Федеральное государственное бюджетное образовательное учреждение   
высшего образования

|  |  |
| --- | --- |
|  | «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 активных процессов

Студент группы ИУ3-73 (подпись) 26.02.2018 А.А. Коровин

Руководитель курсовой работы (подпись) 26.02.2018 А.М Иванов

Москва, 2017

Реферат

В работе предложена архитектура OSGI компонента мониторинга активных процессов. 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

[Введение 3](#_Toc506539821)

[1 Исследовательская часть 3](#_Toc506539822)

[1.1 Выбор и обоснование способа работы с активными процессами 3](#_Toc506539823)

[2 Конструкторская часть 3](#_Toc506539825)

[2.1 Техническое задание 3](#_Toc506539826)

[2.2 Выбор технических решений, удовлетворяющих интересам ЗС 3](#_Toc506539827)

[2.3 Структура проекта 3](#_Toc506539828)

[2.3.1 Плагин com.bmstu.process.manager 3](#_Toc506539829)

[2.4 Диаграмма компонентов 3](#_Toc506539830)

[2.5 Диаграмма классов 3](#_Toc506539831)

[2.6 Алгоритм работы ПО 3](#_Toc506539832)

[3 Технологическая часть 3](#_Toc506539833)

[3.1 Запуск разработанного приложения 3](#_Toc506539834)

[3.2 Анализ искходного кода с помощью метрик качества 3](#_Toc506539835)

[3.3 Анализ зависимостей в коде системы 3](#_Toc506539836)

[3.4 Тестирование на корректность работы 3](#_Toc506539837)

[3.5 Реализация Junit тестов 3](#_Toc506539838)

[Заключение 3](#_Toc506539840)

[Список литературы 3](#_Toc506539841)

Введение

Вычислительные системы в наше время достигли определенного уровня сложности, при котором человеческих возможностей для их запуска и поддержания становится недостаточно. Решением данной проблемы стала автономность и автоматизация некоторых процессов. Одна из главных задач автономных вычислений – улучшение вычислительных систем при снижении фактора человеческого участия.

В самоуправляемых автономных системах человек, как оператор, приобретает совершенно новую роль: вместо прямого управления системой, он определяет общие политики и правила, которые определяют процесс самоконтроля.

В данной курсовой работе проектируется как раз система описанного выше типа, используемая для мониторинга активных процессов.

1. Исследовательская часть

Настоящий раздел является теоретическим и включает:

* Выбор способа работы с активными процессами;
* Обзор функциональных возможностей технологии Sigar.
  1. Выбор и обоснование способа работы с активными процессами

Технология Sigar позволяет отслеживать состояние активных процессов операционной системы.

* 1. Обзор функциональных возможностей технологии Sigar

1. Конструкторская часть
   1. Техническое задание

Изучить соответствующие системы, спроектировать интерфейс компонента, реализовать компонент, спроектировать JUnit тесты, провести тестирование, описать требования, конструкцию, особенности сборки проекта в документации.

* 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 2 – Заинтересованные стороны и их интересы по отношению к системе

|  |  |
| --- | --- |
| **Интересы заинтересованных сторон** | **Технические решения** |
| Отслеживание событий изменения состояния активных процессов. | При любом изменении состояния активных процессов, 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. |

* 1. Структура проекта

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

Таблица 3 – Разработанные плагины и их описание

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

* + 1. Плагин com.bmstu.process.manager

Данный плагин содержит 2 пакета:

* com.bmstu.process.manager;
* com.bmstu.process.manager.model.

Пакет com.bmstu.process.manager содержит интерфейс IActiveProcessListener, который включает в себя 1 метод:

Таблица 4 – Методы интерфейса IActiveProcessListener и их описание

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| void handleEvent(ActiveProcessEvent event); | Добавляет событие, содержащее информацию о добавлении, изменении и удалении информации о процессах. |

Пакет com.bmstu.process.manager содержит интерфейс IActiveProcessManager,

который включает в себя 1 метод:

Таблица 5 – Методы интерфейса IActiveProcessManager и их описание

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| Collection<Process> getActiveProcesses(); | Возвращает действительную информацию о состоянии активных процессов. |

Пакет com.bmstu.process.manager.model содержит 3 класса:

* ActiveProcessEvent.java
* Process.java

Таблица 6 – Классы пакета com.bmstu.coursework.ethernet.manager.model и их описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| Process | public Process(String name, List<String> args)– конструктор  public String getName() – возвращает имя процесса.  public List<String> getArgs() – возвращает параметры процесса. |
| ActiveProcessEvent | public ActiveProcessEvent(Collection<Process> activatedProcesses, Collection<Process> deactivatedProcesses)– конструктор;  public Collection<Process> getActivatedProcesses()– возвращает добавленную информацию об активных процессах;  public Collection<Process> getDeactivatedProcesses()– возвращает информацию о деактивированных процессах; |

* + 1. Плагин com.bmstu.process.manager.client

Данный плагин содержит 1 пакет:

* com.bmstu.process.manager.clients

Пакет com.bmstu.process.manager.client содержит 2 класса:

* ActivateProcessListener;
* DeactivateProcessListener;

Таблица 7 – Классы пакета com.bmstu.process.manager.clients и их описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| ActivateProcessListener | public void handleEvent(ActiveProcessEvent event) – выводит информацию об активированном процессе. |
| DeactivateProcessListener | public void handleEvent(ActiveProcessEvent event) – выводит информацию о деактивированном процессе. |

* + 1. Плагин com.bmstu.process.manager.service

Данный плагин содержит 1 пакет:

* com.bmstu.process.manager.service.

Пакет com.bmstu.process.manager.service содержит 1 класс:

* ActiveProcessManager;

Таблица 7 – Классы пакета com.bmstu.coursework.ethernet.manager.impl и их описание

|  |  |
| --- | --- |
| Класс | Состав и описание |
| ActiveProcessManager | public ActiveProcessManager()– конструктор;  public Collection<Process> getActiveProcesses()– возвращает действительную информацию об активных процессах;  public void activate(Map<String, Object> properties) – запускает поток, активирует сервис;  public void deactivate() – останавливает работу потока, деактивирует сервис;  public void bindListener(IActiveProcessListener listener)– добавляет прослушивающего клиента;  public void unbindListener(IActiveProcessListener listener)- удаляет прослушивающего клиента;  private void throwEvent(ActiveProcessEvent event) – сообщает клиентам о событии;  private Collection<Process> createProcesses() – создает информацию о процессах; |

* 1. Диаграмма классов

Для наглядности, с помошью редактора UML Papirus была создана диаграмма классов. Весь проект состоит из четырех пакетов. Функционал всех пакетов и их компонентов был описан выше.

