Reasonable Python or how to Integrate F-Logic into an Object-Oriented Scripting Language

Markus Schatten
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, Varaždin, HR-42000, Croatia
markus.schatten@foi.hr

***Abstract*— Python is an object-oriented scripting language known for its ability to support various programming paradigms. In Python one can write procedural, functional, object-oriented, and thanks to metaclasses even aspect-oriented code. Even if some efforts were done to support the last major programming paradigm, logic programming is still not supported in a Python programmer friendly way.**

In this paper a solution that aims on this target using F-Logic (particularly FLORA-2), which syntax is much more compatible with the Python language than traditional Prolog syntax, is presented. In order to make such an integration useful ZODB (Zope Object Base) is used to facilitate permanent storage of Python objects, while the FLORA-2 engine built on XSB is used for reasoning facilities.

To take advantage of logic programming concepts like facts, rules, variables or queries, special logical Python object are introduced. In the end some examples of usage are shown and future development guidelines are given.

I. Introduction

Python is a well known dynamic object-oriented scripting language initially implemented by Guido van Rossum for the Amoeba operating system, later published on USENET and made public available. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types (like lists, tuples, dictionaries etc.), classes, and metaclasses. The language comes with a large standard library that covers areas such as string processing (regular expressions, Unicode, calculating differences between files etc.), operating system interfaces (system calls, filesystems, TCP/IP sockets etc.), Internet protocols (HTTP, FTP, SMTP, XML-RPC, POP, IMAP, CGI programming etc.), and software engineering (unit testing, logging, profiling, parsing Python code etc.) [7].

One of the great advantages of the Python programming language is the fact that it supports various programming paradigms like procedural, functional, object-oriented or even aspect oriented programming. Even if there were some interesting attempts [5] [1] [2] to introduce the logic programming paradigm into Python due to incompatibilities to traditional Prolog syntax this task is still not achieved.

In [5] Petuello and later Zagrodnick, created a Python interface to SWI Prolog called PyLog. It allows interaction between Python and Prolog code, loading of Prolog programs, issuing queries and the useage of query results. The impossibility to use Python objects in an intuitive way is a major drawback of this interface.

In [1] and [2] Berger and Coelho, respectively, propose a metaprogramming approach to introduce logic programming into Python. Berger uses Pythons method override facilities to provide a more intuitive way to produce Prolog-like code in Python. These solutions still lacks the possibilities to completely integrate logic programming into Python since logic programming concepts are in a certain way separated from the rest of the 'normal' Python code. They also build upon traditional Prolog syntax which seems not to be the optimal solution for an object-oriented programming language like Python.

As opposed to traditional Prolog syntax frame logic based languages [3] seem to be much more compatible with Python syntax. Especially the object-oriented knowledge base language FLORA-2 [10] has very interesting (but rather coincidental) similarities in its syntax to the Python syntax. Additionally FLORA-2 integrates F-Logic, HiLog and Transaction Logic which makes it a powerful knowledge representation and inference language. In [6] Yang, Kifer and Zhao argue that FLORA-2 "is a flexible and natural framework that combines rule-based and object-oriented paradigms".

Another system to mention here is the Zope Objectbase (ZODB) developed by the Zope Corporation [12]. It's a simple object-oriented database which has the capabilities to store Python objects in a permanent way. However it lacks any reasoning facilities since all object are stored in a rather large Python dictionary. By combining the reasoning engine of FLORA-2 and the capabilities of ZODB it is possible to use knowledge bases in Python, as argued further.

