

Applications At Your Service

Mahesh H. Dodani, IBM, U.S.A.

1 APPLICATION ARCHITECTURE

“Some things are always the same, particularly the business problems facing IT organizations. Corporate management always pushes for better IT utilization, greater ROI, integration of historically separate systems, and faster implementation of new systems; but some things are different now. Legacy systems must be reused rather than replaced, because with even more constrained budgets, replacement is cost-prohibitive. You find that cheap, ubiquitous access to the Internet has created the possibility of entirely new business models, which must at least be evaluated since the competition is already doing it. Systems must be developed where heterogeneity is fundamental to the environment, because they must accommodate an endless variety of hardware, operating systems, middleware, languages, and data stores. Within a business environment, a pure architectural definition of a SOA might be something like "an application architecture within which all functions are defined as independent services with well-defined invocable interfaces which can be called in defined sequences to form business processes.” – [Migrating to a Service Oriented Architecture](#)

Over my last three articles, I have laid a foundation for a Service Oriented Architecture (SOA) as the enterprise architecture of the [globally integrated enterprise](#) and focused on how to define and establish the business side of the enterprise through a well defined [business architecture](#). In this article, I continue our journey into the IT architecture side and start by focusing on the application architecture (in later articles I will cover the other aspects of the IT architecture including the information and infrastructure architectures.) To reiterate, as shown in Figure 1, the application architecture is a key component of aligning IT with business. In particular, the application architecture elaborates services required to implement the defined business model and process. The services required to implement the business model and process are defined as applications that can be realized using existing legacy, packaged and remote applications and services. The application architecture also elaborates the way that consumers of the business services interact with the process and applications – as defined by the user interface and interaction mechanisms (e.g. via portals, browsers, and mobile devices.)

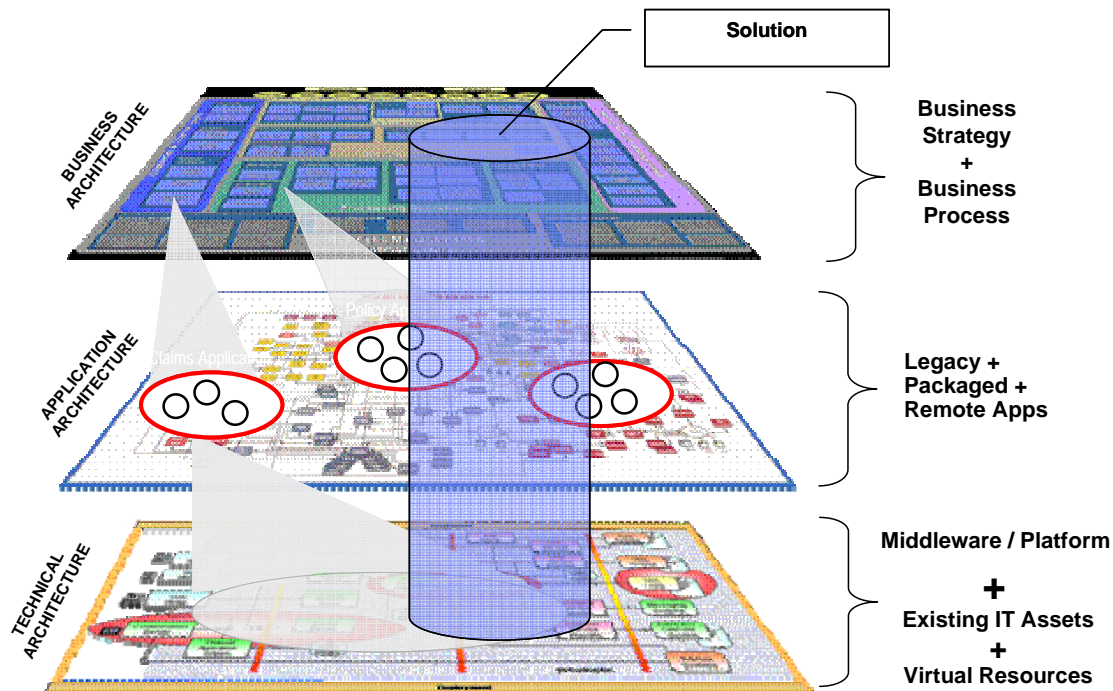


Figure 1: Aligning IT with Business

In the context of SOA, the application architecture focuses on reuse of existing assets, and is primarily concerned with the following:

