Domain Specific Languages and Python

Safe Hammad
Python Northwest
21st April 2011
What is a Domain Specific Language (DSL)?

“A programming or declarative language dedicated to a particular problem domain.”

Typically the language will be engineered to contain the bare minimum of constructs and consist of keywords familiar to users knowledgeable in the problem domain.

c.f. General purpose language e.g. Python
Why do I want a DSL?

**Domain Specific Language:**
The elements of the language are closely aligned to concepts in the problem domain making it:

- Natural (and hopefully easy) for a user to represent domain specific data.
- Quick to grok data represented in the DSL.
Why do I want a DSL (continued)?

**Domain Specific Language:**

- To represent data or express a problem more cleanly than an existing and perhaps more general method of representation, for example, XML.

- Full control of the constructs in the language give the creator of the DSL the freedom to simplify that language as much as possible.
Yikes!

<!-- Only allow logins from 9am to 5pm -->
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
      <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
    </Apply>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">
      09:00:00
    </AttributeValue>
  </Apply>
</Condition>

<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
    <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
      AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
  </Apply>
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">
    17:00:00
  </AttributeValue>
</Condition>
"Programs must be written for people to read, and only incidentally for machines to execute."
- Abelson & Sussman, SICP
Better!

LOGIN ALLOWED 09:00 TO 17:00
What about POP (Plain Ol' Python)?

```python
login_allowed(start='09:00', end='17:00')
```
What about POP (Plain Ol' Python)?

```python
login_allowed(start='09:00', end='17:00')
```

c.f. LOGIN ALLOWED 09:00 TO 17:00
Plain Ol' Python as a DSL - Pros

- It's clearer than many other languages!
- We don't have to reinvent the wheel.
- It prevents the proliferation of YALs (yet another language).
- It's easy to code the functionality behind it. For example `login_allowed()` can be a simple function which sets up access control accordingly.
Plain Ol' Python as a DSL - Cons

• It's clear to programmers but perhaps not so clear to the non-tecchy. It's full of unneeded brackets, colons and quotes. Perhaps Ruby would be a better choice! *[Insert flames here]*

• The syntax is a little unforgiving. Miss out a bracket or colon at your peril!
Other ready to roll languages

So if we don't want to use a general purpose programming language, how about something more suited to representing data structures. e.g. YAML (Yet Another Markup Language):

```
login allowed:
- start : 09:00
- end   : 17:00
```
Other ready to roll languages

So if we don't want to use a general purpose programming language, how about something more suited to representing data structures. e.g. YAML (Yet Another Markup Language):

```
login allowed:
  - start : 09:00
  - end   : 17:00

  c.f. LOGIN ALLOWED 09:00 TO 17:00
```
YAML - Pros

- YAML allows you to define arbitrarily deep data structures (like XML and JSON but with less fluff).
- It's a well defined language with a forgiving syntax and a good off-the-shelf parser.

*Disclaimer*: I don't tend to use YAML in my day to day life, but it seems to make sense, particularly when the DSL is designed to hide the wordiness of XML.
YAML - Cons

- You're still bound by the syntax of YAML. (Yet More Colons And Dashes).
- You miss a small opportunity to take that extra step to make the language both specific and natural for the domain.
I'm convinced. So what next?

So you've created a DSL, now you need to parse it. Options include:

- **PyParsing** – An easy to use Pythonic parser where the grammar is written directly in Python.
- **PLY** – A more traditional Python parser based on `lex` and `yacc`.
- **import re** – Compact and easy to use for simple languages.

Interestingly, all of the above can (almost) be considered to implement their own DSLs.
from pyparsing import Word, nums

parser = "LOGIN ALLOWED" + \
    Word(nums) + ":" + Word(nums) + \
    "TO" + \
    Word(nums) + ":" + Word(nums)

dsl = "LOGIN ALLOWED 09:00 TO 17:00"
print parser.parseString(dsl)

# ['LOGIN ALLOWED', '09', '':', '00', 'TO', 
#  '17', '':', '00']
import re

parser = re.compile(r'LOGIN\s+ALLOWED\s+'
                     r'([^\s]+):([^\s]+)\s+'
                     r'\s+TO\s+'
                     r'([^\s]+):([^\s]+)\s')

dsl = "LOGIN ALLOWED 09:00 TO 17:00"
print parser.match(dsl).groups()

# ('09', '00', '17', '00')
And finally ...

Think carefully about whether you a DSL is suitable for your non tecchy users. Perhaps what you really need is a GUI!
re.compile(r'\'THANKS!\'')

Safe Hammad
http://safehammad.com
@safehammad