SOA antipatterns

When the SOA paradigm breaks

Skill Level: Introductory

Gary Farrow (gary.farrow@uk.ibm.com)
Technical Solution Architect
IBM

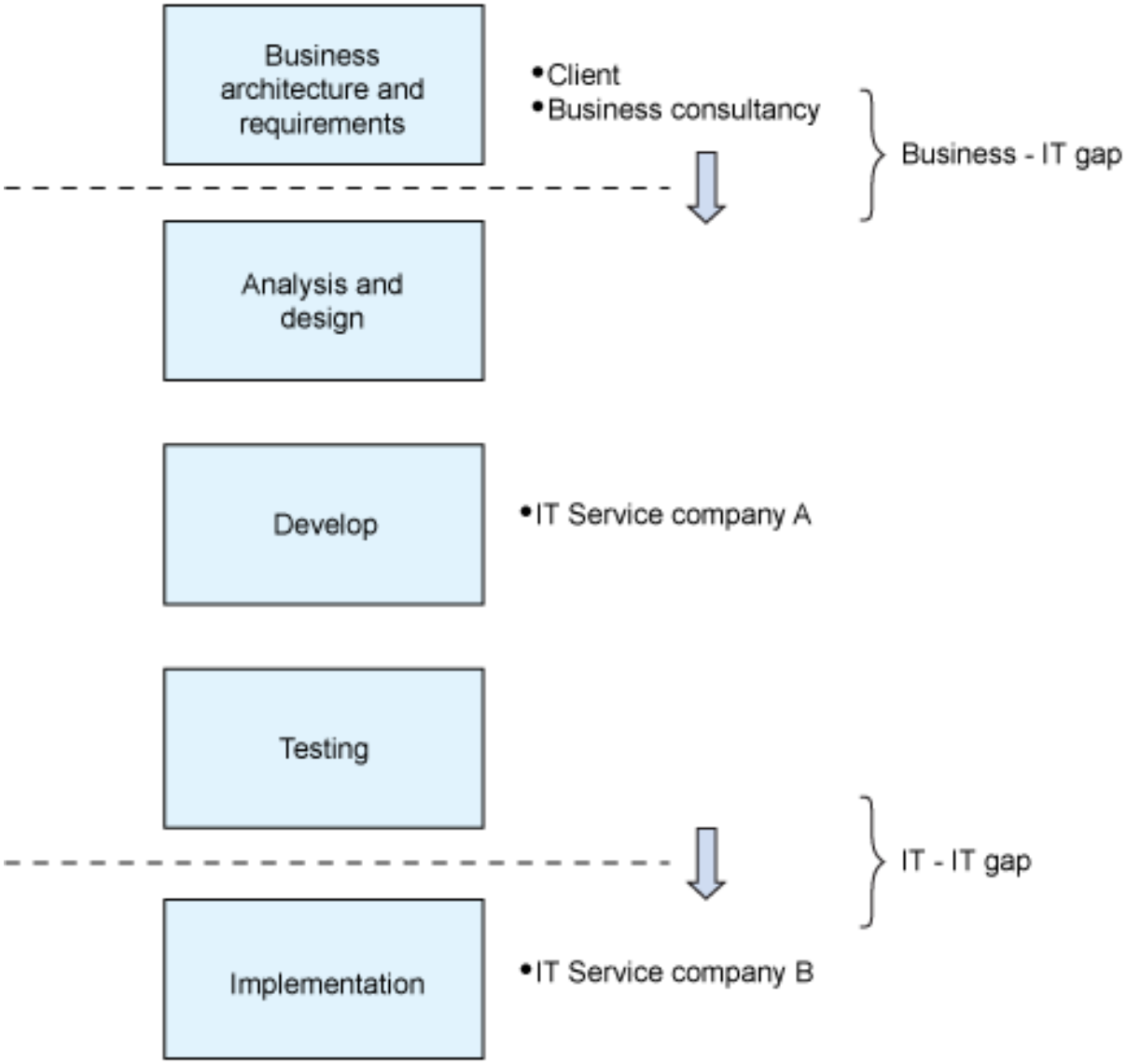
09 Jun 2009

Service-Oriented Architecture (SOA) is the de-facto architectural approach for many IT initiatives. It is therefore important to understand the circumstances where this paradigm breaks, as this can significantly impact the delivery of IT programs. This article highlights two SOA antipatterns that define problems that can occur in the execution of SOA deliveries. A simple frame of reference for SOA is first introduced in the form of a layered reference architecture. The reference architecture is then used to illustrate the underlying reasons for the occurrence of the antipatterns. For each antipattern a description is provided that highlights the root causes of the problem and the approaches to re-factoring the solution, hence facilitating successful delivery.

