ABout Time Linux - dependency engine (v1.0)

TEAM TRAILBLAZER (GROUP 2)

Y.H.C. Janse
y.janse@student.ru.nl

G.J.N.M. Chorus
g.chorus@student.ru.nl

C.J.P. Nellen
c.j.p.nellen@student.ru.nl

J. Böhmer
j.bohmer@student.ru.nl

Eric D. Schabell
erics@abtlinux.org

December 15, 2005

Contents

1 Introduction

ABout Time Linux (AbTLinux) is a software development project that plans to provide a source-based Linux distribution. The resulting toolset is a software package management system capable of providing the needed infrastructure for installing software packages and maintaining the target systems integrity. The system being developed in this project is generally known as *abt package manager*.

An integral part of the abt package manager is a section for handling dependency issues among software packages. These issues arise when changes in packages affect other related packages, which then lead to problems in software compilation. These can be caused by installing, removing, rebuilding, et cetera. Events like these must be dealt with by a dependency engine. The part of the abt package manager that takes care of these problems is referred to as the *dependency engine* and it is the main focus of this document.

This document covers the requirement specification for the dependency engine. The format used in this document is described in Use Cases: Requirements in Context (Kulak and Guiney, 2003). The first six chapters detail the project scope by defining the exact problem-statement, stakeholders and their view, project risks and statement of work. Chapters seven through nine describe the requirements in the form of use cases, both separately and in relation to each other. The last chapters focus on business rules, non-functional requirements and terminological definitions.

The authors of this specification are students from the Radboud University of Nijmegen, The Netherlands. These students have been invited by the AbTLinux project leader to put their newly acquired skills in requirements engineering to the test. This team of students is known to the AbTLinux developers as Team Trailblazer.

2 Problem Statement

Most AbTLinux developers have worked on Linux distributions in the past and have grown tired of working on badly documented designs. Their experiences showed that this leads to badly organized growth paths for distributions and hard to manage and maintain software modules.

All AbTLinux developers agree that this is not the right way to work. A project should have clear goals, consistent documentation and robust coding practices. This is the main reason for the existence of AbTLinux. The AbTLinux developers feel it is about time to *do it the right way*.

One of the first steps in doing it the right way involves gathering requirements. For AbTLinux these needs and wishes are for a great deal already present in the minds of the developers. But for AbTLinux to be successful these requirements need to be extrapolated and documented in a defined notation. This makes sure that all current *and future* developers understand each other and pursue the same goals.

The most important requirements for the abt package manager project as a whole have already been gathered. These are specified in design document (Eric D. Schabell, 2005). Now, the requirements for the dependency engine project are next.

3 Stakeholder Analysis

The dependency engine project stakeholders are:

Eric D. Schabell
Eric is the founder of AbTLinux and the project leader of abt package manager project and wrote the first requirements specification for the package manager in which the dependency engine plays an important role. Eric has experienced bad project management himself when working in the SourceMage/GNU Linux project (started in 2001) as a developer. He is affiliated with the Radboud University of Nijmegen as a scientific programmer in the Information and Knowledge Systems project group (IRIS).

Bas van Gils
Bas is a PhD student at Radboud University of Nijmegen in the Information and Knowledge Systems project group (IRIS). His strong preference of free software led to using a lot of different Linux distributions. He used SourceMage GNU/Linux for a long time but was not satisfied. Bas is also working as a developer for AbTLinux.

Linux Community
The Linux Community in general is also a stakeholder in this project. Research done on possible solutions when coping with package dependency issues is released under the FDL license, which makes it available to the public domain. Ease of use in working with the abt Package Manager and the dependency engine is important for the Linux community as the final end-users.

4 Mission, Vision and Values

Mission
Improper design and documentation of software projects lead to prolonged project life, miscommunication, software errors and bad maintenance. These things cause a lot of frustration and should be prevented.

Vision
Their vision is to provide a source-based Linux distribution based on a toolset that has been documented from the beginning of the design cycle. The AbTLinux team want to do this the right way by applying often omitted steps for maximizing project and quality control.

Values
They value clear goals, clear documentation and clear coding guidelines. The source-code in the end-product will be released under the GNU General Public License (GPL); all documentation is licensed under the GNU Free Documentation License (FDL). Whenever the developers are presented with a problem, the Keep It Simple Silly (KISS) principle is applied. The AbTLinux priorities are their users and free software.

5 Statement of Work

Scope	Gathering and documenting requirements for the dependency engine. This excludes implementation aspects.
Objectives	Project objectives: • Finding out what the users need. • Documenting users needs. • Avoiding design assumptions. • Resolving conflicting requirements. • Eliminating redundant requirements. • Reducing overwhelming volume. • Ensuring requirements traceability.
Application Overview	The applications involved in this project: • *abt package manager* - Asks the dependency engine to resolve dependency issues for specific/certain packages. • *dependency issues* - Provides means to solve dependency issues between packages.
User Demography	The user involved in this project: • *abt package manager* - The tool for managing packages on AbTLinux. Uses the dependency engine to resolve dependency issues and provide methodes to deal with dependency breakage. • *maintenance programmer* - The programmer who performs maintenance on the system by fixing or extending it. According to Eric D. Schabell there are no further actors in the system; the dependency engine is only accessed from the abt package manager.

Constraints	The constraints involved in this project: • The requirements specification of *abt package manager* is not yet final (v0.5) and therefore subject to change. • All documentation must be typeset in $\LaTeX\,2_{\varepsilon}$. • Communication with stakeholders must be in English.
Assumptions	The requirement specifications of the *abt package manager* project as a whole don't change drastically regarding the *dependency engine*.
Staffing and Costs	Yves Janse $12 \times 6 = 72$ Guido Chorus $12 \times 6 = 72$ Chris Nellen $12 \times 6 = 72$ John Böhmer $12 \times 6 = 72$ Total 288 hours
Expected Duration	12 weeks. Final deadline 15th december.

