosters adoption and raises awareness of the importance the organization is placing on SOA.

A decentralized team will not be able to do this as easily because they are not concentrated on the same set of goals and values. As a result, customer service will suffer, hindering SOA adoption.

Drives Compliance and Consistency and Aligns Priorities

Having the same goals, methodologies, and standards to adhere to creates consistency in standards, policies, processes, and communications. This consistency also leads to customers receiving the same level of support. Without consistency, units and groups will start to have reservations about adopting an SOA approach.

A singular unit driving at the same goals and communicating the same vision is likely to have a much more positive effect than other types of organizational structures. It helps to ensure that the SOA message does not get lost. In the early phases of SOA adoption, SOA Governance can benefit from a high level of structure to ensure the enterprise is receiving the same consistent message.

In addition, a dedicated team eliminates various priority conflicts that would otherwise result in poor customer services, miscommunication between team members, increased workload for program management, and lower productivity. A centralized team also results in higher productivity by reducing the likelihood of redundant activity across the SOA effort.

Cost Verification

When deciding to adopt SOA, many companies require business validation in addition to technical practicability. An SSSC makes

it easier to assess the total cost of implementing an SOA program. Centralizing SOA assets and resources makes it easier to identify the total costs. It also facilitates capacity planning, resource allocation, and identification of training and education needs.

Conversely, if the resources are shared or are not one hundred percent committed to SOA activities, it complicates the applicability of a cost structure. Unless there is a strict time-reporting process in place, the ambiguity of the assets and resources needed to further mature SOA will be present. It is also difficult to cross-train resources, which will lead to support and consistency issues. Ultimately, this could lead to difficulties in gaining additional funding as SOA matures, dampening the effectiveness of SOA.

Experience Drives Expertise

A Federated SSSC will build expertise over time. This is accomplished by the sharing of information, ideas, and understanding of SOA. Cross-training should also take place to reduce the affects of turnover. Turnover is a fact of business and SOA success requires both simple, logical processes and expertise to ensure the right level of technical and business governance is in place.

In addition, an experienced team will be suited to handle new challenges and tasks. An experienced and federated service center could drive toward goals more quickly by finding new and inventive means to achieve objectives.

Finally, a centralized team will be able to explore and digest new technologies, platforms, and methodologies more quickly, providing valuable research and development information back to the Governance Organization. This information could influence long-term planning and decisions.

A decentralized team will ultimately question others' knowledge because they do not communicate regularly. Without strong leadership, teams will begin to question others' authority. It could lead to higher turnover, which will affect overall team performance.

Data Reliability

Data collection and reporting is one of the chief processes for any Shared Service Center. An SSSC will be able to standardize and update the type and data collected for such items as service contracts. This is a pivotal role in ensuring that the right data is collected, maintained, and distributed. This data builds trust in the SOA environment, which is one of the hurdles that any SOA program must overcome. Accurate metrics reporting and metadata modeling will also support policy and process decisions along with identifying gaps in infrastructure, security, and other areas.

Quality

By adhering to governance standards for service and performance, the shared service center plays a key role in ensuring that functional, performance, and overall test management is maintained. It is imperative that services adhere to the highest standards for interoperability and conformance to ensure reusability and that quality and granularity of the services are built for broad adoption.

A decentralized organization will require more transactions between governance bodies, potentially creating process bottlenecks and miscommunication issues that will lower the quality of the services being developed. A decentralized organization will also require adherence to the same report systems and process tools that many organizations lack.

Ownership

As the number of services and the dependency upon SOA increases, it will be important for the Shared Service center to take ownership of some or all of the broadly shared services. By managing the care and feeding of the services, the life cycle, support, upgrades, and management of services will be significantly simplified. Ownership models and costs will also be centralized, creating quicker time-to-market, reduced organizational complexity, and straightforward cost structure.

A decentralized model has the potential to become quite complex and confusing. It may become complex in the processes, organizations, and people needed to agree regarding contract approval, upgrade plans, and orchestrations. It may become confusing with regard to the variations in the process steps based upon different ownership models, decisions points, and people needed for service usage negotiations.

