Elevating Interaction Requirements for Web Service Composition

M. Hepner M. T. Gamble R. Gamble
Department of Mathematical & Computer Sciences
University of Tulsa
gamble@utulsa.edu

Abstract

Web services are increasingly utilized to create integrated applications from existing components. However, incompatible interaction expectations between Web service interfaces and/or non-Web service interfaces participating in the overall application can inhibit the integration. Frequently, these conflicts are resolved on a case by case basis. Advantages can be gained by understanding integration conflict resolution as formal interaction requirements. This approach enables evaluation of recurring integration conflicts during requirements analysis (rather than as a testing step), thus providing a more complete resolution that is abstracted from specific runtime interaction instances.

1. Introduction

Service-oriented architectures (SOAs) provide a design framework in which to utilize open standards such as Web services (WS) to publish, discover, and communicate between service interfaces. These standards have a positive impact on integration. However, it is likely that substantial conflicts will arise which inhibit interoperability. This is especially true when legacy or third party systems (TPS) are migrated to a service interface [1]. Typically, integration approaches target particular service interoperability needs without regard for more complex application integration. These "hot spots" of integration difficulty focus the attention of service integration designers, but they may also serve to detract from more systemic integration concerns that cross multiple interfaces.

It can be argued that the move to a service-oriented design view is a true paradigm shift requiring service interactions be modeled abstractly as a first-class entity [2]. For example, data that is passed among the same components via non-WS interfaces can compete, conflict, or make inconsistent WS data exchanges. Such exchanges, cast in a WS framework tend to treat the service components as fully encapsulated behind WS interfaces. This is often not the case. Therefore, we advocate elevating the interaction requirements of WS to a level of abstraction in which multiple interface interactions can be expressed to form a *service interaction requirements document*.

Without consideration of the non-WS interfaces present in a component, there can be no guarantee of the correctness of the information processed and exchanged. At a practical level, most interesting systems are large, complex, and contain many proprietary components [3]. Documentation and understanding of the implicit activities of interaction within the system is often lacking. WS integration requirements therefore include:

- A consistent shared data model.
- Verification of all needed data for WS responses.
- Fault management, restart, and shutdown behavior must be made consistent with application expectations.
- Security controls must align with data sharing.

We outline three major requirements that, if not met, can lead to interoperability problems among WS. Once discovered, a design plan for solutions to these problems can be placed in a service interaction requirements document to facilitate resolution.

2. Relevant Work

Requirements for interoperability or integration are examined in terms of migrating existing non-service components to WS [4-6]. The requirements of concern are granularity of service, performance, and semantic meta-data management. Thus, the focus of this work does not include interoperability conflict analysis between WS but rather the use of SOA design approaches as a solution to interoperability in existing architectures, components, and data sources.

Other work examines the process of assessing integration requirements fulfillment. However, in this context, *integration requirements* refers to the

application level requirements that component interaction must satisfy [7]. Thus, this assessment evaluates the integration solution's ability to support the stated application requirements but does not examine the cause of interoperability conflicts, choice of solutions, or flexibility and maintenance issues.

Frequently, integration projects do not apply analysis and design to the selection of an integration solution. This may be caused by a lack of understanding of the basis of interoperability conflicts or application of other decision criteria which are outside the bounds of design (e.g., technology commitments, managerial or corporate standards mandates and limitations of designer skillsets). In particular, the influence of commercial component vendors on integration solutions can be examined from a viewpoint analysis perspective [8].

For SOAs, interoperability refers to the ability of a service to be invoked by any potential client [9]. Interoperability conflicts have been researched in the form of post-integration analysis and in the improved interoperable design of components in the form of WS [1, 10, 11]. Open standards that increase WS interoperability include XML, WSDL, SOAP, HTTP, and UDDI[1]. Using these standards, WS bypass many, but not all, of the issues related to interoperability.

3. Motivating Example

We define a component as an independent and distinct unit that performs some processing task. Components may be legacy systems, third party systems (TPSs), and newly developed systems that may or may not be service based.