II. Introducing Logic Programming Concepts into Python

Most important logic programming concepts to introduce into Python addressed here are logic variables, facts, rules and queries. In FLORA-2 logic variables are denoted by any word starting with an uppercase letter or the underscore ('_'). A similar way is used in Python, by creating a class of objects named Variable which behavior is shown in the following Python interactive shell session.

```
>>> X = f.Variable( 'X' )
>>> X
<f.Variable instance at 0xb7dd854c>
>>> print X
```

1-4244-1148-3/07/$25.00 ©2007 IEEE.

```
X
>>> X == 27
>>> print X
X = 27
```

When analyzing facts, rules and queries we can conclude that they are all instances of a construct which has the form:

$$\texttt{Head} : -\texttt{Body}. \tag{1}$$

$$\_ : -\texttt{Body}. \tag{2}$$

$$\texttt{Head} : -\_. \tag{3}$$

(1) is a rule where Head is the rule head, and `Body` is the rule body (denoted by `Head :- Body.`). (2) is a query where `Body` is the query string (denoted by `?- Body.`). (3) is a fact where `Head` is the actual fact (denoted by `Head.`). We can conclude that we need only one class when introducing these concepts into Python which behavior is shown in the following interactive shell session (where Bogus is a logical class as argued further).

```
>>> x = f.Construct()
>>> print x

>>> h, b1, b2 = f.Bogus(), f.Bogus(),
f.Bogus()
>>> x & h
>>> x.__class__
<class f.Fact at 0xb7ae356c>
>>> del x
>>> x = f.Construct()
>>> x << b1 & b2
>>> x.__class__
<class f.Query at 0xb7ae365c>
>>> del x
>>> x = f.Construct()
>>> ( x & h ) << b1 | b2
>>> x.__class__
<class f.Rule at 0xb7ac5dac>
```

One can see that x which is an instance of the Construct class dynamically changes its class to Fact, Rule or Query according to its content (e. g. its head and body). The bitwise shift ('<<'), the bitwise or ('|') and the bitwise and ('&') were overridden in order to allow logic programming constructs, and stand for ':-', ';' and ',' respectively. This means that a logic programming construct like:

$$\texttt{X} : -(\texttt{Y}, \texttt{Z}); \texttt{W}. \tag{4}$$

in Python would be written like:

$$\texttt{X} << (\texttt{Y}\&\texttt{Z})|\texttt{W}. \tag{5}$$

Which is similar to both FLORA-2 and Python syntax. In addition to these two classes some additional classes were defined to ease such behavior.

III. Translating Python Objects into F-molecules

Another task to be accomplished is the translation of Python objects into F-molecules in FLORA-2 syntax in order to allow reasoning about them. Python has a feature which eases this translation and can be seen in the following interactive shell session.

```
>>> class a:
...  def __init__( self, b1 = 1, b2 = 2 ):
...    self.b1 = b1
...    self.b2 = b2
...
>>> y = a()
>>> dir( y )
['__doc__', '__init__', '__module__', 'b1', 'b2']
>>> y.__dict__
'b1': 1, 'b2': 2
>>> y.__class__
<class __main__.a at 0xb7a27b9c>
```

We can conclude that Python objects allow us to query their internal structure and content. Using this feature a multiple recursive algorithm was developed which translates Python objects into FLORA-2 F-molecules. For example the y object from the example when translated into FLORA-2 would be:

```
>>> tr = py2f.py2f()
>>> tr.f2obj( y )
'py0xb7db4e2cpyobj____main____py__ :
pyapyclass____main____py__["b1"->1, "b2"->2]'
```

To have unique object names in our knowledge base we use the objects internal memory addresses and some additional information about its module. When an object is translated it is automatically stored in a ZOBD object base for later retrieval and permanent storage. Python objects which are not logical constructs as argued earlier, are considered to be facts.

IV. Connecting FLORA-2 and Python

In order to use the FLORA-2 engine from Python an interface had to be developed. FLORA-2 is build upon the OpenSource XSB Prolog engine which is developed in C and thus has a C interface [8]. The Simplified Wrapper and Interface Generator (SWIG) was used to create an interface between XSB and Python (but could be easily extended to any other language supported by SWIG i. e. AllegroCL, C# - Mono, C# - MS .NET, CFFI, CHICKEN, CLISP, Guile, Java, Lua, MzScheme, Ocaml, Perl, PHP, Ruby, or Tcl/Tk).

