It is about time for the AbTLinux dependency engine (v1.0)

L. Klein Holte ludo@abtlinux.org
M. Jansen marco@abtlinux.org
J. Mutter jorgm@abtlinux.org
E.D. Schabell erics@abtlinux.org

February 6, 2006

Contents

1 Introduction

The workings of a software package management system (SPMS) is very complicated. Itis a tool to automate the process of managing a systems software packages. These kinds of tools are most commonly used on Unix-like systems, in particular Linux. Such systems rely heavily on their package management system which manage large amounts of software packages. A typical installation includes hundreds of packages, not an uncommon occurrence on modern day systems. Packages can have dependencies on other packages. It is essential to have some method to handle these dependencies systematically in a package manager. This is the goal of this dependency engine project, to define the necessary requirements to handle the dependencies for the ABout Time Linux (AbTLinux) distributions software package manager. AbTLinux stands for ABout Time Linux and it focuses on delivering a source-based Linux distribution.

Clear documentation is one of the main goals behind the development on the AbTLinux project. As the projects website states, "Clearly documented design, clear development goals leading to each release and just getting them done. Most members of the AbTLinux project have worked on other Linux distribution projects and have grown tired of working on badly documented designs." (AbTLinux, 2005) This requirements document teams goals consists of continuing this structured and clear way of gathering information for the dependency engine part of the AbTLinux project.

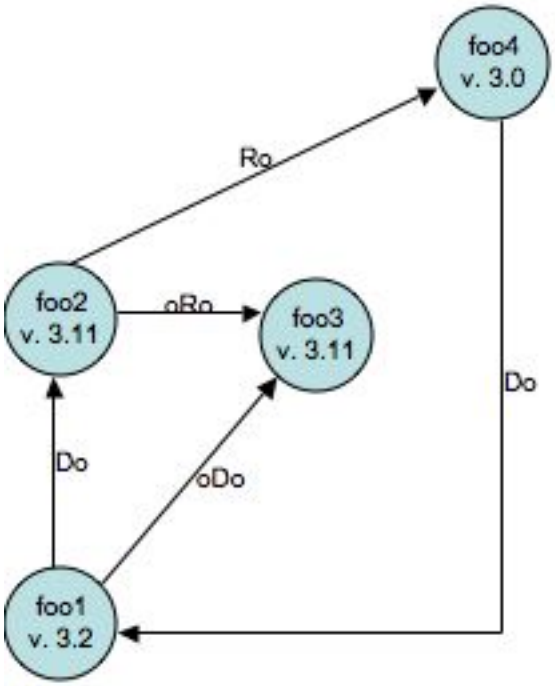
2 Problem Statement

There are two main problems: one has something to do with the developing process and the other is complex product-oriented challenge.

First: problems with most (if not all) other Linux distributions are due to bad documenting.

Second: Linux distributions may consist of hundreds of packages and these packages may have dozens of dependencies. Finding a way in this jungle of dependencies is a serious problem in an SPMS. All kinds of dependencies must be taken into account. For example, if a package manager only deals with build time dependencies then a software package can be installed and compiled. A problem pops up when one runs into run time dependencies, which are needed for starting and running the program. Another core problem is sorting a list of software packages to build in an order such that the minimum number of rebuilds are achieved. When a program has to be installed, re-installed, or reconfigured, it may have an affect on some other package(s).

Running example: the graph in Figure 1 shows our running example dependency tree which will be used throughout the rest of this document.



Legenda:
Package foo1 version 3.2 Depends on package foo2 version 3.11.
Package foo1 version 3.2 optionally Depends on package foo3 version 3.11.
Package foo2 version 3.11 optionally Relies on package foo3 version 3.11.
Package foo2 version 3.11 Relies on package foo4 version 3.0.
Package foo4 version 3.0 Depends on package foo1 version 3.2.

Figure 1: Example software package dependency tree

3 Mission, Vision and Values

3.1 Mission

The developers of the AbTLinux project have grown tired of working with badly documented Linux distributions what results in hard to manage code bases for the tools. Finding the right order for handling the dependencies between the different packages is another major question. Usually this is done using, for example, alphabetical ordering, which may result in dealing with the same package more than once. We have already seen an example of this problem in section 2.

3.2 Vision

The aim of this depEngine project is to develop a well documented dependency engine for AbTLinux which will solve the problems described in section 2.

3.3 Values

The main values of AbTLinux are its clear documentation and being a source based distribution. Clear documentation is obviously also the main value for the development of the dependency engine. We hope to accomplish this by providing clearly documented requirements for the project.

3.4 Executive Sponsors viewpoint

Most members of this project have worked on other Linux distribution projects and have grown tired of working on badly documented designs. This leads to badly organized growth paths for these distributions and hard to manage code bases for the tools. The common expression is that it is like hurding cats. We feel that it should not be like that. With clear goals, clear documentation and clear coding practices that everyone from user on down to developer can benefit from our project.

4 Statement of Work

An outline of exactly what will be done by the depEngine project group will be presented here. We provide the scope, objectives, an application overview and outline current project constraints that have an effect on the depEngine project.

4.1 Scope

The scope of our project consists of listing the requirements for the dependency engine of AbTLinux. The dependency engine can, given a command and a (set of) packages, deliver amongst others a proper build order for the packages. An example of this was given in the Problem Statement. Commands can be:

- install
- remove

- rebuild
- reconfigure
- upgrade
- downgrade
- repair

4.2 Objectives

We intend to obtain the basic requirements needed basic dependency handling for the dependency engine of the AbTLinux project. Our objective is to deliver clearly documented requirements which will ultimately be used for implementing the dependency engine for AbTLinux.

4.3 Application overview

The dependency engine is part of the distribution AbTLinux. It must be able to:

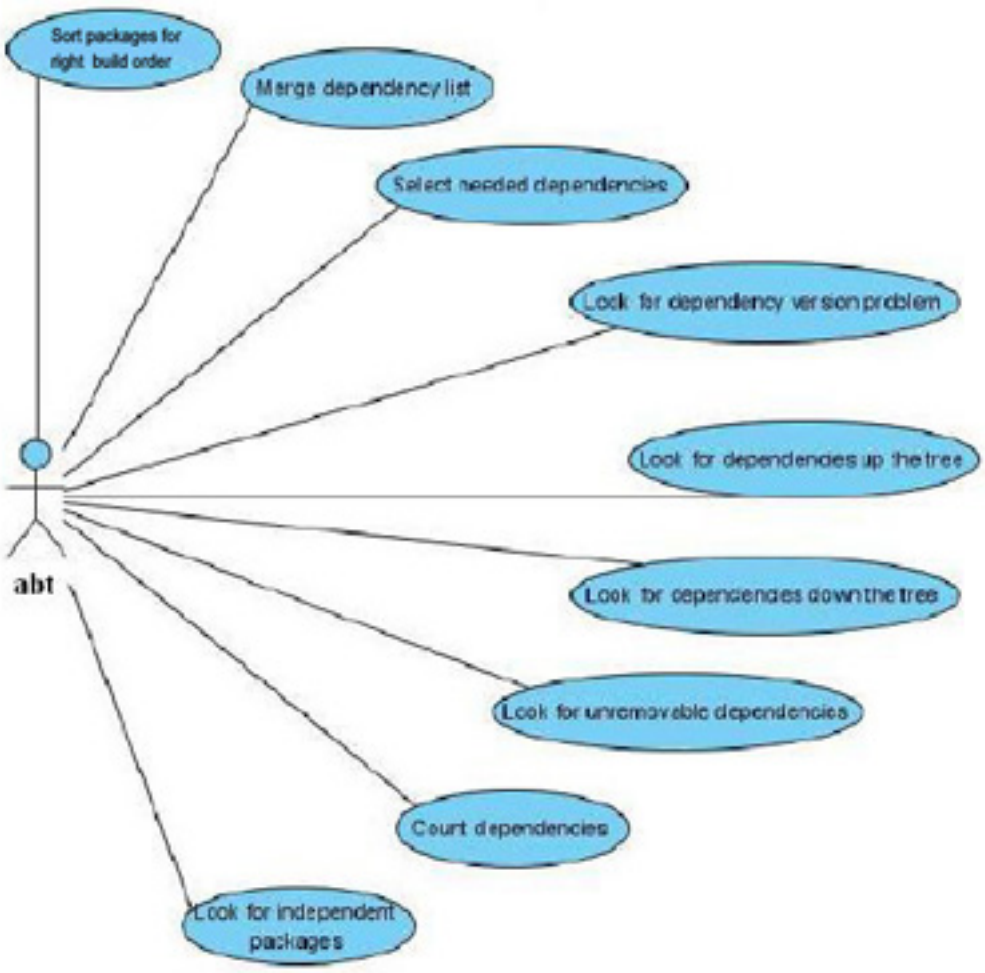
- provide a depends tree (up and down) for the packages
- given a package list, sort for proper build order
- provide a count of a package's dependencies
- provide a count of packages dependent on a given package
- provide a list of independent packages
- provide for depends versions
- log dependency engine actions.

