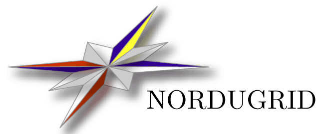

KnowARC D2.5-1
29/5/2007

The Handbook to Dynamic Runtime Environments

Technical Description and Installation Guide

Daniel Bayer[*,1], Steffen Möller[*,1], Frederik Orellana[2]

[1]University of Lübeck, Institute for Neuro- and Bioinformatics, Ratzeburger Allee 160, 23538 Lübeck. Germany [2]University of Copenhagen, Nils Bohr Institute, Blegdamsvej 17, 2100 Copenhagen, Denmark

Abstract

These pages introduce to the dynamic Runtime Environments (DREs) for Grid Computing with the Advanced Research Connector (ARC) middleware (Ellert et al., 2007). This effort represents both a formalism for the description of DREs and a mechanism for their installation.

This work was performed within the KnowARC EU-project.

Availability: `http://dre.knowarc.eu:8080`

1 Introduction

A major motivation for grid projects is to stimulate new communities to adopt computational grids for their causes. From the current grid user's viewpoint, the admission of users of a very different education will suddenly impose difficulties in the communication between site maintainers. One will not even understand the respective other side's research aims. Hence, the proper installation of non-standard software (Runtime Environments) is not guaranteed and the priorities of manual labour will be mostly disjunctive.

A core problem remains to distribute a locally working solution, the Know-How, quickly across all contributing sites, i. e., without manual interference. Every scientific discipline has its respective own set of technologies for the distribution of work load. E. g., research in bioinformatics requires access to so many different tools and databases, that few sites, if any, install them all. Instead, the use of web services became a commodity, with all the problems with respect to bottlenecks and restrictions of repeated access.

The EU project KnowARC[1] amongst other challenges with the here presented work extends the NorduGrid's Advanced Research Connector (ARC) grid middleware (Ellert et al., 2007) towards an infrastructure for the automated installation of software packages.

An automation of the software installation, referred to as dynamic Runtime Environments, seems the only approach to use the computational grid to its full potential. Components of workflows shall be spawned as jobs in a computational grid using dynamic Runtime Environments rather than as shared web services. The grid introduces an extra level of parallelism that web services cannot provide. The required short response times and the heterogeneous education of site-administrators on a grid demand an automatism for the installation of software and databases without manual interference.

This document presents basic concepts and a description of the dynamic RE framework as it was implemented with respective instructions to set it up locally. Outlooks to future developments are presented, for a detailed view on the full functionality of the envisioned system it is suggested to consult the Design Document[2].

2 Overview

To install software and keep it up to date on grid resources is cumbersome, as a large part of this work has to be performed manually. This puts a heavy load on the local site admins and contributes to a slow acceptance of grid in areas which depend on complex software applications.

The proposed dynamic treatmeant of Runtime Environments (RE) aims at improving this situation in a way such that application software environments, or REs are installed both automatically and dynamically:

- the installation process will run without human supervision,
- an installation might be started as late as just prior to a job's execution,
- legacy REs are removed when not used for some time.

For simplicity, software aimed at being used on remote sites is agglomerated to *packages*. Such packages that are prepared to be installed withouth manual intervention are referred to as dynamic Runtime Environments. The functionality to dynamically install such packages at remote sites of a computational grid seeds the demand for the following compents, as described in the ARC Design Document (The KnowARC Consortium, 2007).

Janitor Software for the installation and management of software packages at a particular site

Catalog Web site or service announcing the availability of software packages. The description of the Runtime Environments is presented in the resource description framework (RDF) format.

Repository A storage component accomodating the actual packages.

The dynamic software installation process relying on these components consists of the following steps:

[1] `http://www.knowarc.eu`
[2] `http://www.knowarc.eu/documents/Knowarc_D1.1-1_07.pdf`

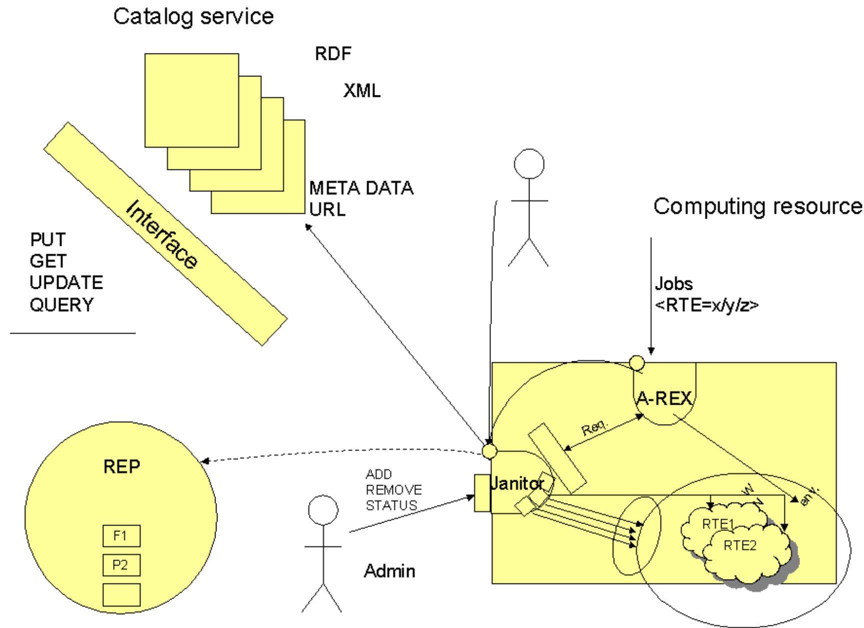

Figure 1: **Sketch of a full-scale dynamic Runtime Environment management framework.** *Depicted at the top left is a Catalog that presents descriptions of Runtime Environments (RE) to all participants in the grid. The bottom right describes the components that are involved in the deployment of a RE at a site. The Janitor interacts with the A-REX (the Grid Manager in ARC 0.6.x) to learn about incoming jobs and their requirements. Future development will bring a separate interface for distinguished users to control the installation of Runtime Environments.*

1. A compute element (CE) accepts a new job that requires REs.
2. The Janitor determines a list of packages that are needed to provide the requested REs by contacting the Catalog.
3. This list is crosschecked against already installed packages (automated or manual installs make no difference) and those packages that are installable.
4. The Janitor retrieves missing packages from the Repository.
5. The Janitor follows the packages' instructions for their installation.
6. Once the RE is dynamically provisioned, the CE executes the job in the (freshly) installed application environment.
7. Upon job completion, the Janitor decides if the RE shoulds be cached for later jobs or be deinstalled.

Figure 1 shows an overview sketch of a possible full scale implementation of the dynamic Runtime Environment management framework as it is laid down in the ARC design. The current implementation described in this document implements just a subset of the full framework. In particular, the Janitor and Catalog components do not exhibit properly defined interfaces suitable for managing Catalog entries and Runtime Environments. Furthermore, the integration of the Janitor with the other components of the computing element is naturarly not the final, especially due to the fact the A-REX service is not available yet. Therefore, the current implementation will be extended towards a framework for the management of Runtime Environments that addresses a prime concern of users in production environments: the verification of a correct implementation whereindividual automated installations can be tested externally (and manually) for their correct behaviour.

