FOSDEM 2019

Minimalistic Languages Devroom

Why JSON when you can DSL?

Creating file formats & languages that fit your needs

Jérôme Martin

- Developer at OVH (hosting/datacenters)
- Former developer at Ubisoft (video games)

What we expect from JSON

JSON is used to represent structured data:

```
{
  "name": "My pony ranch",
  "owner": {
    "name": "Jerome",
    "surname": "Martin",
  },
  "ponies": [
    { "name": "Rarity", "level": 3 },
    { "name": "Applejack", "level": 2 },
    { "name": "Twilight", "level": 2 },
  ]
}
```

It allows some types only: strings, numbers, arrays, dicts… that’s it.

What if we want self-references?

We need to add IDs:

```
{
  "name": "My pony ranch",
  "owner": {
    "name": "Jerome",
    "surname": "Martin",
  },
  "ponies": [...]
}
```

➡

```
{
  "name": "My pony ranch",
  "owner": "1234", ← here
  "owners": {
    "1234": { ← here
      "name": "Jerome",
      "surname": "Martin"
    }
  },
  "ponies": [...]
}
```

Good luck to keep them in sync...

What if we need to count some elements?

```
{
  "name": "My pony ranch",
  "maxLevel": 3,
  "ponies": [
    { "name": "Rarity", "level": 3 },
    { "name": "Applejack", "level": 2 },
    { "name": "Twilight", "level": 2 },
  ],
  "numberOfMaxLvlPonies": ???, ← cannot be computed with JSON only
}
```

We need a program to process our data.

JSON is not enough

In fact, data structures are not enough!

- They fail at self-referencing.
- They fail at representing processes and computations.

What does everyone do?

Let’s add javascript to it!

```
// sample Grunt file from any JS project
module.exports = function (grunt) {
  "use strict";
  require("matchdep").filterAll("grunt-*").forEach(grunt.loadNpmTasks);

  grunt.initConfig({
    pkg: grunt.file.readJSON("package.json"),
    distdir: "dist",
    srcdir: "src",
    transdir: ".work/.trans",
    testdir: ".test/",
    builddir: ".work/.tmp",
    name: grunt.file.readJSON("package.json").name,

  // thousands of lines ...
```

What were simple data files have become monsters. WHY? :(

How data structures betrays us

"A data structure is just a stupid programming language" – Bill Gosper

JSON, XML, HTML, CSS... are all stupid programming languages.

When we try representing concepts with them, abstraction inherently leaks.

We always end up writing the missing abstraction layer by hand.

In need for meta

We try to make them less stupid:

- CSS → Less
- Javascript → Babel
- HTML → Mustache templates / JSX syntax
- C → Preprocessor
- C++ → Templates
- Python → meta-classes
- C# → Reflection

What if I told you

There’s a simpler way.

There’s a more personal way.

The DSL way

Domain Specific Languages

→ The abstraction becomes a language.

Examples: Makefile, Regexps, SQL, Qt, GameMaker Language...

All those languages are used to represent complex data structures.

The DSL way

Structured data will always be better expressed with a specific language for that domain than a generic data structure.

- Banking data → banking language
- Game data → game language
- Medical data → medical language

But isn’t writing a full language excessive?

Rule #1: Abstraction leaks

Any data eventually becomes a DSL *naturally* by leaking through abstractions.

→ Natural growth

Most programs are tools made to prevent this leakage (a.k.a. "middleware").

Only most of the time they become a badly written half implementation of lisp.

So why not using lisp in the first place?

Racket

The language-oriented programming language

racket-lang.org

As a lisp language, it allows writing itself *by design*.

Racket is specialized in writing Domain Specific Languages (DSL):

- #lang slideshow
- #lang racket/gui
- #lang scribble
- #lang video
- #lang web-server

How to make a DSL

It takes 5 lines of Racket to implement a generic parser for any language:

```
#lang racket/base
(provide (except-out (all-from-out racket/base)
                     #%module-begin)
         (rename-out [module-begin #%module-begin]))

(define-syntax-rule (module-begin stx)
  (#%module-begin 'stx)) ; ← insert your logic here
```

Save those lines in a file "*my-lang.rkt*" and you got yourself a full **reader**, **parser** & **expander** with all the standard functions from Racket.

Code == Data

Using the lisp syntax called "s-expressions", you blur the frontier between code and data.

```
#lang web-galaxy
(response (pony id)
  (define the-pony (get-pony-by-id id)) ; ← database fetching
  (html
    (head
      (style
        (.pony (border 1 'solid 'pink) ; ← CSS
               (background-color 'dark-pink))))
    (body
      (div ([class "pony"]) (pony-name the-pony)) ; ← HTML
      (javascript
        (function (feed-pony elt) ; ← Javascript
          (add-class elt "fed")))))))
```

Full project: **github.com/euhmeuh/web-galaxy**

Code == Data

```
#lang virtual-mpu/mpu
(mpu "6802"
  (registers (a b sr [ix 16] [sp 16] [pc 16]))
  (status sr (carry overflow zero negative interrupt half))
  (interrupts interrupt [irq #xFFF8]
                        [soft #xFFFA]
                        [nmi #xFFFC]
                        [restart #xFFFE])
  (operations
    ;; branches
    (bcc "Branch if carry clear" (rel) (branch (carry?) rel))
    (bcs "Branch if carry set" (rel) (branch (not (carry?)) rel))

    ;; other operations...
    ))
```

Full project: **github.com/euhmeuh/virtual-mpu**

Scribble

The Racket documentation language

```
#lang scribble/base

@title{On the Cookie-Eating Habits of Mice}

If you give a mouse a cookie, he's going to ask for a
glass of milk.

@section{The Consequences of Milk}

That ``squeak'' was the mouse asking for milk. Let's
suppose that you give him some in a big glass.
```

Take away

- When data structures are not enough, we seek more abstraction.
- DSLs are the best way to express those abstractions as languages.
- Racket is a language specialized in writing languages.
- By putting your "middleware" logic into a language, you make sure it can evolve and always fit your domain.
- You don’t have to write ugly data transformation scripts ever again!
- You provide your team with a single piece of documentation concerning your domain: the language spec.

Come make your language today!

Join me at the booth **K.4.201** at **17:20** to make your own language!

Yes, you can actually make your own language in an hour with Racket!

See you there!

Jérôme Martin
Web developer at OVH
Racket contributor
`github.com/euhmeuh`
`"jerome" on racket.slack.com`