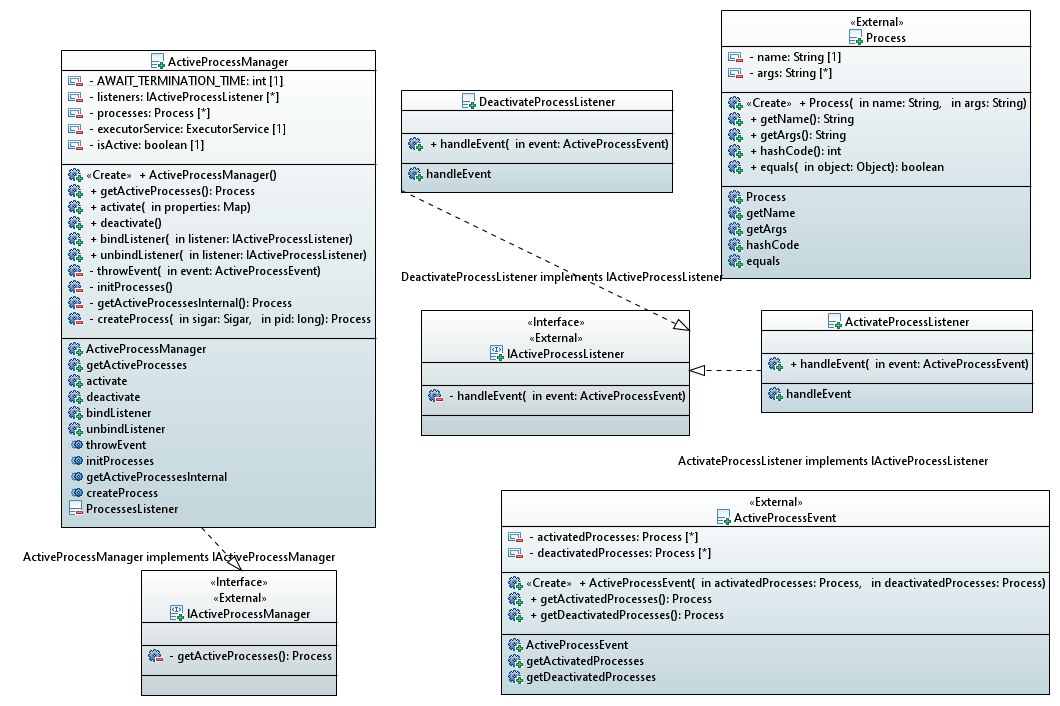


Рисунок 2 – Диаграмма классов

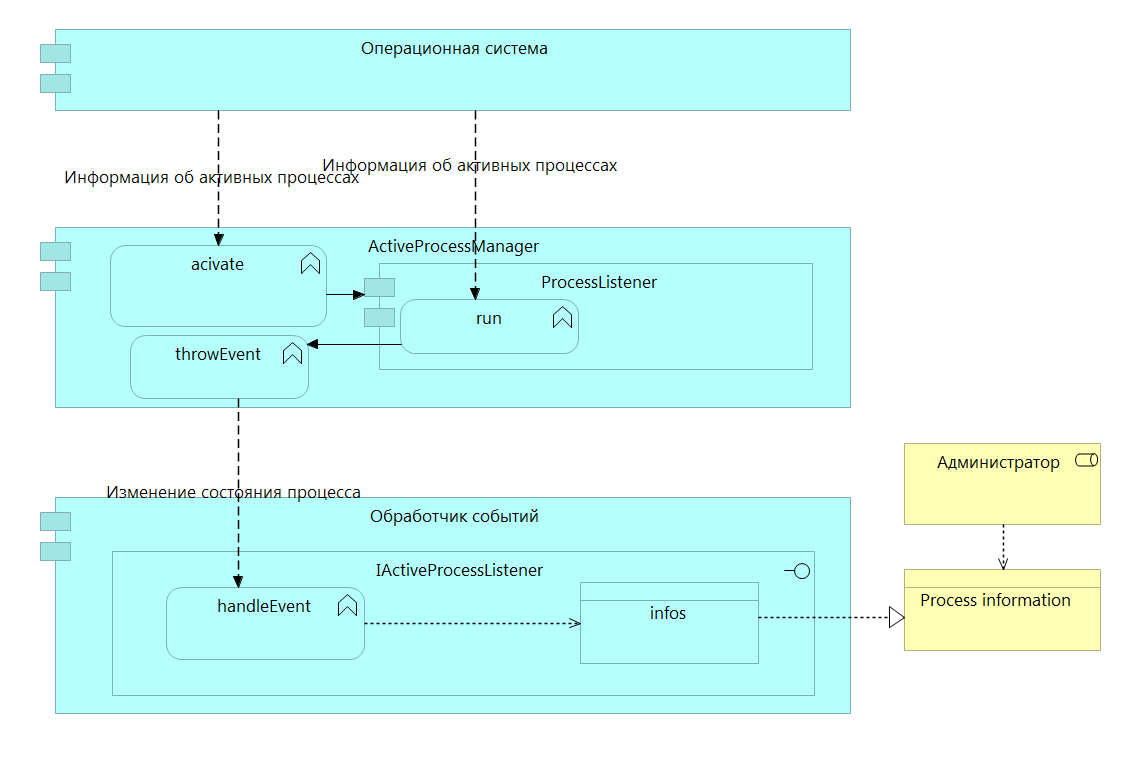


Рисунок 3 – Диаграмма компонентов

* 1. Алгоритм работы ПО

Окончательный алгоритм работы разработанного ПО следующий:  
1) Активация сервиса;  
2) Активация клиентов, их занесение менеджером в коллекцию клиентов;  
3) Запуск менеджером потока, в котором он через некоторые промежутки времени опрашивает порты;   
4) Полученная после запроса информация о процессе сравнивается с последней сохраненной информацией об этом же процессе;  
5) Если два набора информации не идентичны, то создается событие, которое отправляется на обработку клиентам;  
6) В зависимости от типа поступившего события, выбирается нужный клиент.  
7) Сообщение обрабатывается и выводится в консоль  
8) Цикл повторяется

1. Технологическая часть
   1. Запуск разработанного приложения

Исходный код проекта доступен в репозитории GitHub [3].

Сервис ActiveProcessManager взаимодействует с двумя клиентами ActivateProcessListener, DeactivateProcessListener. Запуск приложения осуществляется с помощью конфигурационного файла, который содержит настройки, необходимые для успешного запуска приложения.

* 1. Анализ искходного кода с помощью метрик качества

Далее на рисунке 4 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).