The prototypical implementation that is described in this document was performed with the scripting language Perl[3], which is central within ARC for the interaction with the backend queueing systems. The integration with the current ARC is performed both in the languages Perl and C++.

3 Components

The components described in this section are separate modules that in the current implementation communicate with each other via the file system or via the shell interface of executable files. A schema for web services to further abstract those interactions and to allow for remote access has been designed to foster the management of dynamic Runtime Environments. The schema is described in the web services description language (WSDL). The implementation of the WS-based management interface of main components is suspended until the completion of the development of the next generation ARC's Hosting Environment Daemon (HED) modul.

3.1 Janitor

3.1.1 Functionality

The Janitor component, a local service deployed on the computing element is responsible for the following:

1. Interacts with the local job execution service (currently grid manager, later A-REX) and triggers dynamic installation of Requested REs.
2. Interacts with the Catalog component to determine available REs and their dependencies on packages.
3. Recursively retrieves packages from Repositories that are required by REs.
4. Automatically installs fetched packages, sets up the local RE.
5. Implements a software cache by keeping dynamically installed REs on the computing element for a configurable time span.
6. Periodically purges prior installations.
7. Enforces local policies by ensuring that only "allowed" REs are installed.
8. Maintains internal states of dynamic Runtime Environments deployed on the computing element.
9. Provides information on available, installed REs for the local Information System component.

Constraints on Runtime Environments for deployment A major concern for the development of the DRE framework was the avoidance of manual interference whenever possible. Thus, site-administrators are given the option to constrain the eligibility of packages for the installation. This can be expressed by statements on properties of the packages, i.e. their size on the disc, the runtime memory footprint or their expected time for installation. Allowing for the use of ontologies for the description of Runtime Environments, also semantic statements are possible, e.g. to allow all libraries for statistics.

3.1.2 Implementation details

The Janitor is the main active component. It installs Runtime Environments to a grid system or removes them. The initiation of the installation may be automatic or manual. The current implementation only fosters automated installations. The management interface enabling manual control and the verification will be added upon the arrival of the HED component of next generation ARC. The current implementation of the Janitor exposes it functionality via a shell-based interface.

[3]http://www.perl.org

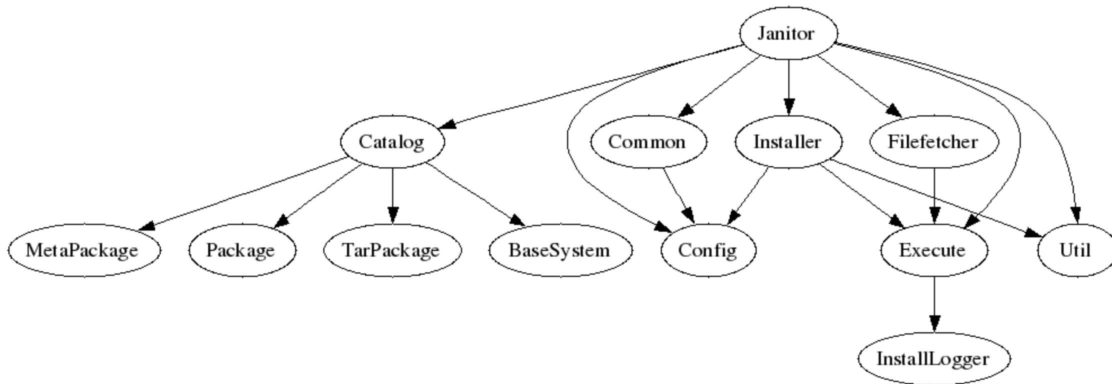

Figure 2: **Modules of the Janitor and their call-dependencies**

The shell-based Janitor interface The interface is kept as simple as possible. Essentially, there are three different commands:

- "`Janitor new <jobid> <rte1> ... <rten>`": This registers job *jobid* with the Janitor. The Janitor checks if it is possible to deploy the requested RE and returns either true or false. It also performs the installation if possible.
- "`Janitor info <jobid>`": Lists i.e. the location of the runtime scripts which must be sourced by the job *jobid*.
- "`Janitor remove <jobid>`": Deletes the job *jobid* from the Janitor's database.

For some features the Janitor must be run with system priviledges. For this, a very minimalistic (and possibly still unsafe) suid-root Wrapper is provided in rjanitor.c. This is required since the sticky bits under UNIX do not work for shell scripts.

Perl Modules The main language for the implementation of the functionality of the DRE functionality is Perl. And it is solely required (exceptions are the integration with the Grid Manager and the Web server) for the Janitor. In the pre-web-service implementation the Catalog remains a static web page. The Perl code is split into multiple modules as depicted in figure 2. The modules can be separated into two functional groups. One addresses the retrieval of information from the Catalog RDF file in the left major branch of the figure. The other addresses the process of fetching and installing the packages.

Preparation of WS interface for the Janitor The current prototype does not provide web service interface to the Janitor. The respective infrastructure for ARC to be shared by all its grid components is still under development. In anticipation of the upcoming web services infrastructure of the next generation ARC, a proposal for WS-based interface description has already been provided. The WSDL files, and their conversion to the HTML format, are distributed with the source code in the interfaces folder and are available online[4]

The WS-based management interface of the Janitor will provide two core features:

Sharing of custom libraries: Additions to the larger software packages can more easily shared between jobs submitted.

Verification: Installed Runtime Environments can be annotated for their reliabilty.

These features are considered as the technical breakthrough for an adoption of the dynamic Runtime Environments in production environments. Physicists will distribute their libraries, bioinformaticians will be interested in the distribution of updates to databases or the general addition of software available on the grid.

[4]http://dre.knowarc.eu:8080/wsdl

State	Description
unavailable	The RTE is not available for the basesystem the site uses.
installable	The RTE is available for the basesystem the site uses and it will be automatically installed if a job needs it.
installing/a	A job requested the RTE and it is currently installed
installing/m	The RTE-adminstrator requested the installation of the RTE. Its currently being installed.
failed	The installation process failed.
installed/a	The RTE is installed dynamically.
installed/m	The RTE ist installed manually by the RTE-administrator
broken/m	The RTE is installed but failed validation by the RTE-administrator
validated/m	The RTE is installed and successfully passed validation by the RTE-administrator
removal pending	The RTE is still installed but will be removed as soon as possible. It is not available to new jobs.
removing	The RTE is currently being removed.
installed/s	The RTE was installed in the old fashioned way by the site admin
broken/s	The RTE was installed in the old fashionend way and failed validation by the RTE-administrator
validated/s	The RTE was installed in the old fashioned way and was successfully verified

Table 1: **States a Runtime Environment can possibly be in.**

States for RTEs A main motivation for the managed, manual induction of dynamic RTE installation is the manual verification of the installed packages prior to their use in production. With an automation of the installation, the verification of that process shall be performed externally to that process. At this time, only the automation of the installation has been implemented. To reflect the progress the external verfication has made, runtime environments are said to be in states. The current implementation lists installable RTE aside the installed RTE in the grid information system, in order to stimulate grid clients to submit packages. The here described states will be represented to the clients in upcoming developments.

