Config : :Model and configuration upgrades during package upgrade

Dominique Dumont

Debian Perl Group

Oct 2010

Outline

Configuration is often painful !

Configuration upgrade is often difficult for a user :

- Surprise question during upgrade
- Edit a text file outside of /home
- Read man pages
- Ensure consistency
- Leave spurious files

Basic configuration may also be difficult...

Objective 1 : Make configuration easier for users

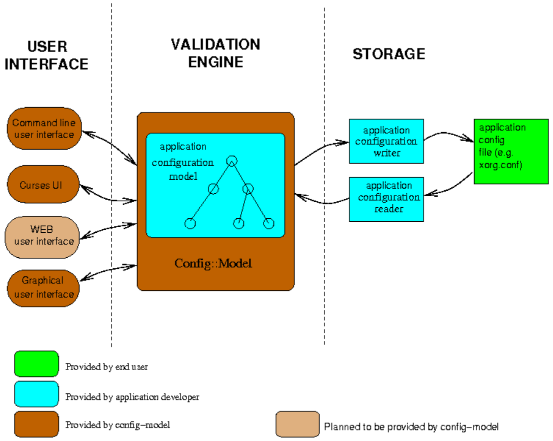
Handle configuration upgrade smoothly (mostly no interaction)
Provide a graphical interface with :

- Integrated help
- Default values
- Validation of configuration data
- Several levels of skills *(from beginner to master)*
- Search

Objective 2 : Make maintenance easy for developers

- Configuration tool and upgrader must be easy to maintain :
    - Avoid ad-hoc validation code (e.g. don't rewrite Webmin)
    - Base validation on "meta-data" : the *configuration model*
    - Generate interfaces (graphicals or not) from the model
    - Model contains properties to upgrade configuration
    - GUI to create and maintain models
- Minimise code required to read or write configuration files :
    - Use existing libraries (Config : :Ini, Config : :Augeas – and all Augeas lenses...)
    - Provide basic classes to help configuration reads and writes

Config : :Model design


USER INTERFACE
VALIDATION ENGINE
STORAGE
Command line user interface
Curses UI
WEB user interface
Graphical user interface
application configuration model
Config::Model
application configuration writer
application configuration reader
application config file (e.g. xorg.conf)
Provided by end user
Provided by application developer
Provided by config–model
Planned to be provided by config–model

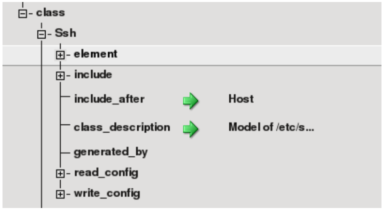
What is a model ?

Config is represented in a tree. The model defines its structure :

- A class is represented by a node
- A parameter is represented by a leaf

Each class contains :

- a set of elements (parameters)
- optional : a specification to access configuration file (backend)



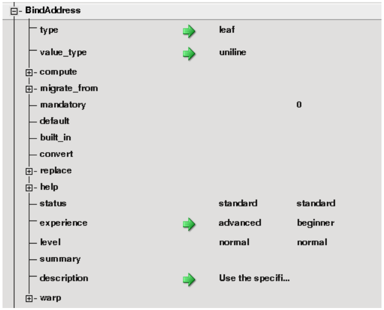
```
⊟- class
  ⊟- Ssh
    ⊞- element
    ⊞- include
     - include_after        Host
     - class_description    Model of /etc/s...
     - generated_by
    ⊞- read_config
    ⊞- write_config
```

model GUI

Simple elements

Each element has :

- a type (leaf, hash, list, node)
- constraints (integer, max, mini ...)
- a default value
- a description and a summary (for integrated help)
- an experience level (beginner, advanced, master)
- a status (normal or obsolete)



```
BindAddress
  type                 leaf
  value_type           uniline
  compute
  migrate_from
  mandatory                          0
  default
  built_in
  convert
  replace
  help
  status               standard      standard
  experience           advanced      beginner
  level                normal        normal
  summary
  description          Use the specifi...
  warp
```

Model GUI

Unknown elements

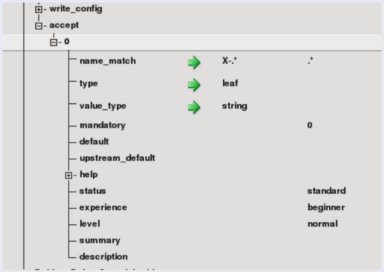
Murphy's law

- Software evolve
- You don't know everything
- X-* parameters

Declare a fallback

Declare condition where an unknown element can be *accepted*

accept specification



```
⊞- write_config
⊟- accept
   ⊟- 0
      ├ name_match        X-.*        .*
      ├ type              leaf
      ├ value_type        string
      ├ mandatory                     0
      ├ default
      ├ upstream_default
      ⊞- help
      ├ status                        standard
      ├ experience                    beginner
      ├ level                         normal
      ├ summary
      └ description
```

