# ABout Time Linux - dependency engine (v0.1)

Pepijn Arts
pepijnarts@student.ru.nl

Matthijs van Roosmalen
matroos@abtlinux.org

Eddy Klomp
epjklomp@rediffmail.com

Ilona Wilmont
ilonawilmont@student.ru.nl

December 14, 2005

Eric D Schabell
erics@abtlinux.org

## Contents

## 1 Introduction

ABout Time Linux (ABT Linux) is a project that aims to develop a source-based Linux distribution that has been clearly documented from the very start of the design process. In doing so we hope to be able to build a stable and solid distribution that will be both user-friendly and easy for developers to maintain. The system will be build bottom up, and strict guidelines for requirements and design decisions will be followed.
At the heart of this distribution lies the dependency engine (depEngine) that can check for dependencies between packages. These packages can either be new to the system, or exist on the system already. Packages are arranged in a hierarchy, or dependency tree. Our aim is to gather proper and complete requirements for the depEngine, and in doing so we want to pay attention to each relevant aspect of the system making use of Use Cases and Use Case Diagrams.

## 2 Problem statement

This project aims to develop a dependency engine for the AbT Linux Package Manager that has been properly documented by means of requirements gathering. The current Linux distributions are badly documented and problems occur only too frequently with the dependency engine. Dependencies between packages cannot be checked properly, and problems with broken dependencies are left to the user to solve. This makes operations like installation, removal and upgrading a nightmare. During the course of this project, these issues will be taken into account. Clear documentation and development goals should lead to a structured way of designing and building each system release, and a hassle free package management environment.

## 3 Stakeholder Analysis

**Stakeholder List**

- *Eric Schabell*
  - Position Project founder
  - Country Netherlands
  - Executive sponsor's viewpoint At the moment there is not a single dependency engine in existence that handles dependencies between system components in a proper way. They all give up halfway during for instance the installation of an upgrade version. These problems have been left to the user to solve. This project aims to build a dependency engine that can check for dependencies both up and down the dependency tree. The depEngine can check for Relies On and Depends On relations, and can see whether these relations are optional or static. If this project succeeds, operations will be much easier to perform. This means the system can be organized in a clear and structured way, and the system will be much easier

to upgrade and maintain. Both users and developers will benefit from this project. Users will be provided with a stable OS, and developers can work in an environment whose structure is easy to understand and work with.

- *Jose Bernardo*
  - Country Portugal

  Jose worked together with Eric and Laurent on a project called SourceMage GNU/Linux. Jose is still member of the SourceMage team. In this project he helped with several discussions about AbT Linux.

- *Laurent Wandrebeck*
  - Position Package repository manager
  - Country Roubaix, France

  Laurent worked with several Linux distributions over the years, his interest in this project was peaked by the fact that he likes to fully control the systems he workes with, and that he is fond of optimalization in these systems. He knows Eric and Jose from another project he was involved in; the SourceMage GNU/Linux.

- *Bas van Gils*
  - Country Netherlands

  Bas workes at the same lab as Eric (Radboud University of Nijmegen; Institute for Computing and Information Sciences), and is completing a Ph.D.. Like Jose he helped in several discussion about the AbT Package Manager, and about the whole Abt Linux project.

## 4 Mission, Vision and Values

**Mission: what will the project do?**
The goal of this project is to design a subsystem of the abt package manager, the depEngine. This is an essential component of the AbTLinux project. It provides functionality that sets it apart from other, badly documented Linux systems with inefficient package handling. The depEngine deals with all kinds of dependencies, and is called for whenever a situation arises where dependencies are involved. Clear documentation and development goals should lead to a structured way of designing and building, as opposed to current Linux systems with insufficient documentation which leads to badly organized growth paths.

**Vision: what will the end product be?**
The final product will be a stable, working, properly documented dependency engine which will provide a convenient way of working for both users and developers.

**Values: basic principles to guide the project.**
The values for the depEngine project are basically inherited from the AbTLinux project in general, as it is an integral part of this bigger project. As such, the project is guided by the common values of the Linux community; providing software freely available to everyone, in an open source format. Additionally, it is deemed important that the project is well structured and documented from scratch, in order to develop a superior product in an organized manner. It can be described as ”Linux done right”.

# 5 Statement of Work

**Scope**
In our view the abt Package Manager is an essential part of the project, because the abt Package Manager is the only user that has to communicate with the depEngine. That makes the abt Package Manager the only AbTLinux tool in our scope. We also keep the human actors out of our scope. One can say human actors are necessary because they have to keep the depEngine up to date. In our view maintenance is a non-functional requirement. So there isn't a seperate use case involving maintenance and the necessary human actors.

**Objective**
Our objective is to write proper requirements based on the investigation of the dependencies in AbT Linux. So we make the requirements for building the dependency engine.

**Application overview**
The dependency engine is part of the AbT Linux tool, and deals with all kinds of dependencies in the following situations:

1. installation
2. removal
3. rebuild
4. reconfigure
5. upgrade
6. downgrade
7. repair

**User demography**
The users of the dependency engine are not really the users of AbT Linux. They may become developers of the AbT Linux distribution. So the users, or in this case user, of the dependency engine is the AbT Linux tool itself. The dependency engine is providing services that the AbT Linux tool can call upon.

**Staffing**

| | |
|---|---|
| *Matthijs van Roosmalen* | Fourth year Information Science (master) |
| *Eddy Klomp* | Second year Information Science (bachelor) |
| *Pepijn Arts* | Second year Information Science (bachelor) |
| *Ilona Wilmont* | Second year Information Science (bachelor) |

### Division of responsibility

| Responsibility | Deadline | Responsible | Partner |
|---|---|---|---|
| Introduction - design nr. 1 | 12 october | Matthijs van Roosmalen | Pepijn Arts |
| Introduction - design nr. 2 | 14 november | Matthijs van Roosmalen | Pepijn Arts |
| Introduction - complete | 12 december | Matthijs van Roosmalen | Pepijn Arts |
| Problem statement - design nr. 1 | 12 october | Pepijn Arts | Matthijs van Roosmalen |
| Problem statement - design nr 2 | 14 november | Pepijn Arts | Matthijs van Roosmalen |
| Problem statement - complete | 12 december | Pepijn Arts | Matthijs van Roosmalen |
| Stakeholder list/analysis - design nr. 1 | 12 october | Ilona Wilmont | Eddy Klomp |
| Stakeholder list/analysis - design nr. 2 | 14 november | Ilona Wilmont | Eddy Klomp |
| Stakeholder list/analysis - complete | 12 december | Ilona Wilmont | Eddy Klomp |
| Mission Vision Values | 12 october | Ilona Wilmont | Eddy Klomp |
| Statement of Work | 3 october | Eddy Klomp | Ilona Wilmont |
| Risk analysis | 12 october | Eddy Klomp | Ilona Wilmont |
| Use Case Survey - design nr. 1 | 12 october | Matthijs van Roosmalen | Pepijn Arts |
| Use Case Survey - design nr. 2 | 14 november | Matthijs van Roosmalen | Pepijn Arts |
| Use Case Survey - complete | 12 december | Matthijs van Roosmalen | Pepijn Arts |
| Use Case Diagram - design nr. 1 | 12 october | Pepijn Arts | Matthijs van Roosmalen |
| Use Case Diagram - design nr. 2 | 14 november | Pepijn Arts | Matthijs van Roosmalen |
| Use Case Diagram - complete | 12 december | Pepijn Arts | Matthijs van Roosmalen |
| Use Cases - design nr. 1 | 14 november | Eddy Klomp | All |
| Use Cases - complete | 12 december | Eddy Klomp | All |
| Scenarios - design nr. 1 | 14 november | Ilona Wilmont | Eddy Klomp |
| Scenarios - complete | 12 december | Ilona Wilmont | Eddy Klomp |
| Domain Models - design nr. 1 | 14 november | Pepijn Arts | Matthijs van Roosmalen |
| Domain Models - complete | 12 december | Pepijn Arts | Matthijs van Roosmalen |
| Business Rules Catalogue - design nr. 1 | 12 october | Eddy Klomp | Ilona Wilmont |
| Business Rules Catalogue - design nr. 2 | 14 november | Eddy Klomp | Ilona Wilmont |
| Business Rules Catalogue - complete | 12 december | Eddy Klomp | Ilona Wilmont |
| Non-functional Requirements - design nr. 1 | 12 october | Matthijs van Roosmalen | Pepijn Arts |
| Non-functional Requirements - design nr. 2 | 14 november | Matthijs van Roosmalen | Pepijn Arts |
| Non-functional Requirements - complete | 12 december | Matthijs van Roosmalen | Pepijn Arts |
| Terminological Definitions - design nr.1 | 12 october | Pepijn Arts | Matthijs van Roosmalen |
| Terminological Definitions - design nr.2 | 14 november | Pepijn Arts | Matthijs van Roosmalen |
| Terminological Definitions - complete | 12 december | Pepijn Arts | Matthijs van Roosmalen |
| Executive Sponsor Viewpoint | 12 october | Ilona Wilmont | Eddy Klomp |
| Business Process Definitions - design nr. 1 | 14 november | Eddy Klomp | Ilona Wilmont |
| Business Process Definitions - complete | 12 december | Eddy Klomp | Ilona Wilmont |
| Gui Metaphors/Storyboards - design nr. 1 | 14 november | Matthijs van Roosmalen | Pepijn Arts |
| Gui Metaphors/Storyboards - complete | 12 december | Matthijs van Roosmalen | Pepijn Arts |

