EasyExtend

Kay Schlühr

kay@fiber-space.de

EE – Everything begun...

... with a tiny arrow operator as a visual clue

```
>>>  -> A0 A4 00 00 3F 00
```

EE – But where do we find arrows ?

easy to parse as console input

$$\Big\downarrow$$

easy to parse as an element of enhanced language?

EE – Why EasyExtend... ?

Extending Python is not a simple effort

- PEP 306 - lots of manual work! Independent file regenerations and C-file modifications, including compiler changes -> fork of both language and runtime.
- Python standard library parser not retargetable. Needs to be replaced
- No tools for creating, transforming and validating parse trees in the Python source base
- Lex/Yacc and ANTLR are not well integrated with Python runtimes

EE - Basic Requirements

Create *EasyExtend* as a pure Python framework for language extensions which

- enables extending Python *conservatively* (orthogonal extensions)
- parses from EBNF grammars
- supports parse tree constructors, transformers and validators
- uses existing Python compiler

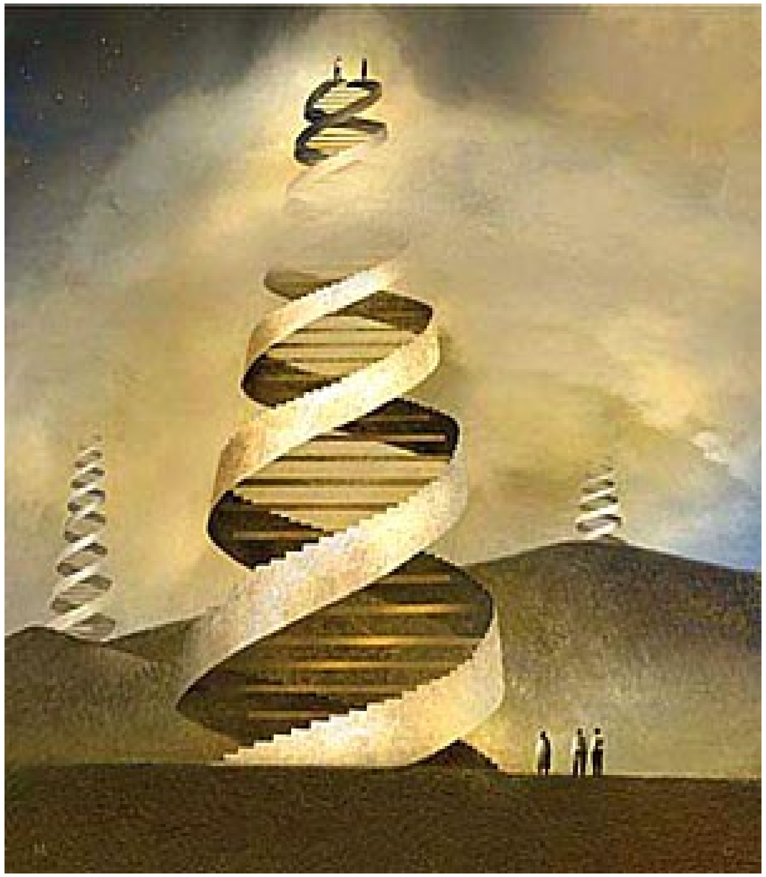
EE – Applications

What are the application domains of EasyExtend?

- Domain specific language extensions **of** Python
- Little languages running on top of Pythons VM
- X – compilers
- Refactoring, runtime monitoring, code coverage, source code checking etc.
- Macro systems

EE – Python and beyond

Steps into the fiber-space



- PEP 4100 – Making components more language like
- PEP 5100 – Adding *anymigrate* to the component library
- PEP 6100 – Communication bridges with extraterrestrial life
- PEP 50 (Meta-PEP) – Non-human requirements. Give transhumanity a chance.

