Different Ways to Build a DSL

Pat Hanrahan

Approaches

External DSL

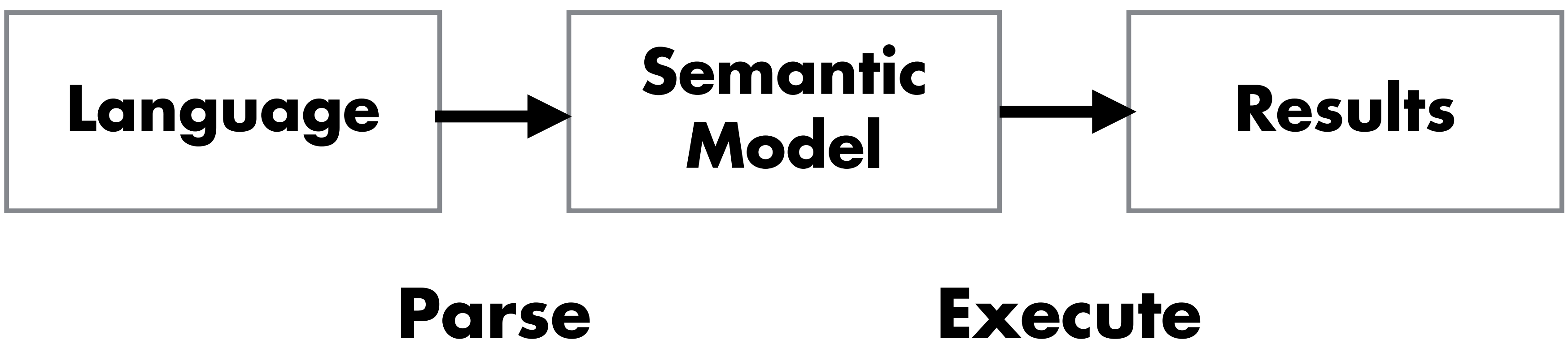
An external DSL is implemented as a standalone language.

Embedded (Internal) DSL

An internal DSL is embedded within a another language. Ideally, the host language has features that make it easy to build DSLs.