During the project the project members will meet every monday at 2pm.

## 6 Risk Analysis

*These Risks are based upon an interview with Eric Shabell.*

**Risk 1**

| | |
|---|---|
| **Name:** | Integrating difficulties |
| **Description:** | Delay is possible when integrating the depEngine in the AbT Linux OS . |
| **Days lost:** | Maximum of 30 days |
| **Likelihood of occurrence:** | 60% |
| **Risk rating:** | 18 |

**Risk 2**

| | |
|---|---|
| **Name:** | Incomplete requirements |
| **Description:** | Some requirements may be incomplete due to the requirement project deadline. |
| **Days lost:** | Maximum of 10 days |
| **Likelihood of occurrence:** | 20% |
| **Risk rating:** | 2 |

**Risk 3**

| | |
|---|---|
| **Name:** | Inconsequent abstraction level |
| **Description:** | The abstraction level of some requirements can be to high or to low. |
| **Days lost:** | Maximum of 10 days |
| **Likelihood of occurrence:** | 20% |
| **Risk rating:** | 2 |

**Risk 4**

| | |
|---|---|
| **Name:** | Incompatible requirements |
| **Description:** | Some requirements turn out to be incompatible with each other after the project deadline |
| **Days lost:** | Maximum of 10 days |
| **Likelihood of occurrence:** | 10 |
| **Risk rating:** | 1 |

## 7 Use Case Surveys

| | |
|---|---|
| **Use Case Number:** | 1 |
| **Use Case Name:** | Create dependency tree upwards. |
| **Initiating Actor:** | AbT package manager |
| **Use Case Description:** | Creates a dependency tree for packages in an upward direction.<br>The search for relevant packages goes no deeper than a specified depth N.<br>Packages must be version-compatible to the package the AbT package manager will be modifying, meaning that versions must either be the same, higher or irrelevant. |
| **Iteration:** | Focused |
| **Complexity:** | High |
| **Priority:** | Essential |
| **Source:** | Eric Schabell |
| **Comments:** | The creation of the dependency tree is one of the main functions of the depEngine.<br>This use case can be combined with Use Case 2 if a tree displaying both upward and downward relations is required. |

| | |
|---|---|
| **Use Case Number:** | 2 |
| **Use Case Name:** | Create dependency tree downwards. |
| **Initiating Actor:** | AbT package manager |
| **Use Case Description:** | Creates a dependency tree for packages in a downward direction. The search for relevant packages goes no deeper than a specified depth N. Packages must be version-compatible to the package the AbT package manager will be modifying, meaning that versions must either be the same, higher or irrelevant. |
| **Iteration:** | Focused |
| **Complexity:** | High |
| **Priority:** | Essential |
| **Source:** | Eric Schabell |
| **Comments:** | This use case can be combined with Use Case 1 if a tree displaying both upward and downward relations is required. |

| | |
|---|---|
| **Use Case Number:** | 3 |
| **Use Case Name:** | Provide list of dependencies upwards |
| **Initiating Actor:** | AbT package manager |
| **Description:** | This part of the depEngine will provide a list of the upwards dependencies a package has to the AbT package manager. These dependencies will be obtained from the dependency tree and only displayed up to a desired depth N provided by the AbT package manager. There are four kinds of dependencies, namely: Depends-On and Relies-On relations which can both be mandatory as well as optional. In addition to this, certain versions of a package might be required and this can be specified in the dependency in the following three forms: 1) a specific version, 2) a specific version or higher, 3) any version will do. Having this list of dependencies at its disposal is essential for the AbT package manager. |
| **Iteration:** | Focused iteration |
| **Complexity:** | Medium |
| **Priority:** | High |
| **Source:** | Eric Schabell |
| **Comments:** | This is the main functionality in its most basic form |

| | |
|---|---|
| **Use Case Number:** | 4 |
| **Use Case Name:** | Provide list of dependencies downwards |
| **Initiating Actor:** | AbT package manager |
| **Description:** | This part of the depEngine will provide a list of the downwards dependencies a package has to the AbT package manager. These dependencies will be obtained from the dependency tree and only displayed up to a desired depth N provided by the AbT package manager. There are four kinds of dependencies, namely: Depends-On and Relies-On relations which can both be mandatory as well as optional. In addition to this, certain versions of a package might be required and this can be specified in the dependency in the following three forms: 1) a specific version, 2) a specific version or higher, 3) any version will do. Having this list of dependencies at its disposal is essential for the AbT package manager. |
| **Iteration:** | Focused iteration |
| **Complexity:** | Medium |
| **Priority:** | High |
| **Source:** | Eric Schabell |
| **Comments:** | This is the main functionality in its most basic form |

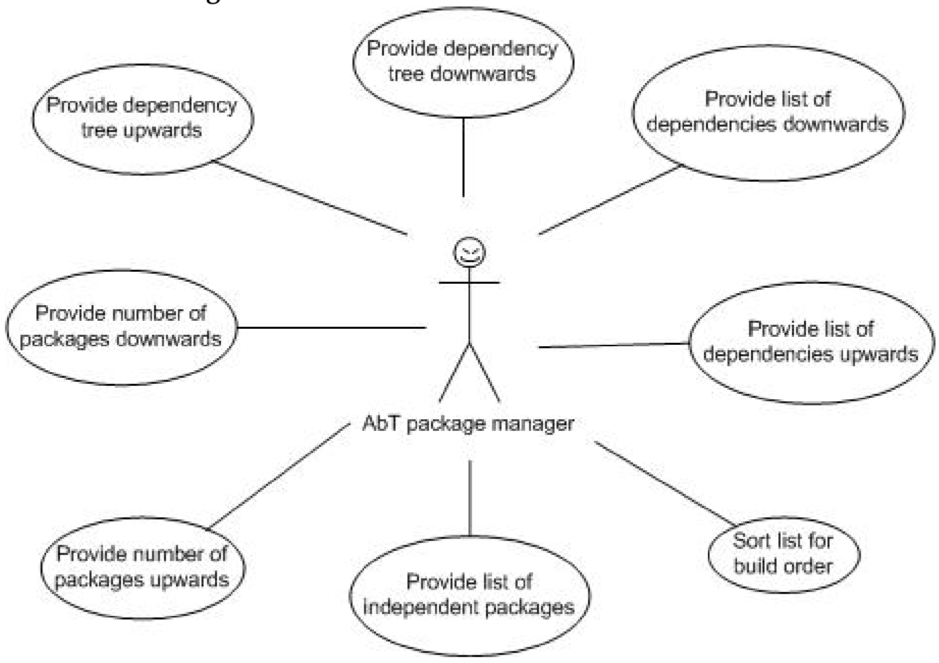
| | |
|---|---|
| **Use Case Number:** | 5 |
| **Use Case Name:** | Sort list for build order |
| **Initiating Actor:** | AbT package manager |
| **Description:** | This part of the depEngine will provide the sorting functionality for the AbT package manager, which it needs to make properly sorted lists of packages. These sorted lists will be used when the AbT package manager needs to build certain packages, and ensure this is done in an efficient and time-saving way. Traditionally, such functionality was not available to package managers, causing packages to be built several times as new dependencies came up during the build process. The AbT package manager aims to improve on this. |
| **Iteration:** | Focused iteration |
| **Complexity:** | Most Complex |
| **Priority:** | Essential |
| **Source:** | Eric Schabell |
| **Comments:** | Important core funtionality, main raison d'etre of the depEngine |

