The Basil Project

(or “What I took away from GRAD school”)

Jonathan Riehl
University of Chicago
Outline

- Project History
- Project Goals
- Project Design
- Work, work, work, work...
Project History

- 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

- .h File
- GRAD
- .i File
- SWIG
- User modified
- Wrapper Code
- C++ Compiler
- C++ Code
- 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.
Project Goals

- 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.
Parser Generator Integrations

Grammar Input File → Grammar Input Parser → Grammar Syntax Tree → Grammar Back End

Grammar Generator

Grammar Model

Parser Generators

Partial Integrations for: Bison, pgen

Applications
Current Parsers: Python, C
Applications

• Code generators
  – Translators, Compilers

• Model viewers and utilities
  – Paths

• Model editors
  – Grammarian
Put it all together...
Work, work, work...

• What? More Python Compilers?
  – Refactor standard compiler module.
    • Provide support for multiple bytecode outputs (Python, Lua, Parrot).
  – PyPy - Python written in Python.
  – Xython - Son of PyFront w/type inference.

• Improve parser generator support.
Even more work!

• 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!
Thanks!

- 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