These states are specific for every computing element and communicated between the Janitor and the Execution Service. Figure 3 displays the transitions between the states that a Runtime Environment may be in during its life time at a particular omputing element. The manually induced transitions are marked in red, he automated transitions in black. A transition between states can be induced automatically (i. e. by the advent of a job requesting a particular dynamic RTE) or manually by the site's supervisor or an individual with respective rights to use the Janitor's web service.

Upon presentation of a package to a Catalog, a computing element may classify a package to be *installable* if all the dependencies are installable or already *installed*. The installation can be performed manually (*installing*/**m**) or in an automated fashion (. . . /**a**). Should the installation process return an error, then the installation has *failed*. Once the installation succeeded, the installed package is validated for its correctness. Should that process fail, then the package's state it is said to be *broken*.

Automatically installed packages can be removed by the automatism. A manually installed package or one that has failed to be installed, can only be removed upon manual induction. The . . . /**s** states represent those Runtime Environments that are installed in the original manual way of RTE installation in ARC 0.6.

3.1.3 Integration with the current ARC middleware

Figure 4 presents an overview of how the current implementation of the Janitor integrate with the other components available on the computing element.

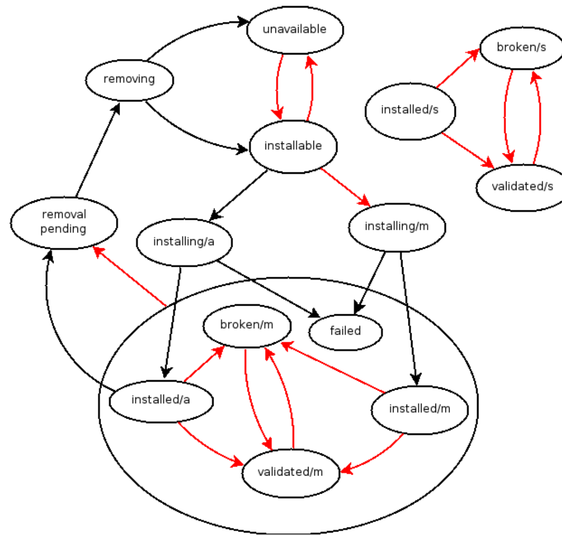

Figure 3: **Relationships between the possible states of Runtime Environments.** *Red arcs represent human interaction. The distinction between* **/a**, **/m** *and* **/s** *states does not need to be visible for all clients.*

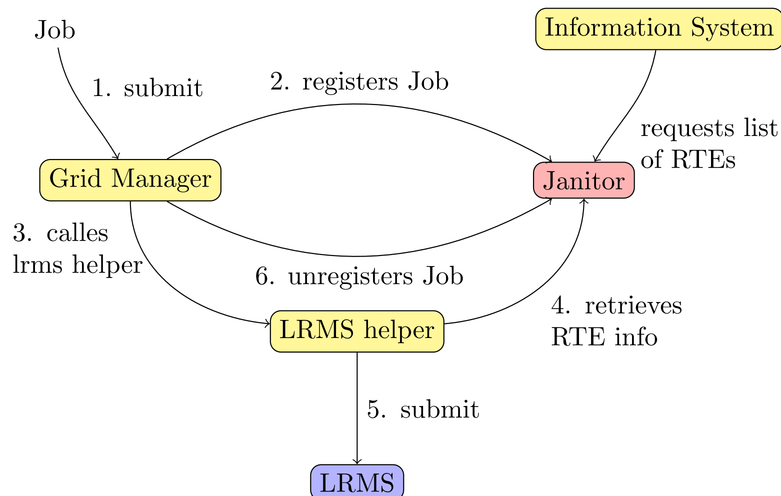

Figure 4: **Integration of the Janitor with the ARC middleware.**

Integration with the local Information System For collecting the information on RTE the Grid Information System calles one of the helper scripts in `libexec` (e. g. `cluster-pbs.pl`) which uses the module `InfosysCluster.pm`. Usually this modules checks for RTEs by searching the filesystem for their runtime scripts.

The only needed modification for integrating the Janitor is changing this module in such a way, that it does not search the filesystem itself but asks the Janitor which RTEs are available. For efficiency the Janitor has a perl module interface (`RuntimeEnvironments.pm`) for this. So it is not nescessary to create a new perl process.

This polling for RTEs is done on a regular basis by the local Information System. This is why the corresponding arc in figure 4 has no number.

Integration with the Grid Manager The Janitor is meant to be executed by the Grid Manager (Konstantinov, 2004). Basicaly there are two possiblities to integrate the Janitor. Either (i) by changing the batch system backend scripts or (ii) by changing the Grid Manager itself. In the second case a small change of the backend scripts is also needed. The first approach allows to use an unmodified ARC installation. The second approach of a direct integration needs changes of the binaries but is more powerfull. In the following we describe the second approach.

Figure 4 shows how the Janitor and the Grid Manager interacts. The numbers indicate the order of the interactions. These are:

1. **submit:** The Job is submitted to the Grid Manager as usual. Up to now there is no support for special attributes related to DRE. In the future there will be support for attributes like `validated`.
2. **registers Job:** During the state `PREPARING` the Grid Manager downloads the data needed by the job. Now this state is extended in such a way that also the Janitor is called. In this case the Job is registered. This means the Grid Manager sends the Job-ID and the list of needed RTEs to the Janitor. The Janitor checks if these RTEs are available. If necessary they are dynamically installed. If they are not available a failure is returned and the Job is put into the error state. Else the new state is `SUBMITTING`.
3. **calles LRMS helper:** In the state `SUBMITTING` the Grid Manager calles a backend script, called LRMS helper in the figure, to submit the job to the local batch system. Nothing was changed at this point.
4. **retrieves RTE info:** The LRMS helper constructs a script which is send to the local batch system for execution. For this it needs to know the locations of the runtime scripts of the RTEs used by the Job. So the LRMS helper was changed in such a way that it askes the Janitor for these. For this it only needs to know the Job-ID.
5. **submit:** Finally the constructed script is send to the local batch system. At this point no changes are necessary.
6. **unregisters Job:** After execution of the Job the Grid Manager continues in the state `FINISHING`. Here the results are uploaded. This state was changed in such a way that the Grid Manager also calles the Janitor and unregisters the Job.

The integration described is for the case with a shared filesystem available on all worker nodes, which is used for installation of DREs. If there is no such filesystem the actual integration might be different. Especially step five.

