mancoosi
managing software complexity

Extension of an existing package manager to produce traces of ugradeability problems in CUDF format

Deliverable 5.2

Nature : Deliverable
Due date : 31.07.2010
Start date of project : 01.02.2008
Duration : 40 months

SEVENTH FRAMEWORK PROGRAMME

Information Society Technologies

Specific Targeted Research Project
Contract no.214898
Seventh Framework Programme: FP7-ICT-2007-1

A list of the authors and reviewers

Project acronym	Mancoosi
Project full title	Managing the Complexity of the Open Source Infrastructure
Project number	214898
Authors list	Pietro Abate André Guerreiro Stéphane Laurière Ralf Treinen Stefano Zacchiroli
Internal review	Gustavo Gutierrez
Workpackage number	WP5
Deliverable number	2
Document type	Deliverable
Version	1
Due date	31/07/2010
Actual submission date	02/08/2010
Distribution	Public
Project coordinator	Roberto Di Cosmo

Abstract

One of the objectives of the Mancoosi project is to resolve some of the problems that users of Free and Open Source Software distributions experience when trying to install, remove, or upgrade packages installed on their machines. The specific goal of Workpackage 5 is to to build a data base of problem reports generated from such user requests to a meta-installer, which then will be used by the Mancoosi project, and the research community in general, to develop better algorithms to compute upgrade paths.

The design of the languages used to produce these error reports have been described in an earlier deliverable. The format of the report produced on a user machine is specific to the software distribution used, but follows a general project-wide scheme called DUDF (Distribution Upgradeability Description Format). Problem reports are uploaded to a server specific to the software distribution and translated by the distribution editor into a common format called CUDF. The *Common Upgradeability Description Format* (CUDF) is a format for describing upgrade problems independently of a specific FOSS distribution. Reports in CUDF format are then transferred from the distribution editor's server to a central server of the Mancoosi project, where they will be used in the construction of a project-wide problem database.

This document summarizes the work done by three different GNU/Linux distributions Debian, Mandriva and Caixa Mágica to generate problem reports on user machines in DUDF, transfer them to the distribution editors, and to convert them there into the common format CUDF.

Contents

List of Figures

Chapter 1

Introduction

Modern software systems are deployed in the form of a collection of components from which users may chose the software packages that they wish to install on a machine. This choice is not permanent and is subject to modification when a user applies updates, installs additional software in order to satisfy new requirements, removes unwanted software, or replaces one software component by an alternative component that provides the same functionality. In Free and Open Source Software (FOSS) collections, the set of available software components itself is rapidly changing, which adds further dynamics to the scenario.

In the FOSS world, software components are commonly called *packages*, and a collection of packages is called a *distribution*. It is the job of the *distribution editor* to create and maintain a coherent distribution. Each distribution editor chooses his format of *metadata* that describes some abstract properties of the packages in the distribution. The most important pieces of information in the package metadata are the name and version number of a package, its requirements, what it conflicts with, and the features it provides. In the GNU/Linux world, two families of metadata formats are most commonly used: the RPM format which stems from the Redhat distribution, and the Debian format defined by the distribution editor of the same name. There are some important differences not only between these two families of metadata formats, but also between different instances of these as defined by different distribution editors. Even when syntactic similarities might lead to the impression that the formats are more or less the same there still might be an essential difference in the *semantics* of the metadata.

One of the objectives of the Mancoosi project is to resolve some of the problems that users experience when trying to modify the installation status of packages on their machine. A request to change that installation status may come in the form of an installation, upgrade, or removal request, or as a combination of these. In the remainder of this document we will refer to any of these requests commonly as *upgrade requests*. In particular, Workpackage 4 of the Mancoosi project aims at developing better algorithms for finding solutions (in terms of versions of packages to install and remove) of such a user request. In Workpackage 5, we will build a database of problem reports regarding failed attempts to modify the installation status of the packages that where produced with instrumented versions of currently popular tools. This database will then be available as data set to to researchers working on better algorithms, both in Workpackage 4 of the Mancoosi project, and researchers who do not participate in Mancoosi.

A user wishing to change the installation of packages on his machine issues an upgrade request to a tool that knows about the currently installed and available software packages. Based on

this information the tool decides on a set of packages (as precise versions) to install, upgrade or remove. We call such a tool a *meta-installer* in order to distinguish it from low-level *installers*. Meta-installers invoke the low-level installers in order to remove a certain package, or to install a certain package on the machine. Examples of meta-installers common in the GNU/Linux universe are `apt-get`, `aptitude`, `URPMI`, `apt-rpm`, `smart`, and `cupt`, while examples low-level installers are `rpm` and `dpkg`.

The overall architecture of the infrastructure is described in Figure 1.1 and defined in Deliverable 5.1 [TZ08]:

1. The meta-installer available on user machines will be instrumented so that they can generate a report of a failed package upgrade attempt. The format of the report is specific to the software distribution used, but follows a general project-wide scheme called DUDF (Distribution Upgradeability Description Format, see Section 1.2).
2. Problem reports are uploaded to a server specific to the software distribution.
3. Collected reports are translated by the distribution editor into a common format called CUDF (Common Upgradeability Description Format, see Section 1.3).
4. Translated reports in CUDF format are transferred from the distribution editor's server to a central server of the Mancoosi project, where they will be used in the construction of a project-wide problem database.

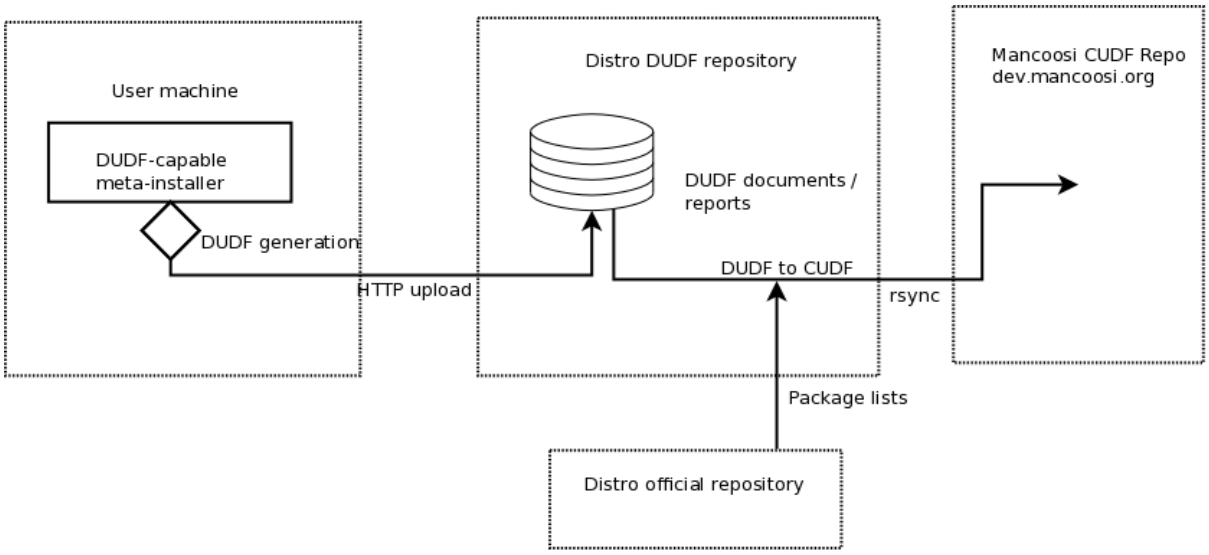


Figure 1.1: Reporting infrastructure

The current deliverable describes this process up to the point where CUDF documents are obtained by the distributor editors by translation of DUDF problem reports submitted by their respective users. The subsequent stages of transferring CUDF documents to the project-wide server and the establishment of the problem data base will be subject of Deliverable 5.3 [511].

1.1 Anonymity of Submissions

The objective is to collect specific upgrade attempts to make them available to Mancoosi and the community. DUDF generators are available in the distributions Debian, Mandriva and Caixa

Mágica but are not installed by default. Their installation and activation is a deliberate decision of the system administrator. Data is collected anonymously; no user information is embedded in the generated DUDF documents. However, in order to track requests per machine, a host identifier key (hostid) might be embedded in the DUDF document. The hostid is a random string which is generated once during the installation of the utility and stored on the system. This key, together with a unique document identifier which is generated randomly each time a new document is generated, ensures uniqueness of the submission, anonymity of the sender and traceability of the DUDF document.

1.2 Overview of the DUDF format

The *Distribution Upgradeability Description Format (DUDF)* was defined in [TZ08, TZ09c]. The overall structure of a DUDF document that is used to report a problem is as follows (see also Figure 1.2). Note that some parts of the specification are specific to the distribution and hence are not defined in the generic DUDF format.

Timestamp a timestamp to record when the upgrade problem was generated.

Problem identifier (i.e. *uid*) a string used to identify this problem submission uniquely among all submissions sent to the same distribution editor.

Package status (i.e. *installer status*) the status of packages currently installed on the user machine. This item is installer-specific, but may also contain data specific to the meta-installer in case the meta-installers maintain some additional information (for instance, the reason why a certain package was installed).

Package universe the set of all packages which are known to the meta-installer, and are hence available for installation. This item is specific to the meta-installer, system architecture and distribution.

Requested action the modification to the local package status requested by the user. This item is specific to the meta-installer. An example of a requested action is "install package ocaml in any version which is at least 3.10, and remove the package acroread", which would be expressed in the concrete syntax of the specific meta-installer.

Desiderata user preferences to discriminate among possible alternative solutions (e.g. "minimize download size", or "do not install experimental packages"). The exact list of possible user preferences depends on the distribution, and on the capabilities of the meta-installer (for instance, for Debian's `apt` these may be defined in the file `/etc/apt/preferences`).

