

Service-Oriented Architecture: A Primer

By Michael S. Pallos

Service-Oriented Architecture (SOA) is the next wave of application development. SOA lets heterogeneous environments and applications exist while leveraging existing applications and infrastructure. This fosters code reuse, reducing costs and risks while speeding time-to-market. Has the "silver bullet" finally arrived? No, but we have seen the consolidation of a wealth of lessons learned over the past 30 years of IT development.



During the client/server era, Unix servers were going to replace the mainframe. The enterprise was to become a federation of presentation devices, business servers, and database servers. We've seen that the mainframe and legacy systems aren't going anywhere. New development languages come, but never seem to go. The current bandwagon is Java and eXtensible Markup Language (XML). Some developers believe everything should be written in Java. This isn't cost-effective, nor is retraining staff. Current resources must be leveraged. Most organizations have a mix of COM+, CORBA, asynchronous messaging, synchronous messaging, and hard-coded point-to-point. The mind-set of one-size-fits-all isn't realistic. Accept diversity and leverage the best that each technology has to offer.

The buy-vs.-build decision is best reached on a case-by-case basis. Sometimes, a package should be purchased. When purchased, integration into the enterprise is required, expected, and demanded. Learn to integrate purchased packages into the enterprise. Organizations should learn to take advantage of current resources, including developers, software languages, hardware platforms, databases, and applications. SOAs offer a practical approach to delivering business return while leveraging many aspects of the existing organization. SOA represents an evolution of lessons learned. Corporations that adopt the SOA paradigm will be at a strategic advantage. This article explains why — and how to implement an SOA.

Defining SOA

SOA is the aggregation of components satisfying a business driver. It comprises components, services, and processes. Components are binaries that do specific tasks. These binaries each have a defined interface and usually one job (e.g., “validate user” or “obtain credit rating”) to do well. A service is simply a grouping of components (executable programs) to get the job done. Example: “process loan application.”

The key focal point of an SOA is the business process. The grouping of components satisfies the process, letting the application process pattern more closely represent the business. This higher level of application development provides a strategic advantage, facilitating more focus on the business requirement. The business driver is satisfying the requirements, as opposed to the technical mechanism of application development. A service is built using components. These components are executable programs. So development and ownership costs, as well as implementation risk, are reduced.

Why Invest in SOA?

SOA leverages an organization's existing investment by taking advantage of current resources, including developers, software languages, hardware platforms, databases, and applications. Implementation details — such as components, servers, client invocations, and databases on which programs run — are independent from the process definitions and patterns. This adaptable, flexible style of architecture provides the foundation for shorter time-to-market, and reduced costs and risks in development and maintenance.

SOA can be evolved based on existing system investments rather than requiring a full-scale system rewrite. Per leading research companies, organizations that properly use SOA achieved a 40 percent gain in development resource utilization. They also reduced costs and risks while easing technical support management and monitoring by a factor of 10.

Components

Components are executable programs. The greater the granularity, the stronger the ability to reuse and leverage. Components leverage existing resources; their use promotes reusability, reducing risks and costs. Grains of sand in a desert are analogous to lines of

code. They're difficult to pile up into a cohesive entity, prone to blowing apart in a strong wind, and difficult to differentiate when piled into mounds. However, bricks made of sand are analogous to executable programs (components). They present a solid look and feel, they carefully shield the user from examining their internal structures, they do one thing well, and you can build almost any imaginable structure by placing them together in a set pattern. One would rather build with bricks than sand.

Component Types

Our industry currently recognizes three types of components: technical, business, and application templates.

Technical components are technology-specific and independent of the business vertical. Technical components are required for application development, but aren't specific to business drivers. Examples include error handling, data compression, and security. Technical components are often static, self-contained, highly reusable, and relatively inexpensive. Examples include ActiveX controls for Microsoft's Visual Basic and SWING controls for Java.

Business components handle specific tasks based on business rules. Credit card approval is an example. Business components are currently the least common since vertical knowledge is required. The market for pre-built business components, sometimes called application building blocks, is growing as vendors take an industry domain focus.

Application templates are the aggregation of existing components satisfying a major portion of business requirements. This toolbox offers an attractive alternative to in-house development. Solid application templates allow for easy assimilation of application logic. They bring many benefits and leverage existing systems.

Services Defined

Services are the logical grouping of components to satisfy a business requirement. While a collection of components can comprise a business service, a composite of business services can comprise a process flow. Details are hidden from the application developer.

Service Types

SOAs are designed for programmatic access, not terminal access. So the inter-



**Service-Oriented
Architecture (SOA)
is the next wave
of application
development.**

face to the service must be understood. Three choices for incorporating business services are via:

- Contracts
- Messages
- Repository.

