WP2: Metamodel for describing system structure and state

Davide Di Ruscio
Dipartimento di Informatica
Università degli Studi dell'Aquila

diruscio@di.univaq.it

Outline

- What is due at T0+12
- Overview of D2.1
  - Scripts analysis
  - MANCOOSI metamodel
  - Supporting the evolution of the metamodel
- Conclusions and future plan

What is due at T0+12

Description of work:

Task 2.1 Find use cases that capture the most used and relevant information that is used while managing complex systems. In particular, given the standard life-cycle of a GNU/Linux system, identify what are the elements and artifacts that should be modeled.

Task 2.2 Develop a metamodel that captures the previously identified elements and that allows a reasonable description of the state of the system and the representation of its evolution over time, taking also into account non-functional aspects. Particular attention will be dedicated to building the metamodel in an adaptable and extensible way, so that it could be used in other contexts besides the reference one.

What is due at T0+12

Deliverables:

Name	Due date	Description
D2.1	t0+12	Metamodel for describing system structure and state.
D2.2	t0+24	Instantiation of the metamodel on a wide-used GNU/Linux distribution.
D2.3	t0+36	Model-based framework for managing the complexity and the state of the GNU/Linux instantiation.

Milestones:

Name	Due date	Description
M2.1	t0+12	First version of the metamodel.
M2.2	t0+24	First version of the model for a given GNU/Linux distribution
M2.3	t0+36	Final version of the framework for a given GNU/Linux distribution and validation.

What is due at T0+12

Deliverables:

Name	Due date	Description
D2.1	t0+12	Metamodel for describing system structure and state.
D2.2	t0+24	Instantiation of the metamodel on a wide-used GNU/Linux distribution.
D2.3	t0+36	Model-based framework for managing the complexity and the state of the GNU/Linux instantiation.

Milestones:

Name	Due date	Description
M2.1	t0+12	First version of the metamodel.
M2.2	t0+24	First version of the model for a given GNU/Linux distribution
M2.3	t0+36	Final version of the framework for a given GNU/Linux distribution and validation.

Overview of D2.1

Title

Metamodel for describing system structure and state

Internal review

Sophie Cousin, Arnaud Laprevote, Paulo Trezentos

Overview of D2.1

1. Introduction
2. Standard life-cycle of FOSS distributions
3. Models for supporting the upgradeability of FOSS distrubutions
4. Analysis of FOSS distributions
5. MANCOOSI metamodel
6. Supporting the evolution of the MANCOOSI metamodel
7. Conclusions

A debhelper autoscript templates

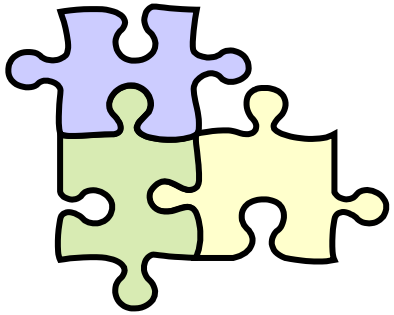
B Fedora “autoscript” snippets

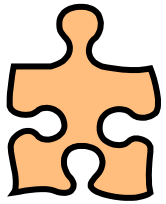
C Mandriva macros

Overview of D2.1

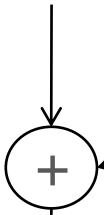
Standard life-cycle of FOSS distributions

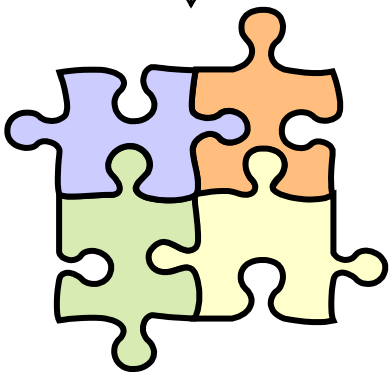
Configuration 1





package
libapache2-mod-php5





Configuration 2

Standard life-cycle of FOSS distributions


```
# apt-get install libapache2-mod-php5
```

(1) request

```
Reading package lists... Done
Building dependency tree... Done

The following NEW packages will be installed:
  libapache2-mod-php5
0 upgraded, 1 newly installed, 0 to remove and
  0 not upgraded.
Need to get 2543kB  of archives. After this
operation, 5743kB of additional disk space
will be used.
```

(2) dep. resolution

```
Get:1 http://va.archive.ubuntu.com
hardy-updates/main libapache2-mod-php5
5.2.4-2ubuntu5.3 [2543kB]
Fetched 2543kB in 2s(999kB)
```

(3) package retrieval

(5a) pre-configuration

```
Selecting package libapache2-mod-php5.
(Reading database ... 162440  files and
dirs installed.)
Unpacking libapache2-mod-php5
(from .../libapache2-mod-php5_5.2.4-2
ubuntu5.3_i386.deb)
```

(4) unpacking

```
Setting up libapache2-mod-php5
(5.2.4-2ubuntu5.3)
```

(5b) post-configuration

Standard life-cycle of FOSS distributions

» The installation of php5 executes the code defined in the following *postint* script

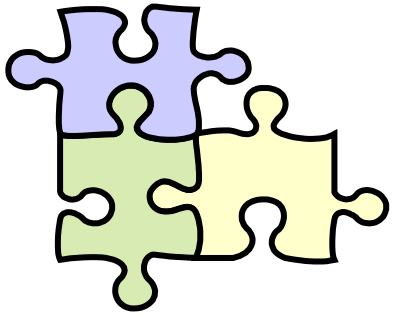
```
1 #!/bin/sh
2 if [ -e /etc/apache2/apache2.conf ] ; then
3     a2enmod php5 >/dev/null || true
4     reload_apache
5 fi
```

» The Apache module php5, installed during the unpacking phase, gets enabled by the above snippet invoking the *a2enmod* command

» The Apache service is then reloaded to make the change real

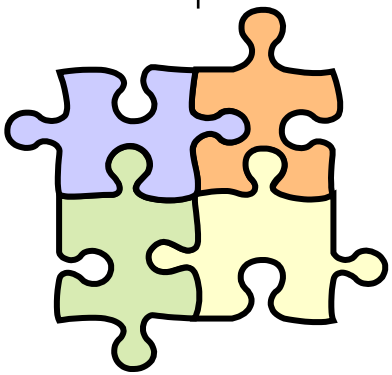
Standard life-cycle of FOSS distributions

