Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б07-мм

Проектирование и создание системы для анализа зависимостей пакетов в дистрибутивах GNU/Linux

БУРАШНИКОВ Артем Максимович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ст. преп. каф. ИАС К. К. Смирнов

Санкт-Петербург
2024

Оглавление

Введение

Во время работы с операционной системой установка, обновление и управление программами являются неотъемлемой частью пользовательского опыта. В ОС семейства Linux эти процедуры происходят с использованием **пакетов** [1]. Пакетом называется архив специального формата, который содержит необходимые бинарные и конфигурационные файлы, список действий, выполняемых во время установки, а также любую другую важную информацию, в том числе данные о зависимостях. При этом в каждый такой архив включен только необходимый минимум компонент, остальные указываются в качестве зависимостей. Например, в один могут быть вложены бинарные файлы, во второй — общие библиотеки, в третий — документация.

Linux состоит из ядра и сопутствующего ему программного обеспечения, вместе называемых **дистрибутивом**. Несмотря на то, что дистрибутивы формируются примерно из одного и того же набора популярных программ и полезных библиотек, сами пакеты, их формат и зависимости могут различаться. Например, в CentOS пакет **openldap** зависит от **openssl**, а в Ubuntu — от **gnutls**. Кроме того, для одного конкретного дистрибутива набор зависимостей у некоторого пакета может оказаться другим при использовании системы с другой архитектурой процессора, причем особенности могут быть как в компонентах, так и в версиях одной и той же компоненты.

Эти и другие характеристики учитываются при проведении анализа пакетов и их метаданных в Linux между разными архитектурами и дистрибутивами. Некоторыми причинами и целями такого анализа являются следующие пункты.

1. *Совместимость и переносимость.* Анализ зависимостей помогает при разработке совместимого и переносимого между устройствами программного обеспечения.

2. *Оптимизация и эффективность.* Оптимизированные версии программ могут быть разработаны для конкретных процессоров. Ана-

лиз зависимостей позволяет определить, какие компоненты и где требуют оптимизаций.

3. *Специализированные решения.* В некоторых случаях различные архитектуры и дистрибутивы могут использоваться для специализированных задач (например, встроенные системы, или высокопроизводительные вычисления). Анализ пакетов способствует адаптации системы к конкретным требованиям и использованию специализированных компонент.

Ввиду отсутствия единого формата архивов и их метаданных, существенных различий между дистрибутивами и используемыми ими пакетными менеджерами проведение глубокого межархитектурного и междистрибутивного анализа является актуальной проблемой. В рамках семестровой практики научным руководителем была поставлена задача спроектировать и создать на языке Python систему, которая бы помогла её решению.

1 Постановка задачи

Целью работы является создание системы, которая помогает проводить междистрибутивный и межархитектурный анализ пакетов и их зависимостей в операционных системах Linux, причем особое внимание должно быть уделено архитектуре процессора RISC-V, чья экосистема является приоритетной. Для её выполнения были поставлены следующие задачи:

1. провести обзор инструментов, позволяющих анализировать метаданные пакетов с целью выбора функциональности создаваемого инструмента;

2. проведя анализ форматов метаданных, выбрать два дистрибутива, для которых обозначенная функциональность будет реализована;

3. реализовать приложение на языке Python, учитывая требуемую функциональность и особенности выбранных дистрибутивов.

2 Обзор предметной области

Для проектирования системы с целью выявления желаемой функциональности необходимо ознакомиться с инструментами, предоставляющими анализ пакетов. Кроме того, нужно выбрать первоначальные дистрибутивы и проанализировать форматы, в которых представлены метаданные. В конце раздела указаны использованые сторонние библиотеки.

2.1 Существующие аналоги

В конкретном дистрибутиве используется своё программное обеспечение, позволяющее провести требуемый анализ. Кроме этого интерес представляли приложения, занимающиеся агрегаций информации о пакетах. Четыре инструмента, представленные в дальнейших секциях, были отобраны по следующим критериям:

- активно поддерживается разработчиками;
- родной для дистрибутива или кросс-платформенный;
- может выводить информацию о метаданных для нескольких архитектур и дистрибутивов.