Introduction

Traditional delivery approaches are based on phases of the system development life cycle with different organizations fulfilling a different part of the life cycle. Furthermore, in such approaches, an emphasis is placed on one supplier delivering a complete system or sub-system. A simplified view of the delivery life cycle and a typical allocation of organization responsibilities is shown in Figure 1 below.

Figure 1. Traditional delivery life cycle and allocation of organization responsibilities



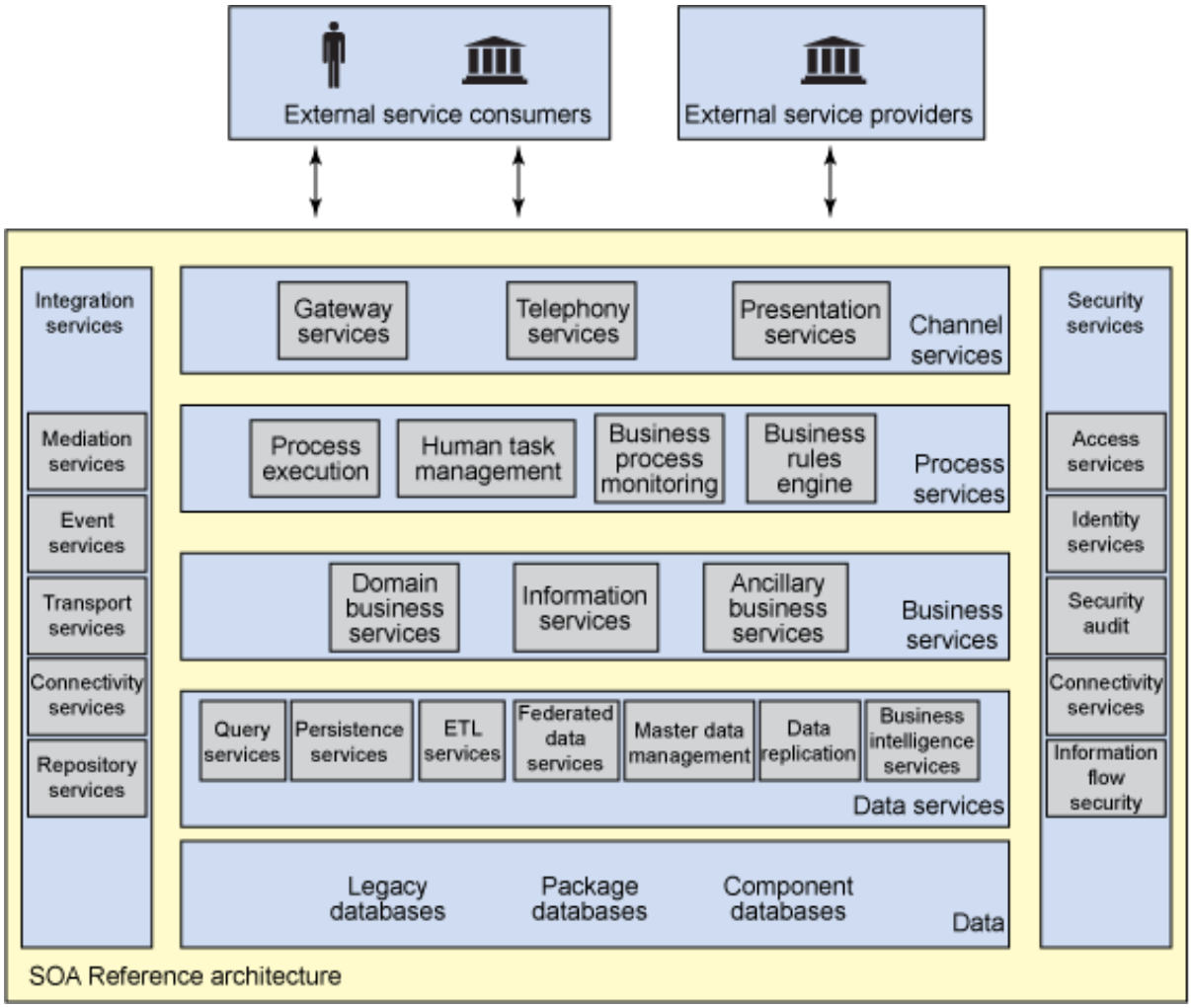
The advent of SOA infers a layered architectural model . This provides the opportunity for varied delivery approaches, whereby different parties deliver specific elements of the services layers. Experiences within such collaborative SOA engagements have identified problem scenarios that may arise. These are described in this article in the form of antipatterns.