Configuration 1




```
apt-get remove libapache2-mod-php5
```

Configuration 2



Standard life-cycle of FOSS distributions

» Upon php5 removal the reverse will happen, as implemented by the php5 *prerm* script:

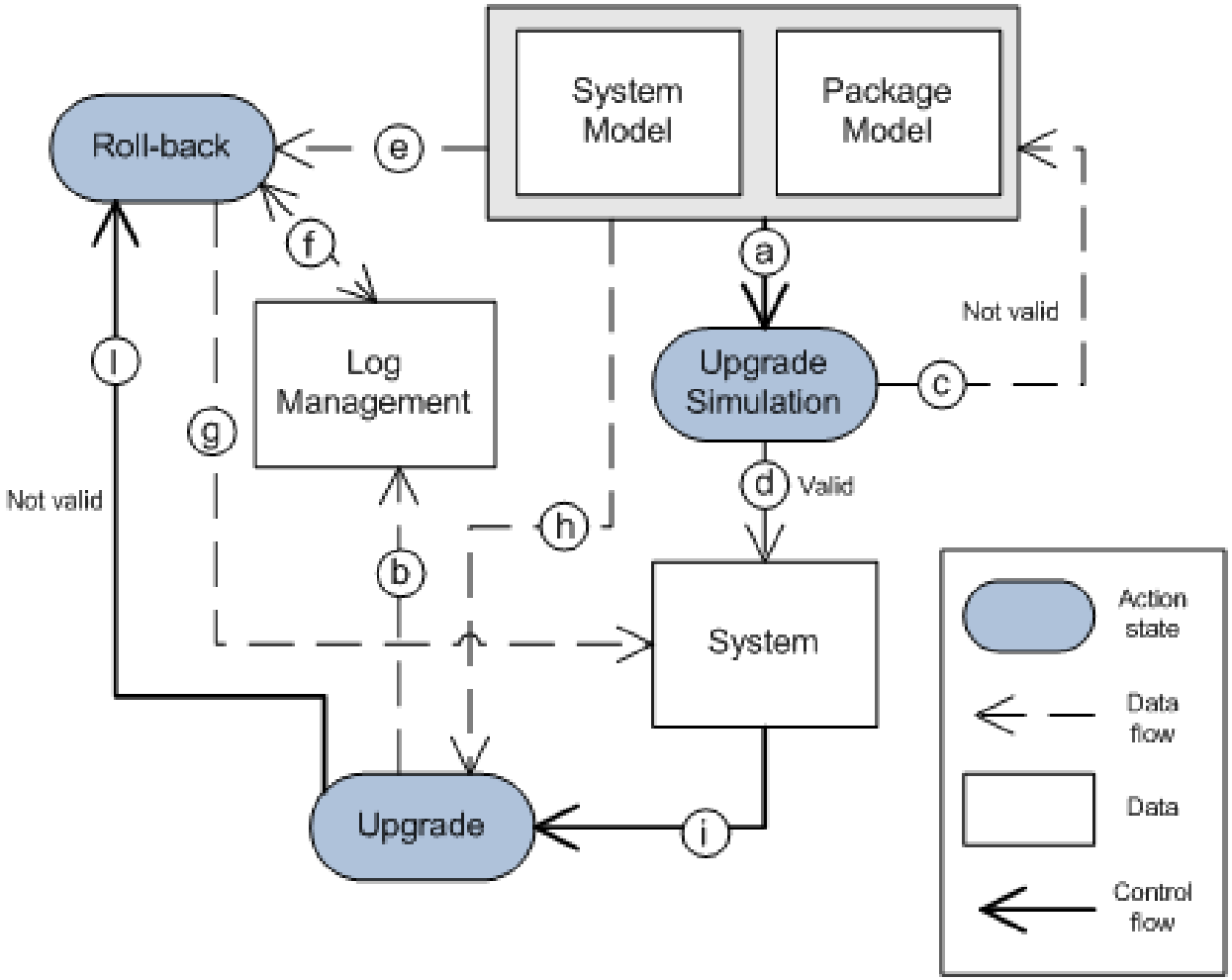
```
1 #!/bin/sh
2 if [ -e /etc/apache2/apache2.conf ] ; then
3     a2dismod php5 || true
4 fi
```

Standard life-cycle of FOSS distributions

- *prerm* is executed before removing files from disk, that is necessary to avoid reaching an inconsistent configuration where the Apache server is configured to rely on no longer existing files
- Inter-package dependencies is not enough to encode this kind of situation
  - Apache does not depend on php5 (and should not, because it is useful also without it), but while php5 is installed, Apache needs specific configuration to work in harmony with it
  - The bookkeeping of such configuration is delegated to maintainer scripts

Models for supporting the upgradeability of FOSS distributions

» A model-driven approach to manage the upgrade of FOSS distributions is proposed



Overview of D2.1

4. Analysis of FOSS distributions

- 4.1 Maintainer script analysis: Debian GNU/Linux
  - 4.1.1 Scripts generated from helpers
  - 4.1.2 Analysis of scripts "by hand“
- 4.2 Maintainer script analysis: RPM
- 4.3 Stemming out the elements to be modeling
- 4.4 Uncovered elements
- 4.5 Conclusions

Analysis of FOSS distributions

- The procedure we followed for clustering consists of the following steps:
  - Identify scripts entirely generated
  - Identify scripts only composed of blank lines, comments, etc
  - Study of scripts written “by hand“. In this respect, we have collected scripts in clusters to concentrate the analysis on the representative of each identified equivalence class

Maintainer script analysis: Debian GNU/Linux

- The Debian Lanny distribution consists of 23’823 packages
- All the five kinds of scripts have been considered (*preinst*, *postinst*, *prerm*, *postrm*, *config*)
- Considering 5 maintainer scripts per package we obtain a (potential) universe of 114’115 scripts (23’823 x 5)
- Luckily, the universe of the scripts to be studied consists of 25’440 (22,3 %) since 88’675 (77,7%) of the potential universe are missing