4.4 Constraints

Some of the constraints were set by current AbTLinux project standards. The first constraint is the use of the English language because the developers come from different countries. The second constraint is the use of LaTeX for the document because LaTeX is the de facto standard for the production of technical and scientific documentation.

5 Use Case Survey

The use case diagram as shown below provides a complete overview of the use cases to be handled in this section. What follow is a survey of the discovered use cases, providing a general description of each use case.



5.1 Use Case Survey 1

Use Case Survey Name:	**Look for dependencies down the tree**
Use Case Survey Number:	001
Initiating actor:	abt
Description:	Look for existing package dependencies, i.e. packages which the given package is depending on.
Completeness:	Complete for Focused Stage
Maturity:	Mature for Focused Stage
Dependency:	None
Source:	Presentation by Eric Schabell
Comments:	Used in all operations except for Remove.

5.2 Use Case Survey 2

Use Case Survey Name:	**Look for dependencies up the tree**
Use Case Survey Number:	002
Initiating actor:	abt
Description:	Look for packages up the tree, i.e. packages depending on the given package.
Completeness:	Complete for Focused Stage
Maturity:	Mature for Focused Stage
Dependency:	None
Source:	Presentation by Eric Schabell
Comments:	Used in all operations, except for Install

5.3 Use Case Survey 3

Use Case Survey Name:	**Generate complete dependency list**
Use Case Survey Number:	003
Initiating actor:	abt
Description:	Generates the complete list of dependencies up and down the tree.
Completeness:	Complete for Focused Stage
Maturity:	Mature for Focused Stage
Dependency:	None
Source:	Presentation by Eric Schabell
Comments:	Used in operations Rebuild, Reconfigure, Downgrade, Upgrade and Repair

5.4 Use Case Survey 4

Use Case Survey Name:	**Select needed dependencies**
Use Case Survey Number:	004
Initiating actor:	abt
Description:	Selects the dependencies needed for the operation required by abt.
Completeness:	Complete for Focused Stage
Maturity:	Mature for Focused Stage
Dependency:	None
Source:	Presentation by Eric Schabell
Comments:	Used in all operations.

5.5 Use Case Survey 5

Use Case Survey Name:	**Sort packages for right build order**
Use Case Survey Number:	005
Initiating actor:	abt
Description:	Given a list of dependencies, create the right build order. In building the list, checks are performed to locate redundant operations and to locate cycles.
Completeness:	Complete for Focused Stage
Maturity:	Mature for Focused Stage
Dependency:	None
Source:	Presentation by Eric Schabell
Comments:	Used in all operations.

5.6 Use Case Survey 6

Use Case Survey Name:	**Look for dependency version problems**
Use Case Survey Number:	006
Initiating actor:	abt
Description:	Checks the versions of all dependencies to see if they are the same.
Completeness:	Complete for Focused Stage
Maturity:	Mature for Focused Stage
Dependency:	None
Source:	Presentation by Eric Schabell
Comments:	Used in Downgrade, Upgrade.

5.7 Use Case Survey 7

Use Case Survey Name:	**Count dependencies**
Use Case Survey Number:	007
Initiating actor:	abt
Description:	Counts all dependencies up or down the tree.
Completeness:	Complete for Focused Stage
Maturity:	Mature for Focused Stage
Dependency:	None.
Source:	Presentation by Eric Schabell
Comments:	Usable in all operations.

5.8 Use Case Survey 8

Use Case Survey Name:	**Look for unremovable dependencies**
Use Case Survey Number:	008
Initiating actor:	abt
Description:	The required dependencies are checked in the list of dependencies up the tree and sent as a report to abt.
Completeness:	Complete for Focused Stage
Maturity:	Mature for Focused Stage
Dependency:	None
Source:	Presentation by Eric Schabell
Comments:	Used in operation Remove.

5.9 Use Case Survey 9

Use Case Survey Name:	**Look for independent packages**
Use Case Survey Number:	009
Initiating actor:	abt
Description:	This Use Case is used to look for independent packages.
Completeness:	Complete for Focused Stage
Maturity:	Mature for Focused Stage
Dependency:	None
Source:	Presentation by Eric Schabell
Comments:	Used if AbTLinux ask for this action.

6 Use Cases

Here you will find the use cases worked out in as much detail as is needed to define each requirement.

6.1 Use Case 1

Use Case Name:	**Look for dependencies down the tree**
Authors:	AbTCans
Dates:	11-12-2005
Iteration:	Focused
Description:	Look for existing package dependencies, i.e. packages which the given package is depending on.
Actors:	abt
Preconditions:	Dependency engine has access to the package dependency description.
Triggers:	Triggering these actions in case of all operations, except for Remove.
Basic Course of Events:	1. Abt submits a request for a list of dependencies down the tree from a given package (and a depth). 2. Look for needed packages down the tree (to a given depth). 3. Generate a list dependencies down the tree. 4. Apply downtree-algorithm (see Appendix) to fill the list. 5. Return the list of dependencies up the tree. 6. Write actions to the log.
Alternative Path:	In step 2, if no dependencies are found, an empty list is returned.
Exception Paths:	If dependency data can not be retrieved from a package, return ERROR.
Assumptions:	Abt has a log to write actions to.
Postconditions:	Generated list of dependencies down the tree.
Related business rules:	None

6.2 Use Case 2

Use Case Name:	**Look for dependencies up the tree**
Authors:	AbTCans
Dates:	13-12-2005
Iteration:	Focused
Description:	Look for packages up the tree, i.e. packages depending on the given package.
Actors:	abt
Preconditions:	Dependency engine has access to the package dependency description.
Triggers:	Triggering these actions in case of all operations, except for Install.
Basic Course of Events:	1. Abt submits a request for a list of dependencies up the tree from a given package (and a depth). 2. Look for packages up the tree (to a given depth). 3. Generate a list dependencies down the tree. 4. Apply tree-algorithm (see Appendix) to fill the list. 5. Return the list of dependencies up the tree. 6. Write actions to the log.
Alternative Path:	In step 2, if no dependencies are found, an empty list is returned.
Exception Paths:	If dependency data can not be retrieved from a package, return ERROR.
Assumptions:	Abt has a log to write actions to.
Postconditions:	Generated list of dependencies up the tree.
Related business rules:	None

