**Case Study**

RailCo's original goals were to upgrade its automation systems so that it could remain competitive and continue its business relationship with its primary client, TLS. RailCo had lost TLS as a customer when a competitor managed to provide air brake parts at a lower price while also interfacing with TLS's B2B system. RailCo rushed to catch up, producing a pair of Web services designed only for use with the TLS system. This allowed RailCo to regain its position as a TLS vendor.

These two initial Web services were

* Invoice Submission Service
* Order Fulfillment Service

(Another service was added later to interact with the TLS Notification Service.)

However, even though RailCo had successfully reconnected with TLS, it had lost its exclusive relationship. It now found itself in a position where it had to bid against an aggressive competitor for every purchase order issued; therefore, it was still losing revenue.

The only way RailCo could avoid significant downsizing was by finding new clients. To accomplish this, RailCo needed to continue pursuing the online vendor marketplace with other transit companies providing B2B solutions. It then became evident that RailCo's current set of Web services was insufficient for this purpose. Because they had been designed solely for use with TLS, they were not useful for interacting with other customers that dictated different business and transaction requirements.

RailCo was then faced with an important decisioneither develop a custom set of services for each new client or start from scratch and build a standardized set of services generic enough to facilitate multiple clients. It chose the latter option and decided that the best way to achieve this goal was to overhaul its existing environment in favor of an SOA.

RailCo's two primary business processes are:

* Order Fulfillment (accepting and processing purchase orders from a client) and
* Invoice Submission (sending an invoice to a client).

RailCo proceeded with a service-oriented analysis that decomposed its business process logic into a series of service candidates. This revealed the need for the following potential services and service layers:

* A business service layer consisting of two task-centric business services.
* An application service layer comprised of four application services.

RailCo did not have the technology or the budget to invest in middleware capable of providing orchestration. It therefore chose not to pursue centralizing its business logic in an orchestration service layer.

Instead, it was decided to represent each business process with a task-centric business service that would act as a controller for a layer of application services. The following services were modeled and then designed:

* Invoice Processing Service (task-centric)
* PO Processing Service (task-centric)
* Legacy System Service (application)
* Polling Notification Service (application)
* Transform Service (application)
* Metadata Checking Service (application)

Reusability and extensibility in particular were emphasized during the design of its application services. RailCo wanted its initial SOA to consist of services that supported both of its current business processes, while being sufficiently extensible to accommodate future requirements without too much impact.

To realize the Invoice Submission Process, RailCo was able to compose these services into a two-level hierarchy, where the parent Invoice Processing Service coordinates the execution of all application services (Figure A.1).

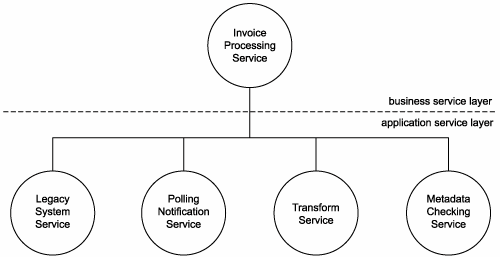


Figure A.1. RailCo's service composition that automates its Invoice Submission Process.

The Order Fulfillment Process can now be automated via the PO Processing Service, which reuses two of the same application services used by the Invoice Submission Process (Figure A.2).

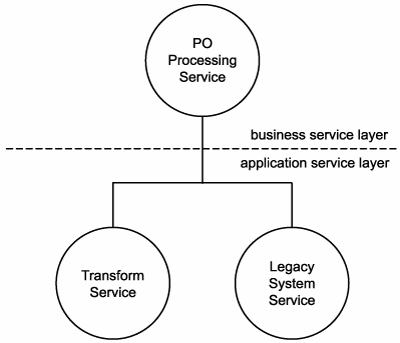


Figure A.2. The Order Fulfillment Process is automated by a PO Processing Service that composes two reusable application services.

In the face of some bad news involving the departure of the .NET consultants responsible for delivering their original Web services, RailCo was able to put internal resources to good use. Subsequent to a training effort, the new SOA was created as a J2EE solution.

RailCo has fulfilled its original goals by producing an SOA that supports two service-oriented solutions. RailCo can now continue its online transactions with TLS while confidently seeking new customers. Additional clients introducing new requirements can be accommodated with minimal impact. Its standardized application service layer will likely continue to offer reusable functionality to accommodate the fulfillment of new requirements. And any functional gaps will likely be addressed by extending the services without significantly disrupting existing implementations.

Further, should RailCo decide to replace its task-centric business services with an orchestration service layer in the future, the abstraction established by the existing application service layer will protect the application services from having to undergo redevelopment.

Upon completing this project, RailCo discovers a side benefit to its new solution environment. By having established the Legacy System Service (which is essentially a wrapper service for its accounting system) as part of its application service layer, it has opened up a generic endpoint that can facilitate integration.

This provides the potential for RailCo to enable interoperability between its accounting system and its contact management application (first introduced in Chapter 2). By allowing these two environments to share data, RailCo can more efficiently take on and service new clients with coordinated contact and financial history profiles.