| | |
|---|---|
| **Use Case Number:** | 6 |
| **Use Case Name:** | Provide number of packages upwards |
| **Initiating Actor:** | AbT package manager |
| **Use Case Description:** | This part will provide the number of packages which have a specific, given package dependent on them (upwards) to the AbT package manager. This number includes both direct and indirect packages, with a requested depth N.<br>The dependencies can be Depends-On and Relies-On relations, both mandatory as well as optional. |
| **Iteration:** | Focused |
| **Complexity:** | Medium |
| **Priority:** | Normal |
| **Source:** | Eric Schabell |

| | |
|---|---|
| **Use Case Number:** | 7 |
| **Use Case Name:** | provide number of packages downwards. |
| **Use Case Description:** | This part will provide the number of packages dependent of a specific, given package (downwards) to the AbT package manager. This number includes both direct and indirect packages, with a requested depth N.<br>The dependencies can be Depends-On and Relies-On relations, both mandatory as well as optional. |
| **Iteration:** | Focused |
| **Complexity:** | Medium |
| **Priority:** | Normal |
| **Source:** | Eric Schabell |

| | |
|---|---|
| **Use Case Number:** | 8 |
| **Use Case Name:** | Provide list of independent packages. |
| **Initiating Actor:** | AbT package manager |
| **Use Case Description:** | Provides a list of packages that have no dependencies to the AbT package manager. |
| **Iteration:** | Focused |
| **Complexity:** | Low |
| **Priority:** | Low |
| **Source:** | Eric Schabell |
| **Comments:** | Independent packages are found either at the root of the tree or are not linked to the tree at all. |

**Use Case Diagram:**


Provide dependency tree downwards
Provide dependency tree upwards
Provide list of dependencies downwards
Provide number of packages downwards
Provide list of dependencies upwards
AbT package manager
Provide number of packages upwards
Provide list of independent packages
Sort list for build order

# 8 Use Cases

| | |
|---|---|
| **Use Case Name:** | Create dependency tree upwards. |
| **Authors:** | Ilona Wilmont, Pepijn Arts |
| **Dates:** | Facade complete 12-10-2005<br>Filled complete 10-11-2005<br>Focused complete 11-12-2005 |
| **Iteration:** | Focused |
| **Description:** | Creates a dependency tree for packages in an upward direction.<br>The search for relevant packages goes no deeper than a specified depth N. Packages must be version-compatible to the package the AbT Package Manager package manager will be modifying, meaning that versions must either be the same, higher or irrelevant. |
| **Actors:** | AbT Package Manager package manager |
| **Preconditions:** | 1. A valid search depth for N has been specified.<br>2. N must be a whole number and greater than 0. |
| **Triggers:** | The AbT Package Manager package manager needs to perform an operation for which higher-level packages in the dependency hierarchy are needed. |
| **Basic Course of Events:** | 1. The AbT Package Manager package manager sends an instruction to create a dependency tree for packages higher up in the hierarchy to the depEngine.<br>2. The depEngine checks which packages are going to be influenced by the package manager's actions, going no deeper than level N.<br>3. The depEngine checks if the package versions are compatible.<br>4. The depEngine also checks the dependency relations between these packages.<br>5. The depEngine records the relevant packages and relations into a tree structure.<br>6. The depEngine sends the tree to the AbT Package Manager package manager.<br>7. The depEngine records all its actions in a log-file. |
| **Exception path:** | If the package versions are incompatible, an exception will be thrown. |
| **Exceptions:** | If the specified search depth N is invalid, a new valid one will have to be asked for. |
| **Assumptions:** | There are no dependencies or packages unknown to, or hidden from the depEngine. |
| **Postconditions:** | 1. A dependency tree has been created and is available to the AbT Package Manager package manager.<br>2. The dependency tree correctly displays the dependency relations and the package hierarchy. |
| **Related business rules:** | 1, 2, 4, 5. |

| | |
|---|---|
| **Use Case Name:** | Create dependency tree downwards. |
| **Authors:** | Ilona Wilmont, Pepijn Arts |
| **Dates:** | Facade complete 12-10-2005<br>Filled complete 10-11-2005<br>Focused complete 11-12-2005 |
| **Iteration:** | Focused |
| **Description:** | Creates a dependency tree for packages in a downward direction. The search for relevant packages goes no deeper than a specified depth N. Packages must be version-compatible to the package the AbT Package Manager package manager will be modifying, meaning that versions must either be the same, higher or irrelevant. |
| **Actors:** | AbT Package Manager package manager |
| **Preconditions:** | 1. A valid search depth for N has been specified.<br>2. N must be a whole number and greater than 0. |
| **Triggers:** | The AbT Package Manager package manager needs to perform an operation for which lower-level packages in the dependency hierarchy are needed. |
| **Basic Course of Events:** | 1. The AbT Package Manager package manager sends an instruction to create a dependency tree for packages on a lower level in the hierarchy to the depEngine.<br>2. The depEngine checks which packages are going to be influenced by the package manager's actions, going no deeper than level N.<br>3. The depEngine checks if the package versions are compatible.<br>4. The depEngine also checks the dependency relations between these packages.<br>5. The depEngine records the relevant packages and relations into a tree structure.<br>6. The depEngine sends the tree to the AbT Package Manager package manager.<br>7. The depEngine records all its actions in a log-file. |
| **Exception path:** | If the package versions are incompatible, an exception will be thrown. |
| **Exceptions:** | If the specified search depth N is invalid, a new valid one will have to be asked for. |
| **Assumptions:** | There are no dependencies or packages unknown to, or hidden from the depEngine. |
| **Postconditions:** | 1. A dependency tree has been created and is available to the AbT Package Manager package manager.<br>2. The dependency tree correctly displays the dependency relations and the package hierarchy. |
| **Related business rules:** | 1, 2, 4, 5. |

| | |
|---|---|
| **Use Case Name:** | Provide list of dependencies upwards |
| **Authors:** | Ilona Wilmont, Pepijn Arts, Matthijs van Roosmalen |
| **Dates:** | Facade complete 12-10-2005<br>Filled complete 10-11-2005<br>Focused complete 11-12-2005 |
| **Iteration:** | Focused |
| **Description:** | The depEngine will provide an upwards dependency list of a package to the AbT Package Manager package manager, up to requested depth N.<br>These dependencies can be Depends-On and Relies-On relations, both mandatory as well as optional.<br>Each dependency can be on either 1) a specific version, 2) a specific version or higher, 3) no specific version. |
| **Actors:** | AbT Package Manager package manager |
| **Triggers:** | 1. The AbT Package Manager package manager needs to perform an operation for which higher-level packages in the dependency hierarchy are needed.<br>2. A user is just curious about the dependencies that exist between packages. |
| **Preconditions:** | 1. An acyclic dependency tree has been created and is available to the depEngine.<br>2. N must be a whole number and greater than 0. |
| **Basic Course of Events:** | 1. The AbT Package Manager package manager requests a list of upwards dependencies for a package up to depth N.<br>2. The depEngine compiles this list from the dependency tree.<br>3. The depEngine returns the list to the AbT Package Manager package manager.<br>4. The depEngine logs all its actions in a log-file. |
| **Exceptions:** | If the number N provided in step 1 is invalid, a new one will have to be given. |
| **Assumptions:** | There are no dependencies unknown to, or hidden from, the depEngine. |
| **Postconditions:** | 1. The dependency list must be received by the AbT Package Manager package manager. |
| **Related business rules:** | 1, 2, 5 |