This information item is optional.

Tool identifiers two pairs $\langle name, version \rangle$ uniquely identifying the installer and meta-installer which are in use, in the context of a given distribution. One pair identifies the *installer* used, the other the *meta-installer* used.

Distribution identifier a string uniquely identifying the distribution run by the user (e.g. `debian`, `mandriva`, `pixart`, ...), among all the implementations of DUDF.

Outcome either the new local package status as seen by the used meta-installers (in case of *success*) or an error message (in case of *failure*, i.e. the meta-installer was not able to

- dudf:
  - version: 2.0
  - timestamp: *timestamp*
  - uid: *unique problem identifier*
  - distribution: *distribution identifier*
  - installer:
    - name: *installer name*
    - version: *installer version*
  - meta-installer:
    - name: *meta-installer name*
    - version: *meta-installer version*
  - problem:
    - package-status:
      - installer: *installer package status*
      - meta-installer: *meta-installer package status*
    - package-universe:
      - package-list$_1$ (format: *format id.*; filename: *path*; url: *url*): *package list*
      - ...
      - package-list$_n$ (format: *format identifier*; filename: *path*): *package list*
    - action: *requested meta-installer action*
    - desiderata: *meta-installer desiderata*
  - outcome (result: *one of "success", "failure"*):
    - error: *error description* (only if result is "failure")
    - package-status: (only if result is "success")
      - installer: *new installer package status*
      - meta-installer: *new meta-installer package status*
  - comment: *additional, user-provided information* (optional)

Figure 1.2: The DUDF skeleton: information items and containers corresponding to problem/outcome submissions.

fulfill the user request). The error message format is specific of the used meta-installer, it can range from a free-text error message to a structured error description (e.g. to point out that the requested action cannot be satisfied since a given package is not available in the package universe).

Note that we are only interested in meta-installer failures, that is failures that occur before the meta-installer could call the package installer. Hence, when the meta-installer signals an error, the actual installation status of the packages on the system or their configuration has not changed yet.

The instantiation of these generic parts will be described in the sections on the RPM-specific DUDF format (Section 2.1) and the Debian-specific DUDF format (Section 3.1).

1.3 Overview of the CUDF format

The *Common Upgradeability Description Format* (CUDF) is a format for describing upgrade problems independently of a specific FOSS distribution. It was first defined in [TZ08], and later updated in [TZ09a].

A CUDF document describes a package universe (comprising both the available and the installed packages), and an upgrade request. It consists of three parts

1. a preamble which in particular serves to define extra information fields that can be used in the package stanzas;

2. a list of package stanzas, each of them consisting of a list of properties. Among these properties, the `name` (a string) and the `version` (an integer) together uniquely identify a package within an upgrade problem description. The `depends` fields describes the requirements of the package, it is a negation-free propositional formula of package names that may be qualified with a constraint on their version number, like `ocaml > 17`. The `conflicts` property specifies a list of packages, possibly qualified by a constraint on their version number, that must not be installed together with that package. The `provides` property gives a list of so-called *virtual packages* or *features* realized by that package. These may be used to satisfy dependencies of other packages, and must be taken into account in the conflicts of other packages. Packages listed in the `provides` possibly come in a specific version. The `installed` property specifies whether the package is currently installed, or merely available for installation. Finally, the `keep property` of a package is only relevant when the package is already installed, it specifies whether that package must remain installed in exactly that version, or in any version, or whether it may be replaced by another package providing it, or if there is no such requirement.

3. an upgrade request, which consists of an installation, a removal, and an upgrade (in the specific sense) request. Each of them is a list of package names, possibly qualified by a constraint on their version number, that must be installed, resp. removed, in any claimed solution to the request. An upgrade request behaves almost like an install request except that in a solution there must be only one version of the package installed, and that it must not be smaller than any of the originally installed versions.

The complete and precise syntax and semantics of CUDF is defined in [TZ09a]. Two points concerning the semantics deserve mention here since they will be important for the translation of the different DUDF formats to CUDF:

- The model allows the installation of several versions of packages concurrently with the same name. This feature was taken over from the semantics of the RPM model, but is in opposition to the Debian model which allows at most one version of a package to be installed at a time.

- The conflicts of a package do *not* apply to virtual packages provided by the same package. This was taken over from the Debian package model; in fact this feature allows us to translate the Debian-DUDF format to CUDF while preserving the Debian uniqueness requirement of installed versions of a package (see Section 3.4).

1.4 Structure of this document

In the rest of the document we will describe the work done both for FOSSdistributions that are based on the RPM-format (Chapter 2), and for distributions that are based on the Debian package model (Chapter 3). For each of these two families of distributions we will describe

1. how the generic DUDF format has been instantiated for that specific family of distributions;
2. the instrumentation of the distribution-specific package installation tools to generate a problem report on the user machine;
3. the server infrastructure that was created in order to collect problem reports that are submitted by users;
4. the translation from DUDF reports into the common CUDF format.

The RPM-family of distributions is represented in Mancoosi by two commercial GNU/Linux distributions: Caixa Mágica and Mandriva. We will indicate the cases in which there are differences between the specific solutions employed by these distribution editors. The Debian family is represented in form of the Debian FOSS distribution by some individual members of the Mancoosi project who are committed users or project members.

Chapter 2

RPM-based distributions

2.1 The DUDF format for RPM-based distributions

In this section we describe the DUDF encoding adopted by RPM-based distribution. To avoid duplication of code and effort, Mandriva and Caixa Mágica share a common format to encode the package status and universe. However, because of differences between `urpmi` and `apt-rpm`, DUDF files embed distribution-specific information such as meta-installer configuration files.

2.1.1 The RPM-DUDF format

In this section we describe the parts of the DUDF that are common to each RPM-based distribution and that are not specific for Caixa Mágica or Mandriva. Meta-installer specific details are described in Sections 2.1.3 and 2.1.2.

Package universe

Synthesis file format. The synthesis format is used to encode meta-data information [Man08]. A synthesis file is a compressed textual file used by librpm to build its internal database that generated from RPM binary packages. A synthesis file contains a list of stanzas (see Figure 2.1). Each stanza is composed by a sequence of non-empty lines; each line is a @-separated list of elements with the first element of the line being the type of the line. Lines are always in the same order with the $@info@$ chosen as stanza marker delimiting the entry. A stanza has five entries with the following types:

provides	:	list of functionalities provided by this package
requires	:	list of packages that must be installed for this package to be installed
obsoletes	:	used as hint to the solver to upgrade a package
conflict	:	list of packages that are in conflict with this package
summary	:	package description
info	:	information about this package

The four first types (`provides, requires, obsoletes, and conflict`), are a list of package names, possibly with a constraint on the version of the package which is of the form `package [flag version]` where flags are $<=, >=, <, >$ or $==$.

The entry of type `info` has the following format:

```
@info@name-version-release.arch@epoch@size@group@
```

where the `name-version-release.arch` and `epoch` identifies the package while `size` is the size of the binary package and `group` is the category to which the package belongs (i.e. `kde-games`).

```
@provides@openldap1[== 1.2.12-4mdk]
@requires@libldap1[==1.2.12-4mdk]@rpm-helper[*]@/bin/sh[*]@/bin/sh[*]@
  bash@libc.so.6@libc.so.6(GLIBC_2.0)@libc.so.6(GLIBC_2.1)@
  libc.so.6(GLIBC_2.3)@libcrypt.so.1@libcrypt.so.1(GLIBC_2.0)@
  libnsl.so.1@libpthread.so.0@libpthread.so.0(GLIBC_2.0)@
  libpthread.so.0(GLIBC_2.1)@libpthread.so.0(GLIBC_2.3.2)@
  libresolv.so.2@libtermcap.so.2
@summary@LDAP servers and sample clients.
@info@openldap1-1.2.12-4mdk.i586@0@2054148@System/Servers
```

Figure 2.1: Fragment of a synthesis file

Meta-data information embedded in a DUDF document are either encoded extensionally in the synthesis format or intentionally. In the former case, the entire synthesis file (uncompressed) is added as a `CDATA` section in the DUDF document. In the latter case, we add a new element embedding a remote reference to the synthesis file using the following *rnc schema*:

```
package_list =
   element package-list {
      attribute dudf:format { text }?,
      attribute dudf:filename { text }?,  # must be an # absolute path
      attribute dudf:url { xsd:anyURI }?,
      attribute dudf:include { xsd:anyURI }?,
      any*
   }
```

This XML relax-ng fragment is an extension of the normative relax-ng DUDF specification (see [TZ09c]) where we add the **include** attribute in order to specify a remote URL where to fetch the synthesis package.

```
<package-universe>
  <package-list
    format="synthesis_hdlist"
    include="http://dudf.caixamagica.pt/lists/5fb9db2a60b38707fbd1de33825e3790"/>
[...]
```

Figure 2.2: Intensional reference to a remote synthesis file

Both Mandriva and Caixa Mágica have deployed a website containing all published official synthesis files. Caixa Mágica stores all synthesis files from June 2009 on, they can be retrieved from the Caixa Mágica website using the URL schema `http://dudf.caixamagica.pt/lists/MD5SUM` where the `MD5SUM` variable is the md5sum digest of the synthesis file package. Mandriva synthesis files are listed from the following address: `http://doc4.mandriva.org:8087/synthesis/`, and individual synthesis files can be accessed using their MD5 signature from the following URL pattern: `http://doc4.mandriva.org:8087/synthesis/MD5SUM/download`. The synthesis files are uploaded to the Mandriva DUDF server by using an incremental process that is described in Section 2.2.