6.3 Use Case 3

Use Case Name:	**Generate complete dependency list**
Authors:	AbTCans
Dates:	12-12-2005
Iteration:	Focused
Description:	Generates the complete list of dependencies up and down the tree.
Actors:	abt
Preconditions:	Dependency engine has generated lists up and down the tree.
Triggers:	Triggering these actions in case of operations Rebuild, Reconfigure, Downgrade, Upgrade and Repair.
Basic Course of Events:	1. Abt submits a request for a list of dependencies up and down the tree and provides a list of dependencies up the tree and a list of dependencies down the tree. 2. The list of dependencies up the tree is merged with the list of dependencies down the tree. 3. Return the complete list of dependencies. 4. Write actions to the log.
Alternative Path:	None.
Exceptions:	If merge action fails, return ERROR.
Assumptions:	Abt has a log to write actions to.
Postconditions:	Generated list of all dependencies.
Related business rules:	None

6.4 Use Case 4

Use Case Name:	**Select needed dependencies**
Authors:	AbTCans
Dates:	13-12-2005
Iteration:	Filled
Description:	Selects the dependencies needed for the operation required by abt.
Actors:	abt
Preconditions:	Dependency engine has generated a list of dependencies. Dependency engine has specified an operation to be performed.
Triggers:	Triggering these actions in case of all operations.
Basic Course of Events:	1. Abt submits a request for a list of needed dependencies and provides a list of dependencies, as well as the operation to be performed with this list. 2. The list of dependencies is checked for dependencies needed to perform the given operation. 3. DepEngine returns the list of needed dependencies. 4. Write actions to the log.
Alternative Path:	In step 2, if no needed dependencies are found, return an empty list.
Exceptions:	If selecting operation fails, return ERROR.
Assumptions:	Abt has a log to write actions to.
Postconditions:	Generated list of all needed dependencies.
Related business rules:	None

6.5 Use Case 5

Use Case Name:	**Sort packages for right build order**
Authors:	AbTCans
Dates:	11-12-2005
Iteration:	Focused
Description:	Given a list of dependencies, create the right build order. In building the list, checks are performed to locate redundant operations and to locate cycles.
Actors:	abt
Preconditions:	Dependency engine has generated a list of dependencies.
Triggers:	Triggering these actions in case of operations Install, Rebuild, Reconfigure, Upgrade.
Basic Course of Events:	1. Abt submits a request for a sorted list of dependencies and provides a list of unsorted dependencies, as well as the operation to be performed with this list. 2. Choose algorithm to be applied according to requested operation. 3. Apply algorithm to sort the list of dependencies. 4. Return the list of sorted dependencies. 5. Write actions to the log.
Alternative Path:	None.
Exceptions:	If sorting operation fails, return ERROR. If algotithm returns cycle alert, return ERROR.
Assumptions:	Abt has a log to write actions to.
Postconditions:	Correct build order for operation to be performed.
Related business rules:	None

6.6 Use Case 6

Use Case Name:	**Look for dependency version problems**
Authors:	AbTCans
Dates:	11-12-2005
Iteration:	Focused
Description:	Checks the versions of all dependencies to see if they are the same.
Actors:	abt
Preconditions:	None
Triggers:	Triggering these actions in case of operations Upgrade or Downgrade.
Basic Course of Events:	1. AbT submits a request for a version check and provides a list of dependencies. 2. Each package on the list is checked to see if its version matches the desired version. 3. Packages that need to be upgraded or downgraded are placed on a list. 4. The list is returned to abt. 5. Write actions to the log.
Alternative Path:	None.
Exceptions:	If sorting operation fails, return ERROR. If no version can be found in some package, return ERROR.
Assumptions:	Abt has a log to write actions to.
Postconditions:	Generated list of needed dependencies.
Related business rules:	None

6.7 Use Case 7

Use Case Name:	**Count dependencies**
Authors:	AbTCans
Dates:	13-12-2005
Iteration:	Focused
Description:	Counts all dependencies up or down the tree.
Actors:	abt
Preconditions:	Dependency engine has generated a list of dependencies.
Triggers:	Triggering these actions in case of all operations.
Basic Course of Events:	1. Abt sends a list of dependencies up or down the tree, as well as the depth of the tree to which the elements need to be counted. 2. The elements on the list(s) are counted. 3. This number is returned to abt. 4. Write all actions to the log.
Alternative Path:	In step 1, if abt does not specify a depth, count all packages on the list.
Exceptions:	None.
Assumptions:	Abt has a log to write actions to.
Postconditions:	Generated count of dependencies.
Related business rules:	None

6.8 Use Case 8

Use Case Name:	**Look for unremovable dependencies**
Authors:	AbTCans
Dates:	11-12-2005
Iteration:	Focused
Description:	The required dependencies are checked in the list of dependencies up the tree and sent as a report to abt.
Actors:	abt
Preconditions:	Abt has generated a list of dependencies up the tree.
Triggers:	Triggering these actions in case of operation Remove.
Basic Course of Events:	1. Abt submits a request for a check for unremovable packages, as well as a list of dependencies up the tree. 2. The list of dependencies is checked for required dependencies on given package (one level up). 3. If the list does not contain required dependencies, return list. 4. Write actions to the log.
Alternative Path:	If the list contains required dependencies, return REPORT.
Exceptions:	If checking operation fails, return ERROR.
Assumptions:	Abt has a log to write actions to.
Postconditions:	Generated list of all dependencies to be deleted or a warning message that there are required dependencies on the list.
Related business rules:	None

6.9 Use Case 9

Use Case Name:	**Look for independent packages**
Authors:	AbTCans
Dates:	12-12-2005
Iteration:	Focused
Description:	This Use Case is used to create a list of independent packages.
Actors:	abt
Preconditions:	None.
Triggers:	Abt requests a list of independent packages
Basic Course of Events:	1. Abt submits a request for a list of independent packages. 2. Look for independent packages up and down the tree. 3. Return a list of all independent packages. 4. Write actions to the log.
Alternative Path:	In step 3, if no independent packages are found, return empty list.
Exceptions:	If checking operation fails, return ERROR.
Assumptions:	Abt has a log to write actions to.
Postconditions:	Generated list of all dependencies to be deleted.
Related business rules:	None

7 Scenarios

Each use case will be worked out using the running example found in Figure 1. These scenarios will provide enough detail for the design phase to work each use case out in a concrete implementation.

7.1 Scenarios Use Case 1

Use Case Name:	**Look for dependencies down the tree**
Use Case Steps:	1. DepEngine retrieves a request for dependencies down the tree for package foo1 v. 3.2. 2. DepEngine creates a list ”dependencies down the tree”. 3. DepEngine finds dependencies foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2. 4. DepEngine adds these dependencies to the list ”dependencies down the tree”. 5. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] to abt. 6. DepEngine writes actions 1, 2, 3, 4, 5 to the abt log.
Use Case Steps:	1. DepEngine retrieves a request for dependencies down the tree for package foo1 v. 3.2 with depth 1. 2. DepEngine creates a list ”dependencies down the tree”. 3. DepEngine finds dependencies foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0. 4. DepEngine adds these dependencies to the list ”dependencies down the tree”. 5. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0] to abt. 6. DepEngine writes actions 1, 2, 3, 4, 5 to the abt log.

Use Case Name:	Look for dependencies down the tree
Use Case Steps:	1. DepEngine retrieves a request for dependencies down the tree for package foo3 v. 3.11. 2. DepEngine creates a list ”dependencies down the tree”. 3. DepEngine finds no dependencies down the tree for given package. 4. DepEngine returns [ ] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.