Model analysis

- Read the application man pages :
  - Find the structure of the tree
  - Identify configuration parameters, their constraints and relations
  - Decide what to do with unknown parameters (error or accept ?)
  - Identify potential upgrade issues (deprecated parameters mentioned in doc)
- Find several valid examples :
  - To verify that the documentation was understood
  - For the non-regression tests

Model declaration

In summary, configuration documentation is translated into a format usable by Config : :Model :

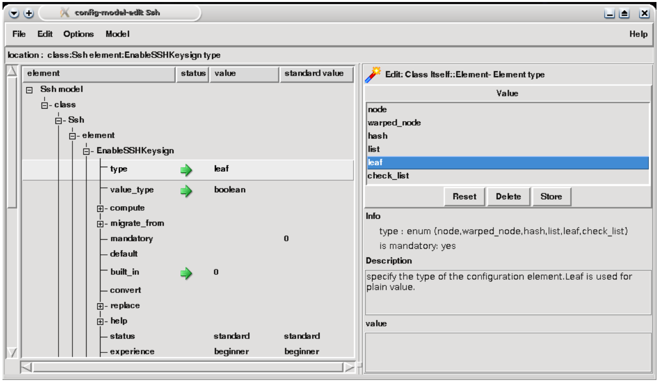
- The structure is translated into configuration classes
- Configuration parameters into elements
- Constraints into element attributes

```
name => 'Ssh',              # class name
element => [
  EnableSSHKeysign => {     # element name
    type => 'leaf',
    value_type => 'boolean',
    built_in => '0',        # default value
    description => 'Setting ...',
  },
]
```

See `http://sourceforge.net/apps/mediawiki/config-model/index.php?title=Creating_a_model`

Declaration (easier mode)

Since writing a data structure is not fun (even with Perl), a model can be created with a GUI :



location : class:Ssh element:EnableSSHKeysign type

```
element                     status  value     standard value
Ssh model
  class
    Ssh
      element
        EnableSSHKeysign
          type                      leaf
          value_type                boolean
          compute
          migrate_from
          mandatory                           0
          default
          built_in                  0
          convert
          replace
          help
          status                    standard  standard
          experience                beginner  beginner
```

Edit: Class Itself::Element- Element type

Value
node
warped_node
hash
list
leaf
check_list

Info
type : enum (node,warped_node,hash,list,leaf,check_list)
is mandatory: yes

Description
specify the type of the configuration element.Leaf is used for plain value.

value

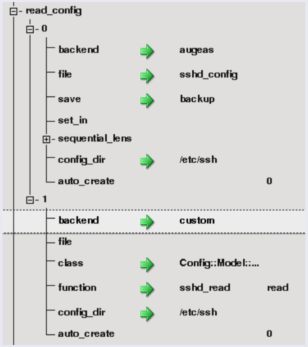
From time to time, do a Menu → Model → test

Reading configuration files

In the model

- Declare the mechanism (*backend*)
  - Built-in (Perl file, Ini file...)
  - Plug-in (Backend class)
  - custom → call-back must also be provided
- Mechanism parameters
- Specifications are tried in order

Example



```
read_config
  0
    backend          augeas
    file             sshd_config
    save             backup
    set_in
    sequential_lens
    config_dir       /etc/ssh
    auto_create                    0
  1
    backend          custom
    file
    class            Config::Model::...
    function         sshd_read     read
    config_dir       /etc/ssh
    auto_create                    0
```

Writing configuration files

In the model

- Not needed if write specification is the same as read
- Same parameters as read spec
- Tried in order until first success

Note

With these specifications, configuration can be migrated from one syntax to another.

Example

```
write_config => [
 {
  backend     => 'augeas',
  save        => 'backup',
  config_dir  => '/etc/ssh',
  file        => 'sshd_config',
 },
 {
  backend    => 'custom',
  class      => 'C::M::OpenSsh',
  function   => 'sshd_write',
  config_dir => '/etc/ssh'
 }
],
```

Prepare configuration updates
For smooth upgrades

For application designers :

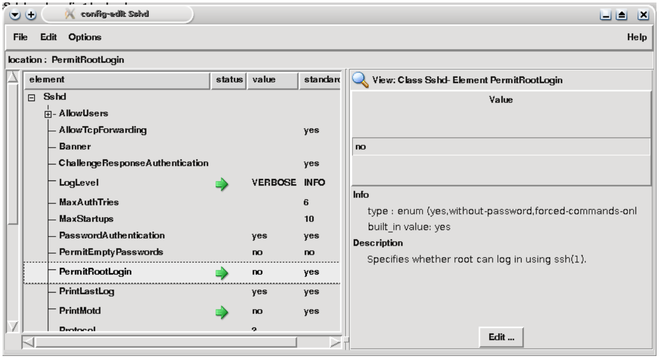
1. No new parameters <-> no new problems
2. Picking parameter name and value : A good name is better than 3 pages of doc
3. Default values : Application can work with an empty config file

But, if needed, model and backend can specify :

- How to replace a value (replace)
- Obsolete parameters (status)
- How to migrate a value (migrate_from + formula)
- Migration from one syntax with another (backends)
- How to accept unknown parameters (e.g. leaf or list ?)

