ABout Time Linux - dependency engine (v0.3)

ing. Aran Arunakiri
aran.arunakiri@gmail.com

ing. David Campbell
dscampbell@student.ru.nl

ing. Bertram Ramspeck
B.D.Ramspeck@student.ru.nl

ing. Robert van Haren
r.vanharen@student.ru.nl

Eric D. Schabell
erics@abtlinux.org

December 15, 2005

Contents

1 Introduction

This is the requirements document for the Dependency Engine sub-project of ABout Time (AbT) Linux, produced by project group 6 (the RUnners). It describes, in a structural manner, the requirements for this project.

AbTLinux is a new Linux distribution born out of the dissatisfaction with previous Linux versions documentation. Main goal of the AbTLinux project is to provide a Linux distribution that is clearly documented from the start on. Every design decision and implementation will be documented extensively. This Document will adhere to this principle.

The document starts of with an abstract view of the goals and aspects of the project in the form of a problem description and Mission, Vision, Values. It also lists stakeholders, domain models and business rules to help put the project into context and set boundaries to help flesh out further developments.

Having clearly set the borders of the project the requirements are gathered and depicted in this document in the form of use cases. Finally, to test these use cases several scenarios are made for each use case showing how it contributes to the functionality of the Dependency Engine project.

2 Problem Statement

Most members the AbTLinux Project team, under supervision of project leader Eric Schabell, 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, because goals, documentation and coding practices are all unclear.

Additionally AbTLinux, like any Linux distribution, is a modular system and thus consists out of many packages. These relate or depend on each other in various ways. This causes problems while compiling and building an AbTLinux suit. It is difficult to get a view of what these dependencies are and how the packages are effected when the configuration changes.

3 Stakeholder Analysis

We have been able to identify the following groups that might have a stake in this project.

- AbTLinux project team under supervision of project leader Eric Schabell. This is the project team that is responsible for the development of AbTLinux and for this project specifically the dependency engine.

- Future users of AbTLinux and the Package manager. They are the the reason this project exists. AbTLinux must be a better Linux then other distributions or people will not use it.

- The general Linux community. If the project is successful then other Linux developers all over the world will benefit from the good documentation. They might even be inspired to document their own work in a better way.

4 Mission, Vision and Values

Mission: Why is there a need for AbTLinux and moreover the Dependency Engine: Linux Distributions are often insufficiently documented or not documented at all. AbTLinux is a project that plans to provide a source-based Linux distribution based on a tool set that has been documented from the beginning of the design cycle. Project Leader and Executive Sponsor Eric Schabell on the mission of this project: "I have had enough of the badly documented package managers out there."

Vision: What will the Dependency Engine and the project achieve:
Clearly documented design, and clear development goals leading to each release so that every aspect of the AbTLinux distribution is traceable right back to its requirement. Eric Schabell's vision of this project: "We need to do as we promise, document and design a package manager for a source-based Linux distribution. The DepEngine is a primary tool to be used by the 'AbT' package manager."

Values: How, through what principles, will the mission and vision be translated to results:

- The AbTLinux project will be thoroughly documented from the start of the project.
- Each new release will have clear development goals.
- Each functionality aspect of the Dependency Engine (and later parts of the AtTLinux project) can be traced back to its requirements based on Use Cases.

5 Statement of Work

5.1 Project Scope

In other distributions of Linux dependencies were badly documented, therefore AbTLinux is providing a dependency engine. Project group RUnners is charged with the requirement analysis process.

The scope of this process includes the creation of requirements in the form of use cases to adequately define the system with which dependencies and relations between packages can be identified and shown. Additionally scenarios will be written for these use cases that can be used to test the use case and later on the dependency engine. Business Rules are catalogued. And finally domain models of the proposed dependency engine and its environment will be made.

Project group RUnners will not create the actual Dependency Engine or provide requirements for any other aspect of AbTLinux but the Dependency Engine. For RUnners the project stops when the requirements for the Dependency Engine are gathered and documented.

5.2 Objectives

The Requirement gathering project of the AbTLinux Dependency Engine has several objectives that, when accomplished, will result in a requirements document that will let the development team build a properly functioning Dependency Engine, that will form the core of the Package Manager. These objectives are listed below.

- Analyse and correctly define the problems that must be solved with the DepEngine project and AbTLinux in general.
- Analyse the risks that involve the requirement gathering process.
- Catalogue and inventories the various requirements of the Dependency Engine in the form of Use Cases.
- Test these Use Cases with scenarios to make sure that they are correct.
- Catalogue Business Rules that are relevant to the requirements for the Dependency Engine.
- Get a good picture of the domain of the Dependency Engine and its environment.

5.3 Application Overview

The Dependency Engine is just a small part of the Package manager, but it will make up an important part of the AbTLinux distribution. It will be at the core of the Package Manager that can be used to manage the different packages of the system. The Dependency Engine will keep track of package dependencies and will guard system integrity by indicating these dependencies to the user.

5.4 User Demography

The Dependency Engine will be a sub system of the Packages Manager for AbTLinux. The functions of this engine will be used through the Package Manager by the user of the ABT system. In this light the only user of the Dependency Engine is the Packet Manager. This program will use the Dependency Engines functions to help it keep track of which packages are in the system and what their relation to each other is.

With regard to users of the Packet Manager, they are probably system administrators and people that know a bit more about the system then the general user. The general user will not use most of this functionality since it is unlikely that they will try to modify the system extensively. They only want it to work.

5.5 Constraints

There are several constraints to the project that follow both from Business Rules and the Project Scope. Here we give a summary:

- Only requirements for the Dependency Engine are gathered in this document. (Source: Project Scope)
- All documentation about the AbTLinux project should be made in LaTeX Format. (Source: Project Guideline by Eric Schabell)
- Every development decision and implementation should be documented. (Source: Business Rule 3)
- Documentation must be in English. (Source: Project Guideline by Eric Schabell)

5.6 Assumptions

Eric knows about technical details of AbTLinux and Linux in general, like package dependencies. We assume the information given to us by Eric to be correct.

5.7 Planning, Staffing and Cost

We made the following Staffing and Cost calculation.

