ThoughtWorks®

Patterns of Internal DSLs

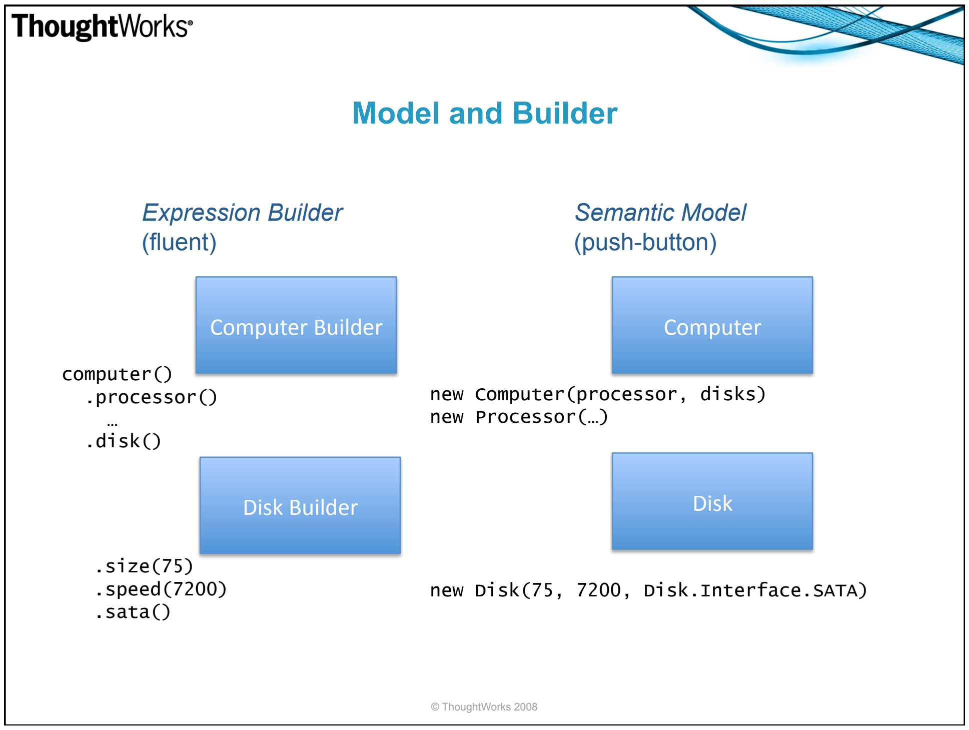
Martin Fowler

http://martinfowler.com/dslwip

ThoughtWorks®

Typical Internal DSL

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



ThoughtWorks®

Model and Builder

Expression Builder (fluent)

Semantic Model (push-button)

Computer Builder

```
computer()
  .processor()
   …
  .disk()
```

Disk Builder

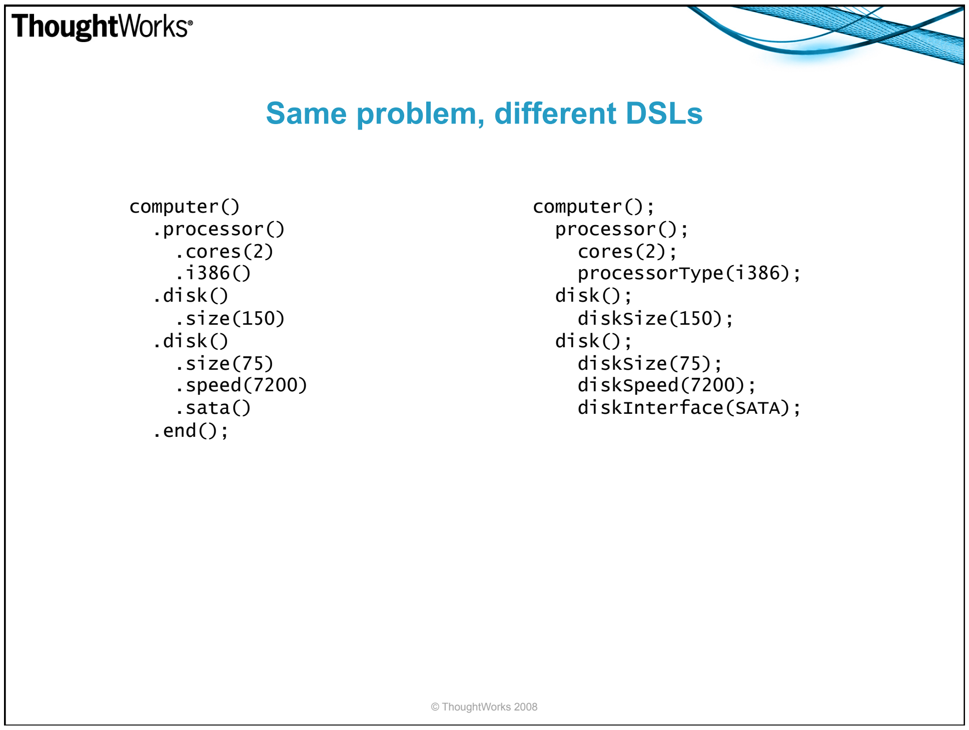
```
   .size(75)
   .speed(7200)
   .sata()
```

