RULES FEST

A Domain-Specific Language and Rule Engine for Python

Michael Joseph Walsh
mjwalsh{nospam}mitre.org

Engineer

The MITRE Corporation

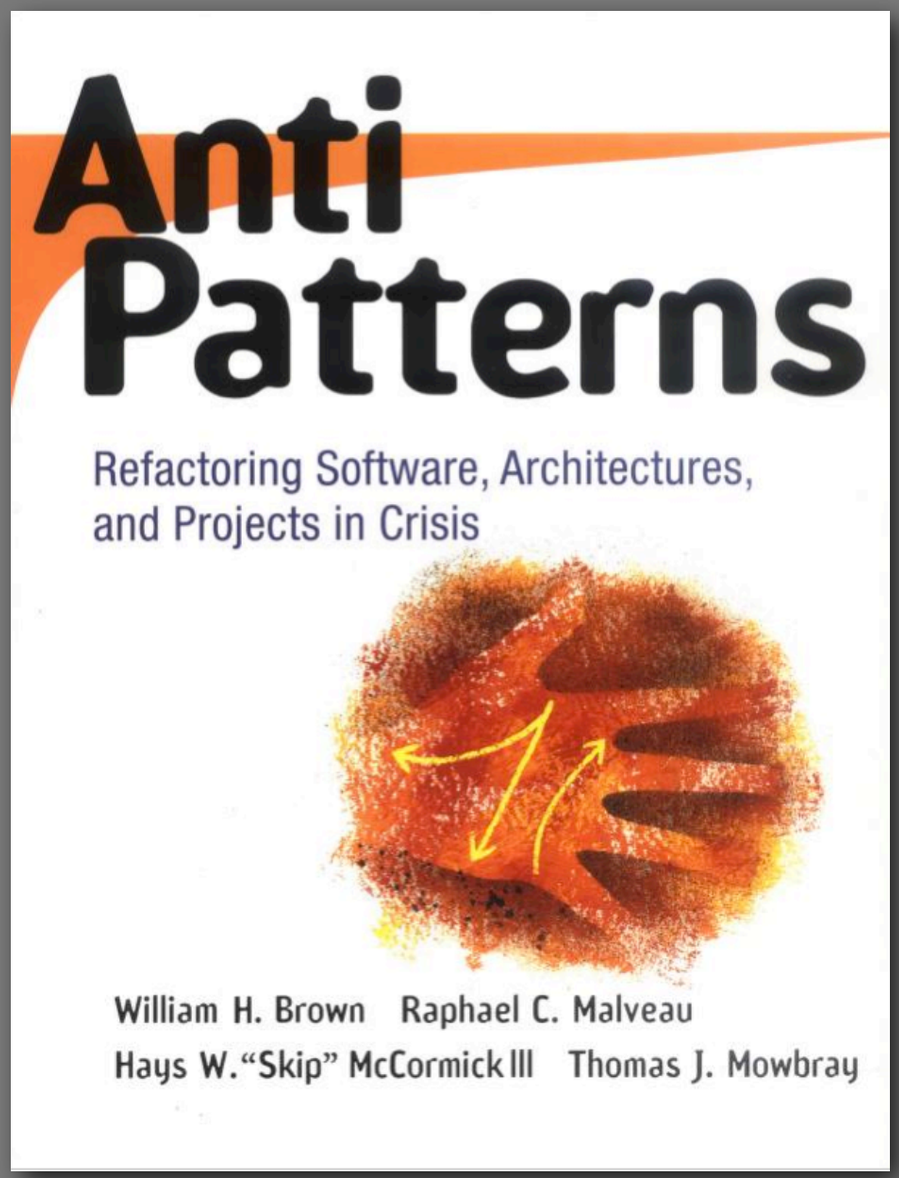
about

the MITRE corporation

about me

the backstory

Design and development philosophy



AntiPatterns

|'antē 'patərns|

a pattern that tells how to go from a problem to a bad solution

“Selling a Product You Can't Realize”

promising you’ll be able to deliver a marvelous product that you know in advance you will be incapable of developing.

Already being on the hook to deliver a marvelous rules engine that would solve all the projects problems, I didn’t want to find out later I wasn't capable of delivering what I had promised.

"Reinventing The Wheel"

"Go ahead. Grab that chisel and start pounding. You know you want to."

I'm not spoiling anything by saying I rolled my own rules engine, but in doing so I didn't want to unnecessarily create one when others already existed.