2.1.1 `Apt`

`Apt`[1] — это пакетный менеджер, по умолчанию поставляемый вместе с дистрибутивом Debian и производными системами. Согласно документации, `apt` является высокоуровневой оболочкой для других инструментов, поэтому для просмотра предоставляемой функциональности нужно обратиться к `apt-cache`[2]. Следующие возможности утилиты согласуются с целью работы:

[1] Документация apt. Дата обращения: 13 декабря 2023 г.
[2] Документация `apt-cache`. Дата обращения: 13 декабря 2023 г.

1. `depends` отображает список каждой зависимости, которую имеет пакет, а также все возможные другие пакеты, которые могут удовлетворить эту зависимость;

2. `rdepends` отображает список пакетов, для которых данный является зависимостью;

3. `pkgnames` выводит имена пакетов, о которых знает `apt-cache`.

Одним из существенных минусов `apt` является то, что программу имеет смысл использовать только на производных Debian.

2.1.2 `Pactree`

`Pactree`[3] — утилита для дистрибутива Arch Linux, позволяющая выводить на экран деревовидную структуру зависимостей для пакетов. Из функциональности отмечаются те же пункты, что и для `apt`. Дополнительно имеется возможность вывода результата запроса в формате `Graphviz`[4] для последующей визуализации полученного графа.

Минус `pactree` заключается в том, что на момент написания работы Arch Linux официально не поддерживает набирающую популярность и активно развивающуюся архитектуру RISC-V [2].

2.1.3 `Debtree`

`Debtree`[5] — инструмент для Debian и производных систем. Функциональность сосредоточена на построении графа зависимостей. Выводит результет на языке `dot`, читаемом утилитой `Graphviz`. Возможности аналогичны уже обозначенным приложениям, однако обладает дополнительными опциями для построения графов (например, цвет, глубина и т.д.). Отдельно стоит отметить возможность при вызове команды добавить опциональный аргумент **arch**, указав архитектуру, зависимость пакета на которой требуется показать.

[3] Документация pactree. Дата обращения: 13 декабря 2023 г.
[4] Библиотека Graphviz для визуализации графов
[5] Документация debtree на ресурсе Ubuntu. Дата обращения: 13 декабря 2023 г.

К сожалению, `Debtree` можно использовать только на производных Debian. Кроме того, несмотря на то, что можно указать архитектуру, без дополнительных системных настроек будет использована архитектура системы, на которой запущено приложение, вне зависимости от того, какая архитектура указана аргументом.

2.1.4 `Repology`

`Repology`[6] в отличие от уже рассмотренных приложений напрямую не позиционируется как система анализа пакетов. Проект является агрегатором репозиториев и смежных ресурсов, предоставляющих установочные архивы. `Repology` имеет обширный список возможностей для анализа версий библиотек, наличия или отсутствия их в репозиториях. Согласно описанию, сервис отслеживает и позволяет сравнивать версии пакетов из более чем 120 репозиториев. Хотя `Repology` и не предоставляет никакой логики для сравнения зависимостей, особое внимание уделено некоторым аспектам технической реализации проекта в целом, потому что они интересны для проектируемой в рамках семестровой работы системы.

Во-первых, сервис представляет собой веб-приложение, разделенное на несколько независимых компонент: **webapp**[7], **updater**[8] и **ruleset**[9]. Во-вторых, сами метаданные скачиваются из архивов соответствующих дистрибутивам репозиториев (те же репозитории использует, например `apt`), затем сервис производит парсинг загруженных метаданных и сохранение их в базах данных SQLite. Указанные особенности реализации оказались интересны для проектируемой системы, так как позволили внедрить модульный дизайн и обеспечить переиспользование полученной из репозиториев информации об архивах.

[6] Информация о ресурсе Repology. Дата обращения: 13 декабря 2023 г.
[7] Репозиторий компоненты webapp сервиса Repology. Дата обращения: 13 декабря 2023 г.
[8] Репозиторий компоненты updater сервиса Repology. Дата обращения: 13 декабря 2023 г.
[9] Репозиторий компоненты ruleset сервиса Repology. Дата обращения: 13 декабря 2023 г.

2.2 Результаты анализа аналогов

В таблице 1 представлены рассмотренные инструменты и основная функциональность каждого из них.