EE – from Ω back to α

Import EasyExtend and create a new fiber – in honour of Europanto (http://en.wikipedia.org/wiki/Europanto)

```
>>> import EasyExtend
>>> EasyExtend.new_fiber("europanto_07", prompt="ep> ")
... [EasyExtend]+-[fibers]
                  +- [europanto_07]
                      +- __init__.py
                      +- conf.py
                      +- fiber.py
                      +- parsetable.py
                      +- Grammar
                      +- [fibermod]
                          +- __init__.py
                          +- symbol.py
                          +- token.py
                      +- [fibercon]
>>>
```

EE – the europanto_07 console

Taking a first look at the europanto_07 fiber

```
>>> EasyExtend.run("europanto_07")
*** Modify parsetable.py file ***
___________________________________________________________

 europanto_07

 On Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) ...
___________________________________________________________

ep> 1+1
2
ep> quit
___________________________________________________________

>>>
```

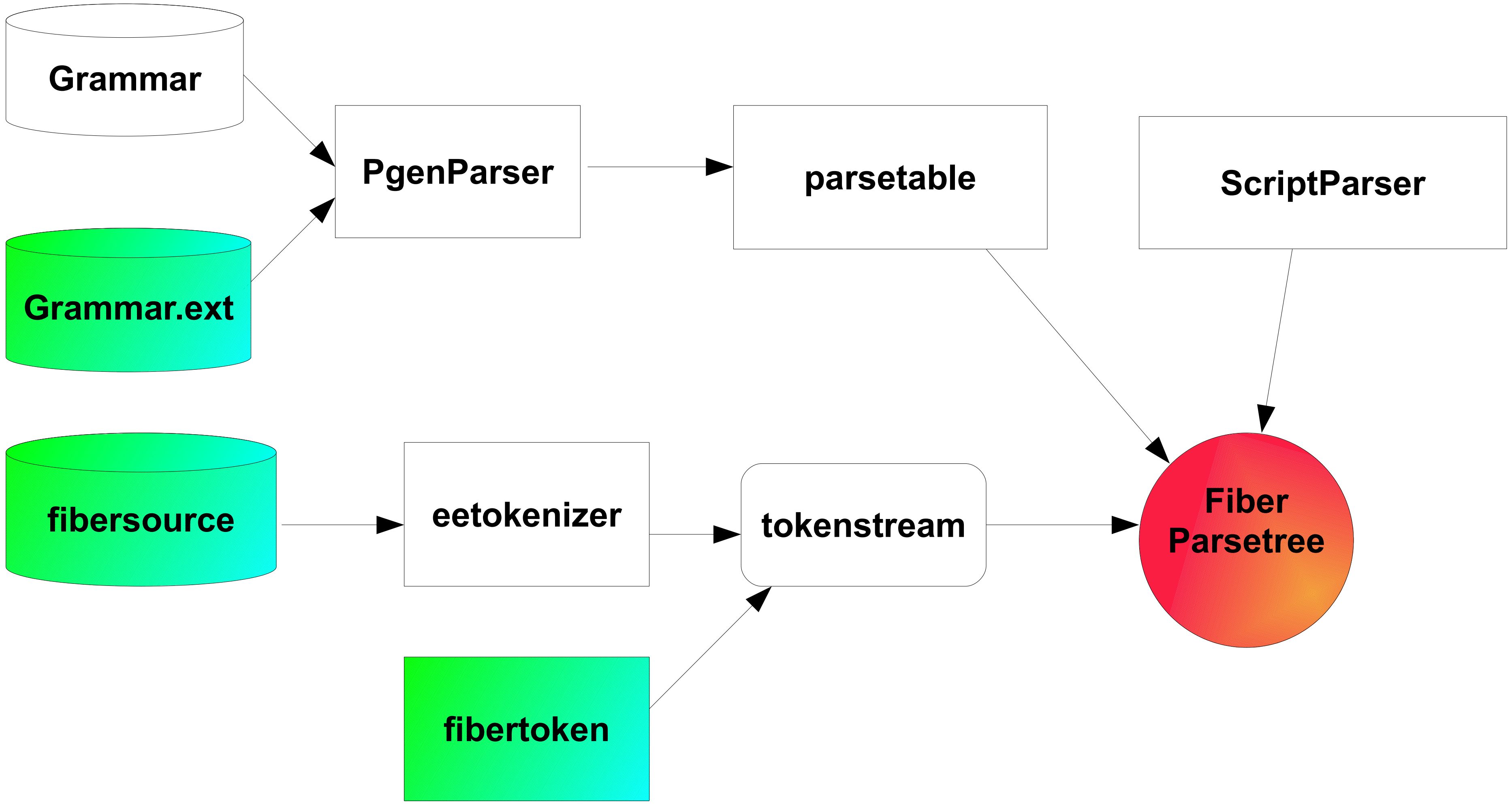
EE – how to proceed ??...??