7.2 Scenarios Use Case 2

Use Case Name:	Look for dependencies up the tree
Use Case Steps:	1. DepEngine retrieves a request for dependencies up the tree for package foo1 v. 3.0. 2. DepEngine creates a list ”dependencies up the tree”. 3. DepEngine finds dependencies foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2. 4. DepEngine adds these dependencies to the list ”dependencies up the tree”. 5. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2] to abt. 6. DepEngine writes actions 1, 2, 3, 4, 5 to the abt log.
Use Case Steps:	1. DepEngine retrieves a request for dependencies up the tree for package foo1 v. 3.2 with depth 1. 2. DepEngine creates a list ”dependencies up the tree”. 3. DepEngine finds dependency foo2 v. 3.11 Ro foo4 v. 3.0. 4. DepEngine adds this dependency to the list ”dependencies up the tree”. 5. DepEngine returns [foo2 v. 3.11 Ro foo4 v. 3.0] to abt. 6. DepEngine writes actions 1, 2, 3, 4, 5 to the abt log.

Use Case Name:	**Look for dependencies up the tree**
Use Case Steps:	1. DepEngine retrieves a request for dependencies up the tree for package foo1 v. 3.2 (in this scenario we assume the dependency foo4 v. 3.0 Do foo1 v. 3.2 does not exist). 2. DepEngine creates a list ”dependencies up the tree”. 3. DepEngine finds no dependencies up the tree for given package. 4. DepEngine returns [ ] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.

7.3 Scenarios Use Case 3

Use Case Name:	Generate complete dependency list
Use Case Steps:	1. DepEngine retrieves a request for a complete dependency list, as well as a list dependencies down the tree [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] and a list dependencies up the tree [foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2]. 2. DepEngine creates a list ”complete dependency list”. 3. DepEngine merges [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] and [foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2], resulting in [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2; foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2] 4. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2; foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine retrieves a request for a complete dependency list, as well as a list dependencies down the tree [ ] and a list dependencies up the tree [foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2]. 2. DepEngine creates a list ”complete dependency list”. 3. DepEngine merges [ ] and [foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2], resulting in [foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2] 4. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.

7.4 Scenarios Use Case 4

Use Case Name:	Select needed dependencies
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Install foo1 v. 3.2. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2 to list Needed Dependencies. 4. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo4 v. 3.0 Do foo1 v. 3.2] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Remove foo2 v. 3.11. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1, v. 3.2 Do foo2, v. 3.11 to list Needed Dependencies. 4. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Rebuild foo2 v. 3.11. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1, v. 3.2 Do foo2, v. 3.11 to list Needed Dependencies. 4. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.

Use Case Name:	Select needed dependencies
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Reconfigure foo2 v. 3.11. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1, v. 3.2 Do foo2, v. 3.11 to list Needed Dependencies. 4. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Upgrade foo2 v. 3.11 to foo2 v. 3.2. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1, v. 3.2 Do foo2, v. 3.11; foo2, v. 3.2 Ro foo4 v. 3.0; foo2, v. 3.2 oRo foo3 v. 3.2 to list Needed Dependencies. 4. DepEngine returns [foo1, v. 3.2 Do foo2, v. 3.11; foo2, v. 3.2 Ro foo4 v. 3.0; foo2, v. 3.2 oRo foo3 v. 3.2] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Downgrade foo2 v. 3.11 to foo2 v.3.1. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1, v. 3.2 Do foo2, v. 3.11; foo2, v. 3.1 Ro foo4 v. 2.0; foo2 v. 3.1 oRo foo3 v. 3.1 to list Needed Dependencies. 4. DepEngine returns [foo1, v. 3.2 Do foo2, v. 3.11; foo2, v. 3.1 Ro foo4 v. 2.0; foo2 v. 3.1 oRo foo3 v. 3.1] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.

Use Case Name:	Select needed dependencies
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Remove foo3 v. 3.11. 2. DepEngine creates list Needed Dependencies. 3. DepEngine finds no needed dependencies. 4. DepEngine returns [ ] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.

7.5 Scenarios Use Case 5

Use Case Name:	Sort packages for right build order
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11] from abt and operation Install/ Rebuild/ Reconfigure foo1 v. 3.2. 2. DepEngine creates list Sorted Dependencies. 3. DepEngine applies sorting-algorithm to [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2]. 4. Algorithm returns [foo1 v. 3.2, foo2 v. 3.11, foo4 v. 3.0, foo3 v. 3.11] to DepEngine. 5. DepEngine returns [foo1 v. 3.2, foo2 v. 3.11, foo4 v. 3.0, foo3 v. 3.11] to abt. 6. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Install/ Rebuild/ Reconfigure foo1 v. 3.2. 2. DepEngine creates list Sorted Dependencies. 3. DepEngine applies sorting-algorithm to [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2]. 4. Algorithm returns cycle alert to DepEngine. 5. DepEngine returns ERROR to abt. 6. DepEngine writes actions 1, 2, 3, 4, 5 to the abt log.

Use Case Name:	Sort packages for right build order
Use Case Steps:	1. DepEngine receives list [foo1 v. 4.0 Do foo2 v. 3.3; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.2 Ro foo4 v. 5.0; foo2 v. 3.11 ORo foo3 v 3.11] from abt and operation Upgrade foo1 v. 3.2 to foo1 v. 4.0. 2. DepEngine creates list Sorted Dependencies. 3. DepEngine applies sorting-algorithm to [foo1 v. 4.0 Do foo2 v. 3.3; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.2 Ro foo4 v. 5.0; foo2 v. 3.11 ORo foo3 v 3.11]. 4. Algorithm returns [foo1 v. 4.0, foo2 v. 3.3, foo4 v. 5.0, foo3 v. 3.11] to DepEngine. 5. DepEngine returns [foo1 v. 4.0, foo2 v. 3.3, foo4 v. 5.0, foo3 v. 3.11] to abt. 6. DepEngine writes actions 1, 2, 3, 4, 5 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Remove foo2 v. 3.11. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1, v. 3.2 Do foo2, v. 3.11 to list Needed Dependencies. 4. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Rebuild foo2 v. 3.11. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1, v. 3.2 Do foo2, v. 3.11 to list Needed Dependencies. 4. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.

Use Case Name:	Sort packages for right build order
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Reconfigure foo2 v. 3.11. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1, v. 3.2 Do foo2, v. 3.11 to list Needed Dependencies. 4. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Upgrade foo2 v. 3.11 to foo2 v. 3.2. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1, v. 3.2 Do foo2, v. 3.11; foo2, v. 3.2 Ro foo4 v. 3.0; foo2, v. 3.2 oRo foo3 v. 3.2 to list Needed Dependencies. 4. DepEngine returns [foo1, v. 3.2 Do foo2, v. 3.11; foo2, v. 3.2 Ro foo4 v. 3.0; foo2, v. 3.2 oRo foo3 v. 3.2] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Downgrade foo2 v. 3.11 to foo2 v.3.1. 2. DepEngine creates list Needed Dependencies. 3. DepEngine adds foo1, v. 3.2 Do foo2, v. 3.11; foo2, v. 3.1 Ro foo4 v. 2.0; foo2 v. 3.1 oRo foo3 v. 3.1 to list Needed Dependencies. 4. DepEngine returns [foo1, v. 3.2 Do foo2, v. 3.11; foo2, v. 3.1 Ro foo4 v. 2.0; foo2 v. 3.1 oRo foo3 v. 3.1] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.

Use Case Name:	Select needed dependencies
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Remove foo3 v. 3.11. 2. DepEngine creates list Needed Dependencies. 3. DepEngine finds no needed dependencies. 4. DepEngine returns [ ] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.