Deliverable Outlines:

Façade milestone	**Filled milestone**	**Focused milestone**
October 13	*November 17*	*December 15*
Introduction	Introduction	Introduction
Problem statement	Problem statement	Problem statement
Stakeholders list	Stakeholders list	Stakeholders list
Mission Vision Values	Mission Vision Values	Mission Vision Values
Statement of Work	Statement of Work	Statement of Work
Risk analysis	Risk analysis	Risk analysis
Use Case Survey	Use Case Survey	Use Case Survey
Use Case Diagram	Use Case Diagram	Use Case Diagram
	Use Cases (filled)	Use Cases (focused)
	Several scenarios	Scenarios
	Domain models (sketch)	Domain models
Business rule catalogue	Business rule catalogue	Business rule catalogue
Non-functionals	Non-functionals	Non-functionals
Terminological definitions	Terminological definitions	Terminological definitions

6 Risk Analysis

No.:	#01
Category:	Project team
Risk:	**Project member which is sick for a long time or quits, reduces productivity with 25%.**
Resolution needed by:	5 days after it occurs.
Status:	Unresolved.
Days lost if it occurs:	9 days.
Likelihood it will happen:	$\pm$ 10%.
Risk rating:	0.9

No.:	#02
Category:	Universe of Discourse
Risk:	**Defining conflicting requirements might lead to unnecessary scope-growth.**
Resolution needed by:	10 days after it occurs.
Status:	Unresolved.
Days lost if it occurs:	10 days.
Likelihood it will happen:	$\pm$ 20%.
Risk rating:	2.0

No.:	#03
Category:	Communication
Risk:	**Little or slow feedback from stakeholders might delay the project.**
Resolution needed by:	2 days after contact has been initiated.
Status:	Unresolved.
Days lost if it occurs:	2 days.
Likelihood it will happen:	$\pm$ 40%.
Risk rating:	0.8

To deal with these risks we add the risk ratings above to a total of 3.7 and therefore try to reserve approximately 4 days before the deadline. These days can be used if one or more of the risks should actually occur giving us still the chance to make the iteration deadline.

7 Use Case Survey

The Use Case Survey shows the basic essential interactions which the dependency engine must support. It also shows the state which the Use Cases are in.

Use Case number:	UC#01
Use Case name:	**Give upward dependency tree for a given package.**
Initiating actor:	abt package manager, dependency engine.
Description:	Give dependency tree for a given package with optional depth n. The dependency tree exists of packages which are dependent on the given package.
Completeness:	Focused.
Maturity:	Adult.
Dependency:	UC06.
Source:	Requirements document v0.5 and AbTLinux.org forums.

Use Case number:	UC#02
Use Case name:	**Give downward dependency tree for a given package.**
Initiating actor:	abt package manager, dependency engine.
Description:	Give dependency tree for a given package with optional depth m. The dependency tree exists of packages on which the given package depends.
Completeness:	Focused.
Maturity:	Adult.
Dependency:	UC06.
Source:	Requirements document v0.5 and AbTLinux.org forums.

Use Case number:	UC#03
Use Case name:	**Calculate the number of packages dependent on a given package.**
Initiating actor:	abt package manager
Description:	abt package manager requests how many packages are dependent on the given package. The system analyses the dependencies of the given package and calculates the number of dependent packages. The system then returns the count.
Completeness:	Focused.
Maturity:	Adult.
Dependency:	UC01, UC06.
Source:	Requirements document v0.5 and AbTLinux.org forums.

Use Case number:	UC#04
Use Case name:	**Calculate the number of packages a given package depends on.**
Initiating actor:	abt package manager
Description:	abt package manager requests how many packages are needed for the given package. The system analyses the dependencies of the given package and calculates the number of packages. The system then returns the count.
Completeness:	Focused.
Maturity:	Adult.
Dependency:	UC02, UC06.
Source:	Requirements document v0.5 and AbTLinux.org forums.

Use Case number:	UC#05
Use Case name:	**Give list of isolated packages.**
Initiating actor:	abt package manager
Description:	The abt package manager requests a list of packages wich have no dependency relationships.
Completeness:	Focused.
Maturity:	Adult.
Dependency:	UC03, UC04, UC06.
Source:	Requirements document v0.5 and AbTLinux.org forums.

Use Case number:	UC#06
Use Case name:	**Register all system actions.**
Initiating actor:	dependency engine
Description:	The dependency engine requests to register a given action.
Completeness:	Focused.
Maturity:	Adult.
Dependency:	None.
Source:	Requirements document v0.5 and AbTLinux.org forums.

Use Case number:	UC#07
Use Case name:	**Create internal dependency tree.**
Initiating actor:	dependency engine
Description:	The system creates a complete (1 or more) internal dependency trees for all installed packages.
Completeness:	Focused.
Maturity:	Adult.
Dependency:	UC06.
Source:	Requirements document v0.5 and AbTLinux.org forums.

Use Case number:	UC#08
Use Case name:	**Give sorted buildlist for a given unsorted buildlist.**
Initiating actor:	abt package manager
Description:	The abt package manager request a sorted buildlist for a given unsorted buildlist. The system sorts the list using the ammount of dependencies per package. It then returns the sorted buildlist.
Completeness:	Focused.
Maturity:	Adult.
Dependency:	UC4, UC06.
Source:	Requirements document v0.5 and AbTLinux.org forums.

8 Use Cases

8.1 Textual Use Cases

Use Case Nr:	**UC01**
Use Case Name:	**Give upward dependency tree for a given package.**
Authors:	Team Trailblazer
Dates:	December 14, 2005
Iteration:	Focused
Description:	Give dependency tree for a given package with optional depth n. The dependency tree exists of packages which are dependent on the given package.
Actors:	abt package manager, dependency engine.
Preconditions:	Internal dependency tree is populated and up-to-date and available.
Triggers:	abt package manager, UC03.
Basic Course of Events:	1. The system receives a request to give the dependency tree of a given package. 2. The system selects the given package in the internal dependency tree. 3. The system moves a maximum n away from the desired node using the internal dependency tree. If n is not given the default value $n = 10$ will be used. 4. The system marks and registers the RO, DO, oRO and oDO packages in a tree subset. 5. The system returns the tree subset to the initiating actor. 6. The system triggers UC06.
Alternative paths:	• Depth n cannot be reached while moving through a dependency tree. The system stops at the last dependency in the current path and continues with the next path using backtracking.
Exception paths:	• The given package is not found. The system triggers UC06 and reports back to the initiating actor.
Assumptions:	None.
Postconditions:	A tree subset with a depth n is available to the initiating actor.
Related business rules:	BR2, BR3.

