Functional Programming Architecture

Asked 16 years, 3 months ago Modified 7 years, 7 months ago

Viewed 17k times

43

I'm familiar with object-oriented architecture, including use of design patterns and class diagrams for visualization, and I know of service-oriented architecture with its contracts and protocol bindings, but **is there anything characteristic about a software architecture for a system written in a functional programming language?**

I know that FP has been used for medium-size to large scale projects. Paul Graham wrote the first incarnation of Yahoo! Store in Common Lisp. Some lisp development systems are complex. Artifical intelligence and financial systems written in functional languages can get pretty big. They all have at least some kind of inherent architecture, though, I'm wondering if they have anything in common?

What does an architecture based on the evaluation of expressions look like? Are FP architectures more composable?

Update: Kyle reminded me that SICP is a good resource for this subject.

Update 2: I found a good post on the subject: *How does functional programming affect the structure of your code?*

architecture functional-programming

Share

Improve this question

Follow

edited Oct 13, 2008 at 0:27

asked Sep 18, 2008 at 1:33

Mark Cidade

99.8k ● 33 ● 229 ● 237

7 Answers

Sorted by: Highest score (default)

24

The common thread in the "architecture" of projects that use functional languages is that they tend to be separated into layers of algebras rather than subsystems in the traditional systems architecture sense.

For great examples of such projects, check out XMonad, Yi, and HappS. If you examine how they are structured, you will find that they comprise layers of monadic structure with some combinator glue in between.

Also look at The Scala Experiment paper which outlines an architecture where a system is composed of components that abstract over their dependencies.

Share Improve this answer

Follow

answered Sep 18, 2008 at 23:49

Apocalisp

35k ● 8 ● 108 ● 158

Thanks for your answer. I'm learning about denotational semantics ((in Haskell) [en.wikibooks.org/wiki/Haskell/Denotational_semantics] and (in general) [scss.tcd.ie/Andrew.Butterfield/Teaching/CS4003/…) and your answer resonates with what I'm learning. Decomposing a problem into mathematical constructs (e.g. the algebras you were talking about) in such a way that they can be composed into a larger system. – Tim Stewart Apr 21, 2014 at 14:24

1 Isn't saying "components that abstract over their dependencies" in FP like saying "Create a higher level of procedures and data structures" in imperative/OOP? My specific question was - HOW do you describe these higher-order abstractions? OOP has UML, with class diagram playing the primary role. What does FP have? – Victor Sergienko Dec 9, 2015 at 10:28

FP have integrated documentation (e.g Rust) used to describe these components. No need of UML or something else. You can create a doc closed to your code, and if you do it conscientiously, it will be more readable than any UML diagrams, I think. – Apitronix Nov 4, 2020 at 8:03

11

I'm currently working on the book "Design and Architecture in Functional Programming". It describes many design patterns and approaches that are exist in pure FP world (primary language is Haskell), but not only. The book teaches you how to build big application from scratch with pure and impure state, multithreading,

network, database, GUI, how to divide it into layers and obtain simplicity. It also shows how to model domains and languages, how to organize and describe architecture of the application how to test it, and even more.

The list of topics includes:

- Approaches to architecture modeling using diagrams;
- Requirements analysis;
- Embedded DSL domain modeling;
- External DSL design and implementation;
- Monads as subsystems with effects;
- Free monads as functional interfaces;
- Arrowised eDSLs;
- Inversion of Control using Free monadic eDSLs;
- Software Transactional Memory;
- Lenses;
- State, Reader, Writer, RWS, ST monads;
- Impure state: IORef, MVar, STM;
- Multithreading and concurrent domain modeling;
- GUI;
- Applicability of mainstream techiques and approaches such as UML, SOLID, GRASP;
- Interaction with impure subsystems.

The book is based on the Haskell projects I'm researching, especially a SCADA application Andromeda. The code for this book is available here. While the book is under development (it will be done till at the of 2017), I can recommend you to get familiar with my article "Design and Architecture in FP" here (Rus).

UPDATE

I shared my book online (first 5 chapters). See post on Reddit

Share Improve this answer Follow

edited May 22, 2017 at 1:13

answered Nov 7, 2016 at 4:16

Alexander Granin

983 ● 15 ● 20

3

The largest commonality you'll find in functional languages is using functions to store data. It's a bit like using accessor functions on an object without the object. Instead, the function is created in an environment where it has access to the data it needs. Now this function can be passed and used anywhere and still retain the ability to use the data.

