BMC Software Inc.

Technical Disclosure Publication Document
Application Integration Manager (AIM)

Author

Vincent J. Kowalski

Posted: June 2009

Overview

This document describes an invention, the Application Integration Manager (AIM), which applies Service Oriented Architecture (SOA) to the problem of managing the interactions between applications that comprise integrated solutions.

Background

In this section we describe what is meant by Service Oriented Architecture (SOA) and the problems we are trying to solve by applying it. In the next section, the Application Integration Manager (AIM) invention that implements and elaborates this SOA vision is described.

Service Oriented Architecture (SOA) is best described by Figure 1 below. Formal computing standards that correspond to the components in the diagram are Universal Description Discovery and Integration (UDDI) for the web services registry and Web Services Description Language (WSDL) and SOAP (originally, Simple Object Activation Protocol) for the interactions between service client and service provider. Numerous technology vendors as well as Open Source organizations provide implementations of these standards. This invention does not re-invent SOA or its supporting standards and their implementations. Instead, it creates added value that is above and beyond what is defined and provided for by such base technology.

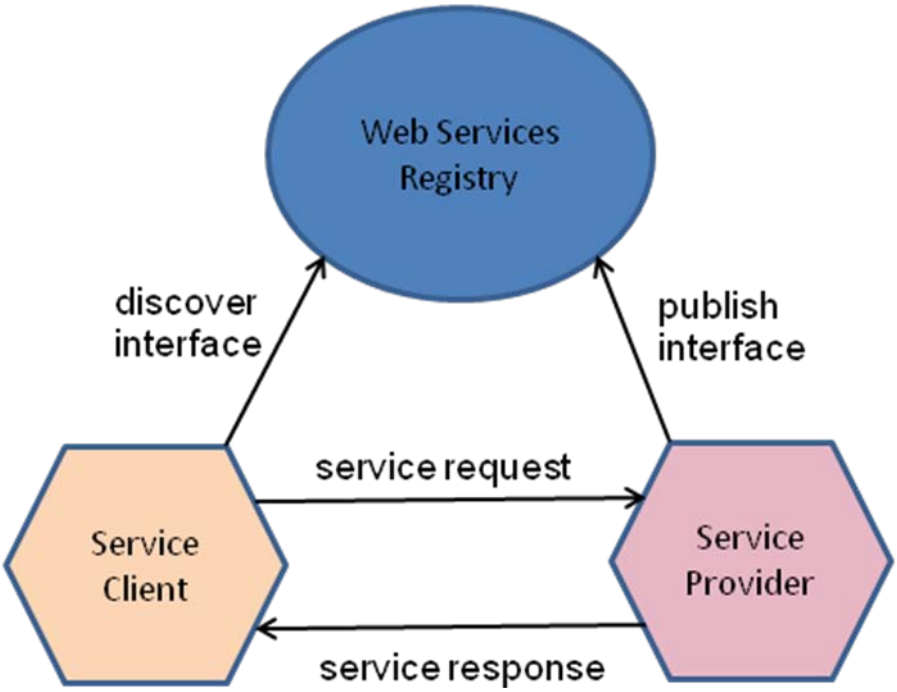

Figure 1. Service Oriented Architecture (SOA)

In this architecture, service providers are able to publish their interfaces in a web services registry. Potential service requestors or clients of those services can discover these service interfaces and their corresponding endpoints or implementations in the registry. Such clients can then invoke the services of the providers dynamically. This interaction model is the essence of loose coupling and has a number of significant advantages over more traditional application integrations. These advantages include:

- Location independence (clients don't need to know the endpoint they will talk to until run-time)
- Version independence (implementations can be versioned without the integrations breaking)
- Implementation independence (different endpoints that have different technology can implement the same interface)

In particular, SOA will address the following problems that consistently arise with more traditional integration technology:

- Brittleness
- High cost of development and maintenance
- Difficulty of Installation (impacting customers' time-to-value)
- Lack of agility (causing customer to be averse to upgrades)

Solution

The Application Integration Manager (AIM)

The Application Integration Manager (AIM) uses the basic concepts of SOA and elaborates key aspects of it to create a complete solution that allows the integration of various applications and application components. Without these additional features, SOA is a conceptual, enabling framework and not much more. AIM, described below, is the realization of the SOA conceptual framework that provides the tools and services to achieve the kinds of loosely-coupled, dynamic software integrations that are the intent of SOA.

