Enterprise Integration Agility

by Jeremy Deane

According to Programmable Web in 2010 the rate of growth in public Web APIs doubled[1]. This exponential trend continues in 2011 resulting in an ever more connected web. This connected contagion is not just relegated to the domain of Web 2.0 but has infected the corporate world. In fact, companies are becoming more reliant on Software as a Service (SAAS) to provide key business functions. In this article, we will explore several options for rapidly delivering flexible-integrated solutions.

Houston we are tightly coupled!

With each passing moment applications become more interconnected. At the same time application technologies are rapidly evolving. Runtime platforms such as .Net and Java now support a myriad of languages each with its own nuances (e.g. dynamic, strongly typed, functional, and object-oriented). A polyglot software engineer has at his disposal an array of tools and techniques to manufacture any type of application. But just as important is how these heterogeneous applications are integrated in such a way that allows them to evolve independently over time.

The conventional approach is to integrate heterogeneous applications using web services. After all, web services are based on open standards and current application frameworks have made it easy to expose existing functionality via REST or SOAP. In addition, deadlines and budgetary constraints provide viable excuses for implementing such point-to-point (P2P) integrations. Unfortunately, P2P integrations tend to use synchronous protocols and tightly couple the integrated applications. Furthermore, for each additional P2P integration the cost of maintaining all P2P connections increases exponentially[2].

What is required when connecting heterogeneous applications is integration agility. This agility can be achieved using a Message, Service or Resource Oriented Architecture. Each architectural style is distinct in its philosophy but common to all these styles is the use of middleware and Enterprise Integration Patterns[3]. A centralized logging application and its clients are used to highlight the differences between these three architectural styles.

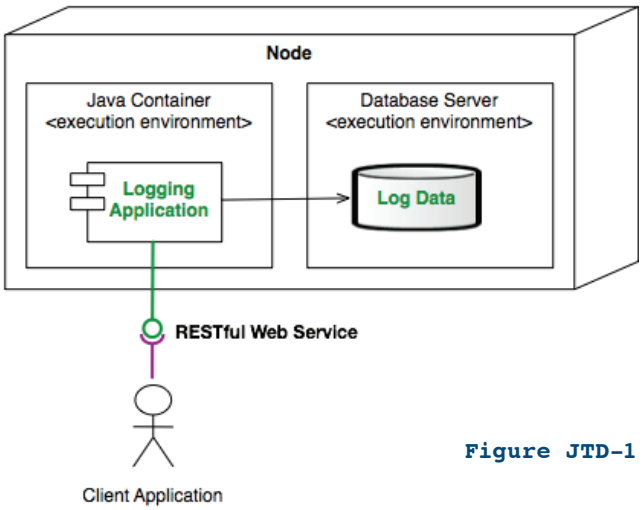

Figure JTD-1

The logging application, Figure JTD-1, is a Spring MVC web application that provides a RESTful API for submitting and retrieving application log entries. Client applications submit log entries directly to the logging application (P2P integration). This synchronous integration makes the clients responsible for multi-threading the call and submitting the log entry either as an XML document or a JSON object. Most importantly, any change to the logging service API directly impacts all clients.

1 API Growth Doubles in '10, Social and Mobile Are Trends

2 Bottom Line SOA by Marc Rix

3 Enterprise Integration Patterns

Message Oriented Architecture

Message-Oriented Architecture (MOA) is the most mature of the three architectural styles with proprietary solutions such as IBM MQ and Tibco available since the mid-90s. Fortunately, open source message-oriented solutions are now available including RabbitMQ, JBoss HornetQ and Apache ActiveMQ. Common to all these solutions is the concept of a message that is composed of a header and body. Message headers contain properties about the message such as its unique identification, content type, and where to reply-to (if necessary). The content of the message body ranges from a serialized object to plain text.

Message-oriented solutions utilize an asynchronous protocol for exchanging messages between two brokers using queues or topics (a.k.a. Publish-Subscribe). Even though the communication mechanism between the brokers is asynchronous, the delivery of the message can be guaranteed using local storage (e.g. file directory or database). Message brokers can either be embedded within an application or exist as standalone middleware.

Apache ActiveMQ is one of the most widely used open source message-oriented middleware solutions. ActiveMQ has a small footprint and is very easy to install and configure. In addition, ActiveMQ supports clustering and failover, enabling highly available topologies. Finally, ActiveMQ supports a wide range of protocols such as HTTP, TCP, XMPP and even UDP.