After having created the new super duper europanto_07 language we have to admit:

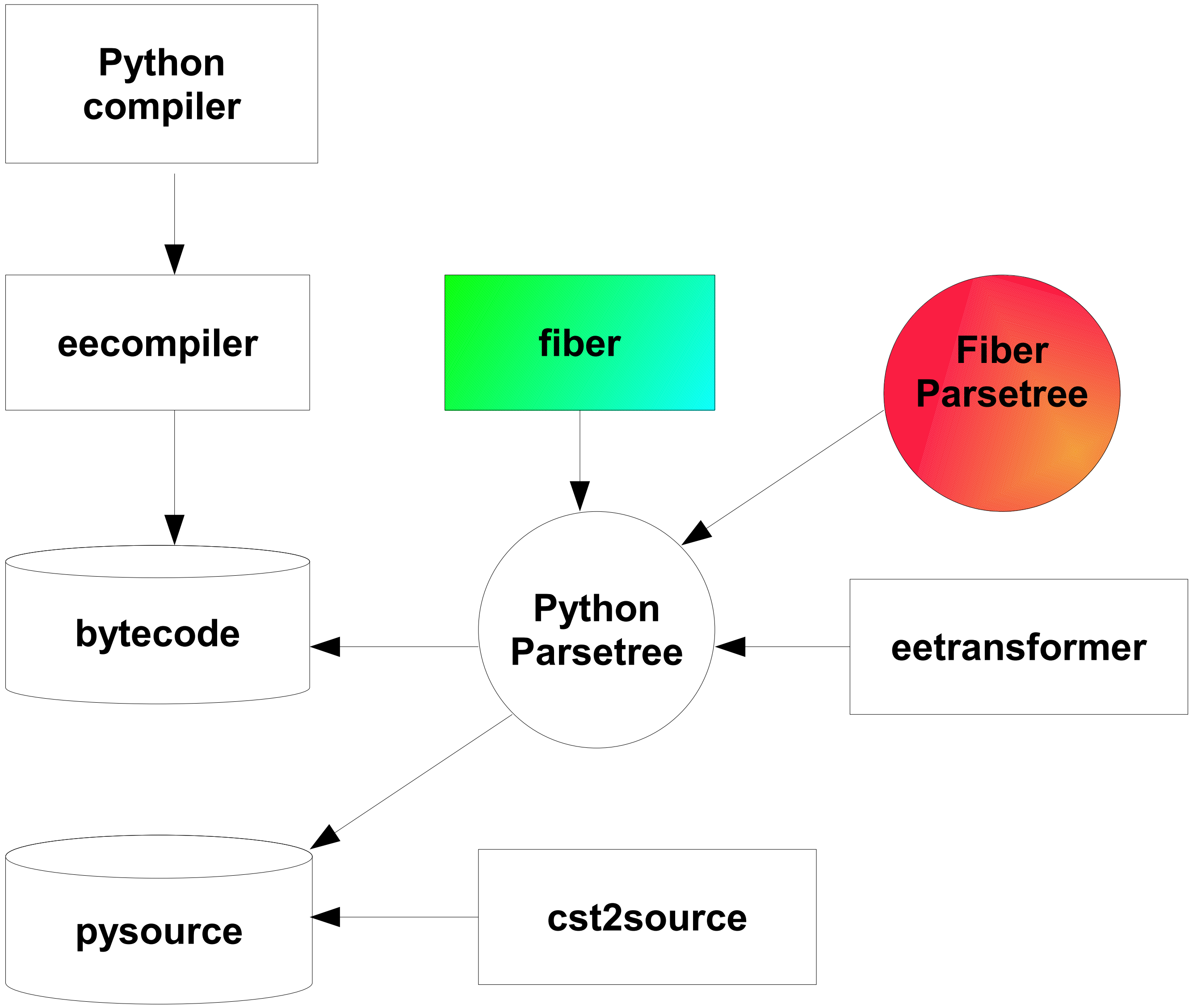
> it is yet nothing but Python with a fancy new prompt and a cute name.

Before we proceed we take a look at different parts of EE's framework.

