Федеральное государственное бюджетное образовательное учреждение   
высшего образования

|  |  |
| --- | --- |
| Gerb-BMSTU_01 | «Московский государственный технический университет  им. Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана) |

ФАКУЛЬТЕТ – Информатика и управления

КАФЕДРА – Информационные системы и телекоммуникации

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине

Разработка программного обеспечения

на тему

Разработка OSGI сервиса мониторинга сетевых таблиц маршрутизации

Студент группы ИУ3-73 (подпись) 23.02.2018 Д.М. Старченко

Руководитель курсовой работы (подпись) 23.02.2018 А.М Иванов

Москва, 2018

Реферат

В работе предложена архитектура OSGI компонента мониторинга сетевых таблиц маршрутизации. Архитектура реализована с использованием объектно-ориентированного языка программирования Java. В целях проверки работоспособности программного обеспечения разработана методика тестирования с помощью библиотеки JUnit. Результаты проведенных тестов доказывают функциональность OSGI компонента.

**Содержание**

[Техническое задание 5](#_Toc506539821)

[1 Теоретическая часть 6](#_Toc506539822)

[1.1 Таблица маршрутизации 6](#_Toc506539824)

[2 Конструкторская часть 8](#_Toc506539825)

[2.1 Выбор технических решений, удовлетворяющих интересам ЗС 8](#_Toc506539827)

[2.2 Диаграмма компонентов 9](#_Toc506539832)

[2.3 Диаграмма классов 10](#_Toc506539832)

[2.4 Структура проекта 10](#_Toc506539828)

[2.4.1 Плагин com.bmstu.route.table 10](#_Toc506539829)

[2.4.2 Плагин com.bmstu.route.table.client 12](#_Toc506539830)

[2.4.3 Плагин com.bmstu.route.table.service 12](#_Toc506539831)

[2.5 Принцип работы разработанного ПО 13](#_Toc506539832)

[3 Технологическая часть 14](#_Toc506539833)

[3.1 Запуск разработанного приложения 14](#_Toc506539834)

[3.2 Анализ искходного кода с помощью метрик качества 14](#_Toc506539835)

[3.3 Анализ зависимостей в коде системы 15](#_Toc506539836)

[3.4 Тестирование на корректность работы 16](#_Toc506539837)

[3.5 Реализация Junit тестов 17](#_Toc506539838)

[Заключение 18](#_Toc506539840)

[Список литературы 19](#_Toc506539841)

# Техническое задание

Разработка OSGI сервиса мониторинга сетевых таблиц маршрутизации:

* изучить соответствующие системы;
* спроектировать интерфейс компонента;
* реализовать компонент
* спроектировать JUnit тесты, провести тестирование;
* описать требования, конструкцию, особенности сборки и запуска в документации.

# 1 Теоретическая часть

1.1 Таблица маршрутизации

Таблица [маршрутизации](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) — электронная таблица ([файл](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB)) или [база данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), хранящаяся на [маршрутизаторе](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80) или [сетевом компьютере](https://ru.wikipedia.org/wiki/%D0%A3%D0%B7%D0%B5%D0%BB_%D1%81%D0%B5%D1%82%D0%B8), которая описывает соответствие между адресами назначения и [интерфейсами](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81), через которые следует отправить пакет данных до следующего маршрутизатора. Является простейшей формой правил маршрутизации.

Таблица маршрутизации обычно содержит:

* Адрес сети или узла назначения, либо указание, что маршрут является маршрутом по умолчанию;
* [Маску сети назначения](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D0%BA%D0%B0_%D0%BF%D0%BE%D0%B4%D1%81%D0%B5%D1%82%D0%B8) ;
* Шлюз, обозначающий адрес маршрутизатора в сети, на который необходимо отправить пакет, следующий до указанного адреса назначения;
* Интерфейс, через который доступен шлюз;
* Метрику.