External DSLs

Language Implementation



calc.py

lexical analysis
syntactic analysis
interpretation

Advantages

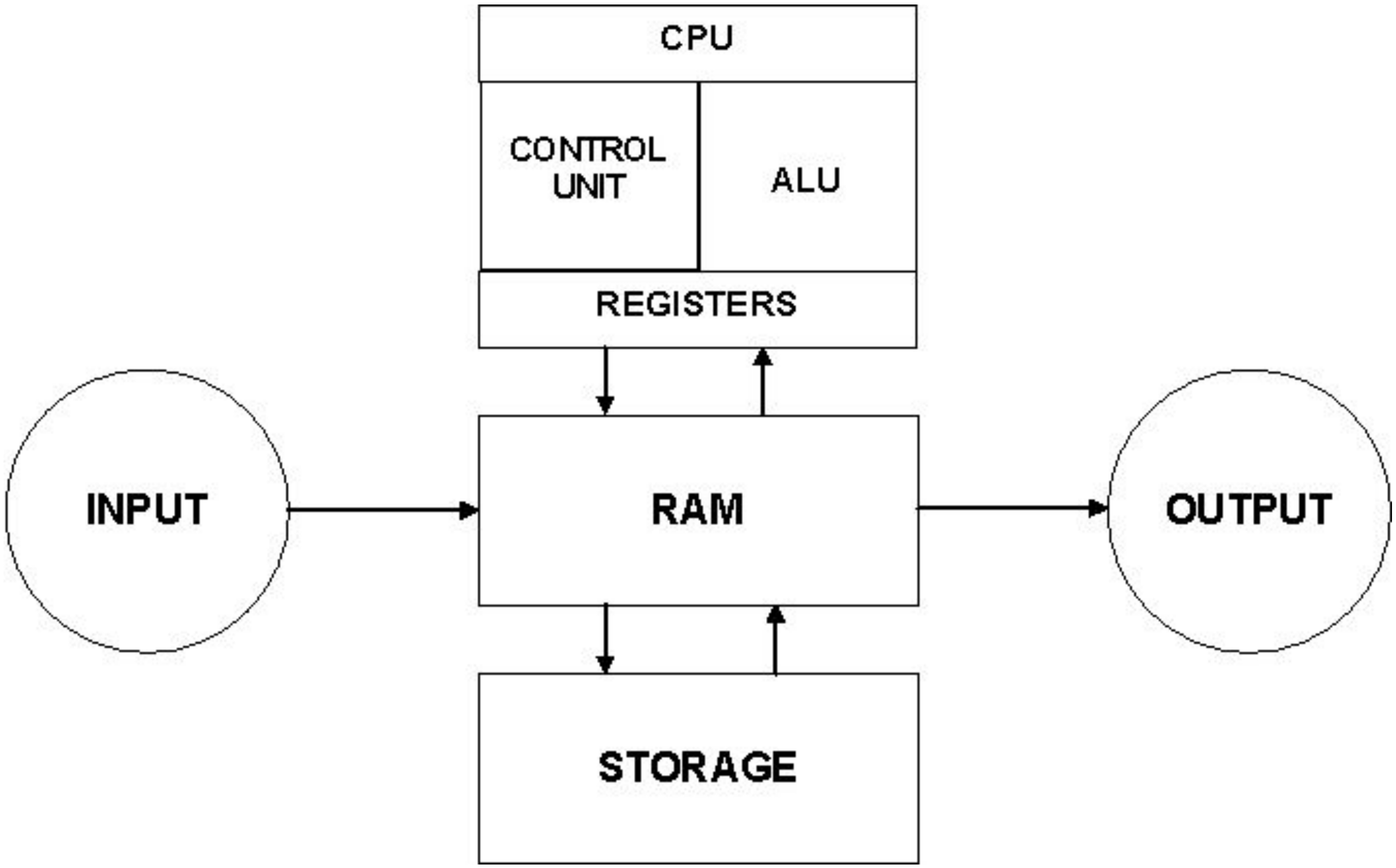
- Flexibility (syntax, semantics)
- Simple languages are simple

Disadvantages

- Yet-Another-Programming-Language
- Syntactical cacophony
- The slippery slope of generality
- Interpretation is slow
- Hard to interoperate with other languages
- No tool chain: IDE, debugger, profiler, ...

Embedded DSLs

Model of a Computer



```
// Semantic model

Processor p = new Processor(
    cores=2, speed=2500, isa=i386 );

Disk d1 = new Disk(
    size=150, speed=UNKNOWN, interface=null );

Disk d2 = new Disk(
    size=75, speed=7200, interface=SATA );

return new Computer(p, d1, d2);

// From Fowler 2010
```

```
// Function sequence.

computer();
  processor();
    cores(2);
    speed(2500);
    i386();
  disk();
    size(150);
  disk();
    size(75);
    speed(7200);
    sata();
```

```
// OpenGL

glMatrixMode(GL_PROJECTION);
glPerspective(45.0);

for( ;; ) {
    glBegin(TRIANGLES);
        glVertex(…);
        glVertex(…);
        …
    glEnd();
}

glSwapBuffers();
```

```
// OpenGL “Grammar”

<Scene> = <BeginFrame> <Camera> <World>
<EndFrame>

<Camera> = glMatrixMode(GL_PROJECTION) <View>
<View> = glPerspective | glOrtho

<World> = <Objects>*
<Object> = <Transforms>* <Geometry>
<Transforms> = glTranslatef | glRotatef | …
<Geometry> = glBegin <Vertices> glEnd
<Vertices> = [glColor] [glNormal] glVertex
```

Fluent Interface

“Composable API Calls”

```
// Nested functions:

computer(
  processor(
    cores(2),
    speed(2500),
    i386
  ),
  disk(
    size(150)
  ),
  disk(
    size(75),
    speed(7200),
    SATA
  )
);
```

```
// Method chaining.

computer()
  .processor()
      .cores(2)
      .speed(2500)
      .i386()
   .disk()
      .size(150)
   .disk()
      .size(75)
      .speed(7200)
      .sata()
   .end();
```

http://d3js.org/

https://jquery.com/

http://d3js.org/

```
// Lynq

int count =
   (from character in Characters
   where character.Episodes > 120
   select character).Count();
```

Operator Overloading

https://docs.python.org/2/reference/datamodel.html

“Overloading”

Not all “operations” can be intercepted

- Arithmetic operators
- Iteration operators
- Function definition?
- Type/class definition?
- Equality?
- Assignment?