Use Case Nr:	**UC02**
Use Case Name:	**Give downward dependency tree for a given package.**
Authors:	Team Trailblazer
Dates:	December 14, 2005
Iteration:	Focused
Description:	Give dependency tree for a given package with optional depth m. The dependency tree exists of packages on which the given package depends.
Actors:	abt package manager, dependency engine.
Preconditions:	Internal dependency tree is populated and up-to-date and available.
Triggers:	abt package manager, UC04.
Basic Course of Events:	1. The system receives a request to give the dependency tree of a given package. 2. The system selects the given package in the internal dependency tree. 3. The system moves a maximum m away from the desired node using the internal dependency tree. If m is not given the default value $m = 10$ will be used. 4. The system marks and registers the RO, DO, oRO and oDO packages in a tree subset. 5. The system returns the tree subset to the initiating actor. 6. The system triggers UC06.
Alternative paths:	• Depth m cannot be reached while moving through a dependency tree. The system stops at the last dependency in the current path and continues with the next path using backtracking.
Exception paths:	• The given package is not found. The system triggers UC06 and reports back to the initiating actor.
Assumptions:	None.
Postconditions:	A tree subset a depth m is available to the initiating actor.
Related business rules:	BR2, BR3.

Use Case Nr:	**UC03**
Use Case Name:	**Calculate the number of packages dependent on a given package.**
Authors:	Team Trailblazer
Dates:	December 14, 2005
Iteration:	Focused
Description:	abt package manager requests how many packages are dependent on the given package. The system analyses the dependencies of the given package and calculates the number of dependent packages. The system then returns the count.
Actors:	abt package manager
Preconditions:	Internal dependency tree is populated and up-to-date and available.
Triggers:	UC05, abt package manager.
Basic Course of Events:	1. The system receives a request for calculating the number of packages which depends on a given package. 2. The system triggers UC01 with $n = 1$. 3. The system counts the number of packages in the registered dependency tree. 4. The system returns the number of packages to the initiating actor, both RO and DO. 5. The system triggers UC06.
Alternative paths:	None.
Exception paths:	• UC01 returns an error. The system reports the error back to the initiating actor.
Assumptions:	None.
Postconditions:	The count of packages dependent on a given package is available to the initiating actor.
Related business rules:	BR3.

Use Case Nr:	**UC04**
Use Case Name:	**Calculate the number of packages a given package depends on.**
Authors:	Team Trailblazer
Dates:	December 14, 2005
Iteration:	Focused
Description:	abt package manager requests how many packages are needed for the given package. The system analyses the dependencies of the given package and calculates the number of packages. The system then returns the count.
Actors:	abt package manager
Preconditions:	Internal dependency tree is populated and up-to-date and available.
Triggers:	UC05, UC08, abt package manager
Basic Course of Events:	1. The system receives a request for calculating the number of packages a given package depends on. 2. The system triggers UC02 with $m = 1$. 3. The system counts the number of packages in the registered dependency tree. 4. The system returns the number of packages to the initiating actor, both RO and DO. 5. The system triggers UC06.
Alternative paths:	None.
Exception paths:	• UC02 returns an error. The system reports the error back to the initiating actor.
Assumptions:	None.
Postconditions:	The count of packages dependent on a given package is available to the initiating actor.
Related business rules:	BR3.

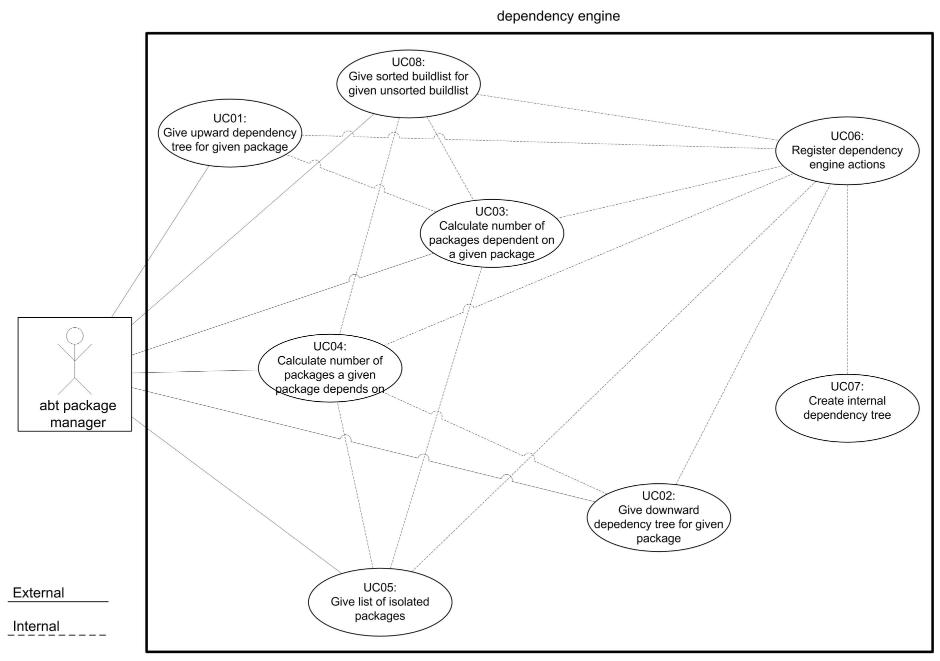
Use Case Nr:	**UC05**
Use Case Name:	**Give list of isolated packages.**
Authors:	Team Trailblazer
Dates:	December 14, 2005
Iteration:	Focused
Description:	The abt package manager requests a list of packages wich have no dependency relationships.
Actors:	abt package manager
Preconditions:	The 'installed packages list' is available and populated.
Triggers:	abt package manager
Basic Course of Events:	1. The system receives a request for a list of all packages with no dependency tree. 2. The system requests the 'installed packages list' from the abt package manager. 3. For each package, the system: • Triggers UC03 and register the returned count. • Triggers UC04 and register the returned count. • If both counts are zero, the system registers the package in the list to be returned. 4. The system returns the list with registered packages. 5. The system triggers UC06.
Alternative paths:	None.
Exception paths:	None.
Assumptions:	None.
Postconditions:	An unsorted list of packages with no dependencies is available to the initiating actor.
Related business rules:	BR2, BR3, BR4.

