Remote and mass management of systems with finstall

Ivan Voras
<ivoras@freebsd.org>
What is it – description of the idea
A few words about the protocol
Details of the implementation
Examples
Future plans
What is this all about?

- First there was finstall [pronounced eff-in-stall]
  - Google SoC project
  - Not abandoned
  - But stalled, ENOTIME, ENOMONEY
- Important concept of finstall: complete separation of the GUI and the back-end
- The back-end does the work
- Communicates with the client via a RPC-like protocol
Frontend - Backend
The idea

- Use the backend part for system installation and configuration via direct interface
- Enables remote management of systems
- The obvious question: is it similar to Kickstart?
  - Yes, it's going in roughly the same direction
  - It's not there yet
  - It needs much more automation
  - Polish the rough edges
- The backend's name is SysToolD
Backend capabilities

- Simple XML-RPC protocol
  - Developed in Python so some functions are dynamically typed – will need to change in the future
- Offers high(ish)-level functionality to clients
  - Get / set basic system information
  - Get / set data from config files
  - Device partitioning, formatting (newfs), mounting
  - Network configuration
  - ...

A bit about implementation

- trunk/bybackend in Subversion, in SF.Net
- Written in Python
- Good sides:
  - Easy to prototype
  - Easy XML-RPC
  - Easy string, XML parsing, etc.
- Bad sides:
  - Needs Python
  - Cannot directly access C structures
Python implementation invokes command-line system utilities (like `sysctl(8)`, `newfs(8)`)

- Some argument passing, parsing, etc.
- The backend is intended to run as a background daemon
- The daemon optionally issues UDP broadcasts for discovery (for the installer)
RPC Functions (1)

- GetHostId()
- GetDMESG()
- GetHostName()
- GetPhsyMem()
RPC Functions (3)

- `GetLoaderSetting()` / `SetLoaderSetting()`
- `GetConf()` / `SetConf()`
- `GetHostName()` / `SetHostName()`
- `GetShells()`
- `AddUser()`
- `GetNetworkInterfaces()` / `ConfigureNetworkInterface()`
- `SetDefaultRouter()`
- … etc.
How to use it

- **Step ONE:**
  - The system needs to run systoold.py
  - a) regular system – rc.d
  - b) PXE boot for installing
  - c) bootable ISO image for installing

- **Step TWO:**
  - Access the daemon's services with XML-RPC
  - Python XML-RPC
  - Any other XML-RPC
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
    <methodName>examples.getStateName</methodName>
    <params>
        <param>
            <value><i4>41</i4></value>
        </param>
    </params>
</methodCall>
XML-RPC libraries

- “Script” languages have it easy...
  - Python, Perl, PHP, Flash, JavaScript etc.
- C, BSD-Licensed: http://xmlrpc-c.sourceforge.net/
- Java, Apache Licensed: http://ws.apache.org/xmlrpc/
C example (the most complicated)

result = xmlrpc_client_call
    (&env,
     "http://xmlrpc.host/",
     "GetSomething",
     "(ii)",
     (xmlrpc_int32) 5,
     (xmlrpc_int32) 7);
SysToolD doesn't enforce a mode of use – it's a tool for configuration and administration.

**INSTALL mode**
- Can be used to install a fresh system
- The front-end is the installer which connects to localhost (or optionally to a remote host)

**MANAGEMENT mode**
- Used to (re)configure existing systems
- Usually used by remote clients
Modes of use

- SysToolD doesn't enforce a mode of use – it's a tool for configuration and administration

- MANAGEMENT mode
  - Used to (re)configure existing systems
  - Usually used by remote clients
Example 1

```python
from xmlrpclib import ServerProxy
host = ServerProxy("http://10.0.0.10:1025")
host.InstallRemotePackage("apache22")
host.SetConf("apache22_enable="YES"")
host.SetLoaderSetting("accf_http_load="YES"")
```
Example 1

from xmlrpclib import ServerProxy
host = ServerProxy("http://10.0.0.10:1025")

- Boilerplate code – create a proxy object for XML-RPC
- Looks the same in every language
- Simple
Example 1

- The “meat” of the script
- Note: error checking is pretty much non-optional here

```java
host.InstallRemotePackage("apache22")
host.SetConf("apache22_enable="YES""")
host.SetLoaderSetting("accf_http_load="YES""")
```
Real-world example

- Needs more automation
- Generally:
  - Have a list of SysToolD-enable hosts
  - OR...
  - Gather the list by listening to broadcasts
- Inspect environment(s) of host(s)
- Create threads and (re)configure each host in parallel
Security

- SysToolD is not a remote root shell but is as close to it as doesn't matter
  - Can modify rc.conf and reboot
- Need to bar unwanted accesses
- There is no fine-grained access control once users get to SysToolD
- Current solution: SSL certificates
  - Users need a certificate signed by a server-accepted CA
Current state of development

- A bit slower than expected – part of finstall
  - Can pick up if funding is found
- Features get added when needs shows
- XML-RPC has proven to be a good and robust thing for this kind of usage
- Python has proven to be good for development with minimal problems
Future development

- Automation
- CLI tools
- GUI tools
  - The idea is to have a list of machines (or a icon spread) and have users right-click on a machine and say “run this operation”
  - Would like it to remain in Python because of easy development
    - If the protocol is retained, the implementation details can change