Manage Risk

SOA brings a ubiquitous nature to IT assets and services that were previously viewed as vertical applications contained by the platform for which they were built. As a result of this omnipresent nature, there are risks with the development and use of SOA assets. A Federated Shared Service Center can proactively review and identify various threats and devise solutions to overcome.

A decentralized staff would most likely be very reactive in nature. As a result, potential threats could be missed or overlooked, which could cause issues ranging from loss or corruption of data to theft of highly sensitive or private data.

The SOA shared service center is the hammer in the arm of the SOA governance organization. Moreover, the SOA Shared Service Center executes against the SOA Governance goals and facilitates in driving the SOA roadmap. It also provides important feedback to the SOA Governance committee on the progress of SOA. Finally, it is in most instances the face to the customer, so it is important to ensure that the SSSC is a diverse group of knowledge experts that are committed to achieving the goals of the SOA Program.

As with SOA Governance, an SOA Shared Service Center should be built on an adopted SOA maturity model. This will ensure that the SOA Program, its governance model, and the SSSC are aligned and following the same goals.

Second, identify resources, both existing and new, that will compose the SSSC. These resources should be based on the roles identified that best fit the SOA Program. This step is essential to determine the total budget required.

Third, ensure that existing SOA-related projects and development efforts are being attended to and that these efforts do not experience any pain from the current level of SOA maturity. In many instances, it may make sense to adopt an exception process depending on where the project currently is within the service development life cycle.

Finally, begin to identify future SOA development efforts to exercise new policies, procedures, and standards. This will also assist in refinement and gap analysis. As progress is made, begin to gather and analyze metrics and data that will further establish a solid SOA governance program and appropriate customer service and governance enforcement form the Shared Service Center.

About the Author

Thaddeus Marcelli is a principal SOA srchitect at MomentumSI. He has many years of practical knowledge in the SOA framework space and has faced many SOA implementation challenges.

SOA Governance

Start small and build incrementally

BY KYLE GABHART

If governance were a house, you'd have the option of either building it from the ground-up or attempting to haul a complete house in on a large truck. While the latter is possible, it's fraught with difficulty. The house doesn't lend itself well to transport.

It may be damaged during the move. It may not fit on your lot or connect smoothly to your utilities, requiring modifications to be made on the spot. The first option, building the entire house on-site, certainly has its challenges (proper design, accurate implementation, quality assurance), but the risks are much lower and there's opportunity to adapt as it's built (move walls, change windows and doors, etc.). Generally, organizations should choose this option. They should choose to start small and build incrementally.

Incremental Governance

While no formal governance stages have been defining by any industry groups, many organizations experience an evolution similar to the one identified in Figure 1.

Phase 1: Informal, ad hoc oversight provided by Subject Matter Experts (SMEs)
Phase 2: Formalized oversight through workflow, standard documentation, and identification of applicable standards
Phase 3: Establish best practices, select and promote common design patterns around architecture and service layering
Phase 4: Drive enterprise alignment between business and technology teams along end-to-end business processes

How these incremental governance phases are manifested will vary from enterprise to enterprise. For some it may emphasize infrastructure investments. Other governance strategies are more focused on techniques and methodologies while others will involve more committees and organizational changes. Regardless of what form the governance takes, the key is to start small and build incrementally.

Adopting Governance

Governance adoption should follow a simple, pragmatic approach. Overriding goals during the governance adoption process should be risk mitigation and increasing operational predictability to reduce risk. The following simple steps are recommended:

1. Define a governance roadmap with objective, measurable milestones
2. Identify technology and organization changes required for each stage
3. Clearly identify and document business value that is applicable at each stage

That last recommendation is the step that is most often missed. So often organizations will put a plan of action in place without any understanding of what benefit the business will gain from each iteration or each level of maturity. This is crucial to ensuring that your SOA is relevant, your governance is sufficient without being overkill, and ultimately is core to achieving a return on your service-oriented investment.