	Apt	Pactree	Debtree	Repology
Прямые зависимости	✓	✓	✓	✓
Обратные зависимости	✓	✓	✓	✓
Сравнение метаданных	✗	✗	✗	✗
Построение графа	✗	✓	✓	✗
Кросс-платформенность	✗	✗	✗	✓
Интерфейс	CLI	CLI	CLI	REST API

Таблица 1: Сравнение инструментов.

Таким образом, вывод для пакета его прямых и обратных метаданных — основная функция, присущая всем утилитам такого рода. Далее, не все рассмотренные приложения могут рисовать полный граф зависимостей и никакая из утилит не может напрямую проводить сравнение метаданных пакетов между различными архитектурами ни в рамках одного дистрибутива, ни между пакетами в рамках различных дистрибутивов. Так, в основу созданного инструмента заложены следующие особенности:

1. сравнение по прямым зависимостям пакетов между различными архитектурами в рамках одного дистрибутива;
2. вывод имен пакетов и архитектур, для которых можно произвести сравнение;
3. архитектура приложения должна быть гибкой и позволять расширять фукциональность, используя не только зависимости, но и другие потенциально значимые метаданные;
4. агрегация пакетов из различных дистрибутивов;
5. пользовательский интерфейс;
6. кросс-платформенность инструмента.

2.3 Формат пакетов в Linux

Для дистрибутивов Linux существуют две основные категории пакетов: `rpm` и `deb` [3]. Пакеты формата `rpm` в основном используются с Red Hat, Fedora, Suse и производными дистрибутивами в то время как пакеты `deb` применяются для Debian и его производных. И `rpm`, и `deb` представляют собой сжатые архивы, содержащие все необходимые данные для корретного использования системой. Кроме того, они включают метаданные, описывающие атрибуты и требования, касающиеся окружения, в котором установливается или настраивется пакет. Пакеты `rpm` кодируют спецификации в бинарной форме и включают их в состав своих архивных файлов, в то время как для `deb` используется текстовое представление, что делает их более удобными для обработки. Несмотря на указанные различия, спецификации зависимостей в метаданных для обоих типов пакетов практически идентичны.

2.4 Спецификация зависимостей

Прежде всего необходимо понимать отношения, используемые между пакетами[10]. Ниже на примере спецификации формата метаданных в дистрибутиве Debian приведен список таких отношений с пояснениями.

1. *Depends.*
   Пакет **A** не будет настроен, пока все пакеты, перечисленные в этом поле не будут корректно настроены. Пакет может находиться в системе в “не настроенном” состоянии.

2. *Recommends.*
   Для пакета **A** в этом поле перечислены пакеты, которые обычно устанавливаются вместе с **A**.

3. *Suggests.*
   Перечисленные в этом поле пакеты связаны с пакетом **A** и, воз-

[10] Руководство Debian. Дата посещения 14 декабря 2023 г.

можно, могут улучшить его полезность, но установка только **A** без них разумна и ничем не ограничена.

4. *Enhances*.
   Работает как *Suggests*, но в обратную сторону: в этом поле указаны пакеты, для которых пакет **A** попадает в *Suggests*, потенциально улучшая их полезные свойства.

5. *Pre-Depends*.
   Означает почти то же, что и *Depends*, но дополнительно требует завершения установки указанных в поле пакетов до установки пакета **A**.

6. *Conflicts*.
   Пакеты, указанные в этом поле, не могут существовать в системе одновременно с пакетом **A**.

7. *Provides*.
   В этом поле указаны виртуальные пакеты[11], которые предоставляются пакетом **A**.

8. *Replaces*.
   Если пакет **B** указан в этом поле для пакета **A**, то **B** должен быть удалён, прежде чем будет установлен **A**.

Понимание того, какие взаимотношения встречаются между пакетами, позволило однозначно истолковывать те ситуации, когда поля метаданных в различных дистрибутивах не совпадают. Так, для пакетов формата `rpm` поле *Depends* встречается в виде *Requires*, но по своей сути означает одно и то же.

2.5 Выбор дистрибутивов

Выбраны две операционые системы, чьи особенности на первичном этапе учтены при разработке. Обе ОС поддерживают RISC-V и входят в

[11] Виртуальные пакеты в руководстве Debian. Дата посещения 14 декабря 2023 г.

