BMC Software Inc.

Technical Disclosure Publication Document

Enterprise Service Bus (ESB) Insulation Service

Author

Vincent J. Kowalski

Posted: May 2011

Overview

This document describes the invention, the Enterprise Service Bus (ESB) Insulation Service, which provides a mechanism that decouples software applications from specific implementations of ESB technology. By providing this decoupling, the ESB Insulation Service allows application solutions to swap out ESB technology without the need to re-engineer the application logic. In addition, ESB components can be evolved or brought on-stream at arbitrary times as there is no dependence of the applications on the actual ESB implementation.

Background

This invention involves the use of a Universal Description Discovery and Integration (UDDI)-based web services registry for the dynamic publication and subsequent lookup of service endpoints. The use of such a registry represents a major step forward in the ability of applications to work as integrated solution suites rather than as collections of point products.

The use of the web services registry requires service providers to publish their interfaces and associated service implementations (or endpoints) to this registry and then service requestors (or clients) search this registry for services that they need to bind to. This use of the registry affords integrated solutions the following significant advantages over more traditional application programming interface (API)-based integrations:

- location independence
- version independence
- implementation independence

While these are significant advantages, there are improvements over this architecture that can be made. By providing encapsulated functionality in a bus architecture service-oriented architecture (SOA)-based solutions can be implemented in a simplified and seamless manner. This is the purpose and intent of an ESB.

Problem

An ESB is itself an abstraction layer on top of an implementation of an enterprise messaging system. This enables integration architects to exploit the value of messaging without writing code. Unlike the more classical enterprise application integration (EAI) approach of a monolithic stack in a hub and spoke architecture, an enterprise service bus builds on base functions broken up into their constituent parts, with distributed deployment where needed, working in harmony as necessary.

In real-world implementations, ESBs are implemented as a set of services and/or service proxies where there are two broad classes of such services:

- user-defined services, and
- system-provided services

User-defined services are those services that are plugged into the “back-end” of the service bus by the enterprise using the service bus to make such service available to potential clients. These would presumably implement some business logic that would be useful to a broad class of applications. Examples of these could be incident management and manipulation, event management, or systems property/parameter management services.

System-provided services are those services that are fundamental to any SOA application and have broad (perhaps universal) applicability. The list of these services generally includes the following:

- Invocation (SOAP message processing)
- Routing
- Messaging
- Mediation
- Registry
- Authentication, authorization & single sign-on
- Orchestration
- Administration and management

These services are generally provided by the ESB either as a proxy to some well-know service (for example, UDDI operations for a registry service) or as part of the interface that defines the functionality of the ESB (as is often the case with mediation services).
It is with these system-provided services that there is an inherent problem for users of ESBs. Applications being written to such interfaces provided by the ESB will, as a matter of necessity, need some or all of such services. By writing to specific interfaces of a given ESB, applications lock themselves into that specific ESB. Such lock-in is undesirable for a number of reasons, including but not restricted to:

- There may be changes in technology (e.g., going from proprietary ESB to open source ESB)
- There may be advances in technology—requiring a change in ESB
- There may be changes in business relationships (e.g., BMC may acquire or be acquired by an ESB vendor)
- Customers may require ESBs that they have already deployed in-house

For these and other reasons, lock-in to specific ESB is undesirable. It is this problem of ESB lock-in that we are addressing with this ESB Insulation Service solution.

Solution

The solution involves abstracting the most widely-used and fundamental services that are generic in an environment that makes use of an ESB. By providing this abstraction itself as a service, applications can write to the insulation service (or its sub-services) and not become dependent on any vendor's ESB. The insulation service provides operations for most of the services, namely:

- Invocation (SOAP message processing)
- Routing
- Messaging
- Mediation
- Registry
- Authentication, authorization & SSO
- Orchestration
- Administration and management

These sub-services will be detailed in the sub-sections that follow. By providing such services as a thin layer over services already provided by an ESB, the insulation service indeed insulates applications from dependencies on any specific ESB vendor or implementation. BMC can then swap out ESB implementations and, assuming there will be an implementation of these services in the new ESB, do so without perturbing or breaking existing applications that make use of the ESB.