The second version of the centralized logging solution, Figure JTD-2, has the logging application subscribing to a topic hosted by a standalone ActiveMQ message broker. Client applications can post log entries to a topic using the Java Messaging Service (JMS) or can post log entries using a RESTful web service. The clients are now decoupled from logging application. (see Figure JTD-2)

A key advantage of this message-oriented solution is that other applications can subscribe to the log entry topic. For instance, the security team may want to monitor all log entries for Personally Identifiable Information (PII) or the operations team may want to generate Simple Network Management Protocol (SNMP) alerts in the case of a critical error. In addition, Apache Camel can extend the base message exchange patterns supported by ActiveMQ to support more complex integration patterns such as dynamic routing and re-sequencing.

Service Oriented Architecture

Thomas Erl best describes Service Oriented Architecture (SOA) as an architectural style based on a set of service principles[4]:

- Services are abstractions
- Services are discoverable
- Services are stateless
- Services are reusable
- Services are composable
- Services are based on contract
- Services promote loose coupling

A service abstracts functionality provided by an application such that it is discoverable and reusable by client applications. The stateless exchanges between the client and service provider are governed by a contract. Most of these principles apply to web services. However, as stated earlier web services do in fact tightly couple the client and service provider.

An Enterprise Service Bus (ESB) is middleware that implements enterprise integration patterns and is a key component of SOA. The core capabilities provided by an ESB include mediation, routing, and transformations.

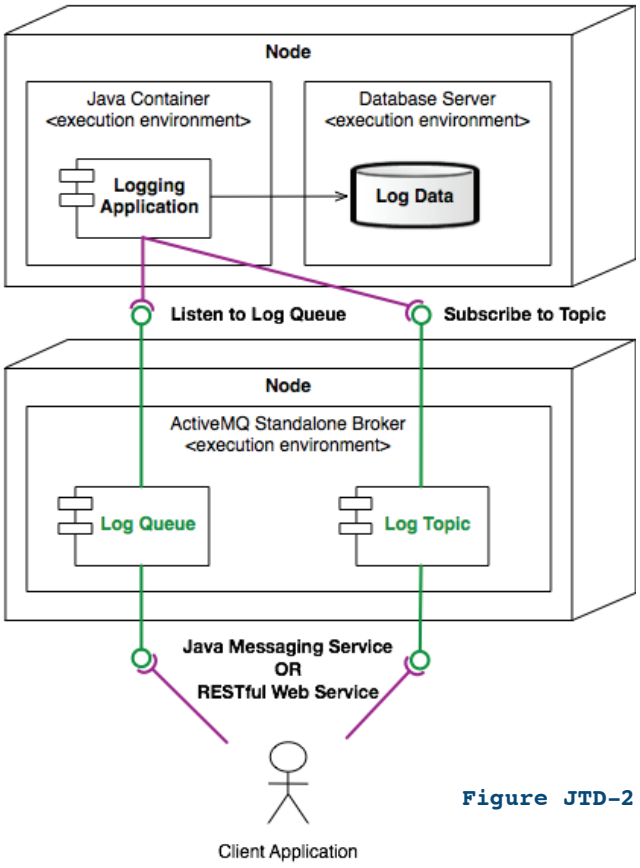

Figure JTD-2

4 SOA Principles by Thomas Erl

In some cases, an ESB may also provide some level of orchestration support. While many of the initial ESB solutions were existing Java EE Servers augmented to support integration patterns, Mule was built from the ground up to function as an ESB.

Mule is a lightweight open source ESB that's foundation is Staged Event Driven Architecture (SEDA). SEDA is similar to pipes and filters in that it breaks down a request into a set of stages each reacting to an event. Each event can be processed in a separate thread allowing the overall process to take advantage of a multi-core environment. Furthermore, Mule has matured over the years from an initial set of specialized object libraries to a declarative configuration approach based on the Spring Framework.