Use Case Nr:	**UC06**
Use Case Name:	**Register all system actions.**
Authors:	Team Trailblazer
Dates:	December 14, 2005
Iteration:	Focused
Description:	The dependency engine requests to register a given action. The action of registering an action is itself not resistered
Actors:	dependency engine
Preconditions:	
Triggers:	UC01, UC02, UC03, UC04, UC05, UC07, UC08.
Basic Course of Events:	1. The system receives a request to register an action. 2. The system opens the datastore. 3. The system writes a description of the action along with the current date and time in the datastore. 4. The system closes the datastore.
Alternative paths:	None.
Exception paths:	• The datastore is unavailable. The system reports this critical error to the abt package manager itself.
Assumptions:	None.
Postconditions:	The action is registered in the datastore.
Related business rules:	BR1, BR2, BR3.

Use Case Nr:	**UC07**
Use Case Name:	**Create internal dependency tree.**
Authors:	Team Trailblazer
Dates:	December 14, 2005
Iteration:	Focused
Description:	The system creates a complete (1 or more) internal dependency trees for all installed packages.
Actors:	dependency engine
Preconditions:	Installed package list available with all headers included.
Triggers:	dependency engine
Basic Course of Events:	1. The system receives a request to create the internal dependency tree. 2. The system reads the installed package list. 3. For each not marked package, the system: Marks packages that have been added to avoid cycling. Uses backtracking to cover all packages. When reaching a dead end, the tree or this package is finished. Continue from chronological next which is not yet marked. 4. The system triggers UC06.
Alternative paths:	• The system detects two installed packages with different versions. The system then reports back to the initiating actor. Finally the system triggers UC06.
Exception paths:	• A required package is not available in the 'installed package list'. The system cannot continue with tree building and reports back to the initiating actor. Finally the system triggers UC06.
Assumptions:	None.
Postconditions:	An internal dependency tree has been created containing all packages and their dependencies, based on a set of installed packages.
Related business rules:	BR3, BR4.

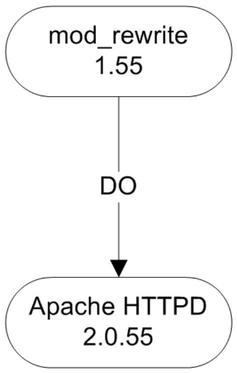
Use Case Nr:	**UC08**
Use Case Name:	**Give sorted buildlist for a given unsorted buildlist.**
Authors:	Team Trailblazer
Dates:	December 14, 2005
Iteration:	Focused
Description:	The abt package manager request a sorted buildlist for a given unsorted buildlist. The system sorts the list using the amount of dependencies per package. It then returns the sorted buildlist.
Actors:	abt package manager
Preconditions:	All packages in the given unsorted build list exist in the internal dependency tree. The given buildlist contain all required and related packages.
Triggers:	abt package manager
Basic Course of Events:	1. The system receives a request to sort a build list for a given unsorted build list. 2. For each package in the given unsorted build list, the system triggers UC04 and registers the count. 3. The system orders the unsorted buildlist by number of upward dependencies per package, starting with package containing highest sum. 4. For each package, the system sequentially: Mark package and add the package to the sorted build list. Recursively add all upward dependencies for the package. Mark and add them to the sorted build list. 5. The system triggers UC06.
Alternative paths:	None.
Exception paths:	None.
Assumptions:	None.
Postconditions:	A sorted build list is available to the initiating actor.
Related business rules:	BR2, BR3

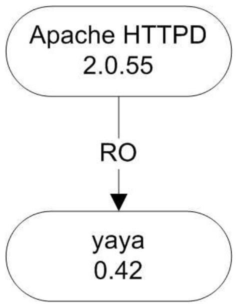
8.2 Use Case Diagram


dependency engine
UC08: Give sorted buildlist for given unsorted buildlist
UC01: Give upward dependency tree for given package
UC06: Register dependency engine actions
UC03: Calculate number of packages dependent on a given package
UC04: Calculate number of packages a given package depends on
UC07: Create internal dependency tree
UC02: Give downward depedency tree for given package
UC05: Give list of isolated packages
abt package manager
External
Internal

9 Scenarios

These scenarios for Use Cases are based on the following packages Apache-httpd-2.0.55, mod_rewrite-1.15, OpenSSL-0.7.9g and yaya-0.42; the package named pack-0.42 is a package that is not present in the system.

Use Case Name: UC01	**Scenario**
Scenario S1:	1. The abt package manager asks the dependency engine for the upward dependency tree of Apache-httpd-2.0.55 with $n = 1$. 2. The dependency engine selects the package Apache-httpd-2.0.55 in the internal dependency tree. 3. The dependency engine move one step up in the internal dependency tree to package mod_rewrite-1.55. 4. The dependency engine marks and registers the package mod_rewrite-1.55 in a tree subset. 5. The dependency engine provides abt package manager with the dependency tree as followed:  6. The dependency engine requests the action to be logged.
Scenario S2:	1. The abt package manager asks the dependency engine for the upward dependency tree of pack-0.42 with $n = 1$. 2. The dependency engine provides abt package manager with the error message: "The package does not exist." 3. The dependency engine requests the action to be logged.

