Config::Model and configuration upgrades during package upgrade

Dominique Dumont

Debian Perl Group

Feb 2011
Outline

1 Why
   - Configuration upgrade problems
   - Objectives

2 Config::Model
   - Overview
   - Configuration model creation
   - User point of view

3 Package upgrades

4 Status
Configuration is often painful!
Boy, so much doc to read

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...
Configuration is often painful!
Boy, so much doc to read

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

Better user experience:
- Smooth the learning curve
- 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 1: Make configuration easier for users

Better user experience:

- Smooth the learning curve
- 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

Help the developers help the users...

Config tool and upgrader must be easy to maintain:

- Avoid ad-hoc validation code
- Base validation on "meta-data": the configuration model
- Generate interfaces from the model
- Model designed to upgrade user configuration
- GUI to create and maintain models

Minimise code required to read or write config:

- Use existing libraries (Config::Ini, Augeas)
- Provide base classes to help configuration reads and writes
Objective 2: Make maintenance easy for developers
Help the developers help the users...

Config tool and upgrader must be easy to maintain:

- Avoid ad-hoc validation code
- Base validation on "meta-data": the configuration model
- Generate interfaces from the model
- Model designed to upgrade user configuration
- GUI to create and maintain models

Minimise code required to read or write config:

- Use existing libraries (Config::Ini, Augeas)
- Provide base classes to help configuration reads and writes
Config::Model design
Very high level...
What is a model?
Kind of a blue print

Configuration is represented as a tree.
The model defines its structure:
- A config class is a node
- A config parameter a leaf

Each class contains:
- elements (parameters)
- optional: specs to access config file (backend)
- upgrade instructions

model GUI
What is a model?
Kind of a blue print

Configuration is represented as a tree. The model defines its structure:

- A config class is a node
- A config parameter a leaf

Each class contains:

- elements (parameters)
- optional: specs to access config file (backend)
- upgrade instructions
What is a model?
Kind of a blue print

Configuration is represented as a tree. The model defines its structure:
- A config class is a node
- A config parameter a leaf

Each class contains:
- elements (parameters)
- optional: specs to access config file (backend)
- upgrade instructions
Simple elements
For most config parameters

Each element has:
- a type (leaf, hash, list, node)
- constraints (int, max, min...)
- a default value
- a description and a summary (for integrated help)
- an experience level (beginner, advanced, master)
- a status (normal or obsolete)

Model GUI
Unknown elements
S#!t happens

Murphy’s law
- Software evolve
- Easy to miss parameters
- X-* parameters

Declare a fallback
Declare condition where an unknown element can be accepted

accept specification

Dominique Dumont
Unknown elements

S#!t happens

Murphy’s law

- Software evolve
- Easy to miss parameters
- X-* parameters

Declare a fallback

Declare condition where an unknown element can be accepted

accept specification

```
write_config

- accept

- name_match
  - type
  - value_type
  - mandatory
  - default
  - upstream_default

- help
  - status
    - experience
    - level
    - summary
    - description
```

Dominique Dumont

Config::Model
Config warnings and repairs
Between right and wrong

Protect the user
Model can specify:
- warning conditions
- warning message

Help user fix warning
- Declare fix instructions
- Applied by user
- Optional

warning specification

Dominique Dumont
Config warnings and repairs
Between right and wrong

Protect the user
Model can specify:
- warning conditions
- warning message

Help user fix warning
- Declare fix instructions
- Applied by user
- Optional

warning specification

Dominique Dumont
Model analysis

RTFM alert

Read the application man pages

- Find the (hidden) structure of the config tree
- Identify config parameters, their constraints and relations
- Decide: accept or reject unknown parameters?
- Identify potential upgrade issues (deprecated parameters)
- Find upgrade paths

Find several valid examples

- To verify that the documentation was understood
- For the non-regression tests
Model analysis

RTFM alert

Read the application man pages

- Find the (hidden) structure of the config tree
- Identify config parameters, their constraints and relations
- Decide: accept or reject unknown parameters?
- Identify potential upgrade issues (deprecated parameters)
- Find upgrade paths

Find several valid examples

- To verify that the documentation was understood
- For the non-regression tests
Model declaration
The hard way

Config doc is translated into a format usable by Config::Model:

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

### Ssh element

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

See http://search.cpan.org/dist/Config-Model/lib/Config/Model/Manual/ModelCreationIntroduction.pod
Since writing a data structure is not fun (even with Perl), a model can be created with a GUI:

From time to time, do a Menu → Model → test
Read/Write backend
No Perl knowledge required...

