# Service-Oriented Architecture vs. Service Chaos

BY JAMES MADISON

Creating a Service-Oriented Architecture (SOA) requires more than simply exposing a collection of services on the network. That's widely accepted, but what does it take to have an SOA and not just a chaotic collection of incoherent services? Here are 10 key factors:

**1. Explicit human-readable contract:** All services in a true SOA must have a specification document written in business analyst-level English that describes every operation, request parameter, and response value involved in every service. This ensures that people throughout the enterprise can understand the value of the service without having to be too technical or having to contact the service owners. Such a contract isn't that daunting. It takes only a few sentences to describe any given element, so even a complex service with a dozen operations, each with a dozen input and output values, can typically be described in a few hundred sentences. This contract doesn't include business data definitions (described later).

**2. Explicit system-readable contract:** The parts of the human-readable contract important for system processing can be converted to languages computers understand. Most companies use Web Service technology as their service implementation of choice. Web Services must always come with a Web Services Description Language (WSDL) contract outlining the service operation, and all data moved by the service must have an associated XML schema definition. XML schema is a language that defines how data should be structured. This lets computers ensure that any given body of data is structured properly. In particular, it lets computers programmatically enforce a fairly significant amount of the human-readable contract.

**3. Business data definitions:** Besides the operations, parameters, and return values that facilitate operation of the service itself, all the business data a service exposes or uses also must be defined. This is one of the most difficult yet important characteristics of an SOA to achieve. It's difficult because it requires that the organization actually have a formalized set of definitions for its most important data values—surprisingly few organizations do. It's important because it eliminates the need to rely on the expertise at the system of record, and it gives service callers confidence in the data provided. For example, in the insurance industry, premium is earned using several different algorithms, each of which can produce legitimately different results. If a service publishes "EarnedPremium" with no reference to algorithm used to earn the premium, the value of that field is, at best, reduced, and, at worst, causes the validity of the data to be called into question when it fails to

balance against another value named "EarnedPremium" exposed by another service.

**4. Enterprise functionality:** Individual systems are generally built with the needs of the individual business areas they support in mind. This generates systems focused on solving problems as seen by one small area of the company. Unless the value of using this functionality in multiple systems across the enterprise can be clearly defined, it shouldn't be turned into a service. If such enterprise value does exist, the service that exposes the functionality must also provide a set of input operations that allow the calling system to indicate what type of specialized behaviors it needs and a set of output indicators that allow the calling system to understand what the service actually did. For example, inputs might be whether a timeout should occur and whether partial results should be returned on time out, and an output might be the percentage complete for the partial results. Such inputs and outputs give the service everything it needs to be controlled by and communicate with the calling system and maximizes its ability to meet the slightly varying needs of systems across the enterprise. Naturally, these inputs and outputs would be precisely defined in the contract, as discussed.

**5. Implementation decoupling:** Defining a clear contract and creating functionality that's meaningful to the enterprise are part of the larger theme of making the service capable of delivering value in a manner that's well-defined and decoupled from any specific context. The well-defined part is primarily concerned with what callers see, but the context decoupling must propagate down through every layer of implementation of the service. Coupling includes such things as:

- Modifying data in a database that's also used by a system not related to the service
- Using the local system time to generate timestamps that are seen by callers in other time zones
- Using rules engines that are under control of an area other than those accountable for the service contract.

For example, insurance companies have been printing checks as part of claim processing for years, and printing checks requires only a few input fields, so it would seem a good candidate to turn into a service when another department determines they need check printing functionality. However, the check printing code was based on the assumption that checks would be printed only as part of a claim, so the code references the claim number in many different places. This coupling requires either generating dummy claim numbers and entries in several tables in the claim database, changing the otherwise stable code, or creating an entirely new check printing service implementation that's fully decoupled.