Through such an interface FLORA-2 is then loaded as a module into XSB. Additional wrapper classes were developed to ease communication between XSB and Python, and FLORA-2 and Python, as shown in the following interactive shell session.

```
>>> f = interface.Flora2()
[FLORA2 specific output]
>>> f.consult( 'test' )
[FLORA2 specific output]
>>> f.query( 'X:person[ Y -> Z ].', ['X',
```

```
'Y', 'Z'] )
[{'Y': 'age', 'X': 'mirko1', 'Z': '40'},{'Y':
'age', 'X': 'mirko2', 'Z': '42'}]
>>> f.close_query()
```

We can conclude that such an interface allows for using XSB and FLORA-2 engines from Python. Compiling and loading of Prolog and FLORA-2 specific programs is also supported. Queries can be issued directly whereas return variables have to be provided in a Python list as the second argument. The results of queries are Python lists which elements are Python dictionaries where each dictionary represents one solution. Keys of each dictionary are the provided return variables, and the matching values are the returned values from the knowledge base.

V. Putting it All Together

After describing the particular parts of this integration it is possible to connect all together in order to facilitate logic programming in Python. All previously described parts constitute a Python module. In addition to theses parts a new class was defined which integrates them. In particular this class represents an F-Logic base with storing and querying facilities.

Any object to be stored in to the knowledge base is first stored in the ZODB in order to be persistent. Afterwards it is translated into FLORA-2 syntax and loaded into the FLORA-2 engine. Additionally the FLORA-2 code is saved in an external file so the state can be restored later. This behavior of the system is transparent to the Python programmer.

To query the knowledge base one can either use FLORA-2 syntax or create special query objects using logical variables and logical constructs. In the following interactive shell session we first create a construct and define a logical class which we will use for querying the knowledge base.

```
>>> x = Construct()
>>> class query_object( Logical ):
...  def __init__( self, a, b ):
...    Logical.__init__( self )
...    self.a = a
...    self.b = b
```

Now we can create a query object and modify it to fit our needs. Note how it is possible to use a logical object to query for any object of any class by overriding its type and class type attributes with logical variables. We can also override the names of attributes by inserting a logical variable in the place of the attributes name.

```
>>> q = query_object( Variable( 'X' ),
Variable( 'Y' ) )
>>> x << q
>>> print x
?- py0xb7d6cdacpyobj____main____py__ :
pyquery_objectpyclass____main____py__[
"b"->Y, "a"->X].
>>> q.__type__ = Variable( 'Z' )
>>> print x
?- Z : pyquery_objectpyclass____main____py__[
"b"->Y, "a"->X].
>>> q.__classtype__ = Variable( 'W' )
>>> print x
?- Z:W[ "b"->Y, "a"->X].
>>> q.__dict__[ Variable( 'V' ) ] = q.__dict__[
'a' ]
>>> del q.__dict__[ 'a' ]
>>> print x
?- Z:W[ "b"->Y, V->X].
>>> del q.__dict__[ 'b' ]
>>> print x
?- Z:W[ V->X ].
```

To query the knowledge base some facts and/or rules have to be stored in it. A simple data object class which subclasses the Persistent class is created in the following.

The subclassing allows all instances of the class to be stored in the ZODB. Instances to be stored in the knowledge base are also created in this interactive shell session.

```
>>> class data_object( Persistent ):
...  def __init__( self, a, b ):
...    self.a = a
...    self.b = b
...
>>> d1 = data_object( 1, 2 )
>>> d2 = data_object( 3, 4 )
```

Now it is possible to create the knowledge base which is just an Python object like any other and insert the data objects.

```
>>> fb = FBase( 'my_flbase' )
[FLORA-2 specific output]
>>> fb.insert( d1 )
[FLORA-2 specific output]
>>> fb.insert( d2 )
[FLORA-2 specific output]
```

