Ola Bini: Programming Language Synchronicity

Ruby, Java, Lisp, Io, JRuby. Programming language archeology, creation and discovery.

TISDAG, JUNI 03, 2008

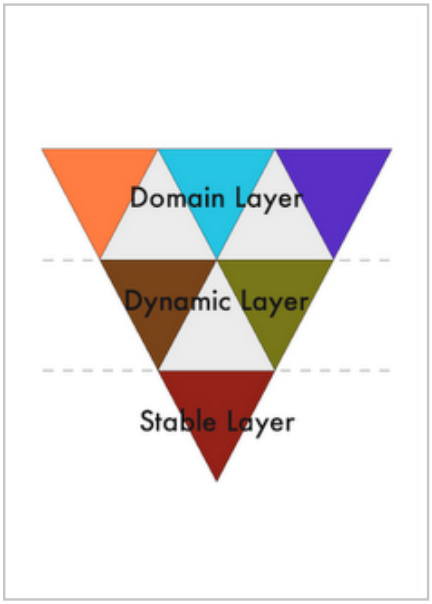
Fractal Programming

This is a continuation of my previous posts describing layers of code written in different programming languages. I have thought about the things involved for a while, and had several discussions with people about it. There were some parts that I didn't describe as well as I thought in my posts, and I will try to do better in this one.

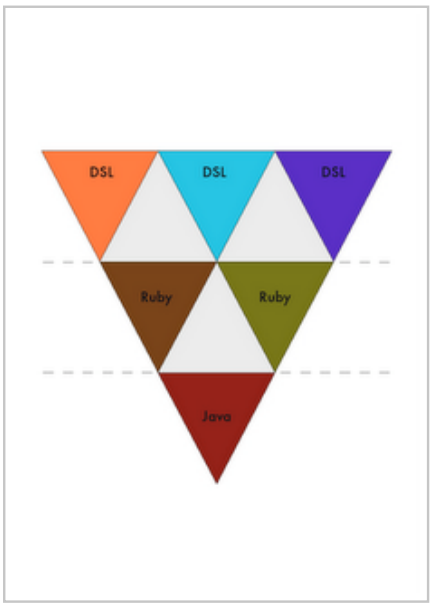
The core of these ideas are based on polyglot programming, the thinking that you should use several different languages in a project, based on which languages are better suited for different parts of it. Another term for this concept is Language-oriented programming. So how do you organize a polyglot system? The most natural way for me is to divide it into layers. In most cases you will find that different categories of languages will be better suited to different layers of the application.

In my original post I identified three layers that can be used to organize polyglot systems. These layers are the stable layer, the dynamic layer, and the domain layer. There are several reasons for organizing them this way, and I'll take a harder look at each of the layers further down. But first let me note that these layers are usually depicted in the form of a pyramid, with the stable layer being that base. That is definitely not how I think about it. In fact, I see it as an inverted pyramid, where the stable layer is the tip of the pyramid, providing the base. The Dynamic layer is the middle part. The domain layer should be the largest part and will very often include more than one dynamic language. So in my mind I represent the different domain languages as smaller pyramids standing upside down, covering the base area. Now, the dynamic layer can also be divided into smaller parts like this, based on language or functionality. This is a bounded fractal representation, which is the reason for the title of this blog post.

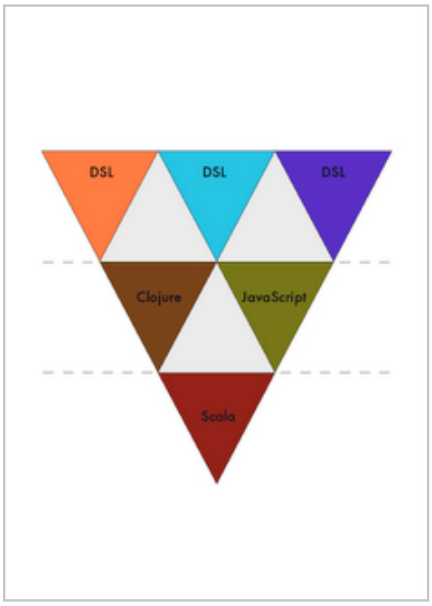
This diagram shows how I think about it:



Of course, the smaller pyramids can be all the same language and system, or several different ones. It all depends on the application or system you are building. So you can for example use a combination of Ruby, Java and external or internal DSLs:



Or you could use Clojure, Scala and JavaScript:



Or any other combination you can imagine. As long as the combination is what's best suited for the problem.

Let's take a look at the definitions of the different layers. There have been some discussion about the names I've chosen for them, so let me describe a little more what the responsibility of each part is, and why it's in that part of the system.

The Domain Layer

This layer is the simplest. This is where all the actual domain rules are defined. In general that means one or more domain specific languages. It doesn't really matter if they are internal or external. This model see them as the same layer. This part of the system is what needs to be malleable enough that it should be possible to change rules in production, allow domain experts to do things with it, or just plain a very complicated configuration. The languages

used in this layer are mostly external DSLs, but can also include extremely DSL-friendly languages like Ruby, Python or Groovy.

The Dynamic Layer

Neal Ford argues that this layer isn't so uch about dynamic, as it is about essence. That was never my intention. The problem is that even if you take a language like Scala, which is usually classified as an essential language, Scala requires compilation. To me, compilation is ceremony, which means that it's one extra thing you don't want to care about when writing most of your application code. That's why this layer needs to be dynamic. This is where languages like Ruby, Groovy, Python, JavaScript, Clojure and others live.