In addition to the advantage of avoiding ESB lock-in, the insulation service also provides that advantage of presenting a simpler, more convenient and coherent set of services. In real-world implementations, system services of ESBs can be bit low-level or otherwise difficult for application developers to incorporate into their applications. By providing this insulation service, we can provide a level of API that would enhance the convenience of using the ESB, thus improving the productivity of our developers. Note: some of the functions in the insulation service may be implicit and not directly invoked by any client, but instead done on behalf of the client by the ESB.

In the subsections that follow, the sub-services of the insulation service are provided. These are first characterized and then operations are enumerated and described. This is done herein at a high level of abstraction. Detailed interface signatures will be provided in a subsequent software design document on this subject.

Invocation (SOAP Message Processing)

Invocation services are essential to any ESB. This service enables clients of services to make use of service providers in their own respective business logic. The ESB will receive requests for service and provide either a proxy to the service provider or delegate the request to a registered service provider as it deems necessary. The functionality of the invocation service may or may not be explicitly exposed to clients of the ESB. Such functionality includes, but is not restricted to:

- Service binding (use of a service provider by a client)
- Service delegation (done on behalf of the client)
- Service mapping (translation of a request into an actual invocation)

Routing

The routing service ensures that service invocations are sent to the appropriate providers. Such routing will often be done on behalf of a service client. However, how that routing is done will be based on a number of factors such as the content of the message, the policies in place regarding message routing, the access rights of the client, quality of service and so on. As such, the list of operations made available by the routing service will include, but not be restricted to:

- Routing policy definition
- Routing determination parameters/criteria
- Routing rules definition
- Routing query (find what policies, rules, etc. are in place)
- Routing execution

Messaging

The messaging service of the ESB is related to how invocations of services and data transfer occur across the ESB and by participants in the ESB. As such, the messaging service may include the following operations:

- Message passing
- Message adaption
- Message enhancement

Mediation

The mediation service is a defining service of most (if not all) ESBs. In its most basic form, the mediation service provides a mapping of service request to service providers. Having such a mapping in place allows for policies, rules, transformations, and translations to take place between the request and the actual servicing of that request (hence the name: mediation). The mediation sService will include, but not necessarily be restricted to the following operations:

- Lookup service
- Translate service
- Transform service
- Define policy
- Query policy
- Execute policy

Registry

All user-defined services will need to be registered with the ESB. This will allow easy look-up and location of services based on some search criteria. In many cases, the ESB will hide the workings of the registry and do the registry operations on behalf of the client. Indeed, this simplification of registry interaction is one compelling value of the ESB. However, there will be times (for example, when a service is first made available through the ESB) when such registry operations will need to be exposed to users of the ESB. Registry operations include, but are not confined to:

- Service publication
- Service query

- Notification (e.g., when a service becomes registered or de-registered notify these listeners)

Authentication, Authorization & SSO

As the ESB is a critical resource, it must be accessed in a way that does not compromise the security of the enterprise in which it is deployed. As such, the ESB must have means for service providers needing to register their services as well as clients of those services to authenticate to the ESB. In addition, it would be a burdensome task for a client to be expected to know and provide authentication and authorization credentials for each and every different service it needs to interact with. It would be far better for a client to simply authenticate once with the ESB and then have the ESB handle authentications on behalf of the client. This implies that some SSO capability is required on top of basic authentication and authorization functions. Given all this, the operations such authentication, authorization, and SSO would include, but are not restricted to:

- Login
- Logout
- Create user
- Revoke user
- Remove user
- Query users
- Authorize user (for a given resource)
- Un-authorize user

Orchestration

Orchestration enables the arrangement of service invocations to be done according to some well-defined workflow or business process. Developers of such workflow-based applications can think in terms of the business processes they are trying to support rather than the underlying “plumbing” that is required to make such components talk to one another.