| | |
|---|---|
| **Use Case Name:** | Provide list of dependencies downwards |
| **Authors:** | Ilona Wilmont, Pepijn Arts, Matthijs van Roosmalen |
| **Dates:** | Facade complete 12-10-2005<br>Filled complete 10-11-2005<br>Focused complete 11-12-2005 |
| **Iteration:** | Focused |
| **Description:** | The depEngine will provide a downwards dependency list of a package to the AbT Package Manager package manager up to requested depth N.<br>These dependencies can be Depends-On and Relies-On relations, both mandatory as well as optional.<br>Each dependency can be on either 1) a specific version, 2) a specific version or higher, 3) no specific version. |
| **Actors:** | AbT Package Manager package manager |
| **Triggers:** | 1. The AbT Package Manager package manager needs to perform an operation for which lower-level packages in the dependency hierarchy are needed.<br>2. A user is just curious about the dependencies that exist between packages. |
| **Preconditions:** | 1. An acyclic dependency tree has been created and is available to the depEngine.<br>2. N must be a whole number and greater than 0. |
| **Basic Course of Events:** | 1. The AbT Package Manager package manager requests a list of downwards dependencies for a package up to depth N.<br>2. The depEngine compiles this list from the dependency tree.<br>3. The depEngine returns the list to the AbT Package Manager package manager.<br>4. The depEngine logs all its actions in a log-file. |
| **Exceptions:** | If the number N provided in step 1 is invalid, a new one will have to be given. |
| **Assumptions:** | There are no dependencies unknown to, or hidden from, the depEngine. |
| **Postconditions:** | 1. The dependency list must be received by the AbT Package Manager package manager. |
| **Related business rules:** | 1, 2, 5 |

| | |
|---|---|
| **Use Case Name:** | Sort list for build order |
| **Authors:** | Eddy Klomp, Pepijn Arts, Matthijs van Roosmalen |
| **Dates:** | Facade complete 12-10-2005<br>Filled complete 10-11-2005<br>Focused complete 11-12-2005 |
| **Iteration:** | Focused |
| **Description:** | Sorts a list of packages for an efficient build order and returns it to the AbT Package Manager package manager. |
| **Actors:** | AbT Package Manager package manager |
| **Triggers:** | The AbT Package Manager package manager wants to perform a build action. |
| **Preconditions:** | 1. A list of packages that need to be build has been created.<br>2. N must be a whole number and greater than 0. |
| **Basic Course of Events:** | 1. The AbT Package Manager package manager provides a list of (to be built) packages to the depEngine.<br>2. The depEngine checks the dependencies between the packages, resolving any possible cycles in the process.<br>3. The depEngine sorts the list of packages for an efficient build order.<br>4. The depEngine returns the sorted list to the AbT Package Manager package manager.<br>5. The depEngine logs its actions to a log-file. |
| **Alternative Path:** | When in step 2, the depEngine is not able to resolve the cycle-problems and user input is needed, (for example an optional dependency is causing a looping cycle) it will pose the problem to the AbT Package Manager package manager, possibly for the user to solve. After it is solved, continue at step 3. |
| **Assumptions:** | There are no dependencies or packages unknown to, or hidden from, the depEngine. |
| **Postconditions:** | A list of packages sorted for efficient build order has been returned. |
| **Related business rules:** | 1, 2, 5 |

| | |
|---|---|
| **Use Case Name:** | provide number of packages upwards |
| **Authors:** | Eddy Klomp |
| **Dates:** | Facade complete 12-10-2005<br>Filled complete 10-11-2005<br>Focused complete 11-12-2005 |
| **Iteration:** | Focused |
| **Description:** | Provide the number of packages upwards, to the AbT Package Manager package manager.<br>This number includes both direct and indirect packages, with a requested depth N.<br>The dependencies can be Depends-On and Relies-On relations, both mandatory as well as optional. |
| **Actors:** | AbT Package Manager package manager |
| **Triggers:** | The AbT Package Manager package manager needs to perform an operation for which higher-level packages in the dependency hierarchy are needed. |
| **Preconditions:** | 1. An acyclic dependency tree has been created and is available to the depEngine.<br>2. N must be a whole number and greater than 0. |
| **Basic Course of Events:** | 1. The AbT Package Manager package manager requests the number of packages upwards to a depth of N, to the depEngine.<br>2. The depEngine searches the dependency tree upwards.<br>3. The depEngine counts these packages.<br>4. Repeat steps 2 and 3, until the search depth has been reached.<br>5. The depEngine returns this value to the AbT Package Manager package manager.<br>6. The AbT Package Manager package manager receives the number of packages upwards, with a depth of N.<br>7. The depEngine logs all actions in a log file. |
| **Alternative path:** | If the package already has been counted (4), this package will not be counted again. |
| **Exceptions:** | If the number N provided in step 1 is invalid, a new one will have to be given. |
| **Assumptions:** | There are no dependencies unknown to, or hidden from, the depEngine. |
| **Postconditions:** | 1. The number of packages upwards has been provided to the AbT Package Manager package manager. |
| **Related business rules:** | 1, 2, 5 |

| | |
|---|---|
| **Use Case Name:** | provide number of packages downwards |
| **Authors:** | Eddy Klomp |
| **Dates:** | Facade complete 12-10-2005<br>Filled complete 10-11-2005<br>Focused complete 11-12-2005 |
| **Iteration:** | Focused |
| **Description:** | Provide the number of packages downwards, to the AbT Package Manager package manager.<br>This number includes both direct and indirect packages, with a requested depth N.<br>The dependencies can be Depends-On and Relies-On relations, both mandatory as well as optional. |
| **Actors:** | AbT Package Manager package manager |
| **Triggers:** | The AbT Package Manager package manager needs to perform an operation for which lower-level packages in the dependency hierarchy are needed. |
| **Preconditions:** | 1. An acyclic dependency tree has been created and is available to the depEngine.<br>2. N must be a whole number and greater than 0. |
| **Basic Course of Events:** | 1. The AbT Package Manager package manager requests the number of packages downwards to a depth of N, to the depEngine.<br>2. The depEngine searches the dependency tree downwards.<br>3. The depEngine counts these packages.<br>4. Repeat steps 2 and 3, until the search depth has been reached.<br>5. The depEngine returns this value to the AbT Package Manager package manager.<br>6. The AbT Package Manager package manager receives the number of packages downwards, with a depth of N.<br>7. The depEngine logs all actions in a log file. |
| **Alternative path:** | If the package already has been counted (4), this package will not be counted again. |
| **Exceptions:** | If the number N provided in step 1 is invalid, a new one will have to be given. |
| **Assumptions:** | There are no dependencies unknown to, or hidden from, the depEngine. |
| **Postconditions:** | 1. The number of packages downwards has been provided to the AbT Package Manager package manager. |
| **Related business rules:** | 1, 2, 5 |

| | |
|---|---|
| **Use Case Name:** | Provide list of independent packages. |
| **Authors:** | Ilona Wilmont, Pepijn Arts |
| **Dates:** | Facade complete 12-10-2005<br>Filled complete 10-11-2005<br>Focused complete 11-12-2005 |
| **Iteration:** | Focused |
| **Description:** | Provides a list of packages that have no dependencies to the AbT Package Manager package manager. |
| **Actors:** | AbT Package Manager package manager |
| **Triggers:** | The AbT Package Manager package manager needs to perform an operation which requires knowledge of an independent package (see Terminological Definitions). |
| **Preconditions:** | 1. An acyclic dependency tree has already been created. |
| **Basic Course of Events:** | s<br>1. The AbT Package Manager package manager requests a list of independent packages.<br>2. The depEngine searches the dependency tree for such packages.<br>3. The depEngine writes the packages into a list.<br>4. The AbT Package Manager package manager receives the list of packages.<br>5. The depEngine writes all its actions in a log-file. |
| **Exceptions:** | None. |
| **Assumptions:** | The depEngine has access to all packages and dependencies. |
| **Postconditions:** | A list of independent packages has been created and is available to the AbT Package Manager package manager. |
| **Related business rules:** | 1, 5 |