Company A wishes to automate order processing with their customers and suppliers. Systems involved in the application integration include the customer's requesting system, Company A's WS and SupplierX's and SupplierY's ordering systems, each shown with two distinct interfaces. Company A's internal systems used for inventory (*IMS*) and customer management (*CMS*) will also be involved in the order process. A representation of the components required in the application integration is shown in Figure 1. The figure displays only part of the data elements and operations which are relevant to our discussion.

[1]XML is the Extensible Markup Language standard. WSDL is the Web Services Description Language standard. SOAP is the Simple Object Access Protocol. HTTP is the Hyper Text Transfer Protocol. UDDI is the Universal Description, Discovery, and Integration standard.

In the proposed order processing system, customers scan information about product items and place orders for the items. When a customer's *OrderRequest* is received by Company A's WS, then send a request (*ItemInvRequest*) to the Inventory Management System (IMS) to verify the item is in inventory. If the inventory quantity is too small to fulfill the customer's order, an *OrderRequest* is automatically placed to the appropriate supplier (SupplierX or SupplierY). After this processing successfully completes, an *OrderResponse* containing a confirmation is sent to the customer by Company A's WS. The last processing step logs the customer's order in the Customer Management System (*CMS*) for record keeping.

Company A uses an in-house WS to accept customer orders. Both suppliers offer WS interfaces for processing orders. The CMS and IMS systems are TPS components that provide WS interfaces. This allows their integration into the application using WS standards. The CMS and IMS systems also provide non service interfaces. For example, the CMS system contains a product catalog that is automatically updated by suppliers. The IMS system periodically updates its own internal inventory database with item information from the product catalog.

4. Service Interaction Requirements

The presence of multiple interfaces indicates less encapsulation of functionality and interaction of a component than may be expected by the service integrator. We advocate the formulation of comprehensive interaction requirements that include non-WS interfaces. Thus, all interfaces of a component must be exposed and uniformly expressed as requirements for interaction in order to compose WS properly and without conflict.

For the specification concerns in this paper, we make the following two assumptions about the application integrations being undertaken: (1) a component can have multiple interfaces, and one of the component interfaces is a WS interface. Each interface is distinguished by the function it performs. Overlapping functionality would be grouped into a single, "larger" interface description.

Let Interfaces(C) be the set of interfaces of component C, where $\text{Interfaces}(C) = \{I(C)_1, \ldots, I(C)_k\}$ with finite k. For the purpose of showing the need for requirements elucidation, we designate two specific interface types $I(C)_s$ for the WS interface and $I(C)_r$ for an active, non-WS interface. We can later expand this as needed to multiple interfaces of the same type or multiple different types and use the same principles.