In the first antipattern—Interface Bloat—an over-emphasis is placed on generalizing the data structures passing to and from a specific architectural layer. The result of this is a "bloating" of the interface whereby more data than is necessary is passed between layers.

In the second antipattern—Architecture Redundancy—the responsibilities of the different architectural layers are ignored. The result of this is that some layers add no architectural value and merely act as a pass-through for data.

Reference architecture definition

Figure 2. SOA reference architecture



An SOA reference architecture is shown in Figure 2. This represents a conceptual-level architecture for enterprise, service-oriented solutions. It provides a framework for architecture services within an organization. The basic premise of the reference architecture is that it promotes the architecture itself as a set of underlying services. Each of these services can be considered an Architectural Building Block (ABB). The concept of Architectural Building Block aligns to the definition provided in TOGAF 8.1 (The Open Group Architecture Framework; see Resources) for use when constructing early solution definition and scoping.

The purpose of the SOA reference architecture is to:

- Provide a logical grouping of related architecture services

- Achieve a clear separation of concerns between architectural building block responsibilities
- Provide a comprehensive taxonomy of the set of architecture services

The reference architecture can be used in many different ways. Some examples are:

- To clarify architecture principles and illustrate the architectural impact of the chosen principles
- To illustrate enterprise architecture solution patterns for given business scenarios, defined using collaborating architectural building blocks
- As a tool for enterprise architecture roadmap planning by illustrating which architectural services are required at a given point in time to support a business program
- To illustrate mappings of architecture services or building blocks to specific technology implementations
- To illustrate organizational boundaries of responsibilities in a multi-supplier delivery environment where different suppliers are responsible for different parts of the end-to-end delivery.

In this article, the reference architecture is used to illustrate how work was apportioned between IT delivery partners and also to facilitate the explanation of the underlying failings arising from the antipatterns.

Antipattern 1. Interface bloat

This antipattern (also known as *Data Tsunami*) relates to an over-specification of the interface between the Presentation Services layer and the Process Services layer.

Specification

Name	Interface Bloat
Also known as	Data Tsunami
Most frequent scale	System, Enterprise
Re-factored solution name	Process
Root causes	Mistrust, time pressure, inexperience
Unbalanced forces	Management of functionality. Management of performance
Anecdotal evidence	Unclear requirements or specification

Description

The context for this antipattern was a collaborative IT delivery where both a client organization and a system integrator delivery partner (in this case IBM) were responsible for delivering components of a full solution. This is an increasingly common situation, especially for SOA initiatives.

The IT partner was responsible for providing the Web portal, whereas the client organization had responsibility for producing business components and supporting data services. Thus, in terms of the reference architecture shown in Figure 2, organization delivery responsibilities were split by layer as follows:

- The IT partner delivering services within the Channel Services architecture layer
- The client delivering services within the Process, Business, and Data Services Layer

