ThoughtWorks

JavaOne

Language-Oriented Programming and Language Workbenches: Building Domain Languages Atop Java™ Technology

Neal Ford

ThoughtWorker/Meme Wrangler
ThoughtWorks
www.thoughtworks.com

Session TS-1589

What This Session Covers

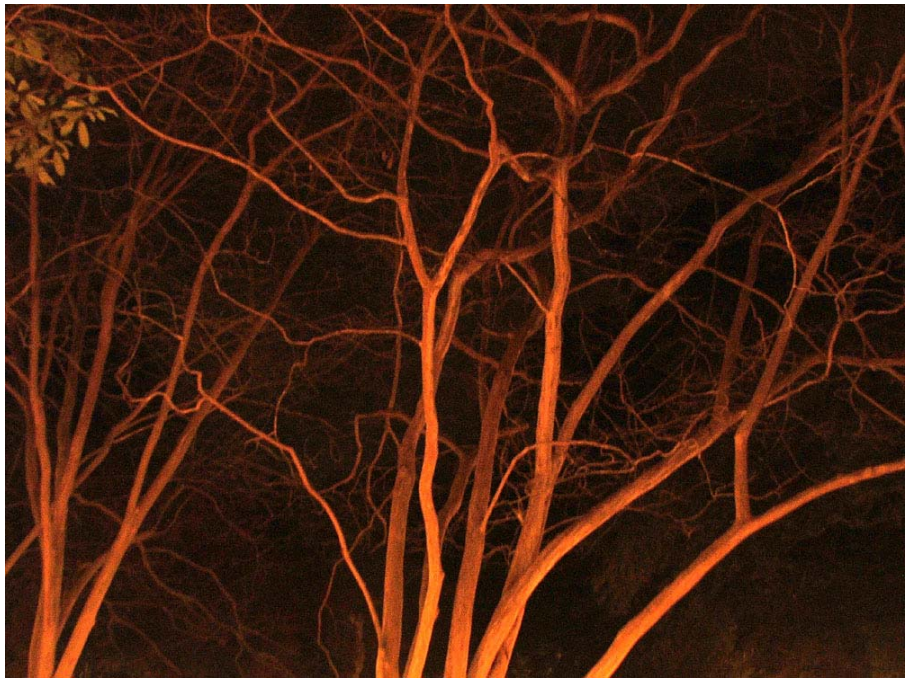
- Motivation
- Internal vs. external DSLs
- Building internal DSLs in
  - Java technology
  - Groovy
  - Ruby (via JRuby)
- Building external DSLs
- DSL best practices

Questions

- Why is there so much XML mixed in with my Java code?
- Why do we need things like aspects?
- Why won't everyone shut up already about Ruby on Rails?
- Is there an evolutionary step beyond object-oriented programming?

Modeling the World With Trees

Modeling the Real World



Changing Abstraction Styles

- Layers of abstraction using language
  - Not trees
- Trees and hierarchies still exist
  - Underneath a stronger abstraction layer
  - Objects, aspects, generics, et al become the building blocks for DSLs
- Allows developers to work at a higher level of abstraction
- Declarative vs. imperative programming

Why Use DSLs for Abstraction?

- "Iced decaf triple grande vanilla skim with whip latte"
- "Scattered, smothered, covered"
  - The Waffle House Hash Brown language has 8 keywords (all inflected verbs)
  - Scattered, smothered, covered, chunked, topped, diced, peppered, and capped
- "Route 66, swinging, easy on the chorus, extra solo at the coda, and bump at the end"
- "OMFG D00d Bob is t3h UBER 1337 R0XX0RZ LOL"

Including Your Business

- Even if you are a Java technology ace
  - You still have to learn the DSL for your business on day 1
  - This is the hardest part of your job

Observation

Every non-trivial human behavior has a domain specific language.

Nomenclature

- Coined by Martin Fowler
- Domain-specific language
  - A limited form of computer language designed for a specific class of problems
- Language-oriented programming
  - The general style of development which operates about the idea of building software around a set of domain specific languages

DSLs vs. APIs

- An API has an **explicit** context

```
Coffee latte = new Coffee(Size.VENTI);
latte.setFatContent(FatContent.NON_FAT);
latte.setWhip(Whip.NONE);
latte.setFoam(Foam.NONE);
latte.setTemperature(Temp.EXTRA_HOT);
latte.setStrength(5);
```

DSLs vs. APIs

- DSLs have an implicit context
- Consider the real-world examples
  - The context is never mentioned
  - Once a context is established, repeating it over and over is just noise