For more information on migration applied to software packages, see `http://wiki.debian.org/PackageConfigUpgrade`

Configuration GUI



location : PermitRootLogin

element	status	value	standard
Sshd			
AllowUsers			
AllowTcpForwarding			yes
Banner			
ChallengeResponseAuthentication			yes
LogLevel	➡	VERBOSE	INFO
MaxAuthTries			6
MaxStartups			10
PasswordAuthentication		yes	yes
PermitEmptyPasswords		no	no
PermitRootLogin	➡	no	yes
PrintLastLog		yes	yes
PrintMotd	➡	no	yes
Protocol		2	

View: Class Sshd- Element PermitRootLogin

Value

no

Info

type : enum (yes,without-password,forced-commands-onl

built_in value: yes

Description

Specifies whether root can log in using ssh(1).

Note : In the menu, change "Option → experience" to show more parameters

Configuration and package upgrades

Package upgrade :

- RedHat : Configuration evolutions leave rpm.new or rpm.save file
- Debian : Configuration evolution either :
  - trigger questions (often cryptic)
  - expose details to user with a diff
  - leave spurious files (dpkg-new or dpkg-old)

In all cases

Merging configuration requires good knowledge from user.

Configuration and package upgrades

Proposal

Use Config : :Model to merge :

- user data from config file
- package/upstream evolutions from model

Models with merge capability can be implemented by :

- Upstream projects
- Distributions (Debian, RedHat ...)
- Derived distribution (Knoppix, SkoleLinux ...)

Each can improve model coming from upstream

See proposal for Debian :
`http://wiki.debian.org/PackageConfigUpgrade`

Migration example

sshd_config : TCPKeepAlive option was formerly called KeepAlive.

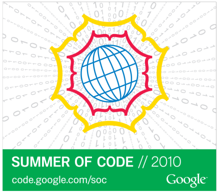
```
KeepAlive => { value_type => 'enum',
               status     => 'deprecated',
               type       => 'leaf',
               choice     => [ 'no', 'yes' ]
             },

TCPKeepAlive => { value_type => 'enum',
                  type       => 'leaf',
                  choice     => [ 'no', 'yes' ],
                  migrate_from
                  => { formula   => '$keep_alive',
                       variables => { keep_alive => '- KeepAlive' },
                     },
```

Coping with new parameter

sshd_config : Accept new parameters, but emit a warning

```
{
  name => 'Sshd',
  ...
  accept => [ {
    name_match => '.*',  # default will match /^.*$/
    type       => 'leaf',
    value_type => 'uniline',
    summary => 'boilerplate parameter that may hide a typo',
    warn => 'Unknow parameter: please make sure there\'s '
          . 'no typo and contact the author'
   }
  ],
},
```



Package upgrade howto

Debian

In package build instructions (*debian/rules* file) :

```
    dh_config_model_upgrade --model_name Sshd \
    --model_package libconfig-model-sshd-perl
```

RedHat

In postinst :

```
    config-edit --model Sshd -ui none -save
```

Project status

Available Models

- OpenSsh
- Approx
- Dpkg Control Copyright
- Krb5
- `Xorg`

Backend

- INI syntax
- Perl
- YAML
- Dpkg control
- Augeas

Community

- Debian packages
- Rpm packages
- Proposal and patches for dh_config (package upgrades)
- Article in GNULinux Mag France
- 2010 GSoC project based on Config : :Model

Future projects

Interfaces

- Search parameters, values and help
- Annotations (e.g. comments) on-going

backend

- JSON
- XML
- Other ?

We need you !

Config : :Model needs your help :

- Integration in distros
- Multi-level configuration
- Plug-in mechanism for models (Xorg drivers)
- Define mechanism for configuration injection (e.g. mercurial viewer in Apache)

Links

- Config : :Model site
  `http://config-model.wiki.sourceforge.net`
- Config : :Model on CPAN
  `http://search.cpan.org/dist/Config-Model/`
- Config : :Model user mailing list `https://lists.sourceforge.net/lists/listinfo/config-model-users`
- GNU/Linux Mag France n°117 and n°120 "Config : :Model - Créer un éditeur graphique de configuration avec Perl" (2 parts)
- Proposal to use Config : :Model to upgrade configuration during Debian package upgrade
  `http://wiki.debian.org/PackageConfigUpgrade`
- Augeas project `http://augeas.net`