Making Everything Easier!™

2nd Edition

Service Oriented Architecture

FOR DUMMIES®

Learn to:

- Identify what SOA is and why it's important
- Understand the components of a service oriented architecture
- Understand the business and financial imperatives of SOA
- Implement SOA best practices

Judith Hurwitz
Robin Bloor
Marcia Kaufman
Fern Halper

Service Oriented Architecture For Dummies®, 2nd Edition

Chapter 1: Getting to Know SOA
&
Chapter 7: Discovering the Main Components of SOA

ISBN: 978-0-470-37684-3

Contents at a Glance

Table of Contents

Chapter 1

Getting to Know SOA

In This Chapter

- Finding out why you should care about SOA
- Liberating business from the constraints (and tyranny) of technology
- Illustrating the need for SOA
- Saving bundles by using what you have
- Expanding your SOA to customers, partners, and suppliers
- Focusing on function

Service oriented architecture (SOA) is a hot topic being bandied about by IT vendors across the globe. IBM, Hewlett-Packard, Software AG, Oracle, SAP, and Microsoft (just to drop a few names) are all singing from the SOA songbook, and hundreds of vendors are adding their tunes as we speak.

"What's SOA?" you ask. We suspect that you've already skimmed a dozen articles and read (or deleted) hundreds of e-mails from vendors pushing SOA, but the answers you've gotten so far have been, well, vague and inadequate.

The short answer is that SOA is a business approach to building IT systems that allows businesses to

- Leverage existing assets
- Create new ones
- Easily enable the inevitable changes required to support the business

For you impatient readers out there, we expand on this short answer in Chapter 5. However, right now, we think the more important question is, "Why should I care about SOA?" In this chapter, we try to answer this question.

The promise of service oriented architecture is to liberate business from the constraints of technology, unshackling technologists and business leaders from the chains they themselves have forged. ("IT workers of the world, unite! You have nothing to lose but your chains!" as it were.) This has major implications both for the business and for the IT structure that supports the business.

From our perspective, one of the most important aspects of SOA is that it's a *business* approach and methodology as much as it is a *technological* approach and methodology. SOA enables businesses to make business decisions *supported* by technology instead of making business decisions *determined* by or *constrained* by technology. And with SOA, the folks in IT finally get to say "yes" more often than they say "no."

We pronounce *SOA* to rhyme with boa (***bow**-uh*). Stretching it out by clearly articulating each letter (S-O-A) is perfectly acceptable but might leave you stymied when we say things like, "SOA what?"

Business Lib

Executives have come to rely on technology — in terms of reporting, text analytics, projections, graphical representations, risk analysis, and other analytical tools — to make informed decisions for their company. The day-to-day operations of a company have slipped, little by little, into the hands of IT. Quite simply, more and more of the activities of an organization are supported by increasing levels of business process automation — whether its business is to build ships, sell insurance, or manage cities — and since IT implements the automation of business process, business decision makers have become more dependent upon IT. While this increasing use of technology has helped the business in so many ways, technology has also created significant constraints. At many companies, business and IT management operate in very separate worlds without the benefit of a common unifying language. Unfortunately, as organizations become more diverse and complex through mergers, acquisitions, globalization, and the need to manage lots more data, the supporting IT infrastructure has become more cumbersome and brittle after being stretched in so many different ways to keep up with all the changes. This is not good for business, and neither is it good for IT.

We're not advocating that business leaders should (or can) take control of the technology from the hands of IT. Modern businesses are inextricably tied to technology. No sizable business can function without IT — it's as simple as that. However, we are advocating a new world order. Indeed, we advocate that business leaders and IT work together to create this new world order. *Together,* business leaders and IT will communicate how the automated processes of its business should be facilitated, and work together to make it a

reality by using SOA. *Together,* IT and business leaders can determine a strategy that both liberates business from IT and allows IT to create maintainable, extensible, compliant systems to support the initiatives determined by business leaders.

Tech Lib

Just because business has become constrained by technology, don't think that the folks in IT are having a jolly old time basking in their new-found power. On the contrary, the IT staff gets to spend its time in endless meetings accounting for why projects are late, explaining why applications can't easily be adapted to changing business conditions, and pleading for more staff. When some clever marketer presents a new concept for selling more widgets via the Internet or mobile devices or some other new channel, IT management is always the wet blanket, having to explain why (despite the company's investment in all the latest software and hardware) it will take 18 months to implement the new plan.

With SOA, business and IT have a way to meet each other half-way and collaborate using a business focused approach to develop new ways to use technology to grow the firm, help to spot new trends and opportunities, and see new ideas to fruition. But before you go marching off to save the world, though, we have some more explaining to do. A story will help.

A SOA Case Study

Once upon a time, there was an insurance company called ABC Insurance Incorporated. When ABC was born — oh, maybe 150 years ago — it began by selling insurance policies to factories and manufacturers. In those days, there were no computers to mess things up. The company followed business processes that were pretty simple. A nice person sent a letter inquiring about a policy. A smart person set a rate, sold a policy, and hoped that nothing caught fire or blew up. ABC thrived for more than 100 years.

But then, things got complicated. Other companies started stealing ABC's business. Customers were asking for insurance for different kinds of risk. ABC had to change or die.

ABC was an early user of punch-card accounting systems. In the 1960s, ABC bought computers, hired programmers, and built software applications to support its business. In the 1980s, it bought software packages from different

suppliers to help it continue to compete. It bought or built business applications to solve problems all over the company — one at a time. For example, it bought an application for the corporate finance department, created one to handle customer claims, and procured other applications to manage research information about what type of accidents were most common under what circumstances.