Package status

This section of a DUDF document contains a snapshot of the current state of the installer. The RPM database, stored in `/var/lib/rpm`, uses the Berkeley data base as its back-end. It consists of a single file containing all of the meta-information of the installed packages. The database is used to keep track of all files that are changed and created when a user installs a package, thus enabling the user to reverse the changes and remove the package later. Mandriva and Caixa Mágica adopted the same encoding of the package universe and the internal representation of the RPM database used by `apt-rpm` and `URPMI`.

```
<package-status>
  <installer>
    <status filename="/var/lib/rpm/Packages"><![CDATA[[
    [
    "x11-font-bh-75dpi",null,"1.0.0","7mdv2009.1",
    ["mkfontdir","mkfontscale","/bin/sh"],
    ["x11-font-bh-75dpi = 1.0.0-7mdv2009.1"],
    ["xorg-x11-75dpi-fonts <= 6.9.0"],
    [],
    1252461522,
    true
    ],
    [
    "lib64speech_tools1", null,"1.2.96","11mdv2009.1",
    [...]
```

Figure 2.3: Fragment of the RPM installer status encoding

The installer status is encoded in DUDF by serializing the following data structure using the `json` format [Cro06]. The json format has been chosen as a compromise between an xml-based encoding and a binary encoding. Each package in the database is represented by an array of stanzas in DUDF. Each stanza is an array of 10 elements as follows:

`name`	:	*string*
`epoch`	:	*integer*
`version`	:	*string*
`release`	:	*string*
`requires`	:	*array*
`provides`	:	*array*
`conflict`	:	*array*
`obsoletes`	:	*array*
`size`	:	*integer*
`isEssential`	:	*boolean*

where dependencies (requires,provides,conflicts,obsoletes) are arrays of triples as follows:

`name`	:	*string*
`constraint`	:	*string*
`version`	:	*string*

Empty elements are represented as `null` while empty dependencies are written `[]`.

2.1.2 The Mandriva DUDF format

Requested action

The meta-installer used by default on Mandriva systems is `urpmi`. The requested action in the Mandriva DUDF format is the copy of the command-line arguments used in the invocation of `urpmi`. Given a list packages to be installed or upgraded, `urpmi` will never downgrade a package. Any attempt to do so will result in an abort.

The most common requests to the `urpmi` meta-installer and to the `urpme` removal tool are summarized in Figure 2.1.2.

Examples of requested actions

`urpmi python` or `urpmi /usr/bin/python` on a Mandriva Linux 2010.1 system configured with the official Mandriva repositories will both install the latest version of the python package, i.e. python in version 2.6.5, release 2mdv2010.1.

`urpmi /usr/bin/python3.1`, `urpmi python3`, `urpmi python3-3`, `urpmi python3-3.1.2`, or `urpmi python3-3.1.2-1` will all launch the installation of the python3 package (named so for not conflicting with the python package, which actually refers to python-2x version series) in version 3.1.2, release 1mdv2010.1. However `urpmi /usr/bin/python3` will abort since the argument does not match partially a package name nor exactly an entry provided by a package.

`urpme python` or `urpme python-2.6.5` will both remove all the package python-2.6.5-2mdv2010.1 (if previously installed) and all the installed packages having a require dependency provided only by python-2.6.5-2mdv2010.1. But `urpme /usr/bin/python` will abort since "/usr/bin/python" is not part of a package name.

Desiderata

The user preferences related to the upgrade process can be set by the system administrator in the following files, or through `urpmi` command line options:

/etc/urpmi/skip.list (or `urpmi` option `--skip`) contains a list of packages that should not be automatically updated when upgrading the complete distribution (using the command line `--auto-select`). It contains one package expression per line - either a package name, or a regular expression (if enclosed in slashes) to match the name of packages against, using the full name of the package, which has the form name-version-release.arch.

/etc/urpmi/prefer.list (or `urpmi` option `--prefer`) contains a list of packages that should be preferred when a choice occurs for resolving dependencies. When several dependencies satisfy a given constraint required by the current installation, `urpmi` gives preference to the ones listed in the prefer.list file. The file contains one package expression per line; either a package name, or a regular expression (if enclosed in slashes) to match the name of packages against.

Outcome

The outcome consists of an error message stored in a CDATA field when the installation or uninstallation process failed.

request	semantics
urpmi NAME	looks up the package(s) to be installed or upgraded following the algorithm described in pseudo code in Appendix A. Urpmi first tests whether the input matches a package name, then whether it matches exactly the name of a provide dependency. In case of multiple candidates, urpmi uses the contents of the preference files /etc/urpmi/prefer.list and /etc/urpmi/prefer.vendor.list for discriminating between the results. In case several versions of a package selected for installation or upgrade are installed on the system and if a more recent one is available on the repositories, all previous versions are not installed and the new one is installed.
urpmi NAME-VERSION(-RELEASE)	looks up the package(s) to be installed or upgraded following the algorithm described in pseudo code in Appendix A.
urpme NAME	identifies the package to be removed by applying the same procedure as in the case of the command "urpmi NAME", except the lookup is made only against package names, not on the dependencies provided by packages, then removes the currently installed version of the package found, if any, and all the packages requiring a dependency that was provided only by this package. The look up algorithm is described in Appendix A.
urpme NAME-VERSION(-RELEASE)	identifies the package to be removed by applying the same procedure as in the case of the command "urpmi NAME-VERSION(-RELEASE)" except the lookup is made only against package names, not on the dependencies provided by packages, then removes the currently installed version of the package found, if any, and all the packages requiring a dependency that was provided only by this package. The look up algorithm is described in Appendix A.
urpmi –auto-select	upgrade all the packages that are currently installed and for which a newer version is available, except the packages matching an entry present in the file /etc/urpmi/skip.list, described below. The packages that are obsoleted by new ones are automatically removed.

Figure 2.4: Urpmi Requests

2.1.3 Caixa Mágica DUDF format

Requested action

Caixa Mágica uses `apt-rpm` as meta installer which is a fork of apt-get. Hence, the user request is expressed in the same format as for Debian (see Section 3.1).

Desiderata

Debian's `apt-preferences` file is not used on Caixa Mágica's systems. This information is hence not included in the DUDF file, and no `priority` field is computed for Caixa Mágica's related CUDF files.

Outcome

This section includes the free-form error message as reported by apt-rpm and a list of packages the meta-installer found to be non-installable while trying to satisfy the user request.

2.2 Instrumenting RPM meta-installers

2.2.1 apt-rpm (Caixa Mágica)

The DUDF generation for Caixa Mágica is done through a modification of apt-rpm which is the default installer for this distribution. To enable generation of DUDF documents the user has to set a configuration variable: if `APT::Dudf::Store-Report` is set to *true* in any of the apt-rpm configuration files or in a `-o` command-line flag then apt-rpm will save all the information about the user request, status of installed packages and package universe in Caixa Mágica's instantiation of the DUDF format (see Section 2.1.3). In addition, if the option `APT::Dudf::Store-Report-Success` is set to *true* then a trivial report will be generated informing about the successful installation. This kind of report is not specifically related to the goals of Mancoosi WP5 but rather is aiming to gather additional data for Caixa Mágica's quality assurance processes.

In a recent development related to Caixa Mágica version 15 due for release in July 2010 there is a new "user-friendly" process to enable DUDF reporting.

2.2.2 urpmi (Mandriva)

```
A problem occurred. You can contribute to the improvement of the Mandriva upgrade
process by uploading an automatically generated report to a Mandriva server.
No personal information will be transmitted. More information is available at
http://doc4.mandriva.org/bin/dudf/.
Do you want to generate and upload a report? (Y/n)
```

Figure 2.5: DUDF generation and upload permission request

The DUDF generator for Mandriva is developed as an extension of the meta-installer `urpmi`. This extension, named `urpmi-dudf` is provided as a package that can be optionally installed by users to report upgrade requests. Once the package is installed, `urpmi` will display the information and ask the question shown in Figure 2.2.2 when an error occurs. If the user accepts then the meta-installer will generate a DUDF that will be then uploaded to the Mandriva DUDF server, using the procedure described in Section 2.3. The user can also force the generation of DUDF files, upon any request to `urpmi`, by using the command-line option `--force-dudf`.

DUDF documents are generated using a C++ library developed by Caixa Mágica to read and encode the rpm database status and a Perl module which adds the Mandriva specific information.

2.3 Transmitting problem reports for RPM-based distributions

2.3.1 Caixa Mágica

The DUDF documents generated by Caixa Mágica users are uploaded periodically to the Caixa Mágica DUDF repository. Each individual report for failed installations is publicly available at the webpage http://dudf.caixamagica.pt/problems. There is no information, neither in the DUDF files nor the web interface, that discloses personal information from users' systems. The website gives aggregate monthly accounts for failed installations and affected packages.