EE – Parser Framework



EE – Transformer Framework



EE – Grammar.ext

```
# Grammar extension for europanto_07

compound_stmt: (if_stmt | while_stmt | for_stmt | try_stmt | funcdef |
                classdef | on_stmt | repeat_stmt | switch_stmt)

on_stmt: 'on' NAME '=' test ':' suite ['else' ':' suite]
repeat_stmt: ('repeat' ':' suite 'until' ':'
              (NEWLINE INDENT test NEWLINE DEDENT | test NEWLINE ))

switch_stmt: ('switch' expr ':' NEWLINE INDENT case_stmt DEDENT
              ['else' ':' suite])
case_stmt: 'case' expr ':' suite ('case' expr ':' suite)*
```

EE – FiberTransformer

```
module fiber.py - europanto_07
------------------------------


class FiberTransformer(Transformer):
    '''
    Defines fiber specific transformations
    '''

    @transform
     def on_stmt(self, node):
         "'on' NAME '=' test ':' suite ['else' ':' suite]"

    @transform
     def repeat_stmt(self, node):
         """
         repeat_stmt: 'repeat' ':' suite 'until' ':'
          (NEWLINE INDENT test NEWLINE DEDENT | test NEWLINE )
         """

    @transform
     def switch_stmt(self, node):
         "'switch' expr ':' NEWLINE INDENT case_stmt DEDENT ['else' ':' suite]"
```

EE – New Token

```
module fibermod +- token.py - europanto_07
------------------------------

from EasyExtend.eetoken import*

class FiberToken(EEToken):
    def __new__(cls):
        EEToken.__new__(cls)
        #
        # --- insert new token definitions or overwrite existing token here ---
        #
        cls.ARROW = (100, "->",OPERATOR)
        return cls
```

EE – CST support with csttools

- **basic node constructors (cst.py)**

```
single_input(*args) -> CST
file_input(*args) -> CST
decorator(*args) -> CST
...
```

- **searching and finding (csttools.py)**

```
find_node(node, node_id, level = 1000) -> node | None
find_all(node, node_id, level = 1000) -> node | None
```

- **wrapping nodes (csttools.py)**

```
any_test(node) -> test
any_stmt(node) -> stmt
```

- **advanced node constructors (cstgen.py -- provides auto-wrapping)**

```
CST_Assign(name, value) -> expr_stmt
CST_Dict(**dct) -> atom
...
```

EE – Implementation

```
module fiber.py - europanto_07
--------------------------------


class FiberTransformer(Transformer):
    '''
    Defines fiber specific transformations
    '''

    @transform
     def repeat_stmt(self, node):
        """
        repeat_stmt: 'repeat' ':' suite 'until' ':'
         (NEWLINE INDENT test NEWLINE DEDENT | test NEWLINE )
        """
        # keep statement fragments
        _suite = find_node(node, symbol.suite)
        _test  = find_node(node, symbol.test, level=1)  # don't look in suite!

        # use basic node constructor for if_stmt
        until_clause = if_stmt(_test, suite(any_stmt(break_stmt())))

        # place until_clause as the last but one element in _suite
        _suite.insert(-1, any_stmt(until_clause))

        # use CST_While to create a new node. Don't forget to wrap the while_stmt
        # for proper replacement of repeat_stmt
        return any_stmt(CST_While(True,_suite))
```

EE – checkout translation

```
>>> EasyExtend.run("europanto_07","-p")
________________________________________________________

 europanto_07

 On Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08)
________________________________________________________

ep> x = 0
[python-source>
x = 0

<python-source]
ep> repeat:
....    x+=1
.... until: x == 7
....
[python-source>
while True:
    x += 1
    if x == 7:
        break

<python-source]
ep>
```

FS – Fiberspace

What about multiple fibers and fiber transformers being active in the same context?

