INTRODUCTION TO MESSAGE TRANSFORMATION

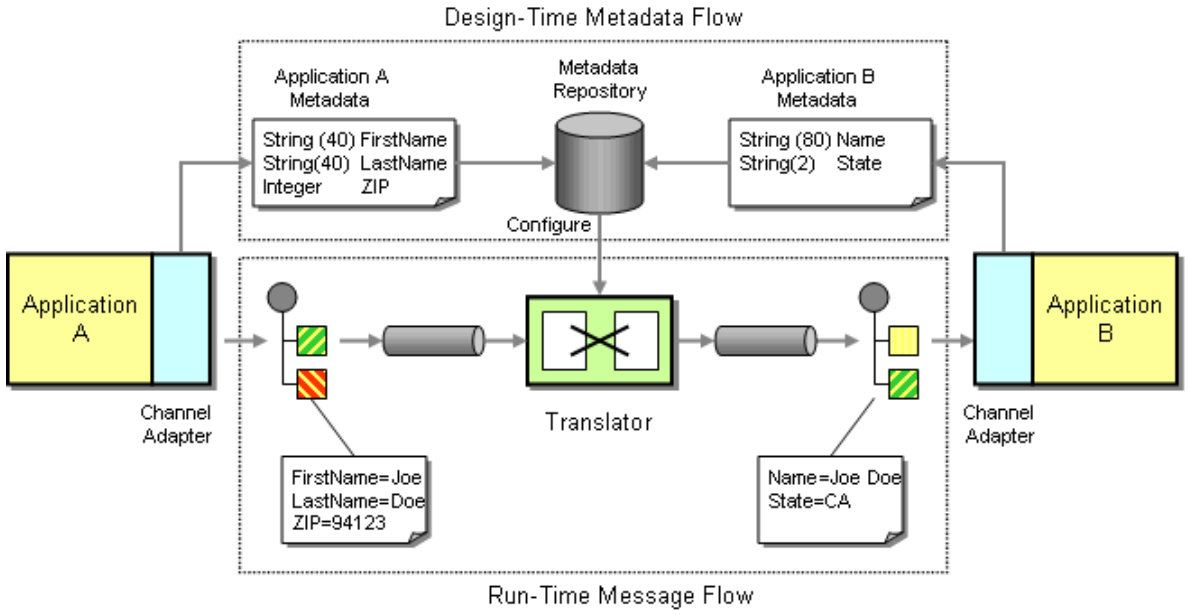
As described in the *Message Translator*, applications that need to be integrated by a messaging system rarely agree on a common data format. For example, an accounting system is going to have a different notion of a Customer object than a customer relationship management system. On top of that, one system may persist data in a relational model, while another application uses flat files or XML documents. Integrating existing applications often times means that we do not have the liberty of modifying the applications to work more easily with other systems. Rather, the integration solution has to accommodate and resolve the differences between the varying systems. The *Message Translator* pattern offers a general solution to such differences in data formats. This chapter explores specific variants of the *Message Translator*.

Most messaging systems place specific requirements on the format and contents of a message header. We wrap message payload data into an *Envelope Wrapper* that is compliant with the requirements of the messaging infrastructure.
Multiple *Envelope Wrapper*s can be combined if a message is passed through across different messaging infrastructures.

A *Content Enricher* is needed if the target system requires data fields that the originating system cannot supply. It has the ability to look up missing information or compute it from the available data. The *Content Filter* does the opposite -- it removes unwanted data from a message. The *Claim Check* also removes data from a message but stores it for later retrieval. The *Normalizer* translates messages arriving in many different formats into a common format.

Message transformation is a deep topic in integration. *Message Channel*s and *Message Router*s can remove basic dependencies between applications by eliminating the need for one application to be aware of the other's location. One application can send a message to a *Message Channel* and worry about what application will consume it. However, message formats impose another set of dependency. If one application has to format messages in another application's data format, the decoupling in form of the *Message Channel* is somewhat of an illusion. Any change to the receiving application or the switch from one receiving application to another still requires a change to the sending application. *Message Translator*s help remove this dependency. Due to the importance of message formats and the transformation between them, we can view any integration solution as two parallel systems. One deals with actual message data, the other one with metadata, the data that describes message data. Many of the patterns used in creation of the message flow can also be used to manage metadata.

For example, a *Channel Adapter* can not only move messages in an out of a system, but it can also extract metadata from external applications and load it into a central metadata repository. Using this repository the integration developers can define transformations between the application metadata and the *Canonical Data Model*.



Metadata Integration

For example, the picture above depicts the integration between two applications that need to exchange customer information. Each system has a slightly different definition of customer data. Application A stores first and last name in two separate fields whereas Application B stores it in one field. Likewise, Application A stores the customer's ZIP code and not the state while Application B stores only the state code.

Messages flowing from Application A to Application B have to undergo a transformation so that Application B can receive data in the required format. Creating the transformation is much simplified if the *Channel Adapter*s can also extract metadata, e.g. data describing the message format. This metadata can then be loaded into a repository, greatly simplifying the configuration and validation of the *Message Translator*. The metadata can be stored in a variety of formats. A common format used for XML messages are XSD's -- XML Schema Definitions. Other EAI tools implement proprietary metadata formats, but allow administrators to import and export of metadata into different formats.

Many of the principles incorporated in these patterns are applicable outside of messaging integration. For example, *File Transfer* has to perform transformation functions between systems. Likewise, *Remote Procedure Invocation* has to make requests in the data format specified by the service that is to be called even if the application's internal format is different.

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