“Design for Sake of Design”

where you get caught up making a beautiful design, and forget your end result must be useful and doable in a finite time.

I simply couldn’t get caught caught up in a lengthy design with a finite amount of time and user base to be concerned with.

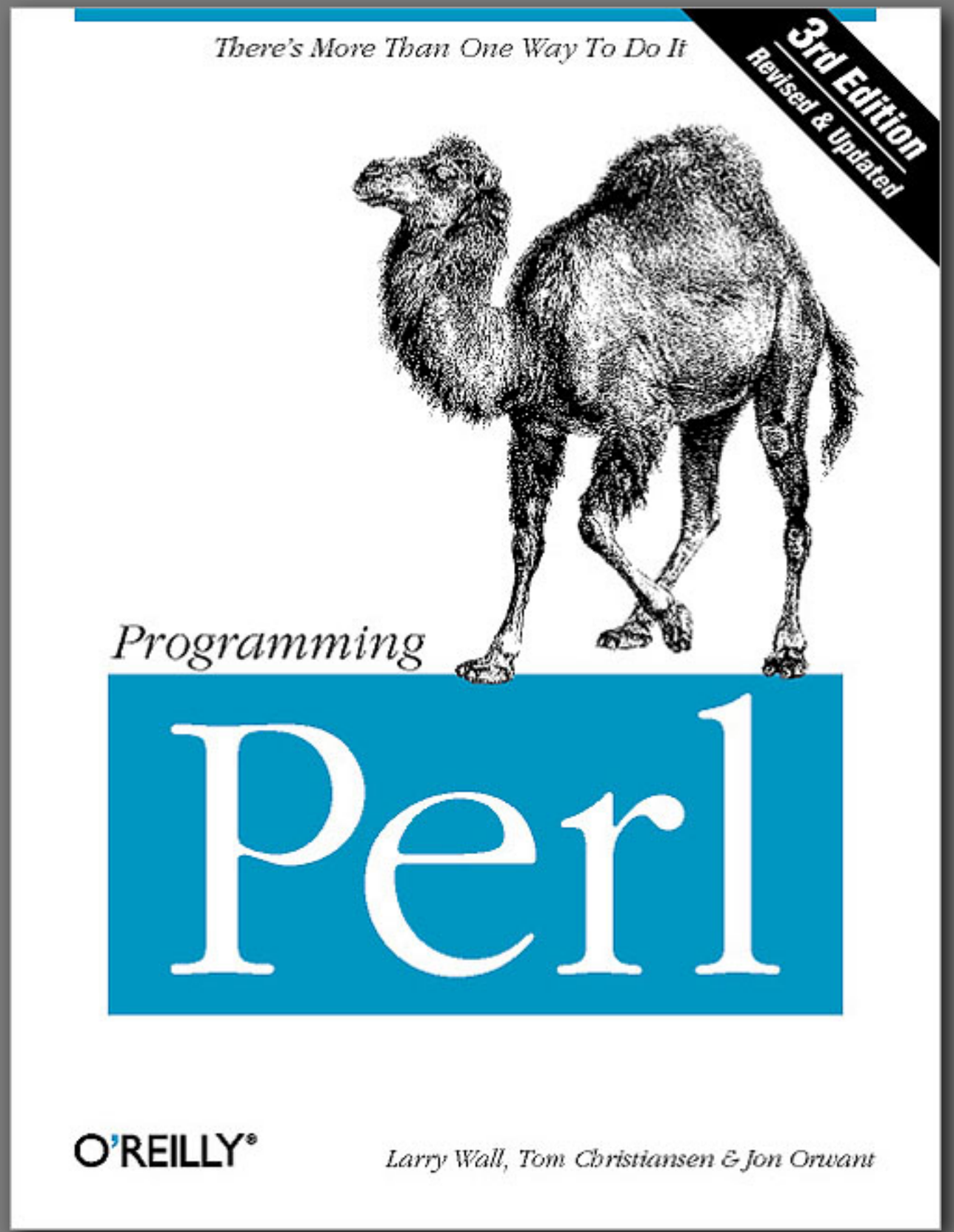
“Voodoo Chicken Programming”

Just wave a dead chicken at the code.

Not having programmed in either Python or ANTLR3, I had a significant learning curved to surmount. I wanted to avoid cobbling something together with the risk of having little understanding of how it worked.

