

## Feature Article



### Understanding Service Composition, Part III: Dealing With Data

by John deVadoss

Published: June 11, 2010 (SOA Magazine Issue XL: June 2010)

[Download PDF](#)

[Digg This](#) • [De.licio.us](#) • [Slashdot](#) • [Technorati](#) • [StumbleUpon](#) • [Google Bookmark](#)

#### Introduction

To their detriment many service-oriented architecture efforts tend to prioritize and focus on the technology connectivity (such as SOAP, HTTP, and WS-\*), between and across services, over the business connectivity (such as, the task of exchanging and communicating business concepts and entities and their semantics). The ability to harmonize business entities and concepts across multiple services is critical to successful service composition.

How does this manifest itself? Let us once again consider the self-service application scenario; but this time, let us consider a customer scenario in a large bank. The customer service representatives require a single view of the customer in order to enable superior customer service, to enable better decision making and to enhance the relationship with the customer, both to retain existing customers as well as to acquire new ones. The challenge with building a service-oriented architecture to support these requirements is that often there is no single store of customer data in the bank; and, the customer data is fragmented across multiple legacy business systems.

In the real world there is often no single identifier for the customer data - the bank may have built some applications in-house, such as it may have acquired some other applications off-the-shelf and some services have been brought on-board as part of recent mergers with other smaller banks. Combining data to provide a single view of the customer is hard enough, but without a common identifier it is that much harder. In the best case, one business service may act as the system of record, with additional customer data living in different systems. In the worst case there may be no true system of record at all.

In this real world scenario, building a service-oriented self-service portal for the customer service representatives involves composing services exposed by these diverse, autonomous systems with their idiosyncratic views of business entities and concepts, and fragmented, replicated and sometimes conflicting data. Dealing with data is a key capability with respect to successful service composition as we will see shortly.

#### Key Concepts

##### *Schema Rationalization*

In order for business services to be able to communicate with each other and to enable meaningful service composition they need to be able to agree on certain notions and to have a shared view of core business entities. For example:

- What is the format for an Address?
- What is the schema for a Contact?
- What are the attributes of a Customer entity?

When implementing green-field service-oriented architectures this is less challenging as we typically have the ability to a priori define these formats, attributes and schemas, which are subsequently then shared by the various services. However, more often than not, we work with already existing business systems and services, integrating and extending these systems to participate in a service-oriented world; therefore, there is a requirement to enable these (legacy) services to be able to talk to each other at the business concept and entity level. These legacy systems and services typically possess their own definitions and views of business concepts and entities. Fortunately, Schema rationalization enables the commonality of many business entities across these diverse definitions and views.

In a nut shell, Schema rationalization is the harmonization, across services, of shared business concepts and entities to enable these services to be able to communicate with each other at the business entity level.

### *Entity Aggregation*

Whilst Schema rationalization is necessary for business services to be able to communicate with each other and to enable service composition, it is not sufficient.

Schema rationalization may enable the creation of a canonical Employee scheme definition that is shared across a set of business services; however, in the real world it is common to find more than one service defining and maintaining its own view of the set of employees. For example, the employee benefits system may define a view of employees in the organization; the employee payroll system may define a payroll specific view for the employee in the organizations; and, the performance appraisal system may define yet another view of the employees in the organization. In such a scenario, how would you present a unified view of the employees? This is where Entity Aggregation comes into play.

Business entities may exist in piece-meal, subsetting fashion across multiple systems. For example, the 401k system may possess a subset of the employee information, as does the employee benefits system, and the need is to aggregate the attributes for the employee from across multiple systems, in order to present a unified harmonized view. This is a somewhat simpler example of entity aggregation. In the real world, business entities are often replicated across multiple legacy systems; sometimes with conflicting schema definitions, and often with duplicate, missing and bad data. Entity aggregation refers to the set of technologies and capabilities required to provide a shared, common view of a business entity, such as an employee or a customer, as in the earlier banking service-oriented architecture, across diverse business systems and services.

### *Master Data Management*

As we have seen earlier, services possess business entities that need to be shared and used by other services. There is a class of these business entities that is usually termed master data.

Master data is used to refer to the critical nouns of an organization, which broadly fall into four sets: people, things, places, and concepts. Often, master data is distinguished by the word, master in the business concept (for example, account master). Master data usually refers to the less-volatile business concepts, and is differentiated from the verbs (purchase, shipping), which fall into what is commonly referred to as transactional data.

Master data tends to be used by multiple services. Errors in master data are likely to cause inaccuracies and errors in downstream service composition and usage. For example, an online retailer rendering an incorrect price on a toy during the holiday season due to an error in the item master can lead to a marketing fiasco.

Managing master data is more and more a necessity for realizing service-oriented architectures, particularly in the context of governance and regulatory compliance.

## Aspects of Implementation

### *Schema Rationalization and Canonical Schemas*

Schema rationalization allows the commonality of many business concepts as we have seen earlier. In the real world there are usually two techniques that are used in practice to achieve schema rationalization.