- Analyzing business processes to discover services, including both identifying services required to perform the individual tasks defined by a given business process as well as analyzing existing applications to identify service providers.
- Creating service providers that support the defined business services and processes that an enterprise exposes to its consumers and users. These services can be created from existing assets, external service providers and new service providers.
- Connecting service consumers to business services and processes, and from these exposed services to service providers by enabling "any-to-any" linkage and communication between services inside and beyond the enterprise, and through infrastructure that ensures Qualities of Service (QoS) including security, reliability, and scalability.

In the following sections, we explore key patterns supporting application architecture in SOA, including application reuse patterns and the main connectivity architectural pattern – the Enterprise Service Bus.



2 APPLICATION INTEGRATION PATTERNS

Figure 2 shows the main application integration patterns established as part of IBM's [Patterns for e-business](#). The direct connection application pattern represents the simplest 1-to-1 interaction between applications. It allows a pair of applications within the organization to directly communicate with each other. Interactions between a source and a target application can be arbitrarily complex, and this complexity can be addressed by breaking down the interactions into more elementary components. More complex point to point connections will have modeled connection rules that control the mode of operation of a connector depending on external factors as shown in Figure 2. Examples include business data mapping rules, autonomic rules, security rules, capacity and availability rules. The indirect or broker application pattern, shown in the figure, separates distribution rules from the applications using a 1-to-N topology. It allows a single interaction from the source application to be distributed to multiple target applications concurrently thereby reducing the proliferation of point-to-point connections. The indirect or broker application pattern applies to solutions where the source application starts an interaction that is distributed to multiple target applications. It separates the application logic from the distribution logic based on broker rules which manages the decomposition/ recombination of the interaction. The broker pattern reuses the direct connection pattern to provide connectivity between the tiers.

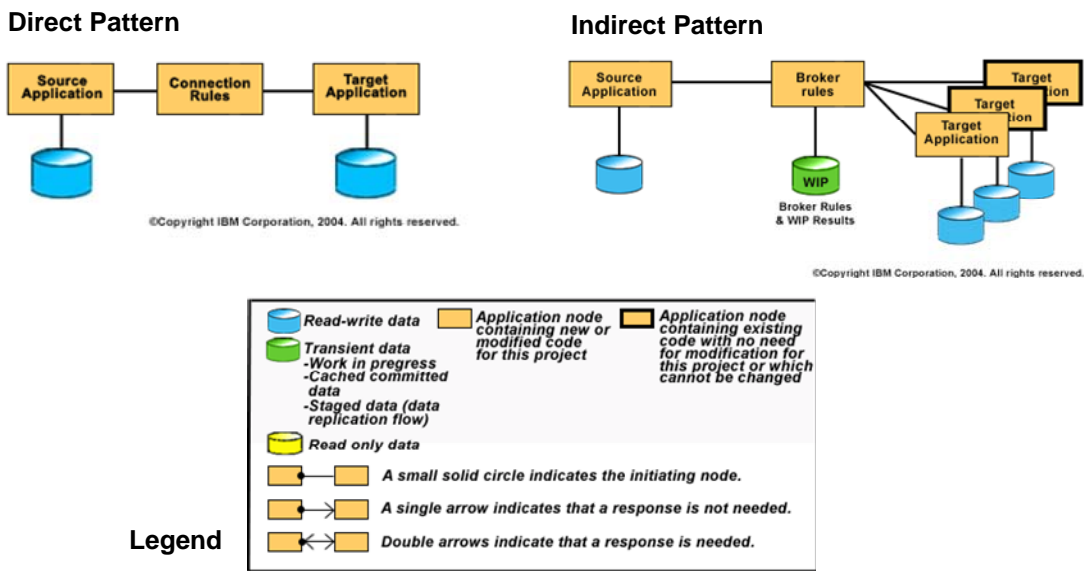
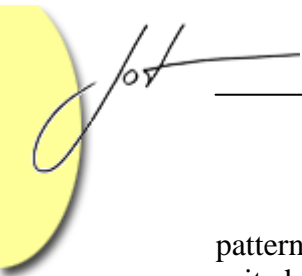


