py.execnet: ad-hoc networking

holger.krekel@merlinux.de at
EuroPython 2006, CERN/Geneva

overview of py lib(s)

- **overview on 'py' lib**:
  - py.path: local and subversion filesystem objects
  - **py.execnet**: ad-hoc distribution of programs
  - py.log: (exp) provides a simple logging mechanism
  - py.code: nice introspection and dynamic compiling
  - py.xml: providing simple xml/html object generation
  - py.magic: provides greenlets (see Armins talk)
  - py.compat: cross version 2.4.1 backported modules
- aims at uniformly running on python 2.2 onwards

Reasons for distributing services

-

  - remote access to local system resources
  - security
  - reliability
  - scalability

Network Protocols

- Network Protocols between services/systems
    - Remote Method invocation (Java/Corba/Pyro)
    - Chat (text) based protocols (http/smtp/DNS/...)
    - “Web” services (SOAP/xmlrpc...)
    - Very good for interoparability
- **Global Standards are useful for large scale co-operative programs!**

The “Standard” problem

- Standardized protocols commonly require:
  - matching/Compatible software versions
  - prior installation, configuration and setup
  - overhead on designing, testing and maintaining the standard!
  - “global identity” (GUID) schemes for referencing

py.execnet concepts

- client side injects local protocol code (”remote_exec”)
- client and “other side” interact through Channels
- Channels can receive and send arbitrary marshallable Python structures
- asynchronously executing program fragements implement synchronous program flow (blocking on channel operations)
- similarity to Stackless Communication model (tasklets/ Channels)

channels and gateways

- gateways hold connections to other processes (wherever)
- gateway.remote_exec() allows you to run source code on the remote side
- **Communication via symmetric channels**
    - interactive example

Example: svnhotsync

- synchronises a remote repo to a local one
- no temp/spool files
- server and client side code less than 100 lines
- runs very reliably for around 10 systems

Example: remote file processing

- (Real life) Problem: a remote system processes data from a "data_in" directory and produces "data_out" items.

- doing this via sftp or rsync+ssh has race conditions and is icky to implement robustly, also the remote system can not be used concurrently. Setting up RMI systems has a lot of development and maitenance/deployment overhead.

- solution: use ssh+python, deploy the protocol and data_in/data_out code from the "using" client side.

Status

- py.execnet is usable for 2-peer distribution / deployment
- "Makes distribution easy but sharing state hard"
- channels cannot span multiple gateways/hops yet
- basically works on win32, osx and linux (ssh not on win32)

development

- basic works from Holger Krekel, Armin Rigo, Jan Balster 2003-2006
- is part of the py lib, used by py.test
- MIT License
- partly funded by the EU IST programme
- source living at http://codespeak.net/svn/py/dist, GPL
- unit-tested on various levels (py.test)

Future

- current development happens on a "demand" basis
  - from ourselves/involved parties, py.test requirements
  - from contributors/users/external sponsoring parties
- support for better sharing
- extending to multi-peer (P2P) architecture
- Dev Contact at py-dev@codespeak.net,
- training/support possible

merlinux