A **contract** is a defined agreement responsible for the interface definition. Interface Definition Language (IDL) is one example of a contract. The interface is described in a pseudo code text file adhering to a predefined syntactical format (IDL). Next, the IDL is processed through a pre-compiler. The output is a client_stub and a skeleton (or server_stub). During this process, the object repository is automatically updated. The infrastructure is responsible for the complexities of the client finding the server. The object repository has details of the requestor/responder relationship. The developer only has to link the stubs with the application. The infrastructure details are hidden.

Messages are another mechanism for defining service interfaces. A message is simply a contiguous block of data, typically combined with a header or trailer section that contains the logistical transport details. Messages are ideal for asynchronous communication in a loosely coupled messaging application.

The last mechanism for defining service interfaces is via a **repository**, simply a common reference point to cross-components and cross-service resources. This central location allows dynamic definition and manipulation of services.

SOA Anatomy

Now that we've laid the foundation, let's build a business service — a loan application — using all the concepts required to implement an SOA.

SOA services are designed for programmatic access, not terminal access. This means they're called by other programs. This brings us to the last piece of the SOA puzzle: client applications. A client application is the program that calls the service. This is extremely powerful. If I've created a client for a loan application, and that client calls a service named "loan offer," I can change the logic of loan offer by simply adding a component. This requires no changes to the client. Let's examine how.

Components are a program, a binary or chunk of code that has one purpose in life. In Figure 1, we have three compo-

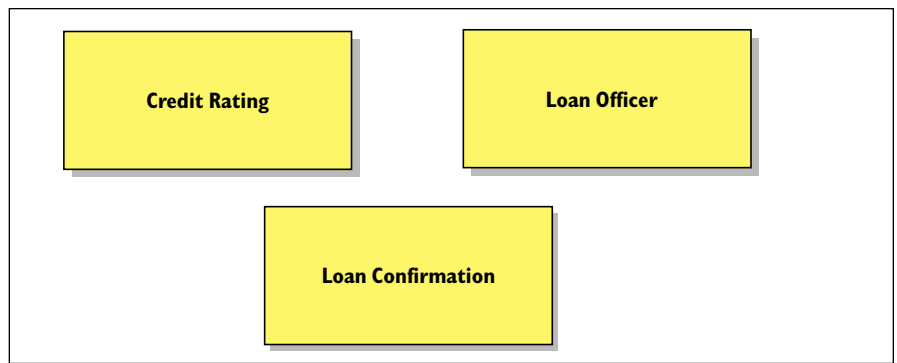


Figure 1 — Current Component Library (Three Binaries)

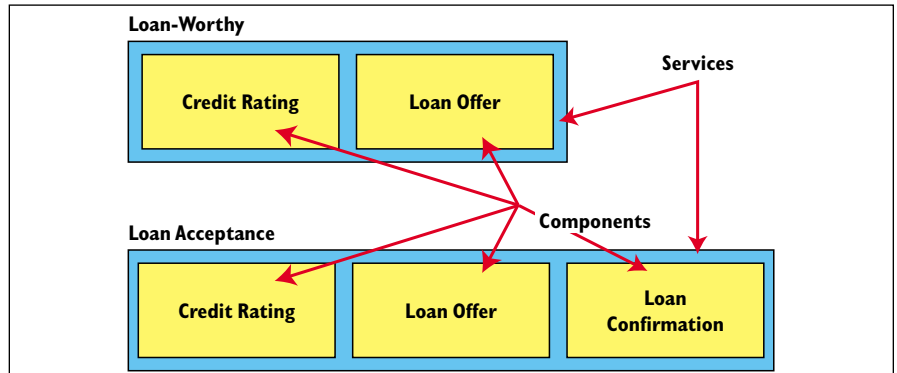


Figure 2 — Services (Composite Services) — Logical Grouping of Components

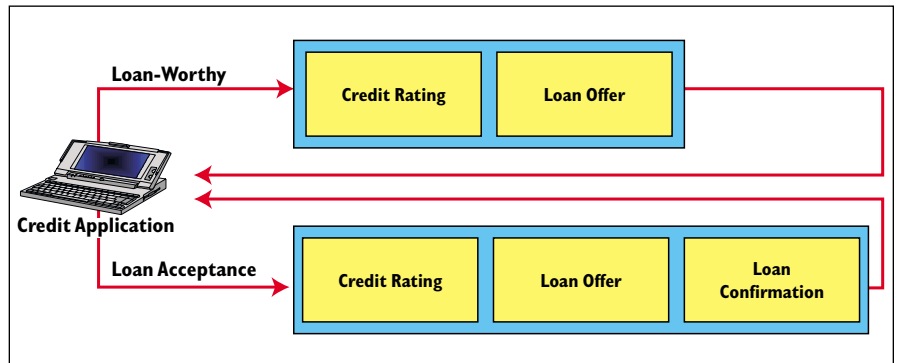


Figure 3 — Client Application Requesting Two Services

nents with three distinct purposes:

- Credit rating assesses an applicant and assigns a credit rating (a value between 0 and 800).
- Loan offer assesses whether to grant a loan based on an applicant's credit rating and, if so, the amount and applicable interest rate.
- Loan confirmation processes the loan.