By using the previously created query object the following results are obtained. Note that the W variable returned the string 'class' which is due to the impossibility of ZODB to store class objects (e. g. they are not persistent).

```
>>> fb.query( x )
[{'X': '1', 'Z': <__main__.data_object
object at 0xb7dde96c>, 'W': 'class'},
{'X': '2', 'Z': <__main__.data_object object
at 0xb7dde96c>, 'W': 'class'}, {'X':
'3', 'Z': <__main__.data_object object at
0xb7c52aac>, 'W': 'class'}, {'X': '4', 'Z':
<__main__.data_object object at 0xb7c52aac>,
'W': 'class'}]
```

This simple example shows how transparent the knowledge base is to a Python programmer. The programmer just has to create an FBase object to store Python objects in the knowledgde base.

To query the knowledge base query objects have to be created. Using logic objects and logical variables this task is intuitive and fair easy to accomplish.

To add rules to the knowledge base one has to use constructs in order to create them and store them. These rules are similar to their Prolog and FLORA-2 counterparts with some minor syntax differences still preserving the flexibility of normal Python code.

VI. Conclusion and Future Work

In this paper an integration of F-logic, especially FLORA-2 and the dynamic object-oriented programming language Python was proposed. This integration showed some advantages to other attempts of supporting logic programming in the Python scripting language. These advantages include more intuitive syntax for Python programmers, complete integration and reasoning over Python objects, and permanent storage.

This integration could allow Python programmers to easier create knowledge base, ontology and semantic web solutions. Other possible usage would include automated applications generation, reasoning about module capabilities, and automated software testing.

Even if this solution is a step forward it has still to be developed further. Future development will include a more network oriented system which should allow multiple knowledge base users, client server architecture and the possibility to exchange Python objects over the network. A main idea is to connect Net Work Spaces (NWS)[4] with the developed system in order to facilitate such a system.

Acknowledgments

This paper was written after a disscusion with Michael Kifer over the FLORA-2 mailing list and a discussion with my mentor Mirko Čubrilo. In this place I'd like to thank them for suggestions and ideas.

References

[1] S. Berger, *Pythologic – Prolog syntax in Python*, on-line <http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/303057>, accessed: 21st November 2006.

[2] F. Coelho, *Extending python with prolog syntax *and resolution**, on-line <http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/360698>, accessed: 21st November 2006.

[3] M. Kifer, G. Lausen, J. Wu, *Logical Foundations of Object-Oriented and Frame-Based Languages*, Journal of the Association for Computing Machinery, May 1995.

[4] Nws-py.sourceforge.net, *NetWork Spaces for Python*, on-line <http://nws-py.sourceforge.net/>, accessed: 21st November 2006.

[5] W. M. Petullo and later C. Zagrodnick, *PyLog*, on-line <http://www.gocept.com/open_source_software/Pylog>, accessed: 19th November 2006.

[6] M. Pilgrim, *Dive into Python*, on-line <http://diveintopython.org/>, accessed: 19th November 2006.

[7] Python.org, *General Python FAQ*, on-line <http://www.python.org/doc/faq/general/>, accessed: 24th March 2007.

[8] SWIG, *Simplified Wrapper and Interface Generator*, on-line <http://www.swig.org>, accessed: 19th November 2006.

[9] XSB, *XSB API documentation*, on-line <http://xsb.sourceforge.net/api/index.html>, accessed: 19th November 2006.

[10] G. Yang, M. Kifer, C. Zhao, *FLORA-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Web*, In Second International Conference on Ontologies, Databases and Applications of Semantics (ODBASE), Catania, Sicily, Italy, November 2003.

[11] G. Yang, M. Kifer, C. Zhao, V. Chowdhary *Flora-2 : User's Manual, Version 0.94, (Narumigata)*, on-line <http://flora2.sourceforge.net>, accessed: 29th November 2006.

[12] Zope.org, *ZODB*, on-line <http://www.zope.org/Wikis/ZODB/FrontPage>, accessed: 21st November 2006.