STUDIOS
ThoughtWorks®

Domain Specific Languages

What, why, how

Ola Bini
JRuby Core Developer
ThoughtWorks Studios

ola.bini@gmail.com
http://olabini.com/blog

ThoughtWorks

Global consulting firm

US, Canada, UK, Australia, India, China - and Sweden

1100 people worldwide

Agile

Open Source

Ruby

Martin Fowler

About me

Ola Bini

From Sweden

JRuby

Ioke

Xample

Member of the JSR232 expert group

STUDIOS
ThoughtWorks®

Agenda

To understand what a DSL is and isn't

When a DSL might be useful

How to go about implementing an internal DSL in Java

How to go about implementing an external DSL with Java

Why are DSLs so common in Ruby?

STUDIOS
ThoughtWorks®

What is a DSL?

"a computer programming language of limited expressiveness focused on a particular domain" - Martin Fowler

STUDIOS
ThoughtWorks®

The benefits of a DSL

- Increase development productivity
  - clear code is easier to evolve
  - easy to achieve but relatively small impact
- Communication with domain experts
  - common language to define system behavior
  - reading is more vital than writing
  - hard to achieve but big impact
- Shift in execution context
  - compile time to run-time
  - different platform (eg, generating SQL)
- Declarative computational model

DSL problems

Cost of building

Language cacophony

Hard to design

Migration

Evolving into generality

STUDIOS
ThoughtWorks®

DSLs are all around you

SQL

struts-config.xml

FIT

rake

LINQ

make

ant

CSS

rails validations

JMock expectations

regular expressions

Hibernate Query Language

graphviz

STUDIOS
ThoughtWorks®

DSLs aren't new

Lisp "Little languages"

Unix utilities - sed, grep, etc

Make tools

Different kinds of DSLs

External

- Separate to Host language
- Needs a compiler/interpreter
- Can be graphical

Internal

- Written in host language
- Conventional use of subset of host language syntax
- Sometimes called embedded DSL or fluent interface

STUDIOS

ThoughtWorks®

Internal DSLs

STUDIOS
ThoughtWorks®

Context

DSLs have an implicit context

Contrast this Car API:

```
Car car = new CarImpl();
MarketingDescription desc = new
MarketingDescription();
desc.setType(“Box”);
desc.setSubType(“Insulated”);
desc.setAttribute(“length”, “50.5”);
desc.setAttribute(“ladder”, “yes”);
desc.setAttribute(“lining type”, “cork”);
car.setDescription(desc);
```

Method chaining

Instead of returning void, return the receiver

StringBuilder.append

This allows chaining of multiple invocations:

```
Log.withRoot("foo")
    .usingAppender("bar")
    .formattedWith("xlayout")
    .usingAppender("quux")
    .formattedWith("ylayout");
```

STUDIOS

ThoughtWorks®

Fluent interfaces: wrapping APIs

Fluent interfaces improve the readability of any code

You can wrap existing APIs in fluent interfaces

Invocation semantics

Chained methods

```
e().i().e().i().o();
```

Nested methods

```
e(i(e(i(o()))));
```

STUDIOS
ThoughtWorks®

Object scoping

Put the code that uses a DSL in the subclass

JMock matchers:

```
context.checking(new Expectations() {{
  one(clock).time();
    will(returnValue(loadTime));
  one(clock).time();
    will(returnValue(fetchTime));

  one(loader).load(KEY);
    will(returnValue(true));
}});
```

STUDIOS

ThoughtWorks®

External DSLs

STUDIOS
ThoughtWorks®

Pros and cons

Sometimes hard to get desired semantics with an internal DSL

External DSLs allow changing the computational model easier

Flexibility in language design

Perceived as more difficult than they actually are

Do complicate builds, as it is a different technology

STUDIOS

ThoughtWorks®

Delimiter directed translation

Chop input primary delimiter (usually line endings)

- Send each line for separate processing

Parser need to keep state for hierarchic context (so try to avoid those)

Look for keywords manually in each line

Or use regular expressions to match input

STUDIOS
ThoughtWorks®

Syntax directed translation

Use a grammar file to specify syntactic structure

Grammar is a DSL to drive a parser

Pure parser only says if a text is part of a language or not

- Not exactly useful in itself

So extract parse tree that gets built during the above process

Grammar file

DSL:

```
events
  doorClosed  D1CL
  drawOpened  D2OP
end

commands
  unlockPanel  PNUL
  lockPanel    PNLK
end
```

Grammar:

```
list : eventList commandList;
eventList : 'events' eventDec* 'end';
eventDec : identifier identifier;
commandList : 'commands' commandDec*
              'end';
commandDec : identifier identifier;
```

STUDIOS
ThoughtWorks®

Building a syntax translator

Write by hand

- Usually recursive descent
- Easy to do from an LL(1) grammar

Use parser generator

- yacc (bison, etc)
- ANTLR
- ... many others

Parts of a syntax translator

Lexer (scanner, tokenizer)

- Breaks text into tokens

Parser (syntactic analyzer)

- Arranges token into parse tree

Semantic analysis

- Checks rules beyond what parser can do

Output production

- Do something useful

Embedded interpretation

Does something directly when the parse element is encountered

Easy to get started

Doesn't scale well

Generally require action code to be embedded in grammar

STUDIOS
ThoughtWorks®

Tree construction

Parser returns a parse or abstract syntax tree

This can be used in any way

ANTLR allow the output tree to be rewritten into a more convenient format

ThoughtWorks STUDIOS

Features of parser generators

- Platform
- Style of grammar
  - BNF or EBNF
  - Grammar class: LL(1), LL(*), LALR(1), SALR ...
  - How code is embedded
  - Separate lexer?
- Tools
- Documentation

Workbenches

- Sophisticated tooling for integrated external DSLs:
  - Microsoft Software Factories
  - Intentional Software (Charles Simonyi)
  - JetBrains Meta-Programming System (MPS)
  - MetaEdit
  - XText
  - Microsoft Oslo
- Schema definitions
- Editor definition
- Code generation

Blurry borders

- External DSL vs general purpose language
  - Is XSLT a DSL?
  - Is R a DSL?
- Internal DSL vs API
- Language Workbench vs Configurable Application
  - Is Access a language workbench?
- Human jargon vs Computer language

STUDIOS ThoughtWorks®

Why the Ruby+DSL marriage?

Ruby allow easy runtime code evaluation

It has a very flexible syntax

- Avoid parenthesis, semicolons

Send code along to methods - deferred evaluation

Good taste in APIs.

ActiveRecord:

```
class Blog < ActiveRecord::Base
  belongs_to :user
  has_many :posts
  validates_presence_of :name
end
```

Q and A