I want to avoid

false laziness,
false impatience,
and false hubris



laziness

absorbing the cost of doing it right up front, so those coming behind you can be saved time

impatience

the anger you have for when things aren't done right

hubris

Excessive pride, the sort of thing Zeus zaps you for. The quality that makes you author what other people won't want to say bad things about.

The “false”, inverts of these, can lead you into making poor choices as to whether

you should build new software, reuse,
or just get on with the damn job, and write something easy

what's a rules engine?

it provides a form of artificial intelligence, an

intellect | ˈintlˌekt |

the faculty of reasoning and understanding

consisting of a working memory retaining

knowledge |ˈnälij|

relevant facts and information

and a set of

rules | rools |

explicit or understood principles governing conduct within a particular activity or sphere

each rule has an optional

condition |kən'diSHən|

boolean evaluations on the state of something

and a suite of one or more actions

action | ˈakSHən |

process of doing something,
typically to achieve an aim

these actions either

further direct the behavior of the system, and/or further inform the system

the engine starts with some

facts |fakts|

truths known about past or present circumstances

uses rules to

infer |in'fər|

deduce or conclude information from evidence and reasoning rather than from explicit statements

more facts

these

facts | fakts |

truths known about past or present circumstances

fire more rules, that

infer |ɪnˈfər|

deduce or conclude information from evidence and reasoning rather than from explicit statements

infer yet more

facts | fakts |

truths known about past or present circumstances

and so on.

Setting out

Facts should be Python objects

object-oriented programming

| ˈäbjəkt ˈôrēˌənt ˈprōˌgramiNG |

a paradigm of using “objects”, data structures consisting of data fields and methods together with their interactions -- to design applications.

Decorator pattern

|ˈdekəˌrātər ˈpatərn|

is a design pattern in object-oriented programming that allows behavior to be added to an existing object dynamically.

"Python is not Java."

Rules should be written in a manner that can be easily

grokked | gräked |

understood intuitively

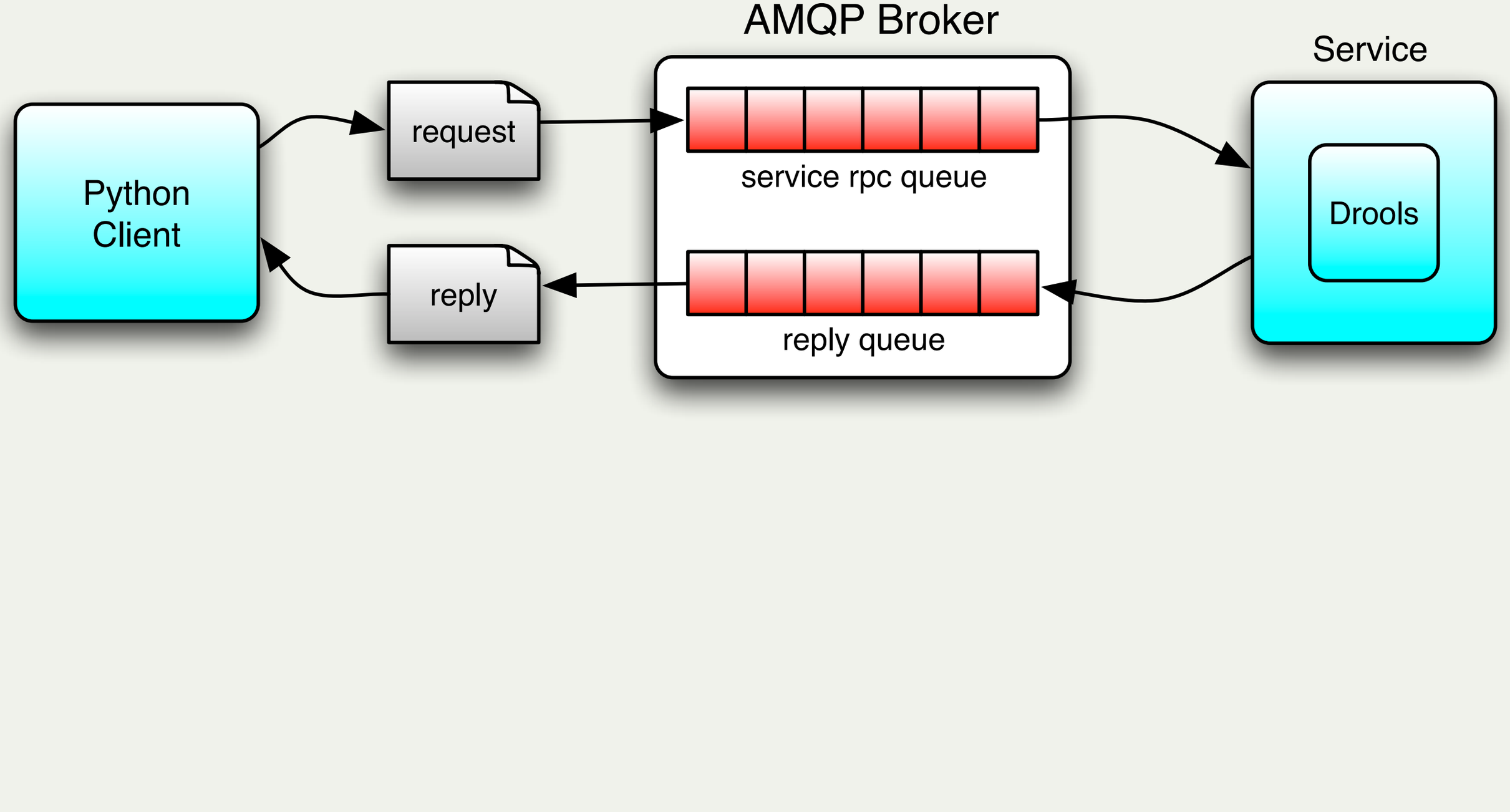
I set out looking for a Python-based Open Source Rules Engine to reuse