Less Is More

In the process of adopting governance incrementally, it's important that you avoid the tendency to implement excessive, heavyweight governance. On the opposite extreme, you should also take

"Goverance should be focused, lean, and ever-present"

Phase	Mode	Description
1	Informal	Ad-hoc oversight provided by Subject Matter Experts (SMEs)
2	Standardized	Formalized oversight through workflow, standard documentation, and identification of applicable standards
3	Operational	Establish best practices, select and promote common design patterns around architecture and service layering
4	Strategic	Drive enterprise alignment between business and technology teams along end-to-end business processes

Figure 1:

care to avoid implementing governance that's nothing more than a formality and a rubberstamp of approval. Governance shouldn't be implemented by one or more powerless committees, or as heavyweight bureaucracy. I call this balanced approach – "lean governance".

Lean governance represents a mindset in which governance is applied as needed. Implement only as much governance as needed and constantly monitor the environment to tweak the degree of guidance and oversight. Governance should be focused, lean, and ever-present. Early in the adoption of SOA, governance should be minimal. It could be as simple as a requirement to support certain standards and maintain service contracts for each service. Over time, the governance policies, processes, and procedures, as well as the corresponding infrastructure, can be built incrementally alongside the maturing of the service-oriented enterprise. Finally, this governance should be applied throughout the project lifecycle. Best practices identify three governance gates – design-time, change-time, and runtime. These three gates serve as checkpoints to ensure that service design, development, and runtime behavior are consistent with enterprise goals and stated best practices.

Summary

In my experience, governance is crucial to any significant organizational change and service orientation is no different. I don't believe that governance has to be at the extremes (powerless committees versus heavyweight bureaucracy), but instead can and should be focused, lean, and ever-present. Initially you don't need very much governance, and may involve more of an emphasis on infrastructure or techniques and methodologies. Regardless of how it's manifested, governance should be planned from the start, matured incrementally, and should provide context for the entire project and program lifecycle. ■

About the Author

Kyle Gabhart is a subject matter expert specializing in service-oriented technologies and currently serves as SOA solutions director for Web Age Solutions, a premier provider of technology education and mentoring. Since 2001 he has contributed extensively to the SOA community as author, speaker, consultant, and open source contributor.

kyle.gabhart@webagesolutions.com

RIA + SOA: The Next Episode

Building next-generation web platforms

BY NOLAN WRIGHT

It's a question we're asked a lot: How do I get started with SOA?

The world of web development is moving away from MVC-based web architectures and toward a client/server model that is probably best described as RIA + SOA, where RIA represents the rich user interface and SOA represents the services that it consumes. There has been a lot of buzz around rich Web 2.0 applications, but they will not become mainstream until the next generation of web platforms emerge – fully integrated platforms that enable RIA + SOA.

State of the Union

Currently, in the standards-based world of HTML, CSS, and JavaScript, RIA developers have to assemble multiple third-party libraries and frameworks in order to build a rich user interface. This "a la carte" approach to building RIAs places an unnecessary burden on the developer. Instead of focusing on building applications, the developer must spend time finding, integrating, and versioning the various pieces of their RIA development platform. The same holds true on the SOA side: developers are left to figure out how to create services and how to integrate them with their RIA front ends. Developers need a platform that addresses every aspect of building an application, so that they can focus on doing what they do best – build applications. The question is: What should a next-generation RIA + SOA platform look like? The best place to start is with the activities that are required to build an RIA + SOA-based application. At a high-level these activities include:

1. Design the "look" of the application

This is the general appearance of an application. It includes things such as color, fonts, graphics, and a general page layout.

Common toolsets: HTML, CSS, and images

2. Integrate widgets

Widgets encapsulate a set of common capabilities within a single component. They typically contain both "look and feel" as well as a set of pre-defined dynamic behaviors. They are a fundamental building block of an RIA.