Project Team Member	Task	Work Load: hours
Aran	Formalising Problem Statement	2
	Creating Use Case 1 and 2 Facade	3
	Creating Use Case 1 and 2 Filled	6
	Creating Use Case 1 and 2 Focused	4
	Creating scenario for Use Case 1 and 2	2
	Creation of Domain Models	4
Bertram	Cataloguing Business Rules	2
	Cataloguing Non Functional Requirements	3
	Creating Use Case 3 and 4 Facade	3
	Creating Use Case 3 and 4 Filled	6
	Creating Use Case 3 and 4 Focused	4
	Creating scenario for Use Case 3 and 4	2
	Creation of Domain Models	4
David	Getting to know LaTeX	5
	Combining work LaTeX	12
	Reformatting tekst to LaTeX	5
	Creating Statement of Work	3
	Creating MVV	1
	Creating Use Case 5 Facade	1
	Creating Use Case 5 Filled	3
	Creating Use Case 5 Focused	2
	Creating scenario for Use Case 5	1
	Creation of Domain Models	1
Robert	Risk Analasys	2
	Creating Use Case 6 and 7 Facade	3
	Creating Use Case 6 and 7 Filled	6
	Creating Use Case 6 and 7 Focused	4
	Creating scenario for Use Case 6 and 7	2
	Creation of Domain Models	4
Risk Calculus		11
Total		**111**

Dead lines:

October 13th: Facade Iteration

November 17th: Filled Iteration

December 15th: Focused Iteration

5.8 Deliverable Outline

The following section outlines the deliverables:

- A Problem statement
- A Statement of Work
- A Risk Analyses
- Use Cases en Use Case Diagrams that describe the requirements
- A Business rules catalogue

- Use Case Scenarios
- Domain Models of the Dependency Engine and it's environment

This list of deliverables is combined in one document and form the basis for the DepEngine project. They will help achieve the objectives set above.

5.9 Expected Duration

We expect this project to be finished during December.

6 Risk Analysis

The following risks are currently identified. One of them, namely number 4 has already occurred.

Nr.	**Category**	**Risk**	**Days lost if occurs**	**Chance of occurrence**	**Rating**
1	Project members	A project member gets ill	3	40%	1.4
2	Project members	A project member decides to cancel the course/study	5	5%	0.25
3	Communications	The project team wrongly interprets requirements stated by Eric Schabell.	4	60%	2.4
4	Communications	The project team wrongly interprets some parts of the project.	6	70%	4.2
5	Technical	The project team can not work with LaTeXdue to insufficient knowledge	3	50%	1.5
6	Technical	Material got lost due to a computer crash or clumsy savings.	1	5%	0.05

7 Use Case Surveys

We identified the following Use Cases for the Dependency Engine project.

7.1 Use Case 1

Use Case Name:	**Provide dependency tree for selected package**
Initiating actor:	Software Package Manager
Description:	The Package Manager wants information on how a selected package relates to other packages in the system. Different options can be selected to generate trees acounting for RO relations, DO relations, etc.
Maturity:	focused
complexity:	Average
Dependency:	Use case 2 and 3 are implemented.
Source:	Provide depends tree *up and down* for packages: from Erics sheets stating wishes for the DepEngine
Comments:	We have asumed that a tree is a hierarchy of packages involved with one selected package and that a package is selected by the user.

7.2 Use Case 2

Use Case Name:	**Provide tree of packages that a selected package depends on (up tree)**
Initiating actor:	Software Package Manager
Description:	The Package Manager wants information on the packages that are depending on a selected package (up tree). Different options can be selected to generate trees acounting for RO relations, DO relations, etc.
Maturity:	focused
complexity:	complex
Dependency:	A repossitory containing information about the package dependencies is present.
Source:	Provide depends tree *up and down* for packages: from Erics sheets stating wishes for the DepEngine
Comments:	We assume a package is selected by the user.

7.3 Use Case 3

Use Case Name:	**Provide tree of packages that depend on selected package (down tree)**
Initiating actor:	Software Package Manager
Description:	The Package Manager wants information on the packages that are depending on a selected package (up tree). Different options can be selected to generate trees acounting for RO relations, DO relations, etc.
Maturity:	focused
complexity:	Complex
Dependency:	A repossitory containing information about the package dependencies is present.
Source:	Provide depends tree *up and down* for packages: from Erics sheets stating wishes for the DepEngine
Comments:	We assume a package is selected by the user.

7.4 Use Case 4

Use Case Name:	**Give number of Package Dependencies (up tree)**
Initiating actor:	Software Package Manager
Description:	The Package Manager requests information on how many packages a specific package depends. This number is displayed on the screen. Several Oprions can be selected, for instance RO, DO and a maximum number of dependencies.
Maturity:	focused
complexity:	Average
Dependency:	Use Case 2.
Source:	Provide count given pkg's depends: from Erics sheets stating wishes for the DepEngine
Comments:	We assume a package is selected by the user.

7.5 Use Case 5

Use Case Name:	**Give number of depending Packages (down tree)**
Initiating actor:	Software Package Manager
Description:	The Package Manager requests information on how many packages a specific package depends. This number is displayed on the screen. Several Oprions can be selected, for instance RO, DO and a maximum number of dependencies.
Maturity:	focused
complexity:	Average
Dependency:	Use Case 3.
Source:	Provide count pkg's dependent on given pkg: from Erics sheets stating wishes for the DepEngine
Comments:	We assume a package is selected by the user.

7.6 Use Case 6

Use Case Name:	**Provide package list in build order**
Initiating actor:	Software Package Manager
Description:	The Package Manager requests information on the build order of the packages in the system. A list of packages is provided which is sorted in the order they need to be built. Different options can be selected including DO, RO, Logging, Version info, etc.
Maturity:	focused
complexity:	Complex
Dependency:	Use Case 3 and 7.
Source:	Given pkg list sort for proper buid: from Erics sheets stating wishes for the DepEngine
Comments:	We asume that if there is no dependency between two packages that they can be built in any order.

7.7 Use Case 7

Use Case Name:	**Generate list of packages without dependencies (orphan packages)**
Initiating actor:	Software Package Manager
Description:	The Package Manager requests a list of packages in the system which are orphans (have no dependencies). Options can be selected like RO, DO and version info.
Maturity:	focused
complexity:	complex
Dependency:	A repossitory containing information about the package dependencies is present.
Source:	Provide list of packages w/o any depends: from Erics sheets stating wishes for the DepEngine
Comments:	

8 Use Cases

The Use Cases that are surveyed have been worked out in more detail. They are depicted in figure 1. The Tables in the following sections show a more detailed view of them.