Declare a backend in the model from this list:

- YAML
- INI files
- Shell vars like (e.g. ”FOO=bar”)
- Augeas
- Perl data structure

Ask if you need a new backend (mail or bug report)
Designing configuration files
Avoid pitfalls

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
4. Choose the right syntax from a standard: No special conventions
Designing configuration files
Avoid pitfalls

For application designers

1. No new parameters $\leftrightarrow$ 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
4. Choose the right syntax from a standard: No special conventions
Designing configuration files
Avoid pitfalls

For application designers

1. No new parameters $\leftrightarrow$ 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
4. Choose the right syntax from a standard: No special conventions
Designing configuration files
Avoid pitfalls

For application designers

1. No new parameters $\leftrightarrow$ 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
4. Choose the right syntax from a standard: No special conventions
Prepare configuration upgrades
For smooth package updates

Model can specify

- How to replace a value (replace)
- Obsolete or deprecated parameters (status)
- How to migrate deprecated values (migrate_from + formula)
- Migration from one syntax with another (backends)
- Whether to accept unknown parameters

For more information on migration applied to software packages, see http://wiki.debian.org/PackageConfigUpgrade
Prepare configuration upgrades
For smooth package updates

Model can specify
- How to replace a value \textit{(replace)}
- Obsolete or deprecated parameters \textit{(status)}
- How to migrate deprecated values \textit{(migrate\_from + formula)}
- Migration from one syntax with another \textit{(backends)}
- Whether to accept unknown parameters

For more information on migration applied to software packages, see http://wiki.debian.org/PackageConfigUpgrade
Prepare configuration upgrades
For smooth package updates

Model can specify

- How to replace a value (*replace*)
- Obsolete or deprecated parameters (*status*)
- How to migrate deprecated values (*migrate_from + formula*)
- Migration from one syntax with another (*backends*)
- Whether to accept unknown parameters

For more information on migration applied to software packages, see
http://wiki.debian.org/PackageConfigUpgrade
Prepare configuration upgrades
For smooth package updates

Model can specify
- How to replace a value \((\text{replace})\)
- Obsolete or deprecated parameters \((\text{status})\)
- How to migrate deprecated values \((\text{migrate\_from} + \text{formula})\)
- Migration from one syntax with another \((\text{backends})\)
- Whether to accept unknown parameters

For more information on migration applied to software packages, see http://wiki.debian.org/PackageConfigUpgrade
Prepare configuration upgrades
For smooth package updates

Model can specify

- How to replace a value (replace)
- Obsolete or deprecated parameters (status)
- How to migrate deprecated values (migrate_from + formula)
- Migration from one syntax with another (backends)
- Whether to accept unknown parameters

For more information on migration applied to software packages, see http://wiki.debian.org/PackageConfigUpgrade
Benefit of smooth configuration upgrades
Enable upstream evolution

Configuration improvement
With smooth upgrades, configuration files can be improved:
- keyword clarification (to reduce doc)
- re-structuration to match user’s point of view instead of application design
- Even syntax change
Benefit of smooth configuration upgrades
Enable upstream evolution

Configuration improvement

With smooth upgrades, configuration files can be improved:
- keyword clarification (to reduce doc)
- re-structuration to match user’s point of view instead of application design
- Even syntax change
Benefit of smooth configuration upgrades
Enable upstream evolution

Configuration improvement

With smooth upgrades, configuration files can be improved:

- keyword clarification (to reduce doc)
- re-structuration to match user’s point of view instead of application design
- Even syntax change
**Configuration GUI**

Your work may be seen by user

Shown only when user requires it, not during upgrade.

```
<table>
<thead>
<tr>
<th>location</th>
<th>PermitRootLogin</th>
</tr>
</thead>
<tbody>
<tr>
<td>element</td>
<td>status</td>
</tr>
<tr>
<td>Sshd</td>
<td></td>
</tr>
<tr>
<td>- AllowUser</td>
<td></td>
</tr>
<tr>
<td>- AllowTcpForwarding</td>
<td>yes</td>
</tr>
<tr>
<td>- Banner</td>
<td></td>
</tr>
<tr>
<td>- ChallengeResponseAuthentication</td>
<td>yes</td>
</tr>
<tr>
<td>- LogLevel</td>
<td>VERBOSE</td>
</tr>
<tr>
<td>- MaxAuthTries</td>
<td>6</td>
</tr>
<tr>
<td>- MaxStartups</td>
<td>10</td>
</tr>
<tr>
<td>- PasswordAuthentication</td>
<td>yes</td>
</tr>
<tr>
<td>- PermitEmptyPasswords</td>
<td>no</td>
</tr>
<tr>
<td>- PermitRootLogin</td>
<td>no</td>
</tr>
<tr>
<td>- PrintLastLog</td>
<td>yes</td>
</tr>
<tr>
<td>- PrintModLog</td>
<td>no</td>
</tr>
<tr>
<td>- Rhosts</td>
<td></td>
</tr>
</tbody>
</table>
```

**Note:** In the menu, change "Option → experience" to show more parameters.
Other interfaces are available:

- Curses
- Shell like (with 'ls' 'cd' commands)
- Telnet like (command and answers)
- Fuse (each parameter is a file)
Config and package upgrades status
Lackluster situation

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.
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.
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.
Better configuration and package upgrades
Towards a solution?