7.6 Scenarios Use Case 6

Use Case Name:	Look for dependency version problems
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Upgrade foo2 v. 3.11 to foo2 v. 3.3. 2. DepEngine creates list Version Problems. 3. DepEngine adds foo2 v. 3.3 Ro foo4 v. 4.0 to list Version Problems. 4. DepEngine returns [foo2 v. 3.3 Ro foo4 v. 4.0] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Downgrade foo2 v. 3.3 to foo2 v.3.1. 2. DepEngine creates list Version Problems. 3. DepEngine adds foo1 v. 3.2 Do foo2 v. 3.11 to list Version Problems. 4. DepEngine returns [foo1 v. 3.2 Do foo2 v. 3.11] to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.
Use Case Steps:	1. DepEngine receives list [foo1 v. 3.2 Do foo2 v. 3.11; foo1 v. 3.2 ODo foo3 v. 3.11; foo2 v. 3.11 Ro foo4 v. 3.0; foo2 v. 3.11 ORo foo3 v 3.11; foo4 v. 3.0 Do foo1 v. 3.2] from abt and operation Downgrade foo2 v. 3.3 to foo2 v.3.1. 2. DepEngine creates list Version Problems. 3. DepEngine cannot retrieve version information from foo1 v. 3.2 4. DepEngine returns error message to abt. 5. DepEngine writes actions 1, 2, 3, 4 to the abt log.

7.7 Scenarios Use Case 7

Use Case Name:	**Count dependencies**
Use Case Steps:	1. DepEngine receives a list [foo1 v. 3.2; foo2 v. 3.11] and a depth of 10 from abt. 2. DepEngine counts items with max 10 depth in [foo1 v. 3.2; foo2 v. 3.11]. 3. DepEngine finds 2 dependencies. 4. DepEngine returns the countvalue of 2 to abt. 5. DepEngine writes actions performed in step 1, 2 and 3 to the abt log.
Use Case Steps:	1. DepEngine receives a list [foo3 v. 3.4; foo4 v. 3.45; foo5 v. 4.33] from abt. 2. DepEngine counts each item in [foo3 v. 3.4; foo4 v. 3.45; foo5 v4.33]. 3. DepEngine returns the countvalue of 3 to abt. 4. DepEngine writes actions performed in step 1, 2 and 3 to the abt log.

7.8 Scenarios Use Case 8

Use Case Name:	Look for unremovable dependencies
Use Case Steps:	1. DepEngine receives request for a check for unremovable packages and a list [foo3 v. 3.11]. 2. DepEngine checks the list [foo3 v. 3.11] for required dependencies one level up. 3. DepEngine finds no required dependencies (only ODo and ORo). 4. DepEngine returns list [foo3 v. 3.11] . 5. DepEngine writes action performed in step 1, 2 and 3 to the abt log.
Use Case Steps:	1. DepEngine receives request for a check for unremovable packages. 2. DepEngine checks the list [foo4 v. 3.0] for required dependencies one level up. 3. DepEngine finds dependency foo2 v. 3.11 Ro foo4 v. 3.0. 4. DepEngine returns a report to abt with a warning that foo2 v. 3.11 Relies on foo4 v. 3.0. 5. DepEngine writes action performed in step 1, 2 and 3 to the abt log.

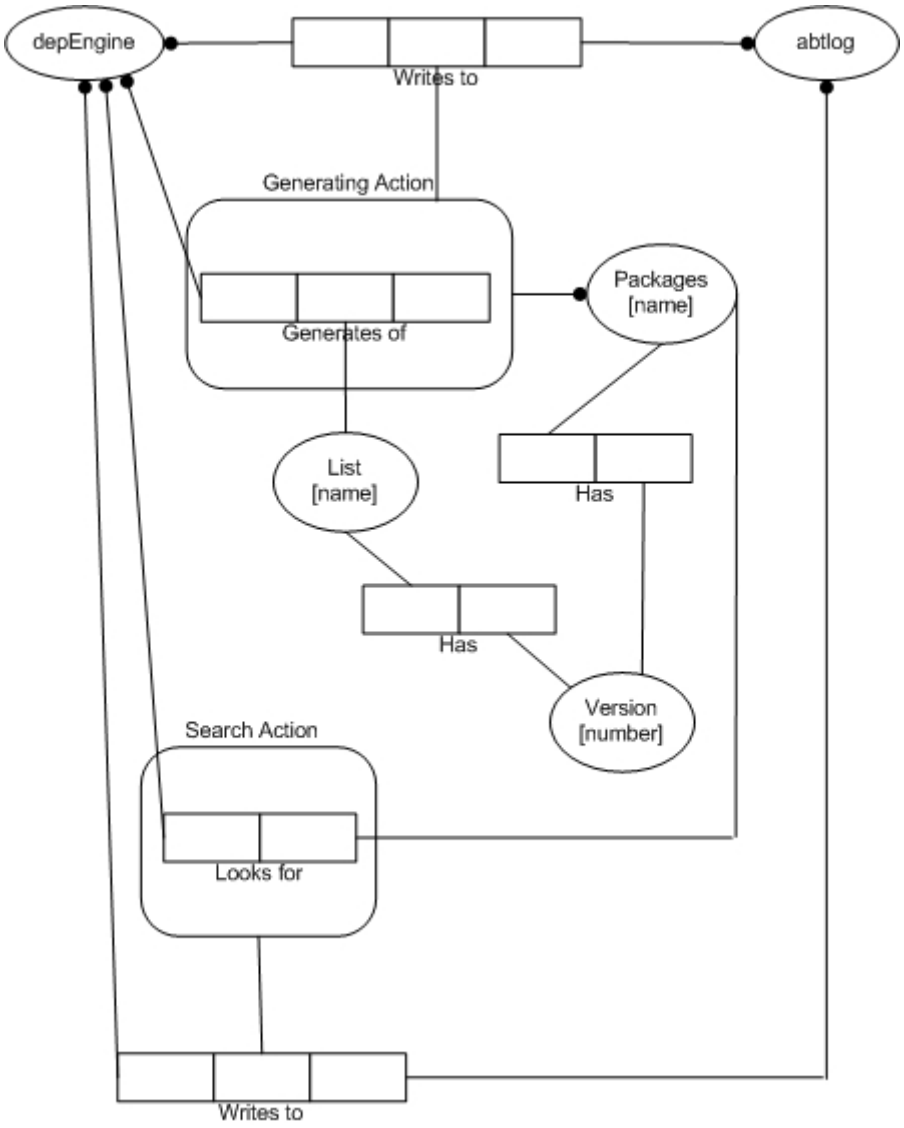
7.9 Scenarios Use Case 9

Use Case Name:	**Look for independent packages**
Use Case Steps:	1. DepEngine receives request for a list of independent packages. 2. DepEngine looks up and down the tree for packages with no dependencies. 3. DepEngine has found independent packages bar1 v. 1.0 and bar2 v. 2.35 (not in example). 4. DepEngine returns [bar1 v. 1.0; bar2 v. 2.35]. 5. DepEngine writes action performed in step 1, 2, 3 and 4 to the abt log.
Use Case Steps:	1. DepEngine receives request for a list of independent packages. 2. DepEngine looks up and down the tree for packages with no dependencies. 3. DepEngine has not found independent packages. 4. DepEngine returns [ ]. 5. DepEngine writes action performed in step 1, 2, 3 and 4 to the abt log.