This worked well for many years, until the 1990s, when ABC found itself competing against financial services companies who decided *they* could sell insurance, too. Suddenly, ABC needed to find new ways to compete so it could sell a larger variety of products to current customers and also find some new customers. Its leaders thought up exciting new solutions based on the knowledge of their business and their customers.

In addition, management thought ABC could expand its business by acquiring other insurance companies with complementary products. ABC could sell these new products to existing ABC customers and sell ABC's products to the customers of the companies they acquired. These smart guys and gals understood business strategy. Everyone got really excited until . . .

Management talked to IT, and IT said, "This is really, really exciting, but we have a *small* problem."

"What could it be?" asked management.

"It's this," said IT. "We can no longer simply buy or build more software applications to implement our innovative plans for new products and services. The business policies and processes that we follow have become more complex. Everything we want to do has to work in concert with what we already have. The very running of our company depends on all the business applications that we built and acquired over years working together smoothly — such as the programs that tally the premiums people pay; administer the claims we process; and make risk analysis, payroll, invoicing, and sales commission calculations. When you come right down to it, our company is the aggregation of all our programs. Everything we need to carry out our day-to-day business functions — including information about our customers, our products, and our risk performance — is locked inside these programs and processes."

"Well," said management, "You can just write new programs to tie everything together. We'll *integrate,* and we'll all be very happy."

And IT said, "Yes, it is possible to *integrate,* but integrating will take *a very, very long time:* at least 18 months. Maybe two years. And by then, you might want more changes that will take another 18 months or two years. By then, it might be too late. And," IT continued, "*it will cost lots and lots of money.*"

Management and IT were very sad. They knew that ABC wouldn't survive if they couldn't find a new way of thinking about business process and technology. So they began asking everyone they knew of any way to save ABC. They searched, and they studied, and they prayed — until one day, a package arrived. In that package were several copies of a yellow-and-black book titled, *Service Oriented Architecture For Dummies,* 2nd Edition.

Both management and IT took copies of the book and read. They were very excited to discover that they didn't have to throw away valuable assets and that they could reap benefits in a short time. In the end, they came up with a *new* strategy, one based on five key elements:

- The IT organization will partner with the business managers to create a high-level map of the business processes, followed by each line of business. This will help identify the similarities, differences, and interrelationships across the business lines for the company.
- The IT organization will create a flexible structure that will turn key IT software assets into reusable business services that can be used no matter how the business changes. These business services will include everything from business processes and best practices to consistent data definitions to code that performs specific business functions.
- The IT organization will begin replacing the hundreds of redundant business services locked in old software with these new reusable services.
- The IT organization will use only accepted industry standards to link these software assets.
- The IT organization will use the service oriented architecture concept described in the rest of this book to begin to create business services that are consistent with how the business operates.

Together, management and IT began a journey. As far as we know, they are living happily ever after. In Part V, we give you many real-life case studies from companies you might recognize that indeed are alive and well and living happily on their journey to SOA.

Better Living through Reuse

One of the biggest deals in the SOA world is the tenet that you don't have to throw away things. You take the stuff (software assets) that you use every day — well, the *best* of the stuff you use every day — and package it in a way that lets you use it, reuse it, and keep on reusing it.

One problem common of many large companies that have been around for a while is that they have lots of similar programs — software applications — representing commonly used business processes. Every time a department wants something slightly different, that department builds its own version of the software so that across a particular company, you might find umpteen versions of more or less the same processes — with, of course, slight variations. Many IT shops have policies and procedures designed to prevent this sort of thing, but when deadlines loom and budgets are tight, it's often easier and quicker to write something from scratch that fills the need rather than to coordinate with other divisions. This sort of duplication becomes a nightmare when one company acquires another and finds that they have similar (but not identical) applications purporting to do the same thing.

These slight variations are precisely what make systems very complicated and expensive to maintain — even one business policy change might affect lots of different applications. In situations like this, it's very difficult to find every instance in every application that needs to be changed. The testing required for this type of application change management takes time away from more innovative development work and can inhibit businesses from getting to market quickly with new products.

With SOA, these important business processes — such as creating an invoice, calculating an interest rate, securing a reservation — become business services. Briefly, a *business service* is a sealed container of software code that describes a specific business process that can be connected to other business processes. (We talk more about this in Chapter 5.) You end up with one single business service for a given function that gets used everywhere in your organization. With SOA, when you need to change a business policy, you change it in only one place. And, because the same service is used everywhere, you have consistency throughout your organization.

For example, you know that if you decide to create a new department in your organization, you're not going to create new Accounting, Human Resources, Legal, Cleaning, Training, and Travel departments to go along with it. Even if you need to add staff, you'll likely use your existing Accounting, HR, Cleaning, Training, and Travel departments to *service* — note the word *service* — this new department.

The problem is that over time, IT ends up embedding redundant function in programs everywhere in the organization. That redundancy — just like having separate Accounting, HR, Legal, Cleaning, Training, and Travel departments for every department — is what SOA ultimately eliminates. This lack of redundancy gives you the same obvious benefits of scalability, consistency, and maintainability.

With SOA, business managers work with IT to identify business services. Together, they determine policy and best practices. These policies and best practices become *codified business services* that represent honed company business processes. No need, for example, to have 30 different variations on an exchange rate translation application, each used by a different department and all requiring IT time for ongoing maintenance. One business service will do. Onward, the new world order!