Figure 2: Application Integration Patterns

In the context of SOA, the direct pattern allows a defined service to be provided directly by an existing asset or external service provider since it is capable of being accessed via a service interface. The indirect pattern makes the existing asset or external provider a service through a broker or service component. The indirect pattern has many sub-



patterns based on the existing asset type. The adapter or gateway sub-pattern is best suited for external service providers, and provides a proxy-based access to the asset by mapping standards-based interface to the asset interface. The application server sub-pattern provides a generalized capability for interacting with multiple target assets, and provides an environment for augmenting an asset's capabilities. It is best suited for reusing packaged applications as service providers. The terminal "emulation" sub-pattern targets mainframe applications. It provides a mechanism to encapsulate a sequence of screen interactions as a "macro" and then to expose these "macros" as a service. These patterns have their own pros and cons – and these should be taken into consideration when making the architectural decisions for a given solution.

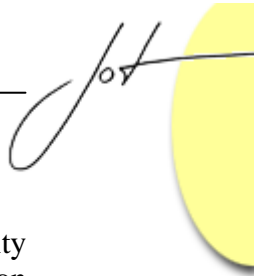
The direct pattern has a shorter deployment cycle especially compared to the indirect pattern. Since the service interface is defined by the asset, no further analysis is required to determine the interface. Furthermore no knowledge of other runtimes (Java, messaging middleware, etc.) is necessary when building the service. On the other hand, consumers become coupled to the asset environment, and therefore it becomes difficult to substitute an alternate asset to provide the service. The direct pattern also places a burden on the asset runtime environment, requiring it to have support for service invocation, be able to match the service requirements, and handle the additional XML processing burden.

The indirect pattern is more suited to support the IT alignment with business by ensuring that the service interface aligns with the business view rather than with existing assets. The service component is used to map between the two worlds, and this facilitates a substitution of the asset as the service provider without impacting the consumer. This service component can allow the business service to be implemented using behavior from more than one asset – the service component aggregates the behavior to realize the service, and can also enable additional capability. The main issues associated with the indirect pattern include longer deployment cycles and more complexity in developing the service component. Generally, the indirect pattern involves the use of connector/adaptor technology between the service component and the backend systems – and usually introduces a middle tier.

The above pros/cons of the patterns provides the basis for the architectural decisions. Choose the direct pattern when the existing asset platform is strategic, expediency is a key business driver, and the development team has skills only on the existing asset. Choose the indirect pattern when several assets need to be aggregated to provide the needed functionality and in some cases a subset of the functionality of an existing asset is required. This pattern is also useful to put a façade on an asset that needs to be replaced or modernized.

3 THE ENTERPRISE SERVICE BUS ARCHITECTURAL PATTERN

The Enterprise Service Bus (ESB), shown in Figure 3, is an architectural pattern that supports connectivity between service requestors and service providers, and forms the



backbone of any SOA realization. The ESB supports this requestor/provider connectivity by handling multiple communication protocols supporting pre-defined interaction patterns. The ESB enables these interactions through defined mediation flows to process request messages and correlated results using defined patterns. The ESB should be flexible in supporting multiple message content models usually based on meta-models.

In an SOA-based implementation, the ESB facilitates a “virtualization” of the services’ identity, protocol and interfaces. Through its capabilities to route service requests, the ESB can virtualize a services’ identity. It also facilitates conversion between different protocols and transformation between various interfaces. Furthermore, the ESB pattern also enables aspect-oriented connectivity to handle security, management, logging, auditing, etc.