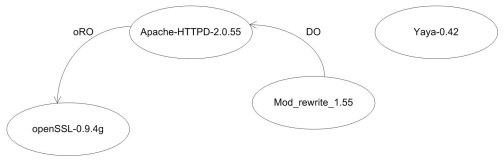
Use Case Name: UC02	Scenario
Scenario S3:	1. The abt package manager asks the dependency engine for the downward dependency tree of Apache-httpd-2.0.55 with $n = 1$. 2. The dependency engine selects the package Apache-httpd-2.0.55 in the internal dependency tree. 3. The dependency engine move one step down in the internal dependency tree to package yaya-0.42. 4. The dependency engine marks and registers the package yaya-0.42 in a tree subset. 5. The dependency engine provides abt package manager with the dependency tree as follows:  6. The dependency engine requests the action to be logged.
Scenario S4:	1. The abt package manager asks the dependency engine for the downward dependency tree of pack-0.42 with $n = 1$. 2. The dependency engine provides abt package manager with the error message: "The package does not exist." 3. The dependency engine requests the failure to be logged.

Use Case Name: UC03	Scenario
Scenario S5:	1. The dependency engine receives a request for calculating the number of packages dependent on package Apache-httpd-2.0.55. 2. The dependency engine triggers UC01 with $n = 1$ for package Apache-httpd-2.0.55. 3. The dependency engine counts the number of packages in the returned dependency tree, which is 1. 4. The dependency engine returns the number 1 to the abt package manager. 5. The dependency engine requests the action to be logged.
Scenario S6:	1. The dependency engine receives a request to calculate the number of packages dependent on package yaya-0.42. 2. The dependency engine provides abt package manager with the number 0. 3. The dependency engine requests the action to be logged.
Scenario S7:	1. The dependency engine receives a request to calculate the number of packages dependent on package pack-0.42. 2. The dependency engine provides abt package manager with the error message: "The package does not exist." 3. The dependency engine requests the failure to be logged.

Use Case Name: UC04	**Scenario**
Scenario S8:	1. The dependency engine receives a request for calculating the number of packages package Apache-httpd-2.0.55 depends on. 2. The dependency engine triggers UC02 with $n = 1$ for package Apache-httpd-2.0.55. 3. The dependency engine counts the number of packages in the returned dependency tree, which is 1. 4. The dependency engine returns the number 1 to the abt package manager. 5. The dependency engine requests the action to be logged.
Scenario S9:	1. The dependency engine receives a request to calculate the number of packages package yaya-0.42 depends on. 2. The dependency engine provides the abt package manager with the number 0. 3. The dependency engine requests the action to be logged.
Scenario S10:	1. The dependency engine receives a request to calculate the number of packages package pack-0.42 depends on. 2. The dependency engine provides abt package manager with the error message: “The package does not exist.” 3. The dependency engine requests the failure to be logged.

Use Case Name: UC05	**Scenario**
Scenario S11:	1. The dependency engine receives a request to provide a list with isolated packages. 2. The dependency engine requests the installed packages-list from the abt package manager. 3. The dependency engine moves through all packages in the list and finds for example yaya-0.42: (a) Counts the number of upward dependencies (UC03) for yaya-0.42 which is 0. (b) Counts the number of downward dependencies (UC04) for yaya-0.42 which is 0. (c) Both are zero so the system registers the isolated package yaya-0.42 in a temporary list. 4. The dependency engine provides the abt package manager with the list containing the package yaya-0.42. 5. The dependency engine requests the action to be logged.

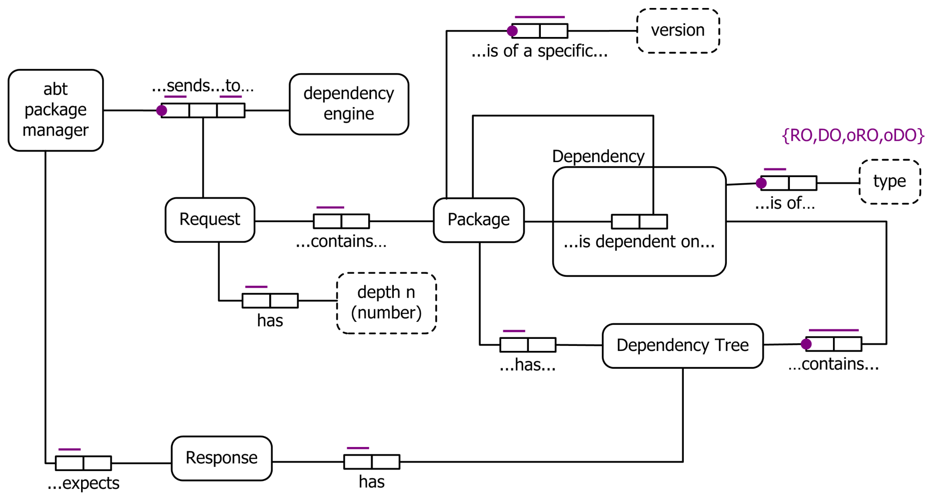
Use Case Name: UC06	Scenario
Scenario S12:	1. The dependency engine receives a request to register the message "Dependency tree generated successfully for package Apache-httpd-2.0.55 with downward depth 2 on March 11, 2005 17:00". 2. The dependency engine opens the datastore "/var/log/depengine" for writing. 3. The dependency engine writes the message above in the datastore. 4. The dependency engine close the datastore. 5. The dependency engine requests the action to be logged.
Scenario S13:	1. The dependency engine receives a request to register the message "Dependency tree generated successfully for package Apache-httpd-2.0.55 with downward depth 2 on March 11, 2005 17:00". 2. The dependency engine fails at opening the datastore "/dev/something" 3. The dependency engine provides the abt package manager with the following error message: "Datastore cannot be opened for writing.".

Use Case Name: UC07	**Scenario**
Scenario S14:	1. The dependency engine requests itself to create the internal dependency tree. 2. The dependency engine requests the installed packages-list from the abt package manager. 3. The dependency engine searches for each package in the installed package list the headers for the dependencies and adds them to the tree. The dependency engine finds for mod-rewrite-1.15 an oDO dependency on Apache-httpd-2.0.55 and for Apache-httpd-2.0.55 a DO dependency on OpenSSL-049g. 4. The following dependency tree is now available for internal use:  oRO Apache-HTTPD-2.0.55 DO Yaya-0.42 Mod_rewrite_1.55 openSSL-0.9.4g 5. The dependency engine requests the action to be logged.