Common toolsets: ExtJS, Dojo, Yahoo YUI and several other small widget projects

3. Add dynamic behavior to the user interface

Creating dynamic behavior in the user interface involves two things:

- Event handling
- Document Object Model (DOM) Manipulation

Event handling is the ability to know when a particular event occurs (e.g., a user clicks a button or a service response is received). DOM Manipulation allows you to dynamically change the user interface based on the receipt of an event.

Common toolsets: JavaScript libraries such as JQuery, Prototype, and Scriptaculous

4. Consume services

Consuming back-end services is a key capability of an RIA. It enables the creation of single page user interfaces that exchange application data with services. It also enables a clean separation between the user interface and the service tier. The most common method for interacting with services is AJAX.

Common toolsets: JavaScript libraries such as JQuery and Prototype

5. Create services

Services provide an interface to data and application business logic.

Common toolsets: There are several frameworks available for creating services in your programming language of choice

Now that we have a sense of what is required to build RIA + SOA-based applications, we can take a look at how these activities should be integrated in order to provide the most value to developers. The following sections outline the defining characteristics of a next-generation RIA + SOA platform.

Support for HTML and CSS

These two languages are perfectly suited for implementing the "look" of an application, plus they are familiar to most developers of web-based user interfaces. There is no reason to re-create the wheel.

Provide an Open Widget Framework

As stated earlier, widgets are a fundamental building block of RIAs. There are several widget toolkits available from the likes of Yahoo, Dojo, and ExtJS. There are also several standalone widgets that have been created by smaller projects and individuals. You can build RIAs using these widgets, but there are some catches:

- It's unlikely that one widget offering is going to meet all of your needs.
- Integrating different third-party widgets may require custom code as well as an in-depth understanding of how each widget works.
- Writing new widgets is challenging because you either have to write to the low-level API of your preferred widget toolkit, or you have to write your own from scratch.
- Some widget frameworks require developers to write a significant amount of JavaScript just to use their widgets, which is problematic for those with little to no JavaScript experience.

In order to address the problems above, an RIA + SOA platform should provide an Open Widget Framework that has the following capabilities:

- Supports integration of existing third-party widgets
- Provides a simple API for creating new widgets
- Enables widgets to be used via simple markup (no JavaScript required)
- Supports a distribution model that makes it easy to submit, find, and use new widgets

An Open Widget Framework will provide developers with one source for obtaining widgets and one simple way to integrate these widgets into their application. If a particular widget is not available, it can easily be created using the Open Widget Framework API.

Given the importance of widgets within RIA development and the fragmented nature of widgets today, an Open Widget Framework should be considered an essential part of any RIA + SOA platform.

Provide an Integrated RIA Programming Model

Developing RIAs requires significantly more user interface code than traditional web-based applications. As a result, a next-generation platform needs to provide an integrated RIA programming model that facilitates and simplifies the key user interface programming tasks. These tasks include:

- Event handling
- DOM manipulation
- Service consumption (AJAX)

Event handling, DOM manipulation and AJAX enable the "rich" in rich Internet application. It doesn't take much investigation to see that they are closely related. To illustrate this point, let's look at the typical steps involved with an RIA login form submission:

- Login button is clicked (event handling)
- Service request is sent (AJAX)
- Some type of activity indicator is displayed (DOM Manipulation)
- Service returns (AJAX)
- Activity indicator is hidden (DOM Manipulation)
- A Login "success" message is displayed (DOM Manipulation)

Despite their close relationship, most frameworks and libraries provide only light integration between the three, leaving the developer to bridge the gap. To illustrate this point, let's look at some code. In the example below, we will set the contents of one combo box when the value of a second combo box changes. This example is written using JQuery:

```
$(function(){
  $("select#comboOne").change(function(){
    $.getJSON("/combo.php",{id: $(this).val(), ajax: 'true'}, function(j){
      var options = '';
```

```
      for (var i = 0; i < j.length; i++) {
        options += '<option value="' + j[i].optionValue + '">' + j[i].optionDis-
play + '</option>';
      }
      $("select#comboTwo").html(options);
    })
  })
})
```

