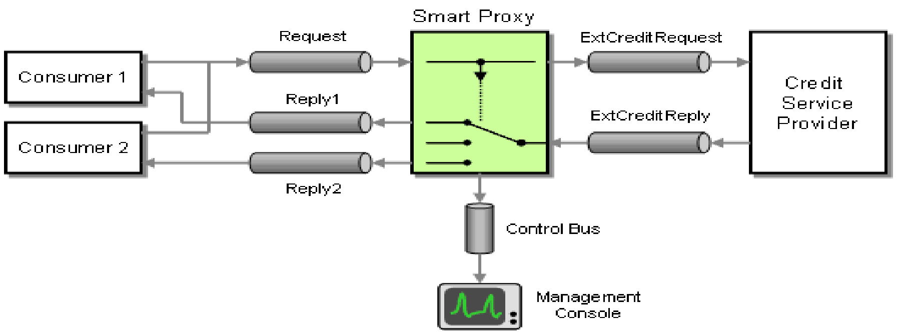
TESTING AND MONITORING

Monitoring the correct execution of messages is a critical operations and support function. The *Message Store* can provide us with some important business metrics such as the average time to fulfill an order. However, we may need for more detailed information for the successful operation of an integration solution. Let's assume we enhance our solution to access an external credit agency to better assess our customer's credit standing. Even if we show no outstanding payments we may want to decline a customer's order if the customer's credit ranking is particularly poor. This is especially useful for new customers. Because the service is provided by an outside provider we are being charged for its use. To verify the provider's invoice we want to track our actual usage and reconcile the two reports. We cannot simply go by the number of orders because the business logic may not request an external credit check for long-standing customers. Also, we have a Quality of Service (QoS) with the external provider. If the response time exceeds a specified time, we do not have to pay for the request.

To make sure we are being billed correctly we want to track the number of requests we made and the time it takes for the associated response to arrive. We have to be able to deal with two specific situations.

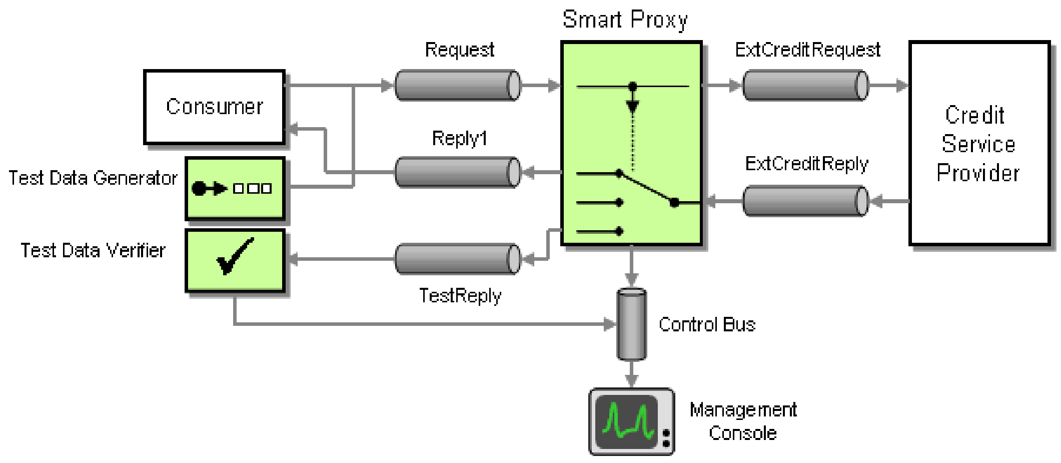
The external service can process more than one request at a time, so we need to be able to match up request and reply messages. Second, since we treat the external service as a shared service inside our enterprise we want to allow the service consumer to specify a *Return Address*, the channel where the service should send the reply message. It could be difficult to match request and reply messages if we don't know which channel the reply message is on.

Once again the *Smart Proxy* is the answer. We insert the *Smart Proxy* between any service consumer and the external service. We replace the *Return Address* specified by the service consumer with a fixed reply channel. We store the original *Return Address* inside the *Smart Proxy* so that it can forward the reply message to the channel specified by the consumer. The *Smart Proxy* also measures the time elapsed between request and reply message from the external service. The *Smart Proxy* publishes this data to the *Control Bus*. The *Control Bus* is connected to a management console that collects metrics from many different components.



Inserting a Smart Proxy to Track Response Times

We also want to make sure that the external credit service is working correctly. The *Smart Proxy* can report cases where no reply message is received within a specified time-out period to the management console. Much harder to detect are cases where the external service returns a reply message but the results in the message are incorrect. For example is the external service malfunctions and returns a credit score of zero for every customer we would end up denying every order. There are two mechanisms that can help us protect against such a scenario. First, we can periodically inject a *Test Message* into the request stream. This *Test Message* requests the score for a specific person so that the result is known. We can then use a *Test Data Verifier* to not only check the fact that a reply was received but also the accuracy of the message content. Because the *Smart Proxy* supports Reply Addresses the *Test Data Generator* can specify a special reply channel to separate test replies from regular replies (see picture).



Inserting Test Messages to Verify Accurate Results

Another effective strategy to detect malfunctioning services that return messages in a valid format but with bad data is to take a statistical sample. For example, we may expect to decline an average of less than one in 10 orders due to the customer’s poor standing. If we decline more than 5 orders in a row this may be an indication that an external service or some business logic is malfunctioning. The management console could e-mail the five orders to an administrator who can then take a quick look at the data to verify whether the rejections were justified.

Summary

We have walked through a fairly extensive integration scenario using different integration strategies such as *File Transfer*, *Shared Database* and asynchronous *Messaging*. We routed, split and aggregated messages. We also added functions to monitor the correct operation of the solution. While the requirements for this example were admittedly simplified the issues and design trade-offs we had to consider are very real. The solution diagrams and descriptions highlight how we can describe a solution in a vendor-and technology-neutral language that is much more accurate than a high-level sequence diagram.

Source:
http://www.enterpriseintegrationpatterns.com/patterns/messaging/Chapter1.html