**6. Operations management:** Exposing system operations as services generally results in little or no information about how the service is then used, how well it's performing, or similar operational management characteristics. At best, the system that exposed the service will have some ability to track this type of information, but having individual systems in possession of this information causes it to be scattered around the enterprise, and getting to such information will be difficult at best. An enterprise that has an extreme propensity for service chaos may even consider exposing operational information as yet another service. By now, the alert reader knows that's not the design path to follow. Instead, an explicit service management architecture must be used to monitor the services, facilitate common operational control, and provide metrics and feedback on operational efficiency. This can be through a centralized platform, by deploying monitoring agents to the service platforms, or several other designs.

**7. Role-based security strategy:** Getting services on the wire is easy. It takes about two hours to set up the initial infrastructure on any developer's desktop, and two minutes to publish any existing object as a service. It then takes

any potential caller two minutes to generate the code to call it. Building out a full-scale production environment naturally requires much more work, but the service development itself is lightning fast. It's precisely this amazing level of ease that's causing the gold rush to the Wild West of service chaos. But putting services on the wire without a careful security design can be dangerous because services can generate information that has legal constraints or financial costs associated with it. For example, in the insurance industry, Motor Vehicle Reports (MVRs) are ordered on customers. MVRs return driver information that has state and federal privacy laws associated with it, and each report can cost up to $15. Just publishing an MVR service for anyone to call can get an organization into legal and financial troubles with minimal technical effort. Preventing this requires a security design based on the nature of the user or system calling the service, and keeping the security design under control requires basing it on a collection of roles that reflect the nature of the business processing.

**8. Traceability:** Services that are called can call other services, and so on, through more and more layers of depth until one service call from system A to system B results in service calls to C, D, E and possibly many others. This complex interdependency of systems can begin to couple systems to each other in a way so brittle that the organization cannot touch this frightening web of services for fear of breaking any number of other systems. Ironically, the theoretical loose coupling that's sold as one of SOA's primary values becomes completely absent in practice! Preventing this gets into some of the more esoteric discussions around SOA, such as hub-and-spoke design, the Enterprise Service Bus (ESB), mediation layers, etc. But as a quickly deliverable, easily managed minimum, all services should implement basic traceability by requiring that all calls include system identification in the call header. If a call is then made from system A that propagates to system E, this can be traced by the management platform.

**9. Orchestration enablement:** Once created, services will be used as groups to achieve higher levels of functionality than individual services can achieve alone. Bringing many services together in this way is called orchestration, and it often requires addressing issues that involve concerns beyond just making a call and getting a response. Services may need to ensure their work is done in an all-or-nothing manner with other services such as two banking services ensuring that a withdrawal from one and a deposit to the other occurs as a complete unit or not at all. Services