Now, let's look at how this could be accomplished using a fully integrated approach to event handling, DOM manipulation, and AJAX.

```
<select id="comboOne"
on="change then r:load.combo2.request">
</select>
<select id="comboTwo"
on="r:load.combo2.response then value[property=rows,text=text,value=value]">
</select>
```

These two examples accomplish the same thing, but in very different ways. The first example requires more code and it's all in JavaScript. The second example uses a simple expression language to do the same thing. Let's take a closer look at the syntax.

```
on="change then r:load.combo2.request"
```

In this expression, the AJAX request message r:load.combo2.request will be sent when the <select> gets a "change" event – pretty simple. Now, let's look at the second expression:

```
on="r:load.combo2.response then value[property=rows,text=text,value=value]"
```

When the AJAX response message r:load.combo2.response is received, the contents of the second combo box are set based on data in the response message.

Let's take this example one step further by adding a visual queue when the response message is received.

```
<select id="comboTwo"
on="r:load.combo2.response then value[property=rows,text=text,value=value]
and effect[Highlight]">
</select>
```

By adding and effect[Highlight] to our expression, we are able to incorporate a subtle effect that lets the user know that the combo box values have changed.

These code examples demonstrate the power and simplicity of a fully integrated approach to event handling, DOM manipulation, and AJAX. Developers who have little to no JavaScript experience will find it easy to learn an expression language like the one above. It will enable them to become productive quickly, since they don't have to climb a steep learning curve. Of course, there are developers who are comfortable with JavaScript and would prefer to use it. As a result, an integrated RIA programming model should also support the use of JavaScript, specifically in a way the supports the separation of behavior from markup. This is commonly referred to as unobtrusive JavaScript. Let's look at an example:

```
<img id="progress_image" src="images/indicator.gif"/>

$("progress_images").on("r:login.request then show").on("r:login.response then
hide");
```

In this example, the <img> markup is separate from the JavaScript code that defines its behavior – i.e., "show" when the login request message is received and "hide" when the login response is received. This type of programming model is good for developers who prefer to do their RIA programming in JavaScript.

An Integrated RIA Programming model is a fundamental component of an RIA + SOA platform. It provides developers with a single integrated mechanism for handling the major RIA programming activities. The net result is that developers can build rich user interfaces faster and with significantly less code than is required today.

Provide an Integrated Services Platform

RIA only gets us halfway to our goal of building a rich application. We still need to provide an answer for the SOA piece of RIA + SOA. Unfortunately, the current crop of Web 2.0 toolkits and frameworks primarily focus on RIA; they provide little to no support for building services. This is problematic because developers are, once again, left to bridge the gap, which makes application development and maintenance unnecessarily more time-consuming and difficult than it should be.

A next generation RIA + SOA platform must address this gap by providing an integrated services platform that provides the following:

- Support for creating services in any programming language
- Seamless interoperability between the RIA and SOA tiers
- Ability to consume local mock services

Historically, web-based frameworks have been built around a particular programming language, but in the world of RIA + SOA this practice seems outdated and unnecessary. RIAs only need to exchange application data with services, so they should be programming language agnostic. The only required link between an RIA and its SOA-based services should be a lightweight message-based contract. This loose coupling between the RIA and SOA tiers opens the door to an integrated services platform that enables developers to use any programming language for service creation without impacting the RIA tier of the application.

An integrated services platform should also provide seamless interoperability between the RIA and SOA tiers. Specifically, it should handle service routing and data marshalling on behalf of the developer. Here's an example of a simple integrated approach to building a service. This particular example uses Java.

```
@Service (request = 'login.request', response ='login.response')
protected void loginRequest (Message req, Message resp)
                                                    throws Exception
{
   String username = req.getData().getString("username");
   String password = req.getData().getString("password");
   User user = UserDAO.login(username,password);
   if (user !=null)
   {
   response.getData().put("success",true);
   response.getData().put("user",user);
```

DataServices WORLD

Welcome to DataServices World at SOAWorld 2008!