Computer

```
new Computer(processor, disks)
new Processor(…)
```

Disk

```
new Disk(75, 7200, Disk.Interface.SATA)
```

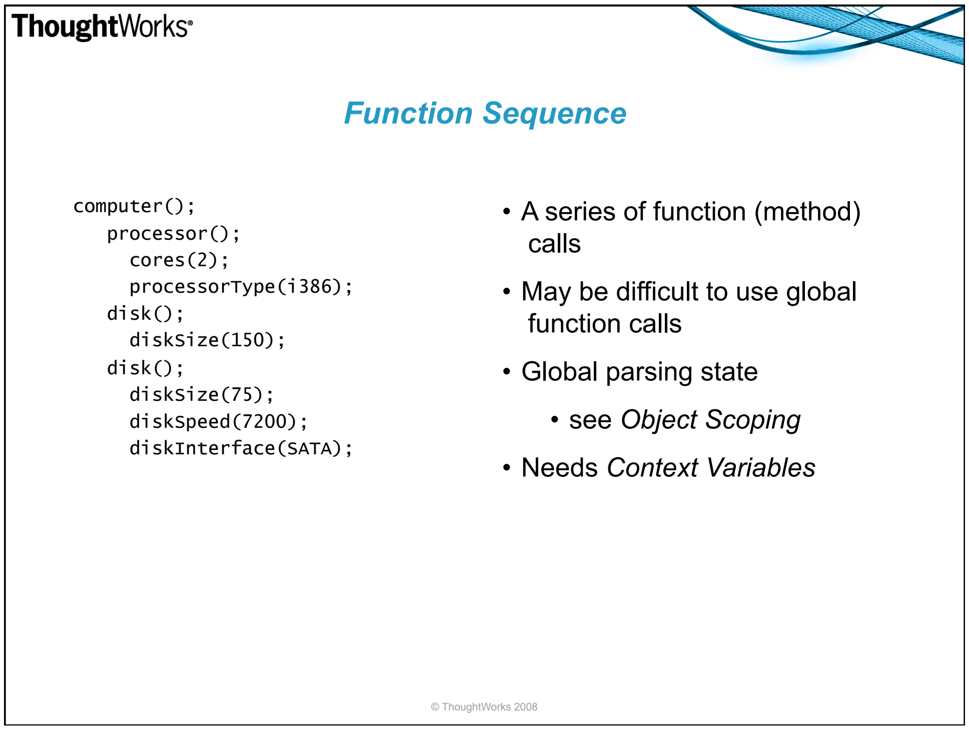


ThoughtWorks®

Same problem, different DSLs

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

```
computer();
  processor();
    cores(2);
    processorType(i386);
  disk();
    diskSize(150);
  disk();
    diskSize(75);
    diskSpeed(7200);
    diskInterface(SATA);
```

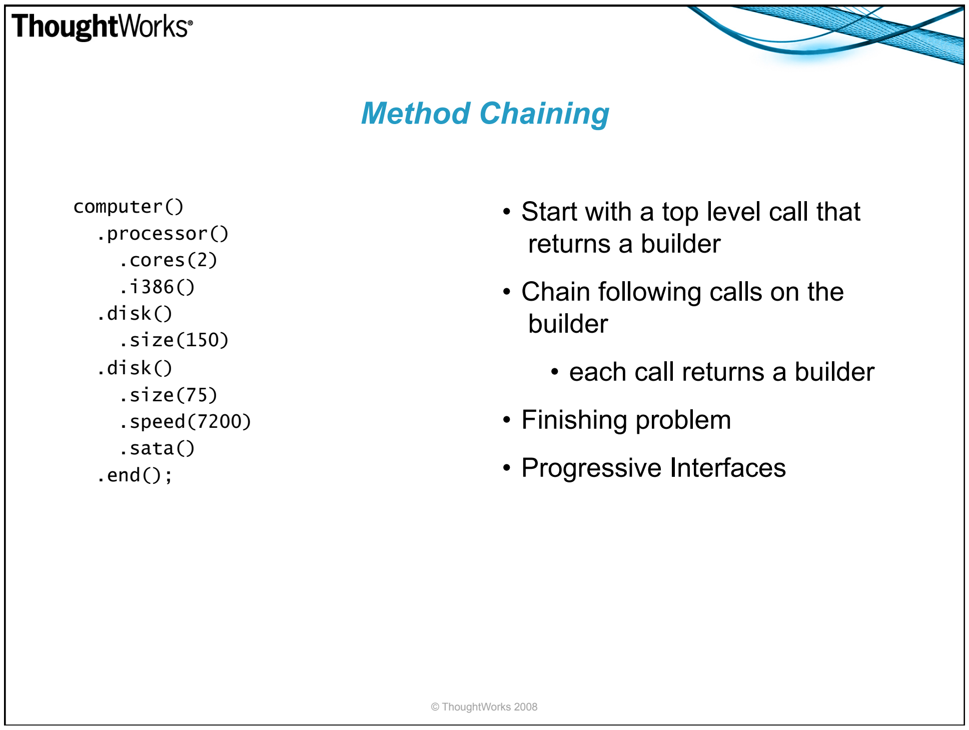


ThoughtWorks

Function Sequence

```
computer();
   processor();
     cores(2);
     processorType(i386);
   disk();
     diskSize(150);
   disk();
     diskSize(75);
     diskSpeed(7200);
     diskInterface(SATA);
```

- A series of function (method) calls
- May be difficult to use global function calls
- Global parsing state
  - see *Object Scoping*
- Needs *Context Variables*

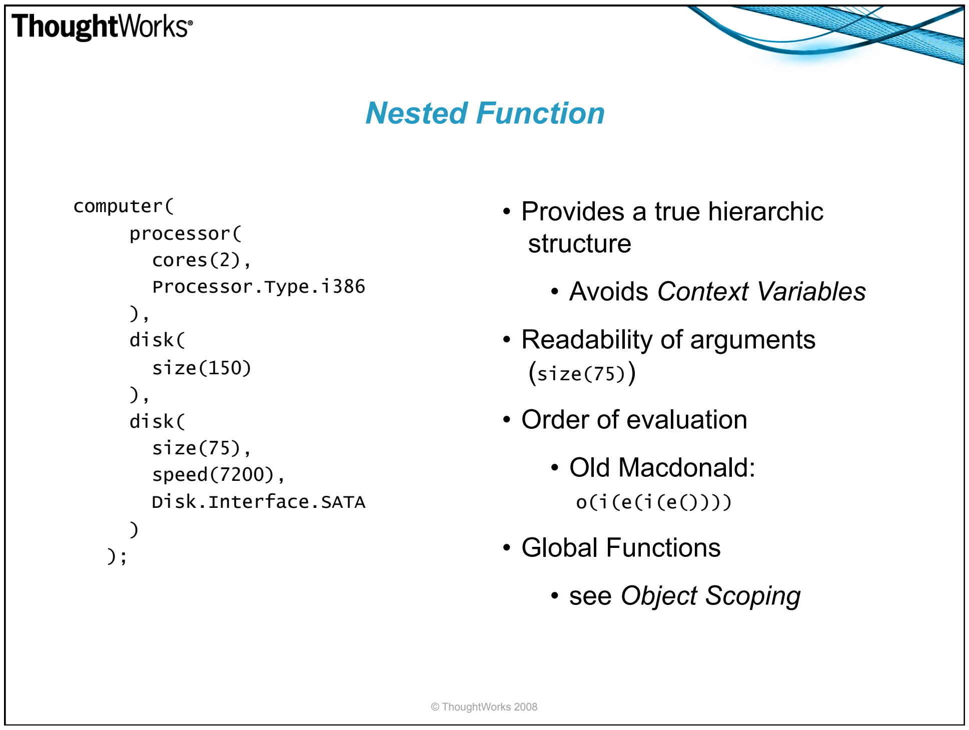


ThoughtWorks

Method Chaining

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

- Start with a top level call that returns a builder
- Chain following calls on the builder
  - each call returns a builder
- Finishing problem
- Progressive Interfaces