Moving in Tandem with SOA

In any formal dance, from the cha-cha to the waltz, form matters. The *form* is what allows you to dance with someone you've never met. When both partners truly know the form, they move in tandem, are flexible, and navigate with ease and grace.

SOA is form. It enables the business to move, change, partner, and reinvent itself with ease and grace. In the beginning, mastering new steps requires focus and attention. Over time, the steps become second nature.

Implicit in the notion of form is standards. Using industry standard interfaces and creating business services without dependencies (more on that later, we promise) allows the business vastly more flexibility than it enjoys today to change its business model, reorchestrate itself, and partner dynamically.

Redundant reiteration, again

For any IT old-timers out there who have labored long and hard in the IT trenches, the concept of software reuse isn't new. You're familiar with subroutine libraries and the great theme of object orientation, and you extol the virtues of standardization. "What's the big deal with SOA?" you ask. "Aren't we already doing this?" Well, yes and no. Yes, because the world of SOA depends on a good understanding of reuse and on the building of reusable components. No, because SOA extends the idea of reuse not only to *Web services* but also to *business services.* (For definitions of *business services* and *Web services,* look in Chapters 5 and 6.) In the world of SOA, the level of granularity shifts profoundly. No longer are we talking simply about reusable low-level components: We're talking about reusable high-level business services. This shift, and its implementation, is no mean feat either for business managers or for IT, but the rewards for everyone are dramatic.

Here's a real-world application. Electrical appliances that you plug in at home today plug in equally well at the office or if you move across town. If you travel abroad, though, you likely need electrical adapters. When standard interfaces don't agree, you must adapt. Likewise, working with industry standards set forth by standards bodies enable autonomous entities (partners, customers, and suppliers) to dance at the ball.

Sweeping Unsightly Technology under the Rug

In the next chapter, we talk a lot about architecture. For those of you who already know a lot about systems architecture and want more nuts and bolts, we suggest you skim quickly through the "conceptual" chapters in Part II to make sure you understand what we mean by the terms we use. Then dive headlong into Part III, which we promise puts meat on the bones and gives you a lot to chew on — metaphorically, of course.

One big reason we think business managers are going to like SOA is because business gets to focus more on business and less on technology, SOA technology has the potential to become more invisible at the business layer, like the plumbing in a well-designed home. In this chapter, we give you an overview of what the business can expect from SOA.

SOA enables business managers and IT to talk in business terms that both sides understand. Without SOA, the IT developer and business manager typically use very different words for the same process: for example, creating an invoice. The IT developer is concerned with APIs (application program interfaces) and how to go about creating customer records from ten different Oracle database tables. The business manager describes the actual *business* process used to create an invoice. With SOA, a business service is a business service is a business service. How that business service is implemented in the technology layer is the purview of IT, and business managers need not worry about it or its associated technical jargon.

Understanding Why SOA Is Different

Perhaps you're skeptical. Perhaps, for as long as you can remember, the software industry has been promising yet another silver bullet to rid you of all business woes. We think now's a good time to repeat that SOA is not about "out with the old, in with the new." SOA is about *reuse:* taking what you have

and structuring it to allow you not only to continue to use it, but to use it securely knowing that future change will be simple, straightforward, safe, and fast. SOA is indeed a journey; it can't be built overnight. But organizations can begin SOA now and can benefit now. Ultimately, SOA renders a business more flexible — and IT more reliable, sustainable, extensible, manageable, and accountable.

We think SOA is the most important mandate facing business and IT today. And because SOA is a joint venture between business managers and IT, we present the basics necessary for everyone to come to the table with a good grounding from a conceptual level.

Chapter 7

Discovering the Main Components of SOA

In This Chapter

- Making SOA happen
- Leaving the driving to the ESB
- Introducing the SOA registry and repository
- Managing business process in a SOA environment

If you've followed this book to this point, reading all about Web services and business processes and composite applications, you may have already noticed (so far) that we do a pretty good job of hiding the gnarly bits of intricate technology that make all this possible. We think, however, that you may still need to know the critical components that make SOA SOA, so we carry on.

In this chapter, we introduce the major components of a service oriented architecture. This is the appetizer chapter. Many components are so important that they get entire chapters of their own, but we introduce them here to show them in relationship to each other and to help you with the big picture.

Making SOA Happen

We show major components of a service oriented architecture in Figure 7-1. The Enterprise Service Bus (ESB), the SOA registry and repository, the business process orchestration manager, service broker, and SOA service manager each have a role to play, both independently and with each other. The ESB makes sure that messages get passed back and forth between the components of a SOA implementation. The SOA registry and repository contain important reference information about where the components of a SOA are located. The business process orchestration manager provides the technology to connect people to people, people to processes, and processes to processes; the service broker connects services to services, which in the end, enables the flow of business process.