Maintainer script analysis: Debian GNU/Linux

- Debian uses *debhelper* tools which are invoked at package build time to automate package-construction tasks
- *debhelper* generates the script code by composing together sequentially one or more of the 52 available templates
- From the point of view of the package maintainer, the template machinery can be completely ignored as long as all needs of writing maintainer scripts are addressed by *debhelper*

Maintainer script analysis: Debian GNU/Linux

- For instance, in the specific case of shared libraries, the debhelper tool *dh_makeshlibs* is used
- It takes care of adding the following two script snippets to the *postinst* and *postrm* maintainer scripts, respectively

postinst-makeshlibs

```
1 if [ "$1" = "configure" ]; then
2     ldconfig
3 fi
```

postrm-makeshlibs

```
1 if [ "$1" = "remove" ]; then
2     ldconfig
3 fi
```

Maintainer script analysis: Debian GNU/Linux

» When the maintainer needs to add specific code to maintainer scripts, which is not provided by templates, *debhelper* enables mixing generated lines with lines written by hand

```
#!/bin/sh
set -e

case "$1" in
  purge | remove)
  [ ! -L /etc/udev/rules.d/xenomai.rules ] || rm /etc/udev/rules.d/xenomai.rule\
s
  ;;
esac

# Automatically added by dh_makeshlibs
if [ "$1" = "remove" ]; then
        ldconfig
fi
# End automatically added section
```

Maintainer script analysis: Debian GNU/Linux

	n. of scripts	LOCs
non-blank	25’440 (100%)	386’688 (100%)
generated (non-blank)	16’379 (64.38%)	162’074 (41.9%)
by hand (non-blank)	9’061 (35.62%)	224’614 (58.1%)

» About 2/3 of all the maintainer scripts are composed only of lines generated using the template mechanism

Maintainer script analysis: Debian GNU/Linux

	n. of scripts	LOCs
non-blank	25’440 (100%)	386’688 (100%)
generated (non-blank)	16’379 (64.38%)	162’074 (41.9%)
by hand (non-blank)	9’061 (35.62%)	224’614 (58.1%)

» About 2/3 of all the maintainer scripts are composed only of lines generated using the template mechanism

Analysis of scripts “by hand”

» All the 9’061 scripts obtained in the previous step have been clustered in groups

» Each group collects scripts that contain exactly the same statements

Group	Occurrence	Representative script name
G1	93	libk/libkpathsea4_2007.dfsg.2-4_amd64.deb.preinst
G2	54	d/dict-freedict-swe-eng_1.3-4_all.deb.postinst
G3	54	d/dict-freedict-fra-deu_1.3-4_all.deb.postrm
G12	24	e/education-geography_0.837_amd64.deb.postrm
G13	24	e/education-development_0.837_amd64.deb.postinst
G14	23	libt/libtcd-dev_2.2.2-1_amd64.deb.postinst
G15	22	g/gpppon_0.2-4+b1_amd64.deb.postinst

Analysis of scripts “by hand”

» The groups are in turn refined in order to find possible recurrent templates

» For instance there are 23 scripts in G14 which contains the following code

```
1  set -e
2  case "$1" in
3      configure)
4      ;;
5      abort-upgrade|abort-remove|abort-deconfigure)
6      ;;
7      *)
8          echo "postinst called with unknown argument \`$1'" >&2
9          exit 1
10     ;;
11 esac
12 exit 0
```

Analysis of scripts “by hand”

» By removing the statements *set –e* and *exit* 0 we defined the following template

```
1  case "$1" in
2      configure)
3      ;;
4      abort-upgrade|abort-remove|abort-deconfigure)
5      ;;
6      *)
7          echo "postinst called with unknown argument \`$1'" >&2
8          exit 1
9      ;;
10 esac
```

which has 69 matches instead of 23

Analysis of scripts “by hand”

- The same approach has been applied to the other groups by inducing the definition of 116 templates
- Then the templates have been studied in order to identify their similarities
- Depending on the template similarities, templates are grouped in *classes*

Analysis of scripts “by hand”

» For instance, *Template2* and *Template4* belong to the same class since their difference is minimal and consists of their *exit* statement only

Template2

```
case "$1" in
    configure)
    ;;
    abort-upgrade|abort-remove|abort-deconfigure)
    ;;
    *)
        echo "postinst called with unknown argument \`$1'" >&2
        exit 1
    ;;
esac
```

Template4

```
case "$1" in
    configure)
    ;;
    abort-upgrade|abort-remove|abort-deconfigure)
    ;;
    *)
        echo "postinst called with unknown argument \`$1'" >&2
        exit 0
    ;;
esac
```

Maintainer script analysis: RPM

- The analysis started from the *spec* files of Fedora
- The universe of the considered packages consists of 4'704 spec files
- From each spec file we extracted the install and uninstall scripts indicated by the keywords *%pre*, *%post*, *%preun*, *%postun*
- We applied the same procedure we used for Debian on a potential universe of 18'816 scripts (4'704*4)
- Actually, the existing scripts are 2'039, that is 10,8% of the potential universe

Maintainer script analysis: RPM

- Similarly to Debian, Fedora makes use of templates for producing maintainer scripts
- By considering the documentation we found 23 templates
- Unfortunately, the templates are not properly delimited in the scripts where they are used
- The analysis has been completely performed by hand
- Also in Fedora, templates play a key role; 1038 scripts are completely generated from them (51% of the total)

Stemming out the elements to be modeled

» We found classes of templates which helped us to define the following modeling constructs for specifying the scripts behavior:

- **File system statements**, that are statements which modify the state of the file system (e.g., rm, copy, mv, etc.)
- **Environment statements**, that are statements which affect the services, shared libraries, modules of a system (e.g., update_menus, depmod, ldconfig)
- **Package setting statements**, that are statements which modify the configuration of packages (e.g., a2enmod that modify the settings of the Apache package)
- **Neutral statements**, that do neither modify the file system nor the environment nor the package settings (e.g. messages, comments, exit)
- **Control statements**, to give an explicit control flow specifying statement executions

Stemming out the elements to be modeled

» The recurring control statements which have been identified are the following:

- **Conditional statement**
- Specialized case statement which, depending on the script in which it is located, can be in turn classified into:
  - **Case postinst**: the admitted conditions are *configure, abort-upgrade, abort-remove, abort-deconfigure*
  - **Case postrm**: the admitted conditions are *purge, remove, upgrade, failed-upgrade, abort-install, abort-upgrade, disappear*
  - **Case prerm**: the admitted conditions are *remove, upgrade, deconfigure, failed-upgrade*
  - **Case preinst**: the admitted conditions are *install, upgrade, abort-upgrade*