Venti half-caf, non-fat, extra hot, no foam, no whip latte

Internal vs. External DSLs

- Internal DSLs sit atop your base language
  - Must follow the syntax rules of the base language
  - Why Groovy and Ruby make better bases
- External DSLs
  - Create a lexer and parser
  - Can take on any syntax you like
    - Let your imagination be your guide!
  - Hard to create…

Fluent Interface

- Creating a readable model
  - Convert APIs to English-like sentences
- Slightly harder to write
- Much easier to read

Car API

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

Car Fluent Interface

```
Car car = new CarImpl().withMarketingDescriptionOf(
            new MarketingDescriptionImpl("Box", "Insulated”☐).
             andAttributeOf("length", "50.5").
             andIncludesA("ladder").
             andAttributeOf("lining type", "cork"));
```

Existing Fluent Interfaces

Fluent Interface: Hamcrest

- Hamcrest is an open source library from Google that creates fluent interfaces around JUnit matchers

```
assertThat(theBiscuit, equalTo(myBiscuit));

assertThat(theBiscuit, is(equalTo(myBiscuit)));

assertThat(theBiscuit, is(myBiscuit));
```

Fluent Interface: Mocks

- JMock

```
class PublisherTest extends MockObjectTestCase {
    public void testOneSubscriberReceivesAMessage() {
        Mock mockSubscriber = mock(Subscriber.class);
        Publisher publisher = new Publisher();
        publisher.add((Subscriber) mockSubscriber.proxy());
        final String message = "message";
        // expectations
        mockSubscriber.expects(once()).
                              method("receive").with(eq(message));
        // execute
        publisher.publish(message);
    }
}
```

Building Internal DSLs

In Java Technology

Example: Logging Configuration

- Logger setup is ugly
- Very API-ish
- Uses
  - A properties file
  - Code
  - An XML file
- Demo
  - LoggingConfiguration.java

Fluent Interface: Wrapping iBatis

- Humane interfaces improve the readability of any code
- You can wrap existing APIs in fluent interfaces
- Example
  - iBatis is an open source O/R mapping tool
  - It drips of API style of coding
  - Wrapping iBatis access in a fluent interface
  - Demo
    - EventPersisterImpl.java

Java Technology: A Calendar DSL

- Goal
  - Create a calendar application in Java technology using DSL techniques
  - Primarily uses fluent interface
  - Demo
    - Appointment.java
    - AppointmentCalendar.java
    - CalendarDemo.java

Building Internal DSLs

In Groovy

Internal DSLs in Groovy

- Groovy makes a better base for DSLs
  - Open classes via categories
  - Closures
  - Looser syntax rules
  - Dynamic typing

Building Blocks: Closures

- Closures mimic scope capturing method pointers
- Like a method, a closure defines a scope
  - Can still reference variables from the enclosing scope
  - Accepts parameters
  - Allows “with” semantics with categories
- In a DSL, provides containership semantics

Building Blocks: Open Classes

- Open Classes via categories
- Groovy allows you to attach methods to an existing class
  - Either Groovy or Java Development Kit (JDK™)
  - Yes, you can add methods to String
- Categories are classes with static methods
  - Each method's first parameter is self
  - Fake object-orientation
- Category demo => Adding methods to String

Time DSL in Groovy

- The goal: create a fluent interface around time spans and calendars
- Target syntax

```
2.days.fromToday.at(4.pm)
```

- Returns a calendar for that date and time
- Demo
  - IntegerWithTimeSupport.groovy
  - CalendarDsl.groovy
  - TestTime.groovy
  - CalendarDslDemo.groovy

Who Returns What?

2.days.fromToday.at(4.pm)

- 4.pm => Integer
- At
  - Accepts Integer
  - => Calendar
- fromToday => Calendar
- Days => Integer
- 2 => Integer

Builders in Groovy

- Builders make it much easier to build structures
  - XML documents
  - Swing user interfaces
- Built using a fluent interface
- Demo
  - Generating XML schema and POJO from a database schema

Groovy: Calendar

- The goal
  - Create an appointment calendar using DSLs
  - Demonstrates
    - Open classes
    - Closures
    - Loose syntax rules
  - Demo
    - Appointment.groovy
    - AppointmentCalendar.groovy
    - IntegerWithTimeSupport.groovy
    - AppointmentCalendarDemo.groovy

Building Internal DSLs

In Ruby

Ruby

- Ruby allows you to take DSL writing much further
- Ruby features that enable DSLs
  - True open classes
  - Closures
  - Really flexible syntax rules