3.2 Catalog and RDF description of RTEs

3.2.1 Functionality & Implementation details

The Catalog describes Runtime Environments. The current implementation represents all information on the Runtime Environments in a single RDF file. The Catalog file is either served through a Web server or distributed together with the Janitor on the computing element. Currently, there is no real service around the catalog file. The Catalog file in the current representation offers the only opportunity for an exchange of formal descriptions of dynamic Runtime Environments. The editing of the RDF catalog file can be performed locally with a text editor or preferably with respective tools like Protégé (protege.stanford.edu).

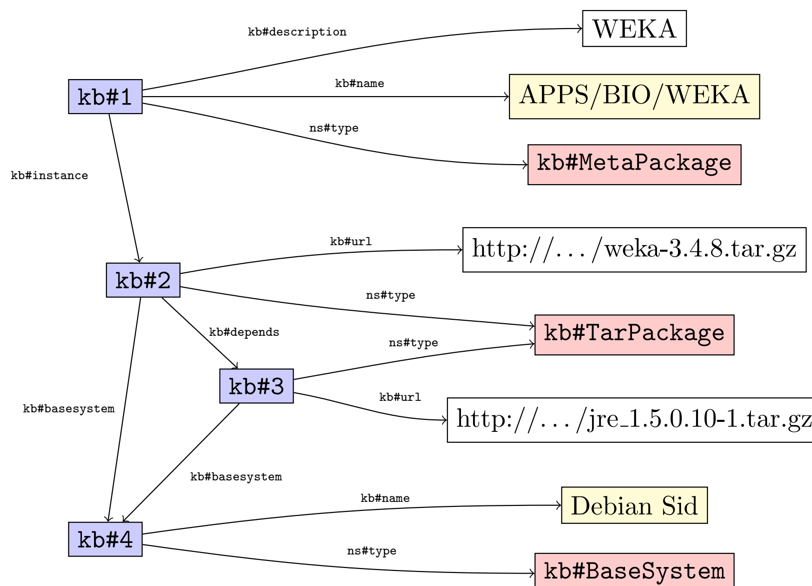

Figure 5: **RDF triplets in the Catalog.** *At the example of the dynamic runtime environment for WEKA, the link from the MetaPackage to its package (the tar file) and the base system (Debian Sid) is shown. The package has a dependency to the Java runtime environment which is another package.*

3.2.2 RDF Schema for the description of Runtime Environments

The Catalog contains RDF-based descriptions of dynamic RTEs. It contains three different types of entities (see figure 6): these are called MetaPackage, Package, and BaseSystem.

The MetaPackage entities describe the RTEs. The current system used within the NorduGrid community consists of an ID, version information and an informal description. Also it links to a web page describing how to manually install the RTE. In the Catalog these informations are stored in MetaPackage entities. Additionally the MetaPackage entities have associated Package entities who describe how to deploy automatically.

The BaseSystem entities are used to distinguish different operating systems of the worker nodes. The only mandatory value they have is their name. But if virtualisation is used, then the BaseSystem may also have a URL value, which describe where to find a minimal image of this kind of basesystem. In the non-virtualised case this node is only used as a hook.

The Package entities describe how to deploy some MetaPackage on some BaseSystem. So each MetaPackage links to possible multiple Packages which in turn link to exactly one BaseSystem.

Actually there are several subtypes of Packages. The subtype implicitly describes how to install a Package. One of these subtypes is the TarPackage. It has a URL describing where to find the tar file containing the software. Another subtype is DebianPackage. This subtype has no URL attribute but a list of Debian packages, which must be installed to provide the requested RTE. Further Package nodes may depend on each other and be implicitly installed.

The link to the physical representation of a software on the hard disk is presented via the *Package* class. It has two specialisations to represent packages of the Debian Linux distribution[5] and such presented as a tape archive (tar) file.

MetaPackages are provided by Packages specific to a BaseSystem. The idea behind this is, that the user specifies the MetaPackage and the site admin the BaseSystem. The Janitor queries the Catalog and learns which Packages it has to install.

The schema formally representing the Runtime Environments, which also serves as the schema of the Catalog, is attached in the Appendix A. Figure 5 shows a part of the RDF Graph describing the WEKA Meta Package.

RDF is machine readable and accessible from all modern programming languages. As an extension of the XML format it can be transformed easily to other representations like HTML. The small utility `list.pl` is

[5] `http://www.debian.org`

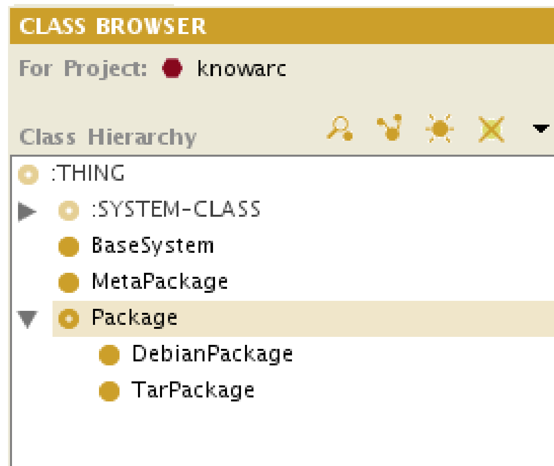

Figure 6: **Screenshot of the class hierarchy as displayed in the tool Protégé.**

provided for that purpose.

3.2.3 Integration with current ARC

The Catalog has no integration with ARC 0.6, except for being specified in the /etc/arc.conf configuration file by its URL. Multiple Catalogs can be specified. The logics to decide for the eligibility of a package that is described in the Catalog for the actual installation at a computing element, is solely left with the Janitor.

3.2.4 HTML interface of the catalog

The dynamic Runtime Environments stored in the catalog are presented on the aforementioned dedicated web page [6]. This site also links to both the formal Catalog in RDF syntax and an automated transformation to HTML. The latter mimics the traditional site describing Runtime Environments in the Runtime Environment Registry [7] in order to minimise issues with an eventual transition to the new system. That page collects descriptions for Runtime Environments to encourage human site administrators to install these.

This html page listing the manually or automatically installable RTEs is prepared by the script `web/list.pl`. This script is meant to be run by a mod-perl enabled Apache. The script itself is only loosely integrated within the Janitor. In the first lines of the script some variables specific to the site are set. To configure the script these have to be changed.

3.3 Repository

3.3.1 Functionality & Implementation

A repository shall offer sets of software packages for download. Any site on the Internet from which data can be downloaded and that is accessible by a unique URL may be accessed as a repository. This includes web services and grid-based solutions for distributed storage.

Preparation of Runtime Environments for the Repository At the time of writing, only the *tape archive* (tar) file format is accepted for the dynamic Runtime Environments, a well accepted file format throughout the UNIX community. This section explains the inner structure of the tar files. Subdirectories are visualised in figure 7.