Stemming out the elements to be modeled

- **Generic case statement**
- **Iterator statement** which, depending on the considered collection, can be in turn classified into:
  - > **Directory iterator**
  - > **Lines of a file iterator**
  - > **Enumeration iterator**
  - > **Input parameter iterator**
  - > **Word iterator**

» Each template (used by debhelper or rpmbuilder) induces a modeling construct

Overview of D2.1

5. MANCOOSI metamodel

5.1 Configuration metamodel

5.2 Package metamodel

5.2.1 Script metaclass

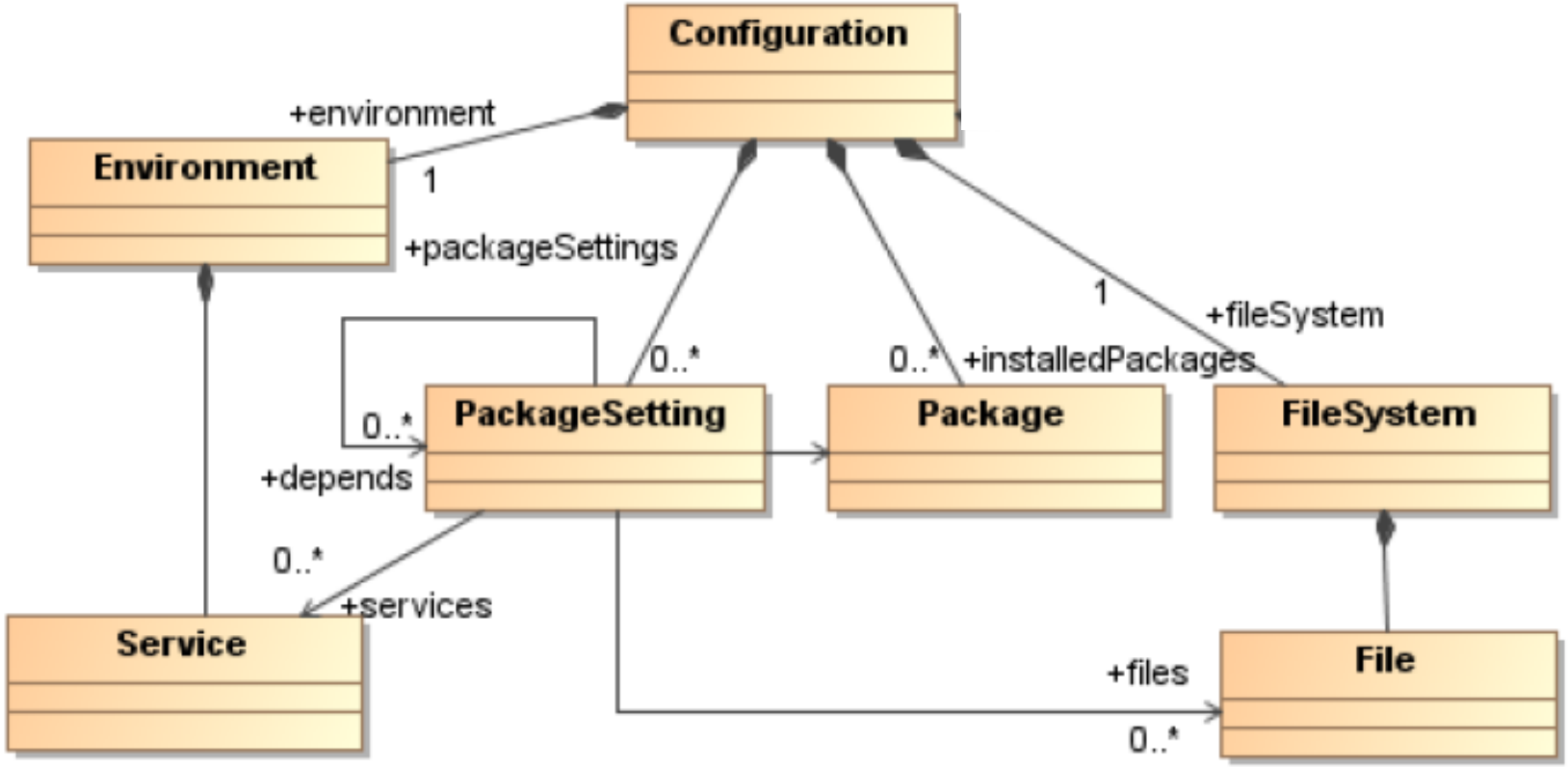
5.2.2 Statement metaclass

- If metaclass
- Case metaclass
- Iterator metaclass
- Return metaclass
- Debhelper template metaclasse