Манипуляции с таблицей маршрутизации позволяют тонко настраивать работу ваших [сетей](http://it.sander.su/IP.php). Для просмотра таблицы маршрутизации достаточно ввести в консоль нужную команду(зависит от вашей ОС). Пример таблицы маршрутизации на стационарном компьютере изображен на рисунке 1.

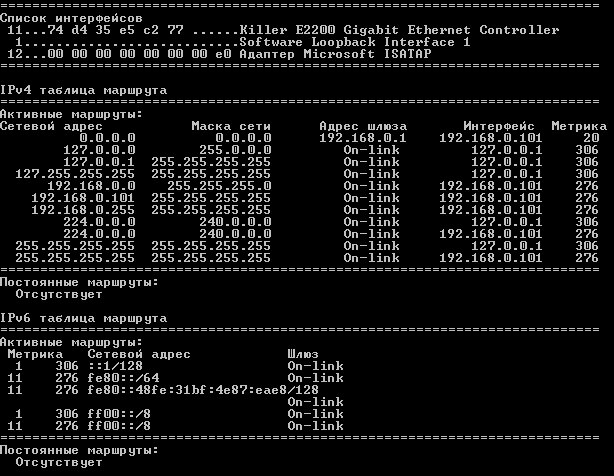


Рисунок 1 – Пример таблицы маршрутизации

Каждая строчка определяет, куда отправлять какие пакеты. То есть для диапазона, задаваемого значениями в колонках "сетевой адрес" и "маска сети" создается сетевой маршрут. Например, адрес 192.168.0.1 и маска 255.255.255.0 означают, что имеется в виду диапазон 192.168.0.\*. Маска всегда имеет вид, когда вначале стоят 255, в конце - нули, а последним ненулевым числом может быть степень двойки минус один. Например, для маски 255.255.127.0 и того же адреса 192.168.0.1 диапазон будет чуть шире, в него войдут и адреса вида 192.168.1.\*.

Третья колонка определяет "шлюз" - тот маршрутизатор, которому нужно послать эти пакеты. В случае, когда там написано "On-link", имеется в виду, что никаких маршрутизаторов не нужно - адрес и так находится в прямой досягаемости. Последняя колонка - метрика. Она определяет предпочтение для маршрута, когда есть варианты. Строчки с наименьшей метрикой предпочтительны при совпадении диапазонов.

Команды для просмотра таблицы маршрутизации, как было сказано ранее, разные, зависит от того, какая операционная система установлена на компьютере.

Примеры таких команд:

1. «netstat –rn» (Таблица маршрутизации в UNIX)
2. «route -n», «ip route» , а так же напрямую просмотрев файл «/proc/net/route» (Таблица маршрутизации в Linux)
3. «router# show ip route» (Таблица маршрутизации в маршрутизаторах Cisco)
4. «ciscoasa# show route»(Таблица в устройствах Cisco ASA)
5. «route print» (Windows)

Так же ОС влияет на формат вывода таблицы. Таблица маршрутизации на ОС Linux выглядит следующим образом:

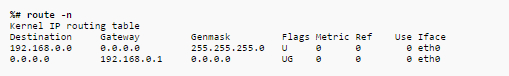


Рисунок 2 – Таблица маршрутизации Linux

И наконец, одна из основополагающих утилит маршрутизации — команда «route». С ее помощью пользователь может прописать, удалить или отредактировать все статические маршруты на используемом компьютере:

1. Добавление маршрута осуществляется через команду: «**route add ip mask gateway metric x if y»,** где ip — адрес или сеть назначения, mask — маска подсети, gateway — шлюз, через который пакет будет идти к месту назначения, x — числовое значение метрики маршрута, y — порядковый номер интерфейса.
2. Удаление маршрута происходит через команду: «**route delete ip»**, где ip — адрес назначения, прописанный в таблице маршрутизации.
3. Модификация маршрута производится по команде: «**route change ip mask gateway metric x if y»**. Модификация маршрута может производиться только в случае смены шлюза или/и метрики интерфейса.
4. Конструкторская часть

2.1 Выбор технических решений, удовлетворяющих интересам ЗС

В таблице ниже представлены результаты выявления и начального анализа заинтересованных сторон (ЗС) и их интересов по отношению к системе.