The tar file contains two directories named *control* and *data*. Software is stored in the latter subdirectory, while the files formally specifying how to deal with such packages are stored in the prior. Upon installation, the content of the data directory is extracted to some directory $BAR. After this unpacking of the tar file,

[6]`http://dre.knowarc.eu:8080/list.pl`
[7]`http://gridrer.csc.fi/`

```
foo.tar.gz
  |
  |-- control/
  |       |
  |       |-- install
  |       |
  |       |-- remove
  |       |
  |       `-- runtime
  |
  `-- data/
```

Figure 7: **Directory structure in tar files of dynamic Runtime Environments.**

the Janitor executes the install skript provided in the control directory. It is executed within the working directory $BAR. The job of this skript is to perform any necessary post-processing.

The Janitor stores the file `control/remove`. It will be executed in the same way as `control/install` just before the tar-package is removed. In most cases `control/remove` will be empty.

Finally, the file `control/runtime` is sourced multiple times by the Grid Manager's job-submit script. After installing the package, the Janitor changes all occurences of %BASEDIR% in the runtime script to $BAR.

To be offered to computing elements for an installation, the such prepared runtime environment must be announced to a Catalog to which the Janitor on the computing element subscribes.

3.3.2 Protoypes

WEKA The WEKA package for machine learning (Frank et al., 2004) and the Java Runtime Environment[8] are available as dynamic Runtime Environments. Further packages for bioinformatics comprise dynamic variants of tools for the analysis of transcription factor binding sites. These are already offered for manual installation via the prior mentioned traditional page representing Runtime Environments for ARC.

CRAN and BioConductor.org To demonstrate the technical proximity to scientific communities that provide packages for the Debian Linux distribution, a tool was prepared to transform Debian packages to dynamic runtime environments (Möller et al., 2007). This effort comprising more than 1700 packages and thus also helps to analyse the scalability of the RDF-based tool for the analysis of dependencies between projects.

ATLAS To address the concerns of the physicists using ARC, a dynamic runtime environment for the ATLAS software suite was prepared. It extends prior work on an automated installation that is available at `http://guts.uio.no/atlas/12.0.6/`. The preparation comprised the following steps:

- The file system path specifications in the automated installation scripts were modified using the Janitor path variables.
- A tarball was prepared containing a directory structure as illustrated in figure 7. The data directory was empty, since the automatic installation script downloads the software from a remote server.
- An entry was added to the Catalog file.

What sets High Energy Physics software apart is it's sheer size. The package in question takes up more than 5 GB. This was a test illustrating the feasibility of using DREs in High Energy Physics. The application of the DRE for ATLAS needs to wait for the planned web service extension of the Catalog. With such a service, e.g. a software manager of a big experiment will be able to deploy software packages on production sites simply by creating a tarball and adding an entry to the Catalog.

[8] `http://java.sun.com`

4 Deployment

4.1 Requirements

In order to deploy the Catalog and the Janitor the following requirements should be satisfied:

Web server A local installation of Apache 2.0 is used for the presentation of the Catalog to users.

Perl The Janitor requires the following non-standard perl packages:

- RDF::Redland[9] (Beckett, 2001)
  This package implements the basic access and query mechanisms for RDF files.
- Log::Log4perl[10]
  The package Log::Log4perl ist optional. The Janitor will work without it, but there will be no logging.
- Time::HiRes[11]
  A Perl Package needed for debug purposes and is optionally to install.

The Janitor is written in Perl. Besides a Perl installation two additional modules are needed:

- `RDF::Redland` and
- `LOG::Log4perl`

These can be easily installed via CPAN or means of the used operating system. On Debian Linux this command can be used:

```
$ apt-get install liblog-log4perl-perl librdf-perl
```

The Jantior comes with an additional script which generates on the fly a web site similar to the old static Runtime Environment Registry. To use this `mod_perl` and the apache webserver must be installed. On Debian Linux these are installed by

```
$ apt-get install libapache2-mod-perl2
```

4.2 Installation

There are multiple ways to install the Janitor. One of these is described below.

1. Extract the contents of the tar archive to a directory, e. g. `/opt/janitor` and make sure the file Janitor.pl is executable.
2. It is suggested to create a user "janitor" and a group "janitor". The Janitor will use this user and group while installing software. A suid root wrapper for the Janitor is neccessary in this case.
3. Some directories are needed for storing the Janitor's database, downloaded files and installed software packages. Make sure, the Janitor has read and write permissions for these directories. Their location is configured in the `arc.conf`.
4. Some changes to the `arc.conf` are required. See the Janitor's POD documentation for details on this.

[9] `http://librdf.org/`, version 1.0.6 or later
[10] `http://log4perl.sourceforge.net/`
[11] `http://search.cpan.org/dist/Time-HiRes/`

```
[janitor]
logconf="/opt/janitor/work/log.conf"
registrationdir="/opt/janitor/work/reg"
installationdir="/grid/runtime/janitor"
downloaddir="/opt/janitor/work/download"
jobexpirytime="10000000"
uid="janitor"
gid="janitor"
allow_base="*tar*"
#allow_base="*"

[janitor/nordugrid]
catalog="/opt/janitor/catalog/knowarc"
allow_rte="*"
deny_rte="*/JAVA/*"

[janitor/bioc]
catalog="/opt/janitor/catalog/bioc_catalog.rdf"
allow_rte="LIBS/*"
```

Figure 8: **Configuration of the Janitor and the Catalog.** *Shown is an example setup for the Janitor that expects two catalogs as RDF files to be offered locally.*

5. To integrate the Janitor with the grid manager and the grid infosystem some files in the ARC `libexec`-directory must be changed. There are to examples on how to do this for the PBS backend. The patch for the case that the whole integration is done via the backend scripts is in `libexec.patch`. The patch for the case that the Grid Manager calles the Janitor directly is in `libexec_int.patch`. For applying on of these patches
   (a) stop the grid daemons,
   (b) change directory to the libexec folder, likely /opt/nordugrid/libexec
   (c) run `$ patch -p1 < pathto/file.patch`
   (d) start daemons again.
6. To remove unused DREs a cron job should be prepared that calls `...janitor.pl sweep` on a regular basis.

4.3 Configuration

In the current implementation, the configuration is only of concern for the Janitor. The Catalog, or multiple Catalogs, are stored as files on the file system and do not require further attention. A cron job that performs regular updates of these files is suggested to set up.

Figure 8 presents an example for the configuration of the Janitor as a part of the arc.conf configuration file. A detailed description of the options is presented in the man page of the Janitor.pl (see attachment B).

The `logconf` option indicates a separate file that specifies the options for the logging of events associated with the handling of DREs. The Log4Perl library comes with its own respective documentation, figure 9 shows an example.

The `registrationdir` stores the information for the Janitor what DREs are in used by what jobs, thus, should not be updated or removed. The `installationdir` specifies the directory to which dynamically installed REs are installed. The `downloaddir` specifies the directory to which packages are downloaded that are specified in the Catalog's description.