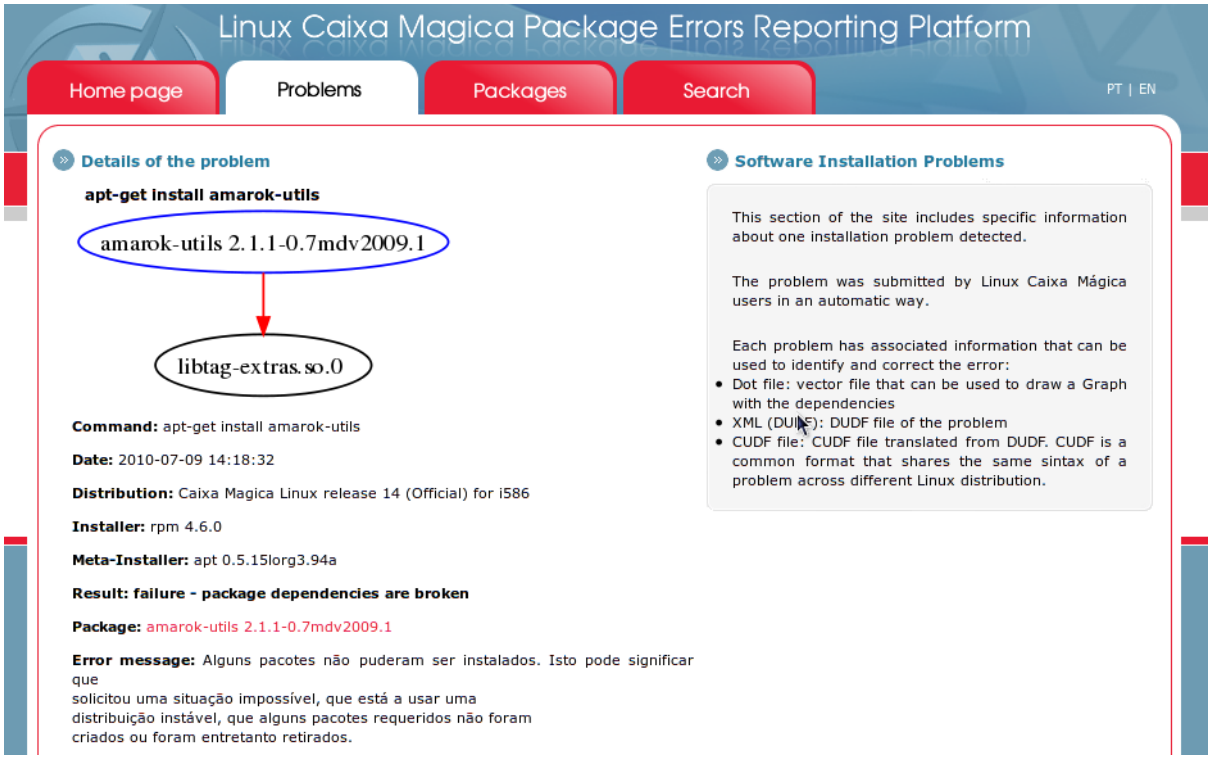


Figure 2.6: dudf.caixamagica.pt website: a DUDF instance

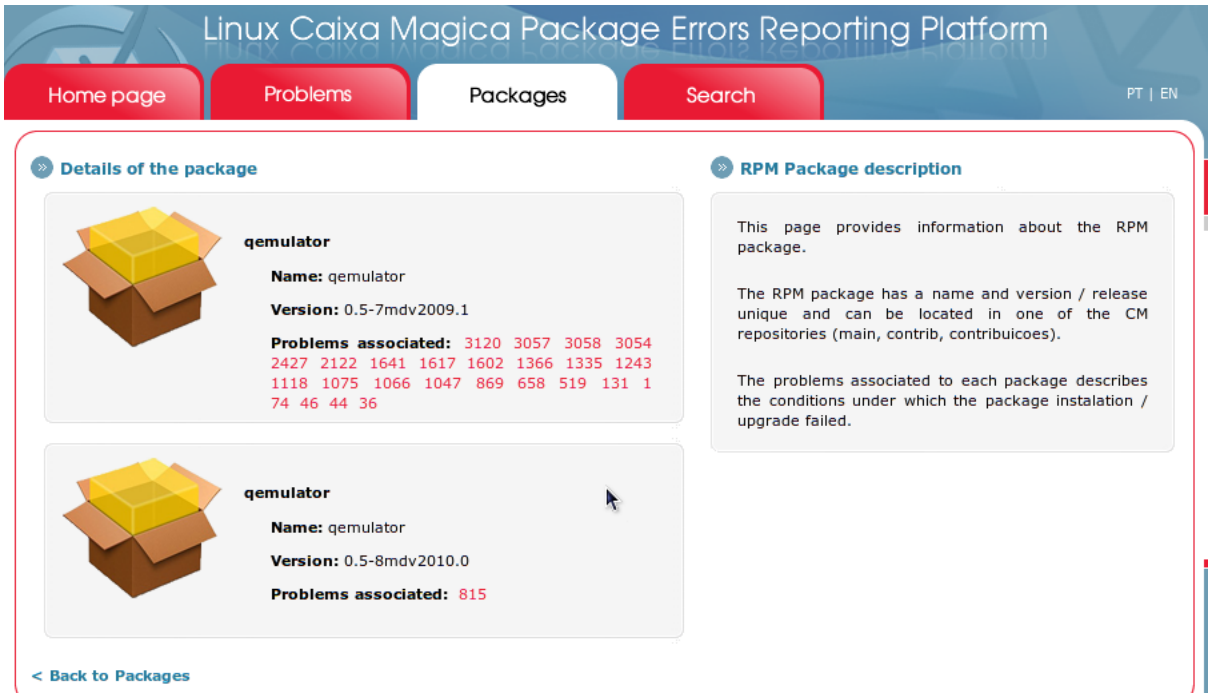


Figure 2.7: dudf.caixamagica.pt website: a report grouped by package

2.3.2 Mandriva

The DUDF reports generated by Mandriva users are uploaded synchronously to the Mandriva DUDF server at the following URL: http://doc4.mandriva.org/bin/dudf/. Figure 2.8 shows a list of uploaded DUDF reports.

The `urpmi-dudf` extension currently does not support asynchronous uploads of DUDF documents. The feature will however be added before the end of the project.

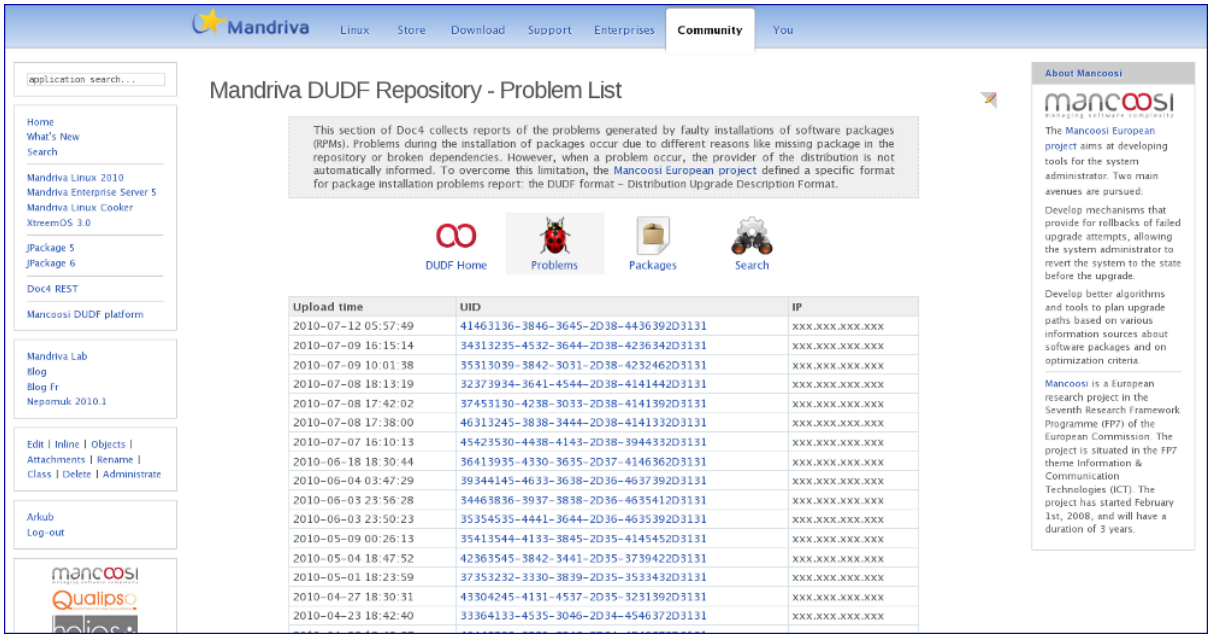


Figure 2.8: Mandriva DUDF Web site: list of DUDF documents uploaded by users

The package universe descriptions encoded in the synthesis format are uploaded incrementally to the Mandriva DUDF server. In order to reduce the DUDF size, just before the generation of a DUDF on a user machine, the `urpmi-dudf` extension checks whether the synthesis files used locally are already stored on the Mandriva DUDF server. In case they are not, they are uploaded to the Mandriva server, so that future reports related to the same synthesis data will simply refer intentionally to the file using its MD5 signature.

2.4 Translating RPM-DUDF to CUDF

2.4.1 RPM universe conversion

This section describes the translation from a RPM package archive to CUDF. The RPM semantics is not formally described, but it is supposed to be consistent with the latest implementation of the RPM utilities. In this document we refer to the implementation of librpm version 4.4.2.2.

RPM comparison function

RPM versions as defined above are compared using four functions: `vercmp`, `epochcmp`, `relcmp` and `rpmvercmp`. The function `vercmp` compares two RPM triples $(epoch, version, release)$.

First we compare the epoch using `epochcmp`. If these are equal then we compare the versions using `rpmvercmp`, and if these are equal again we compare the releases using `relcmp`.

vercmp

Require: $(e1, v1, r1)$
Require: $(e2, v2, r2)$
  $re \leftarrow epochcmp(e1, e2)$
  if $re = 0$ **then**
    $rv \leftarrow rpmvercmp(v1, v2)$
    if $rv = 0$ **then**
      return $relcmp(r1, r2)$
    else
      return rv
    end if
  else
    return re
  end if

`rpmvercmp` is the RPM comparison function as implemented in the RPM library. Since the RPM comparison function has no normative specification, we do not describe its algorithm here.