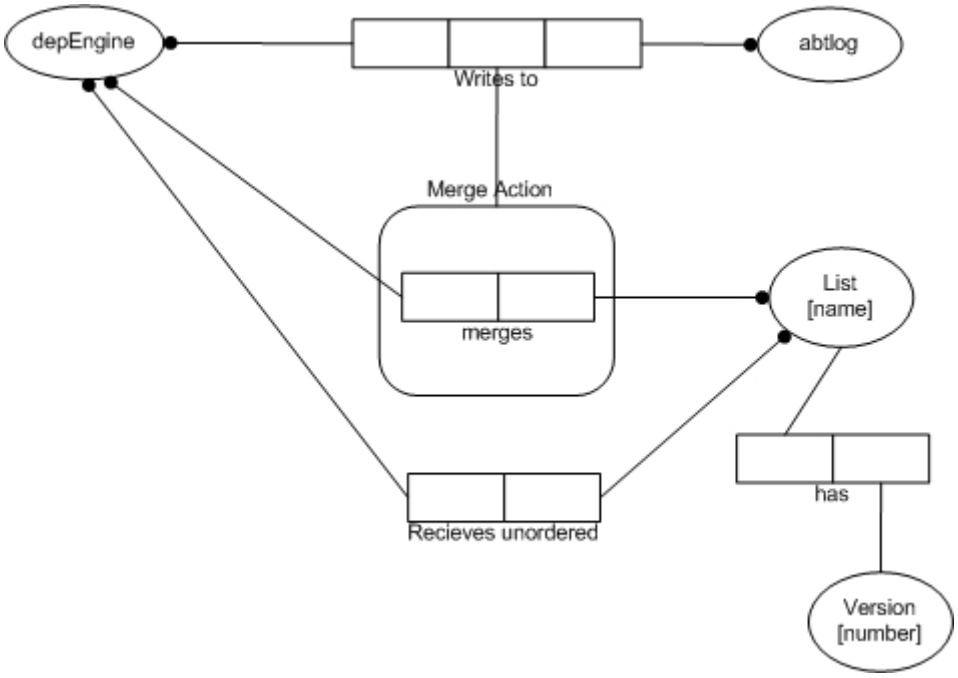
8 Domain Models

Each use case is futher detailed visually in the following sections.