Proposal

Use Config::Model to merge:

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

Models with merge capability can be implemented by:

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

Each can improve model coming from upstream

See proposal for Debian: http://wiki.debian.org/PackageConfigUpgrade
Better configuration and package upgrades
Towards a solution?

Proposal

Use Config::Model to merge:
- user data from config file
- package/upstream evolutions from distributed model

Models with merge capability can be implemented by:
- Upstream projects
- Distributions (Debian, RedHat …)
- Derived distributions (Knoppix, SkoleLinux …)

Each can improve model coming from upstream

See proposal for Debian: http://wiki.debian.org/PackageConfigUpgrade
Example of model with migration
A parameter was renamed by upstream

sshd_config: TCPKeepAlive option was formerly called KeepAlive.

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

TCPKeepAlive => {
    type => 'leaf', # same spec as KeepAlive
    value_type => 'enum',
    choice => [ 'no', 'yes' ], # default status is 'normal'
    migrate_from => {
        variables => {
            keep_alive => '- KeepAlive' # where is the old param
        },
        formula => '$keep_alive', # how is used the old value
    },
},
```
Example of model with migration
A parameter was renamed by upstream

sshd_config: TCPKeepAlive option was formerly called KeepAlive.

KeepAlive => {
    type => 'leaf',
    value_type => 'enum',
    choice => [ 'no', 'yes' ]
    status => 'deprecated', # KeepAlive is a goner
},

TCPKeepAlive => {
    type => 'leaf', # same spec as KeepAlive
    value_type => 'enum',
    choice => [ 'no', 'yes' ], # default status is 'normal'
    migrate_from => {
        variables => {
            keep_alive => '- KeepAlive' # where is the old param
        },
        formula => '$keep_alive', # how is used the old value
    }
},
Accept unknown parameters... with warning

```yaml
{  
  name => 'Sshd',
  ...
  accept => [  
    {  
      name_match => '.*', # match any unknown parameter
      type => 'leaf',
      value_type => 'uniline',
      summary => 'boilerplate parameter that may hide a typo',
      warn => 'Unknown parameter: please make sure there\’s 
              \n              \n              \n              \n              . \n              \n              \n              \n              \n              . no typo and log a bug'
    }  
  ],
},
```
Package upgrade howto

Minimal change in source package

**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
```
Package upgrade howto
Minimal change in source package

**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 1
Functionalities provided by Config::Model

Available Models
- OpenSsh
- Approx
- Dpkg Control Copyright
- Krb5
- Xorg

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

Framework
- Model composition (plugins)
Project status 1
Functionalities provided by Config::Model

Available Models
- OpenSsh
- Approx
- Dpkg Control Copyright
- Krb5
- Xorg

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

Framework
- Model composition (plugins)
Project status 1

Functionalities provided by Config::Model

Available Models
- OpenSsh
- Approx
- Dpkg Control Copyright
- Krb5
- Xorg

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

Framework
- Model composition (plugins)
Community

- Debian packages
- Mageia packages
- Proposal and patches for dh_config (package upgrades)
- Article in GNU/Linux Mag France
- 2010 GSoC project based on Config::Model
Future projects

**Interfaces**
- Search parameters, values and help
- Annotations (e.g. comments) on-going
- Web UI? (help needed)

**Core**
- Support included config files (à la Apache)
- Configuration aggregation (to reduce redundancies between different configs)
- More doc, examples, blogs, FAQ ...

**Backend**
- JSON
- XML
- Need other?
Future projects

Interfaces
- Search parameters, values and help
- Annotations (e.g. comments) on-going
- Web UI? (help needed)

Core
- Support included config files (à la Apache)
- Configuration aggregation (to reduce redundancies between different configs)
- More doc, examples, blogs, FAQ ...

backend
- JSON
- XML
- Need other?
Future projects

Interfaces
- Search parameters, values and help
- Annotations (e.g. comments) on-going
- Web UI? (help needed)

backend
- JSON
- XML
- Need other?

Core
- Support included config files (à la Apache)
- Configuration aggregation (to reduce redundancies between different configs)
- More doc, examples, blogs, FAQ ...
Your users need you!

Users are waiting for

- No question asked, no slag left after upgrade of their favorite application
- Smooth multi-level configuration (e.g. cascaded models for Debian pure-blend project)
Your users need you!

Users are waiting for

- No question asked, no slag left after upgrade of their favorite application
- Smooth multi-level configuration (e.g. cascaded models for Debian pure-blend project)
Why
Config::Model
Package upgrades
Status

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