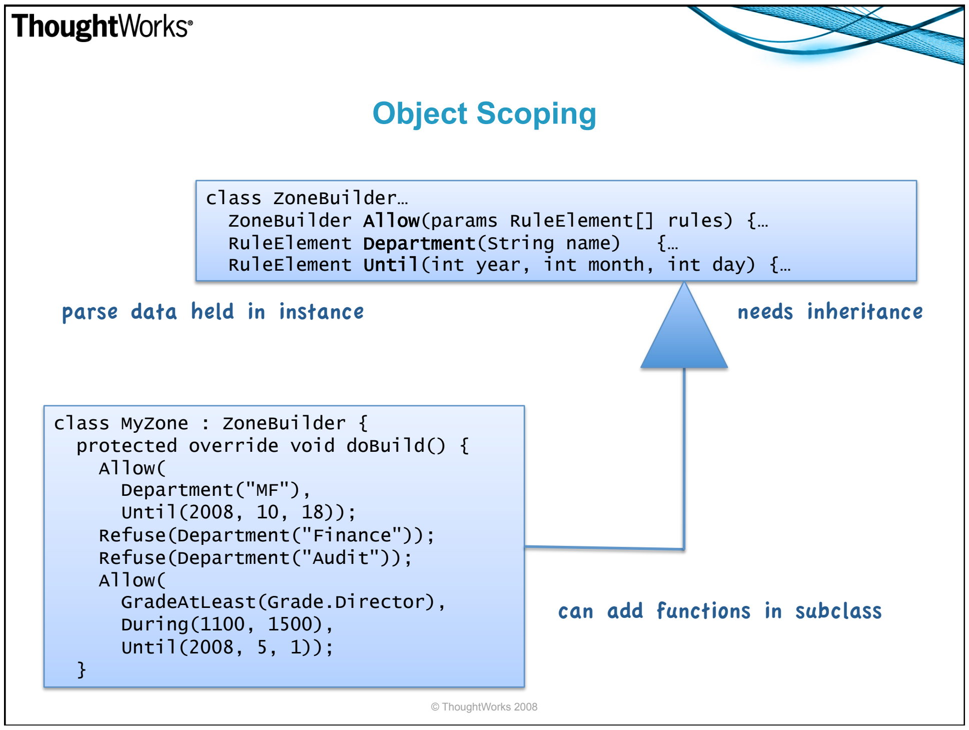


ThoughtWorks®

Nested Function

```
computer(
     processor(
       cores(2),
       Processor.Type.i386
     ),
     disk(
       size(150)
     ),
     disk(
       size(75),
       speed(7200),
       Disk.Interface.SATA
     )
   );
```

- Provides a true hierarchic structure
  - Avoids *Context Variables*
- Readability of arguments (`size(75)`)
- Order of evaluation
  - Old Macdonald: `o(i(e(i(e()))))`
- Global Functions
  - see *Object Scoping*



ThoughtWorks®

Object Scoping

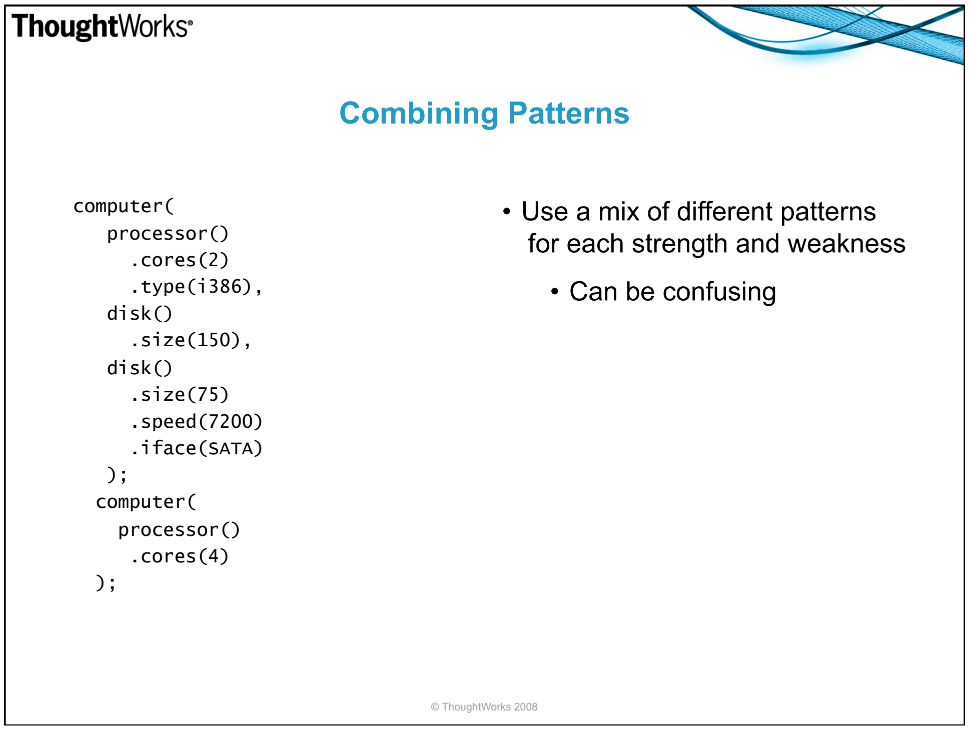
```
class ZoneBuilder…
  ZoneBuilder Allow(params RuleElement[] rules) {…
  RuleElement Department(String name)    {…
  RuleElement Until(int year, int month, int day) {…
```

parse data held in instance

needs inheritance

```
class MyZone : ZoneBuilder {
  protected override void doBuild() {
    Allow(
      Department("MF"),
      Until(2008, 10, 18));
    Refuse(Department("Finance"));
    Refuse(Department("Audit"));
    Allow(
      GradeAtLeast(Grade.Director),
      During(1100, 1500),
      Until(2008, 5, 1));
  }
```