The `epochcmp` function compares the two epoch if they are present, and returns 1 or -1 if only one is present, 0 otherwise. The function `cmp` is the canonical comparison function between integers.

epochcmp

Require: $(e1, e2)$
  if $(e1, e2) = NULL, VAL$ **then**
    return -1
  else if $(e1, e2) = VAL, NULL$ **then**
    return 1
  else if $(e1, e2) = VAL, VAL$ **then**
    return $cmp(e1, r2)$
  else
    return 0
  end if

Similarly, the `relcmp` function compares the two releases if they are both present, and return 1 or -1 if only one is present, 0 otherwise.

relcmp

Require: $(r1, r2)$
  if $(r1, r2) = NULL, VAL$ **then**
    return -1
  else if $(r1, r2) = VAL, NULL$ **then**
    return 1
  else if $(r1, r2) = VAL, VAL$ **then**
    return $rpmvercmp(r1, r2)$
  else
    return 0
  end if

Version expansion

RPM versions are triples of the form `epoch, version, release`. The *epoch* term is an integer and it is used to allow to replace new RPM packages where RPM considers the new package version number to be lower than the installed package. The default epoch is zero and it is usually not specified. The *version* term is a sequence of alpha-numeric characters identifying the upstream version of the package. The *release* term is a sequence of alpha-numeric characters commonly identifying a distribution-specific release code. RPM versions identify *concrete* packages as a triple, while it is written as a string of the form `epoch:version-release` when used in dependency information items. In order to normalize RPM versions into a common format, we rewrite all RPM versions as strings.

Consider the following representation of a RPM package:

```
package: bash
version: 1.3
epoch: 1
release: ex2010
```

We associate to this package the canonical version string `1:1.3-ex2010`. If the epoch is not specified then it defaults to 0.

CUDF version mapping

CUDF versions are strictly positive integers [TZ09a]. RPM versions, which are strings, must hence be mapped from string to integer. However, the naïve approach of having a bijective mapping does not work since the RPM comparison function above does not provide a total order of RPM versions. The problem is highlighted by the example in Figure 2.4.1. Accordingly to the RPM implementation the package `p4` is in conflict with `p3` and `p1` but not with package `p2`. If we map RPM versions to integer versions, and then use the standard integer comparison function, it will be impossible to express this with a simple dis-equality with `p4`. This is actually a bug in the implementation of RPM acknowledged by its upstream authors and fixed in the new version of RPM [rpm10, Pro10a].

```
package: p0                          package: p2
provides: x (= 1.25)                 provides: x ( = 1.27 )

package: p00                         package: p3
provides: x (= 1.24)                 provides: x ( = 1.27-2 )

package: p1                          package: p4
provides: x ( = 1.26 )               conflicts: x ( < 1.27-3 )
```

Figure 2.9: Rpm Bug

One possible solution would be to translate RPM versions to integers and then handle explicitly problematic constraints. For example we could build the following map:

```
1.24   -> 1
1.25   -> 2
1.26   -> 3
1.27   -> 4
1.27-2 -> 5
```

Note that `1.27-3` does not show up in this table. The reason is that version *c*onstraints have to be translated differently from *c*oncrete versions, taking into account the correspondence table above, and the distinction between version and release part. In the above example, a version constraint `< 1.27-3` of `p4` would translate into `< 4` or `= 5`. This solution, despite being potentially correct, was discarded as error-prone and difficult to implement. For this reason the version mapping is performed during the translation phase.

The adopted solution is to change the nature of the problem, therefore loosing the information that package `p4` is in conflict with `x < 1.27-3`, adding an explicit conflict with packages `p1` and `p3`. This means that the constraint expansion is performed during the translation itself and that in the resulting document all conflicts and requires will be expressed using explicit version numbers. As a consequence, the generated CUDF documents are bigger since relevant version numbers are explicitly enumerated.

The function we use to expand RPM constraints is *RpmdsCompare* as implemented in librpm. It takes as input two dependencies constraints of the form `(name, flag version)` and returns true if the dependencies overlap, false otherwise. The fragment in Figure 2.4.1 will be then simply translated as in Figure 2.4.1.

Dependency mapping

Since all version constraints are fully expanded in the CUDF document, the dependencies translation is straightforward. There are two minor points to notice. The first point is that we remove dependencies on files provided by the same package, as well as dependencies on the package itself and dependencies on packages provided by the same package. This reduces the size of the resulting CUDF document and removes redundant information. The second point concerns relations to non-existing packages. In order to remain faithful to the CUDF mission, hence to retain all information contained in the source format, versions are numbered

```
package: p0                          package: p2
provides: x = 2                      provides: x = 4

package: p00                         package: p3
provides: x = 1                      provides: x = 5

package: p1                          package: p4
provides: x = 3                      conflicts: x = 3, x = 5
```

Figure 2.10: Cudf translation of Figure 2.4.1

starting from 2 while non-existing packages are numbered 1. This information is available at the conversion stage because of the constraint expansion in Section 2.4.1.

The mapping algorithm works in two stages. In the first stage, we build a table called *unit table* which maps package names to arrays of constraints. The indices of the array will be used as CUDF versions.

In the second stage, for each dependency expansion, a constraint of the form `(name,flag,version)` will be matched against the list of constraints in the units table. If the constraint overlaps the constraint in the units table then the associated index is added to the dependency list in the form `(name,"=",cudf version)`. Otherwise it is ignored.

Extra properties

Number. In order to maintain a mapping between RPM and CUDF versions, for each CUDF package we add an extra property named `number` of type string that contains the original RPM version of the package. Since RPM dependencies are mainly expressed by `provides`, this information allows only to maintain a partial mapping.

Essential. In RPM, similarly to other distributions, packages are categorized according to their importance or their functionality. Essential packages are those that must always be installed on the system. Essential packages can be upgraded, but cannot be removed. To preserve this property, for each essential packages we add a value `package` to the CUDF property `keep`.

2.4.2 Caixa Mágica request translation

The user request in apt-rpm follows Debian apt syntax and semantics hence the translation follows the rules described in Section 3.4.3

2.4.3 Mandriva request translation

The table below describes the action corresponding to the main urpmi upgrade commands.

request	action
urpmi `name`	The name of the package to be installed is found by using the procedure described in 2.1.2. The request is then translated to the "Install" property
urpmi `name-version(-release)`	The package to be installed is found by using the procedure described in 2.1.2. The request is then translated to the "Install" property with a strict (i.e., "=") version (and release) requirement
urpme `package`	It is trivially translated to the "Remove" property, with no version requirements
urpmi –auto-select	It is translated by listing all installed packages to the "Upgrade" property and additionally adding "Remove: package" for the packages that are obsoleted by newly installed ones.

Chapter 3

Debian-based distributions

3.1 The Debian-DUDF format

In this section we describe the instantiation of DUDF for the Debian distribution, that is the DEBIAN-DUDF format. We focus on the meta-installers `apt` and, to a lesser extent, `aptitude` which are the two meta-installers most commonly used in Debian.

In general, a DUDF document consists of a set of information items (see Section 1.2). Each item describes a part of the upgrade problem faced by the user. In this section we describe the information items that are not part of the generic DUDF scheme but that are specific to Debian.

Installer package status. This item consists of a snapshot of the current state of the installer. On a Debian system, this information is available in the file `/var/lib/dpkg/status` which is encoded in RFC 822 format with one stanza for each package known to the installer. For installed packages a complete stanza with all meta-information about this packages is given, while for not installed packages essentially only the name and the installation status are given. The pertaining information field in a Debian-DUDF document contains in the simplest case a verbatim copy of that file, thus leading to an *extensional* representation of the installer status.

A simple optimization consists in filtering out all package properties from the stanzas that are not used in the translation to CUDF, most notably the `Description` property which accounts for most of the file total size.

A more interesting possible optimization consists in listing for each installed package only their name and version, and relying for the translation to CUDF on the availability of an external archive that contains all the necessary properties of all packages (in all versions). This is an example of *intensional representation*, which is of course only possible when a package is *globally* uniquely identified by its name and its version. This assumption is satisfied when only packages coming from the official distribution are installed. The assumption would be not satisfied in particular when one has installed locally re-compiled packages without changing their version number since some of the meta-information of a package is generated during the compilation of a package, and might hence differ from the meta-information in packages coming from the official distribution (see Section 3.2).

Meta-installer package status. Specific meta-installers may maintain extra information as part of their local package status view. For example aptitude uses `/var/lib/aptitude/pkgstates` to store, among others, information about "leaf packages" that should be automatically removed when no other packages depend on them.

Package universe is the set of all packages known to the meta-installer. The universe for an upgrade problem is available on a machine as the lists of packages known to `apt` (or `aptitude`). They are stored in the directory `/var/lib/apt/lists/` and match the filename pattern `*_Packages`. These files are downloaded when the `apt update` action is invoked, and constitute a local cache of package listings coming from the Debian archive or its mirrors around the world.

For each repository listed in `/etc/apt/sources.list` there are several package lists stored on the local file system, one for each section (e.g: main, contrib, non-free) of that archive which is listed in `sources.list`. Only lists corresponding to repositories which were active at the time of the last update are kept on disk; lists for repositories which have been removed from `sources.list` are removed upon the next update run.

A verbatim copy of all these lists as an extensional section would provide a suitable package universe item in a DUDF document. The size of that document, which would be in the order of magnitude of the size of files downloaded when performing a single `apt-get update` action, could still be considered too large for users who run low on local storage space, or network bandwidth.

Moreover, the list of all *release* files are also embedded in the DUDF document. These files are stored in the directory `/var/lib/apt/lists/` and match the file name pattern `*_Releases`. The *release* files contain additional information associated to the package universe (`*_Packages` files) that are used to correctly resolve an APT request when used in conjunction with a target release identifier. Most notably *release* files also contain information about the architecture of the user machine.

For a given set of "well-known" repositories an alternative is to use an intensional representation, i.e. send as an intension just the checksum (for instance, SHA1) of the corresponding lists. As the lists are byte-to-byte identical copies of the remote repositories one may trust them to retrieve later in the translation to CUDF the complete lists from some archive site. The package lists are the same among different (synchronized) mirrors; this is guaranteed by the apt-secure machinery, as differences would invalidate the Release file signed with the GPG keys of the master Debian archive. To avoid having to search on the server the checksum among all stored checksums one may add the date of the last modification of the file lists on disk together with the architecture specification, to be used as a hint where the exact checksums has to be searched.