десятку самых популярных, согласно данным портала DistroWatch[12]. Релизы (англ. — **Release**) дистрибутивов и так называемые **upstream** репозитории, содержащие пакеты и их метаданные, кроме RISC-V, выбраны произвольно, так как их выбор не влияет на функциональность разработанной утилиты. Учтена возможность расширить приложение добавлением других релизов, репозиториев и дистрибутивов.

Ubuntu Так как разработка проекта велась на системе, на которой установлен Ubuntu, синтаксис метаданных этого дистрибутива представляет собой упрощенный вариант обобщенного синтаксиса Debian, а сами метаданные в `upstream` репозитории лежат в текстовом виде, то он был выбран в качестве первого. Внедрение Ubuntu помогло создать прототип требуемого приложения и консольный интерфейс для конечного пользователя.

Fedora Вторым дистрибутивом выбрана Fedora, потому что метаданные для этой системы в `upstream` репозитории предоставлены в виде базы данных SQLite, что существенно упростило процесс внедрения. Кроме того Fedora использует пакеты `rpm`, в то время как Ubuntu — `deb`. Таким образом, выбор данного дистрибутива позволил создать приложение, способное обращаться с несовместимыми форматами метаданных различных дистрибутивов.

Проведенный обзор инструментов, формата метаданных и выбор дистрибутивов помогли спроектировать требуемый инструмент.

2.6 Использованные технологии

Написание утилиты на языке Python входило в производственное задание. Для поддержания единого стиля кода использован форматтер **black**[13] и линтер **ruff**[14], выбранный как альтернатива **flake8**[15]. `Ruff`

[12] Главная страница портала DistroWatch с рейтингами дистрибутивов. Дата посещения 14 декабря 2023 г.

[13] Главная страница документации black

[14] Главная страница документации линтера ruff

[15] Главная страница документации линтера flake8

применяет тот же набор правил, что и `flake8`, и по сути ничем не отличается, но не требует дополнительной настройки для совместимости с `black`. Кроме того, было принято решение использовать статическую типизацию функций, которая соответствует «best practices», для чего подключен **mypy** с аргументом **strict**.

Для управления сборкой и зависимостями проекта выбран `Poetry`[16]. Этот выбор обладает рядом следующих преимуществ.

- **Декларативное управление зависимостями**. Зависимости и их версии описываются в файле **pyproject.toml**. `Poetry` самостоятельно следит, чтобы указанные библиотеки были совместимы друг с другом.
- **Виртуальное окружение**. Автоматически создается виртуальное окружение для проекта, изолируются его зависимости от системных.
- **Интеграция с инструментами сборки и публикации**. `Poetry` поддерживает не только управление зависимостями и сборку проекта, но также включает в себя функциональность для создания, установки и публикации пакетов.

Далее рассмотрены детали реализации

[16] Главная страница Poetry

3 Детали реализация

В данном разделе затронуты вопросы, касаемые архитектуры приложения, а также отмечены нетривиальные моменты и решения.

3.1 Базы данных

Рассматривался вариант создать приложение, которое ищет информацию о пакете на соответствующем ресурсе в интернете, но такое решение оказалось нецелесообразным ввиду того, что количество запросов заранее не ясно — внешние сервисы, предоставляющие `API`, почти всегда имеют ограничение на количество обращений к ним. Кроме этого, работа приложения зависела бы от работы стороннего веб-сервиса, вдобавок — не все из потеницально полезных порталов в принципе имеют публичный `API`. Так, потребовалось реализовать возможность сохранения данных на диске для последующего переиспользования. Помимо этого, хотелось уметь менять стилистику запросов к метаданным, добавляя какие бы то ни было дополнительные условия или ограничения. Использование баз данных удовлетворяет всем требованиям. Более того, из обзора сделан вывод, что метаданные Fedora лежат в `upstream` в формате `.sqlite`, а в Python имеется встроенная библиотека для импорта с одноименным названием.

Применение `SQLite` позволило упростить процесс внедрения метаданных Fedora, добавило возможность использовать метаданные, хранящиеся на диске, а не в памяти компьютера или на веб-ресурсе, и позволило реализовать создание нетривиальных запросов. Для каждого дистрибутива и релиза используются свои файлы баз данных, причем для Ubuntu доступные в рамках релиза архитектуры находятся в одной базе, которая создаётся и заполняется самим приложением, а для Fedora для каждой комбинации «релиз-архитектура» загружается отдельная база и никак не меняется.