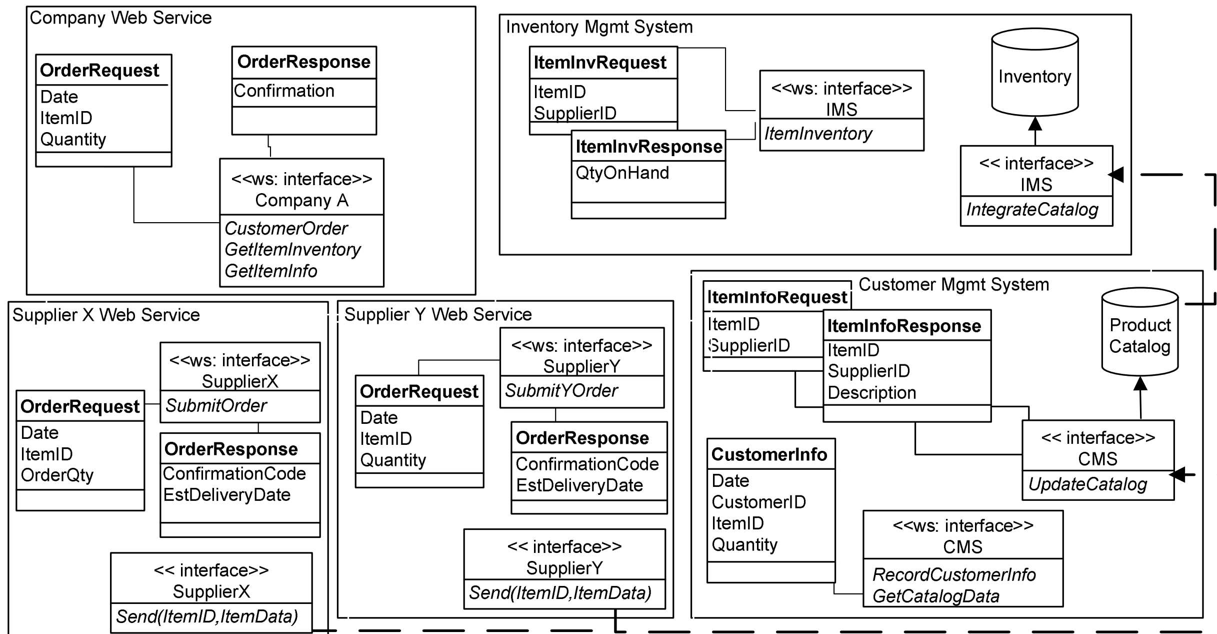

Figure 1. Component Interaction Example

Let D be the set of all possible distinct data elements. We assume that data elements (only) with the same name are semantically equivalent[2]. They are not necessarily syntactically equivalent, however. $I(C)_s$ contains an ordered pair of data element sets $D(C)_s = (\{d_i, \ldots, d_j\}, \{d_n, \ldots, d_m\})$ in which the first set of the ordered pair $D(C)_s$ is request data that can be retrieved by $Request(D(C)_s)$ and the second set is response data that can be retrieved by $Response(D(C)_s)$. $I(C)_r$ contains similarly structured data elements in $D(C)_r$, where $\bigcup D(C)_s$ and $\bigcup D(C)_r$ are subsets of D. Request data is sent by a source component and is received as request data by the sink component. Response data is similarly matched between source and sink.

Let A and B be components with $Interfaces(A) = \{I(A)_s, I(A)_r\}$ and $Interfaces(B) = \{I(B)_s, I(B)_r\}$. We illustrate three major requirements that indicate problems with WS integration.

4.1 Requirement 1 – Equivalent Syntax

The first requirement is that *the data exchanged between components is syntactically equivalent.* All possible interface pairs must be examined given more than two components and more than two interfaces per component. Given that we have limited our discussion to two interfaces – one of type s (WS) and one of type r (non-WS) – we examine the intersection of the data involved in the interaction between components A and B. Formally, if $Request(D(A)_s) \cap Request(D(B)_s) \neq \varnothing$ or $Response(D(A)_s) \cap Response(D(B)_s) \neq \varnothing$, then the WS interfaces share data. If this data does not have equivalent syntax, than an integration solution external to the interfaces is needed to bridge this gap through transformation. The same issue arises for the data sets compared in $D(A)_r$ and $D(B)_r$. The integration solution requirements from the discovery of this type of interoperability conflict should be explicitly stated in the service interaction requirements document.

Table 1 below illustrates the syntax of incompatible items from Figure 1. The data received from the customer in *OrderRequest* of Company A expects the *Date* to be formatted as *mm/dd/yy* (row 1). In contrast, *OrderRequest* of SupplierX uses *mmmddyyyy* date formatting (row 3, Table 1). In addition, the *ItemID* received in the Company A *OrderRequest* has a different format than the *ItemID* contained inside the SupplierX *OrderRequest*. The requirements to resolve these incompatibilities must state what the conflict is and between what entities.

[2] We refer the reader to the vast research on Semantic Web for issues that arise should these not be equivalent.

Table 1. Type 1 Interoperability Conflicts.