Таблица 1 – Заинтересованные стороны и их интересы по отношению к системе

|  |  |
| --- | --- |
| **Интересы заинтересованных сторон** | **Технические решения** |
| Отслеживание событий изменения состояния параметров сетевых таблиц маршрутизации | При любом изменении параметров, а также состояния сетевой таблицы маршрутизации, данное событие обрабатывается одним из классов разработанной системы.  Корректность работы будет проверяться в процессе разработки с помощью юнит тестов. |
| Быстрая и полная передача исходного кода, настроек, документов.  Возможность в дальнейшем совершенствовать систему, например, добавляя новые модули и возможности | Код и настройки разрабатываемого менеджера будут находиться в системе GitHub. Контроль версий будет производиться с использованием системы Git.  Для модульного тестирования будет использована система JUnit. |
| Быстрое внесение изменений и проверка, что они не нарушают работу существующего функционала. | Исходный код будет структурирован по пакетам. Повторно-используемые методы будут вынесены в родительские классы. |

2.2 Диаграмма компонентов

На рисунке 3 представлена диаграмма компонентов разработанного программного обеспечения. При активации сервиса, при помощи аннотации «@activate» происходит первоначальная загрузка таблицы в сервис. В классе RouteTableService содержится метод «CreateChangeEvent», который создает события изменения. Сервис отслеживает данные события и рассылает их клиентам, обновляя перед этим таблицу.

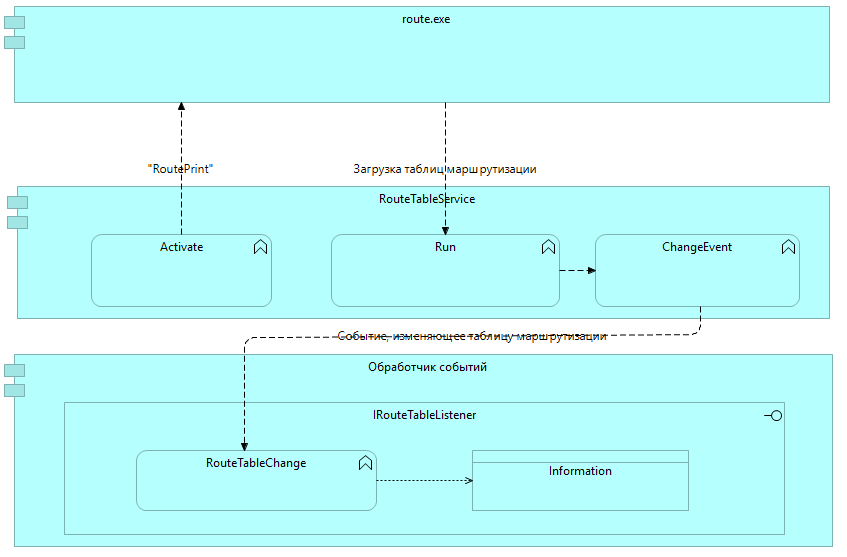


Рисунок 3 – Диаграмма зависимостей компонентов

2.3 Диаграмма классов

Для наглядности, с помощью редактора UML Papyrus была создана диаграмма классов. Весь проект состоит из четырех пакетов. Функционал всех пакетов и их компонентов будет описан ниже.