5.3 Log metamodel

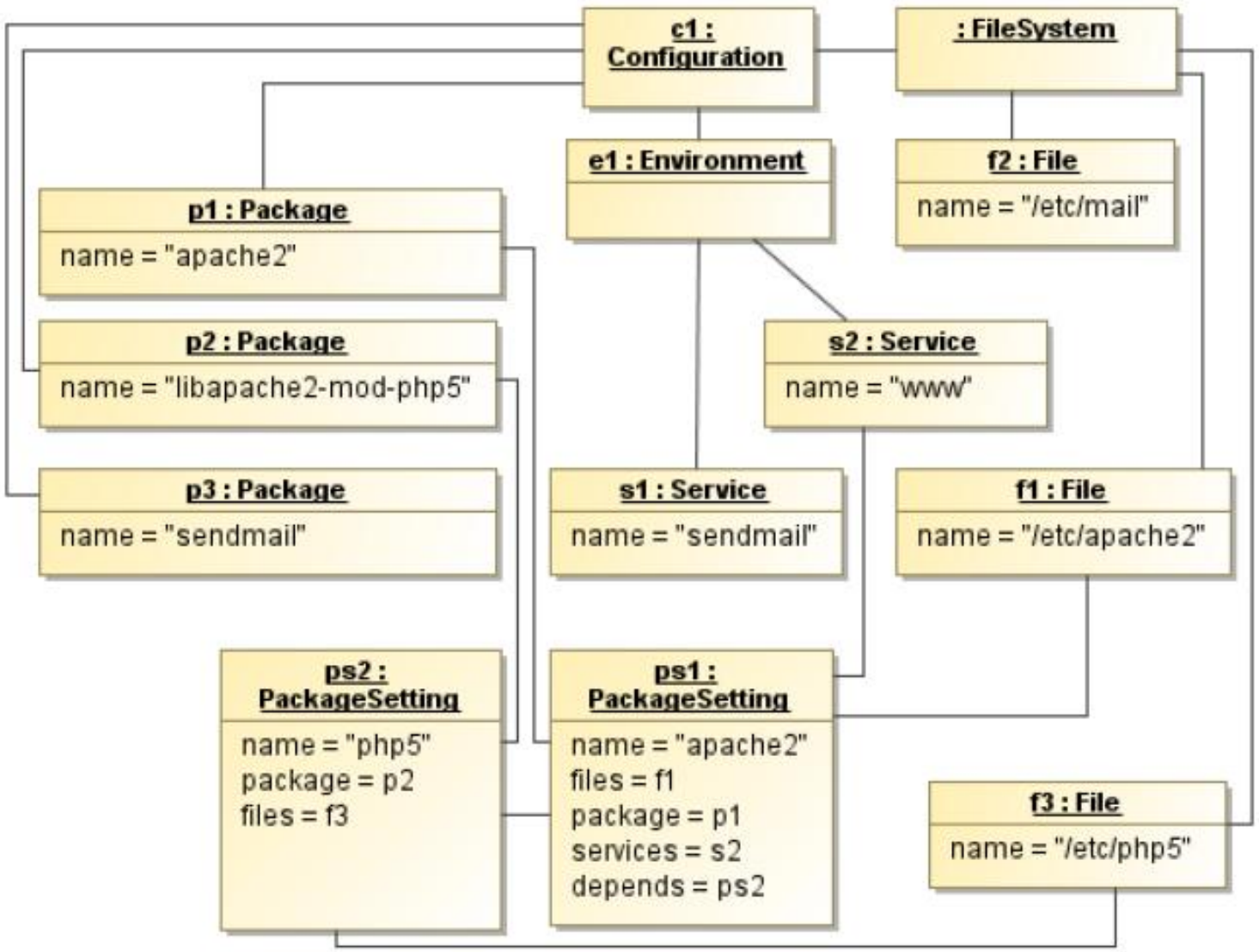
5.4 Conclusions

MANCOOSI metamodel: Configuration



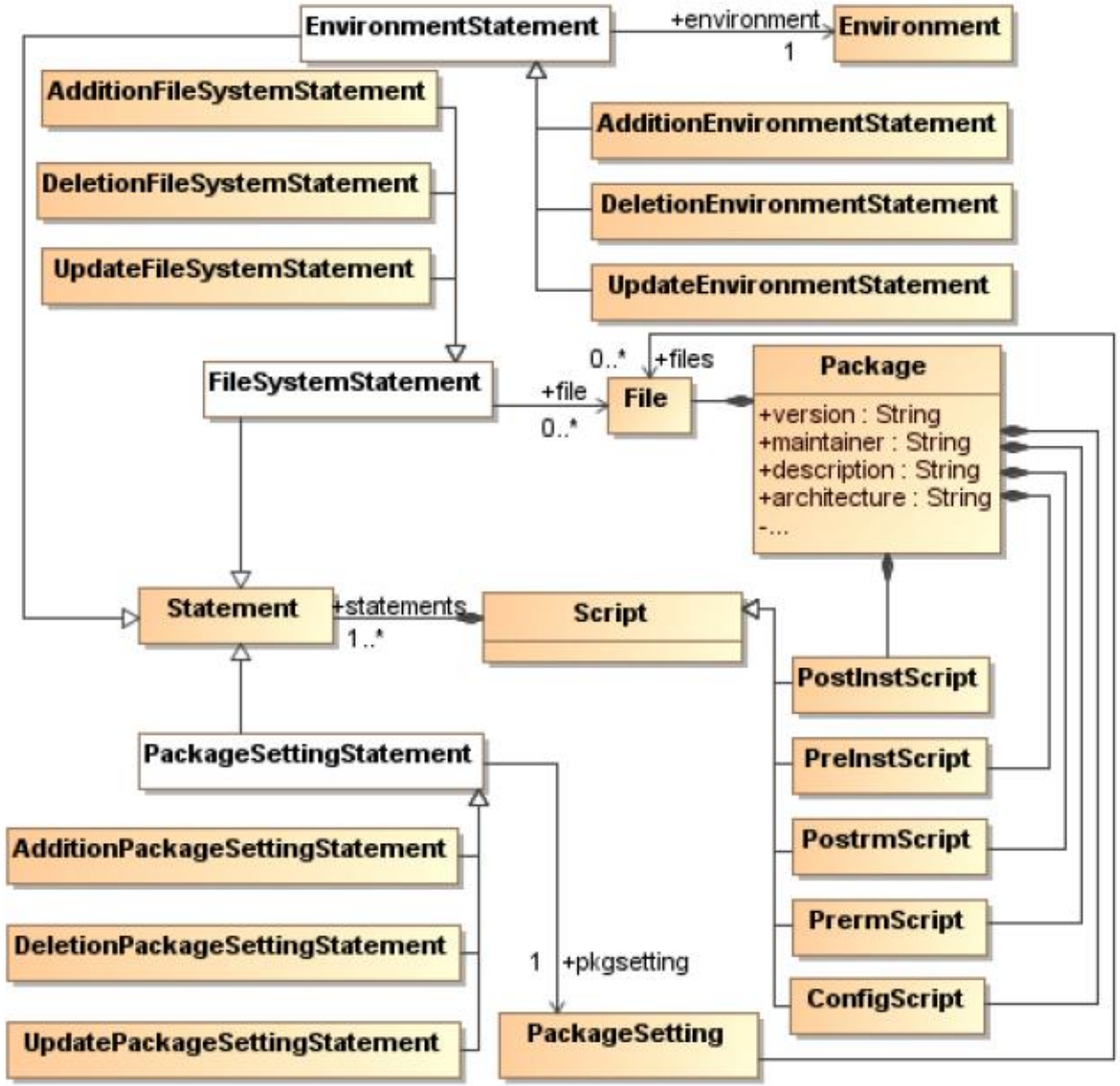
Fragment of the Configuration metamodel

MANCOOSI metamodel: Configuration



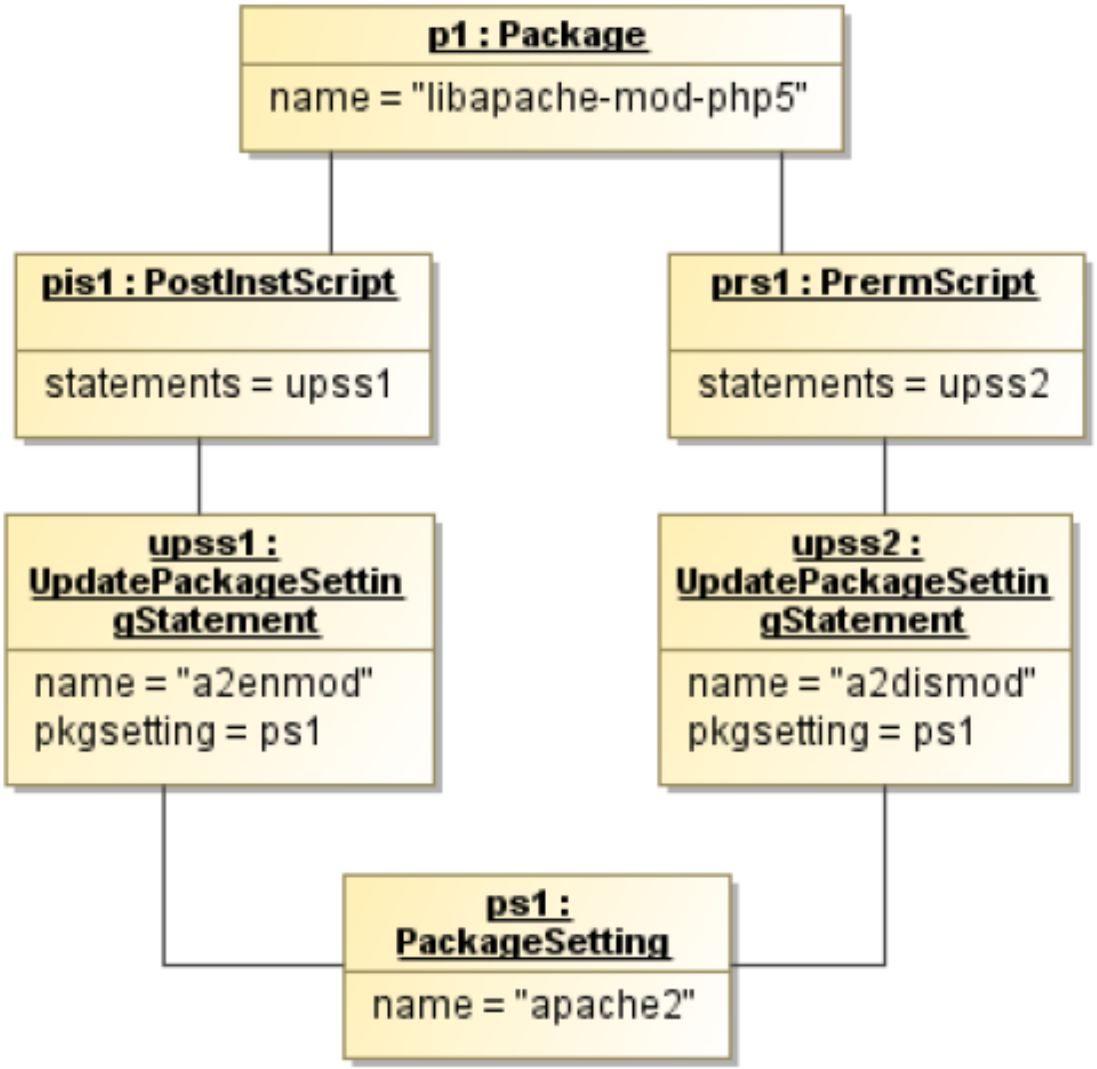
Sample Configuration model

MANCOOSI metamodel: Package