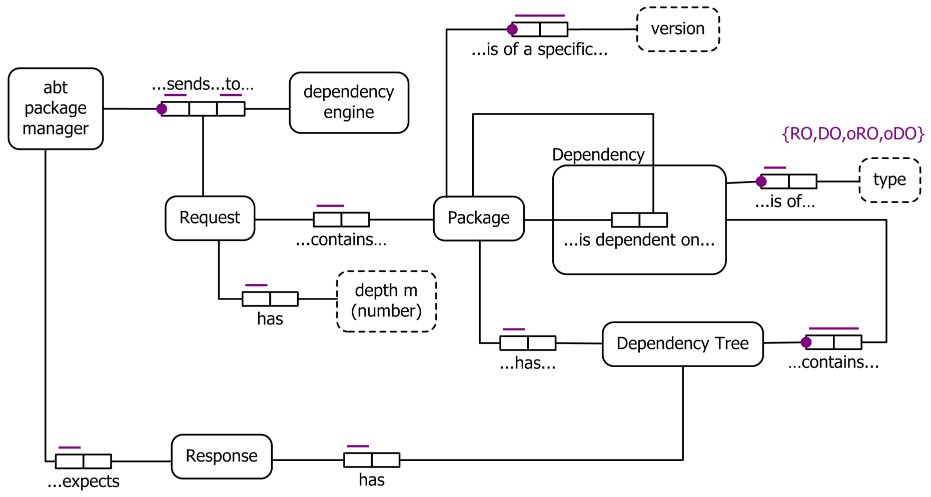
Use Case Name: UC08	Scenario
Scenario S15:	1. The dependency engine receives a request to sort the buildlist which contains Apache-httpd-2.0.55 and OpenSSL-049g. 2. For each item UC04 is triggered. The item with the highest count upwards is started with. In this case both have 1 DO upwards so the first one in the buildlist is picked: Apache-httpd-2.0.55. 3. Creating the buildlist: (a) Add Apache-httpd-2.0.55 to the buildlist + mark. Pick next upwards dependency. (b) Add mod-rewrite-1.15 to the buildlist + mark. No dependencies upwards. (c) Pick next not marked in the unsorted buildlist: OpenSSL-049g. (d) Add OpenSSL-049g to the buildlist + mark. Pick next upwards dependency. (e) Add Apache-httpd-2.0.55 to the buildlist + mark. Pick next upwards dependency. (f) Add mod-rewrite-1.15 to the buildlist + mark. No dependencies upwards. 4. Return the buildlist to the initiating actor as follows: • Apache-httpd-2.0.55 • mod-rewrite-1.15 • OpenSSL-049g • Apache-httpd-2.0.55 • mod-rewrite-1.15

10 Domain Models

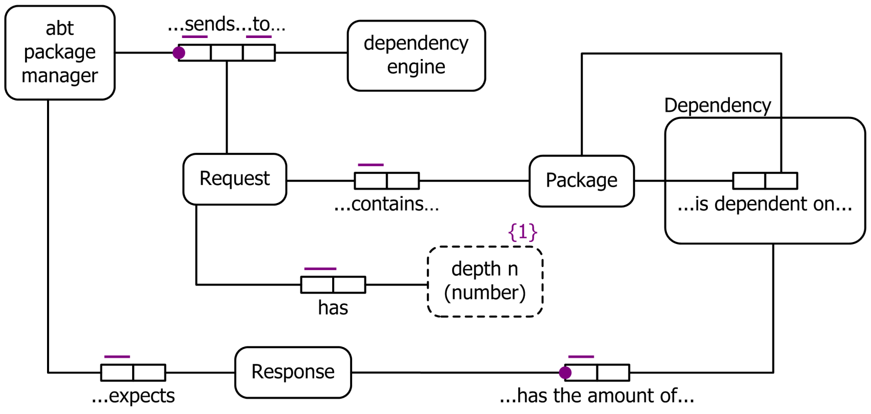
Domain Model
UC01 : Give upward dependency tree for a given package.



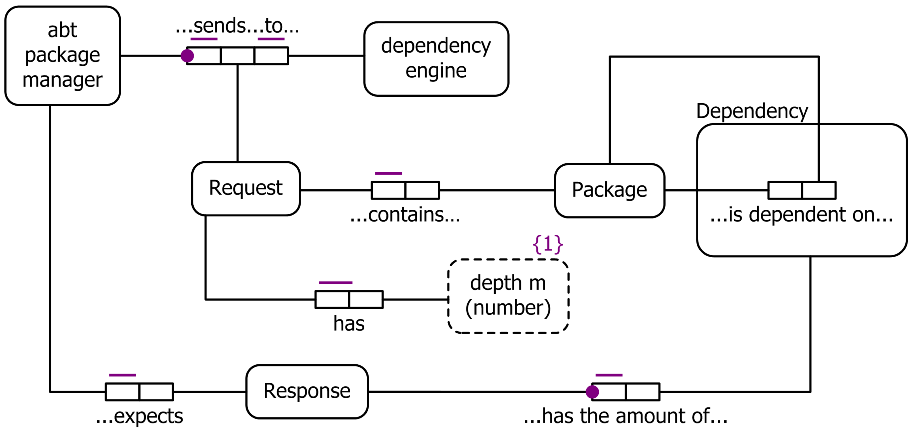
Domain Model
UC02 : Give downward dependency tree for a given package.



