Multi-Middleware Web Services

Steve Vinoski
Chief Engineer of Product Innovation
IONA Technologies
200 West St.
Waltham, MA 02451
vinoski@iona.com

A few years ago, integrating applications simply required putting the right middleware in place. Today, however, the proliferation of different middleware approaches and technologies has complicated matters. Application integration now implies middleware integration. Essentially, we need "middleware for middleware"[1] that allows us to non-invasively integrate disparate middleware systems. The economic downturn of recent years has led many to question large and expensive "rip and replace" projects. Such projects, which attempt to homogenize the enterprise by ripping everything out and replacing it with the latest silver bullet, almost always fail. IT groups instead prefer to leave working systems in place and get them to talk to each other without changing them, regardless of which suppliers they come from, which technologies they employ, what languages they're written in, or what protocols they speak.

Middleware integration requires finding abstractions rich enough to represent a wide variety of design approaches, and yet concrete enough to embrace and support the numerous technologies used for service implementation. Naturally, finding practical middleware integration approaches is not easy. For example, early Web Services adopters embraced SOAP because it was much easier to add to new systems than other traditional middleware. However, SOAP is only a network protocol, and thus it can't provide all the abstractions necessary for multi-middleware integration. In fact, SOAP does nothing to help link together systems that are already deployed, because adding SOAP to such applications is just as invasive as previous middleware approaches. Invasive changes to working systems are error-prone, and they result in expensive and time-consuming retesting and redeployment.

In an ideal world, multi-middleware integration would allow you to create new business services by getting your existing systems to interoperate without changing them at all. But isn't this what Enterprise Application Integration (EAI) is all about? EAI proponents have long claimed that it would supply easy, seamless, and non-invasive integration, but in practice, this has usually not been the case. Typical EAI projects cost millions of dollars, run well behind schedule, and ultimately fail to deliver viable integrated systems.

Fortunately, Web Services technologies provide exactly the abstractions required for non-invasive multi-middleware integration. Specifically, the features and flexibility of the Web Services Description Language (WSDL)[2] enable you to "reverse engineer" existing applications to prepare them for integration with other disparate systems. WSDL types and port/interface definitions support logical descriptions of existing services, while

WSDL binding extensions define physical connections to those services not in terms of SOAP, but in terms of their existing protocols and message formats. Combined with the right underlying "middleware for middleware" support, WSDL-based abstractions and bindings can be used to create integrated systems that employ multiple protocols, multiple message formats, and multiple interconnection patterns.

One project using WSDL in this fashion is the Apache Web Services Invocation Framework (WSIF).[3] Because SOAP alone is not enough for real-world integration, the goal of WSIF is to supply a Java API that allows an application to transparently invoke web services via a variety of protocols. With this approach, service contracts can be defined abstractly using the logical features of WSDL. Then, for each service instance, these logical definitions can be joined with the appropriate physical bindings that define connectivity to that service. The logical/physical separation allows client applications to depend only on the logical portion of the WSDL definition, keeping them independent of the actual protocols and message formats required to communicate with a specific service instance. Client applications rely on separate WSIF *providers* that support connecting to services implemented using EJB, JMS, JCA, or local Java objects. These providers can be dynamically loaded, allowing the actual binding to a service to be chosen at runtime. Adding support for services implemented using another middleware system, such as MQ Series, simply requires writing a new provider. WSIF makes these different provider implementations completely transparent to client applications.

The WSIF approach is definitely a step above the typical non-abstracted, SOAP-only web service applications that other frameworks support. Ultimately, though, multi-middleware integration requires features beyond those of WSIF. In fact, multiple approaches are needed, depending on whether or not non-invasive integration is required.

Non-invasive multi-middleware integration requires WSDL routers that can support multiple protocols and multiple message formats simultaneously, bridging each as required. For example, if application A and application B are based on the same middleware, the WSDL router should pass their messages straight through without any conversions. For applications that use the same message formats but different transports, such as one using SOAP over HTTP and the other using SOAP over MQ, the router should perform protocol conversions but avoid unnecessary message conversions. This "on-demand" conversion approach makes the WSDL router as efficient as possible without penalizing the integrated applications with unnecessary conversions into intermediate formats or protocols.

In cases where application changes for integration are acceptable, however, multi-middleware capabilities can be added directly to each application. This allows them to talk directly to each other through their own internal multi-middleware switches, rather than relying on a centralized router. In essence, this approach is much like WSIF, except that it's not limited only to clients. It allows services to be implemented such that they too can be hosted transparently over multiple protocols and messaging formats.

While Web Services focus is often limited to only SOAP-based applications, it's clear that WSDL fulfills an even more important role in intra-enterprise consolidation and integration. Projects like Apache WSIF, and products such as IONA's Artix[4], which already supports the multi-middleware routing and switching approaches described here, are using WSDL abstractions and binding extensions to maximize the value of existing IT assets. The days of expensive, invasive, and failing EAI mega-projects are over.

References

1. Vinoski, Steve. "*Where is Middleware?*" IEEE Internet Computing, 6(2), Mar/Apr 2002, pp.83-85.
2. *Web Services Description Language (WSDL) Version 1.2*, W3C Working Draft, June 11, 2003, http://www.w3.org/TR/wsdl12/.
3. Apache Web Services Invocation Framework, http://ws.apache.org/wsif/.
4. IONA Artix, http://www.iona.com/products/artix/artix-relay.htm.