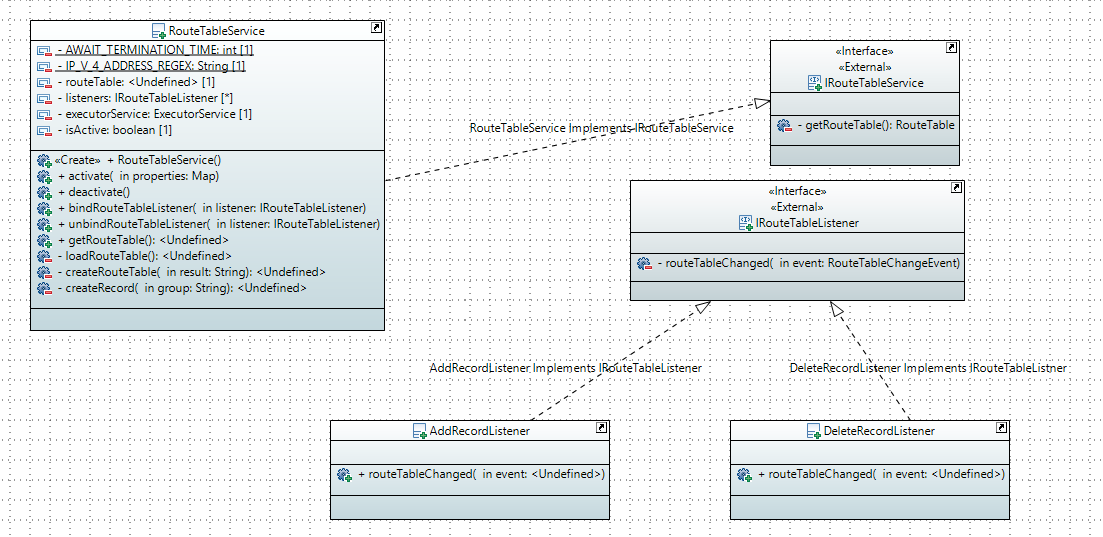


Рисунок 4 – Диаграмма классов

2.4 Структура проекта

Разберем работу написанной системы, для этого опишем состав каждого плагина, из которых состоит наша система.

Таблица 2 – Разработанные плагины и их описание

|  |  |
| --- | --- |
| **Плагин** | **Содержимое** |
| com.bmstu.route.table | содержит интерфейсы для сервисов и клиентов разрабатываемого ПО |
| com.bmstu.route.table.client | содержит классы, являющиеся клиентами разрабатываемого ПО |
| com.bmstu.route.table.service | содержит класс, являющийся сервисом разрабатываемого ПО |

2.4.1 Плагин com.bmstu.route.table

Данный плагин содержит 2 пакета:

* com.bmstu.route.table;
* com.bmstu.route.table.model.

Пакет com.bmstu.route.table содержит интерфейсы IRouteTableListener, IRouteTableService, которые включают в себя 1 метод:

Таблица 3 – Методы интерфейса IRouteTableListener и их описание

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| void routeTableChanged(RouteTableChangeEvent event); | Добавляет событие, содержащее информацию о добавлении, изменении и удалении информации о сетевых таблицах маршрутизации. |

Таблица 4 – Методы интерфейса IRouteTableService и их описание

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| RouteTable getRouteTable(); | Возвращает текущую таблицу маршрутов системы |

Пакет com.bmstu.route.table.model содержит 3 класса

* RouteTable.java
* RouteTableChangeEvent.java
* RouteTableRecord.java

Таблица 5 – Классы пакета com.bmstu.route.table.model и их описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| RouteTable | Public RouteTable – Возвращает записи таблицы маршрутов.  Private Collection<RouteTableRecord> records- Конструктор.  Public Collection<RouteTableRecord> getRecords()-возвращает записи таблиц маршрутизации.  Public String toString()- возвращает строку с информацией(Network , Netmask и т.д.)  public int hashCode() – возвращает hash код объекта  public boolean equals(Object object) – проверяет объекты на равнозначность |
| RouteTableChangeEvent | Public RouteTableChangeEvent-Экземпляр этого класса представляет событие изменения таблицы маршрутов. Содержит удаленные и добавленные записи.  Public Collection <RouteTableRecord> getDeleteRecords - удаление записи.  Public Collection <RouteTableRecord> getAddedRecords – добавляет запись.  Public RouteTable getRouteTable() – возвращает измененную таблицу маршрутов. |
| RouteTableRecord | RouteTableRecord- представляет запись таблицы маршрутизации.  PublicTableRecord(…) – Конструктор  Public String getWebAddress()- получение Web Address  Public String getWebMask()- получение WebMask  Public String getGateAddress() – получение gate Address  Public String getInterface() – получение interface  Public int getMetric() – получение metric  Public String toString()- возвращает строку с информацией(Network , Netmask и т.д.)  public int hashCode() – возвращает hash код объекта  public boolean equals(Object object) – проверяет объекты на равнозначность |