can add functions in subclass



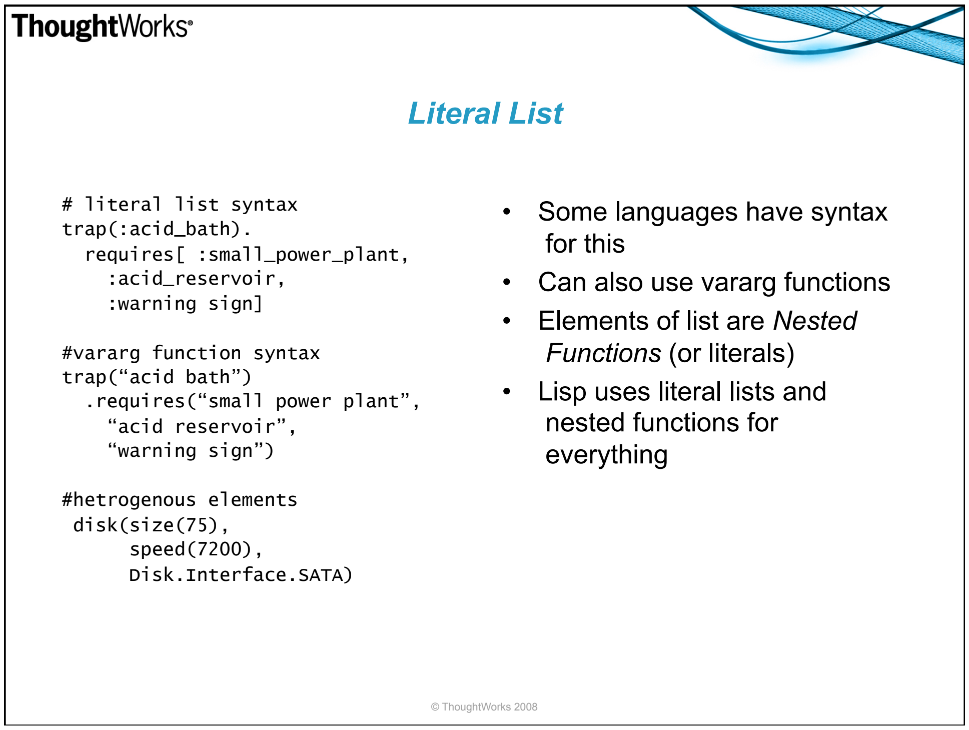
ThoughtWorks®

Combining Patterns

```
computer(
   processor()
     .cores(2)
     .type(i386),
   disk()
     .size(150),
   disk()
     .size(75)
     .speed(7200)
     .iface(SATA)
   );
  computer(
    processor()
     .cores(4)
  );
```

- Use a mix of different patterns for each strength and weakness
  - Can be confusing

© ThoughtWorks 2008



ThoughtWorks®

Literal List

```
# literal list syntax
trap(:acid_bath).
  requires[ :small_power_plant,
    :acid_reservoir,
    :warning sign]

#vararg function syntax
trap("acid bath")
  .requires("small power plant",
    "acid reservoir",
    "warning sign")

#hetrogenous elements
 disk(size(75),
      speed(7200),
      Disk.Interface.SATA)
```

- Some languages have syntax for this
- Can also use vararg functions
- Elements of list are *Nested Functions* (or literals)
- Lisp uses literal lists and nested functions for everything

© ThoughtWorks 2008

ThoughtWorks

Literal Map

```
processor {:cores => 2,
  :type => :i386,
  :speed => 2.2}
```

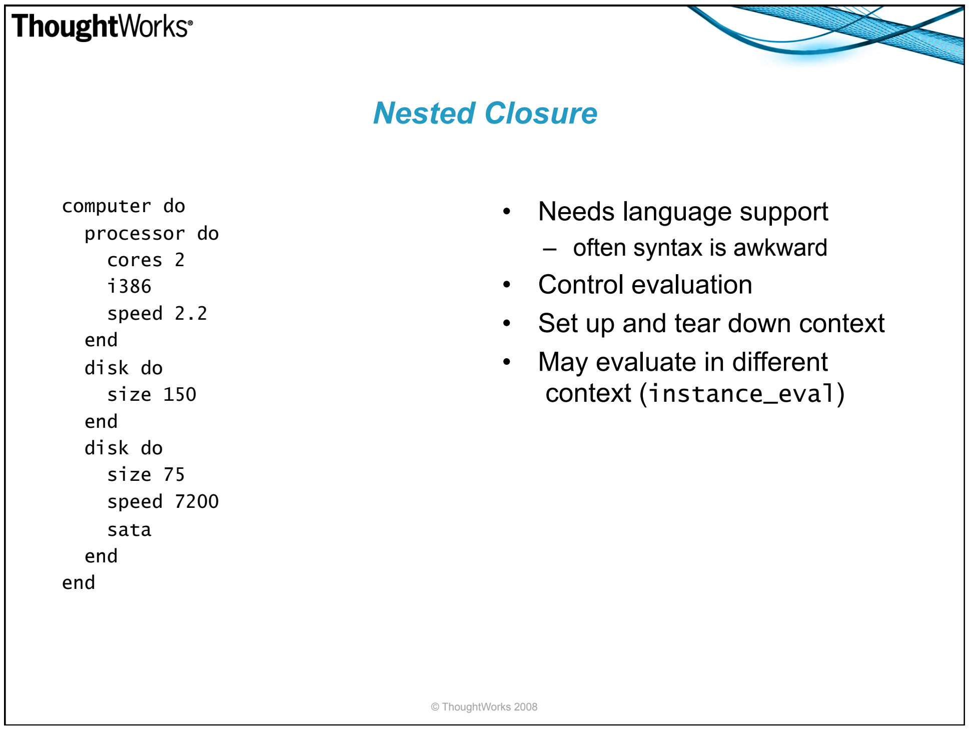
- Some languages have syntax for this
- Can also use named parameters
- Values are literals or *Nested Functions*

ThoughtWorks

Grammar and Patterns

	BNF	Consider…
mandatory list	parent ::= first second	*Nested Function*
optional list	parent ::= first maybeSecond? maybeThird?	*Literal Map* *Method Chaining†*
homogenous bag	parent ::= child*	*Literal List* *Function Sequence‡*
heterogeneous bag	parent ::= (this | that | theOther)*	*Method Chaining*
set	n/a (as bag but only one of each)	*Literal Map* *Method Chaining†*