Requested meta-installer action. This is a textual encoding of the request issued by the user. In case of `apt-get`, for example it is the verbatim copy of the command-line arguments in the invocation of `apt-get`. Interactive meta-installers might need a different encoding. In this document we do not consider all Debian meta-installers, but we focus on `apt`. The most common requests to the `apt` meta-installer are summarized in the following table :

request	semantics
apt-get install PACKAGE	install the most recent version of a package
apt-get install PACKAGE=VERSION	install a specific version of a package.
apt-get install PACKAGE/RELEASE	install the version of a package coming from a specific release; it is applicable when more than one package repositories are listed in `/etc/apt/sources.list`.
apt-get remove PACKAGE	remove the currently installed version of a package.
apt-get upgrade	Packages currently installed with new versions available are retrieved and upgraded; under no circumstances are currently installed packages removed, or packages not already installed retrieved and installed. New versions of currently installed packages that cannot be upgraded without changing the install status of another package will be left at their current version.
apt-get dist-upgrade	dist-upgrade in addition to performing the function of upgrade, also intelligently handles changing dependencies with new versions of packages. dist-upgrade command may remove some packages in order to satisfy the upgrade request.

Additionally, `apt` also supports the option `-t RELEASE` which can be passed to all the requests above to override the preferences of `apt` (see Section 3.4.1) and to force the preference to a certain release.

Meta-installer desiderata Contains relevant user preferences that can be used by the installer to discriminate among different possible solutions. This item is specific to the meta-installer. In the case of `apt` the file `/etc/apt/preferences` lists the user preferences used by the `apt` pinning algorithm (see Section 3.4.1).

Outcome Contains the outcome of the meta-installer. If the meta installer was successful, then it contains the new local package status. If the meta-installer was unable to satisfy the user request, then it contains the error message either in free-text format or a structured error description. In the case of `apt` and `aptitude`, the error is in free-text format stored in a CDATA section.

backports.org	from January 2009
debian	from Mars 2005
debian-ports	from October 2008
debian-security	from Mars 2005
debian-volatile	from Mars 2005

Figure 3.1: Debian archive coverage on `snapshot.debian.org`

3.2 Instrumenting apt-get to produce traces

Dudf-save is a command-line wrapper for `apt-get` and `aptitude`. It collects all package metadata that concerns a user request, as well as the request itself and the output of the meta-installer, and creates from this a problem description in Debian-DUDF format (Section 3.1), which may then be optionally uploaded to a server (see Section 3.3).

A simple invocation of dudf-save (e.g. `dudf-save apt-get -s install firefox`) by default creates a compressed DUDF file in the local directory. As a fully extensional Debian-DUDF document might occupy several tens of megabytes, dudf-save implements an intensional optimization relying on the `snapshot.debian.org` service. Such a service hosts snapshots of the Debian archive since 2005 (see Table 3.2 for the covered archives) and offers a query interface that permits to check whether specific files have been stored as snapshots or not. Among the files that need to be stored in a Debian-DUDF document, `snapshot.debian.org` can store release files (`*Release` pattern) and packages files (`*Packages` pattern).

By default, each time dudf-save needs to store a package or release file, it queries `snapshot.debian.org`,[1] using the SHA1 checksum of the given file as a key. If the file is stored on `snapshot.debian.org` then only its SHA1 checksum is stored in the Debian-DUDF, wrapped into a `<include href="...">` element, otherwise the whole file is stored.

The above "interaction protocol" between dudf-save and `snapshot.debian.org` requires network connectivity during DUDF generation. While we do not consider this to be a relevant constraint—network connectivity is usually available at package installation time, in order to be able to retrieve packages from the network—an option `--self-contained` is offered to disable all intensional optimizations.

As a consequence of the above, the size of a Debian-DUDF document might vary, mainly in accordance with the usage (or not) of intensions to represents packages and release files. According to our observations it may vary between a few hundred kilobytes to five megabytes (of bzip2-compressed data).

The upload to the Mancoosi servers may be done immediately after the creation of the Debian-DUDF document (using the `--upload` command line option) or as a batch using a simple script.

The complete list of options supported by Debian-DUDF is reported in Figure 3.2, whereas an example session of using `dudf-save` is given in Figure 3.3.

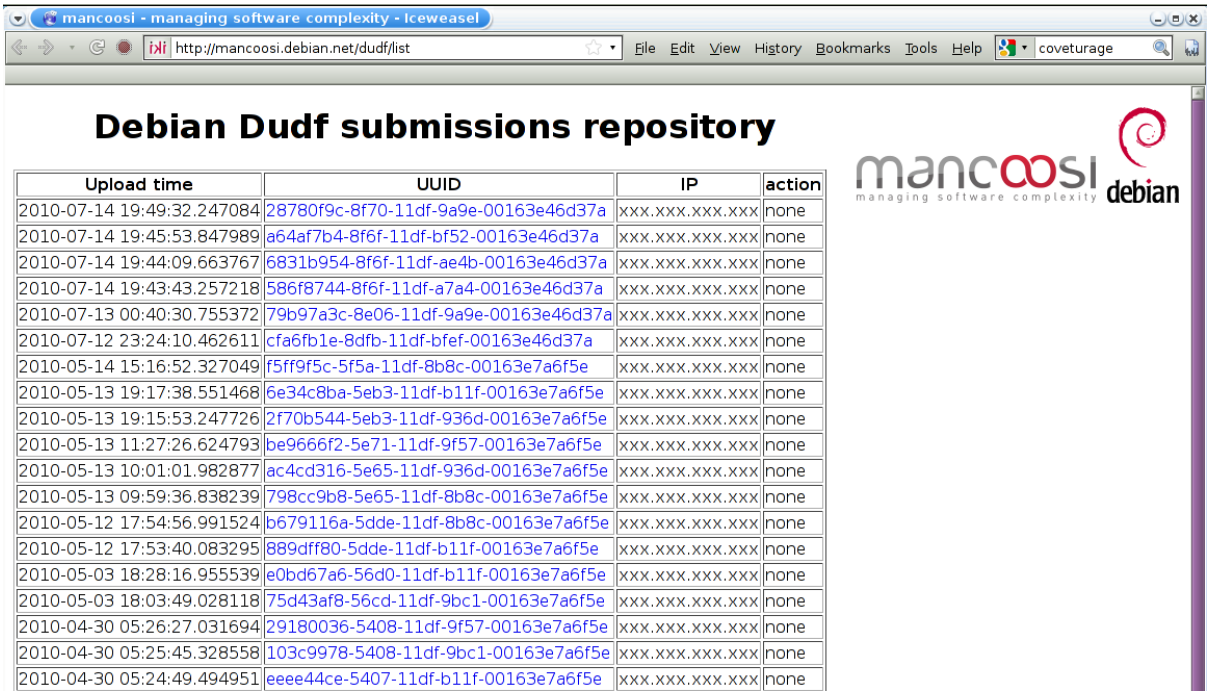
[1] using the JSON-based interface described at http://git.debian.org/?p=mirror/snapshot.debian.org.git;a=blob_plain;f=API;hb=HEAD

```
dudf-save [OPTION]... PKGMANAGER [ARG] ...

ARGs will be passed to PKGMANAGER unmodified

-b, --backend=TEXT
      force a specific (supported) package manager (default: guess
      package manager from the command line)

-c, --comment=TEXT
      store additional comments (default: no comment)

-C, --self-contained
      store information extensively; in particular, store APT lists
      verbatim rather than as  snapshot.debian.org  URLs. -C  will
      avoid several network lookups, but dramatically increase DUDF
      size. (default: use snapshot.debian.org)

-i, --uid=ID
      set a specific unique identifier (default: generate one)

-n, --nocompress
      do not compress DUDF (default: compress using bzip2)

-p, --upload
      upload DUDF to the Mancoosi contest server (default: no upload)

-s, --hostid=ID
      choose  a  specific  host  id; passing the empty string disable
      host id usage (default: read from configuration, if any, or
      ignore it)

-t, --tags=TAGS
      set a list of comma-separated tag to tag the DUDF (default: no tags)

-u, --url=URL
      set URL for DUDF upload (default: http://mancoosi.debian.net/dudf/upload)

-h, --help
      show this help text

--version
      provide version information about dudf-save
```

Figure 3.2: Dudf-save command line options

```
$dudf-save apt-get -s install firefox

INFO:root:dudf: adding /var/lib/dpkg/status
INFO:root:dudf: adding /etc/apt/preferences

INFO:root:dudf: found Packages.bz2 SHA1 for
   /var/lib/apt/lists/ftp.debian.org_debian_dists_unstable_main_binary-amd64_Packages:
   f96af220886939dfc0e03f0cd378d4d417209fb3
INFO:root:snapshot.d.o hit for f96af220886939dfc0e03f0cd378d4d417209fb3
INFO:root:dudf: storing APT list
   ftp.debian.org_debian_dists_unstable_main_binary-amd64_Packages as SHA1

INFO:root:dudf: found Packages.bz2 SHA1 for
   /var/lib/apt/lists/mancoosi.debian.net_debian_unstable_Packages:
   e2550072ebf06cfb25c2a69267666e3da0eaadc5
INFO:root:dudf: storing APT list
   mancoosi.debian.net_debian_unstable_Packages verbatim

INFO:root:snapshot.d.o hit for 8b70448d65fbb6082625b77eea79b352dec9379b
INFO:root:dudf: storing APT Release
   ftp.debian.org_debian_dists_unstable_Release as SHA1

INFO:root:dudf: storing APT Release mancoosi.debian.net_debian_unstable_Release verbatim

Reading package lists...
Building dependency tree...
Reading state information...
Package firefox is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source
E:  Package firefox has no installation candidate

INFO:root:Saving DUDF report to dudf-20100630-114702.xml.bz2 (can take a while ...)
INFO:root:DUDF report stored in: dudf-20100630-114702.xml.bz2
```

Figure 3.3: An example session with `dudf-save`. Empty lines and line breaks have been inserted for readability. The tool uses an intensional representation of package listings and release files when the checksums hits an entry in `snapshot.debian.org`, otherwise it falls back on an extensional (verbatim) representation.



