Integration of Python into SNiFF+

Michael Scharf

Michael_Scharf@TakeFive.co.at

1997

SNiFF+ is a Integrated Software Development Environment originally designed for C++. In 1995 some effort was made to open it for other languages. Here, I will not talk about how we use Python as embedded language for SNiFF+, but how we added support for developing Python code with SNiFF+. SNiFF+ now enables you to browse and debug Python code with the same comfort as for C++.

Parser

For each language SNiFF+ supports, there is a parser to extract symbol information from source the files. Each parser runs in a separate process and SNiFF+ attaches to the parser when a file needs to be re-parsed. The parser calls a set of simple C API functions to feed the symbol database of SNiFF+. The communication between the parser process and SNiFF+ is transparently done in the library.

The interface requires to return the exact start and end portion of each symbol in order to do highlighting in the editor and do exact navigation. Therefore it was not possible to use the parse-tree generated by the Python compiling step.

I started off creating an extension module for Python and I did the parsing in Python. This turned out to be quite slow, so I decided to rewrite a very simple parser using only a lexer (flex). The first running version was built in a few hours, and with some further refinement, the parser does a reasonable job.

Symbol mapping -- Language concepts

The internal representation of languages in SNiFF+2.x is hard-coded for C++. When a new language is added, a language description file has to be created to map language items of the new language to the C++ representation. It is also possible to rename or disable constructs in the GUI. Things like *enumeration*, *typedef*, and *macro* don't appear in the GUI when browsing Python code. Private class members (double underscore) and class variables (assignment at class level) are supported. To emulate syntax coloring, I abused some of the useless symbol types to highlight Python keywords and strings. E.g. strings now get the same character style as macro definitions in C++.

Debugger integration

SNiFF+ supports 3rd party command-line debuggers, by running them in a GUI-shell. It interprets the output and 'types' commands into the shell when some buttons are pressed in the GUI. To add a new debugger type to SNiFF+, a subclass of the C++ class DebuggerAdaptor has to be created (this would be a great thing to do in Python). Since I didn't want to touch any C++ code to do the integration, I just wrote a new debugger in Python based on the bdb which emulates the behavior of GDB (at least the subset of commands SNiFF+ uses).

The most difficult technical problem was, that SNiFF+ stops the debugging session when the program in the debugger shell dies. So, I decide to have the debugger process all the time running, and as soon as the user hits 'run' a new process is forked. This child process does all the debugging work and before the it finishes, it sends all its state (e.g. breakpoints) to the parent process. When 'run' is hit a second time, a new Python session starts that remembers the breakpoints and all modules (except the ones the debugger uses) are imported 'freshly'.

It was also necessary to extend the debugger, that it can run from the command-line with an arbitrary Python program as argument (as you would do with GDB).

The debugger pygdb can also be used in EMACS in the GDB mode.

Open issues

- No cross-reference information. This is a principal problem of dynamically typed languages.

- Debugging Python embedded in applications.
- Alias problems: It seems to be common practice to alias a class name:
```
X=MyClass
class Derived(X): pass
```
  A solution would be, to keep track of simple assignments.
- Name-space Problems: SNiFF+ has no name-space/module support (yet!)
  module1.py
```
class MyClass:pass
```
  module2.py
```
import module1
class MyClass(module1.MyClass)
```
- Since Python does not have explicit variable declarations, each time a member is assigned to, I treat it as declaration of that particular variable. E.g., in the ClassBrowser, you will see the same variable name multiple times.

References

SNiFF	http://www.takefive.com
Python	http://www.Python.org
SNiFF4Python	http://starship.skyport.net/crew/scharf/sn4py
GDB	http://www.gnu.org