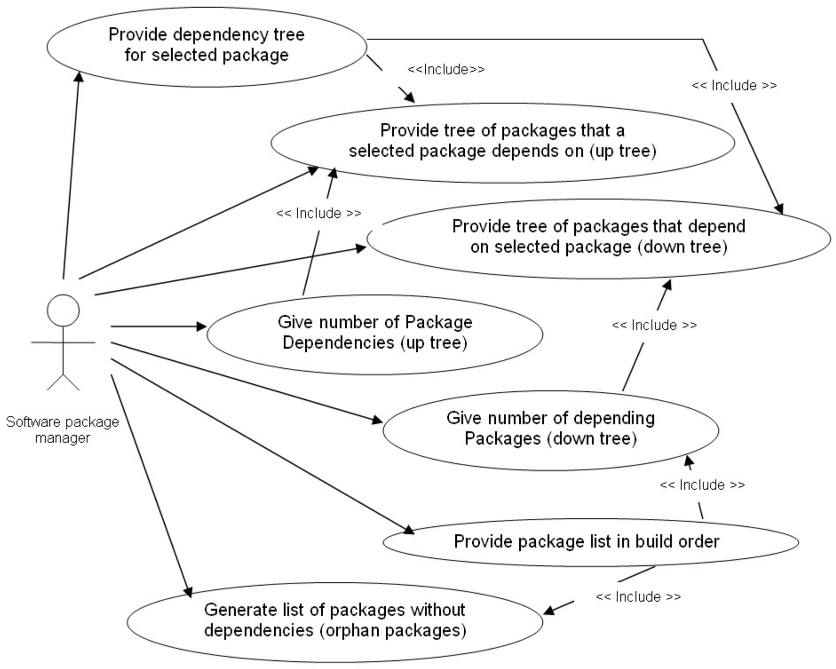
Provide dependency tree for selected package
<<Include>>
<< Include >>
Provide tree of packages that a selected package depends on (up tree)
<< Include >>
Provide tree of packages that depend on selected package (down tree)
Give number of Package Dependencies (up tree)
<< Include >>
Software package manager
Give number of depending Packages (down tree)
<< Include >>
Provide package list in build order
<< Include >>
Generate list of packages without dependencies (orphan packages)

Figure 1: Use Case Diagram

8.1 Use Case 1

Use Case Name:	**Provide dependency tree for selected package**
Authors:	RUnners
Dates:	10-12-2005
Iteration:	focused
Description:	The Package Manager wants information on how a selected package relates to other packages in the system. Different options can be selected to generate trees acounting for RO relations, DO relations, etc.
Actors:	Software Package Manager
Preconditions:	Use case 2 and 3 are implemented.
Triggers:	A function call is done requesting a dependency tree of a given package.
Basic Course of Events:	1. The Package Manager makes a function call to generate the dependency hierarchy tree for a selected package containing option selections. 2. Includes use case 2 with option selections and selected package. 3. Includes use case 3 with option selections and selected package. 4. Returns hierarchy tree of the selected package to the Package Manager.
Alternative paths:	1. The Package Manager makes a function call to generate the dependency hierarchy tree for a selected package containing option selections. 2. Includes use case 2 with option selections. 3. Includes use case 3 with option selections. 4. Returns hierarchy tree to the Package Manager. 5. Log Dependency Engine actions.
Exceptions:	1. The Package Manager makes a function call to generate the dependency hierarchy tree for a selected package containing option selections. 2. Includes use case 2 with option selections. 3. Includes use case 3 with option selections. 4. Use case 2 or 3 reports an error 5. The error is reported to the Package Manager

Assumptions:	A tree is a hierarchy of the entire system. It is possible to show the hierarchy of the packages in a tree.
Postconditions:	1. A tree shows all relevant dependencies for a selected package.
Related business rules:	None

8.2 Use Case 2

Use Case Name:	**Provide tree of packages that a selected package depends on (up tree)**
Authors:	RUnners
Dates:	10-12-2005
Iteration:	focused
Description:	The Package Manager wants information on the packages that are depending on a selected package (up tree). Different options can be selected to generate trees acounting for RO relations, DO relations, etc.
Actors:	Software Package Manager
Preconditions:	All dependencies are traceable.
Triggers:	function call is done requesting a dependency tree of a given package. (up tree)
Basic Course of Events:	1. The Package Manager makes a function call to generate the dependency hierarchy tree (up tree) for a selected package containing option selections. 2. The repository containing information on package dependencies is accessed. 3. Depending on the selected options depending packages are located. 4. Returns hierarchy tree of the selected package to the Package Manager.

Alternative paths:	1. The Package Manager makes a function call to generate the dependency hierarchy tree (up tree) for a selected package containing option selections. 2. The repository containing information on package dependencies is accessed. 3. Depending on the selected options depending packages are located. 4. Returns hierarchy tree of the selected package to the Package Manager. 5. Log Dependency Engine actions. 1. The Package Manager makes a function call to generate the dependency hierarchy tree (up tree) for a selected package containing option selections. 2. The repository containing information on package dependencies is accessed. 3. Depending on the selected options depending packages are located. 4. There is a loop in dependencies 5. The tree till the point of the loop is returned to the Package Manager 6. The Loop is reported to the Package Manager
Exceptions:	1. The Package Manager makes a function call to generate the dependency hierarchy tree (up tree) for a selected package containing option selections. 2. The repository containing information on package dependencies is accessed. 3. The repository can not be accessed 4. The error is reported to the Package Manager
Assumptions:	A repossitory is present containing information on each packages including dependency versions. Depend versions are package related.
Postconditions:	1. A tree shows all relevant dependencies up tree for a selected package.
Related business rules:	1. AbTLinux shall be a stable system, subsystems shall be stable too. 2. The system shall prevent loss of integrity.

8.3 Use Case 3

Use Case Name:	**Provide tree of packages that depend on selected package (down tree)**
Authors:	RUnners
Dates:	10-12-2005
Iteration:	focused
Description:	The Package Manager wants information on the packages that are depending on a selected package (up tree). Different options can be selected to generate trees acounting for RO relations, DO relations, etc.
Actors:	Software Package Manager
Preconditions:	All dependencies are traceable.
Triggers:	function call is done requesting a dependency tree of a given package. (down tree)
Basic Course of Events:	1. The Package Manager makes a function call to generate the dependency hierarchy tree (down tree) for a selected package containing option selections. 2. The repository containing information on package dependencies is accessed. 3. Depending on the selected options packages that the selected package depends on are located. 4. Returns hierarchy tree of the selected package to the Package Manager.
Alternative paths:	1. The Package Manager makes a function call to generate the dependency hierarchy tree (down tree) for a selected package containing option selections. 2. The repository containing information on package dependencies is accessed. 3. Depending on the selected options packages that the selected package depends on are located. 4. Returns hierarchy tree of the selected package to the Package Manager. 5. Log Dependency Engine actions.

