Multiprocess System for Virtual Instruments in Python

An Introduction to Pythics

Brian R. D’Urso
Assistant Professor of Physics and Astronomy

University of Pittsburgh
School of Arts and Sciences
Department of Physics and Astronomy

Oak Ridge National Laboratory
Measurement Science and Systems Engineering Division
Motivation

• How to control laboratory instruments and experiments?
  – Also applies to many models and simulations.

• Commercial software is great for simple problems, but may be inadequate for complex tasks.
  – Programming languages are too specialized.
  – Inadequate multithreading/multiprocessing.
  – Lack of robustness.

• Software developed in proprietary languages may be difficult for students to utilize after they graduate.

• Alternative: use C, C++, Fortran, etc. But:
  – Steep learning curve: GUI building is complex.
  – Error-prone: most laboratory problems require multithreading.
  – Redesign needed for every project.
Solution Requirements

- Use a general-purpose, easy to learn language (Python).
- Cross-platform (Linux, Windows).
- Avoid excessive dependencies (Python > 2.6.0, wxPython).
- Don’t reinvent the wheel.
  - For full functionality: NumPy, matplotlib, Python Imaging Library (PIL).
  - Also useful: PyVISA.
- Data acquisition and processing should not interfere with GUI operation (including loops, timers, instrument timeouts).
  - User-programmer should not have to directly use multithreaded/multiprocess programming techniques.
- Simple, low-fuss method of specifying GUI (must be portable).
  - User-programmer does not have to understand GUI programming.
- Don’t introduce new syntax or special functions.
- Sharing of data between multiple VIs must be simple.
- System must be robust.
Solution: Pythics Overview

- Pure Python implementation.
- Organized as a program that manages ‘documents’ – virtual instruments or VI’s.
  - Single GUI process and thread (mandated by GUI toolkits).
  - Each VI gets its own process for executing actions.
- GUI is web browser style:
  - Tabbed interface, one tab per VI.
  - Layout within each tab is html-like.
- User-programmer writes two files:
  - GUI specification: html/xml.
  - Callback functions: pure Python.
Multithreading/Multiprocessing in Pythics

- **Design:**
  - Minimize bottlenecks.
  - Simple to use: User code runs in a single thread/process.

- **One GUI process.**
  - Main GUI thread.
  - Helper thread for managing request queue.

- **Each VI has an independent action process.**
  - Main thread for executing actions (user code).
  - Additional thread for each timer to trigger periodic actions.
Using Pythics – GUI Specification

- GUI layout (text and controls) by custom xml/html layout engine.
- Supported controls:
  - Images, buttons, text boxes, file dialogs, numeric input/output, spreadsheet, sliders, Python shell, …
  - Plots (wxpython, matplotlib).
- Style specification (fonts, colors) with simplified CSS.
- Many basic formatting tags supported.
- Some ‘special’ controls for side effects:
  - Timers, globals, loading python files, loading default parameters, …
Using Pythics – Example GUI

<html>
<head>
<title>Hello World</title>
</head>

<body>

<h1>Hello World</h1>

<object classid='Button' width='200'>
   <param name='label' value='Run'/>
   <param name='action' value='hello_world.run'/>
</object><br/>

<object classid='TextBox' id='result' width='200'>
</object><br/>

<object classid='ScriptLoader' width='100%'>
   <param name='filename' value='hello_world'/>
</object>

</body>
</html>
Using Pythics – Callback Functions

```python
def run(result, **kwargs):
    result.value = "Hello, world!"
```

• The Python portion of a VI is one or more pure python files.
  – The files should define ‘callback functions’ which are called when GUI events occur (e.g. a button click).

• Callback functions are passed all GUI controls as keyword arguments (the key is the ‘id’ parameter given in the xml file).

• The control objects passed to callback functions are proxies with a simplified interface.
  – Proxy objects handle all interprocess communication.
  – Simple interface: high level properties and methods.

• Otherwise free to do anything in these files.
Additional Features

• Optionally speed up inter-process data transfer with shared memory.
  – Available for canvas images.
  – Useful for array data (plots, tables)?

• Timers live in action process for minimal overhead.
  – Can be aborted at any time (uses events internally).
  – No limit to number of timers per VI.

• Shared data:
  – Use global namespace from multiprocessing package.

• Parameters:
  – Save and restore values stored in controls (including setting defaults).

• Sub-windows:
  – Can encapsulate an entire VI in a single control.
  – Allows effective creation of complex controls.
Putting it All Together

- Pythics allows simple:
  - Specification of GUI.
  - Multiprocessing and multithreading for simultaneous data processing and GUI functionality.
- Limitations:
  - Web page like GUI design.
  - Weakly coupled GUI and code.
- Designed for experiment control.
  - Also useful for controlling models and simulations.
- Builds on many great Python packages:
  - Built-in: multiprocessing.
  - Add-ons: wxPython, NumPy, …
- All released under GNU GPL!
  - See http://code.google.com/p/pythics/
Future Improvements (Wish List)

- Build a community? Get funding?
- Documentation and testing.
- Improved real-time plotting.
  - Need speed for fast updates (~ 10/second) and scrolling.
  - More plot types.
  - Based on matplotlib? Chaco? plplot?
- Printing support.
- Optional graphical GUI editor.
  - XML format makes it fairly easy.
- 3-D rendering widget.
  - Try to port Visual Python (VPython)? PyOpenGL? VTK?
- Operation over network.
  - Front and back ends on different machines.
  - Some built-in support in multiprocessing module.