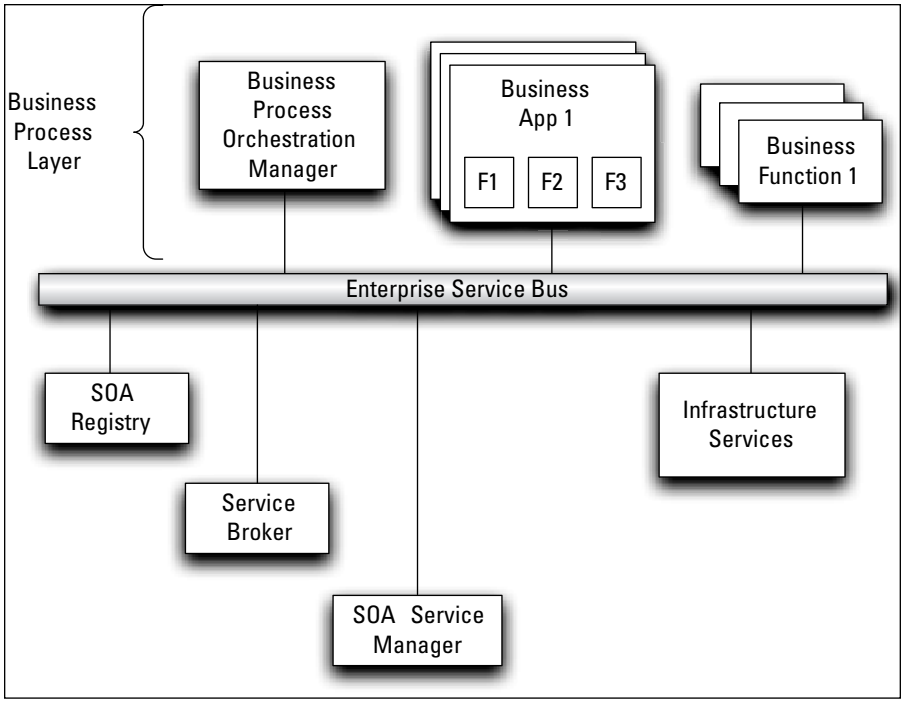

Figure 7-1: Fundamental SOA components.

The role of the SOA service manager is to make sure that the technology underneath the SOA environment works in a consistent and predictable way. The goal is to create an environment where all these components work together to improve the flow of business process. All these services are required to link unrelated technology components as though they were designed to work together. Later in this chapter, we provide additional information on business process management (BPM) as it relates to SOA.

When all these component parts work together and sing the same tune, the result is dependable service levels. A finely tuned SOA helps guarantee service levels.

Catching the Enterprise Service Bus

In service oriented architectures, all the different pieces of software talk to each other by sending each other messages — a lot of messages. The messages are critical to delivering end-to-end service. They must be delivered quickly, and their arrival must be guaranteed. If that doesn't happen, "end-to-end" service quickly becomes "lack of service."

To transport the messages between software components, SOAs typically use an ESB. The ESB is so important to SOA that some people think that you can't have a SOA without one. Other folks think that if you have an ESB, you have a SOA. Neither statement is accurate. You don't need to have an ESB to have a SOA, but you do need a way for the services to communicate with each other. The ESB is a reasonable and effective way to accomplish this goal. You can read more about the ESB in Chapter 13.

An ESB is a bit like a phone system. You can think of it as a special layer that runs on top of the network and provides a guaranteed messaging service for the most important messages on the network, including the messages that the components of SOA continuously send to each other.

Usually, in architecture diagrams, the ESB is represented as a separate pipe through which information and instructions flow. (Refer to Figure 7-1 to see what we mean.) In reality, it's not really a pipe. Rather, the ESB is a collection of software components that manage messaging from one software component to another. A software component connects to the ESB and passes it a message by using a specified format along with the address of the software component that needs to receive the message. The ESB completes the job of getting the message from the sending component to the receiving component.

Welcome to the SOA Registry and Repository

Somebody — or something — has to keep track of all the available business services that you created to represent your most important business processes. All those reusable components have to be recorded somewhere, and that somewhere is the *SOA registry.*

Think of the SOA registry as a kind of electronic catalog where you store information describing what each component does. It has two roles:

- One rooted in the operational environment
- One rooted in the world of programmers and business analysts

In the operational environment, the SOA registry provides reference information about software components that are running or available for use. This information is of particular importance to the service broker. (We talk a lot more about the service broker in Chapter 12.)

For programmers and business analysts, on the other hand, the SOA registry acts as a reference that helps them select components and then connect them to create composite applications that represent business processes. It also stores information about how each component connects to other components. In other words, the SOA registry documents the rules and descriptions associated with every given component.

The SOA registry is extremely important because it acts as the central reference point within a service oriented architecture. The SOA registry contains information (metadata) about all the components that the SOA supports. For that reason, it defines the "domain" of the architecture.

The SOA registry is where you store definitions and other information about your software components so that developers, business analysts, and even your customers and business partners can find the services they need. Business services are "published" in a registry to make them easier to find and use.

The idea of *publishing* Web services is critical to SOA. You can only reuse services that are available for reuse, which means they have to be published first.

Comparatively, the repository is like a central reference point within the software development environment. It stores the source code and the linking information used to build all the programs that run in the operational environment. The SOA repository feeds the service oriented architecture with changes and new components. The SOA repository works within the operational environment and takes on the responsible role of acting as the counterpart of the registry within the development environment.

Simply, here's the difference between the repository and the registry:

- **Repository:** Central reference point for all the components within the software development environment from which services are built
- **Registry:** Central reference point for definitions, rules, and descriptions associated with every service within a SOA environment

I never metadata I didn't like

Metadata means "data about data:" Data that defines, for example, what a set of data items contains. For example, the metadata for a database of books might be *author, title, publisher,* and *classification.* When it comes to SOA registries, metadata encompasses a lot more information: a full definition of each software component, for example, as well as the data that can be passed to it with each particular message it accepts as well as the data that it provides with each response it can give.

