Contents

1 Parallel processing

Parallel processing (multi-threading) is a way for a single program to facilitate multiple processes for a set of tasks. This is especially useful in system integration, since the libraries, web-API's etc. Possibly could involve time-consuming calls. Slow or time consuming integration's could possibly be detrimental to an application, were it implemented in a blocking manner.

To corner this one might segregate the slow call in a single process. Another relevant way to tackle this issue would to be with asynchronous programming.

2 Decoupling

Decoupling is the act of isolating functionality into secluded portions of a codebase. This is relevant in the context of maintainability and test-ability. In a system high coupling refers to the amount of dependencies is exposed across an application. If you were to integrate a library into your application.

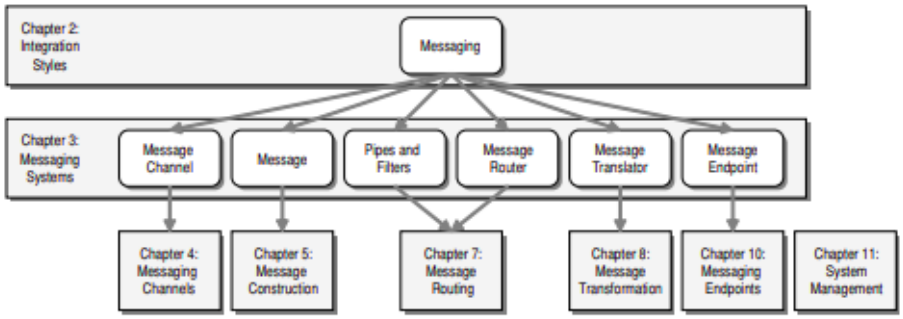
A practical example would be of a high cohesion implementation would be to import the library into a program. This would make it cumbersome to swap the library out later. To solve this issue in a loosely coupled manner you could implement the library in a service and hand this service to the classes that needs the functionality through dependency injection. This would solve the issue of changing all the files that implements the library if you were to swap it out and also abstracting the libraries API.

3 Messaging

Messaging is probably one of the most relevant terms of the ones I have chosen. Messaging is a general term and could probably be used in most contexts when

it comes to transmission of data. But messaging in the context of more sophisticated applications, would most likely refer to specific type of technologies that utilizes asynchronous process to process communication.

Figure 1: This is a diagram illustrating a more sophisticated implementation of a message broker



These technologies is highly relevant for large scale applications, and also for system integration - since every process could be observed as being an integration within the system because they indirectly communicates with each other. Using such technologies often provokes good code standards and highly scalable systems. However it introduces complexity.

4 Queuing

Queuing is a highly useful data structure which in it's most basic form, functions like a literal queue. The first one that gets in, is also the first one that gets out. Web-servers request interface is implemented over this data structure, which is why multiple requests can be resolved even though the web-servers resources implementations might be resolved in a completely synchronous matter.

Message brokers like RabbitMQ is highly dependent on this data structure, since it has to resolve the grouped messages in some kind of logical order. More often than not this would be implemented over a priority queue, which is a more flexible queue where elements has tags denoting their priority. This makes it RabbitMQ capable of putting the most relevant messages to the front of it's queue. The term queuing is the act of putting things in a queue, but in the context of system integration's it would possibly refer to the utilization of sending messages with a message broker.

5 Piping

Piping is a paradigm that revolves around sending data between processes in a continuous non-blocking manner. A single pipe is responsible for a one way communication between two processes. (however bidirectional communication can be achieved by creating two pipes) There is two ends of a pipe, a read and a write end. In the context of a UNIX system, there is a pipe() function, that takes an array of the file descriptors and opens a channel for the communication.

Is this especially useful for system integration? Probably not. But it is fundamental to continuous integration pipelines which is a tool that would be responsible for reassuring that your integration's are running as expected. Pipelines is a sequence of pipes that operates in an asynchronous manner. That is highly efficient since each process in the pipeline can work independent of one another.

6 Routing

Routing is a networking practice that is responsible for sending specific requests messages to the correct destinations. In more sophisticated applications, one would have multiple services possibly to reduce load or traffic. In the context of system integration, one could consider to expose a service responsible for implementing a web-API and exposing a interface with only the relevant functionality for the applications business-logic. Routing is a general term and if you speak of routing in the context of a generalized monolithic back-end, one might immediately think of technologies like a reverse-proxy. In However routing is also at the core of most messaging technologies. A messaging router in the context of a message broker is the software that directs the messages to their correct channels. Hereby abstracting the implementations from the consumers.

> [1] In a large enterprise with numerous applications and channels to connect them, a message may have to go through several channels to reach its final destination. The route a message must follow may be so complex that the original sender does not know what channel will get the message to the final receiver.

7 Transformation

Data might need some transformation in order to be concise for the different parts of the application it might come across. It could be some data that has too much overhead for a specific route, or it could be an entirely different format.

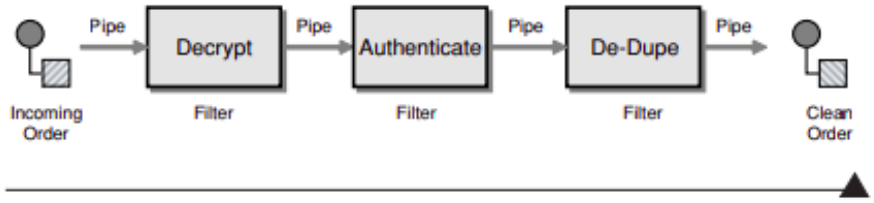
> [1]Various applications may not agree on the format for the same conceptual data; the sender formats the message one way, but the receiver expects it to be formatted another way. To reconcile this, the message must go through an intermediate filter,

As previously noted in the filter, this step could possibly happen in a message filter within an application utilizing a message broker. Or it could happen on the web-server before the actual resource implementations.

8 Filtering

A filter is usually something that sits in front of something in order to validate whether or not the operation should be permitted, to decrypt the message or perhaps to transform the data into something more tailored for the application.

Figure 2: [1]Use the Pipes and Filters architectural style to divide a larger processing task into a sequence of smaller, independent processing steps (filters) that are connected by channels (pipes).



As shown on the diagram, it is common to chain multiple filters with pipes or maybe in a pipeline if there is some extensive IO operations included.

> [1] Each filter exposes a very simple interface: It receives messages on the inbound pipe, processes the message, and publishes the results to the outbound pipe. The pipe connects one filter to the next, sending output messages from one filter to the next

9 Transactions

A transaction is a series of steps that either all resolve and change some data, or some fail and performs a "rollback" and change all the data back to it's original state.

A practical use-case where transactions would be necessary, could be if you were to store some data reflecting another system that you integrate in your own application.

A idiomatic implementation would be to persist the data to your own application, then attempt to persist it to your integration. Should one of them fail, delete the data from the other. This would be relevant in the context of data integration.

10 Caching

In the context of computational expensive functionality of a systems integrated technologies. One might be able to utilize caching in order to improve the performance of the application.

A practical example would be if an application utilized a weather API that had a expensive call to get a extensive weather prognosis. One might encapsulate the call in a time-based job like Cron, and store the data in a cache, one could utilize an in memory database for this, to efficiently be able to extract the data wherever needed in the application.

References

[1] Gregor Hohpe and Bobby Woolf. Enterprise integration patterns designing, building, and deploying messaging solutions.