Alternative paths:	1. The Package Manager makes a function call to generate the dependency hierarchy tree (down tree) for a selected package containing option selections. 2. The repository containing information on package dependencies is accessed. 3. Depending on the selected options depending packages are located. 4. There is a loop in dependencies 5. The tree till the point of the loop is returned to the Package Manager 6. The Loop is reported to the Package Manager
Exceptions:	1. The Package Manager makes a function call to generate the dependency hierarchy tree (down tree) for a selected package containing option selections. 2. The repository containing information on package dependencies is accessed. 3. The repository can not be accessed 4. The error is reported to the Package Manager
Assumptions:	None
Postconditions:	1. A tree shows all relevant dependencies down tree for a selected package.
Related business rules:	1. AbTLinux shall be a stable system, subsystems shall be stable too. 2. The system shall prevent loss of integrity.

8.4 Use Case 4

Use Case Name:	**Give number of Package Dependencies (up tree)**
Authors:	RUnners
Dates:	10-12-2005
Iteration:	focused
Description:	The Package Manager requests information on how many packages a specific package depends. This number is displayed on the screen. Several Oprions can be selected, for instance RO, DO and a maximum number of dependencies.
Actors:	Software Package Manager
Preconditions:	Use Case 2 is implemented.
Triggers:	A package is selected and the function is called upon to calculate dependencies.

Basic Course of Events:	1. The Packet Manager makes a 'Count Package Dependencies' function call to the DepEngine. 2. Includes Use Case 2 with selected package and options. 3. Count packages in the tree that selected package depends on (up the tree). 4. This number is returned to the Packet Manager.
Alternative paths:	1. The Packet Manager makes a 'Count Package Dependencies' function call to the DepEngine. 2. Includes Use Case 2 with selected package and options. 3. Count packages in the tree that selected package depends on (up the tree). 4. This number is returned to the Packet Manager. 5. Log Dependency Engine actions.
Exceptions:	1. The Packet Manager makes a 'Count Package Dependencies' function call to the DepEngine. 2. Includes Use Case 2 with selected package and options. 3. Use case 2 reports an error 4. The error is reported to the Package Manager
Assumptions:	We assume a package is selected by the user. A library file is present containing information on each packages including dependencies.
Postconditions:	1. Number of packages selected package depends on is displayed on the screen.
Related business rules:	None

8.5 Use Case 5

Use Case Name:	**Give number of depending Packages (down tree)**
Authors:	RUnners
Dates:	10-12-2005
Iteration:	focused
Description:	The Package Manager requests information on how many packages a specific package depends. This number is displayed on the screen. Several Oprions can be selected, for instance RO, DO and a maximum number of dependencies.

Actors:	Software Package Manager
Preconditions:	Use Case 3 is implemented.
Triggers:	A package is selected and the function is called upon to calculate the number of packages that depend on the selected package
Basic Course of Events:	1. The Packet Manager makes a 'Count Depending Packages' function call to the DepEngine. 2. Includes Use Case 2 with selected package and options. 3. Count packages in the tree that depend on the selected package (down the tree). 4. This number is returned to the Packet Manager.
Alternative paths:	1. The Packet Manager makes a 'Count Depending Packages' function call to the DepEngine. 2. Includes Use Case 3 with selected package and options. 3. Count packages in the tree that depend on the selected package (down the tree). 4. This number is returned to the Packet Manager. 5. Log Dependency Engine actions.
Exceptions:	1. The Packet Manager makes a 'Count Depending Packages' function call to the DepEngine. 2. Includes Use Case 3 with selected package and options. 3. Use case 3 reports an error 4. The error is reported to the Package Manager
Assumptions:	A package is selected by the user. A library file is present containing information on each packages including dependencies.
Postconditions:	1. Number of packages that depends on the selected package is displayed on the screen.
Related business rules:	None

8.6 Use Case 6

Use Case Name:	**Provide package list in build order**
Authors:	RUnners
Dates:	10-12-2005
Iteration:	focused
Description:	The Package Manager requests information on the build order of the packages in the system. A list of packages is provided which is sorted in the order they need to be built. Different options can be selected including DO, RO, Logging, Version info, etc.
Actors:	Software Package Manager
Preconditions:	Use Case 3 and 7 are implemented.
Triggers:	The DepEngine function that generates a build order list is called for.
Basic Course of Events:	1. The Packet Manager requests a list of packages sorted in build order depending on optional settings. 2. The repository containing information on package dependencies is accessed. 3. All packages without up tree dependencies are located. 4. Includes Use Case 3 for each package found in step 3 and options. 5. Includes Use Case 7 with options. 6. All found trees are returned to the user together with all orphan packages
Alternative Paths:	1. The Packet Manager requests a list of packages sorted in build order depending on optional settings. 2. The repository containing information on package dependencies is accessed. 3. All packages without up tree dependencies are located. 4. Includes Use Case 3 for each package found in step 3 and options. 5. Includes Use Case 7 with options. 6. All found trees are returned to the user together with all orphan packages 7. Log Dependency Engine actions.

Exceptions:	1. The Packet Manager requests a list of packages sorted in build order depending on optional settings. 2. The repository containing information on package dependencies is accessed. 3. The repository can not be accessed 4. The error is reported to the Package Manager 1. The Packet Manager requests a list of packages sorted in build order depending on optional settings. 2. The repository containing information on package dependencies is accessed. 3. All packages without up tree dependencies are located. 4. Includes Use Case 3 for each package found in step 3 and options. 5. Use case 3 reports an error 6. The error is reported to the Package Manager 1. The Packet Manager requests a list of packages sorted in build order depending on optional settings. 2. The repository containing information on package dependencies is accessed. 3. All packages without up tree dependencies are located. 4. Includes Use Case 3 for each package found in step 3 and options. 5. Includes Use Case 7 with options. 6. Use case 7 reports an error 7. The error is reported to the Package Manager
Assumptions:	None
Postconditions:	1. A complete list of all packages is shown in build order.
Related business rules:	None

8.7 Use Case 7

Use Case Name:	**Generate list of packages without dependencies (orphan packages)**
Authors:	RUnners
Dates:	10-12-2005
Iteration:	focused
Description:	The Package Manager requests a list of packages in the system which are orphans. Options can be selected like RO, DO and version info.
Actors:	Software Package Manager
Preconditions:	None
Triggers:	Generate a list of orphan packages function call is done.
Basic Course of Events:	1. The Package Manager makes the function call to generate a list of orphan packages. 2. The repository containing package data is opened by the DepEngine. 3. All packages that do not have any dependency conections to other packages (depending on what options are selected) are gathered. 4. This information is returned to the Packet Manager.
Alternative Paths:	1. The Package Manager makes the function call to generate a list of orphan packages. 2. The repository containing package data is opened by the DepEngine 3. All packages that do not have any dependency conections to other packages (depending on what options are selected) are gathered. 4. This information is returned to the Packet Manager 5. Log Dependency Engine actions.
Exceptions:	1. The Package Manager makes the function call to generate a list of orphan packages. 2. The repository containing package data is opened by the DepEngine 3. The repository can not be accessed 4. The error is reported to the Package Manager
Assumptions:	A library file is present containing information on each packages including dependencies.