Multiple RDF files can be presented as a single conceptional Catalog and is represented by its respective own section in the `arc.conf` file. This way, each source of packages can be constrained individually for the elegibility of packages. The `allow_rte` and `deny_rte` options are regular expressions on the name of the package.

```
# Master Loglevel
# [OFF | DEBUG | INFO | WARN | ERROR | FATAL]
#log4perl.threshold = DEBUG

log4perl.rootLogger = DEBUG, DebugLog, MainLog, ErrorLog, ScreenLog
log4perl.category.Janitor = WARN, DebugLog, MainLog, ErrorLog, ScreenLog

log4perl.appender.DebugLog.Threshold = DEBUG
log4perl.appender.DebugLog = Log::Log4perl::Appender::File
log4perl.appender.DebugLog.filename = /opt/janitor/work/log/debug.log
log4perl.appender.DebugLog.layout = PatternLayout
log4perl.appender.DebugLog.layout.ConversionPattern = %d %p> %m%n

log4perl.appender.MainLog = Log::Log4perl::Appender::File
log4perl.appender.MainLog.Threshold = INFO
log4perl.appender.MainLog.filename = /opt/janitor/work/log/janitor.log
log4perl.appender.MainLog.layout = PatternLayout
log4perl.appender.MainLog.layout.ConversionPattern = %d %p> %m%n

log4perl.appender.ErrorLog = Log::Log4perl::Appender::File
log4perl.appender.ErrorLog.Threshold = ERROR
log4perl.appender.ErrorLog.filename = /opt/janitor/work/log/error.log
log4perl.appender.ErrorLog.layout = PatternLayout
log4perl.appender.ErrorLog.layout.ConversionPattern = %d %p> %m%n

log4perl.appender.ScreenLog = Log::Log4perl::Appender::Screen
log4perl.appender.ScreenLog.Threshold = ERROR
log4perl.appender.ScreenLog.layout = SimpleLayout
```

Figure 9: **Configuration of parameters for logging.** *The logging is performed by the Log4perl library which is configured separately.*

4.4 Security Consideration

Security is a major concern for the grid systems. Automatic software installation inheritently introduces security threats. This section tries to evaluate those and describe the available solutions to limit security risks.

In the current installation, every user authorised to execute a job is also authorised to install a Runtime Environments. Restrictions are only imposed on the set of DRE that are available for installation. Restrictions are imposed by the site admininistrators on the descriptions that are given by the Catalog that is offering the package. These descriptions may explicitly mention DRE names, a regular expression on these, or refer to tags of packages that categorise these. However, the core of these controls lie with the maintainers of the Catalog, who needs to be trusted.

All DRE are installed in separate directories. The provisioning of disk space is the duty of the site administrator. In the current implementation, the installation is completely transparent to the user:

- DREs are not distinguished between *installed* and *installable* in the information system.
- No status information is given at the time an DRE is installed.

Malevolent regular users with respective training in using system exploits to gain root access are likely to find security holes by regularly submitted scripts. The authentication and authentification of users, together with respective logging, is the major defense against such attacks. What is consequently left to be protected against are unwanted sideeffects by the installation of software.

The worst case scenario would be the installation of a Runtime Environment that overwrites system files. With the current implementation, which is based solely on tar files, this is barely possible, unless such is performed by the install scripts that accompany the tar files. However, hereto the installation would have to be performed by a user with system priviledges, for which there is no technical requirement.

The installation of packages from the Debian distribution (or other packages of mainstream Linux distributions) is seeked to reduce the complexity and burden in the maintenance for DRE. In the current implementation Debian packages be installed only by their transformation into tar files. With the advent of the interface for the virtualisation of the grid infrastructure, it is anticipated to work with native packages of the Debian Linux distribution. The reuse of packages that passed many eyeballs - as it is the case with packages from major Linux distribution - security is further increased or becomes as high as with the operating system underneath virtual clients.

Summarising, there is general concern about the security of grid computing. Dynamic Runtime Environments introduce new dangers since a manual control at the grid site is substituted by a remote process that is out the direct supervision of a local site administrator. The signing of packages by known and directly or indirectly trusted developers is a good indicator that no malevolent individuals have tampered with the binary. The site administrators can limit the sources of packages and specify packages that are eligible or excluded from installations.

5 Further considerations, outlook on future developments

Representation of dynamic REs in the information model

Dynamic Runtime Environments require an extended representation in the information model. The Application Software description should be able to distinguish installed REs from installable REs, potentially offer description of extended RE state-like information. This work is planned to be carried out as part of the Glue-2.0 effort of OGF[12].

Integration with Workflow Management

Future development of ARC aims at integrating grid computing with workflow tools for the web-services that have a growing user base in bioinformatics. The challenge is to prepare Runtime Environments for programs or databases and to offer such concisely to users of the workflow environments. In the bioinformatics community, such are today offered as web services. This anticipated developments instead fosters the dynamic installation on the grid whenever appropriate to allow for special computational demands in high-throughput analyses. Conversely, because of the increased complexity of workflows with respect to the already today not manually manageable number of REs, without an automatism for the automated installation of software packages on the grid, the use of workflows in grid computing seems mute.

Implementation of a Catalog service

A Catalog service is planed to be implemented on top of the ARC HED component. This service will render the currently used locally accessible RDF file externally accessible. Selected users are then allowed to remotely add/edit/remove REs to/from to it. The Janitor will access the content of the Catalog through a well-defined Web Service interface.

Integration with the Virtualization work

The RDF Schema nicely prepares for the upcoming virtualisation of worker nodes. How exactly the dynamics are integrated will depend on how dynamic the virtualisation of the nodes is. In the simplest scenario, a worker node's CPU will only be occupied by a single virtual machine and that will not be changed. In this case, there is no difference to the setup of the Janitor with today's static setups.

However, if the base systems are substitutable dynamically, then a Runtime Environment can possibly be offered via multiple base systems. The RDF Schema describes base systems as separate instances and as such differs from the current RE registry. Heuristics that prefer one base system for another can make direct use of the data that is presented in the schema. The integration of packages from Linux distributions in the description of REs is essential to have a means to decide for the equivalence of manual additions and the functionality that comes with base system.

[12] OGF GLUE: https://forge.gridforum.org/sf/projects/glue-wg

6 Acknowledgements

Many thanks go to all our colleagues participating through KnowARC or directly in the development of the ARC infrastructure. Marek Kočan is thanked for testing this document in practice.

References

David Beckett. The design and implementation of the redland rdf application framework. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 449–456, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-348-0. doi: http://doi.acm.org/10.1145/371920.372099.

M. Ellert, M. Gronager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimaki, O. Smirnova, and A. Wäänänen. Advanced resource connector middleware for lightweight computational grids. *Future Generation Computer Systems*, 23(2):219–240, 2007.

E. Frank, M. Hall, L. Trigg, G. Holmes, and IH. Witten. Data mining in bioinformatics using weka. *Bioinformatics*, 20(15):2479–2481, 2004. URL `http://www.cs.waikato.ac.nz/ml/weka`.