2.4.2 Плагин com.bmstu.route.table.client

Данный плагин содержит 1 пакет:

* com.bmstu.route.table.client

Пакет com.bmstu.route.table.client содержит 2 класса:

* AddRecordListener;
* DeleteRecordListener.

Таблица 6 – Классы пакета com.bmstu.route.table.client и их описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| AddRecordListener | public void routeTableChanged(RouteTableChangeEvent event)– выводит информацию о добавлении сетевых таблиц маршрутизации, если есть такие. |
| RemoveAdapterListener | public void routeTableChanged(RouteTableChangeEvent event)– выводит информацию об удалении сетевой таблицы маршрутизации, если такая есть. |

2.4.3 Плагин com.bmstu.route.table.service

Данный плагин содержит 1 пакет:

* com.bmstu.route.table.service

Пакет com.bmstu.route.table.service содержит 1 класс:

* RouteTableService.

Таблица 7 – Классы пакета com.bmstu.route.table.service и их описание

|  |  |
| --- | --- |
| Класс | Состав и описание |
| RouteTableService | Public RouteTableService() – Конструктор  private static final String IP\_V\_4\_ADDRESS\_REGEX = "([\\d+\\.){3}\\d](file:///\\d+\\.)%7b3%7d\\d)+" – Регулярное выражение, определяет шаблон поиска для строк.  Public void activate(Map<String, Object> properties)- Загружает таблицу маршрутизации и запускает процесс мониторинга за изменениями в таблице маршрутизации.  Public void deactivate() – Отключает процесс мониторинга при деактивации сервиса.  Public void bindRouteTableListener(IRouteTableListener listener)- Добавляет слушателя в список  Public void unbindRouteTableListener(IRouteTableListener listener)- Удаляет слушателя из списка  Public RouteTable getRouteTable()- Метод получения таблицы маршрутизаций.  private RouteTable loadRouteTable() – Извлечение из командной строки значений ip-адреса в таблицу маршрутизации.  private RouteTable createRouteTable(String result)- Создает таблицу маршрутизации  private RouteTableRecord createRecord(String group)- Создает запись в таблице.  Private class RouteTableListener implements Runnable- Класс описывающий процесс мониторинга.  private RouteTableChangeEvent createChangeEvent(RouteTable newRouteTable)- Создает изменяющие события  private void sendEvent(RouteTableChangeEvent changeEvent) – Посылает события слушателям |

2.5 Принцип работы разработанного ПО

Принцип работы разработанного ПО следующий: В данном проекте реализован OSGI сервис, который следит за сетевыми таблицами и посылает события изменения таблицы клиентам, при изменении таблицы. Работа программы начинается с запуска класса «RouteTableService», так как в параметрах запуска прописано «immediate = true». Когда активируется сервис происходит первоначальная загрузка таблицы в сервис. Перед активацией сервиса происходит разрешение зависимостей, которые указаны в компоненте сервиса под методом «reference bind». В данном методе происходит загрузка «слушателей». В классе RouteTableService содержится метод «CreateChangeEvent», который создает события изменения. Сервис отслеживает данные события и рассылает их клиентам, обновляя перед этим таблицу.