### *Reader-Makes-Right*

In this approach, on receipt of an incoming message, the receiving service fixes the message so as to be able to process and execute the contents of the message. The source schema (of the incoming message) and destination schema (as used by the receiving service) are compared and a custom map is created and applied in order to transform the incoming message and its contents.

A challenge with this approach is that this often leads to N-squared combinatorics. Every service needs to be aware of the schemas used by the services that it wants to communicate with. With  $n$  services,  $n * (n - 1)$  custom transformations will exist, and as  $n$  gets large the complexity of this approach, it will greatly increase, which leads to the next approach.

### *Canonical Schemas*

In this approach, services agree on a shared, common, canonical set of schemas. This entails modeling the business entities, agreement on naming conventions and on XML name spaces, as well as the data types and attributes.

These canonical set of schemas represent the authoritative, common definition for the shared business concepts and entities and obviates the need for the N-squared combinatorics of the reader-makes-right approach. One side-effect of creating a canonical set of schemas is that other definitions can be derived from this set, leading to traceability; another key side-effect is that it leads to intrinsic interoperability at the business entity level.

A canonical set of schemas is often really a bucket of schemas that collectively serve as the canonical definition - it is not one very large XML schema.

Using a Canonical Schema or set of schemas is double lossy. For example, the transformation from the intrinsic schema for the source service to the canonical schema, and then the second transformation from the canonical schema to the intrinsic schema for the destination service may lead to loss of information twice. In this context, if this loss is not acceptable, services may optimize by using the reader-makes-right approach outlined earlier.

Whilst canonical schemas can be very effective, often the real challenge is to get the stake-holders to agree on the attributes and the schema.

### Entity Aggregation

In designing an entity aggregation service, keep in mind that the motivation is for it to act as a harmonizing source of business concepts, providing a holistic, shared view for business entities and their relationships with other entities.

An entity aggregation service needs to enable location transparency— service consumers as well as service compositions should not need to be aware of the specifics of entity ownership. To meet the needs of service consumers and of the composite workflows and business processes an entity aggregation service needs to store and keep current metadata about the entities and their schemas, as well as, information pertinent to entity ownership.

There are two commonly used approaches to realizing entity aggregation services: the Straight-through-processing approach and the Replication approach.

#### *Straight-Through Processing*

In this approach data is retrieved and harmonized in near real-time. The entity aggregation service will attempt to access entities and present the shared view in a pass-through manner. This has the obvious advantages of decreased latency, and increased currency of the entities, whilst imposing constraints such as that the entity aggregation service needs to have real-time connectivity to the various services.

#### *Replication*

In this approach, entities are replicated and then cached by the entity aggregation service. This approach is often applied when real-time connectivity to services is lacking, or when performance is the primary motivation driving the aggregation of entities.

In the real world, often there is a hybrid approach, where a subset of the entities (the most commonly used for instance) are replicated and cached locally, and the remaining information may be accessed in a straight-through processing manner.

## Master Data Management

There are two basic steps to creating master data: cleaning and standardizing the data, and matching data from the various sources in order to consolidate duplicates. Cleaning the data and transforming it into the master data model involves the normalization of data formats, the replacement of missing values and value standardization etc. Matching data from the various sources to eliminate duplicates is the critical step in creating master data. False matches may lead to the loss of data and matches that are lost or otherwise overlooked may reduce the inherent value of maintaining a master list.

In order to keep the master data current the data needs to be actively managed during the course of its lifetime. There are a number of techniques for managing and consuming master data. Some commonly used approaches include the Single Master approach, the Single Master/Multiple Replicas approach, and the Multiple Master approach.

### *The Single Master Approach*

With the single master approach, there is only one instance of the master data and changes are made directly to this single master instance. Services that require master data need to be modified to use this single master copy. Re-designing a large number of services to use a new source of master data increases both complexity and cost and whilst this approach guarantees master data consistency, it may not be a viable approach in the real world.

### *Single Master, Multiple Replicas Approach*

With the single master, multiple replicas approach, master data changes are made only in the single master instance, but all changes to the master data are propagated to other services, which then locally caches this master data. This is a pragmatic optimization of the previous model; however, services will still need to be modified to prevent end users from trying to update the local cache of the master data.

### *Multiple Master Approach*

With the multiple master approach services are also allowed to change their local cache of the master data and these changes are then replicated back to the master, where they are then merged into the master data. This then triggers a second replication of these master data changes to the rest of the services. This approach can be the least intrusive as far as the services are concerned; however, it does pose some challenges with respect to the consistency of the master data. For example, two of the services may decide to simultaneously change an employee's tax exemptions to different values and the system needs to somehow reconcile this update conflict.

## Conclusion

In order for business services to be able to communicate with each other and to enable meaningful service composition they need to have a shared view of common business entities. Schema rationalization, entity aggregation and master data management are key foundational capabilities that enable business services to be able to communicate with each other.