PyKE

Pychinko

Fuxi |foo-shee|

Could we use Drools?


AMQP Broker
Service
request
Python
Client
service rpc queue
Drools
reply
reply queue

Could we author our own DSL and Rules Engine in Python?

My selection criteria came down to these musts

- target to Python
- be actively maintained
- have an active user base
- be open sourced under a favorable license
- literature supporting,
- tutorials for, and
- a Tool chain the would fit with
  - my workflow, and
  - be multi-platform

ANother Tool for Language Recognition, version 3

ANTL3 allows you to author a

grammar | ˈgramər |

a set of rules which describes how a stream of text defines the syntax of a language

and generate a recognizer

both a

lexer | ˈlekər |

breaks up an input stream into tokens

and a

parser | ˈpärsər |

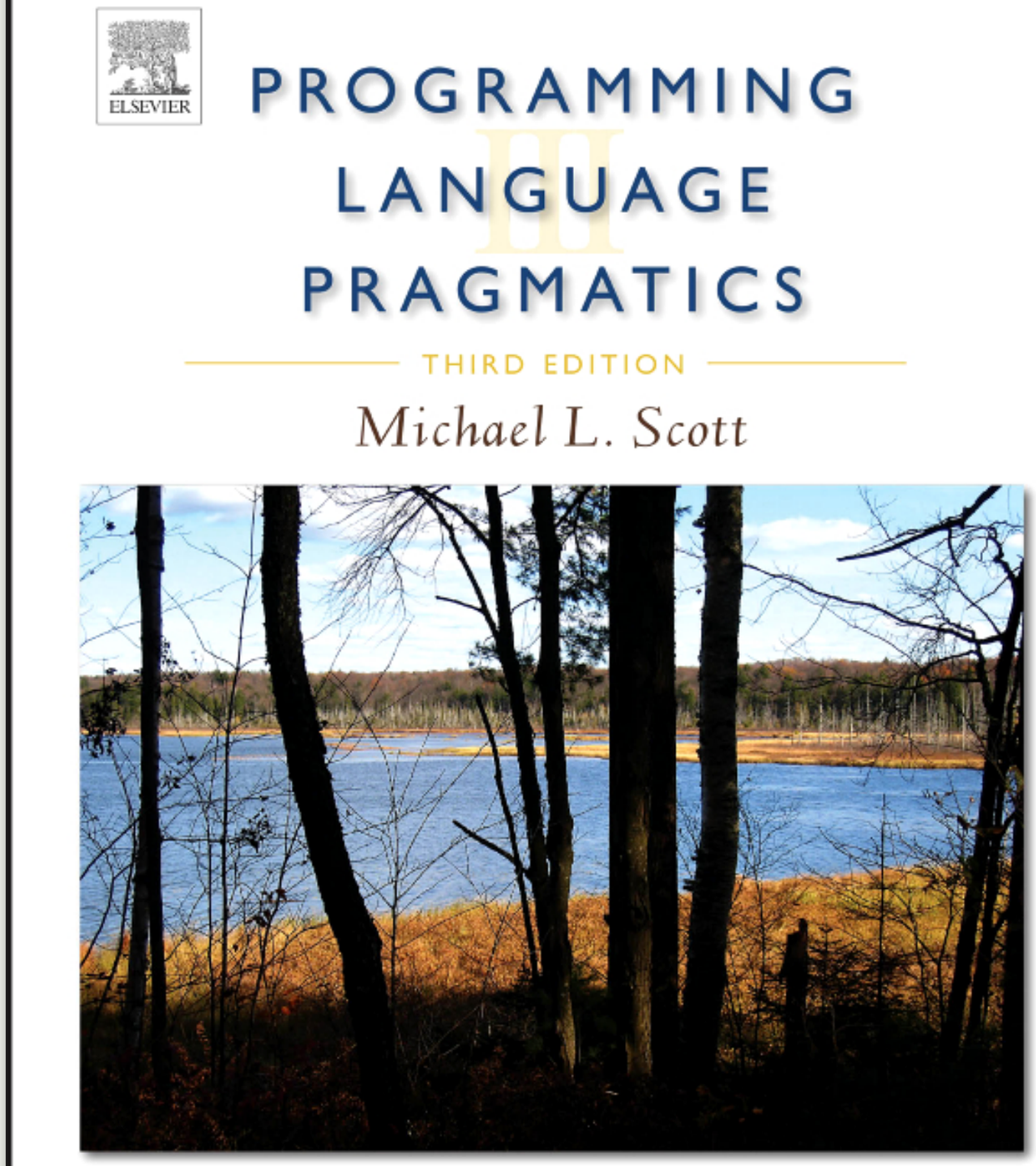
feeding off the lexer's token stream attempting to recognize structure either immediately emitting output or constructs an internal data structure