FS – Motivation : macro fiber

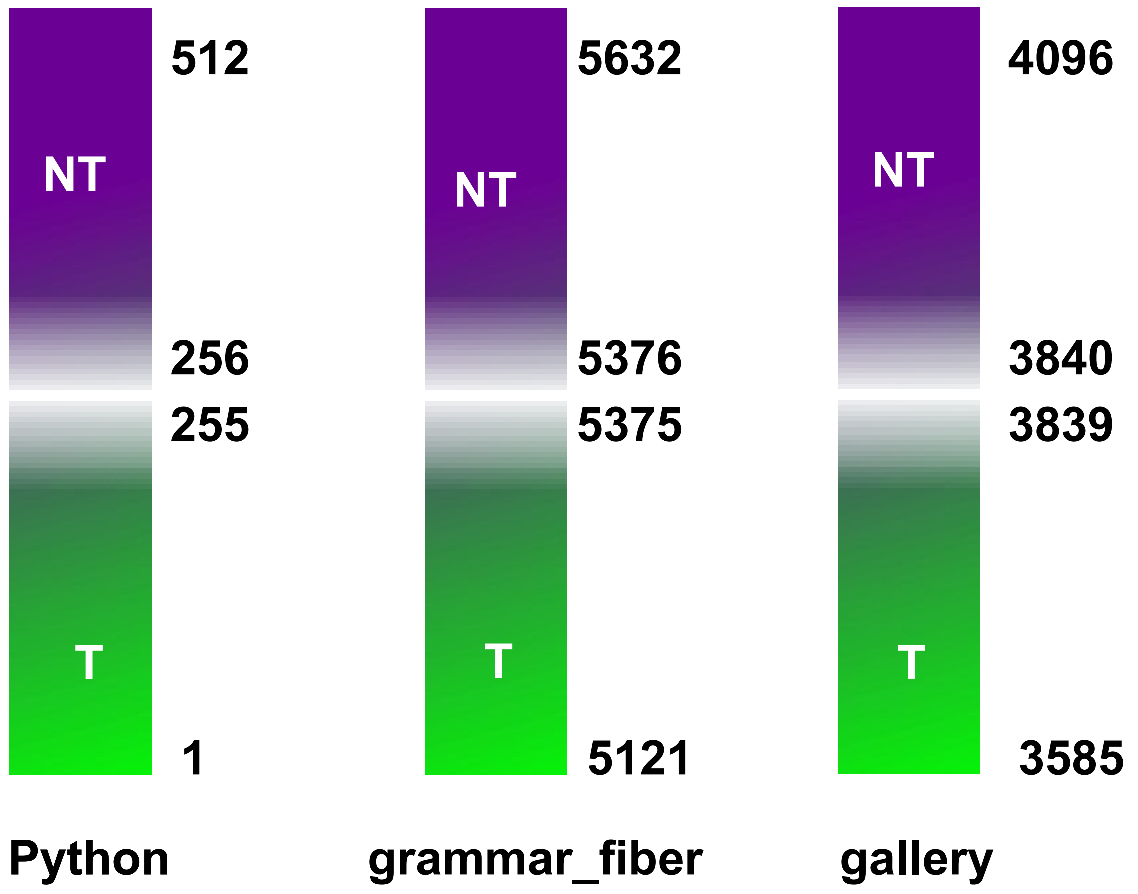
The macro fiber is a fiber used to transform other fibers

- goal of transformation is stated as a macro fiber statement/expression.
- nodes of target fiber are passed into macro fiber expansions -> nodes of macro fiber and target fiber are mixed and must be transformed at the same time.

How to mix fibers?

FS – Mixing fibers

Solution: each fiber has a unique range of node id's



$\Delta\, \mathrm{NT} = 256$

$\Delta\, \mathrm{T} = 256$

gallery.symbol.expr %512 = python.symbol.expr

FS – macro fiber example

```
  module fiber.py - europanto_07
  --------------------------------


from EasyExtend.fibers.macro.fiber import macro

class FiberTransformer(Transformer):
    '''
    Defines fiber specific transformations
    '''

    @transform
    def repeat_stmt(self, node):
        target = """
        while 1:
            <suite_stmts>
            if <test>:
                break
        """
        _stmts = find_all(node, symbol.suite, level=1)    # read stmts of the SUITE
        _test  = find_node(node, symbol.test, level=1)    # read TEST
        return macro(target).expand( {'suite_stmts': _stmts, 'test': _test} )
```

EE - Caveats

What causes myself head scratching?

- AST / CST transformation causes lnotab confusion -> stacktraces may display wrong information
- insufficient support for new file suffixes. Fiber modules are still *.py -> import machinery is not convincing
- no IDE support yet

EE/FS - Outlook

What has to be expected in the not so distant future?

- new, more powerful EBNF parser for non-LL(1) languages
- support for non Python languages and non CPython compilers
- celeryder – type recording machinery, code factoring and X-compilation API
- fiber fusion – automagical combination of different fibers. Advancing the *languages as components* metaphor.
- Python 3.0 support

Thanks for attention!

www.fiber-space.de