June 24, 2008 New York City

Ken North
Chair, DataServices World at SOA World '08

DataServices World is about the confluence of databases, data warehousing, business intelligence, enterprise computing and Internet computing. Its focus is architectures and technologies for accessing data from heterogenous data sources and providing that data to consumers such as components, services and applications.

Distributed processing, high-speed networks, powerful servers, components and collaboration define the landscape of 21st century computing. For building new systems and exploiting legacy applications and data, developers are looking to collaborative applications assembled from distributed components. The emergence of XML and web services spurred an increase in services-oriented architecture (SOA) adoption. SOA today can include web services, grid services, integration services, semantic web services, components and messaging systems.

Developers who've enjoyed success with component-based development are looking to new architectures with services as the new components. But even as the Service Component Architecture and other new technology gain tractions, some system characteristics remain consistent. Today's systems, like their predecessors, typically have a requirement for persistent information and databases.

Many organizations have a variety of persistent data stores, including SQL databases, geo-coded data files, spreadsheets, content management systems and XML. Services, applications, and mashups can consume and integrate data from disparate data sources. In an n-tier enterprise architecture and a service-oriented architecture, the logic for providing data from databases and other data sources resides in data access layers and data services layer.

Today's data services layers encapsulate logic for accessing data stores, typically using standards-based technology such as ODBC, JDBC, ADO.NET and Service Data Objects. These specifications define solutions for uniformly accessing and manipulating data from heterogeneous data sources.

At DataServices World, we'll uncover architecture and technology solutions for accessing, integrating and processing data from multiple sources while guaranteeing security and scalability. These solutions include robust, high-performance data access middleware, optimized databases, efficient protocol handling, tuned queries and state of the art data services. We'll be looking at technology of interest to CTOs, enterprise architects, system architects, information architects, developers, database gurus, consultants and analysts.

DataDirect® TECHNOLOGIES

2008 Diamond Sponsor

REGISTER TODAY AND SAVE

www.dataservicesworld.sys-con.com

```
    return;
}
response.getData().put("success",false);

}
```

In the example above, there are two things to notice. First, a plain Java object is turned into a service by simply adding a "Service" annotation to a Java method. This annotation contains both the service request and service response message that this method handles, which makes routing configuration easy. Second, working with service request and response data is simple and straightforward. In the example, the entire user object is placed into the response message. The service platform handles the data marshalling. The result is that developer can focus on writing service logic instead of writing glue code, which ultimately results in faster development with less code.

Finally, if the contract between the RIA and its services is message-based, then it becomes possible to create local mock services. Local mock services respond to remote requests but they exist locally within the RIA. This is a powerful capability because it opens the door to creating fully functional RIA prototypes without writing a single line of service code. These local mock services can be placed in a single file and removed once user interface development is complete. Let's look at an example.


```
<!-- Progress indicator -->
<img src="images/indicator.gif" style="display:none
on="r:login.request then show or r:login.response then hide"/>

<!-- Login form -->
Username: <input type=text" fieldset="login" id="username"/>
Password: <input type="password" fieldset="login" id="password"/>
<input type="button" value="Login"
on="click then r:login.request"/>

Login Form

<app:script on="r:login.request then execute after 1s">
   $MQ('r:login.response',{'success':true,'username':'foo'});
</app:script>
```

Mock Service

In the example above, we have a login form that generates the service request r:login.request. We also have an image that will show when r:login.request is received and hide when r:login.response is received. The second part of the example is the mock service. The mock service listens for the r:login.request service request and responds with a r:login.response message after one second in order to simulate service latency. The login form has no knowledge of the service location; it simply responds to messages. This simple example demonstrates how an entire RIA prototype can be built without writing any service code. This prototype would be 100% reusable, and it has the added benefit of enabling developers to define the service contracts in advance of service creation. The result is that service creation is greatly simplified because service developers not only have a fully functional prototype as a reference; they have a complete set of service interfaces.