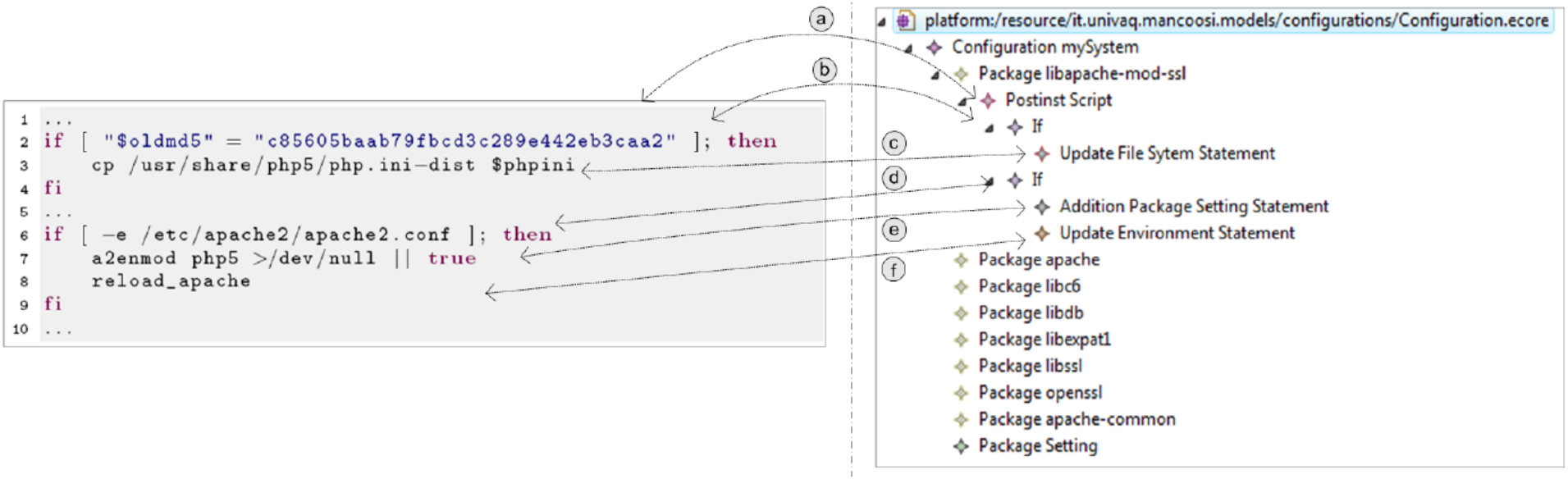
Fragment of the Package metamodel

MANCOOSI metamodel: Package



Sample Package model

MANCOOSI metamodel: Package

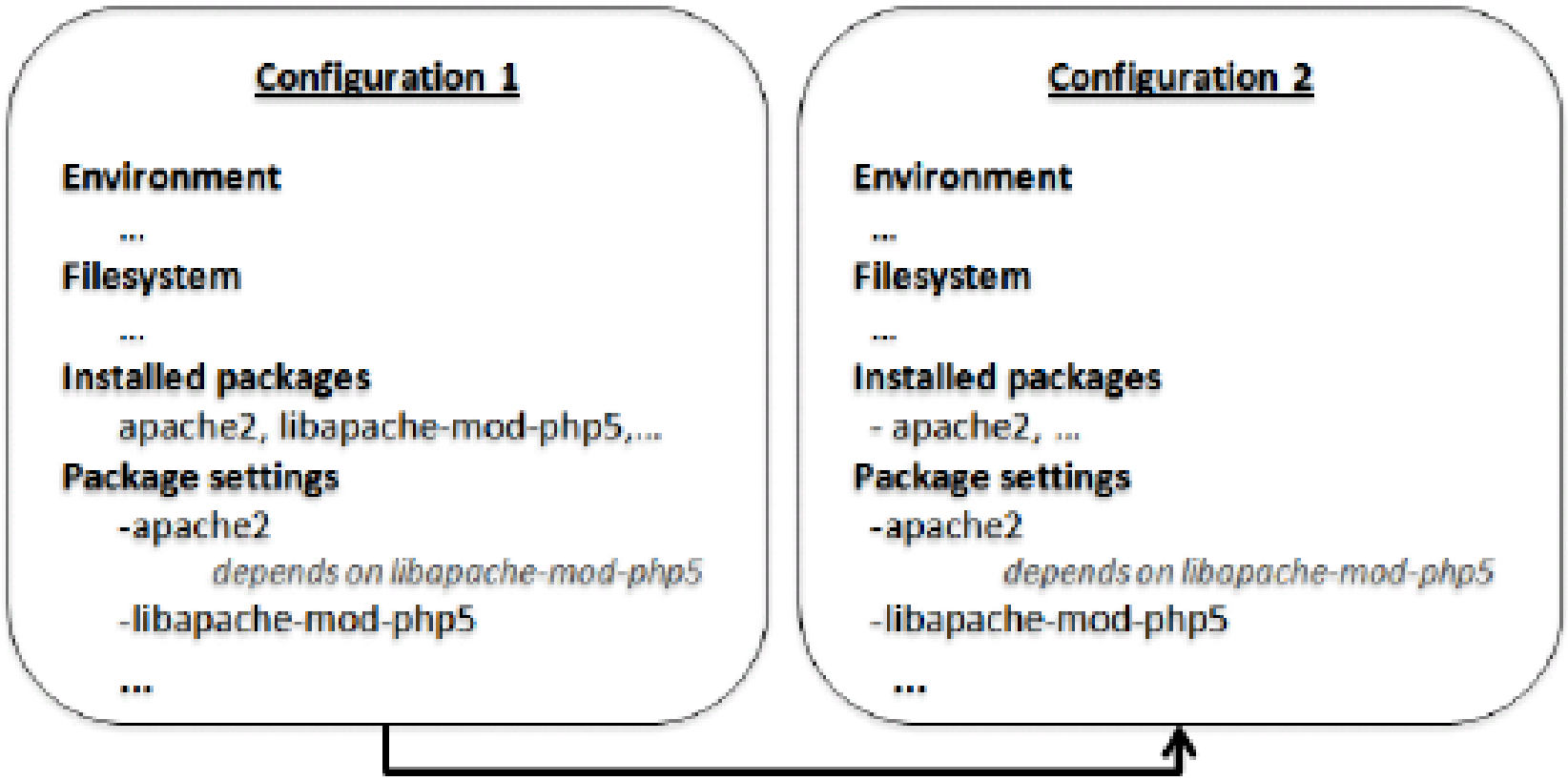



```
1  ...
2  if [ "$oldmd5" = "c85605baab79fbcd3c289e442eb3caa2" ]; then
3      cp /usr/share/php5/php.ini-dist $phpini
4  fi
5  ...
6  if [ -e /etc/apache2/apache2.conf ]; then
7      a2enmod php5 >/dev/null || true
8      reload_apache
9  fi
10 ...
```

Fragment of the libapache2-mod-php5 5.2.6-5 amd64.deb.postinst script

MANCOOSI metamodel: Package

» Currently, the package managers are not able to predict inconsistencies like the following