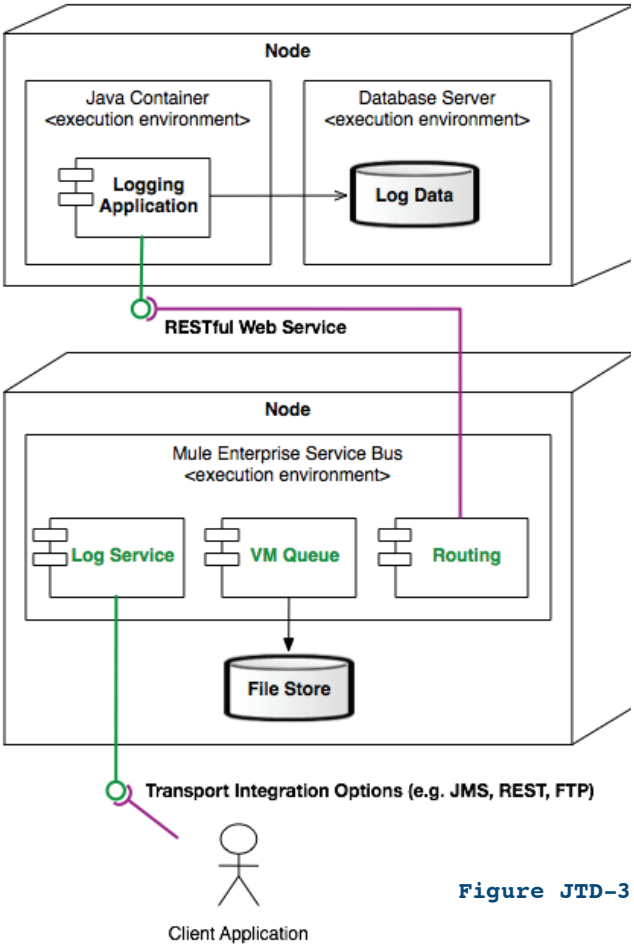

Figure JTD-3

The third version of the centralized logging service, Figure JTD-3, uses Mule ESB as a service proxy to the logging application. Mule ESB allows clients to submit log entries via a myriad of protocols, such as HTTP and JMS, and allows for transformations between various formats, such as XML and JSON. Furthermore, an internal VM Queue is used to guarantee delivery of the log entry to the logging application. This solution has more moving parts but provides more out of the box support for complex integration patterns.

Resource Oriented Architecture

Resource Oriented Architecture (ROA) extends the REST architectural style and provides a deeper, more extensible and transport independent foundation[5]. While RESTful web services require the use of HTTP, a resource-oriented service supports additional transports such as JMS. Resources are the foundation of ROA; a resource is an abstraction of information that can have different representational forms such as a HTML, JSON or XML. Each immutable resource representation is identified by a relative Universal Resource Indicator (URI) and may contain links to other resources.

1060 Research's NetKernel is built upon a Resource-Oriented Computing Platform that provides separation of logical requests for information (resources) from the physical mechanism (code). NetKernel resources are identified by a URI managed within a logical address space. A REST-based micro-kernel handles all requests for resources, resolving the URI from the address space and returning a representation of that resource. Requests to the micro-kernel can also be made to create new resources or update or delete existing resources.

The use of a REST-based microkernel has several key benefits. First, interactions are at a logical rather than physical level. Second, the results of requests are cached, thus decreasing the overall cost of composition and orchestration. For instance, if a set of resources relies on a common resource, the underlying physical code of that common resource is seldom executed. Finally, all internal requests to the microkernel are asynchronous, allowing processing to scale linearly as more cores are added to the host server. (see Figure JTD-4 on the next page)

The fourth centralized logging solution, Figure JTD-4, allows a client to create logging entry resources hosted by NetKernel using almost any protocol. NetKernel in turn delegates persistence of these log entries to the logging application. Similar to the ESB solution, NetKernel also supports out of the box