† check for one of each
‡ at top level

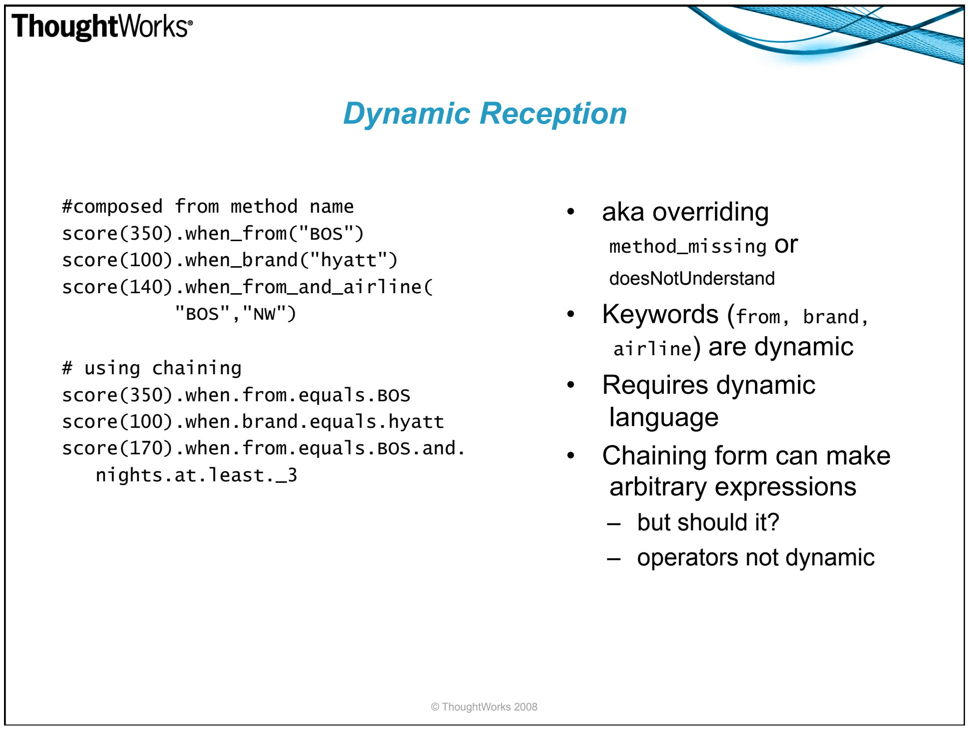


ThoughtWorks®

Nested Closure

```
computer do
  processor do
    cores 2
    i386
    speed 2.2
  end
  disk do
    size 150
  end
  disk do
    size 75
    speed 7200
    sata
  end
end
```

- Needs language support
  - often syntax is awkward
- Control evaluation
- Set up and tear down context
- May evaluate in different context (`instance_eval`)

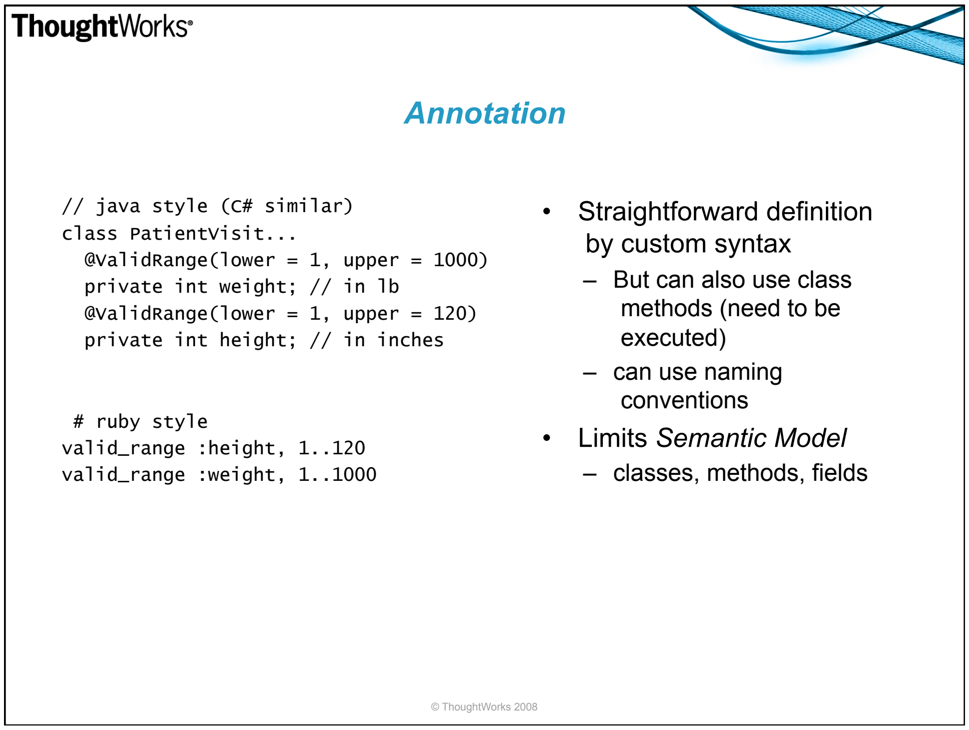


ThoughtWorks®

Dynamic Reception

```
#composed from method name
score(350).when_from("BOS")
score(100).when_brand("hyatt")
score(140).when_from_and_airline(
          "BOS","NW")

# using chaining
score(350).when.from.equals.BOS
score(100).when.brand.equals.hyatt
score(170).when.from.equals.BOS.and.
   nights.at.least._3
```

- aka overriding `method_missing` or doesNotUnderstand
- Keywords (`from`, `brand`, `airline`) are dynamic
- Requires dynamic language
- Chaining form can make arbitrary expressions
  - but should it?
  - operators not dynamic

