MESSAGE CHANNEL THEMES

Deciding to use a *Message Channel* is the simple part; if an application has data to transmit or data it wishes to receive, it will have to use a channel. The challenge is knowing what channels your applications will need and what to use them for.

Fixed set of channels — One theme in this chapter is that the set of *Message Channel*s available to an application tends to be static. When designing an application, a developer has to know where to put what types of data to share that data with other applications, and likewise where to look for what types of data coming from other applications. These paths of communication cannot be dynamically created and discovered at runtime; they need to be agreed upon at design time so that the application knows where its data is coming from and where the data is going to. (While it is true that most channels must be staticly defined, there are exceptions to this theme, cases where dynamic channels are practical and useful. One exception is the reply channel in *Request-Reply*. The requestor can create or obtain a new channel the replier knows nothing about, specify it as the *Return Address* of a request message, and then the replier can make use of it. Another exception is messaging system implementations that support hierarchical channels.

A receiver can subscribe to a parent in the hierarchy, then a sender can publish to a new child channel the receiver knows nothing about, and the subscriber will still receive the message. These relatively unusual cases notwithstanding, channels are usually defined at deployment-time and applications are designed around a known set of channels.)

Determining the set of channels — A related issue is: Who decides what *Message Channel*s are available, the messaging system or the applications? That is to say: Does the messaging system define certain channels and require the applications to make due with those? Or do the applications determine what channels they need and require the messaging system to provide them? There is no simple answer; designing the needed set of channels is iterative. First the applications determine the channels the messaging system will need to provide. Subsequent applications will try to design their communication around the channels that are available, but when this is not practical, they will require that additional channels be added. When a set of applications already use a certain set of channels, and new applications wish to join in, they too will use the existing set of channels. When existing applications add new functionality, they may require new channels.

Unidirectional channels — Another common source of confusion is whether a *Message Channel* is unidirectional or bidirectional. Technically, it's neither; a channel is more like a bucket that some applications add data to and other applications take data from (albeit a bucket that is distributed across multiple computers in some coordinated fashion). But because the data is in messages that travel from one application to another, that gives the channel direction, making it unidirectional. If a channel were bidirectional, that would mean that an application would both send messages to and receive messages from the same channel, which—while technically possible—makes little sense because the application would tend to keep consuming its own messages, the messages it's supposed to be sending to other applications. So for all practical purposes, channels are unidirectional. As a consequence, for two applications to have a two-way conversation, they will need two channels, one in each direction.

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