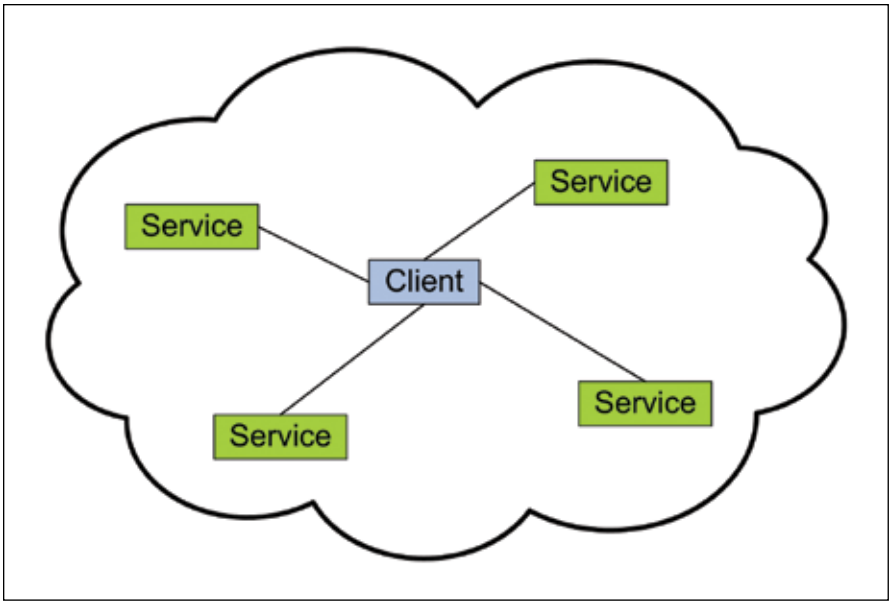


Figure 1: Services Exposed and Called With No Real Architecture

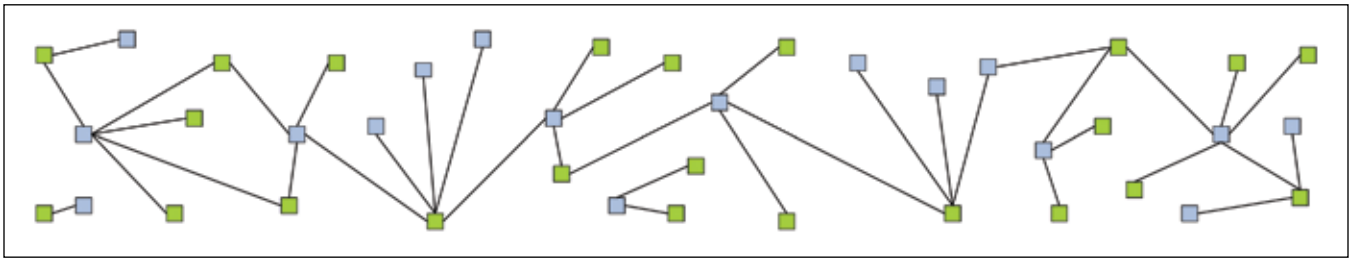

Figure 2: The Chaos That Evolves as More Clients and Services Are Arbitrarily Added

**business integration journal takeaways**

**BUSINESS**

- Require that each service have a contract that defines its operations and data in meaningful business terms.
- Watch for opportunities to implement services that are of an enterprise nature and that can be orchestrated into larger, more meaningful business flows.
- Plan for the extra time needed to produce functionality as a service vs. building the same functionality in a siloed application.

**TECHNOLOGY**

- Write the Web Services Description Language (WSDL) first and then build the system around it; this repeatedly pays off, since many tools can automatically generate functionality just by consuming good WSDL.
- Create a centralized platform that supports functionality that shouldn't or can't be left entirely to individual services such as security, traceability, and operations monitoring.

may need to communicate in a manner that guarantees messages will eventually be delivered even if the receiving service is currently down such as when automatically ordering supplies from a vendor whose systems aren't always available. Services may benefit from interacting with human-driven business processes that take hours or days to complete such as getting a human underwriter review for an insurance quote when the automated rules trigger a red flag. Essentially, the underwriter can be made to look like a service, but his latency has technical implications that must be effectively addressed. These and other orchestration functionalities can become somewhat complex and may require specialized software, but they offer value well beyond that provided by the paradigm of the basic request/response exchange with a single service.

**10. Formalized communication:** Services do no good if nobody knows about them. Every element just discussed must be packaged into a coherent body of communication and training and presented to anyone in the organization who could benefit from the services. If done right, that's a long list of people. Such a body of communication won't be one-size-fits-all. The level of detail and complexity of the communication will likely require customization to at least four levels: technical/implementation, business analyst, customer, and management. For example, geographic operations (e.g., the distance of a house from a flood area, the count of sales outlets within 15 miles, etc.) require a set of services whose functionality is fairly technical. The services make sense to the developers who build against these services, but they make little sense to customers because the services are too technical. When the operations of those services are abstracted to a more conceptual level, the descriptions remain accurate, yet make much more sense to a customer. Thus, the content of a presentation on geographic services will vary quite widely, depending on whether the audience is technically or business-focused.

Figure 1 shows a single client calling several services. There may be some type of contracted behavior, truly reusable functionality, smooth orchestration, meaningful operational management, etc.—or maybe not. There's really no way to tell. Figure 2 shows what happens when this lack of architecture grows. The chaos becomes explosive. Figure 3 shows an entirely different approach. With the same client and same services, several key elements of enterprise value have been added so it's clear to the client, services, and organization precisely what's happening. Figure 4 shows that growing this environment to full enterprise scale is manageable with effective SOA design.

