LOOSE COUPLING OF INTERACTION

There are a couple of major problems with this integration attempt. One of the strengths of the TCP/IP protocol is its wide support so that we can connect to pretty much any computer connected to the network regardless of the operating system or programming language it uses. However, the platform independence works only for very simple messages: byte streams. In order to convert our data into a byte stream we used the BitConverter class. This class converts any data type into a byte array, using the internal memory representation of the data type. The catch is that the internal representation of an integer number varies with computer systems. For example, .NET uses a 32 bit integer while other systems may use a 64 bit representation. Our example transfers 4 bytes across the network to represent a 32 bit integer number. A system using 64 bits would be inclined to read 8 bytes off the network and would end up interpreting the whole message (including the customer name) as a single number.

Also, some computer systems store their numbers in big-endian format while others store them in little-endian format. A big-endian format stores numbers starting with the highest byte first while little-endian systems store the lowest byte first.

PCs operate on a little-endian scheme so that the code passes the following 4 bytes across the network:

232 3 0 0

232 + 3 * 2^8 equals 1000. A system that uses big-endian numbers would consider this message to mean 232* 2^24 + 3 * 2^16 = 3,892,510,720. Joe will be a very rich man! So this approach works only under the assumption that all connected computers represent numbers in the same internal format.

The second problem with this simple approach is that we specify the location of the remote machine (in our case www.eaipatterns.com). The Dynamic Naming Service (DNS) gives us one level of indirection between the domain name and the IP address, but what if we want to move the function to a different computer on a different domain? What if the machine fails and we have to setup another machine? What if we want to send the information to more than one machine? For each scenario we would have to change the code. If we use a lot of remote functions this could become very tedious. So we should find a way to make our communication independent from a specific machine on the network.

Our simple TCP/IP example also establishes temporal dependencies between the two machines. TCP/IP is a *connection-oriented* protocol.

Before any data can be transferred, a connection has to be established first. Establishing a TCP connection involves IP packets traveling back and forth between sender and receiver. This requires that both machines and the network are all available at the same time. If any of the three pieces is malfunctioning or not available due to high load, the data cannot be sent.

Lastly, the simple communication also relies on a very strict data format. We are sending 4 bytes of amount data and then a sequence of characters that define the customer's account. If we want to insert a third parameter, e.g. the name of the currency, we would have to modify both sender and receiver to use the new data format.

Tightly Coupled Interaction

In summary, our minimalist integration solution is fast and cheap, but it results in a very brittle solution because the two participating parties make the following assumptions about each other:

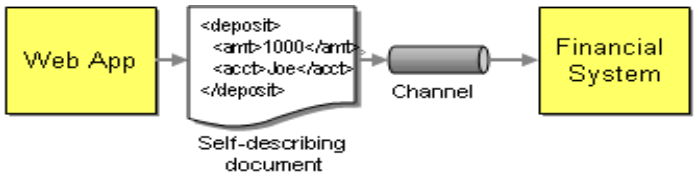
- Platform Technology – internal representations of numbers and objects
- Location – hard-coded machine addresses
- Time – all components have to be available at the same time

- Data Format – the list of parameters and their types must match

As we stated in the beginning, coupling is a measure of how many assumptions parties make about each other when they communicate. Our simple solution requires the parties to make a lot of assumptions. Therefore, this solution is tightly coupled.

In order to make the solution more loosely coupled we can try to remove these dependencies one by one. We should use a standard data format that is self-describing and platform independent, such as XML. Instead of sending information directly to a specific machine we should send it to an addressable "channel". A channel is a logical address that both sender and receiver can agree on the same channel without being aware of each other's identity. Using channels resolves the location-dependency, but still requires all components to be available at the same time if the channel is implemented using a connection-oriented protocol.. In order to remove this temporal dependency we can enhance the channel to queue up sent requests until the network and the receiving system are ready. To support queuing of requests inside the channel, we need wrap data into self-contained messages so that the channel knows how much data to buffer and deliver at any one time. Lastly, the two systems still depend on a common data format. We can remove this dependency by allowing for data format transformations inside the channel.

If the format of one system changes we only have to change the transformer and not the other participating systems. This is particularly useful if many applications send data to the same channel.



Loosely Coupled Interaction

Mechanisms such as a common data format, queuing channels, and transformers help turn a tightly coupled solution into a loosely coupled solution. The sender no longer has to depend on the receiver's internal data format not its location. It does not even have to pay attention to whether the other computer is ready to accept requests or not. Removing these dependencies between the systems makes the overall solution more tolerant to change, the key benefit of loose coupling. The main drawback of the loosely coupled approach is the additional complexity. This is no longer a 10-lines-of-code solution! Therefore, we use a message-oriented middleware infrastructure that provides these services for us. This infrastructure makes exchanging data in a loosely coupled way almost as easy as the example we started with. The next section describes the components that make up such a middleware solution.

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