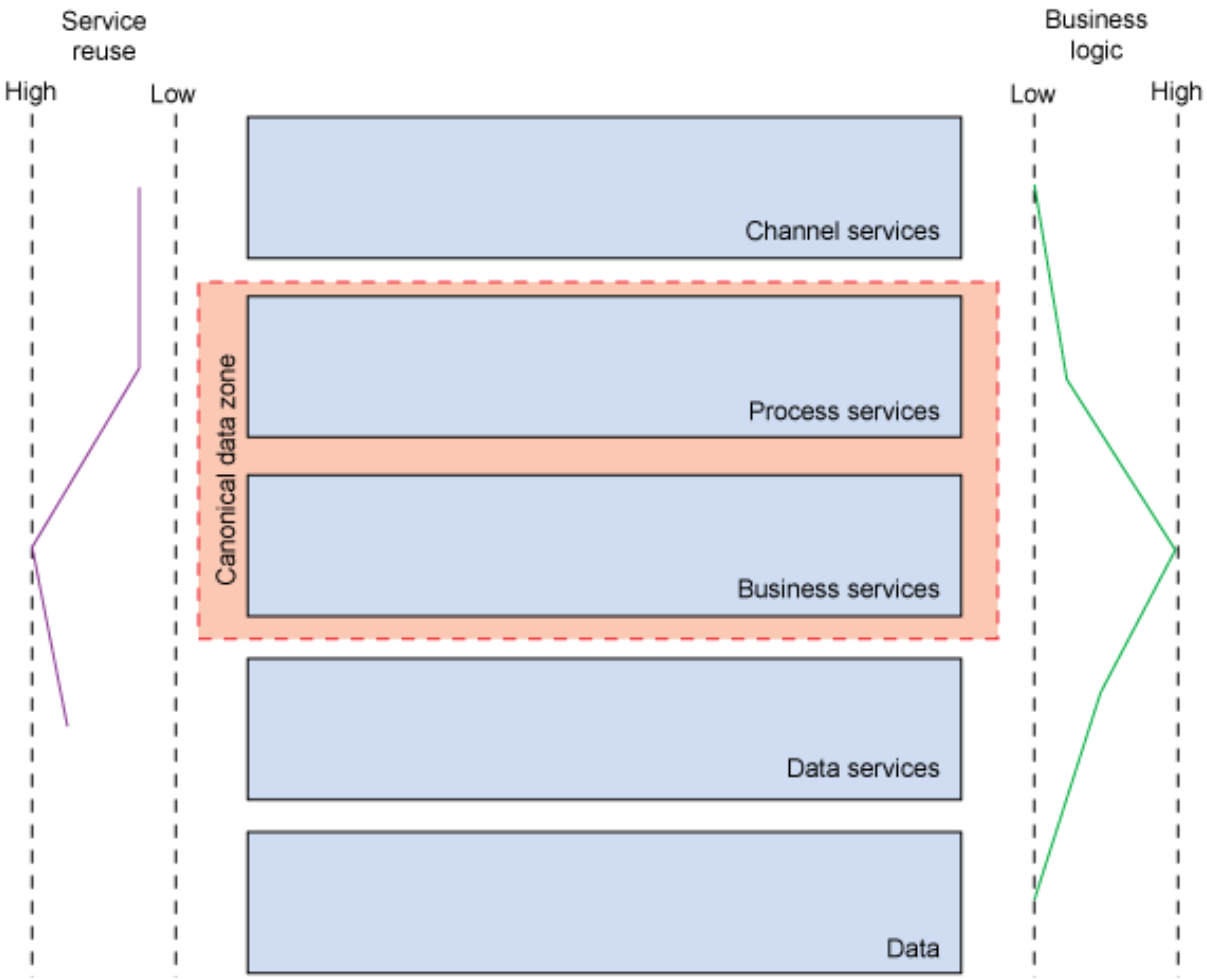
This split in responsibilities across the reference architecture was the root cause of the observed antipattern. Specifically, the client organization was not sufficiently mature in its design process to effectively support an SOA delivery, as was apparent by the following:

- The concept of a formal macro design process was relatively new and hence unproven within the organization.
- The production of formal specification of interfaces between components in advance of coding had not previously been done.
- Being unable to achieve a specific focused specification, a looser, over-specified interface was adopted.

Furthermore, to fulfill one of the key perceived benefits of SOA, there was a desire to build services that could be reused. There resulted an over-emphasis of the need to build reusable services within the Process Services layer of the reference architecture. The consequence of this is that interfaces between the Portal application (in the Channel Services layer) and components in the Process Services layer became too generalized.

Attempting to generalize connections between these layers is considered sub-optimal. Specific Process Services supporting a user interface are unlikely to be able to be generalized to a B2B channel where the dynamics of the interactions are fundamentally different. A better approach is in fact to provide specialized Process Services to support the specific needs of the channel, in this case a Web-based portal. The preferred emphasis on reuse throughout the reference architecture layers is shown in Figure 3.

Figure 3. Mapping of reuse and business logic



A final factor in the emergence of the antipattern was time pressure on delivery. This was such that proper analysis was not undertaken in advance of the detailed technical design and coding.

.

The net consequence of these conditions was that too much data was passed between the Presentation Service components and the Process Services components. The effect of this was:

- Additional query and data transfer time was required.
- Intensive processing was required to parse data in the Presentation Services layer and extract the sub-set of data needed for a specific operation.
- Performance issues arose through slower response time due to marshaling and passing the data.