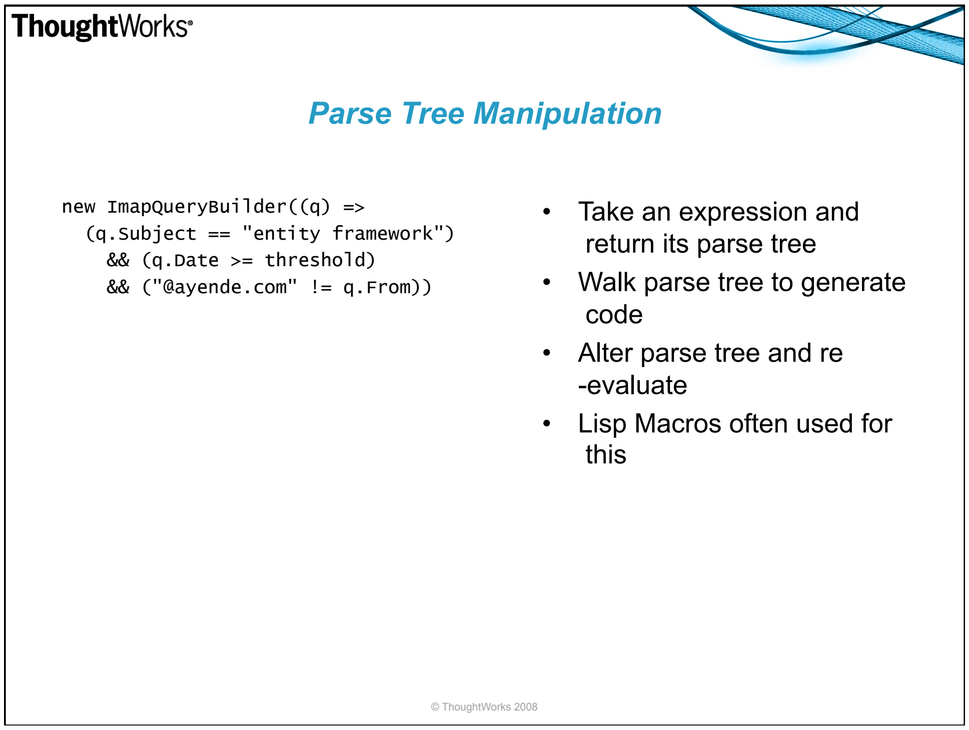


ThoughtWorks®

Annotation

```
// java style (C# similar)
class PatientVisit...
  @ValidRange(lower = 1, upper = 1000)
  private int weight; // in lb
  @ValidRange(lower = 1, upper = 120)
  private int height; // in inches


 # ruby style
valid_range :height, 1..120
valid_range :weight, 1..1000
```

- Straightforward definition by custom syntax
  - But can also use class methods (need to be executed)
  - can use naming conventions
- Limits *Semantic Model*
  - classes, methods, fields

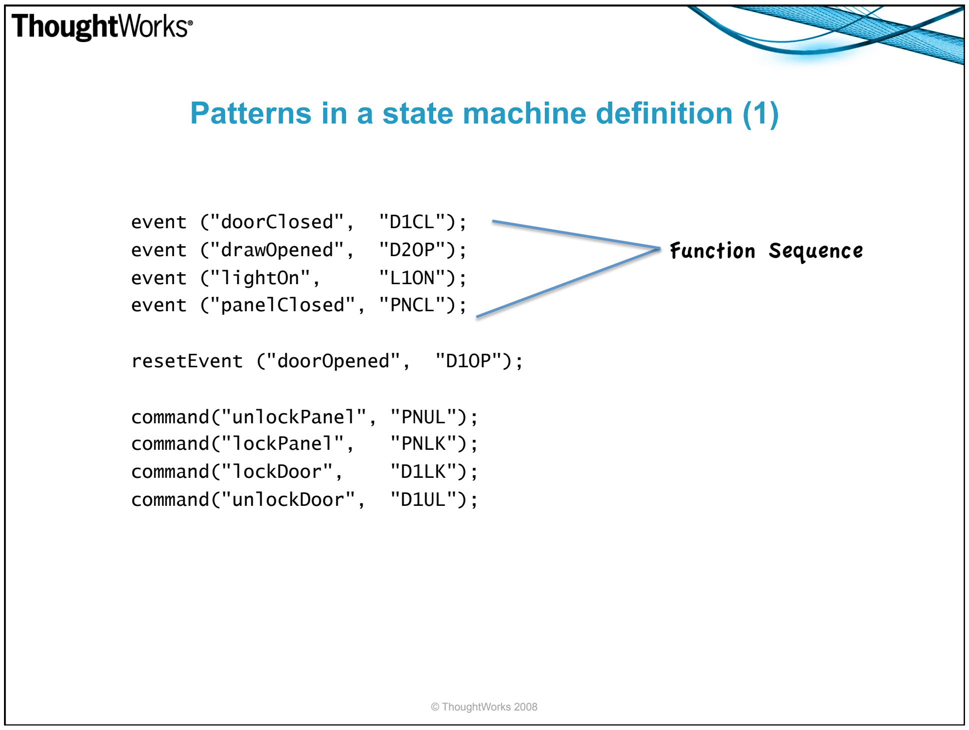


ThoughtWorks®

Parse Tree Manipulation

```
new ImapQueryBuilder((q) =>
  (q.Subject == "entity framework")
    && (q.Date >= threshold)
    && ("@ayende.com" != q.From))
```

- Take an expression and return its parse tree
- Walk parse tree to generate code
- Alter parse tree and re -evaluate
- Lisp Macros often used for this