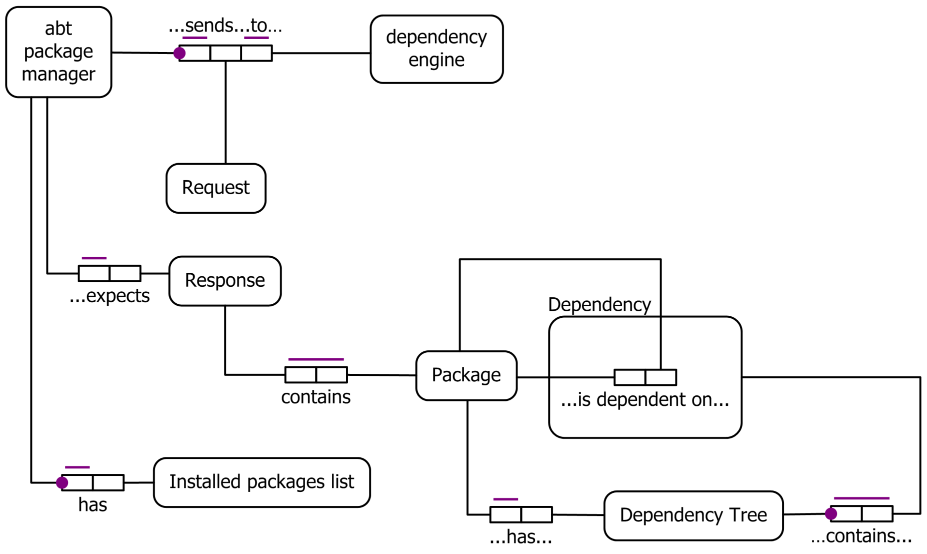
Domain Model
UC03 : Calculate the number of packages dependent on a given package.



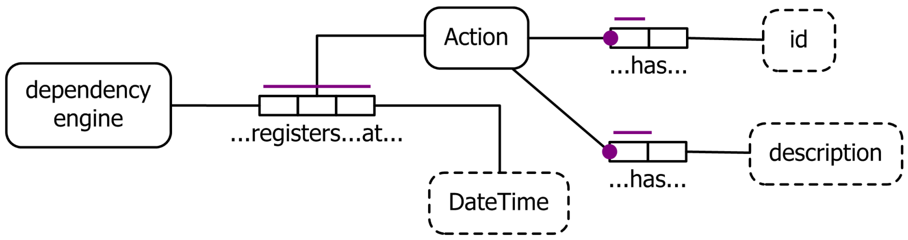
Domain Model
UC4 : Calculate the number of packages a given package depends on.



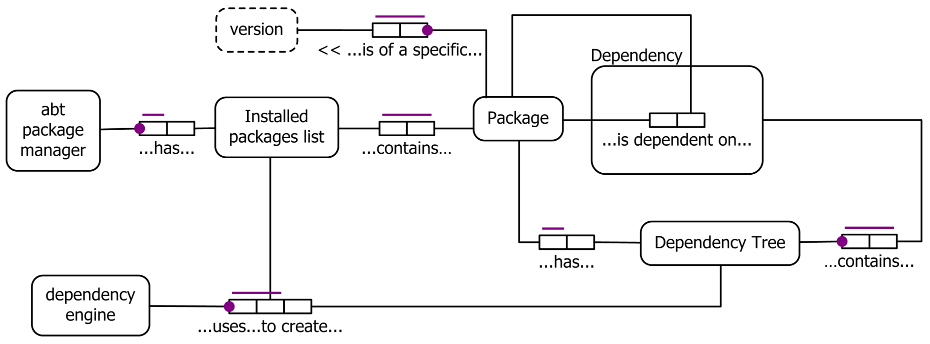
Domain Model
UC05 : Give list of isolated packages.



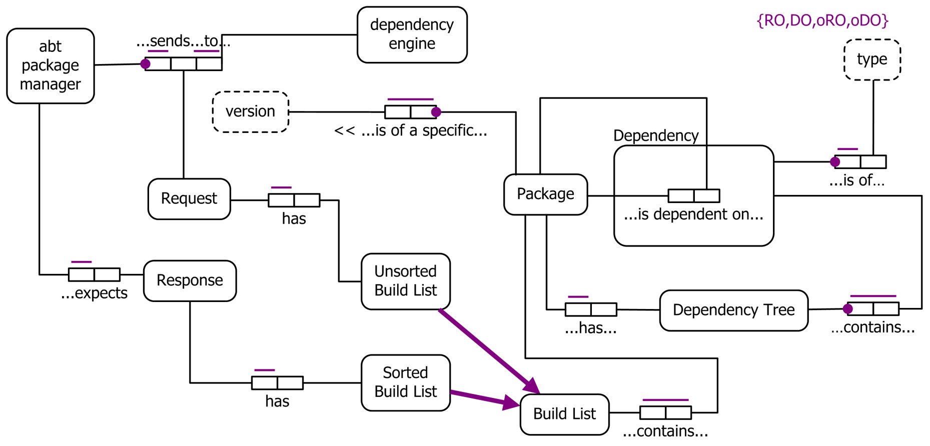
Domain Model
UC06 : Register all system actions.