5 Introduction to Resource Oriented Computing by 1060 Research

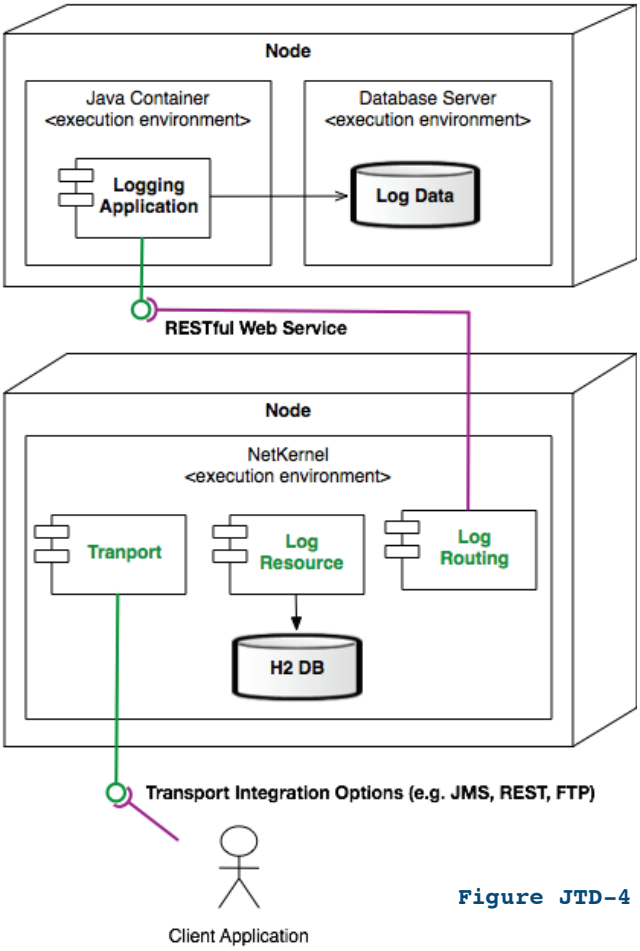

Figure JTD-4

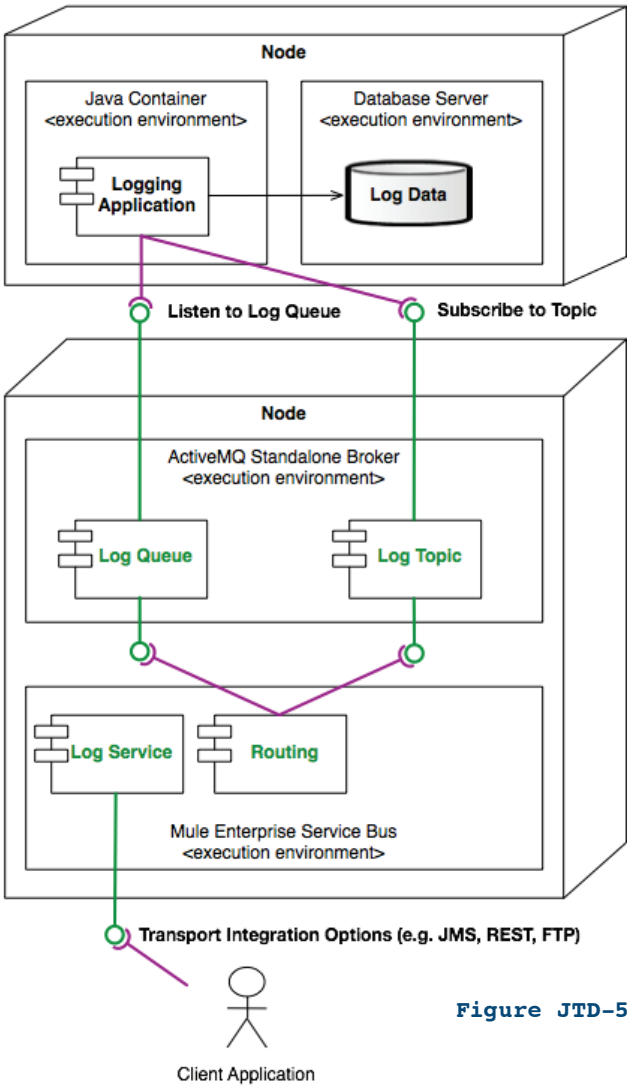

Figure JTD-5

complex integration patterns. The primary difference between NetKernel and Mule is hosting logical resources vs. abstracted services.

Enterprise Integration Agility

The three architectural styles based on messages, services, and resources each provide integration agility and flexibility by decoupling consumers from providers. Eliminating costly point-to-point connections lowers cost of ownership and increases the possibility of reuse. While each style could be used independently, often the styles are combined to address the unique requirements of the integrating applications. (see Figure JTD-5)

The final centralized logging solution, Figure JTD-5, provides a flexible integration platform that is used to exchange messages, call services, or access resources. Mule ESB acts as a service proxy to the logging application and performs any necessary service mediation. ActiveMQ provides a messaging backbone to guarantee delivery of the log entries. And with this solution, either Mule or ActiveMQ can be used to handle complex event processing.

Enterprise Integration agility provides a type of insurance against certain change. Whether there is a change in protocol, data, or technology this approach to application integration allows for adaptation and incremental adoption. Most importantly, this approach provides a foundation for evergreen enterprise solutions.

About the Author

Jeremy Deane is Director of Research & Architecture at Plymouth Rock Assurance and has over 15 years of software engineering experience in leadership positions. His expertise includes Enterprise Integration Architecture, Web Application Architecture, and Software Process Improvement. In addition, he is an accomplished speaker and technical author.

jeremy.deane@gmail.com