Time DSL in Ruby

- Goal
  - Support time ranges in Ruby
- Demo
  - time_dsl.rb
  - time_dsl_test.rb

Ruby Calendar

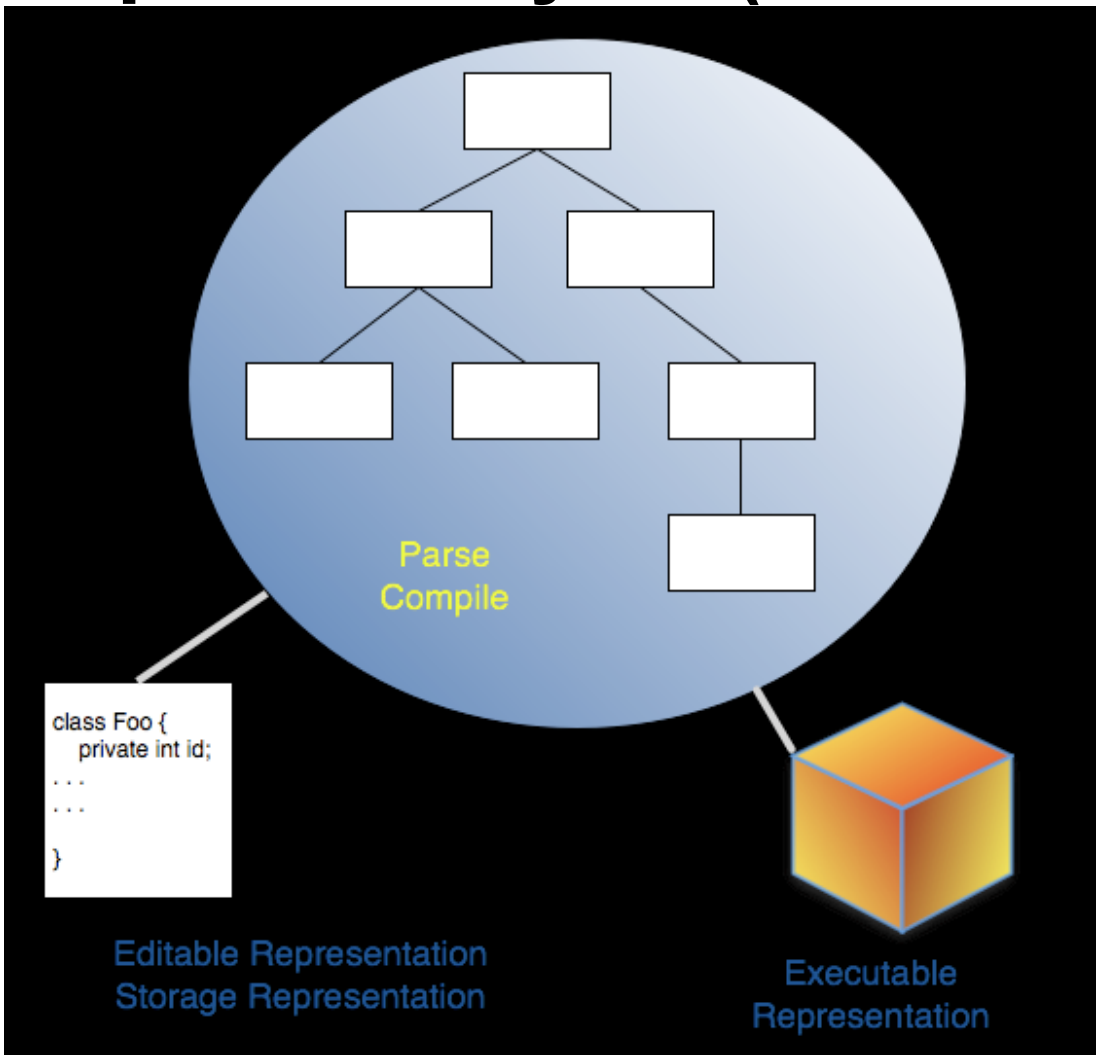
- Our calendar example in Ruby
- Demo
  - calendar_fluent.rb
- Functionally the same as the Groovy one
- Cleaner syntax
  - Less cruft
  - True open classes

Building External DSLs

Language Workbenches

- A language workbench is a tool that supports Language oriented programming
- Today's language workbenches
- Intentional Software (developed by Simonyi)
- Software factories (developed by Microsoft)
- Meta Programming System (developed by JetBrains)