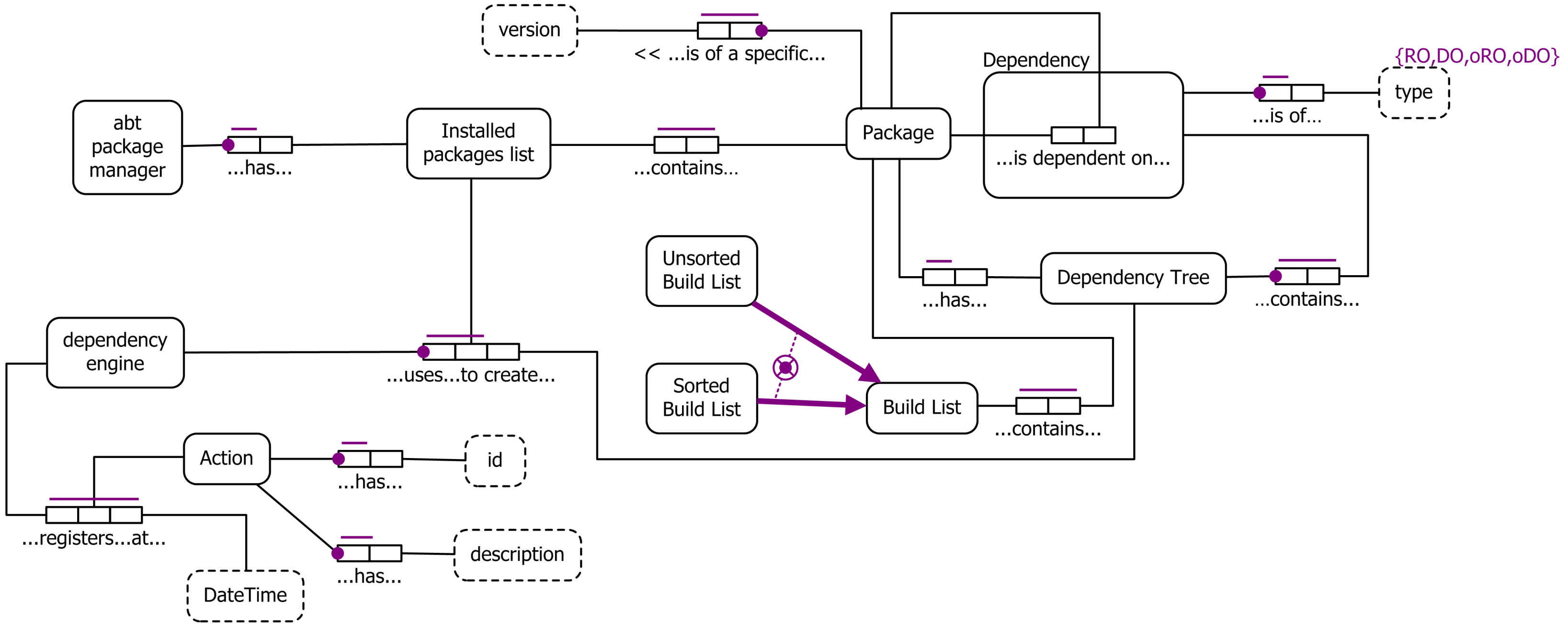
Domain Model
UC07 : Create internal dependency tree.



Domain Model
UC8 :Give sorted build list for a given unsorted build list.



Domain Model
Total overview



11 Business Rules

No.:	BR#01
Rule Definition:	**Every executed action must be logged.**
Type of Rule:	Action triggering
Static / Dynamic:	Static
Source:	AbTLinux Presentation (September 22, 2005).

No.:	BR#02
Rule Definition:	**A dependency is always a relationship between exactly two packages and is always exactly one of the following types: {`DO,RO,oDO,oRO`}**
Type of Rule:	Structural fact
Static / Dynamic:	Static
Source:	AbTLinux Presentation (September 22, 2005).

No.:	BR#03
Rule Definition:	**Packages always have a unique identification.**
Type of Rule:	Structural fact
Static / Dynamic:	Static
Source:	AbTLinux Presentation (September 22, 2005).

No.:	BR#04
Rule Definition:	**Of any given package, no more than one version is installed at the same time.**
Type of Rule:	Structural fact
Static / Dynamic:	Static
Source:	AbTLinux Presentation (September 22, 2005).

12 Non-functional Requirements

Non-functional category:	Integratability
Number:	NFR#01
NRF Name:	**Ability for the system to fit in as part of a larger system.**
Description:	The dependency engine must be designed in such a way that allows seamless integration into the abt package manager. Changes unrelated to the communication between the abt package manager and the dependency engine should not affect the system as a whole.
Completeness:	Final
Maturity:	Young
Source:	Eric Schabell / AbTLinux.org forum.

Non-functional category:	Performance
Number:	NFR#02
NRF Name:	**Effectiveness of the systems processes.**
Description:	The system must be designed in such a way that the dependency engine uses it's resources wisely to maintain overall system stability.
Completeness:	Final
Maturity:	Young
Source:	Eric Schabell / AbTLinux.org forum.

Non-functional category:	Reliability
Number:	NFR#03
NRF Name:	**Confidence in the accuracy of the systems processes.**
Description:	The system must be designed in such a way that the dependency engine is always able to execute correctly when given correct input.
Completeness:	Final
Maturity:	Young
Source:	Eric Schabell / AbTLinux.org forum.

13 Terminological Definitions

A list of definitions used in the Universe of Discourse of the stakeholders which are involved in the dependency engine project.

Definiens	Genus	Differentiæ
`a RO b`	dependency	`a` will be rebuilt any time `b` is rebuilt.
`a DO b`	dependency	`a` will be rebuilt if `b`'s configuration changes.
`a oDO b`	relies-on	`a` optionally depends on `b`.
`a oRO b`	depends-on	`a` optionally relies on `b`.
abt package manager	application	The abt package manager is the package manager being developed for AbTLinux.
build	command	A build or make is the process of compiling the source of a package into the binaries of a package.
build-time dep.	dependency	Dependency that occurs during the building of a software package.
package	archive	An archive that contains a source, header and scripts belonging to a certain software product.
build-list	list	A build-list is a sorted or unsorted listing which packages should be build.
configure	command	Configure is the process of checking the build-environment and writing the build instructions (Makefiles) for a specific package.
dependency	relationship	A dependency is a relationship between 2 packages, the relationship can either be Depends on or relies on.
dependency engine	engine software	A dependency engine is a software application that handles dependencies between all packages within a software application.

dependency tree	data-structure	A dependency tree is a listing of all packages of with their dependencies.
distribution	software	A Unix-like operating system comprising software components such as the Linux kernel, the GNU toolchain, and assorted free and open source software.
make install	command	A make install is the copying of the binaries to a specific location.
optional dep.	dependency	Non-mandatory dependency on a certain package.
package manager	software	A package manager is a software application that manages all aspects of handling software packages, for example installing and upgrading.
rebuild	command	A rebuild is the process of recompiling a specific package and linking to it's source.
required dep.	dependency	Permanent dependency (both relies-on and depends-on) on a certain package.
source	code	The source is a set of instructions which together form a software program.

References

Eric D. Schabell (2005). ABout Time Linux v0.5. *Requirements Document*.

Kulak and Guiney (2003). *Use Cases: Requirements in Context*. Addisson–Wesley, 2 edition.