## 9 Scenarios

| **Use Case Name:** | Create dependency tree upwards.**Scenario** |
|---|---|
| **Use Case Steps:** | 1. The abt package manager has to install foo v2.7.<br>2. For the installation process, the abt package manager requests a dependency tree showing the upwards dependencies for foo v2.7.<br>3. The depEngine searches for packages foo v2.7 depends on, up to a specified depth of 3.<br>4. The depEngine finds that foo v2.7 depends on coffee v2.8 and beef v3.0, and relies on bar v3.1.<br>5. The depEngine checks if all package versions are ¿= v2.7, or if versions do not matter.<br>6. The depEngine creates a dependency tree showing the packages and their relations.<br>7. The depEngine returns the tree to the abt package manager.<br>8. The depEngine records all its actions in the file foo.log. |

| **Use Case Name:** | Create dependency tree downwards. **Scenario** |
|---|---|
| **Use Case Steps:** | 1. The abt package manager has to remove foo v4.5.<br>2. For the process of removal, the abt package manager requests a dependency tree showing the downwards dependencies for foo v4.5.<br>3. The depEngine searches for packages dependant on foo v4.5, up to a specified depth of 4.<br>4. The depEngine finds that dead 6.0, coffee v4.7, beef v7.2 and bar v5.5 are all dependant on foo v4.5.<br>5. The depEngine checks if all package versions are ¿= v4.5, or if versions do not matter.<br>6. The depEngine creates a dependency tree showing the packages and their relations.<br>7. The depEngine returns the tree to the abt package manager.<br>8. The depEngine records all its action in the file foo.log. |

| **Use Case Name:** | Provide list of dependencies downwards. **Scenario** |
|---|---|
| **Use Case Steps:** | 1. The abt package manager requests a list of downwards dependencies for package foo v1.6 up to a depth of 3. <br> 2. The depEngine looks up package foo v1.6 in the dependency tree. <br> 3. The depEngine finds the following dependencies within a depth of 3 levels: bar v6.0 and coffee v4.2. <br> 4. The depEngine returns the list of dependencies 'bar v6.0, coffee v4.2' to the abt package manager. <br> 5. The depEngine logs all actions in foo.log. |

| **Use Case Name:** | Provide list of dependencies upwards. **Scenario** |
|---|---|
| **Use Case Steps:** | 1. The abt package manager requests a list of upwards dependencies for package foo v1.6 up to a height of 1. <br> 2. The depEngine looks up package foo v1.6 in the dependency tree. <br> 3. The depEngine finds only the following dependency 1 level upwards: bar v1.4.4. <br> 4. The depEngine returns the list of dependencies 'bar v1.4.4' to the abt package manager. <br> 5. The depEngine logs all actions in foo.log. |

| **Use Case Name:** | Sort list for build order. **Scenario** |
|---|---|
| **Use Case Steps:** | 1. The abt package manager needs to sort the list of packages: ’foo v1.2, bar v0.04, dead v2.0, beef v3.11’.<br>2. The depEngine determines that bar v0.04 depends on foo v1.2 and that foo v1.2 and dead v2.0 both rely on beef v3.11.<br>3. The depEngine thus creates the following list for optimal build order: ’beef v3.11, dead v2.0, foo v1.2, bar v0.04’.<br>4. The depEngine returns the sorted list to the abt package manager.<br>5. The depEngine logs all actions in foo.log. |
| **Alternative Path:** | 1. If the situation in step 2 is: foo v1.2 depends on dead v2.0, dead v2.0 depends on beef v3.11, beef v3.11 optionally depends on foo v1.2 and bar v0.04 relies on beef v3.11,<br>2. There is now a looping cycle (foo v1.2-¿dead v2.0-¿beef v3.11-¿foo v1.2) due to the optional dependency of beef v3.11 on foo v1.2.<br>3. The depEngine reports this to the abt package manager.<br>4. The abt package manager responds that the optional dependency can be relinquished for now.<br>5. The normal procedure continues at step 3. |

| **Use Case Name:** | provide number of packages upwards. **Scenario** |
|---|---|
| **Use Case Steps:** | 1. The abt package manager asks the depEngine how many packages there are, which have foo v1.0 as a dependent. This search will stop at a depth of 3.<br>2. The depEngine searches and find out that packages beef v1.2 and dead v3.4 have foo v1.0 as a dependent.<br>3. The depEngine counts these packages (2 packages).<br>4. The depEngine searches and find out that package coffee v1.3 has as dependent beef v1.2.<br>5. The depEngine counts these packages (2 packages + 1 package = 3 packages).<br>6. The depEngine searches and find out that package unit v2.0 has as dependent dead v3.4.<br>7. The depEngine counts all packages. (3 packages + 1 package = 4 packages).<br>8. The search depth of 3 has been reached.<br>9. The depEngine returns the result: 4, to the abt package manager.<br>10. The abt package manager receives the result: 4, from the depEngine.<br>11. The depEngine logs all its actions in file foo.log. |
| **Alternative Path:** | If package foo v1.0 has as dependent package beef v1.2 and dead v3.4, and the last package dead v3.4 is also a dependent of beef v1.2 then the abt package manager will count beef 1.2 only once. |