Схема базы данных пакетов Ubuntu, представленная на рисунке 1, является отражением схемы баз метаданных Fedora. Многоточием обозначены пропущенные таблицы, которые полностью дублируют

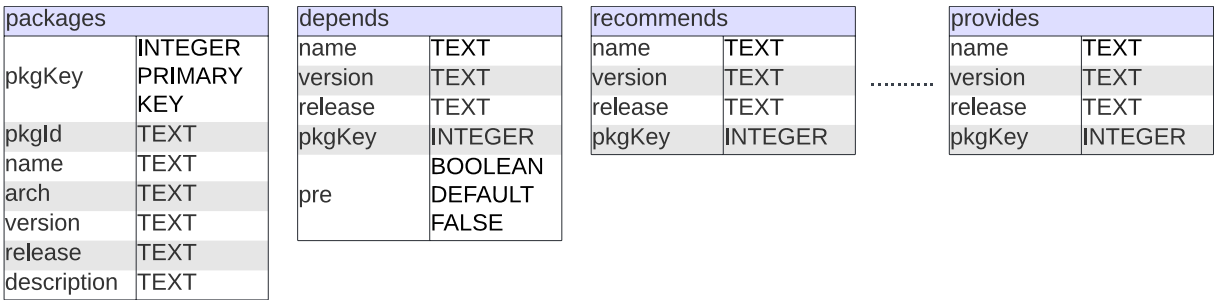

Рис. 1: Схема базы данных пакетов Ubuntu.

схемы между `recommends` и `provides`. Всего таблиц столько, сколько отношений между пакетами.

3.2 API

Реализованная утилита предоставляет конечному пользователю консольное приложение, главное меню которого продемонстрирован на рисунке 2.

```
Usage: depinspect [OPTIONS] COMMAND [ARGS]...

Options:
  --help                Show this message and exit.

Commands:
  diff                  Compare two packages.
  find-divergent        List all packages that have divergent dependencies.
  list-all              List stored architectures and packages for a given distro.
  update                Update metadata stored in databases.
```

Рис. 2: Вид консольного приложения.

Написание `CLI` требует значительно меньше времени, чем написание графического интерфейса, позволяет относительно просто расширять функциональность через добавление новых команд, и соответствут тому интерфейсу, который предоставляют рассмотренные ранее схожие инструменты. Для внедрения консольного приложения использована библиотека `Click`[17], распространяемая под лицензией `BSD-3-Clause`. Из преимуществ библиотеки можно отметить исчерпывающую документа-

[17] Главная страница библиотеки Click

цию с примерами использования и обширную опциональность, что способствовало реализации простого и понятного интерфейса.

3.3 Архитектура

На рисунке 3 представлены модули, связанные отношением «импортирует».