3 Технологическая часть

3.1Запуск разработанного приложения

Исходный код проекта доступен в репозитории GitHub [4].

Сервис RouteTableService взаимодействует с двумя клиентами AddRecordListener, DeleteRecordListener. Запуск приложения осуществляется с помощью конфигурационного файла, который содержит настройки, необходимые для успешного запуска приложения.

3.2 Анализ исходного кода с помощью метрик качества

На рисунке 5 показано соотношение пакетов проекта по их размеру. Видно, что самый большой размер имеет пакет «model».

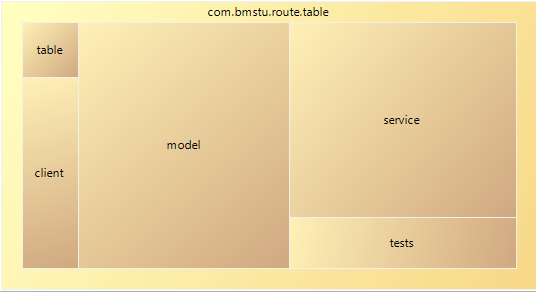


Рисунок 5 - Соотношение пакетов по размеру

Далее на рисунке 6 отображен список всех метрик по разделам. Всего имеется четыре раздела:

* метрики количества (Count);
* метрики сложности (Complexity);
* метрики Роберта Мартина (Robert C. Martin);
* метрики Чидамбера-Кемерера (Chidamber & Kermerer).
* Первый раздел с метриками количества (Count) содержит следующие метрики:
* количество классов верхнего уровня (Unit);
* среднее число внутренних классов на класс (Classes / Class);
* среднее число методов в классе (Methods / Class);
* среднее число полей в классе (Fields / Class);
* число строчек кода (ELOC);
* число строчек кода на модуль (ELOC / Unit).

Второй раздел с метриками сложности (Complexity) содержит всего три различных метрики:

* средняя циклическая сложность (CC);
* метрика Fat (Fat);
* средняя зависимость компонентов между модулями (ACD - Unit).

Третий раздел с метриками Роберта Мартина содержит следующие метрики:

* нормализованное расстояние от основной последовательности (D);
* абстрактность (A);
* нестабильность (I);
* число афферентных соединений (Ca);
* число эфферентных соединений (Ce).

Четвертый раздел с метриками Чидамбера-Кемерера содержит следующие метрики:

* средняя длина метода на класс (WMC);
* средняя глубина наследования (DIT);
* среднее количество классов-наследников (NOC);
* среднее число соединений класса (CBO);
* среднее число методов, которые потенциально могут быть выполнены в ответ на сообщение, полученное объектом этого класса (RFC);
* отсутствие единства методов (LCOM).

Разделы метрик:



Рисунок 6 – Значение метрик

3.3 Анализ зависимостей в коде системы

Рисунок 7 описывает зависимости между пакетами разработанного программного обеспечения:

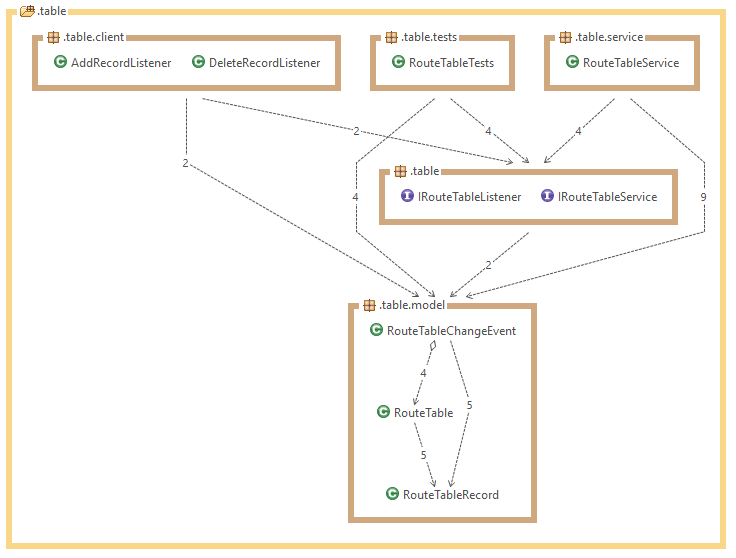


Рисунок 7 – Зависимости между пакетами

3.4 Тестирование на корректность работы

Работоспособность разработанного программного обеспечения проверялась путем подключения и отключения устройства к интернет соединению.