Conclusion

Developers are currently caught in the middle of a fundamental shift in web application development. We are moving away from the server-side MVC frameworks of Web 1.0 and toward a client/server architecture for the web, or more specifically RIA + SOA. As a result of this shift, developers have been left to piece together their web development platform for rich web applications. Of course, change creates opportunity. The opportunity in this case is to build a next-generation web platform that provides end-to-end support for building Web 2.0 applications. At Appcelerator, we saw this shift coming well over a year ago. Our response was to build the Appcelerator Platform, a platform built from the ground up to support RIA + SOA. The examples used in this article are based on Appcelerator. Of course, many different approaches will be taken to create the next generation of web platforms, but while the specific implementations may differ, the general characteristics should remain the same. ■

About the Author

Nolan Wright is co-founder and CTO of Appcelerator, leading the company's product and services organizations. Prior to starting Appcelerator, he led engineering and product management at Vocalocity. He has also held several senior technology, product management and consulting positions with Accenture, Netscape Communications and Vertical One. Wright is a graduate of Vanderbilt University, where he earned his BEEE in Electrical Engineering. For more information, please visit http://www.appcelerator.org.

THERMAL GRADIENT GUARANTEED INCIDENT FREE

$150,000 THERMAL GUARANTEE
AGAINST HARDWARE DAMAGE TO YOUR SYSTEM

WORLD'S ONLY THERMAL GUARANTEE

Introducing the Efficient Enterprise:™ more power, more control, more profits

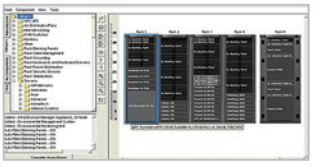
Can your legacy system say the same?

Your service panel limits the amount of power available. Your budget limits the amount of money. You have to stretch every bit of both as far as you can. What you need is the APC Efficient Enterprise.™

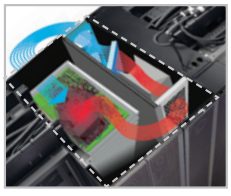
The APC solution offers modular scalability so that you pay only for what you use; capacity management so that you know where to put your next server; and dedicated in-row and heat-containment systems that improve cooling and thermal predictability. An Efficient Enterprise earns you money through the pre-planned elimination of waste. For example, simply by switching from room to row-oriented cooling, you will save, on average, 31% of your electrical costs.

Our system reimburses you

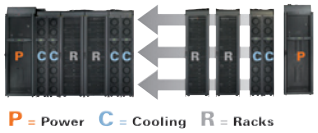
Whether you're building a new data center or analyzing the efficiency of existing systems, your first step is knowing where you stand. Take the online Enterprise Efficiency Audit to see how you can reap the benefits of a smart, integrated, efficient system: more power, more control, more profits.



CAPACITY MANAGEMENT



CLOSE-COUPLED COOLING



P = Power C = Cooling R = Racks

CONSERVE POWER

CONTAIN THE HEAT

Efficiency™ Quotient
DATA CENTER

Visit the APC Technical Session: "Maximize Your Virtualization Payoff; There's More Than You Think" Domenic Alcaro, Director - Enterprise Segment Northeast — Grand Ballroom

APC® by Schneider Electric

e-mail: esupport@apc.com • 132 Fairgrounds Road, West Kingston, RI 02892 USA 998-1240A

SOA and Services as a Service

Why your old data center infrastructure won't scale in the SOA age

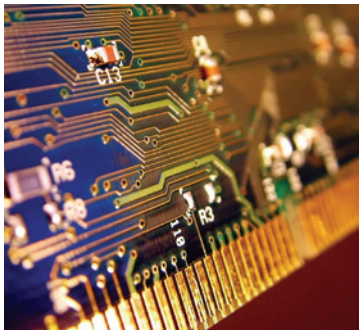
BY DAVID LINTHICUM

While the number of SaaS providers grows, as well as enterprise acceptance, we are really not breaking new ground. In essence, today's SaaS providers offer visual systems, meaning they communicate with a human being. They also provide a single visual interface, and the users have to take both the data and behavior as provided. We could call this an enterprise application that's not much more than a Web site, or an old-Web technology.

