

Establishing Connectors as Integration Services

M. Hepner R. Gamble

Dept. Mathematical and Computer Sciences, University of Tulsa

Tulsa, OK 74104

gamble@utulsa.edu

1. Introduction

A “pure” service-oriented architecture (SOA) is an architectural style with a loose-coupling of services (components) that interact via message passing (connectors) through the standards based and WSDL defined service interfaces regardless of the interacting components’ heterogeneity.

SOA is applied to component-based systems to migrate them to services. Interoperability issues are often resolved using technology specific integration mechanisms (such as Java’s RMI). These mechanisms bypass the WSDL interface and are tightly-coupled to proprietary integration technology, the component location, or implementation details.

Incompatibilities between components and SOAs occur in three main areas: protocol, interaction, and location. Components must support SOAP and HTTP protocols and SOAP fault processing [1]. A component may expect interactions to be synchronous or asynchronous, blocking or concurrent, and these expectations may contradict the SOA or workflow requirements. Typical SOA interactions include one-way messaging, a request-reply interaction, or a basic-callback interaction which involves both synchronous and asynchronous behavior [1]. SOAs expect delayed binding using a UDDI (Universal Description, Discovery, and Integration) process but many components support neither late binding nor UDDI activities [2].

Vendors advocate integrations using a service-based interface together with ad hoc integration functionality, some of which is specifically attached to the component as a wrapper [3]. Conventional wrappers do not provide the flexibility that is needed in evolving systems because for every component upgrade, the wrapper may need reanalysis and redesign [4].

One industry practice is to implement an integration service that will be the component interface, pushing the component outside the service boundary. The problem is that these integration services have no design basis, but are pieced together in an ad hoc fashion that may lead to over integration,

redundancy, and inconsistency in interoperability expectations and implementations.

Connectors, on the other hand, are elemental, composable functions that encourage a loose coupling of interface-specific details from the integration function requirements [4]. We argue that this separation of integration and interface functionality makes a connector ideally suited to resolve SOA integration issues. However, guidelines are needed to architect integration services that better organize connectors between the integration service and the component. We propose guidelines that outline when connectors are best part of the integration service and when they should be outside the service boundary. The guidelines support the primary objectives for migrating to SOAs: *loose-coupling* and *service coordination*.

Services use open standards for message based communication with other services which promotes loose coupling. This loose coupling is exploited through the use of web service coordinations.

SOAs can employ a single service in multiple applications using coordination specifications. In this way, services interact without forming a direct connection with or dependency on other services. Integration solutions must support this interactive style by limiting the deployment of ad hoc code lacking sufficient design rationale. Ad hoc code creates artificial structure or dependencies between services and components that are unrelated. As a result, these tightly-coupled solutions are incompatible with coordination specification standards.

One important coordination function is scoping. This allows a coordination to properly detect and handle service failures. Supporting this ability can affect the positioning of the connector(s) in the architecture. A poorly positioned connector, such as an improperly shared one, may force a relationship among components that prevents a coordination from managing their processing concurrently or sequentially, as needed. We utilize the design mechanism, architectural connectors as integration services, to reduce redundancy in integration

functionality and maintain consistent data, while supporting service coordination specifications.

2. Guidelines for Integration Services

There are logical constraints on using integration services inside a SOA while maintaining the above goals. To start, avoid over integration. Embody an integration service with the *minimal amount of integration functionality* needed for the service to perform the task of making a single component interoperate within the SOA. This reduces the ripple effect of modifications required when that component is updated or changed and may avoid a direct impact on the coordination.

Next, architects should *avoid redundancy*, i.e. placing the same integration functionality in multiple places, as this increases the complexity of coordination specifications. This is mainly with respect to integration services where sharing is appropriate. For connectors outside the service boundary, sharing should only occur if the components for which they facilitate interaction are highly similar. This appears to be somewhat of a subset of minimization, but its objective is slightly different. The previous constraint was to maintain only necessary integration services, leaving the rest as connectors outside the service boundary. Here, we examine the set of integration services and whether they can be shared, overloaded, or redesigned to limit or avoid redundancy.

Finally, integration services must be designed to detect failures or other execution problems that compromise *data consistency*. This constraint relates to the coordination goal of not imposing artificial relationships among components. This constraint also serves to balance the minimization and redundancy avoidance constraints presented above.

In Figure 1, two inventory management systems are shown which both require an integration service to be capable of joining a service coordination. The coordination sends inventory requests to both components. One inventory system uses metric units while the other uses US customary units. The translating connectors are distinct in their conversions, but the routing connector that processes the inventory requests and delivers the responses is the same for each component and can be shared without relating the components to each other. Thus, only the router needs to be represented as an integration service and can place the translated values into a response message without regard to their respective units. This insulates the integration service from changes to units and currency. Further, the integration service design allows either or both inventory management systems to be reused in other coordination specifications without re-analysis or redesign of another integration service. The

integration service reuse is further enhanced by the router design which does not impose a relationship between the two inventory management systems so that concurrent requests can be processed by both systems.

3. Concluding Remarks

The basis of the research is to manipulate and organize connectors to determine what goes inside the service boundary in the form of an integration service and what does not. The goals of minimizing integration service functions, limiting redundancy, and requiring consistency lead to enhancing the workflow specification of the business. Other issues such as security and coverage, with respect to the completeness of the integration functionality, are subjects of future research.

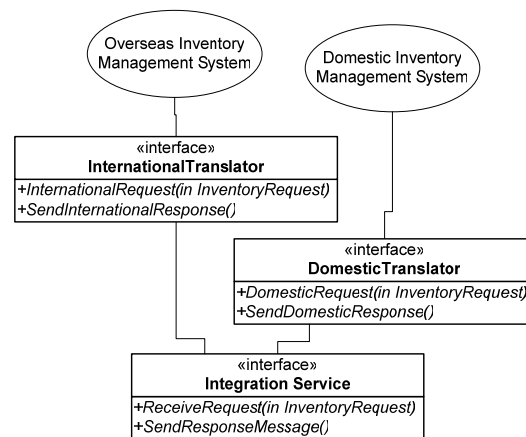


Figure 1. Shared Integration Services

4. References

- [1] <http://www.ws-i.org>, "Web Services Interoperability Organization."
- [2] L. Davis, Gamble, R., Payton, J., Jonsdottir, G., Underwood, D., "A Notation for Problematic Architecture Interactions," presented at Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Vienna, Austria, 2001.
- [3] R. Coqueret, Fiammante, M., "Choosing among JCA, JMS, and Web services for EAI," <http://www-128.ibm.com/developerworks/webservices/library/ws-jcajms.html>, 2003.
- [4] B. Spitznagel, Garlan, D., "A Compositional Formalization of Connector Wrappers," presented at International Conference on Software Engineering, Portland, Oregon, 2003.