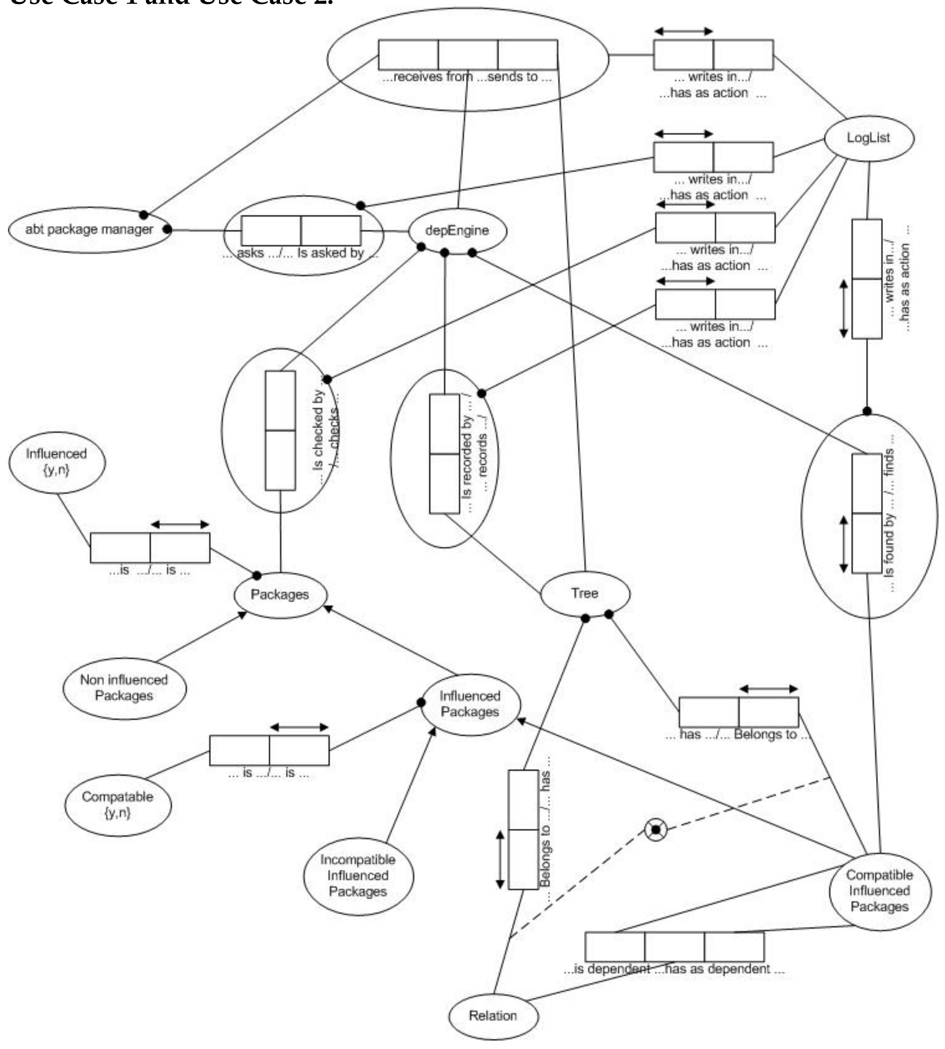
| **Use Case Name:** | provide number of packages downwards. **Scenario** |
|---|---|
| **Use Case Steps:** | 1. The abt package manager asks the depEngine how many packages are dependent of package foo v1.0. This search will stop at a depth of 3.<br>2. The depEngine searches and find out that packages egg v1.2 and mouse v3.4 are dependent of foo v1.0.<br>3. The depEngine counts the packages (2 packages).<br>4. The depEngine seaches and find out that package table v1.5 is dependent of egg v1.2.<br>5. The depEngine counts these packages (2 packages + 1 package = 3 packages).<br>6. The depEngine searches and find out that no package is dependent of mouse v3.4.<br>7. The depEngine counts all packages. (3 packages).<br>8. The search depth of 3 has been reached.<br>9. The depEngine returns the result: 3, to the abt package manager.<br>10. The abt package manager receives the result: 3, from the depEngine.<br>11. The depEngine logs all its actions in file foo.log. |
| **Alternative Path:** | If package egg v1.2 and mouse v3.4 are dependent of foo v1.0, and package egg v1.2 is also dependent of mouse v.3.4 then the package manager will count egg v1.2 only once. |

| Use Case Name: | Provide list of independent packages. |
|---|---|
| **Use Case Steps:** | 1. The abt package manager sends a request to the depEngine for a list of independent packages.<br>2. The depEngine searches the root of the dependency tree and checks for loose packages.<br>3. The depEngine comes up with foo v3.4, beef v4.0 and coffee v1.9.<br>4. The depEngine writes these package names in a list.<br>5. The depEngine returns this list to the abt package manager.<br>6. The depEngine records all its actions in the file foo.log. |

# 10 Domain Models

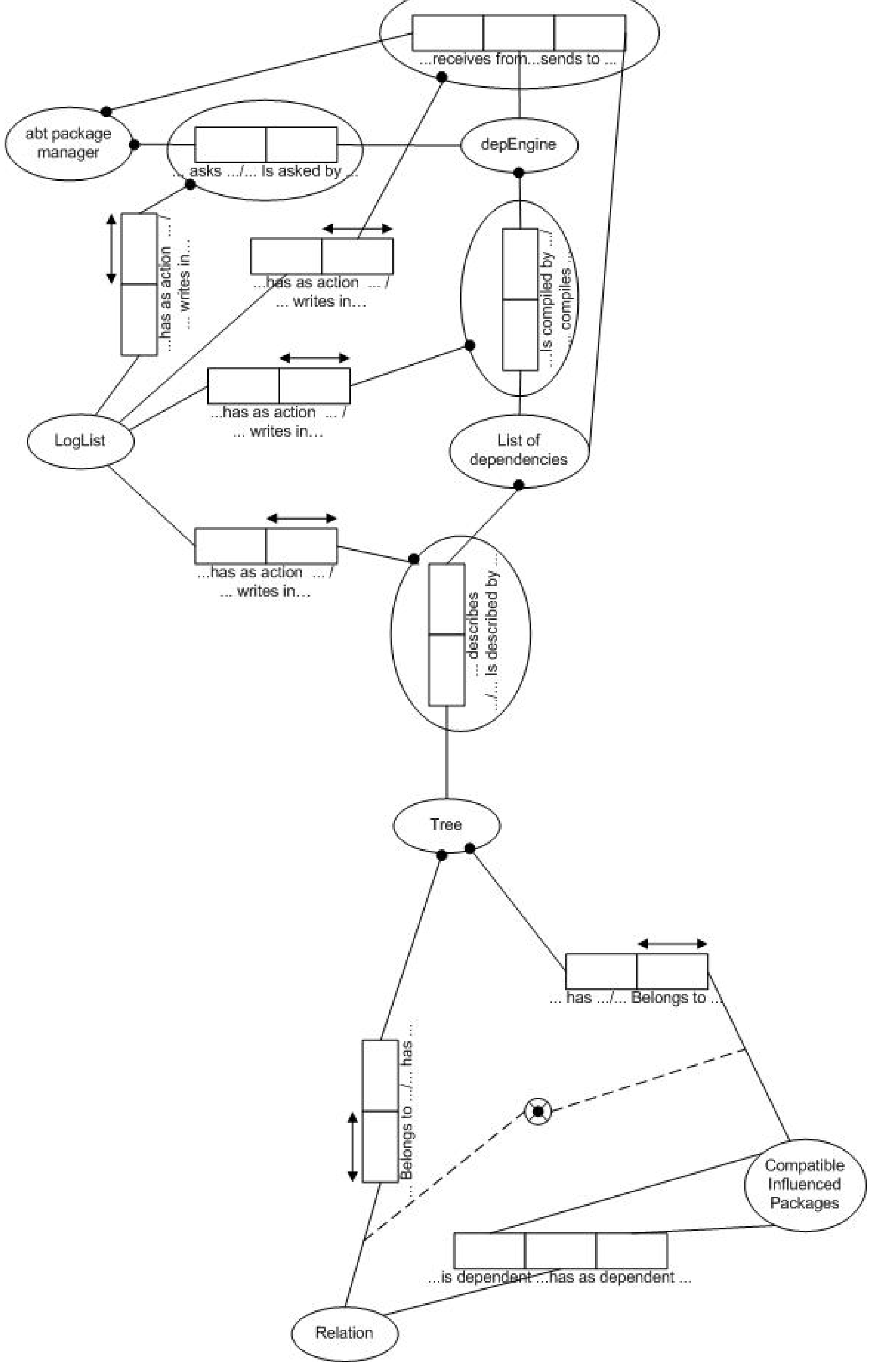
This section provides all the ORM models based on the use cases.