“Monkey patching” like this can be dangerous

Type-directed embedding

```
// Minimal syntax

// Lisp
(cond
  ((= n 10) (= m 1))
  ((> n 10) (= m 2) (= n (* n m)))
  ((< n 10) (= n 0)))

// Smalltalk, Ruby
employee name first
  = employee.name.first
```

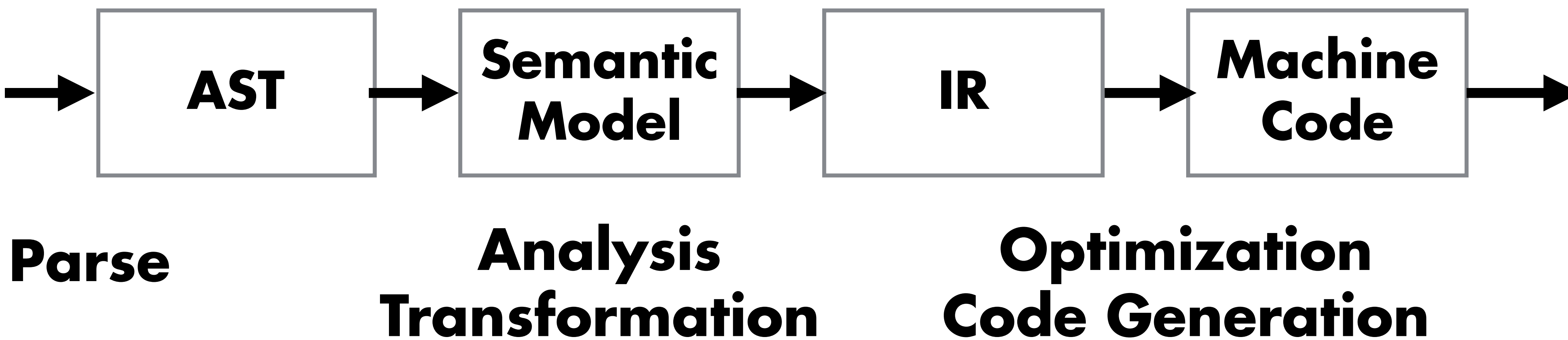
Advantages

- No need to learn another language
- Familiar syntax
- Still have access to general-purpose features
- Can interoperate with other libraries and classes
- Complete tool chain

Disadvantages

- Syntax is rigid and verbose
- Interpreters are still slow
- Hard to debug DSLs using current tool chains
- Hard to limit features in the language
- Still hard to develop

Language Implementation



Terra

A low-level counterpart to Lua

Download **TAR Ball** | View On **GitHub**

Getting Started

API Reference

Publications

Terra is a new low-level system programming language that is designed to interoperate seamlessly with the **Lua** programming language:

```
-- This top-level code is plain Lua code.
print("Hello, Lua!")

-- Terra is backwards compatible with C
-- we'll use C's io library in our example.
C = terralib.includec("stdio.h")

-- The keyword 'terra' introduces
-- a new Terra function.
terra hello(argc : int, argv : &rawstring)
    -- Here we call a C function from Terra
    C.printf("Hello, Terra!\n")
    return 0
end

-- You can call Terra functions directly from Lua
hello(0,nil)

-- Or, you can save them to disk as executables or .o
-- files and link them into existing programs
terralib.saveobj("helloterra",{ main = hello })
```

Like C, Terra is a simple, statically-typed, compiled language with manual memory management. But unlike C, it is designed from the beginning to interoperate with Lua. Terra functions are first-class Lua values created using the `terra` keyword. When needed they are JIT-compiled to machine code.

You can **use** Terra and Lua as…

A scripting-language with high-performance extensions. While the

Zach DeVito
zdevito at stanford dot edu

"The future ain't what it used to be"

- Yogi Berra