Mastering data takes some work

In a SOA environment, access to data becomes a service just like everything else. Enter the concept of *master data.* To avoid chaos, you have to have one common way of naming things related to your business. Here's a quick but elegant example: Most companies have something they call a customer, right? However, each department or division of that company might call the customer by different names. In the accounting department, a customer might be called *account;* in the sales department, the customer might be called *customer name.* Which one is right? If the accounting department doesn't talk to the sales department, it doesn't matter. If the planning organization doesn't care how much money it's making from customers, it also doesn't matter. But in the real world, each department needs to know that it's properly understood when communicating information about a customer to another department. A commonly understood and unifying way to identify information about important things like customers and products is *master data.*

Orchestrating End-to-End Services

Now that we gave you an idea of the purpose of the registry and repository, we need to show you how the business processes are orchestrated. Aside from the registry that holds the information about individual services, three components are involved in orchestrating an end-to-end business process: business process orchestration manager, service broker, and service manager.

Introducing the business process orchestration manager

The *business process orchestration manager* is a software component designed to connect a whole business process from end to end, flowing work from one individual or process to another until the entire business process is carried out. Business process development technologies provide a modeling capability that allows you to model a business process to produce a process flow pattern. The *process flow pattern* is, in effect, the set of instructions that the orchestration manager runs.

Most major software vendors offer tools focused on business process development, with many of these tools having been in existence for a long time. In the 1990s, business process development tools focused around automating workflow for document management systems. More recently, many vendors have repositioned business process development tools as BPM, often in connection with the process requirements of a service oriented

architecture. In fact, BPM is now so closely tied to SOA that many of the SOA software vendors have been investing heavily in new software development or strengthening partnerships to build their BPM solutions.

Every business has process flow, be it casual or formal, efficient or disastrous. Formalizing the process orchestration goes a long way toward documenting business process — which is a good thing, in case you're curious. When using business process management, you can monitor and document how your business actually works. We go into more detail about BPM later in this chapter.

Your friendly neighborhood service broker

Say that your components are published in the SOA registry. The business process orchestration manager strings together processes to make things happen. What more could you need? You need a *service broker.* You've probably come across some sort of broker in your life — a real estate broker, a mortgage broker, a stock broker. The broker is the deal maker; likewise, the service broker performs this function between components. It listens very carefully to all the constraints and concerns on both sides of the equation to make everyone happy.

The service broker is the component that actually makes all the connections between components work. It acts like a needle threading one component to the next in a business process. It uses information about the components it finds in the SOA registry and threads together the components for the workflow engine. The service broker gets things started and hangs in there to ensure that all the components of one business process are threaded together.

Service brokers are considered a middleware product, like the ESB. *Middleware* is the name given to the group of software components that we refer to as plumbing: They enable applications to be connected so that data can pass through from one application to the next. Because most ESBs can also act as a service broker, you often don't need a "standalone" service broker.

The SOA service manager, again

Way back when you could barely spell SOA, we put a black box in a picture and called it the SOA service manager (in Chapter 5, to be precise). By now, you probably understand that there's a ton of stuff in SOA to manage, so this SOA service manager has a lot of important work to do. The SOA service manager is the master conductor, the grand choreographer, the traffic cop, and all-around central point of control responsible for all under-the-hood

SOA orchestration. In essence, this type of orchestration is a form of virtualization. *Virtualization* is the ability to abstract computing resources so that they can be more easily used and managed in a variety of situations. The SOA service manager abstracts the SOA services from the technology that they run on.

Just to get a sense of who's talking to whom, think about business function components passing data and instructions to each other. At the same time, the workflow engine passes around instructions and data. The SOA service manager's agents send information to the SOA service manager, which in turn may communicate with plumbing services. There's a whole lot of talking going on. (We describe this in greater detail in Chapter 14.)

The SOA service manager interacts with the infrastructure services. If any of the components in the end-to-end service have any performance problems, the SOA service manager sends the details to the appropriate infrastructure services, and the infrastructure services try to fix the problem. Aren't you glad that plumbing is invisible to the business?

The SOA service manager is responsible for many things, but above all, it's responsible for ensuring service levels. It uses reports from monitoring agents (initiated by the service broker) to keep track of exactly what's happening. The monitoring agents report on the service level being achieved at each point in the process. The SOA service manager is then in a position to know when the service gets bad or when any part of it fails.

Managing Business Process under SOA

With all this discussion of registries, repositories, brokers, and buses, we want to remind you that the whole point of SOA is to make a business more manageable, more flexible, and more responsive to change. The primary culprit when it comes to instigating change is *business process:* how businesses do things.

Businesses constantly change *how* they do things while not necessarily changing *what* they do. For example, an insurance company might change the methods it uses to introduce new products or how it handles insurance claims, but when all is said and done, it still sells insurance. SOA enables business people to change business processes without having to focus on the underlying technological plumbing. You can concentrate on designing and improving business processes by threading together business services. IT can build composite applications from existing business functions, adding other functions or making changes where necessary. Together, business and IT can determine the flow of work from one person to another (or from a person to a process or from a process to a person) within the larger business process.

The origins of BPM

What we call BPM today is the result of a Western adaptation of management best practices that evolved primarily from Japanese manufacturing. The closest equivalent Japanese term is *Kaizen,* which can be defined as "continuous improvement" — or, perhaps more aptly, "to take apart and put back together in a better way." Beyond continuous improvement, BPM embraces other management methods, such as Total Quality Management and Six Sigma.