**Use Case 1 and Use Case 2:**



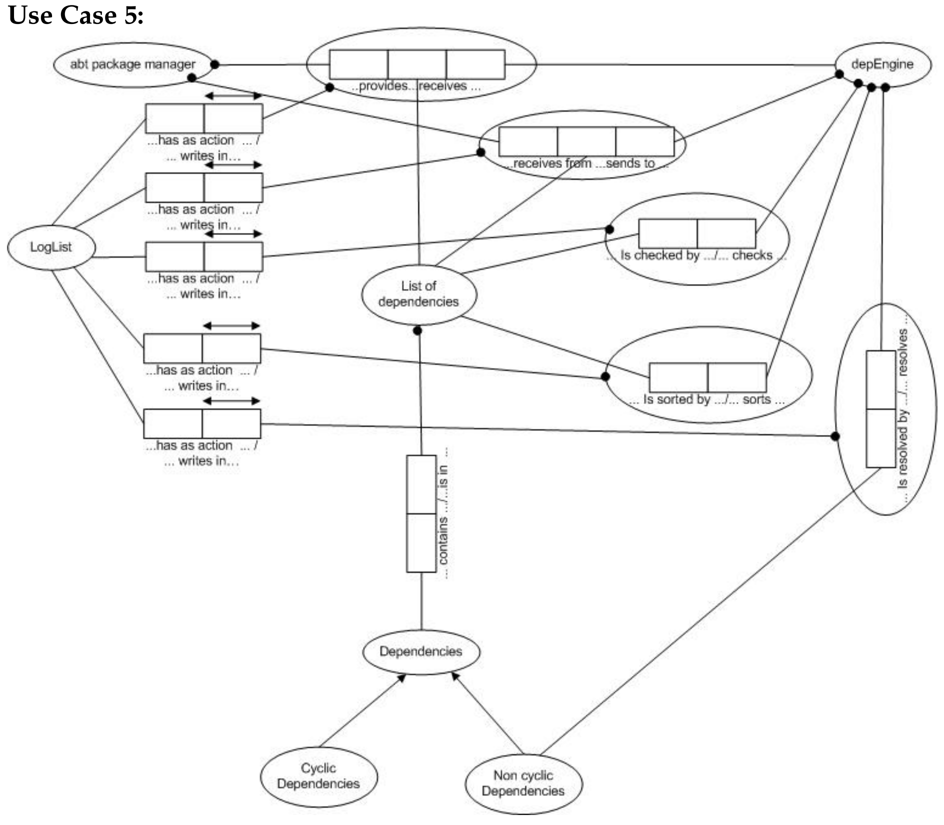
...receives from ...sends to ...
... writes in.../ ...has as action ...
LogList
abt package manager
... asks .../... Is asked by ...
depEngine
... Is checked by ... /... checks ...
... Is recorded by ... / ... records ...
... Is found by ... /... finds ...
Influenced {y,n}
...is .../... is ...
Packages
Tree
Non influenced Packages
Influenced Packages
... has .../... Belongs to ...
Compatable {y,n}
... is .../... is ...
... Belongs to .../... has ...
Incompatible Influenced Packages
Compatible Influenced Packages
...is dependent ...has as dependent ...
Relation

**Use Case 3 and Use Case4:**

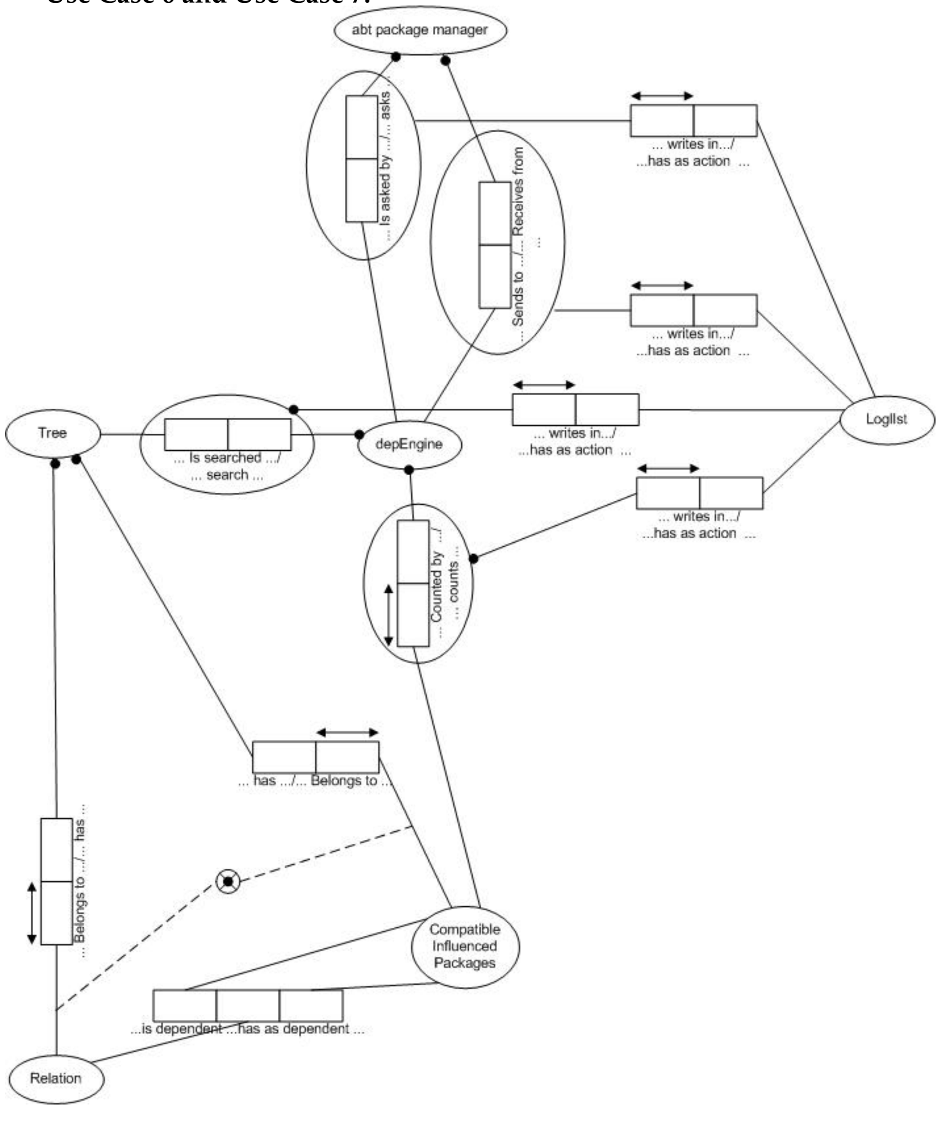


...receives from...sends to ...
abt package manager
... asks .../... Is asked by ...
depEngine
...has as action ... / ... writes in...
...has as action ... / ... writes in...
...Is compiled by .../ ... compiles ...
...has as action ... / ... writes in...
LogList
List of dependencies
...has as action ... / ... writes in...
... describes .../... Is described by ...
Tree
... has .../... Belongs to ...
... Belongs to .../... has ...
Compatible Influenced Packages
...is dependent ...has as dependent ...
Relation