The Pragmatic Programmers

The Definitive ANTLR Reference

Building Domain-Specific Languages

Terence Parr


ELSEVIER
PROGRAMMING
LANGUAGE
PRAGMATICS
THIRD EDITION
Michael L. Scott
MK
MORGAN KAUFMANN

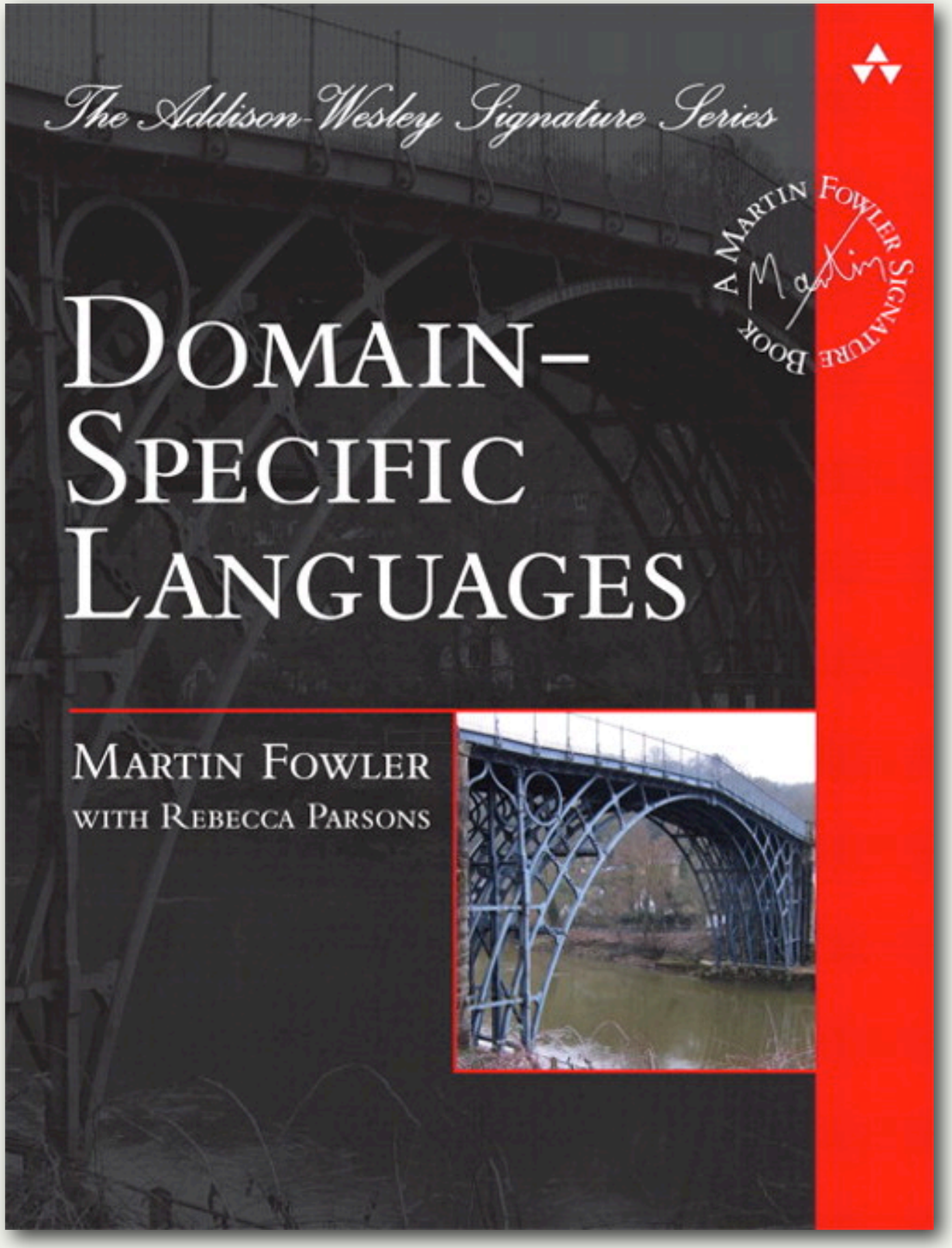
The Pragmatic Programmers

Language Implementation Patterns