Use Case

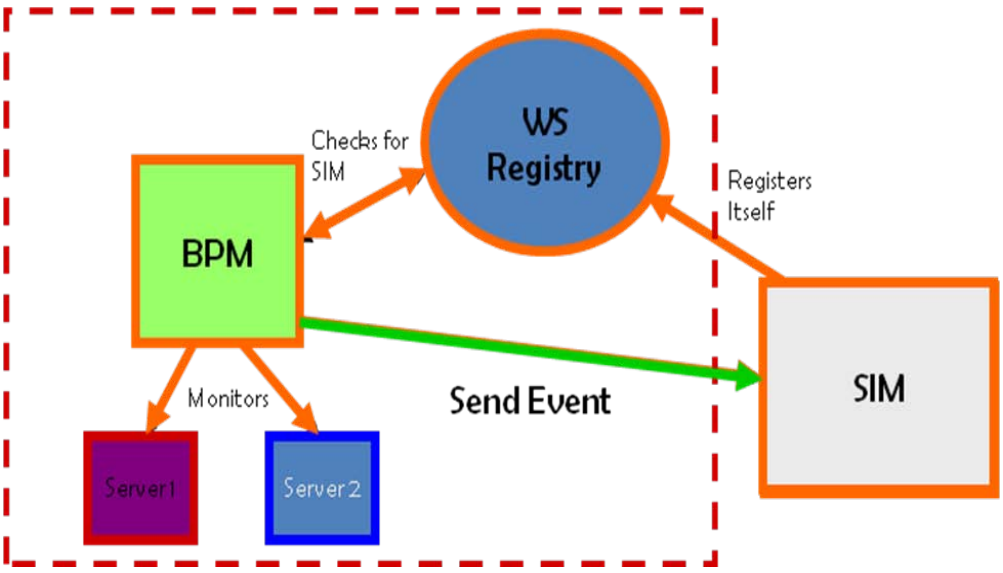

Figure 2. BPM – SIM Integration

Figure 2 above depicts the kind of integrations that are enabled with AIM. In this use case, the customer would like to have near real-time events fed into the Service Impact Manager (SIM) application. This is done so that service models stored in SIM accurately reflect the actual operational nature of the enterprise. Furthermore, let's assume that SIM and BMC Performance Manager (BPM) applications are purchased and/or installed by the customer at greatly different times (for example, SIM is installed one year after the installation of BPM). When SIM comes on-stream it registers its services in the Web Services (WS) Registry. At that point BPM can either find out (by query) or be notified that SIM services are available. Without any additional administrative or configuration efforts by the customer, BPM can now communicate events of significance (e.g., server 2 going off-line) to SIM using the SIM Send Event Service. SIM can use such event information to update its service models accordingly.

Features

To enable integrations such as the one described above, it is necessary to introduce some additional features to the basic SOA framework. These additional features are the essence of this invention. These features are:

- Registry-based Dynamic Integration and Configuration of Applications
- Registry-based Versioning of Service Interfaces and Implementations
- Registry-based Disambiguation of Service Implementations
- Registry-based Resource Management
- Web services-based Inter-Component Communication

Each of the above features will be described below in its own subsection.

Registry-based Dynamic Integration and Configuration of Applications

APIs require client applications to bind to pieces of code that are dependent on implementation details such as the programming language, runtime libraries and so on. In the loosely coupled integration scenario enabled by AIM all these details are hidden from the client. In addition, the client will only need to resolve the location (or *endpoint*) of the service when it needs to (or *dynamically*). AIM enables this loose coupling by requiring the integration APIs to be implemented as web services and then publishes the interfaces of these web services in the registry.

Complementing this dynamic integration capability, AIM enables application configuration by further utilizing the web services registry. A special service, the *Configuration Service*, is registered initially when the registry is installed in an operational environment. Instead of each application having to store numerous configuration properties in a variety of configuration files that need to be read and re-read, the configuration properties of respective applications are provided by this configuration service. This avoids much of the administrative cost associated with manually managing such configuration information.

Registry-based Versioning of Service Interfaces and Implementations

Web services APIs, as with most software, need to be versioned. Versioning of web services APIs arises due to a number of requirements, including bug fixes, enhancements or other updates to either the web services interface that defines a given API or its associated implementation. Although this problem is generic and widespread, there is no formal standard or industry de-facto practice that solves the problem. Many partial solutions are in existence in the industry. The most prevalent form of versioning for web services is done by versioning the XML Schema namespace that is related to the web service. The approach taken in AIM complements this common practice by making use of taxonomical data structures in the registry to support web services versioning.

By using such registry data structures, AIM solves this problem for both web services interfaces and their associated implementations or endpoints. This is a key feature as web services interfaces and implementations may be (and often are) independently versioned. Associated with the storage of this information is a data model that specifies how versions are identified and ordered. Finally, there is a set of web services operations defined that allow a client application to create, query, and modify this version information.

Registry-based Disambiguation of Service Implementations

SOAP-based web services generally have their respective interfaces described in WSDL. For a given WSDL-based interface there may be 0 to many implementations in existence that implement that interface. In the cases of 0 or 1 implementations, client applications have a straightforward job of determining what implementation to bind to. In cases of more than 1 implementation, clients must have some knowledge that is used to determine which endpoint is appropriate. Often this knowledge is hard-wired or determined out-of-band.

AIM solves this service ambiguity problem by providing a data model and access method that includes information about the service implementations that differentiates the implementations from each other. This data model includes information about the location (geographical or otherwise) of the service, the organization (e.g., company or department) the implementation belongs to and so on. These disambiguation criteria are stored as name-value pairs. To enable extensibility, there is a feature to enable user-defined disambiguation criteria to be added.

Registry-based Resource Management

Increasingly, IT resources expose a management interface that can be queried for metrics and in some cases used to actually control such resources. Often, but not always, such interfaces are based on the WS-MANAGEMENT standard.

One problem introduced by the use of such interfaces is how to discover where the endpoints are. Generally this information is communicated through configuration properties or by some out-of-band mechanism. AIM improves upon this by registering all such managed resource endpoints in the web services registry. In addition, the other features of AIM (versioning, disambiguation, etc.) are available to be used in conjunction with this feature.

Web Services-based Inter-Component Communication

The problem is to how to enable web components (portlets, applets, etc.) to inter-communicate in a way that is platform independent. An example of this problem is communicating between a Java Portlet and a SharePoint Web-Part.

AIM solves this problem by providing a simple web services API for inter-component communication. As this is a web services API, it will work across platforms and across implementation languages. In addition, when used with the other features of AIM, endpoints that implement this API will have the advantage of versioning, disambiguation, and configuration management.