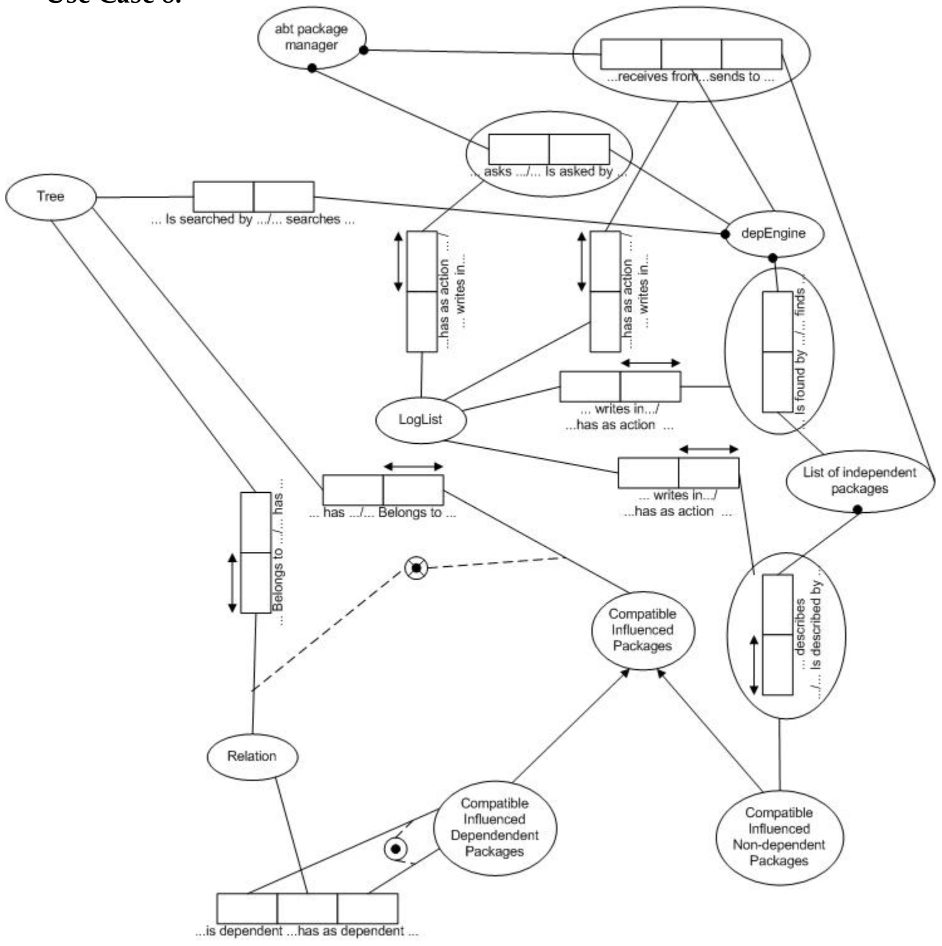
**Use Case 5:**


abt package manager
..provides...receives ...
depEngine
...has as action ... /
... writes in...
...receives from ...sends to ...
...has as action ... /
... writes in...
LogList
...has as action ... /
... writes in...
... Is checked by .../... checks ...
List of dependencies
...has as action ... /
... writes in...
... Is sorted by .../... sorts ...
...has as action ... /
... writes in...
... Is resolved by .../... resolves ...
... contains .../...is in ...
Dependencies
Cyclic Dependencies
Non cyclic Dependencies

**Use Case 6 and Use Case 7:**


abt package manager
... Is asked by .../... asks ...
... Sends to .../... Receives from ...
... writes in.../
...has as action ...
... writes in.../
...has as action ...
Tree
... Is searched .../
... search ...
depEngine
... writes in.../
...has as action ...
LogIlst
... writes in.../
...has as action ...
... Counted by .../... counts ...
... has .../... Belongs to ...
... Belongs to .../... has ...
Compatible Influenced Packages
...is dependent ...has as dependent ...
Relation

**Use Case 8:**


abt package manager
...receives from...sends to ...
... asks .../... Is asked by ...
Tree
... Is searched by .../... searches ...
depEngine
...has as action .../ ... writes in...
...has as action .../ ... writes in...
... Is found by .../... finds ...
... writes in.../ ...has as action ...
LogList
List of independent packages
... writes in.../ ...has as action ...
... has .../... Belongs to ...
... Belongs to .../... has ...
Compatible Influenced Packages
... describes .../... Is described by ...
Relation
Compatible Influenced Dependendent Packages
Compatible Influenced Non-dependent Packages
...is dependent ...has as dependent ...

# 11 Business Rules

**Business Rule Catalogue**

*These Business Rules are based upon an interview with Eric Schabell.*

**Business Rule 1:**

| | |
|---|---|
| **Name:** | Program code |
| **Description:** | All code of the dependency engine must be written in an Object Oriented programming language. |
| **Rule type:** | Structural facts |
| **Static/Dynamic:** | Dynamic (policy may change) |
| **Source:** | Policy rule |

**Business Rule 2:**

| | |
|---|---|
| **Name:** | Dependency types |
| **Description:** | There are 4 types of dependencies. |
| | Structural facts |

1. Relies on
2. Depends on
3. Optionally relies on
4. Optionally depends on

| | |
|---|---|
| **Rule type:** | |
| **Static/Dynamic:** | Static |
| **Source:** | Policy rule |

**Business Rule 3:**

| | |
|---|---|
| **Name:** | Free source code availability |
| **Description:** | AbTLinux will be developed under the GNU General Public License, which makes the source code freely available to everyone. |
| **Rule type:** | Structural facts |
| **Static/Dynamic:** | Static |
| **Source:** | Policy rule |

**Business Rule 4:**

| | |
|---|---|
| **Name:** | Tree contents |
| **Description:** | The packages in the tree include the dependencies of a package. |
| **Rule type:** | Structural facts |
| **Static/Dynamic:** | Static |
| **Source:** | Policy rule |

**Business Rule 5:**

| | |
|---|---|
| **Name:** | All actions in log list |
| **Description:** | All actions of the depEngine will be written in a log. |
| **Rule type:** | Structural facts |
| **Static/Dynamic:** | Dynamic (policy may change) |
| **Source:** | Policy rule |

**Business Rule 6:**

| | |
|---|---|
| **Name:** | Voluntary participation |
| **Description:** | Participation in the AbTLinux project, of which the depEngine is a part, is on a strictly voluntary and unpaid basis. |
| **Rule type:** | Structural facts |
| **Static/Dynamic:** | Dynamic (policy may change) |
| **Source:** | Policy rule |

## 12 Non-functional Requirements

- **Integratability:** the depEngine should work with the abt package manager and be able to communicate with it.
- **Portability:** it should run on all the hardware that AbTLinux will support.
- **Reliability:** all the dependency data provided by the depEngine needs to be accurate.
- **Robustness:** the depEngine should be able to handle errors and special conditions and report them; most notably cycles in dependencies.
- **Performance:** the operations performed by the depEngine should not take a significant amount of time on average hardware.
- **Auditability:** the depEngine should maintain records (log files) of its actions, what happened and who did it.
- **Availability:** whenever the abt package manager is available, so should the depEngine be able to respond to its requests.
- **Data integrity:** none of the data that the depEngine has access to should be lost or corrupted in any way.
- **Installability:** it should require no effort to install the depEngine, it comes bundled with the abt package manager.
- **Localization:** the depengine should be able to support any languages that the AbTLinux project plans to support.
- **Usability:** in order to minimize difficulties using the depEngine, it should work invisible to the user, who interacts only with the abt package manager.

## 13 Terminological Definitions

*In this section we explain some terms used in the document*

- **AbTLinux:**
  An upcoming distribution of Linux.
- **abt package manager:** The abt package manager is the part of AbTLinux which directly communicates with the depEngine. The depEngine exists solely to support the abt package manager, which makes it the sole actor in the requirements.
- **Dependency:**
  When we talk about dependency, we mean the dependency a package has in relation to other packages. The building or configuring of a package affects other packages. There are four kinds of dependencies: the Depends On and the Relies On dependency, which can both be optional or mandatory.
- **Dependency tree:** The dependency tree is a tree-structure of the dependency relations between packages.
- **depEngine:**
  The depEngine is the tool wich reports the dependencies between packages.
- **Depends On:**
  Package A will always be rebuilt when package B is rebuilt.
- **Downgrade:**
  When an older version of a package has replaced a more recent version, we call this a downgrade.
- **Downwards (dependency):** A dependency relation downwards (in the tree) means that the package is needed for another package further down, which depends on it.
- **Independent package:**
  This is a package which does not depend on any other packages, and as such does not have any dependencies upwards in the tree.
- **Linux distribution:**
  Linux has different incarnations, with different interfaces, options and tools, we call those incarnations distributions.
- **OS/Operation System:**
  An operating system is the main program on a system, like Windows, Unix or Linux.
- **Package:**
  A package in Linux can be seen as a piece of code that forms a little program or a part of software.

- **Rebuild:**
  When a package is only recompiled, and no configuration changes have been made.
- **Reconfigure:**
  When a package is recompiled, after configuration changes have been made.
- **Relies On:**
  When package A relies on package B, package A will only be rebuilt if the configuration of package B changes.
- **Upgrade:**
  When a newer version of a package has been installed, we call this an upgrade.
- **Upwards (dependency):** A dependency relation upwards (in the tree) means that the package depends on the one higher in the tree.