Postconditions:	1. A list of all packages without dependencies (orphans) in the system is provided to the Packet Manager.
Related business rules:	1. AbTLinux shall be a stable system, subsystems shall be stable too. 2. The system shall prevent loss of integrity.

9 Scenarios

In order to check the Use Cases and to provide test situations for later on in the project, scenarios have been written for each of the Use Case Paths. The following sections each have one entire Use Case scenario collection. For the instance examples a fake system is created which is illustrated in figure 2.

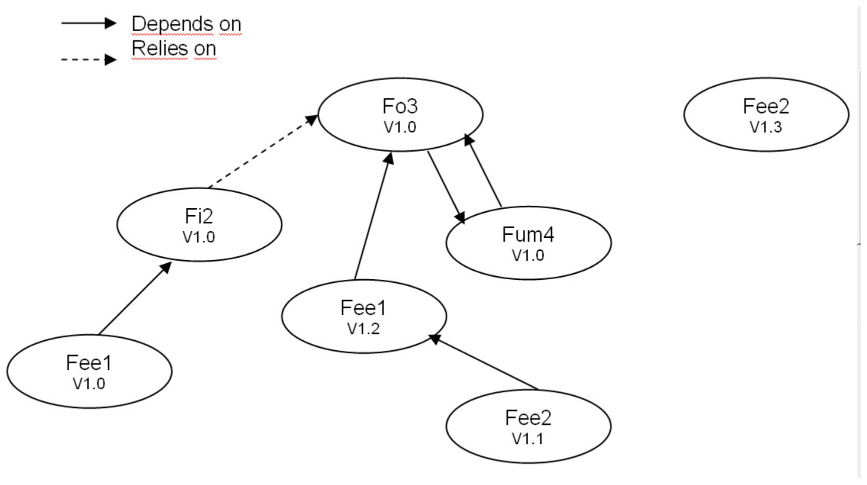

Figure 2: Instance example system.

9.1 Use Case 1

Use Case Name:	**Scenario**
Use Case Steps:	1. The Packet Manager wants to know the entire dependency tree for the Fi2 package. Only DO relations must be shown and version info must be given. 2. Includes use case 2 with options DO relations and version info and package Fi2. 3. Includes use case 3 with options DO relations and version info and package Fi2. 4. The Package manager recieves Fi2-Fee1-NoMorePackages.
Alternative Path:	1. The Packet Manager wants to know the entire dependency tree for the Fi2 package. Only DO relations must be shown and version info must be given. All actions must be logged. 2. Includes use case 2 with options DO relations and version info and package Fi2. 3. Includes use case 3 with options DO relations and version info and package Fi2. 4. The Package manager recieves Fi2-Fee1-NoMorePackages. 5. The DepEngine records all the steps in logfile 'DepEnLog.txt'.
Exceptions:	1. The Packet Manager wants to know the entire dependency tree for the Fo3 package. with no extra options selected. All actions must be logged. 2. Includes use case 2 with no extra options and package Fo3. 3. The the file package.dll can not be openend and use case 2 relays that error 4. The Package manager recieves the message 'package library can not be openend'. 5. The DepEngine records all the steps in logfile 'DepEnLog.txt'.

9.2 Use Case 2

Use Case Name:	**Scenario**
Use Case Steps:	1. The Package Manager wants the version info and names of the RO up tree packages that are depending on package Fi2 V1.0 2. The Package manager recieves Fo3 V1.0 1. The Package Manager wants the names of the DO up tree packages that are depending on package Fee2 V1.3 2. The Package manager recieves no packages
Alternative Path:	1. The Package Manager wants the version info and names of the RO up tree packages that are depending on package Fi2 V1.0. All actions must be logged. 2. The Package manager recieves Fo3 V1.0 3. The DepEngine records all the steps in logfile 'DepEnLog.txt'.

9.3 Use Case 3

Use Case Name:	**Scenario**
Use Case Steps:	1. The Package Manager wants the names of the RO down tree packages that are depending on package Fo3 V1.0 2. The Package manager recieves Fi2 1. The Package Manager wants the version info and names of the DO down tree packages that are depending on package Fi2 V1.0 2. The Package manager recieves the package Fee1 V1.0
Alternative Path:	1. The Package Manager wants the names of the DO down tree packages that are depending on package Fi2 V1.0. All actions must be logged. 2. The Package manager recieves the package Fee1 3. The DepEngine records all the steps in logfile 'DepEnLog.txt'.

9.4 Use Case 4

Use Case Name:	Scenario
Use Case Steps:	1. The Package Manager wants the number of packages that RO package Fo3 V1.0 2. The Package manager recieves the number 1 1. The Package Manager wants the number of packages that DO package Fi2 V1.0 2. The Package manager recieves the number 1
Alternative Path:	1. The Package Manager wants the number of packages that RO package Fo3 V1.0. 2. The Package manager recieves 1 3. The DepEngine records all the steps.

9.5 Use Case 5

Use Case Name:	Scenario
Use Case Steps:	1. The Package Manager wants the number of dependencies for package Fo3 V1.0 2. The Package manager recieves the number 3.
Alternative Path:	1. The Package Manager wants the number of dependencies for package Fum4 V1.0. 2. The Package manager recieves 'there is a dependency loop: Fo3 v1.0 depends on fum4 v1.0, and vise versa'. 3. The DepEngine records all the steps.

9.6 Use Case 6

Use Case Name:	Scenario
Use Case Steps:	1. The Packet Manager wants to know the entire build tree for the Fi2 package together with all orphan packages. Only DO relations must be shown and version info must be given. All actions must be logged. 2. The DepEngine returns Fee v1.0 and Fee2 v1.3
Alternative Path:	1. The Packet Manager wants to know the entire build tree for the Fi2 package. Only DO relations must be shown and version info must be given. All actions must be logged. 2. Fee1 v1.0 is returned by the depEngine. 3. The DepEngine logs all the steps.
Exceptions:	1. The Packet Manager wants to know the entire dependency tree for the Fo3 package. with no extra options selected. All actions must be logged. 2. Includes use case 2 with no extra options and package Fo3. 3. The the file package.dll can not be openend and use case 2 relays that error 4. The Package manager recieves the message 'package library can not be openend'. 5. The DepEngine records all the steps.