Moving forward, we now have the opportunity to leverage discrete services, as needed, for use in both SaaS-delivered enterprise applications and SOA. These are typically Web Services that provide a specific and narrow set of behaviors and data that are meant to become part of a larger application or composite. For instance, address validation services, tax rate calculation services, stock transaction services...you get the idea. They aren't visual services, but can become core components of your SOA and leverage services that you don't have to write, test, or host. These services will exist with your local services in your repository. So you can build core applications by mixing and matching services that you rent, not create. This is the destination for the new Internet, and the next frontier for the existing SaaS players and SOA.

So, where do you get these services? Most major SaaS players such as Salesforce.com, Rightnow.com, and NetSuite.com provide their core application functionality as a service by standing up Web Services that are consumable by their subscribers. Guys like eBay.com and Amazon.com provide similar services that, of course, support their businesses as points of integration, or extend the functionality of their Web sites and core businesses.

However, there are emerging companies that are in the pure service provider business...renting all types of services for all types of purposes. We can call these guys service brokers, because they are providing a Web Services platform for many providers, to many consumers. They provide the provisioning, security, billing, maintenance, and support.



StrikeIron is the one that comes to mind. StrikeIron is not an application provider; it's a SaaS provider where the first S is "Services." They offer a variety of services such as SMS text messaging, tax rate calculations, address validation, etc. Their objective is not to provide the holistic application, but simple components that can be abstracted into enterprise and SaaS applications... as needed...for a small monthly subscription fee. So you don't write it, you don't test it, you don't host it, you just leverage its functionality at a fraction of the cost had you developed the same service in-house.

The major obstacles to leveraging services delivered on-demand that you did not write, test, or host are that the services weren't invented by local developers and aren't hosted in the data center.

The fact is, if you're always considering security issues, you really need to get over it. You'll find that your ability to adopt remotely hosted services now is very much like those who adopted the Web in the early 90s. Those who moved first lead the game and saved a ton of money in the process. Those who waited to accept the Web as a valid source of information had to catch up later. With this movement, most enterprises should be leading the way to leveraging as many rented services as they can, and never, ever building services that somebody has already built. Those services are typically better tested and provide better functionality.

I think this is the next driving force behind SaaS and SOA. As more and more businesses learn that these services exist, they can leverage them and the value is much the same as "traditional SaaS," perhaps even more valuable. Count on the Internet-delivered service provider, including brokers, to be the next big "hype cycle" in this business, and I have to say that it's pretty cool.

About the Author

Dave Linthicum is the CEO of StrikeIron (www.strikeiron.com), which offers Web services on-demand. In additional Dave is the author or co-author of 10 books, a thought leader in the Web 2.0 and SOA space, frequent keynote presenter, and has served as the CTO for 3 technology companies.

david.linthicum@strikeiron.com

Sound Architecture Requires Proper Planning

WEB AGE SOLUTIONS - YOUR TRAINING PARTNER FOR SOA

In all phases of SOA migration, Web Age Solutions provides training and customized services from awareness to implementation. We support vendor specific or generic SOA training tailored to your organization's needs.

Custom training for complementary SOA technologies

XML • WEB SERVICES • WSDL • SOAP • WEBSPHERE • WEBLOGIC • JBOSS • J2EE • SPRING/HIBERNATE/STRUTS • WID/WBI/WMB

Custom training plans for virtually every job role

BUSINESS ANALYST • ADMINISTRATOR • DEVELOPER • ARCHITECT • QA/TESTER • MANAGER • EXECUTIVE • SECURITY

Consulting services for all phases of SOA migration

IBM Premier Business Partner

Add Web Age Solutions to your plan & stay ahead of the competition

www.webagesolutions.com - 877.517.6540 - soa@webagesolutions.com

WEB AGE solutions inc.