- Workflow identification or naming
- Workflow definition
- Workflow execution

Administration and Management

The administration and management sub-service allows for the ESB to be administered and monitored by some external actor, presumably some system administrator. As such, it would include the following functions:

- Query/retrieval of system parameters (heartbeat, resource utilization, etc.)
- Administration commands (startup, shutdown, backup, restore, etc.)
- Administration browsing/console (what services deployed, who are the clients of those services, etc.)

Layered Access to Enterprise Service Bus Services

The ESB architecture is intended to be an incremental, evolutionary advance in a SOA vision, not a rip-and-replace approach. As such, the web services stack and registry that are currently in use by integrated solutions will be in place and accessible in the new bus architecture. In fact, technology based on any step in the evolution of SOA will be to work with technology based on any other step in the evolution as depicted in Figure 1.

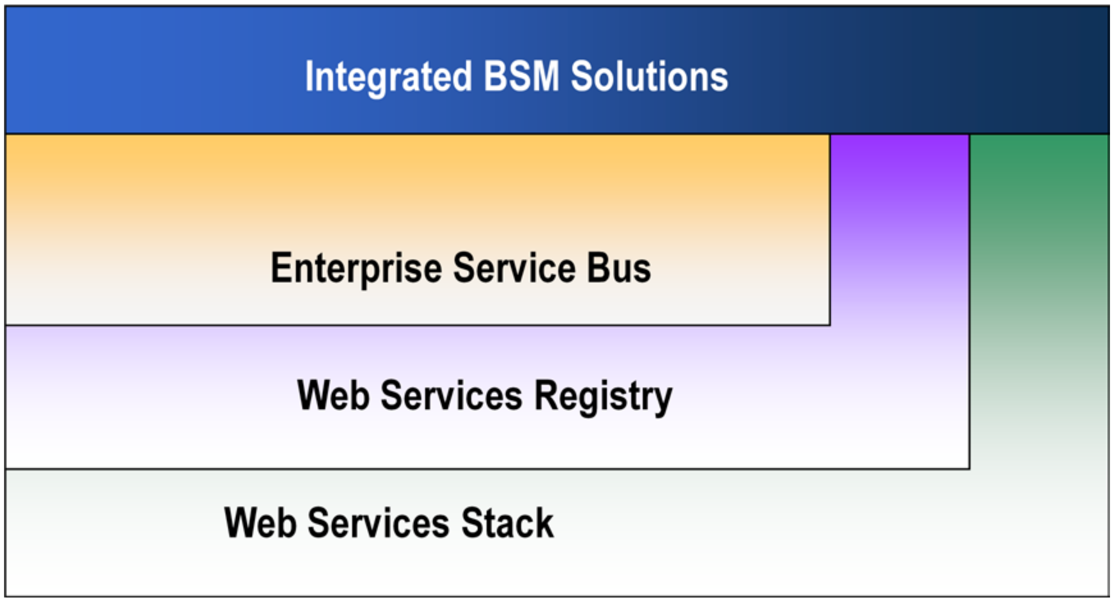

Figure 1: Layered Access to Enterprise Service Bus Services

In Figure 1 above each of the layers (from bottom up) corresponds to the steps in our SOA evolution. In point-to-point integrations, we only have the Web Services stack available to us. In registry-based integrations, the stack and registry are available. And finally, in ESB-based integrations we have the additional mediation and orchestration services available. Note: SSO may be introduced by registry-base integrations, but it is a required part of ESB-based integrations. Note: this ability to involve a bus architecture into an existing web services integration scenario without disturbing the existing integrations or forcing new integrations to comply with the current architecture is a compelling characteristic of this architecture.

Note that although access to services at all levels is allowed, applications may take advantage of the special insulation device provided at the ESB level and take advantage of not needing to know specific service signatures or embed such syntax and semantics into the business logic of their applications. Application developers are encouraged to make use of the insulation service because this will allow technology to be swapped out without breaking existing applications.