Options for integration strategies

1) Externally Configured Jobs, Externally Run
2) Externally Configured Jobs, Internally Run
3) Internally Configured Jobs, Internally Run

Let us start with **option one**; integration jobs that are configured in an external tool, and then scheduled and triggered within that tool itself. This is the strategy that is typical in a large organization. Usually a third party commercial tool, such as Informatica, is purchased and a specialized employee is hired to create and maintain all of the organization's integration jobs. The systems in the organization are integrated at the data level.

Advantages: This option's advantage is that the specialized external system is usually very good at what it does. Adapters and plugins are typically available for every type of input and output that is necessary for integration. Commercial tools also provide support for the tool. Reporting and statistics of job execution is typically included in the tool as well.

Disadvantages: Since the tool is run outside of the organization's systems, an endpoint needs to be created within existing systems. That is, a web service, or a message queue is typically added to facilitate the external communication. Cost is also a concern, as the external commercial system is usually quite expensive. Being externally run, this tool would have to be configured and watched separately from our system.

Conclusion: An externally run external system will present challenges of integration with our system, and will cost us more than the other two options.

Option two is using an external system to configure the integration scripts, and then running the scripts within our system. This strategy would allow us to use the tools to create and maintain the integration scripts. The tools that were evaluated for the option were Pentaho, Mule, and Smooks.

Advantages: Penthao and Mule both have visual tools that are available for the integration developer to use. This tool helps visualize the steps that are in the integration job. All the tools here are open source and do not cost us anything to use. Pentaho has more adapters for input, specifically an Excel spreadsheet input adapter. If the data is first transformed to a delimited flat file, then all three tools can be used. The scripts that are produced by the tools are in XML format, and can be embedded into our existing system.

Disadvantages: The major disadvantage of this strategy is that it is still difficult to provide the level of integration with our system that we are wanting. It is closer than the first option, as we can integrate without the need to create specific endpoint hooks. However, these tools don't make it easy to do this. Mule can be run within our Spring context, thus it can refer to our bean IDs. Mule does not have any adapter for Excel files, which is where a large majority of our data currently comes from. Smooks is great for serializing CSV directly into a Java object. Unfortunately, the data coming in needs to first be transformed. So a combination of Mule and Smooks could work. Pentaho's main disadvantage is that it is difficult to use our own services directly within the tool.

Conclusion: Using an externally created script could provide speed in implementation, since the input and output adapters are already defined. Also, a GUI for creation of the script could be useful. The ability to use our own services for data quality and persistance is advantageous. This strategy provides a good blend of external components and internal system integration.

Option three is using internally configured jobs run internally on the system. We can either enhance our existing Batch sub-system to do integration jobs, or we could leverage the Spring Batch framework. Either way, with this strategy all of the jobs would be configured, managed, and maintained within the current code base. The adapters for reading and parsing the input would have to be created by us, or we would need to integrate with open source libraries (such as POI for Excel reading).

Advantages: We have control of the entire framework, so it is very easy to integrate with our existing services. Configuration of the jobs can be as fine grain as necessary. Also, no extra fluff would be added, since we would only be building what we would need.

Disadvantages: We would need to either spend the time enhancing the existing batch framework, or investigate integration with Spring Batch. Spring Batch requires a specific database structure (which is similar to ours) in order to be implemented. One advantage of Spring Batch is the ability to restart failed jobs or retry failed steps within a job. There would need to be custom transformations written in code for each client. This is similar to the solution that is in the current system, and the management and maintainence of it has proven to be challenging.

Conclusion: The creation of our own framework from scratch, while intriguing, doesn't seem to be a wise use of our resources. Using the Spring Batch Framework would reduce this workload, but would require changing our existing Batch System code. Either solution would involve using other libraries to read in the source data.

Recommendation: I would recommend the use of Pentaho to create the ETL scripts. Since currently a number of different sources (Excel macros, stored proceedures) are used to do this, switching to one tool centralizes the transformation of the data. The question then becomes how do we get the data into the system? Pentaho is weak in using our existing services, but it does support user plugins. I would recommend creating a plugin that can get a reference to a Spring bean, and call a configured method on the bean, sending in the data.