Antipattern 2. Reference architecture redundancy

The second antipattern (also known as *Pass the Parcel*) relates to situations where the same transfer objects and methods are specified in the designs of the components in each architectural layer. The impact of this is that the bulk of the processing logic is pushed into a single architectural layer, making some layers almost entirely redundant and a vehicle only for the "pass through" of data.

Specification

Name	Reference Architecture Redundancy
Also known as	Pass the Parcel
Most frequent scale	Enterprise
Re-factored solution name	Role
Root causes	Time pressure, paradigm misuse
Unbalanced forces	Management of functionality Management of performance
Anecdotal evidence	Lack of a reference architecture Lack of architecture usage principles Service method name appears in all software layers

Description

The context for the occurrence of this antipattern is a circumstance where the SOA paradigm is relatively new for the client organization. This is a common occurrence as many organizations are only now embarking on their first SOA initiative.

In such circumstances an SOA reference architecture may have been explicitly defined or may be implied. However, the in-depth understanding of the reference architecture and its practical application is not yet present within the client organization.

The consequences of this are an incorrect understanding of the responsibilities of each layer in the reference architecture. This in turn manifests itself in:

- The same method signatures appear on different services residing within each of the reference architecture layers.
- An over-generalization of the transfer objects passing between layers
- Business logic being pushed into the Data Services components instead of residing within components within the Business Services layer

The consequence of this is that:

- Certain reference architecture layers are redundant in that they perform no added value and merely "pass-through" data to the next immediate layer in the architecture.
- Reuse of Business Services is not achieved.
- The advantages of the SOA paradigm are negated.

The re-factored solution is one which allows flexibility in the way the reference architecture is applied. The extent of the flexibility can be captured in terms of a set of architecture principles that should accompany any reference architecture. For example, data service operations that merely retrieve or update data, without the application of business logic should be permitted to be called directly from the Channel Services layer. There is no need to go through the Business Services layer in these circumstances. This principle should be embodied as part of the reference architecture definition.

Commentary

A fundamental misconception resulting in both of these antipatterns is a desire to apply a common data format within the Channel Services Layer. This is not typically a sensible design goal. Connections between the Channel Services layer should utilize a data format that is specific to the channel.

An optimal approach is therefore to focus on enforcing the use of a "canonical" data form within the Process and Business Services only. For example, if the channel comprises a Web application that provides views of business data for update, then the transfer of data should be optimized to return and update only that data provided in the view. Similarly, if a channel is a B2B gateway that uses a specific industry data standard, then middleware should be used to transform the channel-specific data format into the chosen canonical data form used within the Process and Business Services layer. This preferred "canonical data zone" is also shown in Figure 3.

A re-factored solution should permit a variety of different service calls from the Channels Services layer, rather than attempting to generalize them. This in turn promotes reuse of services at the Business Services layer rather than the Process Services layer. This approach also better serves the principle of composability for service-oriented architectures, whereby new, coarser grained services can readily be composed from existing finer grained services.

Summary

This article has presented two antipatterns that may arise during the delivery of SOA

initiatives. The impact of these antipatterns can be significant, leading to delivery delays, poor performance, and lack of reuse. In severe circumstances, loss of confidence in the paradigm can ensue with subsequent cancellation of SOA initiatives.

The circumstances leading to the occurrence of these antipatterns, namely collaborative engagements between a client and an IT partner and a lack of practical experience of the paradigm, are common in many SOA engagements. Delivery organizations should therefore be aware of these antipatterns and plan for their prevention. During the actual delivery, they should be cognisant of emerging anecdotal evidence, as highlighted in this article, that will indicate the occurrence of these antipatterns.

Re-factoring and ensuring a clearly articulated set of architecture principles is key to the elimination of these antipatterns. Openness of design and a strong solution governance, transcending commercial boundaries, are also required to prevent these antipatterns from occurring.

Resources

- For more on antipatterns, read the book, Anti-Patterns: Re-factoring Software, Architectures and Projects in crisis, Brown et al, Wiley and sons, New York, 1998.
- Learn more about The Open Group Architecture Framework (TOGAF) .
- Personalize your developerWorks experience with My developerWorks.

About the author

Gary Farrow
Gary is a Technical Solution Architect in the Application Innovation Services UK practice of IBM Global Business Services. His interests include large scale system integration, enterprise service bus architectures and SOA. He has delivered many solutions for the Financial, Government and Transport sectors has a specialist interest in payment processing systems. You can reach Gary at gary.farrow@uk.ibm.com.