Create Your Own Domain-Specific and General Programming Languages

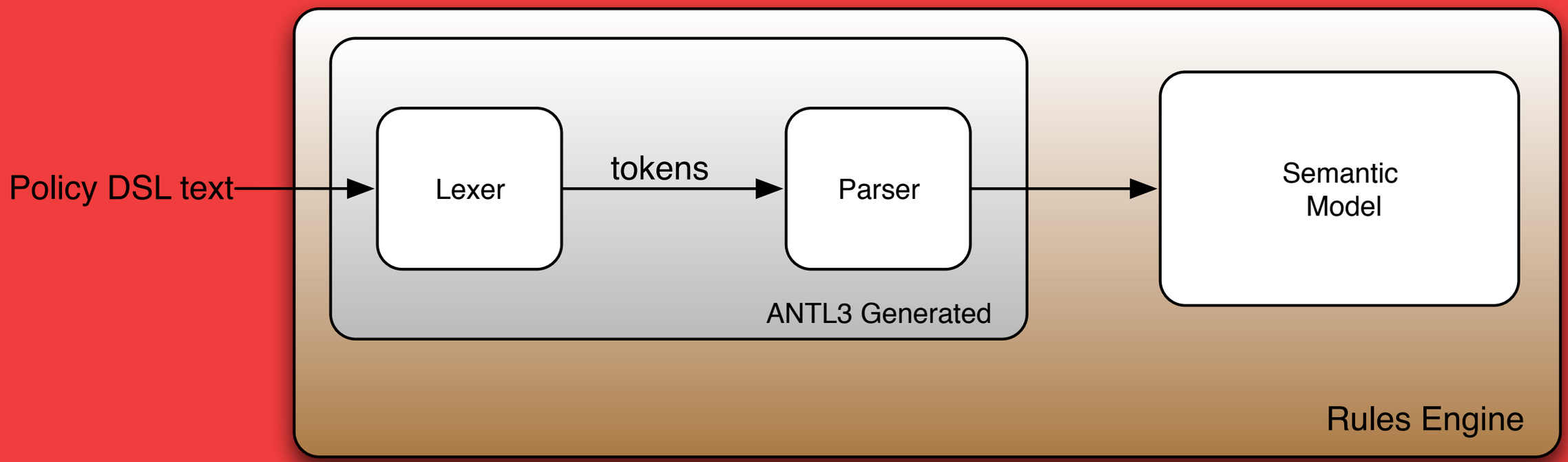
Terence Parr

Edited by Susannah Davidson Pfalzer


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Domain-Specific Languages
Martin Fowler
with Rebecca Parsons