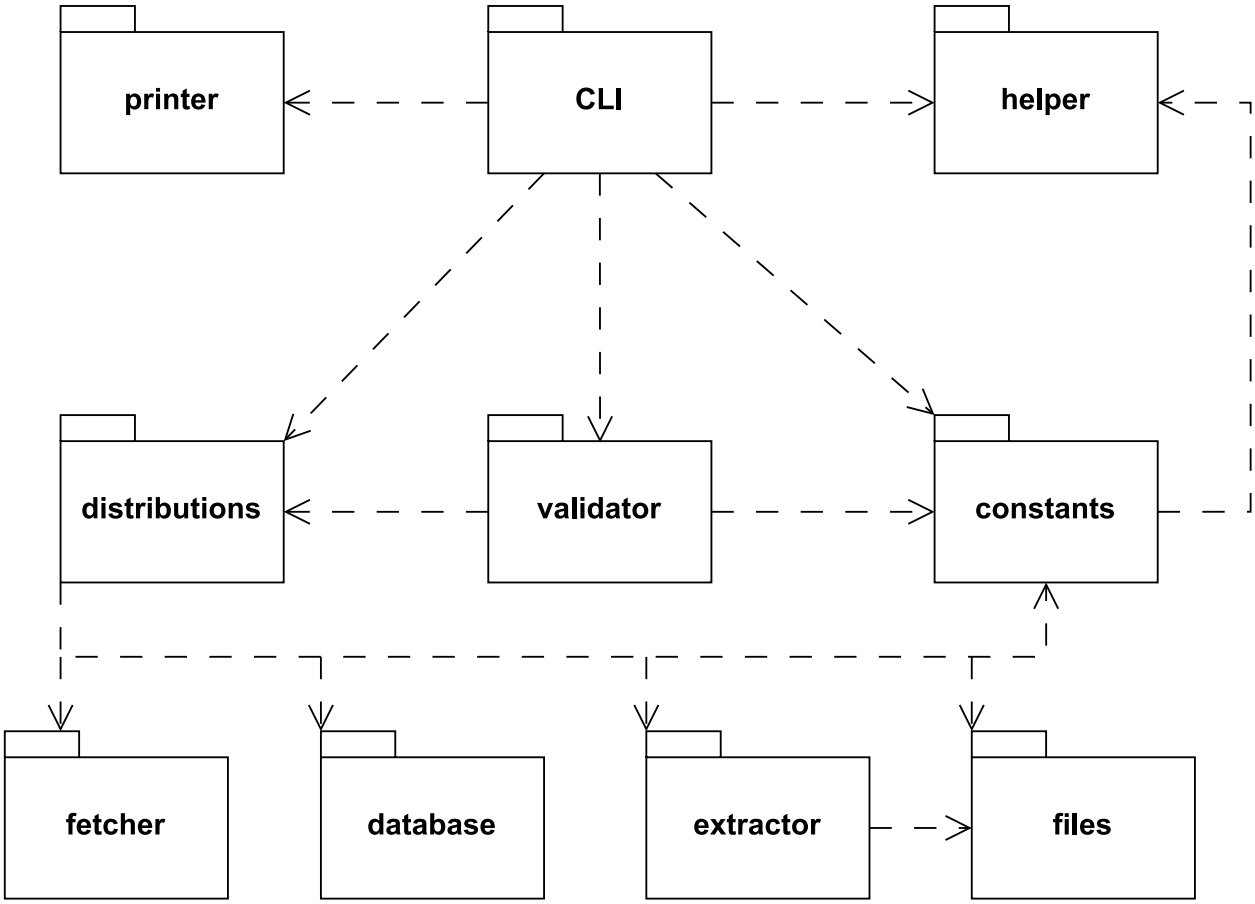

Рис. 3: UML диаграмма используемых в приложении модулей.

Модуль `distributions` собрал в себе несколько подмодулей. Детализированный вид `distributions` изображен на рисунке 4.

Приложение можно разделить на отдельные, независимые друг от друга компоненты, последовательная работа которых приводит к исполнению введенной пользователем команды, что напоминает архитектурный стиль «Каналы и фильтры». Команда от пользователя обрабатывается компонентой **CLI**, которая является точкой входа в программу, предоставляя внешний интерфейс для работы с приложением.