$D(A)_s$	$(A)_s$	Request	Type, Format, UOM
Date	Company A	OrderRequest	String, mm/dd/yy
Item ID	Company A	OrderRequest	String, ####-####
Date	Supplier X	OrderRequest	String, mmmddyyyy
Item ID	Supplier X	OrderRequest	String, ####

4.2 Requirement 2 – Complete Request Data

The second requirement is that *messages requesting service provide complete data.* In WSDL terms, the specific message elements required to invoke the service must be fully populated by the requestor. To illustrate, we introduce component F, with Interfaces(F) containing at least $I(F)_s$. Without loss of generality, let component F have a WS interface that receives a request from component A. Then it must be the case that the request data supplied by A encompasses the request data needed by F or $Request(D(A)_s) \supseteq Request(D(F)_s)$. If this is not the case, then component A cannot provide the complete data needed to request service from F. This raises a requirement for interaction and must be stated in the service interaction requirements document. Such a requirement is easily checked by reviewing the WSDL describing the service interface of F.

In Figure 1, when an item's inventory is requested, the message *ItemInvRequest* is sent from Company A to the IMS. This message contains an *ItemID* and *SupplierID*. Examining the *OrderRequest* message received from the customer, it is evident that no *SupplierID* is supplied to Company A. A requirement is needed to detail the processing to obtain the *SupplierID* before the *ItemInvRequest* can be sent. This processing may be added to Company A's WS, coded externally as an adapter to the IMS, or added into a service coordination specification.

4.3 Requirement 3 – Encapsulated Data

The third requirement is that *WS which share data elements must be (logically) encapsulated with respect to each other*. Data shared between two or more WS must be kept consistent such that the WS interfaces appear to totally encapsulate the components. This requirement targets the potential for competing, conflicting, or inconsistent data elements being exchanged outside the service interfaces. Our first concern is where two interfaces of the same component are either both requestors or both responders that share the same date model. That is, where $Request(D(A)_s) \cap Request(D(A)_r) \neq \varnothing$ or $Response(D(A)_s) \cap Response(D(A)_r) \neq \varnothing$. This is a concern because the non-WS interface can pass or accept the same data as the WS. Therefore, we must verify whether the two interfaces are functionally equivalent.

The second concern is the existence of another component sharing the data either as a requestor or a responder outside of the WS interface. In this case, the intersection above is extended as $Request(D(A)_s) \cap Request(D(A)_r) \cap Request(D(B)_r) \neq \varnothing$ (similarly for Response). This can allow a separate communication path for a shared data element and any elements dependent on it. Encapsulation, from a WS perspective, is broken in two components, A and B, if $Request(D(A)_s) \cap Request(D(B)_s) = \varnothing$ and $Request(D(A)_r) \cap Request(D(B)_r) \neq \varnothing$ (similarly for Response). Informally, the components share data in their non-WS interfaces, yet have no such relationship in their WS interfaces and is therefore invisible to analysis of the WS interfaces alone.

Note that $Request(D(A)_r) \cap Request(D(B)_r) \neq \varnothing$ (similarly for Response) does not necessarily imply that the components interact directly on their non-WS interfaces. Though the same data element appears in their interfaces, they may share data via indirect interactions with other components. The service interaction requirements document must flag the occurrence of data sharing across interfaces in order to guarantee the correctness of the WS composition

In Figure 1, components IMS and CMS have WS interactions containing common data elements (i.e., item identifiers). These same components also have non-WS interfaces which act as sinks for data items (via SupplierX and SupplierY) that eventually contribute to the derivation of *ItemID* in their WS interfaces. The non-WS interfaces allow the components to periodically receive updates to a local store containing item data in the form of a product catalog. To analyze only the WS interfaces, we must assume that each component completely encapsulates its own version of the product catalog, allowing it to be logically self-sufficient. For this assumption to be true, we must verify that the product catalog provided to each is the same and possibly immutable [12].