Scott Stanchfield's ANTLR 3.x Video Tutorial

http://javadude.com/articles/antlr3xtut/


Policy DSL text
Lexer
tokens
Parser
ANTL3 Generated
Semantic
Model
Rules Engine

The Grammar

```
tokens {
    INDENT;
    DEDENT;
}
```

```
file
    : ( NEWLINE | statement )+
    | EOF
    ;
```

```
statement
    : importStmt
    | attributeStmt
    | ruleStmt
    ;
```

```
importStmt
   : importName | importFrom
   ;

importName
   : IMPORT dottedAsNames NEWLINE
   ;

importFrom
   : FROM dottedName IMPORT ( importAsNames |
     LPAREN importAsNames RPAREN) NEWLINE
   ;
```

```
attributeStmt
    : expressionStmt
    ;
```

```
ruleStmt
    : RULE id COLON NEWLINE
          INDENT ( ruleAttribute )*
                 ( when )?
                 then DEDENT
    ;
```

```
when
    : WHEN COLON NEWLINE
          INDENT ( ruleCondition )? DEDENT
    ;
```

```
then
   : THEN COLON NEWLINE
         INDENT ( action )+ DEDENT
   ;
```

```
action
    : attributeAction
    | forgetAction
    | learnAction
    | modifyAction
    | haltAction
    | simpleStmt
    ;
```

```
simpleStmt
   : expressionStmt
   | printStmt
   ;

attributeAction
   : ATTRIBUTE expressionStmt
   ;

forgetAction
   : ( FORGET | DELETE ) OBJECTBINDING NEWLINE
   ;

learnAction
   : ( LEARN | INSERT ) NAME LPAREN ( argumentList )? RPAREN  NEWLINE
   ;

modifyAction
   : MODIFY OBJECTBINDING COLON NEWLINE
       INDENT ( propertyAssignment )+ DEDENT
   ;

haltAction
   : HALT NEWLINE
   ;
```

Semantic Model

```
ruleStmt returns [object] // returns a RuleStmt object.
  : RULE id COLON NEWLINE { $object = RuleStmt([ $RULE.text,
                                                  $id.objec$COLON.text ],
                                                $RULE.getLine(),
                                                $RULE.getCharPositionInLine() ) }
      INDENT ( ruleAttribute
                    { $object.append_child( $ruleAttribute.object ) } )*
             ( when { $object.append_child( $when.object ) } )?
             then { $object.append_child( $then.object ) }  DEDENT
```

The Rule Engine

```
import logging, os
from antlr3 import FileStream, CommonTokeStream, ANTRLStringStream
from intellect.grammar.PolicyParser import PolicyParser
from intellect.Callable import Callable

class Intellect(object):
    def __init__(self):...

    @property
    def knowledge(self):...

    @knowledge.setter
    def knowledge(self, oneOrMoreObjects):...

    @property
    def policy(self):...

    @policy.setter
    def policy(self, value):...

    def learn(self, identifier):...

    def forget(self,  identifier):...

    def forget_all(self):...

    def reason(self, agenda = None):...

    @Callable
    def log(self, msg, name = "intellect", level logging.DEBUG):...
```

A simple example tying it all together

Baa, baa, black sheep,
Have you any wool?
Yes sir, yes sir,
Three bags full.
One for the master,
One for the dame,
And one for the little boy
Who lives down the lane.

Facts

```
import logging, time
import thread, random
from threading import Lock

def grow_wool(sheep):...

class BlackSheep():

    number = 0

    def __init__(self):
        '''
        BlackSheep Initializer
        '''
        self.bags_of_wool = 0
        BlackSheep.number = BlackSheep.number + 1
        self._name = "Sheep #{0}".format(BlackSheep.number)

        self.lock = Lock()
        thread.start_new_thread(grow_wool, (self,))

    @property
    def name(self):
        return self._name

    @property
    def bags_of_wool(self):
        value = self._bags_of_wool
        return value

    @bags_of_wool.setter
    def bags_of_wool(self, value):
        self.lock.acquire()
        self._bags_of_wool = value
        self.lock.release()
```

```
class BagOfWool(object):
    '''
        Used to signify a bag of wool
    '''

    def __init__(self):
        '''
        BagsOfWool Initializer
        '''
```

```
class BuyOrder(object):
    '''
    Used to signify a buy order
    '''

    def __init__(self, count = 1):
        '''
        BuyOrder Initializer
        '''
        self.count = count

    @property
    def count(self):
        return self._count

    @count.setter
    def count(self, value):
        self._count = value
```