The Stable Layer

I view the stable layer as the core set of axioms, the hard kernel or the thin foundation that you can build the rest of your system in. There is definitely advantages to having this layer be written in an expressive language, but performance and static type checking is most interesting here. There is always a tradeof in giving up static typing, and the point of having this layer is to make that tradeof smaller. The dynamic layer runs on top of the stable layer, utilizing resources and services provided.

Another important feature of this layer is that this is where all interfaces are defined. By interfaces I mean external API's. They need to be hard for other clients to be able to trust them. But the implementations for them lives in the dynamic layer, not in the stable. By doing it this way you can take advantage of static type information for your API's while still retaining full flexbility in implementation of them. Languages in the stable layer can be Java, Scala or F#. It should be fairly small compared to the rest of the application, and just provide the base necessary services needed for everything to function.

The most common objection I hear from people about this strategy is the same as for the general polyglot programming idea: if we have a proliferation of languages in a system, it will be harder to find skilled programmers who can work with it.

This objection is true to a degree, but there are several ways around it. First, I have to say that I don't believe this is such a big problem as many others think. Programmers nowadays depend on their tool chains quite heavily, all of them including many advanced features that takes lots of time to learn. But most programmers doesn't even view their languages as tools. In my mind, the programming language is the most important tool. And once we start using better languages for systems, many of the things we need other tools for will disappear or become less of a problem.

I tend to believe that programming languages are quite easy to learn as soon as you understand the fundamental building blocks of programming languages. And if you don't have a fair understanding of these building blocks, I would say that you probably aren't using your current language as well as you should either. I see this as part of being responsible programmers.

I also believe quite strongly that if we used better languages for our code, many code bases would be smaller, easier to understand, easier to maintain and cost less - which means you could afford to find a more skilled programmer to do the work for you. This would mean that both parties win - the programmer gets more interesting work and better code, while the client gets more worth for his money in less time.

Upplagd av Ola Bini kl. 11:26:00 PM 

Etiketter: domain specific languages, fractal programming, polyglot, programming languages

10 kommentarer:

Michelle Montianto sa...

very difficult to understand for me. haha. i'm just a newbie programmer. still studying. hope i can get more

knowledge here. Thanks!

3:35:00 AM

Anonym sa...

I like the idea of multiple languages in principle, but calling it "fractal" would be a bit much, considering the requirements for self-similarity, infinitely fine structure, etc. Maybe "layered" or "standing on the shoulders of giants" would be more accurate.

5:35:00 AM

Greg sa...

So if F# has an interactive mode you can use it for two layers? (arguably all three if your DSL is embedded) I don't know if there is one yet, but OCAML and Haskell both have them, so it's bound to happen for F# sooner or later. Then whither your polyglot argument? Is it just that the mainstream languages are too outdated to accomplish both tasks, or is F# a bad fit for the dynamic layer in another way? (seems unlikely since your only objection to Scala was the lack of an interactive runtime)

8:06:00 AM

Anonym sa...

This only makes sense to me if I think of web development. I can't think of any other example where this inverted pyramid applies. can you provide some?

Anonym.

5:57:00 AM

Mats Henricson sa...

What is it about the dynamic level that requires a language with less ceremony, that doesn't exist in the stable layer?

With other words: Why not use a dynamic language also for the stable layer? If static typing makes sense in this layer, why not also in the dynamic layer?

Also, your definition of the stable level is rather hazy: *the core set of axioms, the hard kernel or the thin foundation that you can build the rest of your system in*. It doesn't help me much. Is the DAO layer roughly the same as the stable layer? If so, why is it called the **stable** layer? It changes all the time in our application!

1:15:00 PM

helium sa...

@Anonym: e.g Games like World of Warcraft or Crysis already use a similar structure but with a lot bigger "stable base" and a thinner "dynamic layer" (WOW and AFAIK Crysis use Lua here).

But I agree that web development probably is one the best examples of where you can use such a project structure.

@greg: While F# has a REPL, AFAIK it doesn't have an interpreter which you could use instead of the compiler.

It's about not compiling the code but just changing it and imidiatly see the results while you use the program / web page / whatever.

@other anonym: Yepp, I don't see a fractal structure here, either.

4:47:00 PM

patrickdlogan sa...

I don't think there is as much of a "stable" vs. "dynamic" dichotomy as you present. Or at least the so called "stable" should be darn small, e.g. the Squeak kernel, the Emacs kernel.

Rather dynamic languages can be used to implement highly efficient "stable" kernels without two different languages. I think you've given yourself false comfort - are you sure you want this? Why?

11:43:00 PM

Fabian martins sa...

Det här inlägget har tagits bort av den som skapade det.

8:52:00 PM

Fabian martins sa...

The problem is not just the difficulties of finding programmers. It is also about finding good programmers, good designers, keep a coherent methodology along the different language domains and to maintain the low TCO of systems.

For the programmers I say: learn as many languages as you can, including different paradigms.

For the enterprises I say: KIS - Keep It Simple. More languages represents more managerial complexity, more system complexity and almost surely more cost.

8:54:00 PM

Farley Knight sa...

@anonymous: Fractals are actually a very appropriate analogy for programming. True, it doesn't follow it's literal definition of self-symmetry, but the beautiful precision and mathematical poetry that is a complex, working system lends itself to a word like fractal.

I encourage Ola to keep promoting the idea of polyglot programming. I think it's about time people get off their high-horse about a "one true language" and learn to express themselves in multiple languages.

@Fabian: I have to completely disagree. As systems grow and become more complex it becomes necessary to be able to deliberately split things up into layers. And, no, you can't just use a cookie-cutter approach of using someone else's framework to decide where the layers live and what they consist of. That's not very innovative :)

5:43:00 PM

Skicka en kommentar

Senaste inlägg Startsida Äldre inlägg

Prenumerera på: Kommentarer till inlägget (Atom)