На рисунке 8 показан результат работы ПО при подключении устройства к интернет соединению:

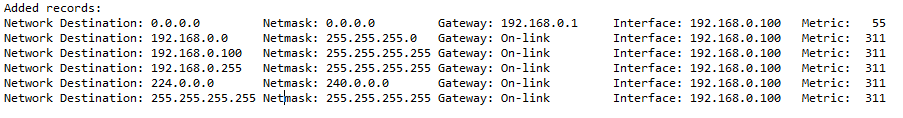


Рисунок 8 – Вывод ПО при подключении к интернету

На рисунке 9 показан результат работы ПО при отключении устройства от интернет соединения:

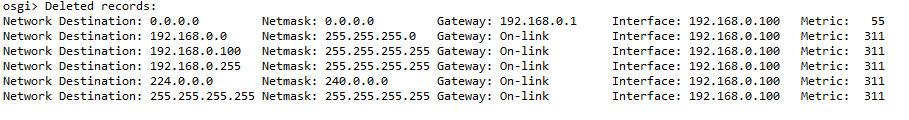


Рисунок 9 – Вывод ПО при отключении от интернета

3.5 Реализация Junit тестов

Для последующего тестирования разработанного программного обеспечения была поставлена задача написания JUnit тестов.

Интеграция JUnit с OSGI происходит с помощью ServiceTracker. ServiceTracker позволяет получить сервисы из OSGI. Для его использования тест необходимо запускать, как plugin-test, иначе система не будет подгружать необходимые плагины.

В ходе выполнения курсовой работы было написано 2 теста, проверяющих следующий функционал ПО:

Доступность сервиса, используется метод assertNotNull, в случае ошибки в поиске сервиса выводится сообщение: "Route table service is null!"

Возможность получения информации о клиентах, используется метод assertNotNull, в случае отсутствия какой-либо получаемой информации выводится сообщение "Route table listener is null!"

Отсутствие ошибок при использовании клиентов.

Перед тестирующими методами используется аннотация Test.

Результаты тестирования JUnit приведены на рисунке 10:

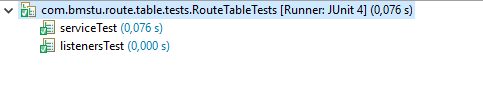


Рисунок 10 – Результат работы тестов для проверки на корректность работы

Вывод

В ходе выполнения курсовой работы:

* Был разработан OSGI сервис мониторинга сетевых таблиц маршрутизации;
* Разработаны Junit тесты;
* Изучены основные принципы работы необходимых технологий;
* Разработано ПО, соответствующее требованиям ТЗ;

Проведены необходимые тесты работоспособности программного продукта. Результаты тестирования свидетельствуют правильное функционирование ПО

Список литературы

[1] Введение в OSGI. Взаимодействие бандлов. Сервисы: [Электронный ресурс]. Режим доступа: <http://samolisov.blogspot.ru/2009/03/osgi_21.html>

[2] Введение в OSGI. Работаем с Equinox – реализацией OSGI R4 от Eclipse Foundation: [Электронный ресурс]. Режим доступа: <http://samolisov.blogspot.ru/2009/03/equinox.html>

[3] Введение в OSGI. Динамический ServiceTracker. Две стратегии использования сервисов.

[4] Github – PosesCapture [Электронный ресурс]. Режим доступа: <https://github.com/DenisStarchenko/RouteTable>