A service is a logical grouping of components required to satisfy a particular business requirement.

Our business driver is to make money by lending money with a high probability of a return on our investment. We first identify if an applicant is loan-wor-

thy. If she is, we lend our money at a specific interest rate. Based on these requirements, we'll create the two services shown in Figure 2:

- Loan-worthy: Is the applicant worthy of a loan?
- Loan acceptance: This is the processing of the loan. Only applicants who are loan-worthy will use this service.

Client applications (see Figure 3) represent the program requesting the business service. The client gathers the applicant's information and requests the services. Our client application also displays the results from the services to the end user. We first gather user name and

social security number and call the loan-worthy service. If the client can obtain a loan from us, we then call the loan acceptance service.

SOA Benefits

SOA delivers several significant benefits:

- **Leverage initial investment** — SOA leverages the initial organization's investment in hardware, development languages, development resources, and architecture. Since a business service is the aggregation of existing components, the responsibility of compatibility among the components isn't the developer's; it belongs to the SOA framework. So if our resource pool is composed of a large population of COBOL developers, we may develop components in CICS COBOL. Once developed and tested, this component is made available to the enterprise.

Using the existing component only requires knowing the interface and name. The component internals are shielded from the outside world. Also shielded are the infrastructure complexities of getting the data to flow through the components. This component anonymity lets organizations leverage current investments, building services from a conglomeration of components built on different machines, running different operating systems, developed in different programming languages and running on diverse architecture.

- **Infrastructure commoditization** — Many aspects of infrastructure development are consistent across all applications. Purchase these components and give your infrastructure commodity status.
- **Faster time-to-market** — As organizational component libraries grow, time-to-market shortens exponentially. New initiatives support rapid development of new business services via reuse of existing services (services can call other services) and components whenever possible. Reusing existing components reduces requirements-gathering time, design time, development time, and testing time. If a component works in its current state, we simply add the component to our service and it should continue to work. No functional development or testing are required, only sys-

tem testing on the complete business service.

- **Reduced cost** — As business demands evolve and new requirements are introduced, the cost of creating technology to satisfy these requirements is greatly reduced when portions of the process flows exist as components in the enterprise.
- **Risk mitigation** — By leveraging existing components, we're less likely to introduce new failures into the creation of a business service. If one reuses a functional component, the probability is high that it will continue to work properly. We also reduce risk by allowing the organizational development staff to maintain its current Integrated Development En-

The SOA is the next wave of application development.

vironment (IDE), programming language, operating system, and databases. The learning curve for development is greatly reduced if the staff can maintain its area of expertise.


- **Continuous improvement cycle for business process** — An SOA provides a clear representation of process flows identified by the order of the components used in a particular business service. This provides business analysts an ideal environment for monitoring business operations. Process modeling is reflected in the business service. Process manipulation is achieved by reorganizing the pieces in a pattern (components in a business service). This allows changing the process flow while monitoring the effects and facilitates continuous improvement.
- **Process-centric processing** — Most architectures tend to be program-centric. Applications are developed for

the programmer's convenience. Often, process knowledge is spread between components. The application is much like a black box, with no granularity available outside it. Reuse requires copying code, incorporating shared libraries, or inheriting objects. In a process-centric architecture, the application is developed for the process. The process is decomposed into a series of steps, each representing a business service (or sets of components). In effect, each service or component functions as a sub-application. These sub-applications are chained together to create a process flow capable of satisfying the business requirement. This granularity lets processes leverage and reuse each sub-application throughout the organization.

Summary

The SOA is the next wave of application development. It's reasonable to wonder what makes SOA different from other technology triumphs such as:

- Object-oriented development
- Reusable C++ class libraries
- Client/server computing (where 80 percent of all projects failed)
- Enterprise JavaBeans (EJB).

The difference is that the SOA leverages an organization's existing investment. Capitalizing on current resources — including developers, software languages, hardware platforms, databases, and applications — SOAs offer a sensible approach to delivering business return. Participating in an SOA will reduce costs and risks while boosting productivity. It's faster, better, and cheaper. Who says you can't have all three? 

About the Author



Michael S. Pallos is a senior solution architect for Candle Corp. He actively architects enterprise solutions for Candle's Consulting & Services clients. He has 15 years of experience in distributed systems and holds numerous copyrights for international applications he developed. e-Mail: Michael_Pallos@candle.com; Website: www.candle.com.