Gartner

Research

Publication Date: 28 February 2005

ID Number: G00126127

Clarifying the Terms 'Event-Driven' and 'Service-Oriented Architecture'

Roy W. Schulte

The term "service-oriented architecture" has evolved in several directions. Developers should understand its multiple meanings and appreciate the distinction between "service-oriented" and "event-driven." These two design models are key to effective distributed applications.

WHAT YOU NEED TO KNOW

The modern interpretations of the terms "service" and "service-oriented architecture" are looser and more varied than the original definition of these terms. "SOA" is now popularly applied to event-driven relationships in addition to its traditional usage describing classical client/server relationships. Regardless of whether the original or modern semantics are used, architects should distinguish between the client/server and event-driven models and apply them appropriately to meet their local business requirements.

ANALYSIS

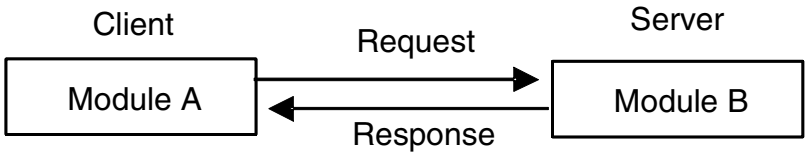
In popular usage, the meaning of the term "service-oriented architecture" (“SOA”) has expanded beyond its original intent and become ambiguous. The multiple new meanings have given rise to confusion about the difference and overlap between "service-oriented" and "event-driven" relationships between software modules. We describe the background of SOA, the current use of the term, and the distinction between the service-oriented and event-driven models.

Historical SOA

Alexander Pasik, then an analyst at Gartner (now an entrepreneur and adjunct associate professor in Columbia University's Computer Science Department), coined the term "service-oriented architecture" in 1994 for a class on middleware that he was teaching. He used "server orientation" to mean client/server, in the classical computer science sense of the term.

Classical client/server is a model of interprocess communication where one process, the client, sends a request message to another process, the server, which performs a task and then sends a response back to the client (see Note 1). Client and server are roles. A process ("A" in Figure 1) is a client because it sends a request, and another process ("B" in Figure 1) is a server because it responds to the request. Client/server relationships have been common in computing for decades, particularly for implementing interactions between an application program and a system software utility. For example, an application may invoke a print server to put a file out on a printer, or it may invoke the operating-system file manager to access a file. However, in 1994 it was relatively uncommon to organize a section of business application logic as a server to be invoked by another (client) part of the application.

Figure 1. Classical Client/Server Model



Source: Gartner Research (February 2005)

The classical concept of client/server is still taught in computer science classes because its properties are helpful for building and maintaining complex systems, particularly distributed systems. A client is unaware of the server’s location, so the server can be on the same machine or on a distant machine (this is location transparency). A server can be invoked by disparate

clients, so it can provide the same function to different applications. The inherent modularity enhances server reuse and application maintenance.

Pasik was driven to create the term "SOA" because "client/server" had lost its classical meaning. Many in the industry had begun to use "client/server" to mean distributed computing involving a PC and another computer. A desktop "client" PC typically ran user-facing presentation logic and most of the business logic. The back-end "server" computer ran the database management system, stored the data and sometimes ran some business logic. In this usage, "client" and "server" generally referred to the hardware. The software on the front-end PC sometimes related to the server software in the original sense of client/server, but that was largely irrelevant. To avoid confusion between the new and old meaning of client/server, Pasik stressed "server orientation" as he encouraged developers to design SOA business applications. Gartner was the first in the industry to use and publish the term "SOA" (see "'Service Oriented' Architectures, Part 1" and "'Service Oriented' Architectures, Part 2" by R. Schulte and Y. Natis, April 1996). In those reports, SOA was explicitly considered to be a synonym for the classical (software-based) client/server model as applied to business application logic.

SOA in 2005

History has repeated. During the past five years, "SOA" has itself lost its original meaning through reinterpretation. It now means different things to different people, with varying degrees of fidelity to its original intent.

1. SOA as Distributed Computing

In its broadest interpretation, SOA is used as a synonym for distributed computing. If two modules run on different computers and send data to each other, some would say that it is SOA. Some people call all of the modules, even the clients, "services."

2. SOA as a Web Services Application

Others apply "SOA" to any application that uses one or more of the Web services standards — particularly Simple Object Access Protocol (SOAP) or Web Services Description Language (WSDL).