9.7 Use Case 7

Use Case Name:	Scenario
Use Case Steps:	1. The Packet Manager wants to know all orphan packages. 2. The package manager receives Fee2 v1.3 3. The DepEngine logs all the steps.
Alternative Path:	1. The Packet Manager wants to know all orphan packages of version 1.0. 2. The package manager receives an empty list of packages. 3. The DepEngine logs all the steps.
Exceptions:	1. The Packet Manager wants to know all orphan packages. 2. The depEngine cannot open packageinfo.dll 3. The Package manager recieves the message package library can not be openend. 4. The DepEngine records all the steps in logfile DepEnLog.txt.

10 Domain Models

To get a better understanding of what exactly the DepEngine is and how it relates to the ABT Linux suit and the Package Manager, domain models are created. Each Use Case has its own domain model. In the domain models a technique is used to describe processes. This technique is described by Dr. H.A. Proper, in Architecture-driven Information System Engineering, section 1, chapter 6, paragraph 6.2.

10.1 Use Case 1

UC1.pdf

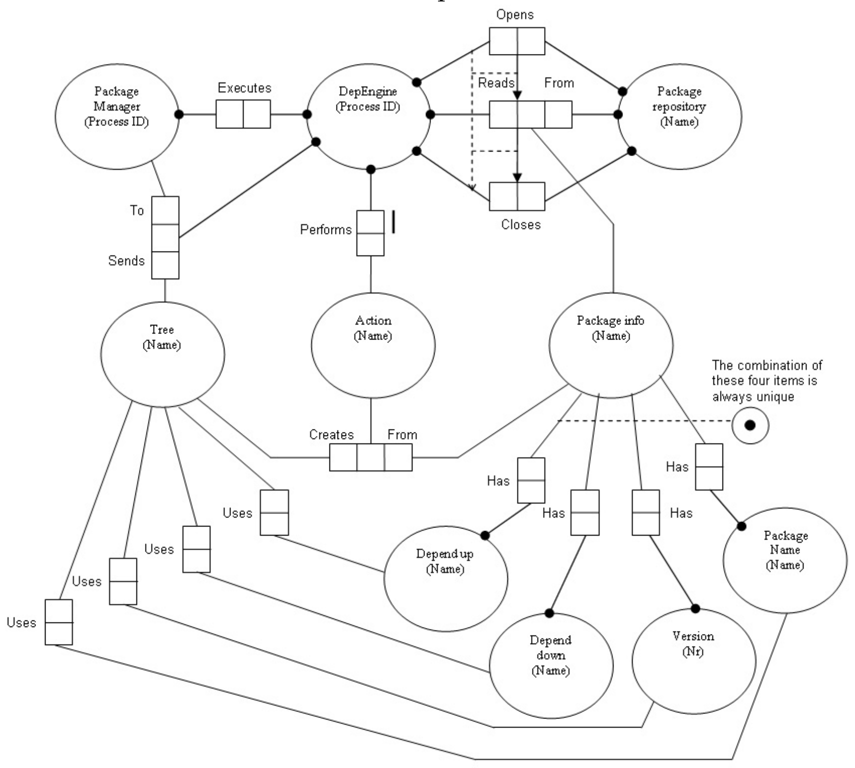

Figure 3: Domain Model of Use Case 1

10.2 Use Case 2

UC2.pdf

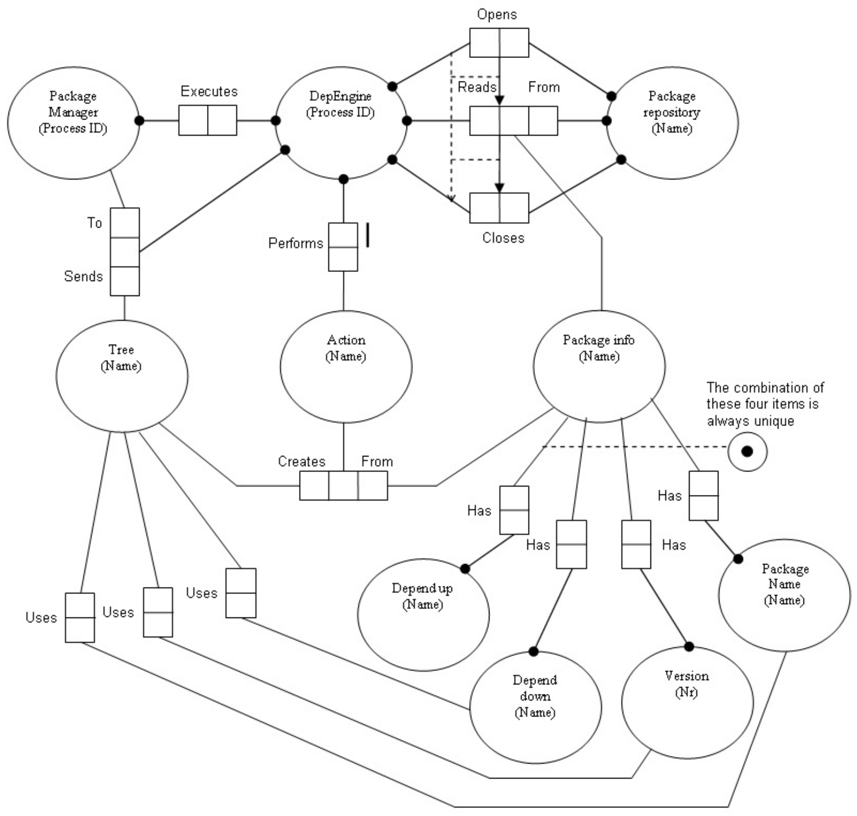

Figure 4: Domain Model of Use Case 2

10.3 Use Case 3

UC3.pdf

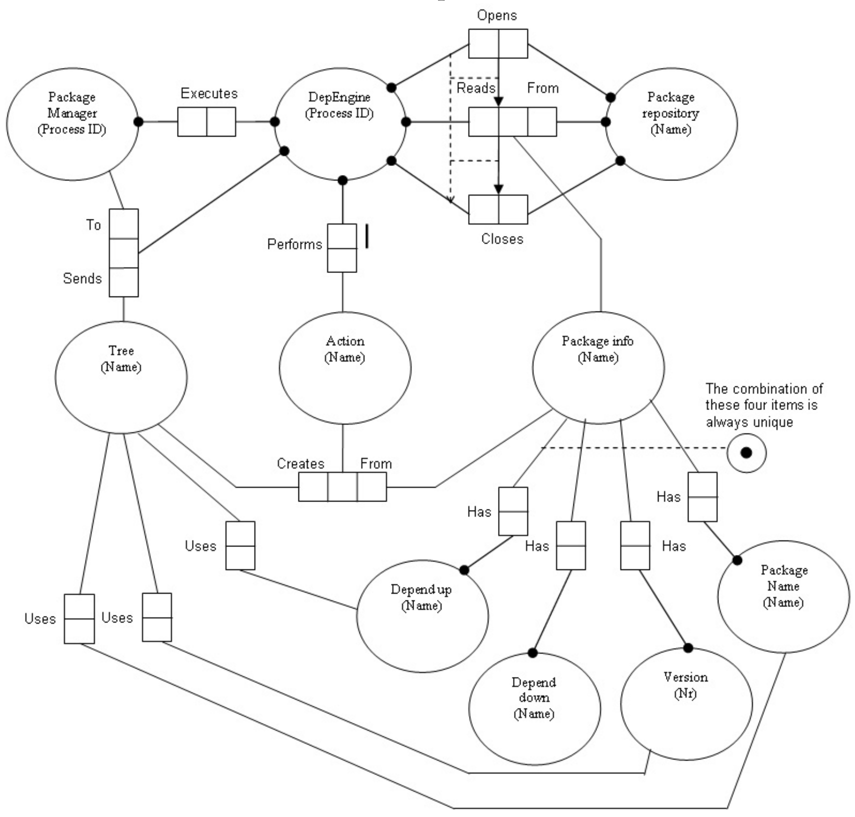

Figure 5: Domain Model of Use Case 3

10.4 Use Case 4 and 5

UC4.pdf

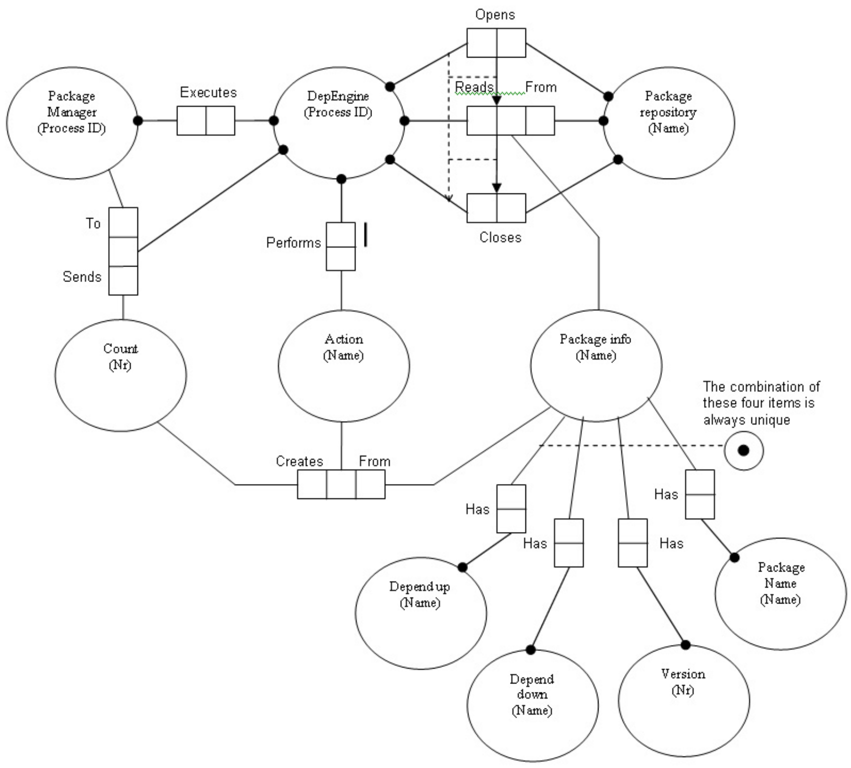

Figure 6: Domain Model of Use Case 4 and 5

10.5 Use Case 6

UC6.pdf

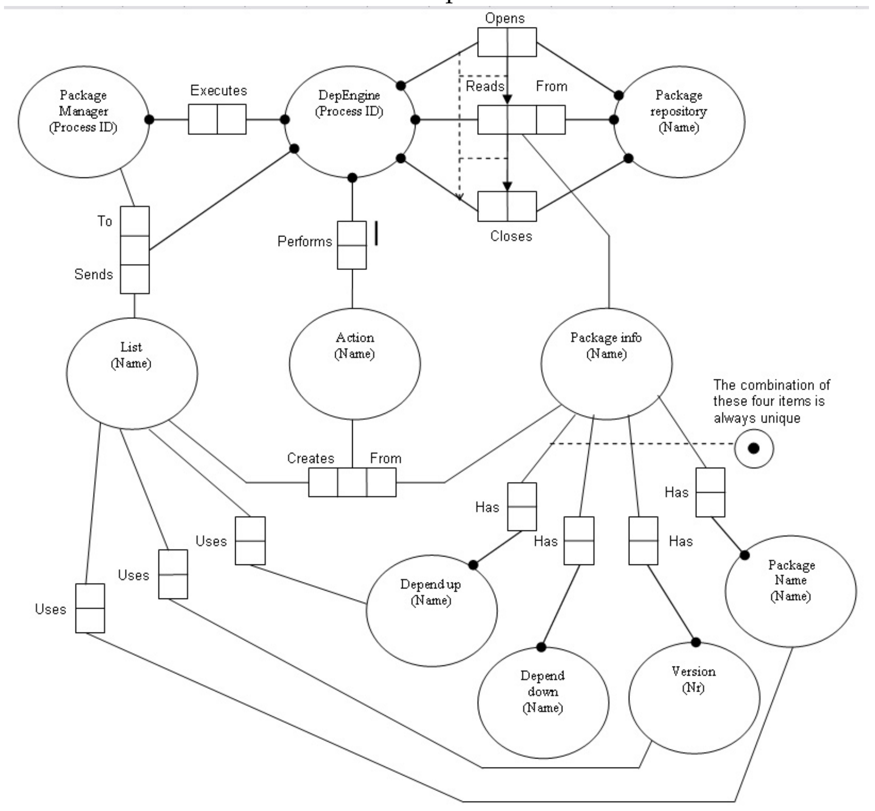

Figure 7: Domain Model of Use Case 6

10.6 Use Case 7

UC7.pdf

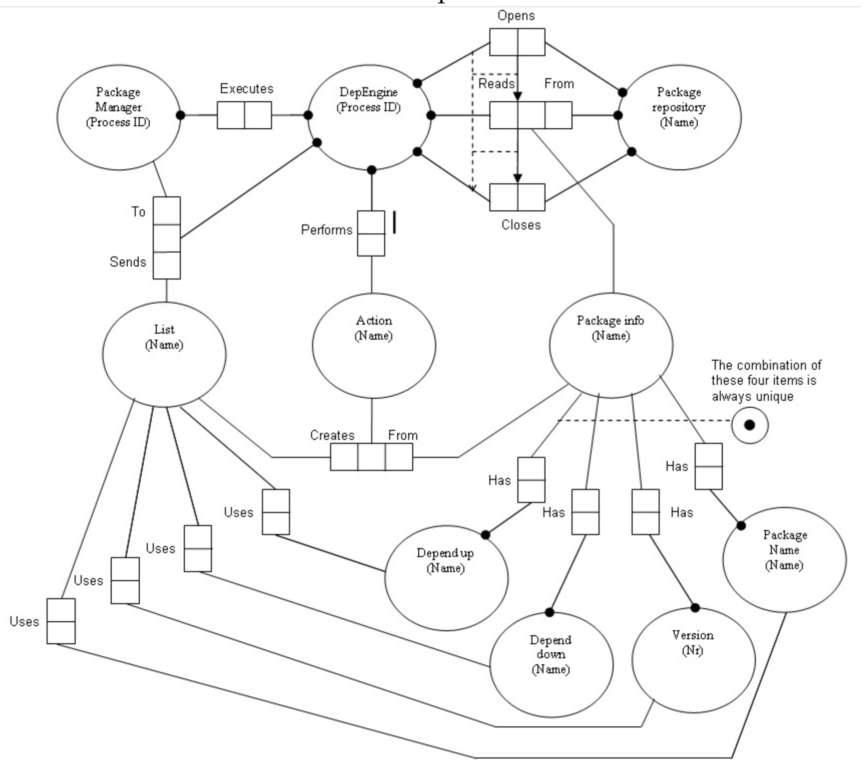

Figure 8: Domain Model of Use Case 7

10.7 System Overview

UC0.pdf

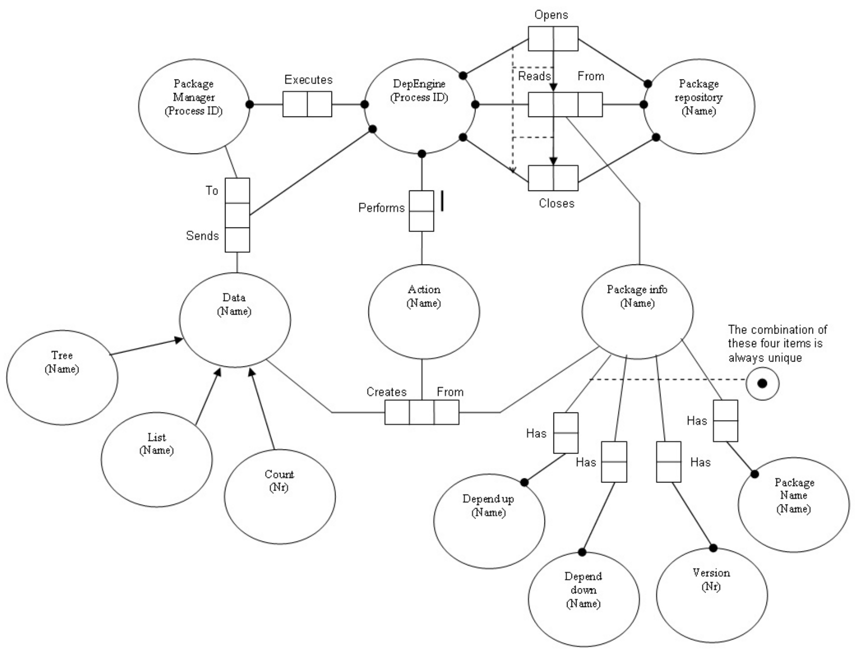

