# Are Services superior to Free Monads?

Yann Esposito

[2021-01-10 Sun]

TODO

A recurring hot topic in the functional programming world is how to make your code scale while keeping professionnal level of code quality.

Quite often in the functional programming we communities and talk people are focusing on enhancing specifics...

To organise your code in a functional paradigm there are many concurrent proposals. And structuring a code application is challenging. The way you need to structure the code generally need to reach a few properties.

1. You should make it easy to test your code
2. You need to support modern features any modern application is expected to provide. Typically ability to write logs, if possible send structured logs events.
3. The code should try to help people focalise on the business logic and put aside irrelevant technical details.
4. Split your applications into smaller (ideal composable) components
5. Control accesses between different components of your applications

The design space is quite open. In Haskell for example, there are different proposed solutions.

One of my preferred one to start with is the Handler Pattern[1]. Because it doesn't need any advanced Haskell knowledge to understand. And also it prevents a classical overabstraction haskell curse I often see within Haskellers. No premature abstraction here. No typeclass.

The main principle behing it is that you create *handlers*. Handlers are *component* focused that each provide a set of methods and functions already initialized.

[1] https://jaspervdj.be/posts/2018-03-08-handle-pattern.html

## Monads, MTL, RIO, Handler Pattern, Free Monad

There are a lot of solutions to architecture a program while keeping all the best properties of functional programming as well as best professional practices.

Here too, there are different level of looking at the problem of code organisation. On the very high level, an application is often understood as a set of features. But for all of thoses features to work together it is generally a lot of work to organise them.

So we can descend the level to look at code organisation. Files organisation, how to group them. Structure of the code organisation. How to put test, etc...

If you strive for composability you generally try to understand how to group "components" and ask yourselve what a componentn should contain. Here is a solution.

## Free Monads/Effect System

Foreword, semantic vs syntax.

The kind of best way to talk about semantic and forget about the syntax is to deal directly with a simplified representation of the AST.

Overall API:

```
(interpret-with
 [effect-1 effect-2 ... effect-n]
 (let [admin-user (get-in-config [:user :admin :user-id])
       admin (get-user admin-user-id)
       admin-email (get admin :email)]
   (log "Admin email" admin-email)
   admin-email))
```

It will be up to the actual instanciation of all `effect-*` to change the interpretation of the body. So some effect could have different interpreation of specific symbols. So here we can imagine that `get-in-config`, `get-user` and `log` are handlers specified in the effects.

One advantage is that to test your code you can simply use stubbed effects. One can use a list users

Real effects and free monads are in fact more powerful than this example is showing. For example, within a free monad, even `let` semantic would be changed. But let's not take this rabbit hole in this article right now.