Aleksandr Konstantinov. The nordugrid grid manager and gridftp server: Description and administrator's manual, 2004. URL `http://www.nordugrid.org/documents/GM.pdf`.

Steffen Möller, Daniel Bayer, David Vernazobres, Albrecht Gebhardt, and Dirk Eddelbuettel. Scientific grid computing via community-controlled autobuilding of software packages across architectures. In *Proceedings of NETTAB 2007, A Semantic Web for Bioinformatics: Goals, Tools, Systems, Applications*, Pisa, Italy, June 12-14th 2007. URL `http://www.nettab.org`.

The KnowARC Consortium. Deliverable d1.1-1: Design document, January 2007. URL `http://www.knowarc.org/documents/Knowarc_D1.1-1_07.pdf`.

A RDF Schema

This section listst the RDF schema for the representation of Runtime Environments.

B Man pages

The inner working of the Perl modules is documented as the Perl-typical POD descriptions that are part of the source code but can be transformed into UNIX-like man pages.

NAME

Janitor.pl − The Janitor

SYNOPSIS

Janitor.pl [new | info | remove | sweep | help]

DESCRIPTION

Mangages Dynamic Runtime Environment for Grid Jobs.

OPTIONS

new <job−id> [RTE1 [RTE2 ...]]
: Registers the job <job−id> within the Janitor. RTE1 ... RTEn are the RTEs the Job requested. If the job was succeesfully registered and all requested RTEs are available the Janitor return true, else false.

info [<job−id>]
: Lists informations about known jobs. If a <job−id> is given detailed informations about this job are listed.

remove <job−id>
: Removes the job <job−id> from the Janitors database.

sweep [−−force]
: Removes dynamically installed RTEs which were not used for a long time. If −−force is given it will also remove old Jobs. These timespans can be configured in the arc.conf.

CONFIGURATION FILES

arc.conf

The Janitor can be configured in the [janitor]−section of the *arc.conf*. The following options are supported.

logconf
: Location of the Log4perl configuration file.

registrationdir
: The directory the Janitor uses to keep track of registered jobs and dynamically installed RTEs.

installationdir
: The directory dynamically RTEs are installed.

downloaddir
: A directory the Janitor uses to store downloaded files.

rteexpirytime
: A time in seconds. RTEs which are unused for this time are removed by sweep.

jobexpirytime
: A time in seconds. Jobs which are older than this are removed by sweep −−force.

uid, gid
: The user and group id to use.

allow_base, deny_base
: With this options the admin may choose which kind of basesystems are allowed. Initially all basesystems are forbidden. In the first step the allow_base options are processed and everything matching them is allowed. In the second step the deny_base options are processed and everything matching them is forbidden. Multiple allow_base and deny_base options are allowed.

allow_rte, deny_rte
: Like allow_base and deny_base, but for RTEs.

Additionally there is a section [janitor/name] in the arc.conf for each catalog used. ''name'' is the (arbitrary) name of the catalog. The following options are supported:

catalog
: The location of the catalogs RDF file. Currently this must be a local file.

allow_rte, allow_base, deny_rte, deny_base

These options have the same functionality as the options with the same name in the [janitor]−section. But they are specific to this catalog. If they are not given the ones in the [janitor]−section are used.

log.conf

The *log.conf* file is used to configure the Log4perl logger. The location of this file is given in the [janitor]−section of the *arc.conf*. For examples how to configure Log4perl see the Log4perl documentation.

ENVIRONMENT

NORDUGRID_CONFIG

Location of the nordugrid arc.conf. If unset /etc/arc.conf is used.

EXAMPLE

[janitor] logconf=''/opt/janitor/work/log.conf'' registrationdir=''/opt/janitor/work/reg'' installationdir=''/grid/runtime/janitor'' downloaddir=''/opt/janitor/work/download'' jobexpirytime=''10000000'' uid=''janitor'' gid=''janitor'' allow_base=''*tar*'' #allow_base=''*'' # #[janitor/nordugrid] #catalog=''/opt/janitor/catalog/knowarc.rdf'' #allow_rte=''*'' #deny_rte=''*/JAVA/*'' # #[janitor/bioc] #catalog=''/opt/janitor/catalog/bioc_catalog.rdf'' #allow_rte=''LIBS/*''

SEE ALSO

http://dre.knowarc.eu:8080

NAME

Janitor::BaseSystem − Manages information about one basesystem

SYNOPSIS

use Janitor::BaseSystem;

METHODS

new()
: The constructor.

name($name)
: Sets and gets the name of the basesystem. The argument is optional.

id($id)
: Sets and gets the id of the basesystem. The argument is optional.

description($description)
: Sets and gets the description of the basesystem.

lastupdate($lastupdate)
: Sets and gets the lastupdate filed of the basesystem.

homepage($homepage)
: Sets and gets the homepage url of the basesystem.

immutable($immutable)
: If `$immutable` is true the basesystem is marked as immuatable. After this it is not possible anymore to change its values. Once marked immutable this process can not be reversed.

NAME

Janitor::Catalog – Implements the Interface to the Catalog

SYNOPSIS

use Janitor::Catalog;

DESCRIPTION

METHODS

new($class, `$file`, `$name`)
: The constructor for this class has two arguments, the name of the catalog file and the name of the catalog. It uses RDF::Redland to open the calaog and returns an object for accessing it.

name()
: This method returns the name of the catalog.

basesystems_supporting_metapackges(@mp)
: Given a list of metapackages (by name) this method returns a list of basesystems (as Janitor::BaseSystem) which support all of these metapackages.

PackagesToBeInstalled($bs, `@mp`)
: Given a basesystem `$bs` (as Janitor::Basesystem) and a list of metapackages `@mp` (as string) this method returns a list of packages (as Janitor::Package) which must be installed to provide the requested metapackages.

: This method returns a tupel. The first element is true iff the requested combination of metapackages is supported by the basesystem. The second element is the list of packages.

BaseSystems()
: Returns a list of all basesystems as Catalog::BaseSystem.

MetaPackages()
: Returns a list of all metapackages as Catalog::MetaPackage.

Packages()
: Returns a list of all packages as Catalog::Package or a subclass of it.

AllowMetaPackage(@mp), DenyMetaPackage(@mp)
: Allows or denies all listed metapackages (by name) to be used. Tags and Wildcards are supported.

AllowBaseSystem(@bs), DenyBaseSystem(@bs)
: Allows or denies all listed basesystems (by name) to be used. Wildcards are supported.

AllowAll(), *DenyAll()*
: Allows or denies all metapackages and all basesystems.

SEE ALSO

NAME

Janitor::Common – Provides some common functions

SYNOPSIS

use Janitor::Common;

DESCRIPTION

This module contains the function *get_catalogs()* which is not only used by the Janitor but also by the module RuntimeEnvironments.

FUNCTIONS

get_catalogs()
    Initializes all configured catalogs and returns a list of these.