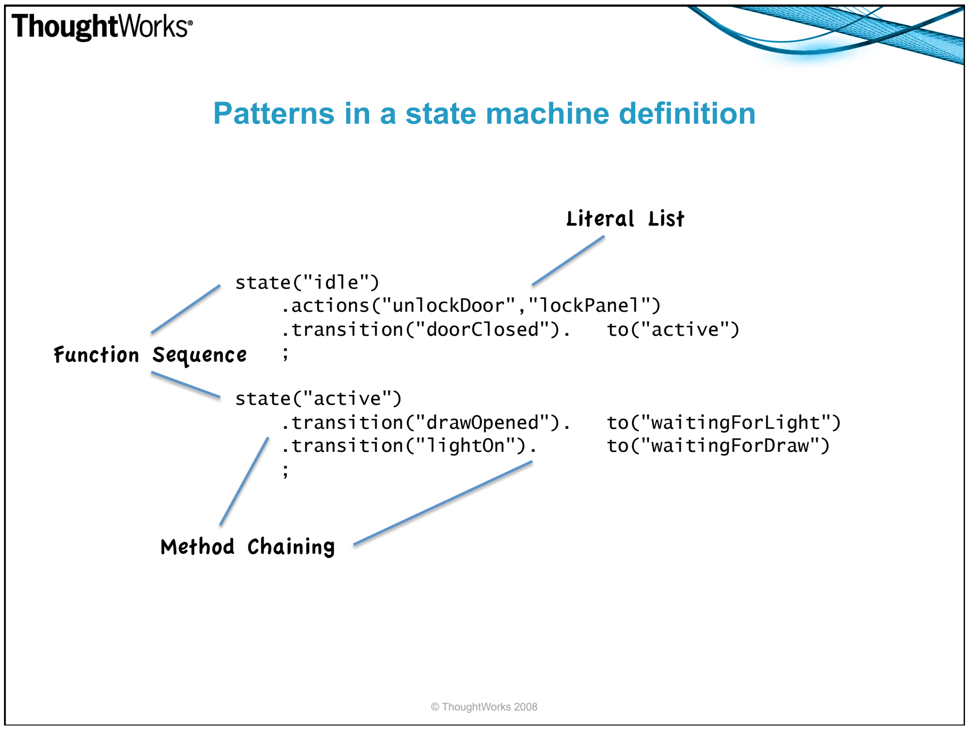
ThoughtWorks

Patterns in a state machine definition (1)

```
event ("doorClosed",  "D1CL");
event ("drawOpened",  "D2OP");
event ("lightOn",     "L1ON");
event ("panelClosed", "PNCL");

resetEvent ("doorOpened",  "D1OP");

command("unlockPanel", "PNUL");
command("lockPanel",   "PNLK");
command("lockDoor",    "D1LK");
command("unlockDoor",  "D1UL");
```

Function Sequence

© ThoughtWorks 2008



ThoughtWorks

Patterns in a state machine definition

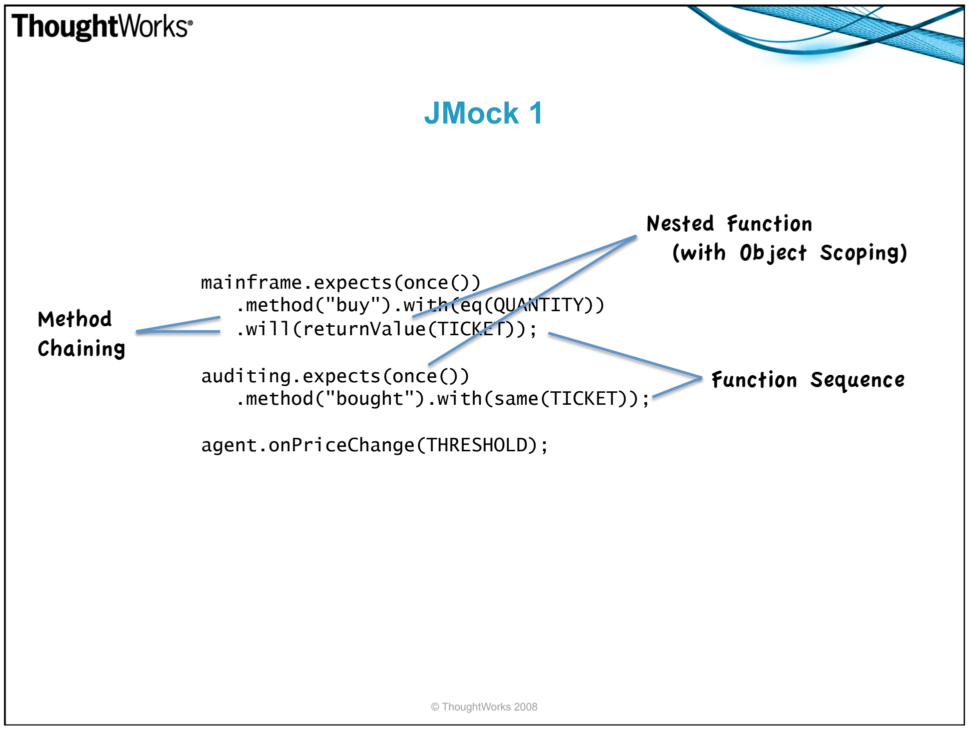
Literal List

```
state("idle")
    .actions("unlockDoor","lockPanel")
    .transition("doorClosed").   to("active")
    ;

state("active")
    .transition("drawOpened").   to("waitingForLight")
    .transition("lightOn").      to("waitingForDraw")
    ;
```

Function Sequence

Method Chaining

© ThoughtWorks 2008



ThoughtWorks

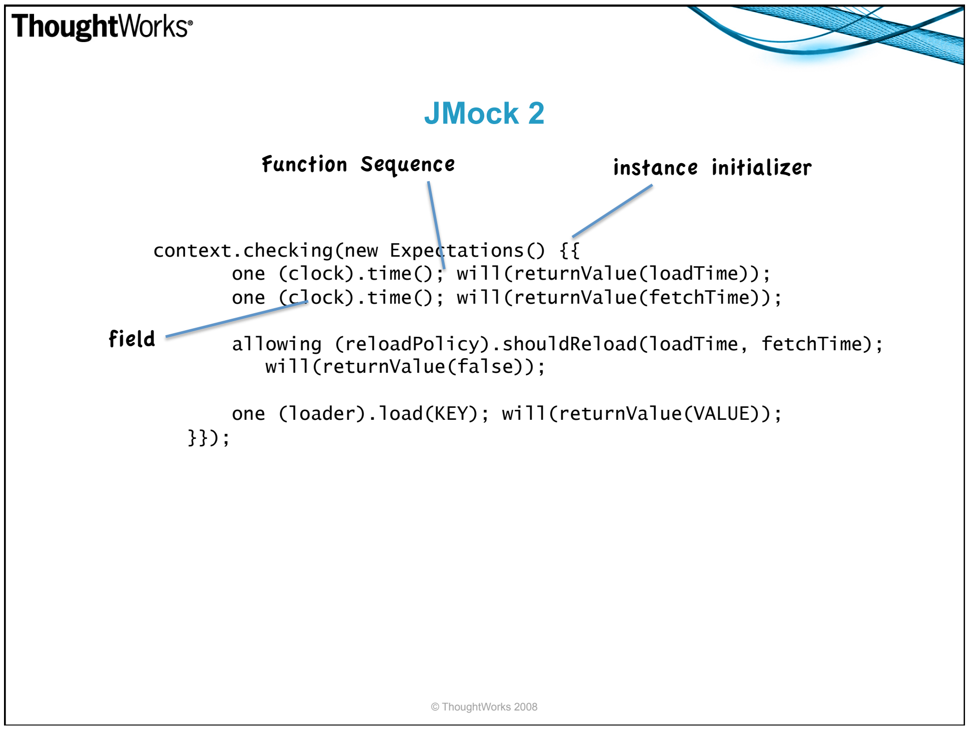
JMock 1

```
mainframe.expects(once())
   .method("buy").with(eq(QUANTITY))
   .will(returnValue(TICKET));

auditing.expects(once())
   .method("bought").with(same(TICKET));

agent.onPriceChange(THRESHOLD);
```

