

The Case for Python

**Ed Petrus,
VP of Engineering
CiraNova, Inc.**

Overview

- ▶ **Requirements for an Extension Language**
 - Abbreviation: Extension Language = EL
- ▶ **The Landscape**
- ▶ **Defacto Standards**
- ▶ **Alternatives to SKILL**
- ▶ **The Case for Python**

Requirements for an Extension Language

- ▶ **Works well with applications requiring large virtual memory (common case in EDA)**
- ▶ **EL code is *not* delivered as object code for the underlying processor**
 - At run time code may be compiled into Byte Code, native object code, or some other format for performance reasons.
- ▶ **Use of an interpreter or VM is generally required**
 - The OS may arrange for the appropriate interpreter to be invoked automatically
 - The VM should be available in a form to be linked into C/C++ based applications
- ▶ **Explicit compilation or linking phases are not needed**
 - Implicit compilation to Byte Code or other low-level representations may be performed as an optimization
- ▶ **Variables, functions, and methods typically do not require type declarations. There are automated conversions or equivalence between types, particularly between strings and other types.**
- ▶ **Powerful built-in types - typically Lists, Association Lists, Hash, literal types: numerics, strings, symbols, classes, etc.**
- ▶ **The ability to generate, load, and interpret source code at run time through an *eval* function**
- ▶ **Minimal user level memory management**
 - Preferably fully automatic memory management requiring no programmer intervention

Requirements, continued

- ▶ **Interface to the underlying operating system, in order to run other programs and communicate with them.**
- ▶ **Plays well with others. Can be easily integrated with larger systems. ELs are often used to "glue" more-rigid systems together.**
- ▶ **Rich I/O capabilities, including pipes, network sockets, file I/O, and file system operations.**
- ▶ **Language interpreter can be embedded within another application, allowing users to:**
 - Automate application functions
 - Provide customized handling of application events
 - Define new application functions using features in the language
- ▶ **The ability to easily map external components to the semantics of the built-in types and/or some component-based protocol (e.g. SWIG, COM)**

The Landscape for EDA

- ▶ **AutoCad is the first known case where a full programming language (Lisp) was embedded in a CAD application**
 - Emacs had an embedded Lisp engine
 - Calma developed a proprietary scripted command language called GPL
- ▶ **Cadence was the pioneer in commercial EDA with SKILL as a full featured extension language [Circa 1985]**
 - SKILL is based on the UCB originated *Franz Lisp* flavor of Lisp
 - Scheme semantics were introduced as an extension to SKILL [Circa 1995]
 - Trivia: SKILL was originally named SCIL
- ▶ **Many EDA companies followed the Cadence EL model using variants of Lisp**
 - Avanti, Mentor, Viewlogic adopted variants of Scheme [Circa 1992]
 - Use of the language for deep access to the data model and full access to physical design tools' functions is a feature common to all EDA extension languages
- ▶ **By the mid 1990's TCL was being used as a scripting language for Verilog-based design tools**
 - TCL originated in the engineering department at UCB as a "Tool Control Language" for CAD programs
- ▶ **Variations on this theme:**
 - TCL used as an extension language for physical design tools - Magma
 - PERL used an extension language for physical design tools - Reshape

Defacto Standards

- ▶ **Standard cell based digital design**
 - Cadence and Avanti tools dominate this space
 - ∴ SKILL and Scheme are the dominant extension languages
- ▶ **Custom digital physical design**
 - Cadence tools dominate ∴ SKILL
- ▶ **Analog, AMS**
 - Cadence ∴ SKILL
- ▶ **Front end design tools**
 - TCL everywhere

Alternatives to SKILL

▶ Scheme?

- Substantial modifications needed to make the language acceptable in EDA
- Not all core language features are supportable for an extension language (e.g. CWCC)
- All EDA companies that adopted Scheme fielded versions modified from the current standard

▶ Tcl?

- Limited language features
- Difficult to extract performance gains through compilation
- Poor binding to C++

▶ Java?

- Large foot-print
- C/C++ binding less than great
- Not so popular in CAD

▶ PERL?

- Not close to SKILL & Scheme heritage

▶ Invent a new language?

- Should not be an option in the 21st century

The Case for Python

- ▶ **Cross between Lisp & C++**
 - Python can be viewed as a close relative of SKILL & Scheme
 - Lowers adoption bar
- ▶ **Object oriented by design**
- ▶ **Strongly typed for an EL**
- ▶ **A full-fledged programming language with powerful semantics and language constructs**
- ▶ **Satisfies most if not all EDA requirements for an EL**
- ▶ **Powerful operating environment**
 - Byte code compiler and VM
 - Fully automatic memory management
 - JIT
- ▶ **Allows excellent binding to C APIs and to C++ classes**
 - Supports dynamic loading of components written in Python/C/C++
- ▶ **Highly portable - VM available for many processor architectures and OS's**
- ▶ **Simplicity and language preciseness were key goals**

Python, continued

- ▶ **Hundreds of 3rd party libraries** in addition to the rich standard set (Tk, Qt, zlib, etc.)
- ▶ **Open source and commercial Interactive Development Environments** available
- ▶ **Actively maintained by a highly respected group in the open source community with 1000's of individual contributors.**
 - BDFL (Benevolent Dictator For Life - Guido van Rossum).
- ▶ **Excellent documentation, books and training material**
 - O'Reilly books
 - www.python.org
- ▶ **Already popular within the EDA community**
 - Python to OA binding available from SI2 (donated by LSI Logic)
- ▶ **Who else uses Python?**
 - Yahoo [maps, groups]
 - Google [web spider components]
 - Linux installer
 - Included with all Linux distributions



Sample Code

Lists

```
>>>alist = [1, 2, 3, 4]
>>>alist[0]
1
>>>def inc(x): return x+10
>>>map(inc, alist)
[11, 12, 13, 14]
>>>map((lambda x: x+3), alist)
[4, 5, 6, 7]
>>>alist.append(5)
[1, 2, 3, 4, 5]
>>>[1, 2] + [3, 4]
[1, 2, 3, 4]
```

Dictionary

```
>>>dict = {'pin':99, 'term':88}
>>>dict['pin']
99
# File I/O
>>>myfile = open('myfile', 'w')
>>>myfile.write('hello \n')
>>>myfile.close()
# Exception Handling
>>> try:
...     doomed()
... except Bad:
...     print 'got Bad'
...     print 'not doomed'
```

Summary

- ▶ **It is of vital importance for native-on-OA EDA companies to converge on one EL**
 - Strongly recommended for the few startups working on OA
- ▶ **Equally important for the EDA industry will be to converge on a software architecture for a component model**
 - Similar to COM (BUT easier to develop and friendlier to use)
 - Similar to Java Beans (BUT uses a language popular in EDA)
- ▶ **C++ & Python are a great combination for building an EDA centric component object architecture**
 - C++ for building foundation layers
 - Python for building components that are simpler to develop and work in a safe sandbox environment