Upload time	UUID	IP	action
2010-07-14 19:49:32.247084	28780f9c-8f70-11df-9a9e-00163e46d37a	xxx.xxx.xxx.xxx	none
2010-07-14 19:45:53.847989	a64af7b4-8f6f-11df-bf52-00163e46d37a	xxx.xxx.xxx.xxx	none
2010-07-14 19:44:09.663767	6831b954-8f6f-11df-ae4b-00163e46d37a	xxx.xxx.xxx.xxx	none
2010-07-14 19:43:43.257218	586f8744-8f6f-11df-a7a4-00163e46d37a	xxx.xxx.xxx.xxx	none
2010-07-13 00:40:30.755372	79b97a3c-8e06-11df-9a9e-00163e46d37a	xxx.xxx.xxx.xxx	none
2010-07-12 23:24:10.462611	cfa6fb1e-8dfb-11df-bfef-00163e46d37a	xxx.xxx.xxx.xxx	none
2010-05-14 15:16:52.327049	f5ff9f5c-5f5a-11df-8b8c-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-05-13 19:17:38.551468	6e34c8ba-5eb3-11df-b11f-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-05-13 19:15:53.247726	2f70b544-5eb3-11df-936d-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-05-13 11:27:26.624793	be9666f2-5e71-11df-9f57-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-05-13 10:01:01.982877	ac4cd316-5e65-11df-936d-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-05-13 09:59:36.838239	798cc9b8-5e65-11df-8b8c-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-05-12 17:54:56.991524	b679116a-5dde-11df-8b8c-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-05-12 17:53:40.083295	889dff80-5dde-11df-b11f-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-05-03 18:28:16.955539	e0bd67a6-56d0-11df-b11f-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-05-03 18:03:49.028118	75d43af8-56cd-11df-9bc1-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-04-30 05:26:27.031694	29180036-5408-11df-9f57-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-04-30 05:25:45.328558	103c9978-5408-11df-9bc1-00163e7a6f5e	xxx.xxx.xxx.xxx	none
2010-04-30 05:24:49.494951	eeee44ce-5407-11df-b11f-00163e7a6f5e	xxx.xxx.xxx.xxx	none

Figure 3.4: The Debian-dudf list page

3.3 Transmitting reports to UPD

Dudf documents generated by dudf-save can be optionally uploaded to the UPD servers. These documents, collected in anonymous form, will be then used to build a database of installation problems and train specialized solvers.

Dudf can be uploaded either upon creation using the `--upload` command line option, or in batch mode using a simple upload script (available in the dudf-save package). Dudf are transferred via http to the UDP servers parsed and stored in a database. The list, in Figure 3.4, of the submitted dudf is available at the url `http://mancoosi.debian.net/dudf/list`.

3.4 Debian DUDF to CUDF translation

In this section we explain the details of the Debian DUDF to CUDF translation. The purpose of the conversion is to preserve the semantics of the Debian archive format in the resulting CUDF document. First we explain how to translate the universe, coping with syntactic and — more importantly — semantic differences between CUDF and the Debian native formats that are embedded in Debian-DUDF. Then we discuss how we translate the meta-installer request into a CUDF request. The latter part is specific to the particular meta-installer used. Here we focus on the meta-installer `apt`.

3.4.1 The Debian `apt` pinning mechanism

In order to correctly translate Debian installation requests one must understand the so-called *pinning* mechanism of `apt`.

Several versions of the same package can be available at the same time when considering packages lists from multiple distributions (for example, stable and testing). In order to give a priority ordering to all available versions, `apt` assigns a numerical value to each package called *pin-priority*. Subject to dependency constraints, `apt` then selects the version with the highest priority for installation. The `apt` preferences file can be used to override the priorities that `apt` assigns to package versions by default, thus giving the user control over which one is selected for installation. The action of changing the default priority for a package is called *pinning*. The pin-priority is, a priori, not related to version numbers: when choosing among different available versions, `apt` will choose the version with the highest pin-priority, not the most recent package according to version.

The `apt` preferences file `/etc/apt/preferences` can be used to control which versions of packages will be selected for installation. `apt` assigns a priority to each file in the package universe according either to a default policy or to the directives contained in the preferences file. When several versions of the same package are available to be installed then `apt` chooses the version of the package with the highest priority.

The complete specification of the format of the `apt` preference file can be found in the man page `apt_preferences (5)`. Here we give two examples. The rule on the right of Figure 3.5 specifies that all packages belonging to the release unstable have a priority of 50. The `Pin` line specifying the keyword `release` has a number of optional arguments (i.e. `version`, `origin`, etc) to refine the selection. The rule on the left side of Figure 3.5 specifies that the package `perl` with versions matching the regular expression `5.8*` has priority 1001.

```
Package: perl
Pin: version 5.8*
Pin-Priority: 1001
```

```
Package: *
Pin: release unstable
Pin-Priority: 50
```

Figure 3.5: `apt` pinning stanzas.

Note. Special cases exists to avoid gratuitous downgrades. The `apt` meta-installer, by default, does not allow to downgrade a package. However this behavior can be overwritten by specifying a priority that exceeds 1000.

Pinning algorithm

The pinning algorithm considers two keys elements. The first, the *target release* is, if set, the single distribution to be considered during the installation process. It can be specified in the configuration file `/etc/apt/apt.conf` or directly from the command line request. By default the target release is not set. The second, the *pinning-priority* (or priority) is a positive integer associated to each package. The `apt` pinning algorithm computes the priority of each package as follows:

If the target release is not specified, then `apt` simply assigns priority 100 to all installed packages and priority 500 to all un-installed packages. Otherwise, the value of the priority for a package is :

100 : to the version that is already installed (if any).

500 : to the versions that are not installed and do not belong to the target release.

990 : to the versions that are not installed and belong to the target release.

Priorities settings can be defined either by *generically* specifying the priority of a release or a section which the package belongs to, or *specifically* by indicating the package name and a version.

3.4.2 Translating a Debian DUDF universe into a CUDF universe

The translation from a Debian universe into a DUDF universe needs to address not only obvious syntactic differences but also semantic differences. In particular, the conversion of the Debian universe involves the following three steps:

1. Version and package name normalization,
2. Adding self conflicts, and
3. Virtual package normalization.

Version and name normalization The CUDF specification [TZ09a] requires version numbers to be integers. In order to normalize Debian versions, it is necessary to collect for each package all versions mentioned in the document (including in conflicts and dependencies), and to sort these versions in ascending order according to the Debian version comparison function (see Section 5.6.12 of [Pro10b]). Then, the versions are mapped to integers $1, 2, \ldots$, so that the order is preserved.

Furthermore, CUDF syntax also requires a normalization on package names by escaping characters that are not permitted by the CUDF specification.

Self conflicts Debian semantics does not allow more than one version of the same package to be installed at the same time. In order to make this constraint explicit in the CUDF document, we add to each package a self conflict without a version.

This implies that we have to rename any virtual package that carries the same name as a concrete package. Without this renaming, installation of a package providing some virtual package

p together with a concrete package p would be artificially excluded by package p conflicting with p.

For example, consider the following Debian packages:

```
Package: foo
Version: 1

Package: foo
Version: 2

Package: bar
Version: 42
Provides: foo
```

In Debian semantics, package `foo` (in either version 1 or 2) can be installed together with package `bar`. Exactly the same file, but interpreted in CUDF semantics, would allow to install versions 1 and 2 of `foo` at the same time, which is not possible in Debian. We hence have both versions of `foo` conflict with `foo` in order to retrieve the mutual exclusion between both versions of `foo`:

```
Package: foo
Version: 1
Conflicts: foo

Package: foo
Version: 2
Conflicst: foo

Package: bar
Version: 42
Provides: foo
```

This renaming has the unwanted side-effect that now `foo` and `bar` are in conflict. We hence rename the virtual package provided by `bar`, and obtain finally:

```
Package: foo
Version: 1
Conflicts: foo

Package: foo
Version: 2
Conflicst: foo

Package: bar
Version: 42
Provides: foo--virtual
```

In the next paragraph we will see a second case where virtual packages must be renamed.

Virtual package normalization In Debian, a package can be both a virtual package and a concrete package. *Dependencies* and *Conflicts* on virtual packages possibly carry a version constraint (Section 7.5 of [Pro10b]). If they do not carry a version constraint then both virtual packages and concrete packages may be used to satisfy a dependency, and both are relevant for conflicts. If a relation carries a version constraint then only concrete packages are relevant. On the other hand, *Provides* in Debian do not carry a version, and can never satisfy a dependency on a package that carries a version constraint. In CUDF, however, provides without a version constraint are quantified universally over all available versions.

For example, consider the following Debian packages:

```
Package: foo
Version: 1
Depends: bar >= 2

Package: bar
Version: 2

Package: extra
Version: 2
Provides: bar
```

The package `foo` has a constrained dependency on `bar` which is in turn provided by the package `extra`. In Debian, this dependency can only be satisfied by the concrete package `bar` and not by the concrete package `extra` since the virtual package `bar` provided by `extra` cannot satisfy a dependency with a version constraint. If we were to propose a naïve translation from the fragment above to CUDF then there would be a mismatch between the Debian and the CUDF semantics. In particular, in the latter case, the package `extra` would allow to satisfy `foo`'s dependencies, while in Debian this is not the case.