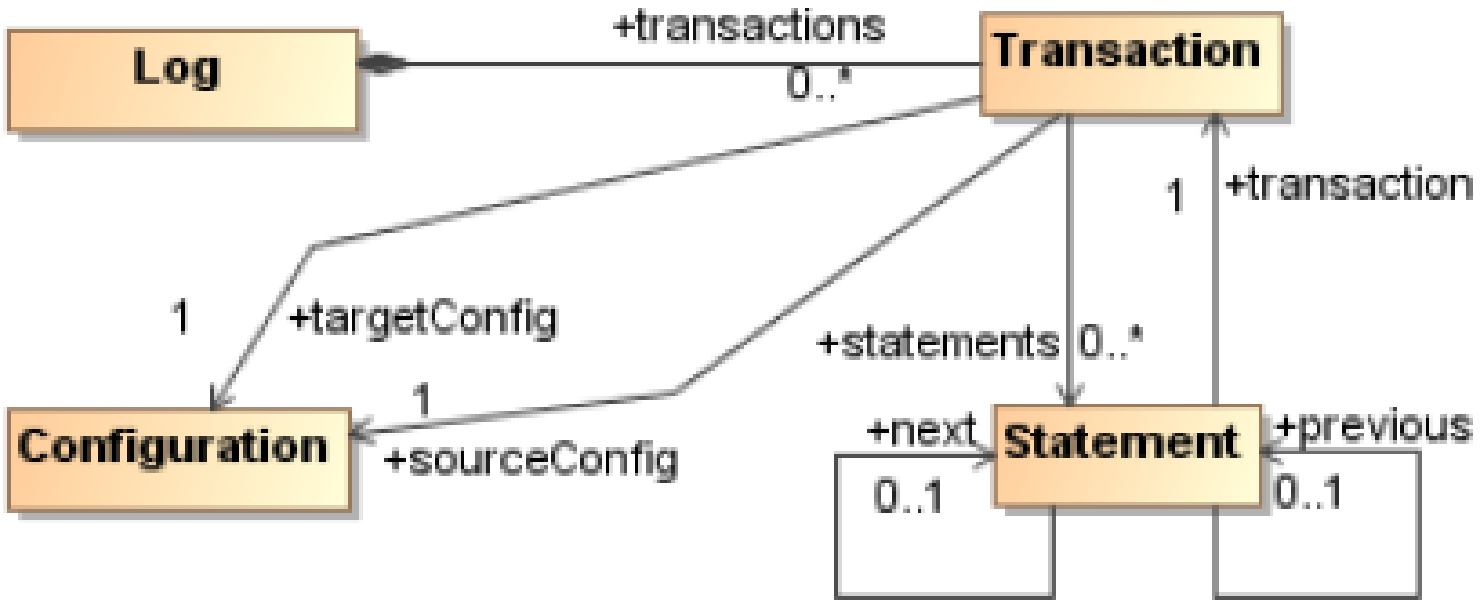


MANCOOSI metamodel: Package

- The Package metamodel gives the possibility to specify an abstraction of the involved maintainer scripts which are executed during the package upgrades

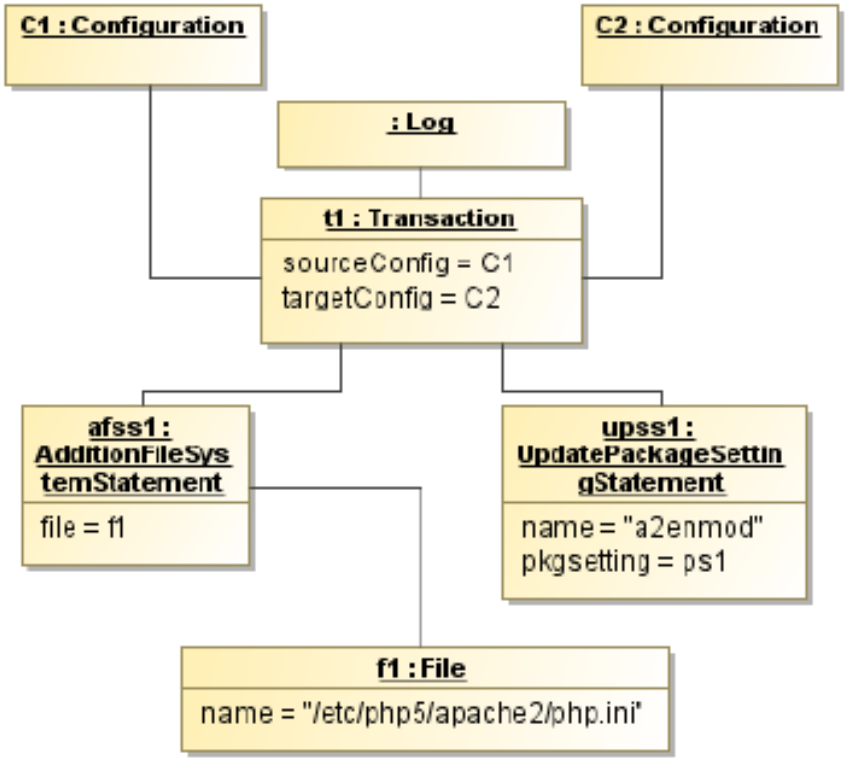
- Consistence checking possibilities are increased and trustworthy simulations of package upgrades can be done

MANCOOSI metamodel: Log



Log Metamodel

MANCOOSI metamodel: Log



Sample Log model

» The installation of the package libapache-mod-php5

- modifies the file system (statement afss1 which represents the addition of the file f1)
- updates the Apache configuration (element upss1)

MANCOOSI metamodel: Log

» The usefulness of log models is manifold:

a) *Preference roll-back*: the user wants to recover a previous configuration, for whatever reason

b) *Compensate model incompleteness*: upgrade simulation is not complete with respect to upgrades, and undetected failures can be encountered while deploying upgrades on the real system. Log models come into play and enable rolling back deployed changes to bring the system back to a previous valid configuration

Overview of D2.1 (4/4)

Conclusions and future plan

- A model-driven approach to manage the upgrade of FOSS distributions has been proposed
- In order to define the modeling constructs which are required to specify system structure and states some FOSS systems have been analyzed
- The MANCOOSI metamodel has been proposed to support:
  - Simulation of package installations
  - Roll-back management
- Trustworthy simulation has to consider the behavior of the maintainer scripts which are executed during package upgrades

Conclusions and future plan

- The proposed metamodels provide with the modeling construct to specify an abstraction of the maintainer script behaviors
- The proposed metamodels will be the foundation to define a new Domain Specific Language (DSL) to specify the behavior of the scripts
- The definition of the DSL consists of
  - abstract syntax,
  - concrete syntax and
  - dynamic semantics

Conclusions and future plan

- While the abstract syntax is given through the metamodels, the concrete syntax and the dynamic semantics are among the main objectives of the deliverable D3.2
- We will instantiate the metamodel on a widely used GNU/Linux distribution, according to Deliverable D2.2
- We will implement a model-based framework for managing the complexity and the state of the GNU/Linux instantiation, according to Deliverable D2.3

References

» The analysis is available in the svn repository at the following address:

https://gforge.info.ucl.ac.be/svn/mancoosi/trunk/uda/script-analysis

» The metamodel has been implemented on the Eclipse platform and the corresponding project is available at:

https://gforge.info.ucl.ac.be/svn/mancoosi/trunk/uda/eclipseProjects/it.univaq.mancoosi.metamodel