INTRODUCTION TO INTEGRATION STYLES

Enterprise integration is the task of making separate applications work together to produce a unified set of functionality. Some applications may be custom developed in-house while others are bought from third-party vendors. The applications probably run on multiple computers, which may represent multiple platforms, and may be geographically dispersed. Some of the applications may be run outside of the enterprise by business partners or customers. Some applications may need to be integrated even though they were not designed for integration and cannot be changed. These issues and others like them are what make application integration difficult. This chapter will explore the options available for application integration.

Application Integration Criteria

What makes for good application integration? If integration needs were always the same, there would only be one integration style. Yet like any complex technological effort, application integration involves a range of considerations and consequences that should be taken into account for any integration opportunity.

The first criterion is **application integration** itself. If you can develop a single, stand-alone application that doesn't need to collaborate with any other applications, you can avoid the whole integration issue entirely. Realistically, though, even a simple enterprise has multiple applications, applications that need to work together to provide a unified experience for the enterprise's employees, partners, and customers.

The other main decision criteria are:

Application coupling — Even integrated applications should minimize their dependencies on each other so that each can evolve without causing problems for the others. Tightly coupled applications make numerous assumptions about how the other applications work; when the applications change and break those assumptions, the integration breaks. The interface for integrating applications should be specific enough to implement useful functionality, but general enough to allow that implementation to change as needed.

Integration simplicity — When integrating an application into an enterprise, developers should strive to minimize changing the application and minimize the amount of integration code needed. Yet changes and new code will usually be necessary to provide good integration functionality, and the approaches with the

least impact on the application may not provide the best integration into the enterprise.

Integration technology — Different integration techniques require varying amounts of specialized software and hardware. These special tools can be expensive, can lead to vendor lock-in, and increase the burden on developers to understand how to use the tools to integrate applications.

Data format — Integrated applications must agree on the format of the data they exchange, or must have an intermediate traslator to unify applications that insist on different data formats. A related issue is **data format evolution and extensibility**—how the format can change over time and how that will affect the applications.

Data timeliness — Integration should minimize the length of time between when one application decides to share some data and other applications have that data. Data should be exchanged frequently in small chunks, rather than waiting to exchange a large set of unrelated items. Applications should be informed as soon as shared data is ready for consumption. Latency in data sharing has to be factored into the integration design; the longer sharing can take, the more opportunity for shared data to become stale, and the more complex integration becomes.

Data or functionality — Integrated applications may not want to simply share data, they may wish to share functionality such that each application can invoke the functionality in the others. Invoking functionality remotely can be difficult to achieve, and even though it may seem the same as invoking local functionality, it works quite differently, with significant consequences for how well the integration works.

Asynchronicity — Computer processing is typically synchronous, such that a procedure waits while its subprocedure executes. It's a given that the subprocedure is available when the procedure wants to invoke it. However, a procedure may not want to wait for the subprocedure to execute; it may want to invoke the subprocedure asynchronously, starting the subprocedure but then letting it execute in the background. This is especially true of integrated applications, where the remote application may not be running or the network may be unavailable—the source application may wish to simply make shared data available or log a request for a subprocedure call, but then go on to other work confident that the remote application will act sometime later.

As you can see, there are several different criteria that must be considered when choosing and designing an integration approach. The question then becomes: Which integration approaches best address which of these criteria?

Application Integration Options

There's more than one approach for integrating applications. Each approach addresses some of the integration criteria better than others. The various approaches can be summed up in four main integration styles:

File Transfer — Have each application produce files of shared data for others to consume, and consume files that others have produced.

Shared Database — Have the applications store the data they wish to share in a common database.

Remote Procedure Invocation — Have each application expose some of its procedures so that they can be invoked remotely, and have applications invoke those to run behavior and exchange data.

Messaging — Have each application connect to a common messaging system, and exchange data and invoke behavior using messages.

Each of the patterns has the same problem statement—the need to integrate applications—and very similar contexts.

What differentiates them is different forces searching for a more elegant solution. Each pattern builds on the last, looking for a more sophisticated approach to address the shortcomings of its predecessors. Thus the pattern order reflects an increasing order of sophistication.

The trick is not to choose the one style to use always, but to choose the best style for a particular integration opportunity. Each style has its advantages and disadvantages. Two applications may integrate using multiple styles such that each point of integration takes advantage of the style that suits it best. Likewise, an application may use different styles to integrate with different applications, so as to choose the style that works best for the other application. Some integration approaches can best be viewed as a hybrid of multiple styles. An integration product or EAI middleware may employ a combination of styles, all of which are effectively hidden in the product's implementation.

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