

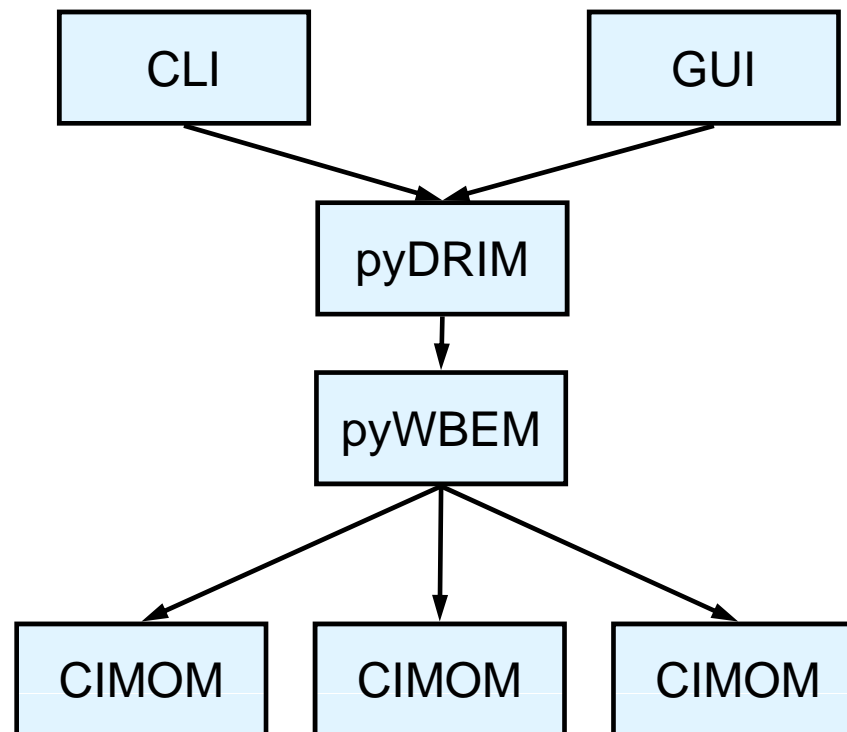


pyDRIM - OpenDRIM Command Line

OpenDRIM Camp Sept. 10, 2007

OpenDRIM Command Line

- Written in python
- Frontend: CLI or GUI
- Backend: pyDRIM and pyWBEM



- PyWBEM is a Python library for making CIM operations over HTTP using the WBEM CIM-XML protocol.
- “a good WBEM client should be easy to use and not necessarily require a large amount of programming knowledge.”
- Also provides python provider interface (not our focus today)
- <http://pywbem.sourceforge.net/>
- Examples:
 - Making connections

```
import pywbem
conn = pywbem.WBEMConnection("https://172.16.80.24", ($name,$passwd))
```
 - CIM Instances and Instance Names

```
insts = conn.EnumerateInstances("CIM_Process")
print insts[0].items()
```

- Hide the details of WBEM operations
- No need to understand Instances, association, key, name...
 - InterateInstance("Process") vs. Process p;
- More efficient for admin tool development
- More intuitive for sysadmins

```
CIMClient client = CIMClient(url,cimNamespace);
CIMInstanceArray processes =
client.enumerateInstances(processClassName);
for(int i = 0; i < processes.size() i++)
{
    if (processes[i]->getCIMPropertyValue(worksetSize) >
        maxMemorySize)
        CIMObjectPath instanceName =
            CIMObjectPath(cimNamespace,processes[i]);
        CIMParamValueArray params;
        CIMParamValue paramSignal(killMethodParamName);
        paramSignal.setValue(CIMValue(9)); //signal
        params.append(paramSignal);

        client.invokeMethod(instanceName,killMethodName,params);
        cont << "kill process <<
        processes[i].getCIMPropertyValue(processName) << endl
    }
}
```



```
from pydrim.connect import Connection
from pydrim import process
Connection conn = Connection(url,('root,')
For p in process.GetProcesses(conn):
    if p.worksetSize > maxMemorySize :
        p.kill(9)
```

- sysv.py: system services
- user.py: user & group
- perf.py: performance data (CPU/Memory/Network/Disk)
- cron.py: chronicle tasks configuration
- syslog.py: system logs information
- rpm.py: RPM packages
- task.py: process information

- Consistent Command Line Interface
 - Tab to complete parameter
- 2 interaction modes
 - Command mode
 - Shell mode
- Format using Curses
- Multithreaded execution
- Extensible

- pydrimplugin
 - All CLI plugins are derived from pydrimplug class
 - common handler of CLI parameters
 - Add_options(), Get_options()...
 - Register handler of CLI plugin
 - common error handling
- Define MODULES=["\$module_name"], so that CLI shell knows its name
- Default plugins
 - rpm/sysv/user/perf/syslog/task/cron

- The most user-friendly tools are the tools that users already know
- CLI based on pyDRIM is easy, but not as easy as ps or top
- add a 'R'
 - Originally user have:
 - shell, cp, login, exec
 - When networked, then user have:
 - rshell, rcp, rlogin, rexec
- add a 'D'
 - Original sysadmin have:
 - Top, ps, useradd, ntsysv
 - When networked, sysadmin have:
 - Dtop, dps, duser, dservice
- DTools
 - Old sysadmin tools enhanced with a little touch of "Distributed"

- Dtop
 - “Find the highest loaded process of all nodes”
 - Collection process data from all nodes
 - Multi-threaded data collection
 - Sort by memory/cpu/pid/priority
- Dps
 - “Find the busiest httpd of the server farm”
- Duser
 - “add user account on all servers”
- Dservice
 - “terminate all ftp services within the network”

Thank You !