Figure 9: Domain Model of entire system

11 Business Rules

We have been able to identify the following business rules.

Nr.	Rule Definition	Type of Rule	Static/ Dynamic	Source
1	AbTLinux shall be a stable system, sub-systems shall be stable too.	Structural fact	Static	Project Guideline by Eric Schabell
2	The system shall prevent loss of integrity.	Action restricting	Static	Project Guideline by Eric Schabell
3	Everything involving AbTLinux shall be documented, from development till after deployment.	Action triggering	Static	Project Guideline by Eric Schabell

12 Non-functional Requirements

We have been able to identify the following non-functional requirements relevant to the Dependency Engine.

Interoperability: The Dependency Engine must easily communicate with the Packet Manager. Functions must have appropriate names that make clear what the function does. The Input and output values must be kept at a minimum for what the functionality requires, so there will be no redundant data exchange.

Integratability: The Dependency Engine is just a small part of the Package Manager. It is important that de Dependency Engine can easily be used by the Package Manager. Function calls must be easy to use and well documented.

Reliability: The Dependency Engine must give reliable results about package dependencies. This is largely dependent on the library the Dependency Engine uses. If the DepEngine makes a statement about the packages and their dependencies the user should be able to trust this statement for 100%.

Robustness: All errors have to pass through to the Packet Manager so that the Manager and users of the Manager can be aware of what is going on in the DepEngine.