In order to reconcile the two different semantics of Debian and CUDF we perform the following translation:

- All package names p in some `provides` property for which there exists a concrete package with the same name or a version constraint associated to it are replaced by a new package name p`--virtual`.
- All relations without a version constraint to a package that exists in the DUDF document as virtual package are expanded to an alternative consisting of all matching concrete or (renamed) virtual packages.
- All relations without a version constraint that match only concrete packages in the DUDF document are left untouched, since for concrete packages the Debian semantics and CUDF semantics coincide.
- All relations carrying a version constraint are left untouched as they are not going to match a virtual package anyway (since these have been renamed)

For instance, consider the following Debian package set - ignoring fields that are inessential here - where the package `bar` is both a concrete package and a virtual package (provided by the package `foo`).

```
Package: foo
Provides: bar

Package: bar
Depends: foo

Package: baz
Depends: bar
```

The corresponding CUDF document is as follows, where the virtual package `bar` is replaced by a new package name `bar--virtual` and the dependency of the package `baz` is replaced by a disjunction as it is not constrained.

```
Package: foo
Provides: bar--virtual        # it's a concrete-virtual package
Conflicts: foo

Package: bar
Depends: foo
Conflicts: bar                #  self conflict !

Package: baz
Depends: bar | bar--virtual
Conflicts: baz
```

Note that in the example, using the Debian semantics, package `foo` and package `bar` can be installed together. Without the renaming packages `foo` and `bar` could not be installed together in CUDF because of the self conflict.

A concrete example of the conversion of a Debian package description to CUDF format is as follows :

```
Package: 6tunnel
Priority: optional
Section: net
Installed-Size: 68
Maintainer: Thomas Seyrat <tomasera@debian.org>
Architecture: i386
Version: 0.11rc2-2
Depends: libc6 (>= 2.3.6-6)
Filename: pool/main/6/6tunnel/6tunnel_0.11rc2-2_i386.deb
Size: 12810
MD5sum: 5471e156d43755878763ec51a86ac1aa
SHA1: 8af63219150ad7079e5fb412c37b0b8e78904159
SHA256: 192db6cede7fc2794bccc6662b29f6935e84a59bb5cbf64b15989d114bc15c8a
Description: TCP proxy for non-IPv6 applications
```

The CUDF conversion results in the following stanza where additional fields not relevant to CUDF have been omitted:

```
Package: x6tunnel
Version: 0
Depends: libc6 >= 1
Conflicts: x6tunnel
```

3.4.3 Translating a Debian DUDF request to a CUDF request

The main semantic difference between the `apt` installation algorithm and the CUDF semantic is related to the package selection mechanism specified in `apt` by the pin priority. The translation of the `apt` pinning mechanism to CUDF can greatly restrict the space of possible solutions by imposing a behavior that exactly mimics `apt`'s.

Mapping `apt` requests

The user request of the meta-installer `apt` is an invocation of the command `apt-get`. The command line is embedded as-is in the DUDF document. We support the following options considering the package priority computed as in Section 3.2 and expressed as an optimization criterion. The option `-t release` and the `/` and `=` operators are just short-cuts to modify the pin-priority of a specific package.

request	**action**
apt-get install `package`	is trivially translated to the "Install" request, with no version requirements
apt-get install `package=version`	is trivially translated to the "Install" request with a strict (i.e., "=") version requirement
apt-get install `package/release`	is translated to "Install" requests for a specific version, just after having looked up the needed version according to the requested release
apt-get remove `package`	is trivially translated to the "Remove" request, with no version requirements
apt-get upgrade	is translated as for "upgrade" and additionally adding "Keep: package" to all installed packages
apt-get dist-upgrade	is translated by listing all installed packages to the "Upgrade" request

`apt-get` pinning information is added as an additional property to each CUDF package (see Section 3.2). For each CUDF package we add an extra field "Pin-Priority" of type integer. The priority of each package is computed from the APT preferences file and command line options.

3.5 Concluding remarks on Debian

3.5.1 Measures

In this section we provide some measures regarding the size of Debian archives. The numbers were recorded on 18/04/2008, on a i386 Debian Sid machine. The numbers per se are basically meaningless since they are affected by many factors, they are just there to provide a rough idea of the order of magnitude of a potential submission. Also, they have been measured with `du`, as such the sizes are disk blocks which overestimate the real byte count.

`/etc/apt/sources.list` contains all sections of both Debian Sid and Debian Testing, a well-known repository for multimedia applications which for legal reasons cannot be distributed by Debian, as well as the usual security repositories.

section	**plain**	**compressed** (gzip)	**notes**
package status `/var/lib/dpkg/status`	2.5 Mb	748 Kb	160 Kb / 24 Kb with trimmed stanzas
package universe `/var/lib/apt/lists/*_Packages`	48 Mb	14 Mb	reduces to 14 Kb for just sending the list md5sums together with `sources.list`
requested action			
resulting package status			

3.5.2 Towards a better representation of `apt` pinning

At the moment, the use of pinning information is left to be encoded and used by a meta-installer. A more generic solution would be to represent pinning information as a hard constraint in the `MooML` language [TZ09b] which has been proposed as a supplement to CUDF.

However, by implementing precisely the same pinning algorithm of apt-get in MooML, all solvers will behave exactly as apt-get instead of doing "better" than apt-get, even if a better solution to the request exists. In particular, since the incompleteness of `apt` comes exactly from the fact that it uses pin-priority strictly (i.e., it does not allow to install a package whose pin-priority is not the highest, even if that would allow to fix a broken dependency), all solvers will be as incomplete (w.r.t. the optimization criteria) as apt-get.

The strategy we have chosen for future work is therefore to map `apt` pinning information to an optimization criterion to be maximized in order to mimic `apt`'s behavior, but without ruling out the existence of other possible solutions. For each CUDF package we add an extra field "Pin-Priority" of type integer. The priority of each package is computed from the `apt` preferences file. Then the CUDF solver will try to maximize one of the following criteria expressed in MooML [TZ09b]:

- maximize the number of installed packages whose "Pin-Priority" is highest
- minimize the (sum of) the differences between the "Pin-Priority" of installed packages and their potential highest "Pin-Priority"

Part A

Urpmi lookup algorithm pseudo-code

```
#looks up the packages to be installed or upgraded when
#the given input is submitted to urpmi
sub lookup_packages_for_installation(input) {

  if (input matches exactly a package name) {
     if ((option(--fuzzy) and (input matches partially other package names)) {
       if ((option(-a)) {
          return (all matching packages)
       } else if (one matching package is present in /etc/urpmi/prefer(.vendor).list) {
          return (preferred matching package)
       } else {
          return (error with match list)
       }
     }
     return (matching package)
  }

  if (input matches exactly one provide name)  {

    if (several packages provide the matched provide) {
      if (one matching package is present in /etc/urpmi/prefer(.vendor).list) {
         return (preferred matching package)
      } else {
         print (choice prompt listing all the matching packages)
      }

    } else
      return (package providing the matching provide)
  }

  if (input matches partially a package name) {
     if (multiple results) {
        if ((option(-a)) {
```

```
            return (matching packages)
         } else if (one matching package is present in /etc/urpmi/prefer(.vendor).list) {
            return (preferred matching package)
         } else {
            return (error with match list)
         }
      }
      return (matching package)
   }
   return (error not-found-message)
}

#looks up the packages to be uninstalled when the given
#input is submitted to urpme
sub lookup_packages_for_uninstallation(input) {
   if (input matches exactly a package name) {
      return (matching package)
   }
   if (input matches partially a package name) {
      if (option(-a)) {
         return (matching packages)
      } else {
         return (error with match list)
      }
   }
   return (error not-found-message)
}
```

Bibliography

[511] Mancoosi WorkPackage 5. UPDB infrastructure to collect traces of ugradeability problems in CUDF format. Deliverable 5.3, The Mancoosi Project, February 2011. To appear.

[Cro06] Douglas Crockford. The application/json media type for javascript object notation, 2006. http://www.ietf.org/rfc/rfc4627.txt.

[Man08] Mandriva. Synthesis format, 2008. http://wiki.mandriva.com/en/Format_of_synthesis.hdlist.cz_index.

[Pro10a] Mancoosi Project. Knowledge and issues on rpm metadata, 2010. http://wiki.mancoosi.org/bin/view/Project/RPMMetadata.

[Pro10b] The Debian Project. Debian policy manual, 2010. http://www.debian.org/doc/debian-policy/.

[rpm10] rpm version comparaison pb, 2010. https://qa.mandriva.com/show_bug.cgi?id=55810.

[TZ08] Ralf Treinen and Stefano Zacchiroli. Description of the CUDF format. Deliverable 5.1, The Mancoosi Project, November 2008. http://www.mancoosi.org/reports/d5.1.pdf.

[TZ09a] Ralf Treinen and Stefano Zacchiroli. Common upgradeability description format (CUDF) 2.0. Technical Report 3, The Mancoosi Project, November 2009. http://www.mancoosi.org/reports/tr3.pdf.

[TZ09b] Ralf Treinen and Stefano Zacchiroli. Expressing advanced user preferences in component installation. In Roberto Di Cosmo and Paola Inverardi, editors, *IWOCE 2009 - International Workshop on Open Component Ecosystem, affiliated with ESEC/FSE 2009*, Amsterdam, The Netherlands, August 2009.

[TZ09c] Ralf Treinen and Stefano Zacchiroli. Upgrade description formats: generalities and DUDF submission format. Technical Report 3, The Mancoosi Project, November 2009. http://www.mancoosi.org/reports/tr1.pdf.