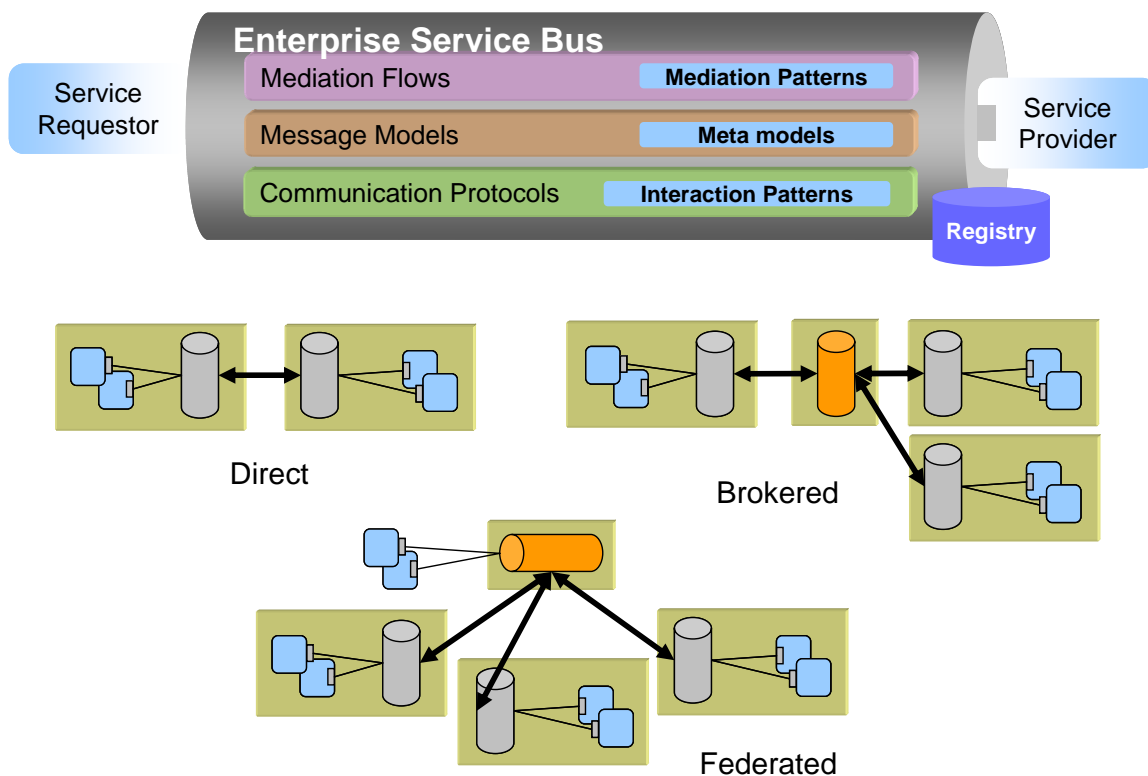


Figure 3: The Enterprise Service Bus Patterns

ESB mediation flows leverage several mediation patterns. Mediation patterns are well-defined types of processing the ESB uses to manipulate the messages during an exchange. Mediation patterns take many forms and many levels of granularity. We have already identified two important mediation patterns – the conversion processing needed to support virtualization of communication protocol and interaction pattern, and the transformation processing needed to support virtualization of interface.

A family of mediation patterns provide virtualization of identity via routing. Routing allows the ESB to send messages from a service requester to a service provider chosen dynamically, based on the conditions at the time of the request. The ESB can provide routing mediation patterns that range from very simple to very complex. The most useful routing mediation patterns are driven via service metadata derived from the service registry. In addition, different routing patterns can contribute to differing qualities of service, for example offering request retry and failover. An ESB may support more complex routing patterns such as distribution of a request and correlated aggregation of responses.

Other mediation patterns support additional message manipulation – examples include message enrichment and message filtering. Still other mediation patterns contribute directly to aspect-oriented connectivity, such as monitoring and logging and access to the services registry, the security and management policy definition points, and other parts of the solution infrastructure. Finally, more complex mediation patterns include capabilities such as complex event processing.

The key ESB patterns are also shown in Figure 3. In the direct pattern, typically we see peer domains connected, generally with little distinction between them, though sometimes interactions tend to remain within a domain, especially in the case of geographic distribution. In the brokered ESB pattern, we typically see peers connected, but where generally the majority of traffic is within a domain. In the federated pattern, the key difference is that the common interaction is within a domain, but a ‘backbone’ e.g. a retail headquarters supplies some services, and may offer peer to peer routing.

In summary, the application architecture is key to aligning IT to business which in turn is an integral part of ensuring successful SOA adoption in the enterprise. We have discussed the key patterns that can be used to make appropriate architectural decisions when designing solutions. In the next article, we continue our journey into the IT side with a focus on the Information Architecture.

About the author



Mahesh Dodani is a software architect at IBM. His primary interests are in enabling communities of practitioners to design and build complex business solutions. He can be reached at dodani@us.ibm.com.