4.4 Interaction Requirements Document

Once the interaction discovery process has been completed, the service interaction requirements document will contain specific statements of potential

interoperability conflicts. This allows proper system design to be performed while taking into consideration how the system may evolve, the ease of system maintenance, and the system execution efficiency. For example, designers can plan to utilize shared integration code when applicable. In addition, application standards can be used to help maintain transaction data consistency and transaction state synchronization rather than creating a new solution for each system-to-system integration. This discovery process will encourage thoughtful analysis of the use and reuse of loosely-coupled integration solutions.

Table 2. Interaction requirements

Interaction Requirement		
	Requirement	**Syntactic Equivalence**
Source	Service/component Request/response Data element	*Company A WS* *OrderRequest* *Date*
Sink	Service/component Request/response Data element	*SupplierX WS* *OrderRequest* *Date*
Conflict	Contributing factor	*Different data formats*
	Resolution	*Convert dd/mm/yy to dddmmyyyy*
	Constraints	*none*

Table 2 above shows an example of an interaction requirement entry in the interaction requirements document. An association between the applicable application constraints and the interaction requirements will be created. This will allow analysis of the interaction resolutions by both the requirement type and the constraints.

Acknowledgement: This work is supported in part by a US AFOSR award FA9550-05-1-0374.

5. Conclusion

Since current integration solutions resolve interoperability conflicts on a case by case basis, there is no method of evaluating the consistent application of these policies. Our approach seeks to make the requirements analysis a repeatable process using a requirement type classification that leads to codifying conflict analysis results within a service interaction requirements document. This approach further allows integration design to be validated for consistency among other application policies and goals such as fault processing, component restarts and shutdowns, and security policy enforcement.

In addition, the identification of system interaction requirements allows future application growth to be considered along with the possibility of component and service upgrades and replacements. Such changes are treated as new integrations and flow back through the same requirements steps. When changes cause new interactions to occur, their interaction requirements may be analyzed to determine their type as well. Placement of integration solutions in the system architecture can have long-term effects for the system evolution and maintenance costs.

6. References

[1] L. Davis, Gamble, R., Hepner, M., Kelkar, M., "Toward formalizing service integration glue code," IEEE Int'l Conf. on Services Computing, 2005.

[2] L. F. Andrade, Fiadeiro, J.L., "Composition contracts for service interaction," *J. Universal Computer Science*, vol. 10, pp. 375-390, 2004.

[3] R. Ellison, J., "Trustworthy integration: Challenges to the practitioner, CMU/SEI-2005-TN-026," Software Engineering Institute, CMU, October 2005.

[4] G. Lewis, Morris,E., O'Brien,L., Smith,D., Wrage, L., "SMART: The service-oriented migration and reuse technique, CMU/SEI-2005-TN-029," Software Engineering Institute, CMU 2005.

[5] I. Wong-Bushby, Egan, R., Isaacson., C., "A case study in SOA and re-architecture at company ABC," Hawaii Int'l Conference on System Sciences, 2006.

[6] V. Niranjan, Anand,S., Kunti, K., "Shared data services: An architectural approach," IEEE Int'l Conference on Web Services, 2005.

[7] T. Wendt, Brigl,B., Winter., A., "Assessing the integration of information system components," presented at Workshop on Interoperability of Heterogeneous Information Systems, 2005.

[8] M. T. Gamble, Gamble, R., and Hepner, M., "Understanding solution architecture concerns," 2nd Int'l Workshop on Models and Processes for the Evaluation of Off-the-Shelf Components, 2005.

[9] M. Stevens, "Service-Oriented Architecture Introduction," *http://www.developer.com/services/article.php/1010451*, 2003.

[10] M. Hepner, Gamble, R., Kelkar,M., Davis,L, Flagg, D., "Patterns of conflict among software components," *J. Systems and Software*, vol. 79, pp. 537-551, 2006.

[11] M. Hepner, Gamble, R., "Connectors as integration devices," 5th IEEE/IFIP Working Conference on Software Architecture (WICSA), Components and Services Workshop, 2005.

[12] P. Helland, "Data on the Outside vs. Data on the Inside An Examination of the Impact of Service Oriented Architectures on Data," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/dataoutsideinside.asp.