NAME

Janitor::Config – Provides access to the *arc.conf*

SYNOPSIS

use Janitor::Config qw(get_config);

DESCRIPTION

This class provides access to the *arc.conf*. It is implemented as a singleton.

To use this module the following code is needed:

```
use Janitor::Catalog;
my $config = Janitor::Config->parse("/etc/arc.conf");
```

after this the configuration values can be accessed. For example

```
my $un = $config->{'janitor'}{'uid'};
```

gives the value of the uid option within the [janitor]–section of the *arc.conf*.

This code can be used to get the singleton in another module:

```
use Janitor::Config qw(get_config);
my $config = get_config();
```

METHODS

Janitor::Config::parse($file)

This method parses the given file and adds its content to the singleton. For debugging purposes this method might be called multiple times for different files. Please note this feature is not tested well enough for production use.

Janitor::Config::get_config()

This function returns a reference to the singleton hash providing access to the configuration file.

Janitor::Config::changed($file)

This function returns a scalar which is true iff the given file was changend more recently than the last read of the configuration file.

Janitor::Config::reinitialize($file)

This method reinitializes the configuration hash, i.e. its content is replaced with the content of `$file`.

NAME

Janitor::DebianPackage − Manages information about one debian package.

SYNOPSIS

use Janitor::DebianPackage;

DESCRIPTION

This class is a decendant of Janitor::Package. Below only the additional methods are listed.

To deploy packages of this kind virtualisation is needed. This is currently not supported by the Janitor.

METHODS

package(@package)
: Sets and gets the list of debian packages which must be installed

debconf(@debconf)
: Sets and gets the list of debconf entires which must be feeded to debconf prior of installation.

NAME

Janitor::Execute – Executes a Command

SYNOPSIS

use Janitor::Execute;

DESCRIPTION

This class is uesed for executing external programms. This is done in a syncronous way, i.e. this process waits until the child process dies.

METHODS

Janitor::Execute::new($workdir, `$logfile`, `$chroot`)

The constructor. Its first parameter is the name of the directory which shall be the cwd of the programm to execute. The second parameter is the location of the file to use as logfile. Iff the third parameter is true chroot is called prior execution of the programm. Root rights are neccessary for this.

The second and the third parameter are optional.

$obj–>uid($uid)

Sets and returns the uid which will be used to execute the command. The parameter is optional.

$obj–>gid($gid)

Sets and returns the gid which will be used to execute the command. The parameter is optional.

$obj–>execute(@command)

Executes the given command. For this it forkes a child process which exec the command. The parent process waits for the child to finish. Currently timeouts are not supported.

This method returns a non-zero value if the execution succeeded.

$obj–>retval

This method returns the return value of the child process. If the process was killed by a signal it returns undef.

$obj–>signal

This method returns the number of signal which killed the child process.

EXAMPLE

Below is a simple example which shows how to use this class.

```
use Janitor::Execute qw(new);
my $e = new Janitor::Execute($workdir, "/tmp/logfile");
if ($e->execute("/bin/echo", "Hello World!")) {
    print "successfully executed /bin/echo\n";
} else {
    print "error while executing /bin/echo\n";
    if (defined $e->retval) {
            print "return value: %s\n", $e->retval;
    } elsif (defined $e->signal) {
            print "killed by signal %s\n", $e->signal;
    } else {
            print "unable to fork?\n";
    }
}
```

After this some debug informations can be found in */tmp/logfile*. The output of the command is also stored in this file.

NAME

Janitor::Filefetcher – Downloads files for the Janitor.

SYNOPSIS

use Janitor::Filefetcher;

DESCRIPTIONS

This class is used for downloading files. Currently it supports the protocols http, https and ftp.

METHODS

Janitor::Filefetcher::new($url, `$destination`)

The constructor creates an object which will be used to download `$url` and store this file in the directory `$destination`.

$obj–>*fetch()*

This method is used to actually download. It returns true iff the file was downloaded successfully. The name of the file will have some random bits. The getFile-method can be used to retrieve the actual name.

$obj–>*getFile()*

This returns the name of the downloaded file.

NAME

Janitor::InstallLogger – writes log messages to a file

SYNOPSIS

use Janitor::InstallLogger;

DESCRIPTION

The Janitor uses Log4perl for logging. But as this is optional it uses this home grown Logger for logging in places where logging must be done. The only such place is the installation of packages by Janitor::Execute.

METHODS

Janitor::InstallLogger::new($logfile, `$mode`)

Creates an InstallLogger which logs to `$logfile`. The permissions of this file are set to `$mode`. The logfile itself is openend. It will be closed again in the destructor.

$obj–>debug, `$obj`–>info, `$obj`–>warn, `$obj`–>error, `$obj`–>fatal

Writes a messages to the log.

NAME

Janitor::Installer – Installs a given package

SYNOPSIS

use Janitor::Installer;

DESCRIPTION

This class is used to install a given package into a directory.

METHODS

Janitor::Installer::new($instdir)

The consturctor for this class. The argument is the name of the directory in which the packages will be installed.

$obj–>install($file)

This method installes the package `$file`. Currently it supports tar packages, which might be compressed with gzip or bzip2.

The installation process includes the unpacking, the execution of the install script and the preparation and storage of the runtime script. It does not include the registration of this new package within the registration directory.

NAME

Janitor::MetaPackage – Manages information about one metapackage

SYNOPSIS

use Janitor::MetaPackage;

METHODS

new()
: The Consturctor.

name($name)
: Sets and gets the name of the metapackage.

id($id)
: Sets and gets the id of the metapackage.

description($description)
: Sets and gets the description of the metapackage.

homepage($homepage)
: Sets and gets the homepage of the metapackage.

instance($instance)
: Sets and gets the instance of the metapackage.

lastupdate($lastupdate)
: Sets and gets the lastupdate of the metapackage.

tag($tag)
: Sets and gets the tag of the metapackage.

NAME

Janitor::Package − Manages information about one package.

SYNOPSIS

use Janitor::Package;

METHODS

new()
: The constructor.

id($id)
: Sets and gets the id of the package.

basesystem($basesystem)
: Sets and gets the basesystem of the package.

NAME

Janitor::TarPackage − Manages information about one tar package.

SYNOPSIS

use Janitor::TarPackage;

DESCRIPTION

This class is a decendant of Janitor::Package. Below only the additional methods are listed.

METHODS

url($url)
: Sets and gets the url of the tar package.

NAME

Jantior::Util – Some functions used by different parts of the Janitor

SYNOPSIS

use Jantior::Util qw(remove_directory manual_installed_rte asGID asUID)

FUNCTIONS

remove_directory($dir)
: Removes the directory `$dir` and all its content. Returns true iff successfull.

manual_installed_rte($dir)
: This function searches in `$dir` for RTEs which are installed in the old fashioned manual way. They are returned as a list.

asUID($in)
: If `$in` is a user name its uid is returned. If `$in` is already numeric it is returned.

asGID($in)
: Same as asUID($in) but for group ids.