"But how do they do that exactly?" you may wonder. Thanks for asking. With all these business processes to manage, the somewhat obvious solution is *business process management* (BPM). BPM is the modern approach to designing and managing business processes, and many business managers and business analysts receive BPM training. All by itself, BPM has contributed significantly to the liberation of business from technology.

While BPM focuses on designing business processes effectively, SOA is an architecture that conveniently allows IT to align with business processes. It's only natural — yet very important — to successful implementation of SOA that SOA and BPM have converged, with BPM software tools becoming a natural part of a SOA development environment. Coupled with SOA, BPM is doubly powerful.

In addition to the BPM methodologies and approaches being bandied about by sharp-dressed consultants in countless corporate conference rooms, you'll find software tools out there that have been created specifically to help automate business process management. They are called, oddly enough, *BPM tools.* BPM tools organize workflows, thread together existing business functions, and create new functions.

With SOA, you want to harvest existing business functions by taking them out of their existing application homes — applications that have provided the "connective tissue" necessary to keep them functioning smoothly. The new connective tissue you need to house the harvested business functions comes in the form of either the business process itself or of a composite application. BPM tools are critical here because they help you design and manage just such business processes.

Business processes actually codify how a business works, and this very codification of business processes is a critical step for any organization subject to any kind of regulatory compliance, such as the Sarbanes-Oxley Act, the Health Insurance Portability and Accountability Act (HIPAA), and a host of others.

BPM enables businesses to monitor business processes, which can lead to continuous improvement by identifying possible changes in a process that could result in better efficiency. Over time, more and more business processes are tied to software. When supported by SOA, continuous business improvement becomes a lot easier because the underlying software is *loosely coupled,* meaning that it can be modified more easily when required. When business needs to change to address strategic opportunities and threats, the flexible service oriented architecture facilitates the change.

BPM terminology

If you find acronyms and abbreviations inherently confusing, the abbreviation BPM stands a pretty good chance of spoiling your whole day because it's used in three different ways:

- **BPM is a management practice.** BPM is a holistic management practice consisting of methods for efficiently aligning an organization with the wants and needs of clients. It promotes business efficiency but also promotes integration with technology. It also embodies the spirit of *Kaizen.* (See the sidebar, "The origins of BPM" for more on *Kaizen.*)
- **BPM tools (where m is for management).** If the BPM in "BPM tools" stands for *business process management,* the name implies that such software tools are built to assist in the implementation of BPM the management practice. Such tools can be comprehensive in terms of capability involving modeling, rules engines, workflow, process monitoring, and deployment.
- **BPM tools (where m is for modeling).** If the M in BPM stands for *modeling,* we're talking about a tool used to model business processes. Such as software development tool carries out only a fraction of what a business process management tool is likely to do. Nevertheless, the term BPM is used by the vendors who sell such tools.

BPM tools

Here's how a basic BPM tool uses the components of process management to make use of SOA:

- **It enables the creation of new business functions.** A developer may add whole new business functions or may simply add logic to run before or after an existing business function. To create new business functions, the BPM tool includes some way of specifying a software process. When a new business function is created, the BPM tool adds the function's details to the SOA registry, including information about how it links to other components.

- **It links business functions from existing applications.** The BPM tool refers to the SOA registry to identify business functions that are published there. It enables a developer to link them to make composite applications or slot them in at the appropriate point in the overall workflow. The BPM tool stores this information in the SOA registry.
- **It programs the business process orchestration manager to carry out the business process.** Using the BPM tool, business analysts design process flows and specify the movement of work from one person to another within a business process, linking the applications that they need to use for the tasks they have to carry out.

Figure 7-2 gives you a graphical representation of how a BPM tool accomplishes these three tasks.

Meticulous readers may notice that in Figure 7-2, the BPM tool isn't actually "managing" anything. Rather, it's merely *creating* a business process. More needs to be done if you want to ensure that the business process is performing effectively, from both these perspectives:

- **The computer perspective:** Does it have enough computer resources?
- **The customer perspective:** Is the customer or service receiver getting a prompt service?

We discuss such performance issues later in this chapter.

If you've read Chapter 5, you may remember that we define a *service oriented architecture* as a software *architecture* for building applications that implement business processes or *services* by using a set of loosely coupled black-box components orchestrated to deliver a well-defined level of service.

The "well-defined level of service" piece comes front and center now. Service *levels,* as the name implies, means that service is not so black and white. By way of illustration, think about service in a restaurant (as we did in Chapter 5): You could have great service or lousy service or so-so service. Or it may have started out great, but when it came to getting your check, your waiter couldn't be found.

Service levels in IT have become critically important in the past decade because business has become more and more dependent upon IT. Furthermore, IT itself has transitioned from using autonomous software packages serving a well-defined, limited set of users to using software delivered as a service over a network to huge numbers of users (just like telephone service and electricity). And just as you know if you've ever been caught in a power outage or experienced downed telephone lines, the lack of IT service (depending on when it happens and how long it lasts) ranges from somewhat inconvenient to ruinous. For many organizations, an hour of IT downtime costs millions of dollars.