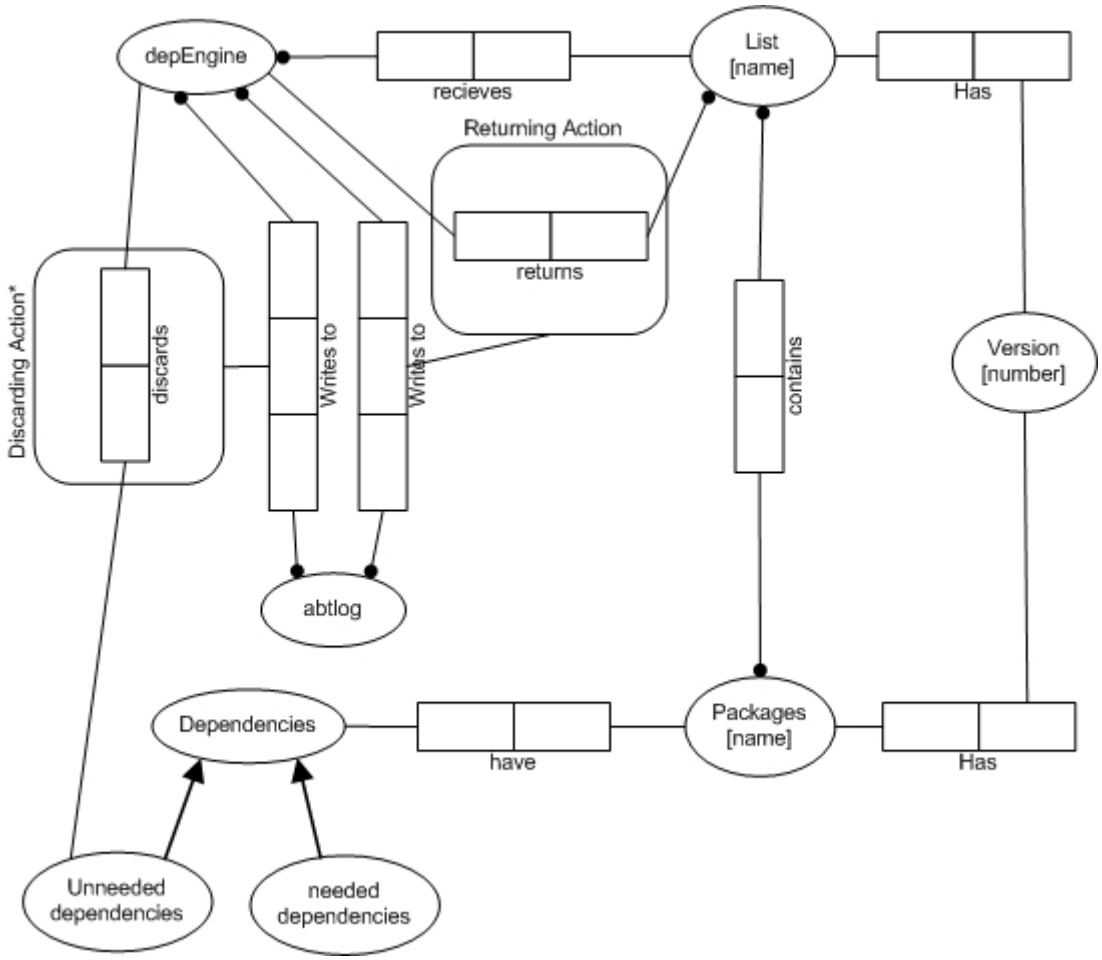
8.1 Domain Model Use Cases 1 & 2



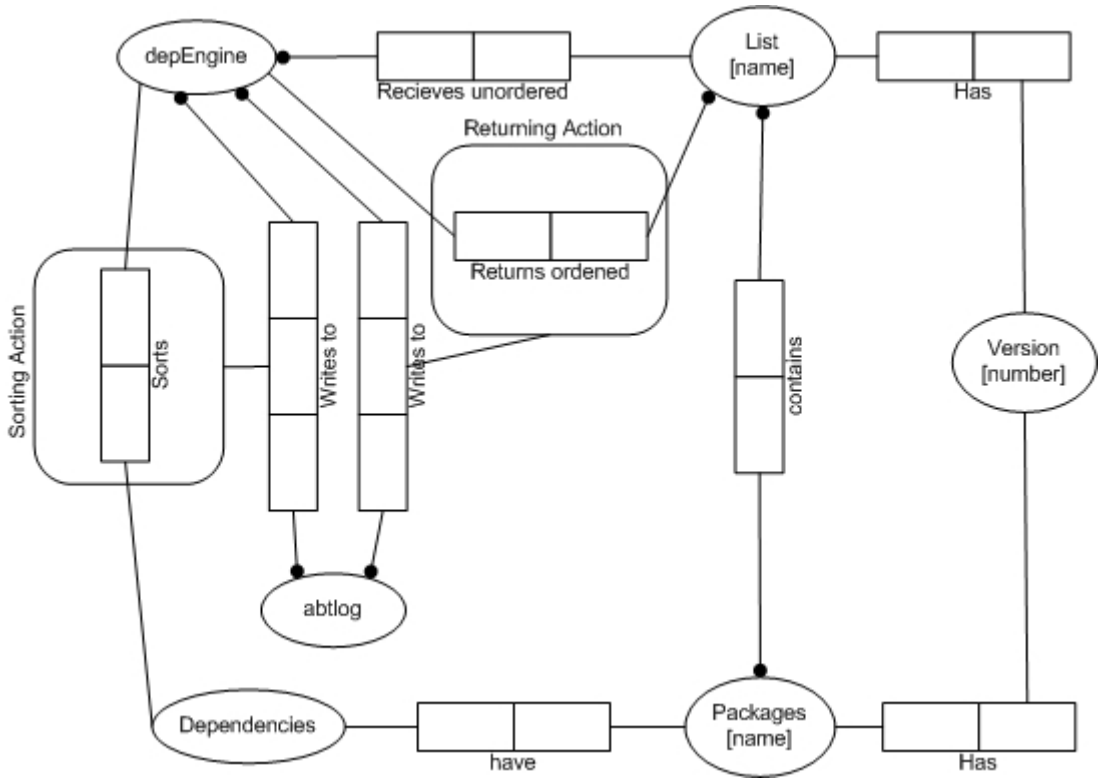
8.2 Domain Model Use Case 3



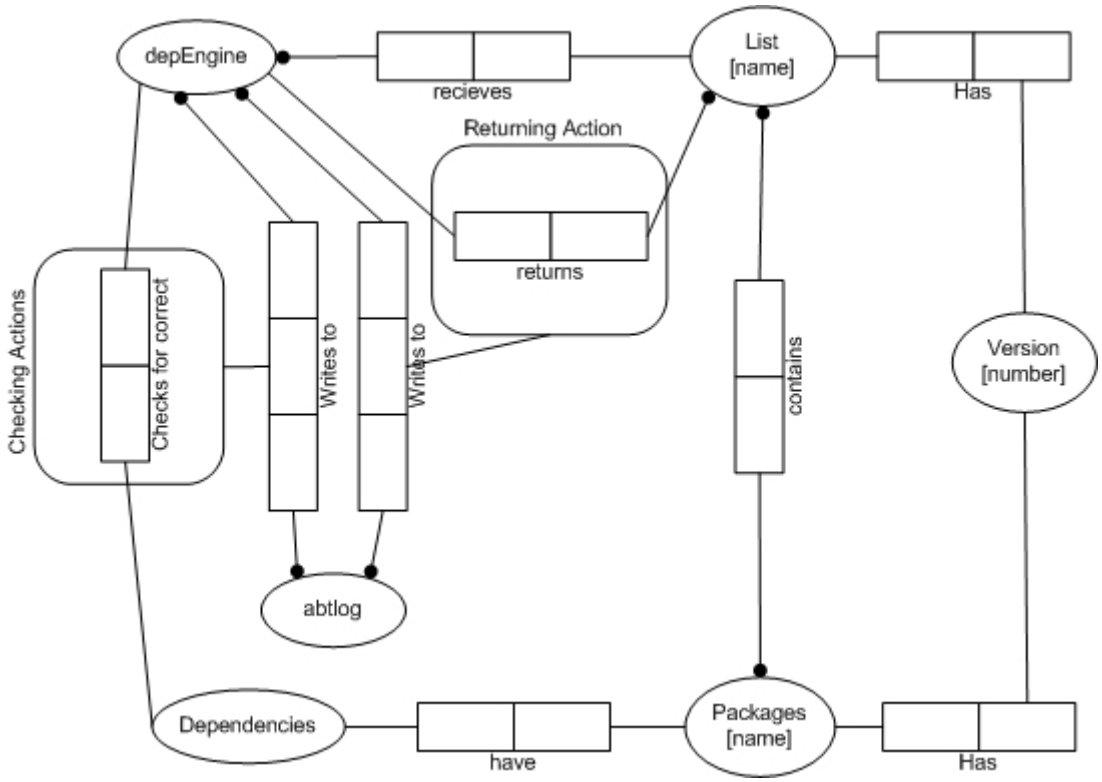
8.3 Domain Model Use Case 4



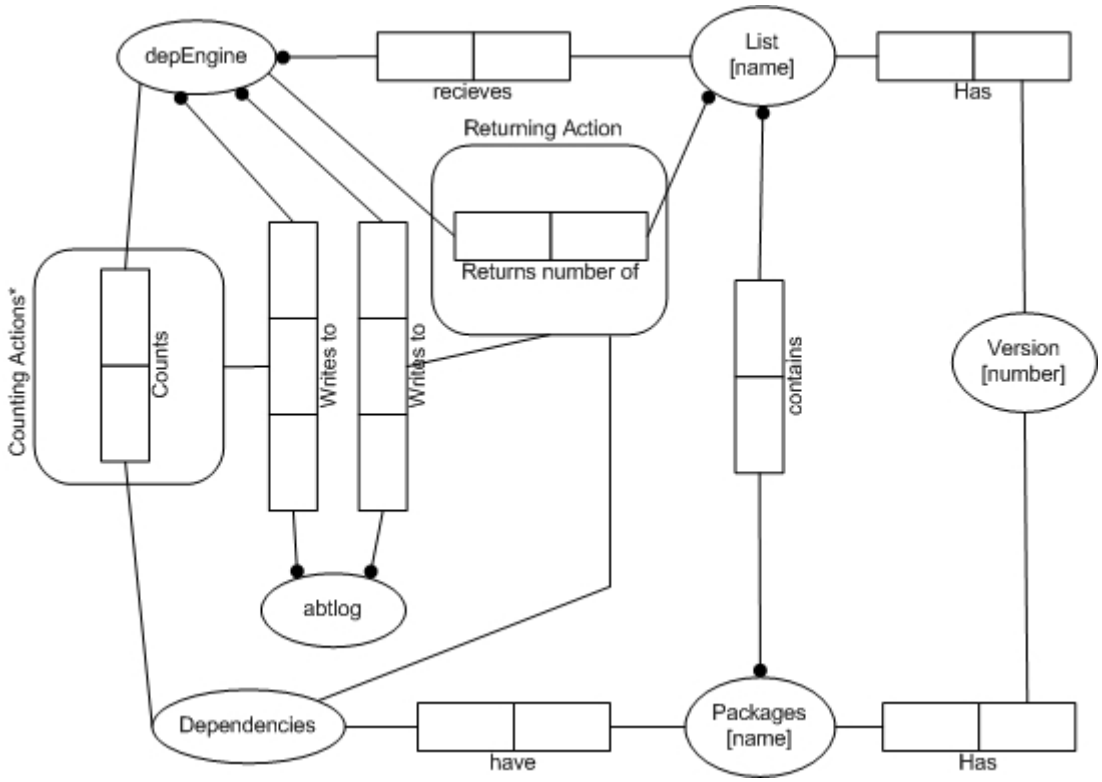
8.4 Domain Model Use Case 5



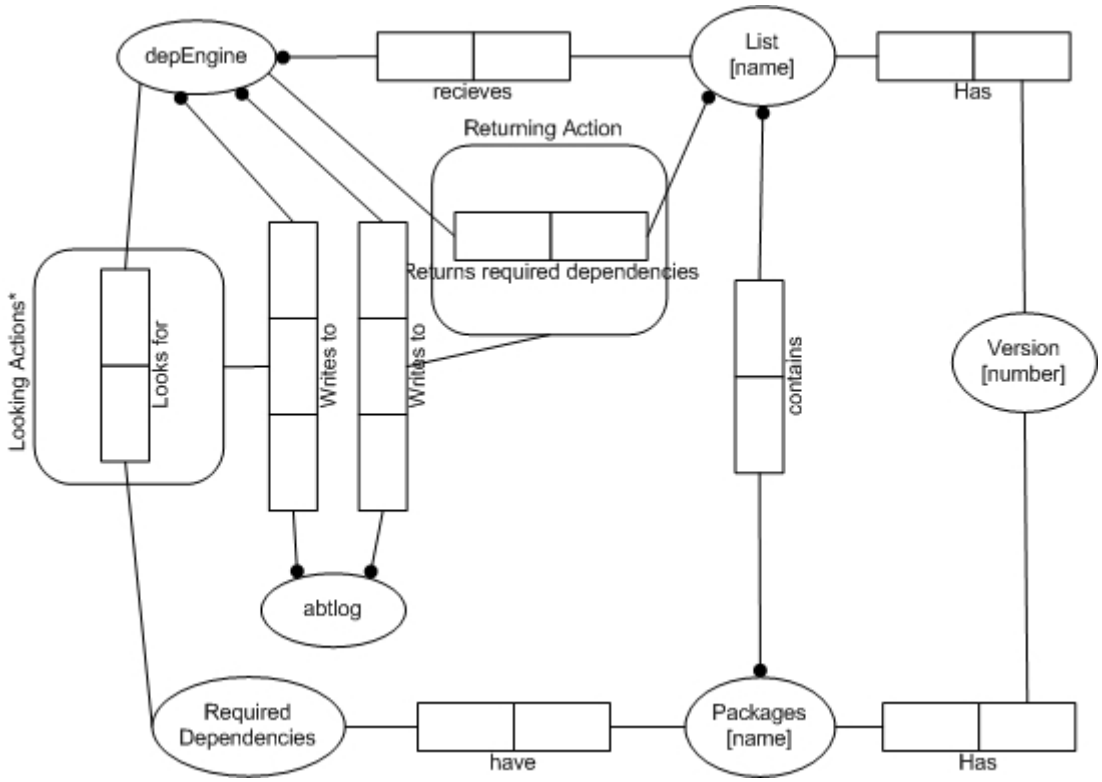
8.5 Domain Model Use Case 6



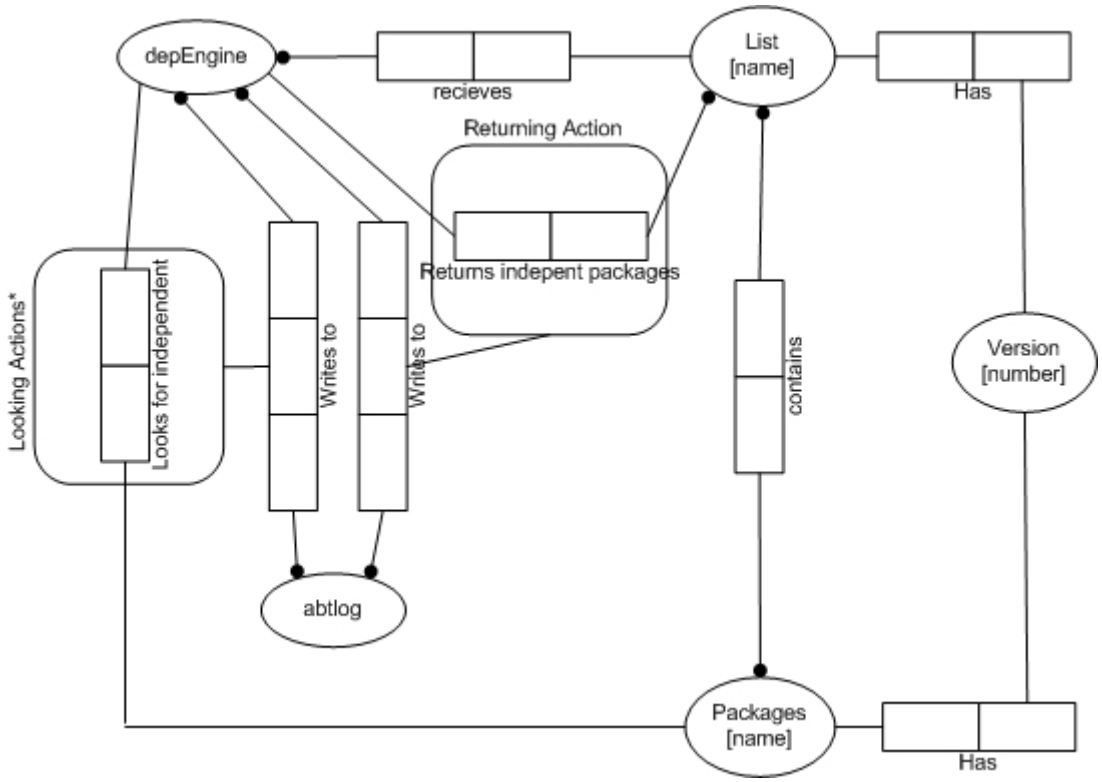
8.6 Domain Model Use Case 7



8.7 Domain Model Use Case 8



8.8 Domain Model Use Case 9



9 Dictionary

Term	Explanation
Software package management system (SPMS, package manager)	A software application that manages all aspects of handling software such as installation, upgrading and uninstalling packages.
Abt	the name of the software package management system in this project.
AbTLinux	The name of the software package management system in this project.
Action	A single task directly performed by the Dependency Engine.
Build list	States the order in which the linked packages must be built.
Build script	Automated build process for certain source or application.
Configure	Change applications configuration.
Cycle	A chain of packages, depending on each other for installation.
Dependency	Link to package that relies or depends on the current package.
Dependency engine (DepEngine)	The part of the SPMS which is responsible handling dependency operations.
Dependency down the tree	Given package depends on another package.
Dependency tree	(Imaginary) representation of all packages and links between them.
Dependency-type	Depends on, Relies on, Optionally Depends on, Optionally Relies on (Do, Ro, ODo, ORo).
Dependency up the tree	Another package depends on given package.
Depends on	If package foo1 depends on package foo2, then package foo1 will be rebuilt any time package foo2 is rebuilt.
Distribution	An end product, something you can install and use, such as the SPMS.
Install	Copy binaries to specified locations.
