"Service Oriented" Architectures, Part 2

Published: 12 April 1996

Analyst(s): W. Roy Schulte

To extract maximum benefit from service-oriented architectures, organizations must understand the design techniques and limitations associated with this approach.

Analysis

A service-oriented architecture is a way of using multiple software tiers to share code and data. Organizations considering this approach should be aware of the following issues:

- Service-oriented architectures use familiar forms of middleware and are not new. The essential notion is to separate application program functions into requesting modules and service modules. Developers see a catalog of available services. Each runtime instance of a service module can be accessed by multiple different requesters. Services can invoke other services. This arrangement has been described in many ways, including software "C/S" (C/S with no hardware implications). The software in both a desktop computer and a server computer can be organized in a service-oriented fashion.

- Most systems implemented with a service-oriented approach today are confined to online applications (i.e., batch processes are outside their scope). However, this is changing with the growing use of message queuing, stored procedures and other middleware that can readily accommodate all modes of work.

- Services may be separated into two tiers, one for business rules and the other for data I/O logic (e.g., SQL statements). This further degree of modularity potentially enables sharing, reuse or replacement of the modules on these tiers. This technique is a separate decision; it not fundamental to the basic service-oriented architecture concept. Not all services need involve data I/O - some services may be strictly computational.

- The thinking behind the service-oriented approach vaguely resembles that of OO systems in the sense that both hide the data from the requesting program and both automatically share data among requesters. But services usually are not implemented with OO technology (although they can be) and usually do not fully encapsulate the data. Services usually do not have exclusive control of enterprise databases; they coexist with other applications that directly access the data without going through the services. However, there are benefits, such as increased integrity and manageability, to not allowing updates to occur outside the shared services. This issue depends on the organization’s design decisions and on the type of middleware that is used. For example, database triggers and OO environments may protect the data from...
uncontrolled updates by the inherent structure of the DBMS and runtime software environment. However, most organizations bypass the services for queries, batch jobs, extracts and other tasks.

Many application functions that will be needed by multiple applications and multiple delivery modes (e.g., C/S, batch, mobile, online, Internet and others) can be deployed in a service-oriented topology. However, it is not a universal solution, capable of replacing all traditional batch, C/S and other topologies. Service-oriented topologies:

- Are not attractive for casual application development because they require some of the inevitable disciplines of reuse, e.g., design standards, quality assurance, formal administration, incentives to encourage reuse and maintenance of a documented inventory of available services (see Note 1).
- Will not replace two-tier C/S applications that are built with desktop-based tools. Legacy PC 4GLs are widely entrenched, and end users often demand control of some of their software development. Small applications whose scope is limited to one person, workgroup or department may be developed faster and easier without the rigor of services.
- Provide no clear benefit for service functions unique to a single application, although it is difficult to predict that any function will not become a candidate for sharing in the future.
- Will probably not be efficient enough to replace all traditional batch applications where the task demands locking, sorting or sequential passes of many records.

Bottom Line: Functions that will be needed by multiple applications and multiple delivery modes can be implemented with server-centric, multitier logical configurations that separate back-end service functions from front-end, delivery-related functions such as presentation and flow control. However, developers must apply service-oriented concepts selectively because they are not appropriate for all applications.

Core Topic
Software Infrastructure: Data and Process Topology

Key Issue
How should data and process be centralized or distributed around the enterprise?

Strategic Planning Assumption
Service-oriented topologies will account for more than one-third of new, mission-critical operational applications by 2001, up from less than 10 percent in 1996 (0.7 probability).
Related Research

Note 1

Techniques for Code Reuse and Sharing

- Service-oriented topologies share logic and data directly by enabling disparate applications to invoke the same instance of executable code. The code may be used by multiple users concurrently (if it is re-entrant) or it may only be serially reusable. However, in either case, it is the same identical code applied to multiple logically overlapping but different applications. Many operating system and other system software services are routinely executed this way, but less than 10 percent of new, user-written mission-critical applications use this technique. Virtually no casual applications use this approach.

- A single instance of executable code can be concurrently shared or serially reused by multiple users of the same application through features such as shared memory, supported by operating systems, language compilers, TP monitors and other runtime environments. This ubiquitous technique is an efficiency measure that applies only when users are executing the identical application. It is not a service-oriented architecture.

- Logic is also reused when source code is copied into another application, or a separate instance of executable code is bound into another application at development time. This leverages the value of the development effort across multiple different applications but does not attempt to share the same instance of executable code and thus is not a service-oriented architecture. This approach to reuse is underutilized despite its obvious merit.

Acronym Key

4GL Fourth-generation language
C/S Client/server
I/O Input/output
OO Object oriented
PC Personal computer
TP Transaction processing
GARTNER HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
USA
+1 203 964 0096

Regional Headquarters
AUSTRALIA
BRAZIL
JAPAN
UNITED KINGDOM

For a complete list of worldwide locations,
visit http://www.gartner.com/technology/about.jsp