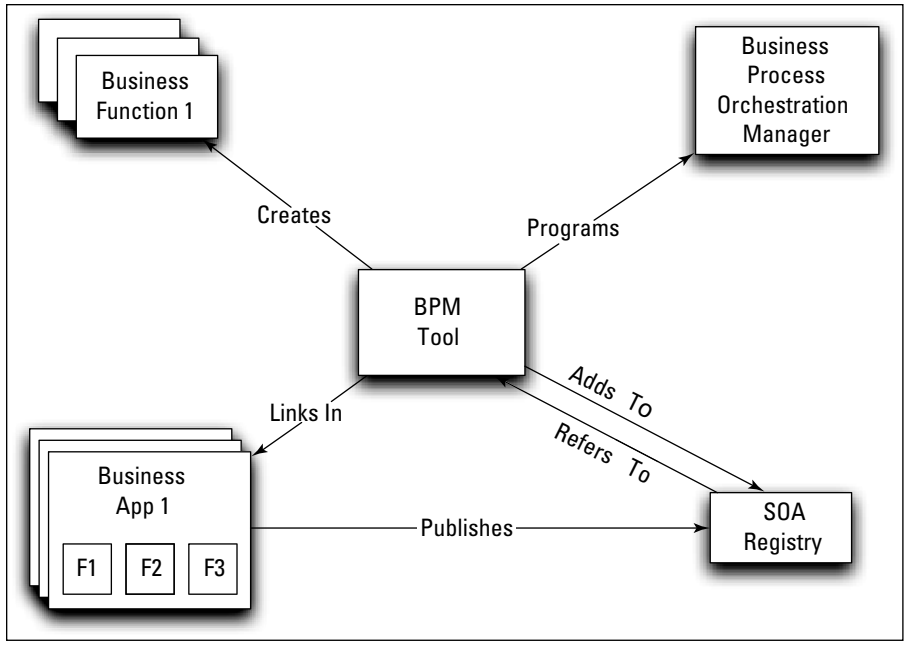

Figure 7-2: A BPM tool.

Thus, businesses that depend on services often enter into agreements with their service providers that guarantee a specific level of service, focused primarily on the availability and speed of service. For example, 100-percent uptime means that systems are available 100 percent of the time with absolutely no downtime — a virtually impossible task.

The ability to guarantee high levels of availability and high speeds of service usually implies higher levels of investment in computer systems, including having redundant systems in place in case of an emergency as well as having extra capacity should the need arise. Naturally, the higher the level of service, the more it costs. Thus, guaranteeing service availability at a 99.999 percent level is significantly more costly than guaranteeing availability at a 99 percent level. See the sidebar, "99.999 percent," elsewhere in this chapter, to find out why.

And perhaps you have gleaned, being the astute reader that you are, that a *service* oriented architecture — being all about service — has the potential to deliver variable levels of service. Will your SOA deliver good service, bad service, so-so service, intermittent service, or unpredictable service? Obviously, anything less than good service (or maybe even great service) will put your entire business at risk.

99.999 percent

Perhaps you're perplexed by why adding 0.999 percent of reliability and availability is so very expensive, and why 99.999 percent should cost so much more than plain old 99 percent. We'll try to explain. (99.999, by the way, is sometimes referred to as *Five Nines* and is said with a great deal of awe and respect.)

Say, for example, that you have a server that's up most of the time — it rarely crashes. Maybe it's down three days per year. That's roughly 99 percent available. Now if you really truly need 100 percent availability (or as close as you can get), you have to do more than just buy another server. You may have to buy multiple servers; dual up all the networking equipment; invest in failover capabilities; and guarantee that you have no single point of failure, such as the power supply. You may need a generator. Do you begin to see the cost of being "always on"? The bigger and more complicated your system, the greater the number of vulnerabilities or points of failure there are likely to be. Ensuring availability (and extra capacity for peak loads) is not a trifling expense.

Businesses whose very livelihoods depend on their being available 24/7 go to great lengths to ensure that availability. For some Web-based businesses, an hour of downtime to business critical systems can cost more than $1 million in lost revenues. Amazon.com for example, currently processes between $1–2 million per hour. So three days of downtime could mean $72 million in lost revenues. Ouch.

Service oriented architectures must make composite applications and business processes available, reliable, and predictable. Although the responsibility of all the choreography needed to make and keep service levels high falls squarely on IT, we think everybody should know the basic principles. And understanding the basic principles can go a long way toward bettering the communication flow between business and IT, which are jointly responsible in the new world order.

Application failures: Let us count the ways

Understanding service levels as they apply to application availability means understanding how and why applications fail as well as the consequences of the failure.

Applications can fail because the hardware they run on fails or because the network connecting the users to the application fails. Or the application itself can fail, or the operating system running the application can fail, or some of the management software managing the application can fail . . . and on and on.

When software components fail, it takes time to find the cause of the failure and to get the system back into action. If finding the cause can be done automatically, so much the better. Even so, this could take more than a minute or two. If it isn't automatic, it's likely to take hours.

Not all problems cause outright failure. Some may simply slow down the application. Just as applications can fail for many reasons, they can also slow down for many reasons.

Measuring service levels

Here are the ways to know that an application is delivering the service business users require:

- Define the service level that the business needs.
- Measure the application's activity to see whether it achieves that level.

Because application interruptions are sporadic, you need to measure service constantly and then average it over a given period of time — a day and/or a week and/or month and/or a year to arrive at a meaningful number.

The measurement of the service level needs to be automatic, which can present a problem. While the service level of individual components of a business process is usually easy to measure, both in terms of availability and response time, measuring the service level of an end-to-end process can be complex. Fairly recent new software tools called Business Activity Monitoring (BAM) tools have emerged to help in this area.