Here's a very simple example. This isn't purely functional as it does change state, but it's common enough:

```
(define (make-counter)
  (let ((count 0))
    (lambda ()
      (set! count (+ count 1))
      count)))

(define x (make-counter))

(x) returns 1

(x) returns 2

...etc...
```

So we have a function, make-counter, that returns another function that has the state of the counter inside. We can call that newly created counter and observe the change inside.

This is how functional programs are structured. You have functions that take functions as arguments, you have functions that return functions with hidden state, etc. It's all a lot cleaner than managing memory yourself.

Share Improve this answer

Follow

answered Sep 18, 2008 at 1:46

Kyle Cronin

79k ● 45 ● 151 ● 167

1 I'm familiar with using closures to maintain state but I'm not so interested in what functional languages have in common so much as I am in what large, well-architected functional programs have in common. – Mark Cidade Sep 18, 2008 at 1:52

2 Sorry, I guess underestimated your familiarity with the field. One thing a lot of newcomers to functional programming try

to do is continue to use variables to encapsulate state, so I was trying to provide a more functionally-oriented way to do that. – Kyle Cronin Sep 18, 2008 at 19:35

1 I prefer Haskell's replace-the-immutable-World approach to changing state :) – Mark Cidade Sep 18, 2008 at 22:12

3

I printed out and looked over Design Patterns in Ocaml, and they use modules and functors (and objects) to recreate the normal design patterns we are used to. It's interesting, but I think they use objects *too* much to really see the benefit of functional languages. FP is very composable, part of it's nature. I guess my short answer is to use *modules* and *functors*.

My current project is pretty big, and we separate each module by files --implicit in ocaml. I've been hunting for a comprehensive resource as well that might have some alternative views or some thoughts on a really successful design that came out of a project.

Share Improve this answer

Follow

answered Sep 18, 2008 at 13:48

nlucaroni

47.9k ● 6 ● 66 ● 86

2

I have worked with some fairly large functional projects. They usually fall into two camps (at least, the ones I have used):

- Extreme scalability/reliability/concurrency.
  Transactional models can be built very tightly into the

language. Concurrent ML is a great example of this, and projects that use it are very hard to get wrong when it comes to concurrency correctness.

- Parsing/modifying frameworks. Many of the design patterns that these frameworks are based on are incredibly easy to formulate/build/modify in functional languages. The visitor pattern is a great example of this.

Share Improve this answer

Follow

answered Sep 18, 2008 at 1:50

hazzen

17.5k ● 7 ● 43 ● 33

> Yes, I'd imagine that functional designs would be more robust for concurrent execution. Parser combinator fraemworks, like in Haskell, are a good example of easier composability you get from a functional paradigm. – Mark Cidade Sep 18, 2008 at 1:57

2

Hopefully not too tangential, but probably interesting for anyone browsing the answers to this question is this presentation Design Patterns in Dynamic Programming by Peter Norvig.

Share Improve this answer

Follow

answered Sep 24, 2008 at 8:10

Stephen

1,055 ● 2 ● 26 ● 41

I think this may help;

> Some of the patterns disappear -- that is, they are supported directly by language features, some patterns are simpler or have a different focus, and some are essentially unchanged.

[AIM-2002-005] Gregory T. Sullivan, Advanced Programming Language Features for Executable Design Patterns "Better Patterns Through Reflection

March 22, 2002

> The Design Patterns book [GOF95] presents 24 time-tested patterns that consistently appear in well-designed software systems. Each pattern is presented with a description of the design problem the pattern addresses, as well as sample implementation code and design considerations. This paper explores how the patterns from the "Gang of Four", or "GOF" book, as it is often called, appear when similar problems are addressed using a dynamic, higher-order, object-oriented programming language. Some of the patterns disappear -- that is, they are supported directly by language features, some patterns are simpler or have a different focus, and some are essentially unchanged.

 answered Sep 18, 2008 at 20:15

Follow

Stephen

1,055 ● 2 ● 26 ● 41

I would expect as much from a functional arhcitecture, especialliy one that's based on a language with some kind of macro or metaprogramming facility, that it would be cleanly absent of any so-called design patterns (i.e., boiler-plate structures). – Mark Cidade Sep 18, 2008 at 22:14