AS SEEN IN...
SOAWebServices JOURNAL

Service Components

The Quickest Path to an SOA

WRITTEN BY **ATUL SAINI**

The emergence of the Enterprise Service Bus (ESB) over the past two years has spurred the deployment of componentized applications based on a Service Oriented Architecture (SOA). SOA enables the development of business systems and processes with loosely coupled components (often called services) facilitating business agility. Much of the industry's focus has been on the architecture of the underlying ESB infrastructure that supports an SOA.

Standards adoption has made it easier to learn new tools, but basic ESB infrastructure improvements haven't significantly reduced the effort involved in deploying and managing new business processes. That's because an ESB is simply a platform that unifies the advantages of multiple previous generations of middleware. This article illustrates that a comprehensive model for service components – the application-level modules wired together over an ESB to support SOA – facilitates rapid deployment of componentized, extensible, agile business processes.

What Are Service Components?

A service component is an independent software application that executes a specific business-level function. For instance, while operations such as "update database table" have a technical meaning, a service component always executes a business-level function. As an example of a service component, consider an application that updates a general ledger in an accounting system or product information in a supply chain system.

This article explores core attributes that a service component must support to enable rapid deployment of an SOA. Since these are mostly independent of the underlying infrastructure over which the modules are deployed, service components can – with basic support from each target platform – be deployed over an ESB, an application server or even a raw Message Oriented Middleware (MOM) platform to create an enterprise SOA.

Key Service Module Characteristics

SOA deployment normally involves orchestration of multiple service components to solve a particular problem. You can reduce the time it takes to deploy an SOA if your service components support the key characteristics below.

A service component:

- ***Can be self-describing:*** A service component is a self-sufficient (self-contained) bundle of software, which can be exchanged between multiple entities and reused with no external dependencies. Service components typically expose interfaces in Web Services Description Language (WSDL) or XML, and can be searched and queried with standard tools, including tools that support the Universal Description Discovery and Integration (UDDI) standard among others.
- ***Can be automatically deployed (usually without manual installation):*** A service component can be remotely deployed, monitored, updated, and controlled over a highly distributed infrastructure environment from a single point of control. Auto-deploy capabilities require that all implementation dependencies (e.g., libraries or other platform-specific files and data) of the component reside in the component when the component is remotely deployed.
- ***Can be developed in multiple languages (C/C++/Java/.NET):*** A service component allows the development of interfaces in any supported language native to a platform without having to create or use language translation code. Since a service component is completely modular, the language used to implement the underlying logic isn't relevant to the exposed interface. So a single distributed SOA process can involve interactions between service components developed in different programming languages. Support for multiple languages is especially attractive for deploying an enterprise SOA since few enterprises have homogeneous environments.
- ***Is configurable, using a contextual user interface (typically specific to each service component):*** Easy configurability for each service component used in a business process implementation is vital for rapid SOA composition and deployment. A service component lets you bundle a customizable configuration interface (specific to each such component) with the component. This gives SOA designers maximum flexibility in deploying SOA processes, while enabling the underlying platform to handle the entire component lifecycle from discovery to configuration, deployment, and runtime management.
- ***Exposes changing, dynamic service definitions on a configuration (usually with input and output schemas that can change based on the configuration):*** For maximum flexibility, a service component must let the invocation interfaces of the component be dynamically generated based on the user's configuration. Introspection of the configuration parameters

of the service should automatically lead to the generation of custom input and output interfaces. The ability to associate different input and output schemas (i.e., interfaces based on configuration) supports the development of a reusable service component that can "adapt" the invocation interface based on requirements of a particular instance of the module. The generated interfaces can be obtained in WSDL, allowing the invocation of the component as a Web Service among other options.

- ***Has pluggable synchronous and asynchronous invocation interfaces:*** An important requirement for flexibility in process design is to ensure that components can be invoked using both synchronous and asynchronous interfaces. This means the interfaces of the component are completely abstracted from the business logic that the component executes. This separation provides the ability to reuse the same service component in different contexts. For instance, a database update can be triggered asynchronously each time mail arrives in a specified mailbox; in a different context, the database update can be invoked synchronously by a Web portal during order processing. Detaching an invocation interface from the body of a service component lets a single technology framework support both Event Driven Architecture (EDA) and request/reply semantics with seamless interoperability – an important goal for any SOA platform.
- ***Has "external" transport bindings, allowing transparent invocation over multiple transports:*** Most enterprise environments have multiple middleware transports in deployment, often with significant investments in technology, including raw TCP sockets, Common Object Request Broker Architecture (CORBA), HTTP, SOAP, Java Messaging Service (JMS), MQSeries, and TIBCO/Rv. External transport bindings let you use a service component with the most optimal transport (or an already available transport) in a specific deployment without having to use transport adapters. External transport bindings permit maximum flexibility in development since different service components in a single business process can use separate transports. The ability to switch transports "on-demand" protects legacy investments and enables optimal performance.
- ***Can be recursively grouped as sub-components in a larger composite component:*** Service components let you create larger components (and processes) that represent larger reusable blocks of functionality specific to a given problem. This recursive grouping capability simplifies the maintenance of large processes (due to compartmentalization), allowing separate portions of the process to be developed and tested independently, potentially by different teams of developers, before the production environment is updated.
- ***Supports multiple deployment options (as a standalone executable embedded inside a container, etc.):*** Flexibility of deployment requires a service component to support multiple deployment options, including deployment as a standalone executable or embedding in a Java 2 Enterprise Edition (J2EE), Microsoft .NET, or Web container. This enables reuse of existing infrastructure investments without adding complexity or managing multiple independent deployments. Multiple deployment options are especially useful for small deployments where independent hardware and software management isn't warranted.
- ***Can be monitored and managed at runtime using standard and remote consoles:*** Agility of business process composition requires service components (which usually run strategic processes for a business) to be individually monitored and managed, with the ability to alert users of potential problems in real-time. The best systems enable component integration with third-party tools such as HP Openview, IBM Tivoli, etc., reducing reliance on redundant management and monitoring systems and providing proactive notifications of potential trouble even before it surfaces.

Conclusion

With service components that support the attributes listed in this article, it's feasible to deploy flexible business processes over a wide variety of platforms. Even purpose-built "SOA platforms" can, at best, support only core infrastructure-level features such as fault-tolerance, re-routing of messages at a transport level, quality-of-service, scalability, and performance. In and of itself, the infrastructure can do little to ease the creation of flexible processes since speed of deployment and composition is largely orthogonal to the infrastructure. For maximum agility, a process has to be composed of modular service components knit together over multiple transports using multiple invocation methods, whether synchronous or asynchronous.

Business:

- Service components represent the industry's first "Lego block" concept, enabling composition of networked processes by relatively non-technical business analysts.
- Applications based on service components are flexible and can be changed without stopping processes, enabling a real-time response to changing conditions.
- Service component-based processes enable reuse of existing infrastructure software assets, reducing the overall cost of deploying new business processes.

Technology:

- Infrastructure platforms such as the ESB or even the new SOA platforms are basically like previous generations of middleware; they don't, by themselves, speed up SOA deployment.
- Service components drive an engineering discipline that forces developers to modularize their solutions, making them more maintainable.
- Service components support both event-driven and request/reply semantics, enabling modular solutions that are easy to change in response to business requirements. ■

About the Author

Atul Saini is CEO and CTO of Fiorano Software Inc. Under his leadership, Fiorano has emerged as a recognized leader in the standards-based integration middleware business.
atul@fiorano.com