3. SOA as the Relationship Between Encapsulated Modules

A third, and more meaningful, approach is to define SOA in a way that reflects good practices in distributed application design. In this approach, SOA is defined as "the relationship between clients and services," where "service" means a software module that has certain properties. There are multiple variations of this definition depending on which properties are required. One interpretation requires all of the following properties, while other interpretations make some of these (particularly the last three) properties optional. A service is a module that:

- Can be reused in disparate applications because it can be invoked by disparate clients.
- Performs exactly one task and has one defined set of inputs and outputs.
- Has an interface (the specification of its input, output and error-handling behavior) that is logically distinct from its implementation (the code and data that carry out the function).
- Has implementation metadata so that the identity and location of the service modules can be dynamically discovered at runtime. Modules can be moved or a new version of the service can be swapped in without affecting any client, as long as the interface does not change.

- Has interface metadata stored in a repository that is available to various developers at development time.
- Has interface metadata that is also available at runtime so that interface attributes can be dynamically discovered. An intelligent client program or an adroit end user could generate or adjust a service request at runtime.

This third approach is more narrow and useful than the broad "any distributed application" concept in Definition 1 above. It is a partial overlap with Definition 2 because the interactions between the modules may be implemented as Web services, but Web services are not required by this third definition.

Confusingly, the word "service" sometimes refers to the task to be performed (as in a "get account balance" service), but elsewhere it refers to the software module (the implementation). We prefer to call the module the "service implementation," which is a type of "business component," rather than overload the word "service," although we concede the popularity of the use of "service" to mean "module."

Modern SOA vs. Original SOA

There are dozens of other modern definitions of "service" and "SOA," although the best are variations of the third approach above. Note that none of the modern usages of "SOA" fully adheres to Gartner's original 1994 notion of SOA. The original SOA inherited the following properties from the classical client/server model:

- When it runs, a server has an identifiable client. Client/server relates exactly one client and one server module (just as a call statement invokes exactly one procedure, never zero or two). Client A can have a relationship with multiple servers in succession, and Server B can serve multiple clients in succession, but each relationship is a one-to-one contract.
- The client directs the flow of control by specifying which service is to be invoked, and when.
- The client delegates some of its work to the server and is dependent on the server. A server can, in turn, act as a client to further delegate work to another subprocedure, and so on.
- The client and server are logically connected to each other and are bound through software. The coupling is "loose" if it works indirectly through an intermediary, as in the case of Web services, for example. However, even loose couplings still use a "find," "bind" and "invoke" sequence (in Web services, the bind and invoke are technically combined into one operation).

Event-Driven Applications

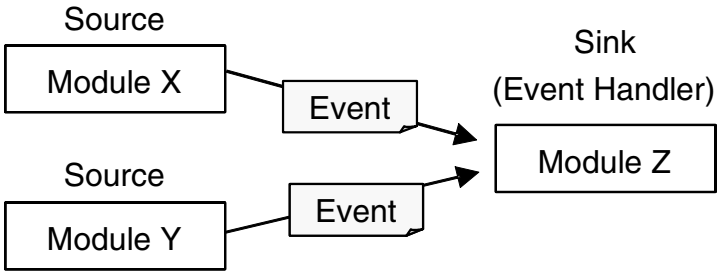
Not all relationships between modules in a distributed application conform to the classical client/server model. Consider this example:

A real-time data feed provider module (X) in London emits a continuous stream of messages ("events") recording transactions of people buying euros with U.S. dollars (US$). Simultaneously, another data feed provider (Y) in Tokyo emits a stream of events recording transactions of people buying US$ with euros in Japan. A program trading engine module (Z) in New York receives both streams of events and compares prices to detect arbitrage opportunities (see Figure 2). When it

notices a difference in exchange rates, it issues a buy order on one exchange and a sell order on the other (or it notifies a person that this profit opportunity exists). X relates to Z by sending data; Y is also related to Z by virtue of sending data. But these relationships do not represent the client/server model:

- Z has no client. The relationship between X and Z (and between Y and Z) is not one-to-one. Each event from X is delivered to thousands of listeners around the world. Furthermore, Z needs input from two event sources (X and Y) before it can act.
- Z undertakes work on its own initiative. Neither X nor Y has passed control of a business process to Z.
- Z is not performing a service for X or Y. X has not delegated any of its work to a subprocedure — it is blindly sending data without knowing what Z or any other recipient will do with it.
- X and Z are not even loosely coupled — they are logically uncoupled. If Z stops running, X and Y continue to operate correctly without any adjustment. Z never sends a response back to X. X and Y did not find, bind or invoke Z.