Nested Function (with Object Scoping)

Method Chaining

Function Sequence



ThoughtWorks

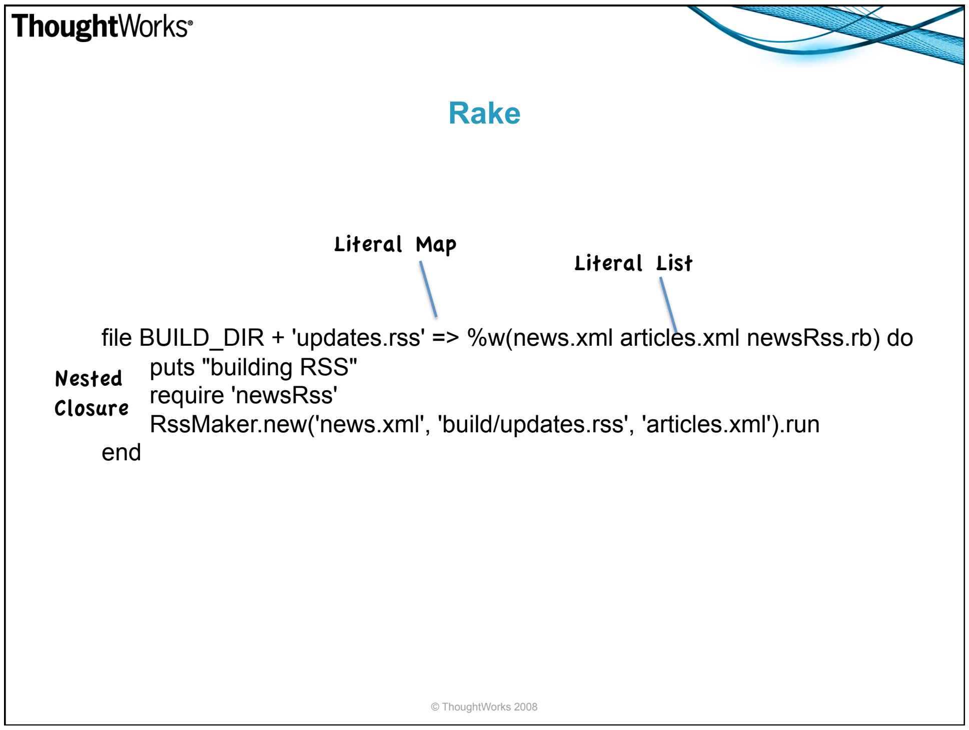
JMock 2

Function Sequence

instance initializer

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

       allowing (reloadPolicy).shouldReload(loadTime, fetchTime);
          will(returnValue(false));

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

field



ThoughtWorks®

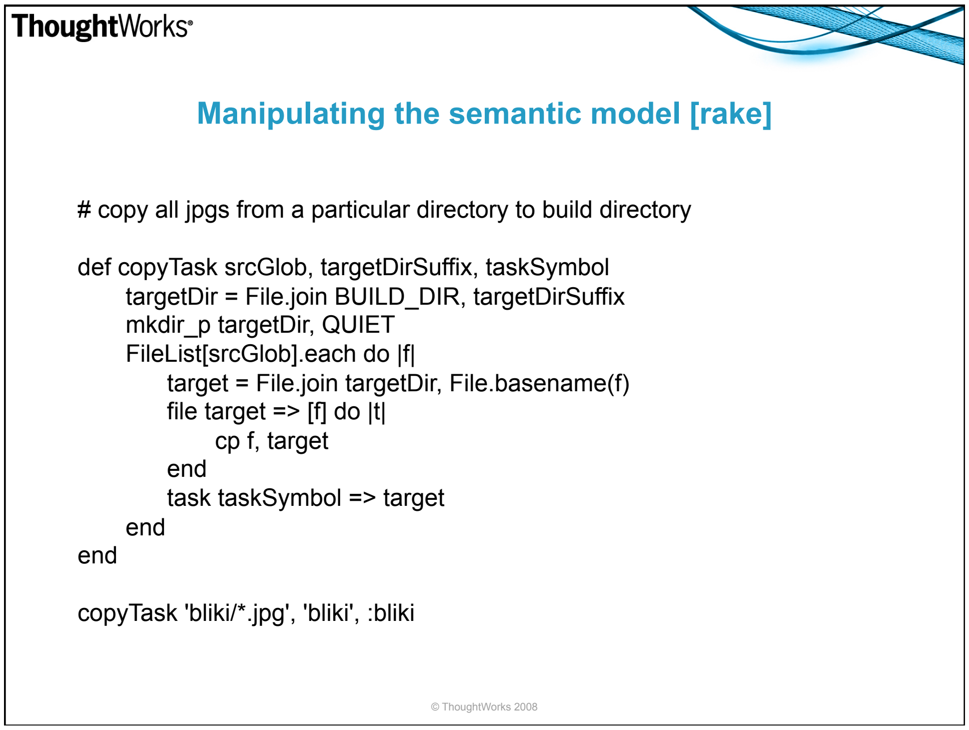
Rake

Literal Map

Literal List

```
file BUILD_DIR + 'updates.rss' => %w(news.xml articles.xml newsRss.rb) do
        puts "building RSS"
        require 'newsRss'
        RssMaker.new('news.xml', 'build/updates.rss', 'articles.xml').run
end
```

Nested Closure



ThoughtWorks®

Manipulating the semantic model [rake]

```
# copy all jpgs from a particular directory to build directory

def copyTask srcGlob, targetDirSuffix, taskSymbol
    targetDir = File.join BUILD_DIR, targetDirSuffix
    mkdir_p targetDir, QUIET
    FileList[srcGlob].each do |f|
        target = File.join targetDir, File.basename(f)
        file target => [f] do |t|
            cp f, target
        end
        task taskSymbol => target
    end
end

copyTask 'bliki/*.jpg', 'bliki', :bliki
```