This list of 10 key points is by no means exhaustive, nor is it unique. Differing needs of organizations and differing styles among the architectural teams building the SOA will produce different specific architectural considerations. What will be consistent, however, is the need to plan for and build out elements of an SOA that are focused on doing far more than just exposing services. All this activity will require some initial effort, but it will require far less effort than trying to control service chaos. **bij**

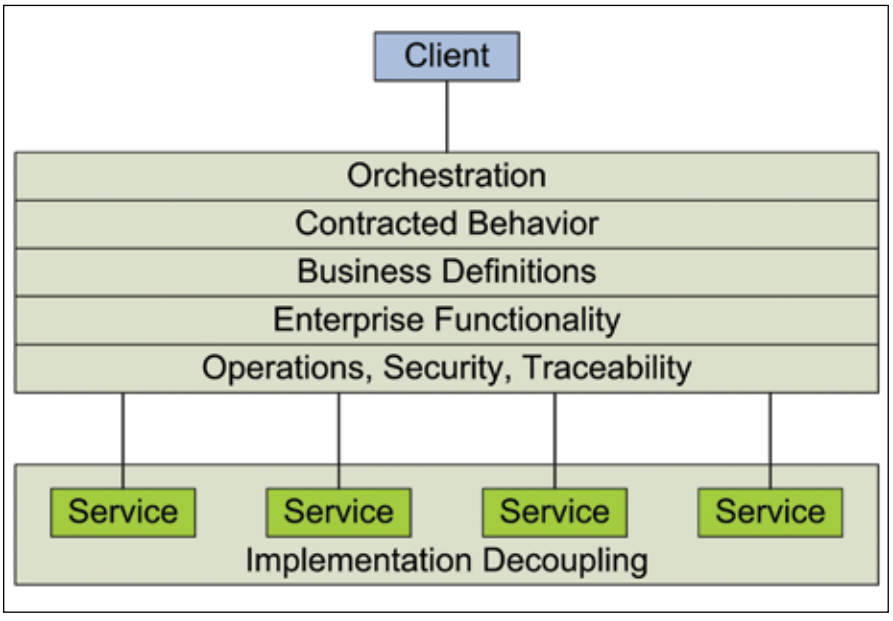


Figure 3: Services in a Well-Designed Architecture

### About the Author

James Madison is an information architect who designs data services at a financial services company.
e-mail: madjim@bigfoot.com

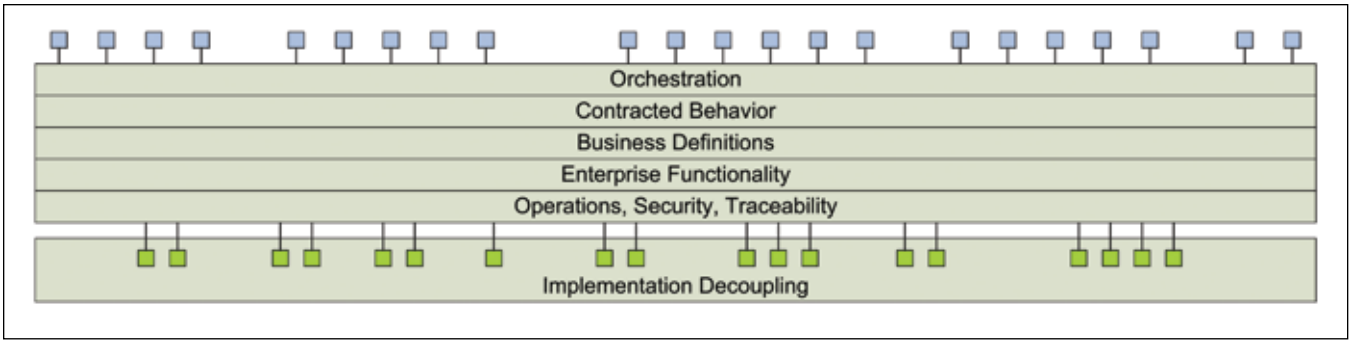


Figure 4: Manageable Growth of Enterprise Functionality as a Result of Implementing SOA