Figure 2. Event-Driven Model



Source: Gartner Research (February 2005)

Z is said to be event-driven, rather than client-driven. This illustration does not mention many aspects of event-driven systems, and not all event-driven applications look like this one. Simple event-driven modules (simple event handlers) are triggered by a single event. For example, when a person clicks a mouse button over an icon in a graphical user interface, an event is sent to trigger an event handler. However, this is still event-driven, not client/server, because there is no control coupling between a client module and the event handler (see "The Growing Role of Events in Enterprise Applications").

If we introduce a third module, a wrapper, we can make an event handler look service-oriented (Client J invokes Service K, which emits an event that causes Module L to execute, so the combination of K+L is a server to Client J). Similarly, we can use a wrapper to construct an event-driven service (Module M sends an event that causes Module N to fire and act as a client to invoke P). However, as long as we look at just the conceptual relationship and avoid indirect communication through a third module, events and services are mutually exclusive models (see Note 2).

Conclusion

Under the original definition of "SOA," where "SOA" is a synonym for classical client/server, an event-driven module is never a service. However, under all three of the modern interpretations of

SOA described in this research, an event handler and event source can be considered to be services, and their relationship would thus be an SOA. In our example, Event Handler Z gets data over a network from Modules X and Y, so modern Definition 1 (above) calls these three modules "services." If the communication from X to Z uses SOAP, Definition 2 would also consider X and Z to be "services." Finally, Module Z might demonstrate some or all of the six properties listed in our third approach to defining SOA, so, again, it could be a "service." Modern SOA is a superset of the original SOA because it allows event-driven as well as client/server relationships. However, it is sometimes a subset of the original SOA because it may require the use of SOAP (as in Definition 2) or a public interface metadata repository (as in some forms of Definition 3). The original SOA did not specify anything about the communication protocol or how interface metadata is handled.

Recommendations

The words "service" and "SOA" are overloaded, so any discussion that uses these terms should begin by specifying which definition is in use. Any definition can work as long as everyone uses the same one.

All new, large applications should be designed in a modular fashion to simplify construction, maintenance, incremental extension and reuse (see "Business Component Architecture Unites Services and Events"). In certain circumstances, the modules should relate to each other using the classical client/server model, but elsewhere the event-driven model will be better. Large applications will need a combination of both. These models are complementary and mutually reinforcing.

The client/server model is appropriate when modules must cooperate on one business function and there is a single, internally driven flow of control. The event-driven model is appropriate when the modules must share knowledge of a single event but will fulfill disparate business functions. Events enable multiple, asynchronous, parallel flows of control, and the flow may be changed by new stimuli from outside the processing environment.

Key Issues

Which software integration architectures and patterns will prevail, and which will fail to gain industry support?

Note 1

Client/Server

In almost all cases, client/server communication is a request followed by a response, although sometimes the response is only an acknowledgment that the request was received. In a degenerate case, there may be no response at all, but the relationship still demonstrates the client/server model as long as the connection is one-to-one and the client specifies the action to be taken. In another special case, the client and server may exchange a series of messages rather than just one message in each direction. This requires that the server retain some state (remember something about that transaction) for the duration of the conversation. Stateful servers should be avoided wherever practical because they are harder to reuse, load balance and failover than a stateless server. Some people consider stateful servers with conversational message exchanges to be outside the bounds of the client/server model. However, we include them as a special case — again, as long as the relationship between the modules is one-to-one and the client controls the action taken by the server.

Note 2

"Event-Driven SOA"

An alternative interpretation of the term "event-driven SOA" mixes the conceptual and physical levels of design. At the conceptual level, two modules are related through the familiar client/server SOA model. The client module cannot complete its work until it receives a reply from a server module. At the physical level, however, the request and reply messages are transmitted using a pair of one-way ("event") messages. This has some important advantages in flexibility and "pluggability" because modules can be replaced, moved or replicated at runtime relatively easily. This arrangement would not be "event-driven" in the sense used in this research because the client is dependent on a reply from the server and control of the server is conceptually coupled to the client.

Acronym Key

SOA	service-oriented architecture
SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language

REGIONAL HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
U.S.A.
+1 203 964 0069

European Headquarters
Tamesis
The Glanty
Egham
Surrey, TW20 9AW
UNITED KINGDOM
+44 1784 431611

Asia/Pacific Headquarters
Level 7, 40 Miller Street
North Sydney
New South Wales 2060
AUSTRALIA
+61 2 9459 4600

Latin America Headquarters
Av. Nacões Unidas 12.551
9 andar — WTC
04578-903 Sao Paulo SP
BRAZIL
+55 11 3443 1509

Gartner