LaTeX	A document typesetting software package which is platform independent.
Linux	An open-source and originally free operating system based on Unix.
Operation	Command given to AbTLinux by the user.
Package	A complete set of data used in the SPMS to manage all aspects of using this single piece of software.
Package (dependency) description	File listing all dependencies on and from given package.
Rebuild	Recompile and link source.
Relies on	If package foo1 depends on package foo2, then package foo1 will be rebuilt any time package foo2 is reconfigured.
Requirement	A characteristic of a system in order to be acceptable to the acquirer.

Term	Explanation
Source	Program code of package.
Unix	A computer operating system originally developed in the 1960s and 1970s.

10 Appendix - algorithms

10.1 Tree-algorithm

1. Given a dependency A Do B (or A Ro B etc.):
2. Check if the list already contains this dependency;
3. If so, continue with next dependency (because it is redundant); if not:
4. Add A Do B to the list and continue with next dependency;

10.2 Sorting algorithm

1. Given a required dependency A Do/Ro B:
2. Check if list contains B followed by A, e.g. [X,Y,B,C,D,A];
3. If so, continue with next required dependency (because it is redundant); if not:
4. Check if list contains A followed by B, e.g. [X,Y,A,C,D,B];
   (a) If so, check if alternatives are available.
      i. If so return to that alternative;
      ii. If not generate cycle alert;
   (b) If not, proceed as follows:
5. If A is not in the list, add B,A in front of the list, e.g. [X,Y,Z] → [B,A,X,Y,Z];
6. If A is already on the list, add B in front of it, e.g. [X,A,Y,Z] → [X,B,A,Y,Z];
7. List alternatives (alternative places for B), in this example only one (in front of X);
8. Continue with next required dependency.
9. Sort optional dependencies (ODo, ORo).

References

AbTLinux (2005). *ABout Time Linux*. AbTLinux, `http://abtlinux.org/page.php?1`. last checked: Dec-2005.