Pipe and Filter Architectural Style

Group Name: Opti
Members: Duaa Abdelgadir (dyaabdel), Rithu Chandrasekar (r7chandr), Saif Mahamood(smahamoo), Sheethala Swaminathan (s2swamin)

What is the Pipe and Filter style?

The Pipe and Filter is an architectural design pattern that allows for stream/asynchronous processing. In this pattern, there are many components, which are referred to as filters, and connectors between the filters that are called pipes. Each filter is responsible for applying a function to the given data; this is known as filtering. Filters can work asynchronously. The final output is given to the consumer, known as a sink.

Most applicable to specific kinds of problems?

The Pipe and Filter style is best for large processes that can be broken down into multiple steps. This way, each filter will be responsible for one of the steps and they can run simultaneously to produce all the data needed for the final outcome. A common example is the UNIX pipe command that pipes output from one function as input to another function.

Engender specific kinds of change resilience / other advantages?

If new steps are introduced into a process, it would be possible to create separate filters for those new steps. Therefore, it would be easy to grow the pipe and filter architecture. It also makes it easy to reuse filters that perform a generic action on various data sets. Also, a major advantage is that it promotes concurrency since different filters can work at the same time if they do not depend on each other. Another advantage is that filters don't share their state and are unaware of what other filters are doing. They only communicate through their input/output channels which reduces coupling between components.

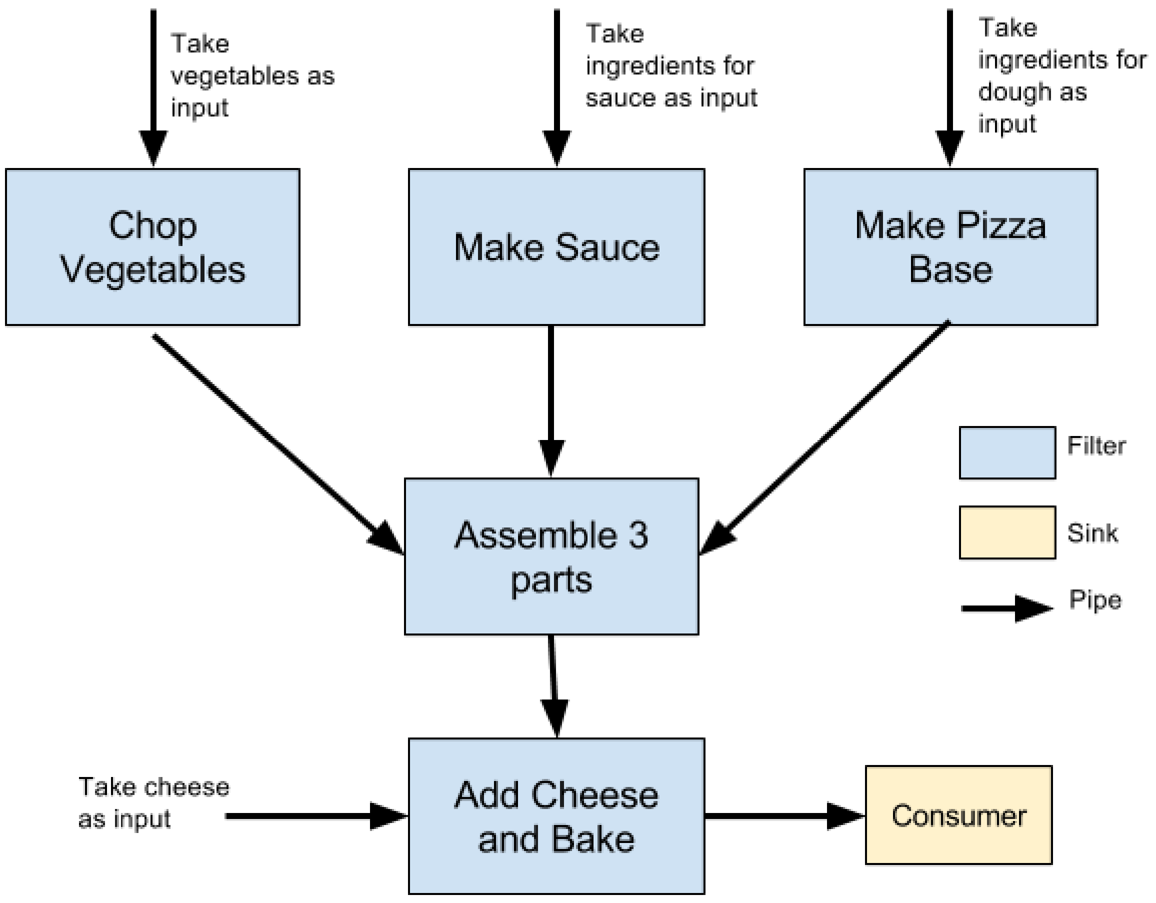
Have any specific negative behaviours?

If there needs to be more interaction between functions (the filters), this would not be a good architecture design since filters only communicate between each other for input/output. Another drawback is that since outputs of one filter are piped to another filter, any errors or mistakes could also be piped.

Support/inhibit specific NFPs?

This architectural style is efficient, as it allows filters to work independently and asynchronously. This style is also scalable, as filters can be added and removed based on the complexity of the program. Since each filter is an independent module, it can be easily added to scale the system. It also inhibits dependability, as one broken filter, could result in a broken system.

The diagram below shows a simple representation of the pipe and filter architectural style. In this example we demonstrate how a pizza is made using this style. We are using 5 filters, with 3 of them working asynchronously. We implement the first few filters to process the raw ingredients to create the basic elements for the pizza(the vegetables, the sauce, and the pizza base). When all three of these have been completed, we can assemble them. After it has been assembled, we can then add the cheese to the pizza, and bake it, before delivering it to our customer, the sink.



Take vegetables as input
Take ingredients for sauce as input
Take ingredients for dough as input
Chop Vegetables
Make Sauce
Make Pizza Base
Filter
Sink
Pipe
Assemble 3 parts
Take cheese as input
Add Cheese and Bake
Consumer