Grammatech, Meet Basil
(or “Geeks, Know Thine Selves”)

Jonathan Riehl
Grammatech, Inc.
July 29th, 2005
Why I Deserve to Live  
(Or: What have I done, and what I want to do.)

Please Give Me a Reason to Live  
(Or: Why you should care.)

Please Give Me a Place to Live  
(Or: No, really I need a place to crash soon.)
I Want to Live

• What I have done.
  – GRAD
  – Paths
  – PyFront

• What I want to do.
  – More of the same, but with a catchier name. (Or is that a more obscure name?)
What I Have Done

- GRAD - “Grammar Based Rapid Application Development”
- Paths
- PyFront
- And beyond...
Back in the day...

- Shuttle flight design required a team of software developers.
- Problem: Large C/C++ simulation code base and custom input language.
- Solution: Python
Python

- Provided a simpler language that could be used by aerospace engineers.
- Could ideally leverage the Python interpreter for simulation input language.
- New problem: Legacy investment vs. migration costs.
• Grammar Based Rapid Application Development
  – Given a language's grammar, we could automatically generate a Python extension module for that language.

• Isn't this SWIG?
GRAD vs. SWIG

User modified

.h File -> GRAD

_.i File -> SWIG

Wrapper Code

C++ Code -> C++ Compiler

Python Extension Module
Paths (GRAD/Paths)

- Find out what your boss hates to do and specialize in it...
- Idea: Rapid application development should feature rapid application testing.
- Can leverage GRAD technology to do this.
Rapid Application Testing Roadmap

• Step 1: Perform control flow analysis.
  – Identify structural test cases.
  – Instrument code for coverage analysis.
• Step 2: Perform data flow analysis.
• Step 3: Generate test vectors.
(Ooops. Funding got cut.)
Meanwhile, back at comp.lang.python...

- Lot of talk about a Python compiler.
- I was building dataflow models of both Python and C/C++, what if I “reversed the stream” of the C/C++ analyzer?

- Originally called Basil, but was able to keep compiler semantics true to Python.
And beyond...

- Everything is starting to look like nails...
- We begin to see patterns which imply opportunities for reuse.
And now a whopping huge intermission.

- A life hangs in the balance, and this guy wants to get philosophical?!
The N Languages Problem

- Originally termed the 500 languages problem.
- But in a world where everyone has their own language, it isn't so outrageous to generalize to an arbitrary number of languages.
In the Beginning...

Language 1  Language 2  Language ...  Language n

Backend

Object Code

This, is GCC.
And Then Came Analysis...

But, that's like, $n \times m$ components!
Hence .NET

Wow, we go from $n \times m$ to $n + m$. 
We go back to $n*k + k*m$, but gain flexibility. (And avoid lock out...)
What I Want to Do

- Provide a framework for analysis and integration of multiple programming languages.
- Provide the ability to prototype new languages.
- These goals are complimentary.
Follow the original GRAD approach: start with parsers.

Use these to build:
- Control Flow Models
- Data Flow Models
- Object Models
- Wacky User Defined Models
• So we have analysis, integration comes with applications of the analysis.

• These applications would support multiple languages right off the bat.

• Example: Provide a code generator for an object model and type map and you have SWIG.
Since we are building language parsers, we already have to play nice with parser generators.

Idea: Expose this infrastructure to assist with language grammar development.
Project Design

- Integrations
  - Gateways to domains beyond the wainscoting and/or framework.

- Parser Generator Integrations

- Language Integrations

- Model Integrations

- Applications
  - The glue between integrations.
Partial Integrations for: Bison, pgen
Current Parsers: Python, C
Modeling Madness!

3\textsuperscript{rd} Party Model Application  \rightarrow  Meta Model Instance  \rightarrow  Meta Model Code Generator  \rightarrow  Model Class Library  \rightarrow  Model Instance
Applications

- Code generators
  - Translators, Compilers
- Model viewers and utilities
  - Paths
  - Test Generation
- Model editors
  - Grammarian
For Instance: A M.S. Thesis

- The Basil Test String Generator
- Application that maps from Basil grammar model to test strings (code generator).

[Diagram showing the process with EBNF Grammar, pgen Grammar, Bison Grammar, EBNF Integration, pgen Integration, Bison Integration, Grammar Model, Testerizer, Test Strings]
Put it all together...
• Grammarian
  – Allows grammar models to be emitted as meta-models.
  – Generates some tests.
  – Working on generating code.
    • Take model and generate a Bison input that will output a syntax tree only – allows actions to be stripped.
• Have been doing tree rewriting recently...
  – Isn't that one way of specifying parser actions?
    • Yes, indeed. Reduction semantics by Felleisen and the PLT, is one example.
  – Solution: PyPat, a top down tree matching language.
Related Work

- Inten(s|t)ional Programming
  - http://intentsoft.com/
  - What does Charles Simonyi smoke and where can I get some?
- Stratego/XT (& SDF) (Visser et al.)
  - http://www.program-transformation.org/
- Ralf Lämmel
- SUIF and related plug ins
  - Recently became aware of OpenAnalysis, an open IR component.
Can I Get a Witness?

- Or possibly just “borrow a feeling”?

http://wildideas.org/basil/
• Okay, maybe I should calm down now.

• And we can talk business...
Why Grammatech?

- Isn't it obvious now?
- Okay, how about this:
  - I work on Basil for free. All of it is under the BSD license (extend and embed away).
  - Grammatech's tool chain could gain functionality by adding very few adapter components.
• Good engineering, but I'm at a theory school. Here's most of what I see in my bag of tricks:
  – Asking software engineering tool makers to eat their own dog food.
  – (Sane) decoupling of syntax and actions.
• In many ways, Basil is a solution looking for a problem.
  – Problems your customers might just happen to have.
• My intuitions can use a tweak.
  – What kind of functionality in this space would wow you?
● PyPat adds a key piece to the puzzle. It is time to demonstrate how it all works.

● What? More Python Compilers?
  – Refactor standard compiler module.
    • Provide support for multiple bytecode outputs (Python, Lua, Parrot).
  – PyPy - Python written in Python.
  – Son of PyFront – PyFront but with type inference.
• More language integrations!
  – Parsers for C++, Java, SML, Fortran(?!)

• UML/XMI Support
  – Will allow modeling using UML tools!
  – Had to use DTD to model translator to date.

• Models!
  – Don't even have old school models in there yet!
Acknowledgements

• The GRAD development team
  – Charlie Fly, Greg Boes, Mark Guerra

• Python community
  – Guido van Rossum, Jeremy Hylton, Fred Drake Jr.

• PyPy community
  – Holger Krekel, Armin Rigo, Samuele Pedroni

• University of Chicago
  – David Beazley, John Reppy
And now my excuse for a punchline...
No, really, I need a place to crash soon.

- School starts here on the 25th?
- The students are going to take back my sublet on the 15th!
- I'd at least like to stay to the 19th.
- Willing to trade labor, money, possibly some gratitude...
Thanks!

- That URL, one more time:

  http://wildideas.org/basil/
  -or-
  jriehl@cs.uchicago.edu