Policy

```
from intellect.examples.rulesfest.BlackSheep import BlackSheep
from intellect.examples.rulesfest.BuyOrder import BuyOrder
import logging, random

rule "Start policy":
   then:
      log("Policy is being reasoned over.", "example", logging.DEBUG)

rule "Baa, baa, black sheep, Have you any wool?":
   when:
      exists BlackSheep()
   then:
      print( "Me:  Baa, baa, black sheep, Have you any wool?" )
```

```
rule "Have you no bags full?":
    when:
        $sheep := BlackSheep( bags_of_wool == 0 )
    then:
        print( "{0}: No, sir, No, sir.".format( $sheep.name ) )

rule "At least 1 bag full?":
    when:
        $sheep := BlackSheep( bags_of_wool >= 1 )
    then:
        print( "{0}: Yes, sir, yes, sir,".format( $sheep.name ) )
        print( "{0}: {1} bags full".format( $sheep.name, $sheep.bags_of_wool ) )
        print( "{0}: One for my master,".format( $sheep.name ) )

rule "At least 2 bags full?":
    when:
        $sheep := BlackSheep( bags_of_wool >= 2 )
    then:
        print("{0}: And one for my dame,".format( $sheep.name ) )

rule "3 bags full?":
    when:
        $sheep := BlackSheep( bags_of_wool == 3 )
    then:
        print( "{0}: and one for the little boy".format( $sheep.name ) )
        print( "{0}: Who lives down the lane.".format( $sheep.name ) )
```

```
rule "Retire the sheep and insert a buy order":
    when:
        $sheep := BlackSheep( bags_of_wool == 3 )
    then:
        print( "Retiring {0}.".format( $sheep.name ) )
        forget $sheep
        insert BuyOrder(random.randint(1, 1))

rule "Time to buy new sheep?":
    when:
        $buyOrder := BuyOrder( )
    then:
        print( "Buying a new sheep." )
        modify $buyOrder:
            count = $buyOrder.count - 1
        learn BlackSheep()

rule "Remove empty buy orders":
    when:
        $buyOrder := BuyOrder( count == 0 )
    then:
        forget $buyOrder

rule "End policy":
    then:
        log("Finished reasoning over policy.", "example", logging.DEBUG)
        halt
```

Rules Engine

```
import sys, logging, time
from intellect.Intellect import Intellect
from intellect.Intellect import Callable
from intellect.examples.rulesfest.BuyOrder import BuyOrder

class MyIntellect(Intellect):
    pass

myIntellect = MyIntellect()

logging.getLogger("example").debug("asking the engine to learn my policy")
myIntellect.learn("./rulesets/example.policy")

#print myIntellect.policy.str_tree("semantic model:")

logging.getLogger("example").debug("asking the engine to learn about BuyOrder")
myIntellect.learn(BuyOrder())


myIntellect.reason()

while True:
    logging.getLogger("example").debug("{0} in knowledge.".format(myIntellect.knowledge)
    time.sleep(5)
    logging.getLogger("example").debug("messaging reasoning engine to reason")
   myIntellect.reason()
```

Output

```
DEBUG   creating reasoning engine
DEBUG   asking the engine to learn my policy
DEBUG   asking the engine to learn about BuyOrder
DEBUG   __main__.MyIntellect :: Policy is being reasoned over.

Buying a new sheep.

DEBUG   __main__.MyIntellect :: Finished reasoning over policy.

DEBUG   [<intellect.examples.rulesfest.BlackSheep.BlackSheep instance at
0x110807f80>] in knowledge.
DEBUG   Sheep #1: adding some wool
DEBUG   messaging reasoning engine to reason
DEBUG   __main__.MyIntellect :: Policy is being reasoned over.

Me:  Baa, baa, black sheep, Have you any wool?

Sheep #1: Yes, sir, yes, sir,
Sheep #1: 1 bags full
Sheep #1: One for my master,

DEBUG   __main__.MyIntellect :: Finished reasoning over policy.
```

```
DEBUG   [<intellect.examples.rulesfest.BlackSheep.BlackSheep instance at
0x110807f80>] in knowledge.
DEBUG   Sheep #1: adding some wool
DEBUG   Sheep #1: wait around for retirement
DEBUG   messaging reasoning engine to reason
DEBUG   __main__.MyIntellect :: Policy is being reasoned over.

Me:  Baa, baa, black sheep, Have you any wool?

Sheep #1: Yes, sir, yes, sir,
Sheep #1: 3 bags full
Sheep #1: One for my master,
Sheep #1: And one for my dame,
Sheep #1: and one for the little boy
Sheep #1: Who lives down the lane.

Retiring Sheep #1.

Buying a new sheep.
```

Outcome

http://github.com/nemonik/Intellect

http://pypi.python.org/pypi/Intellect/

Questions?