Here's an example of a detailed definition of a service level for an order-processing application:

- Application to be available 99.9 percent of the time every weekday from 6 a.m. to 10 p.m. EST.
- In the event of a failure, the application should recover within 20 minutes.
- The response time for order inquiries, changes to orders, and entering new orders should average 1 second and should never take more than 2 seconds, 99.9 percent of the time.
- In the event of degraded service occurring and response times slowing, normal service should be restored within 1 hour.

Defining the service level this way serves both business and IT well. Compliance with this service level agreement would mean that the order-processing application would be unavailable for (at worst) 4 hours, 10

minutes during the whole year — little more than one-half of one business day. The response time could be poor for as much as 4 hours, 10 minutes in the year. In the worst-case scenario, the application is unavailable for 20 minutes when it does fail. And when service degrades, it's always back to normal within an hour.

By the way, note that what we defined here is a *technical service level.* A business might also like to define a *service level* for specific business processes. Thus the goal might be to fulfill a customer's order and dispatch it within 24 hours of receiving the order, 99 percent of the time. Ultimately, if you have well-defined IT service levels, it becomes much easier to monitor the service levels of business processes and improve them.

IT needs the right set of computer equipment and supporting software to deliver a service level of this kind for the application, including the appropriate failover capability as necessary. And if the application gets more users, IT needs to upgrade the computers to keep pace with the resource requirements.

End-to-end service

Delivering high levels of service for a stand-alone application isn't so tough; after all, you have relatively few people to make happy. However, when you involve a network and begin to deliver services across that network, life ceases to be simple.

With SOA, you might (and probably will) link components from different applications. However, the service level that you want to deliver in the new application that you create this way isn't necessarily the same service level delivered by each of the different applications you're taking components from. In fact, they could differ from each other, and your new application could have a service level different from any of its components.

Do you have a headache yet? Consider the development of a new order-processing application that's very similar to an old one but that's written specifically to work on the Internet. You build a few new software components and use most of the components from your old order-processing system. The service levels for the old system might be acceptable, except for one thing: Because this is an Internet application, it needs to run 24/7.

So why not just leave the old application running all the time? Simply put, the old application wasn't designed to run all the time. It was designed to allow data backups to occur at night and for data to be extracted from the database at night when the application wasn't running.

Of course, this isn't an unmitigated disaster. You could just change the old application in some way. However, changing the old application takes time, and it will have to change again whenever your business needs change again. The reality of most businesses is that change is the only thing that is predictable. To support continuous change, linking software components makes much more sense than recoding applications.

When you consider linking many software components from many applications, a bigger issue emerges. Applications that were built to deliver specific service levels linked end-to-end must deliver service end-to-end. Delivering dependable service levels means controlling the end-to-end process, which means you need the SOA service manager.

Taking just one more look at the Process Manager

To round this chapter off, we take another look at the component we simply refer to as the Process Manager in Chapter 6. We were making life easy for you by representing it as a single component. In reality, the activity of process management is carried out by three components: the SOA registry, the workflow engine, and the service broker. (See Figure 7-3.)

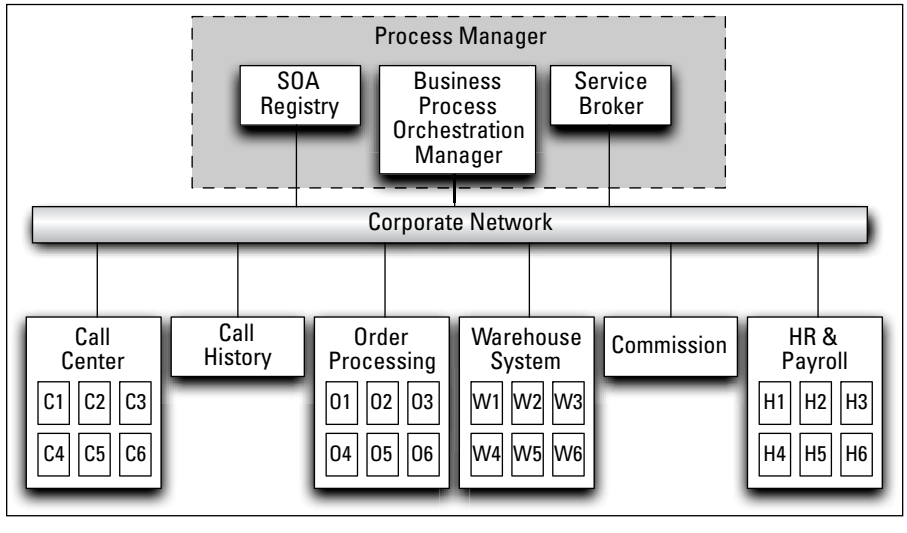

Figure 7-3: Process management.

This diagram ought to look suspiciously like Figure 6-5 in Chapter 6, because it is. The difference is that in this version, we break the component we called the Process Manager into three separate components:

- The **SOA registry** stores information describing what each SOA component does so that business analysts or programmers can select components and connect components to create composite applications. The registry also stores information about how one component connects to another.
- The **business process orchestration manager** connects a whole business process in an end-to-end manner, flowing work from one individual or process to another while the business process is carried out.
- The **service broker** actually makes all the connections between components work. The service broker is a needle that threads together all the software components of a business process by using information it gleans from the registry.

For SOA to achieve the sophistication that we predict it can, you need to pull together the pieces of hardware, software, services, infrastructure, process — and even people — so they act like a single system. The greatest benefit of SOA is that you can change the parts and still have a well-oiled machine.