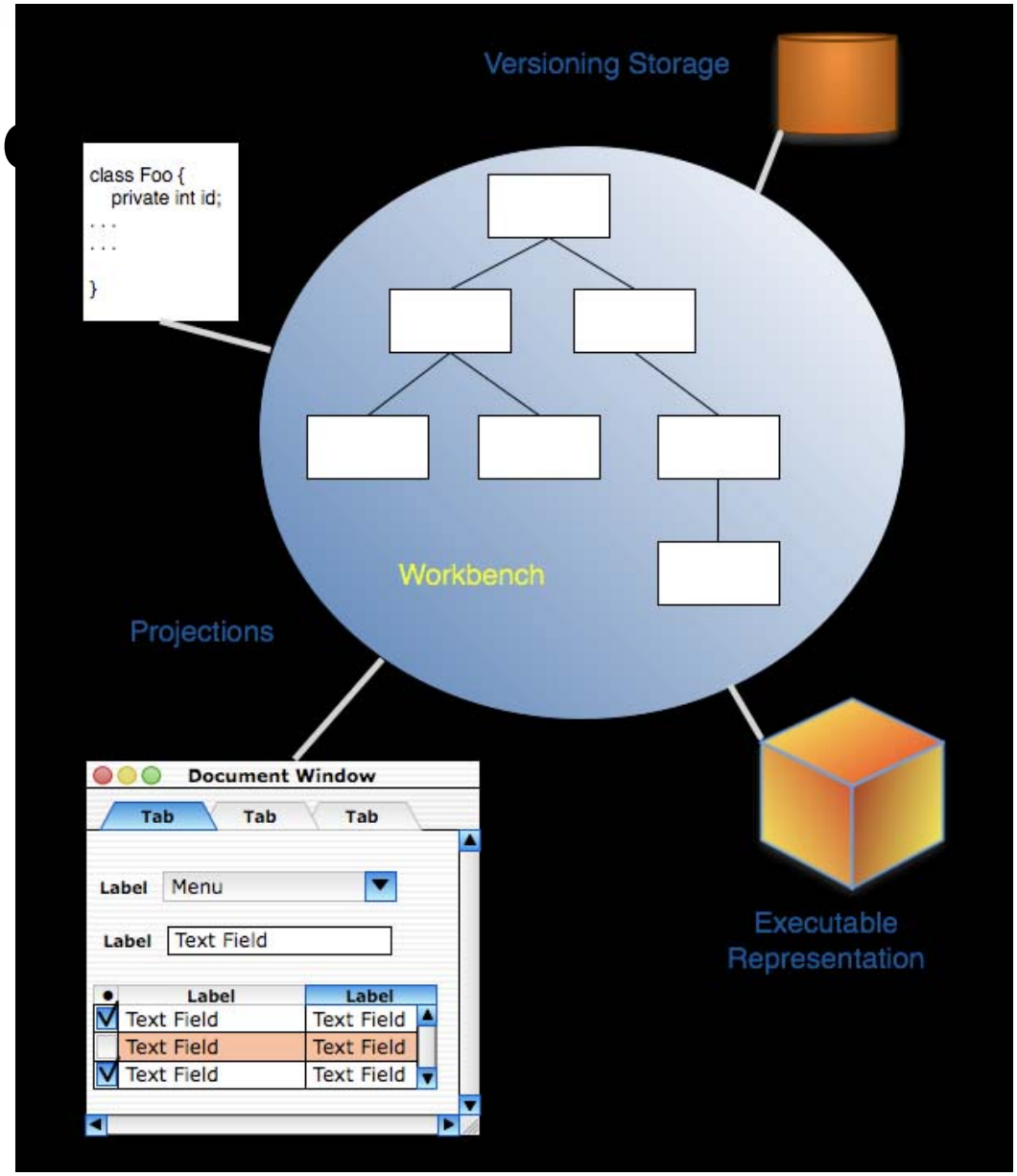
Compilation Cycle (Since CS-101)



“Post-IntelliJ” IDEs

- First tool that allowed you to edit against the abstract syntax tree instead of text
- How refactoring and other intelligent support works