Scalability: The Dependency Engine should function on any system that will support AbTLinux. If it can run AbT it must run the DepEngine too.

13 Terminological Definitions

AbTLinux	-	A Linux distribution, ultimate goal of the ABT project team.
AbTLinux Project team	-	Project team that is developing the AbTLinux distribution and that is currently working on the DepEngine project.
Configuration	-	A collection of packages making up a system, including the dependencies and relations.
DepEngine	-	see Dependency Engine
Dependency Engine	-	Part of the Package manager. It calculates the dependencies of packages on each other.
Dependency relation	-	There are four types of relations between packages: • DO; Depends on: if package X DO Y then X should be rebuilt anytime Y is rebuilt. • ODO; Optionally depends on: same as DO but this can evolve ofer time. • RO; Relies on: if package X DO Y then X should be rebuilt anytime Y configuration is changed. • ORO; optionally relies on: same as RO but this can evolve ofer time.
Dependency tree	-	A hierarchical tree showing how packages depend on each other.
DO	-	see Dependency relation.
Linux distribution	-	A collection of packages that, together form a Linux operating system and peripheral tools.
ODO	-	see Dependency relation.
ORO	-	see Dependency relation.
Package	-	Linux consists out of numerous packages all contributing to some part of the functionality of the system.
Package Information Repossitory	-	yadayada
Package Manager	-	A tool that will let users of AbTLinux manage packages of the system in a correct way.
Package dependency	-	Even though Linux is a modular system some packages require functionality provided by other packages, i.e. they depend on each other.
RO	-	see Dependency relation.
RUnners	-	One of the project teams charged with the requirement gathering process.