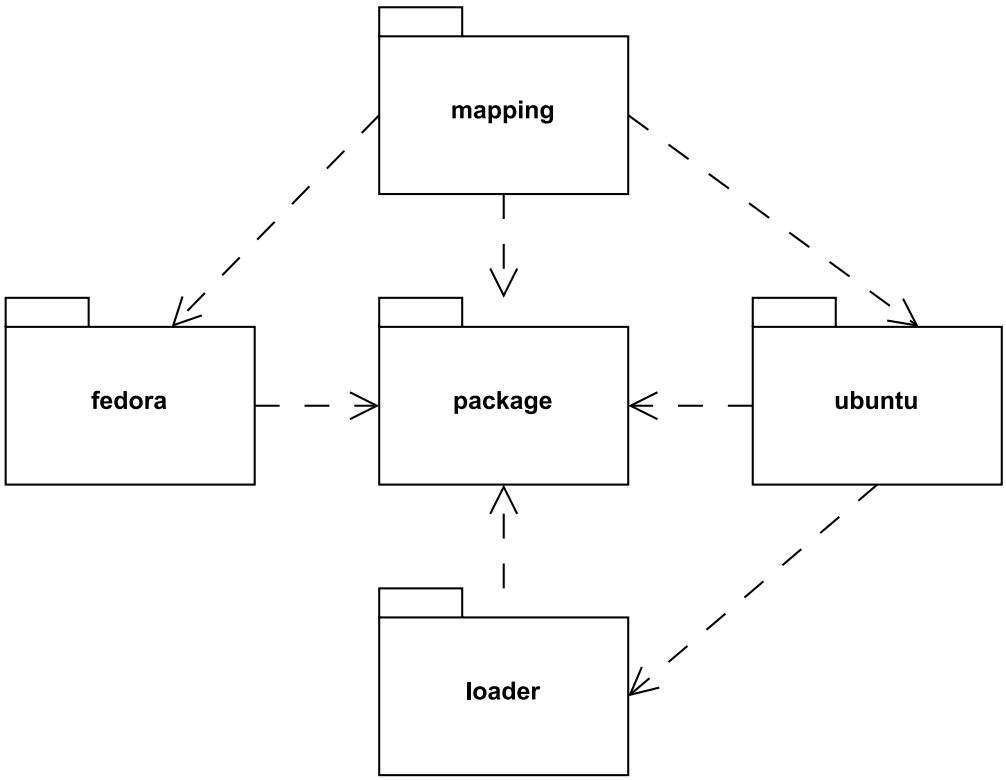

Рис. 4: UML диаграмма подмодулей модуля `distributions`.

Получив на вход команду с пользовательскими данными, `CLI` проводит валидацию параметров с помощью функций в модуле `validator`. В параметрах почти всегда передается **distro** — название дистрибутива. После валидации происходит сопоставление через модуль `mapping` этого параметра с одноименным классом. Через статические методы уже определенного класса происходит дальнейший вызов функций с нужными классу параметрами.

Кроме `CLI` в работе приложения задействованы следующие условные компоненты.

- **Fetcher**, отвечающий за загрузку метаданных, указанных в конфигурационном файле проекта `pyproject.toml`.
- **Extractor**, отвечающаий за обработку загруженных разного рода архивов, содержащих метаданные.
- **Database**, реализующий абстракцию над запросами в базу данных с помощью языка запросов `SQL`.

- **Printer**, собравший функции, реализующие вывод результата в консоль.

Созданный «toolchain» позволяет независимо изменять каждую из компонент, минимально затрагивая остальные.

3.4 Дистрибутивы как классы

Во время поступления команды от пользователя заранее неизвестно, какой порядок действий нужно совершить. Так, для инициализации метаданных Ubuntu нужно выполнить целую цепочку (загрузить архивы, произвести парсинг текста, создать и наполнить базу данных), в то время как для метаданных Fedora — загрузить и распаковать файлы `.sqlite`. Таким образом, создан метакласс **Package**, имеющий набор полей, соответствующих именам полей метаданных, и декларирующий абстрактные методы, сопоставляемые консольным командам, которые реализованы в наследующихся от **Package** дочерних классах **Ubuntu** и **Fedora**. Например, метод `init` у **Ubuntu** запускает требуемую последовательность из `fetch`, `extract`, `parse`, `load`, а для **Fedora** — только `fetch` и `extract`, но с другими параметрами. Данная идея — каждый класс знает, что с собой нужно делать — применяется для всех созданных пользовательских команд и соответствующих им статических методов.

Заключение

В рамках выполнения данной работы были получены следующие результаты:

- проведён обзор четырех инструментов, способных работать с метаданными пакетов, на основе анализа функциональности которых созданы четыре доступные для конечного пользователя команды приложения;
- для дистрибутивов Ubuntu и Fedora реализована отобранная функциональность;
- создано приложение[18] с модульной архитекторой и консольным интерфейсом, позволяющее производить межархитектурный анализ зависимостей пакетов и включающее несколько дистрибутивов.

[18]Репозиторий приложения, https://github.com/artem-burashnikov/depinspect

Список литературы

[1] Garrels M. Introduction to Linux (Second Edition). Fultus technical library. — Fultus Corporation, 2007. — ISBN: 9781596821125. — URL: https://books.google.ru/books?id=Vox6vvQxt0QC.

[2] A Survey of the RISC-V Architecture Software Support / Benjamin W. Mezger, Douglas A. Santos, Luigi Dilillo et al. // IEEE Access. — 2022. — Vol. 10. — P. 51394–51411.

[3] A graph method of package dependency analysis on Linux Operating system / Jing Wang, Qingbo Wu, Yusong Tan et al. // 2015 4th International Conference on Computer Science and Network Technology (ICCSNT). — Vol. 01. — 2015. — P. 412–415.