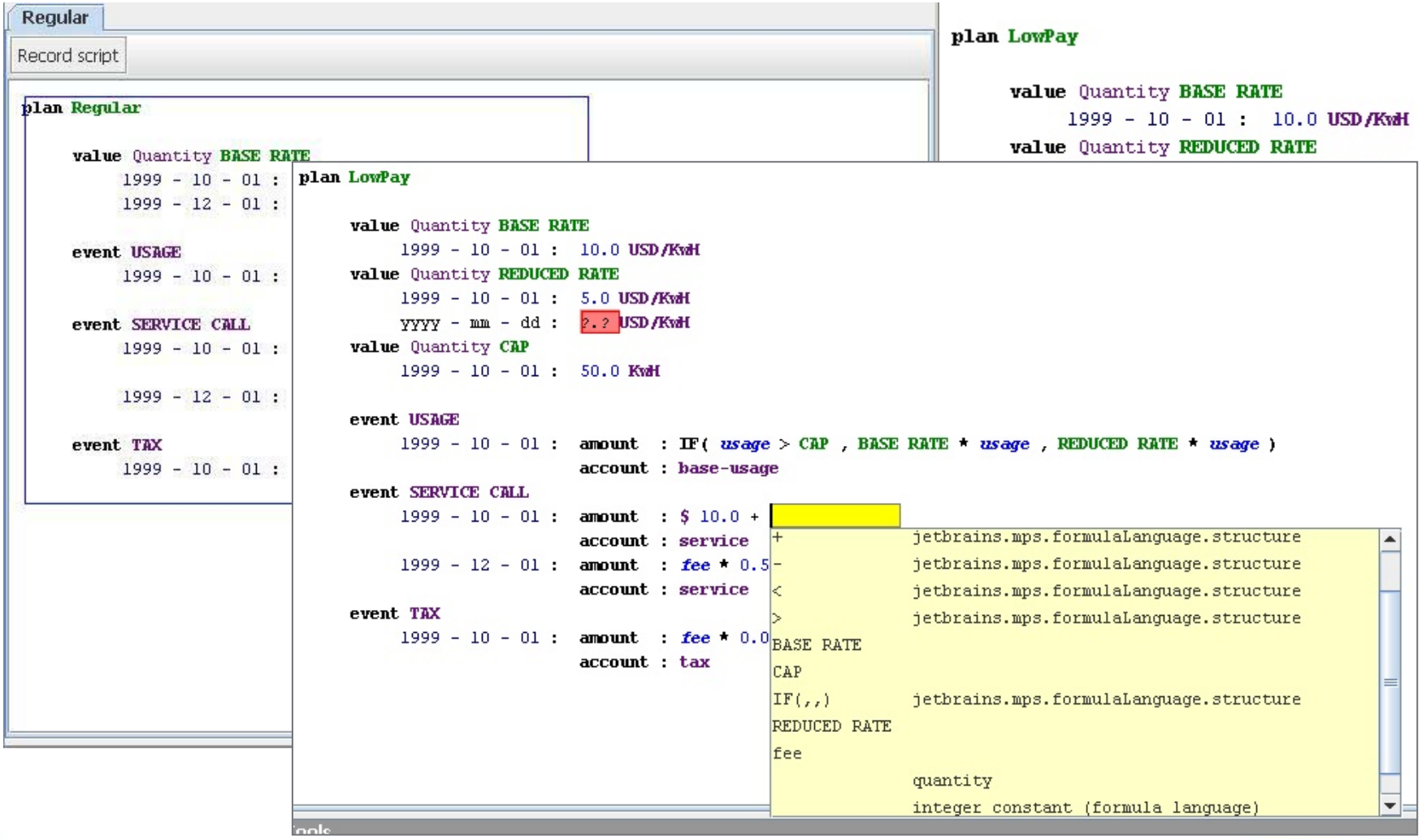
Workbench


Versioning Storage
class Foo {
private int id;
. . .
. . .
}
Workbench
Projections
Executable
Representation
Document Window
Tab
Tab
Tab
Label
Menu
Label
Text Field
Label
Label
Text Field
Text Field
Text Field
Text Field
Text Field
Text Field

Language Workbenches

- Editable representation is a projection of the abstract representation
- Abstract representation has to be comfortable with errors and ambiguities

JetBrains MPS



DSL Best Practices

Start with the End

- When using a flexible base language, envision the perfect result
- The Rake napkin

Test, Test, Test!

- Writing the DSL is the tricky part
  - Using it should be easy
  - Otherwise you've made some mistakes
- Test all the small parts

The Problem Domain

- Keep it as cohesive as possible
- Don’t try to write the entire universe in your DSL
- Better off using a bunch of very specific DSLs
- JetBrains and the way they are using MPS

DSLs

- A huge competitive advantage
  - All your code is abstracted at the problem domain
  - Harder to write, easier to maintain
  - Show your code to your business analysts for verification

Questions?
Samples and slides at www.nealford.com

Neal Ford
www.nealford.com
nford@thoughtworks.com
memeagora.blogspot.com