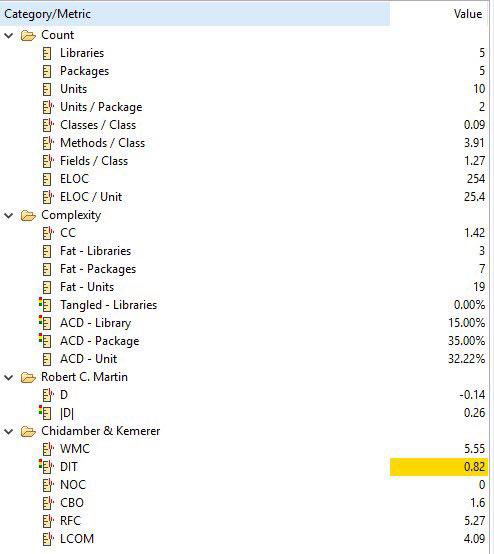


Рисунок 4 – Значения метрик

* 1. Тестирование на корректность работы

Работоспособность разработанного программного обеспечении:

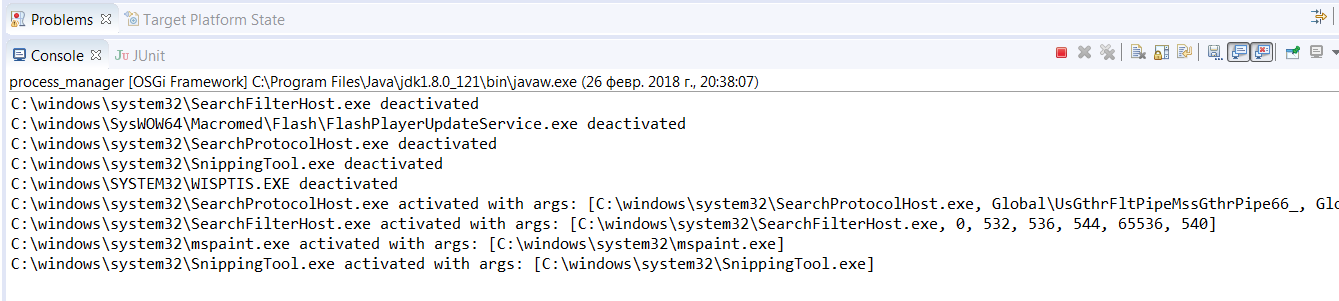


Рисунок 6– Работоспособность программы при активации или деактивации процессов

* 1. 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 3 теста, проверяющих следующий функционал ПО:

Доступность сервиса, используется метод assertNotNull, в случае ошибки в поиске сервиса выводится сообщение “Process manager not found”

Возможность получения информации о процессах, используется метод assertNotNull, в случае отсутствия какой-либо получаемой информации выводится сообщение “Process is null”

Отсутствие ошибок при использовании клиентов.

Перед тестирующими методами используется аннотация Test.

Результаты тестирования JUnit приведены на рисунке 8:

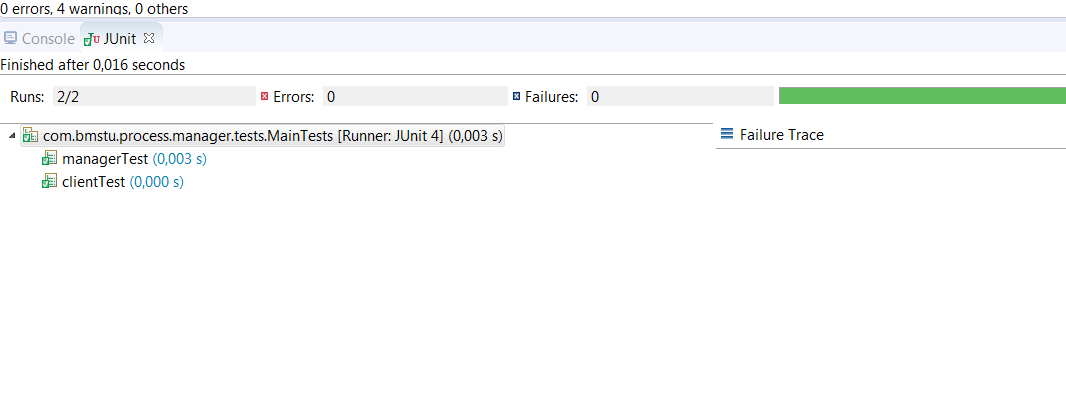


Рисунок 8 – Результат работы тестов для проверки на корректность работы

Заключение

В ходе выполнения курсовой работы был разработан OSGI сервис мониторинга активных процеесов. Работа была выполнена с помощью технологий OSGI и JUnit в среде разработки Eclipse.

В результате выполнения проекта были решены задачи, поставленные в начале работы. Были 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]  Lesson: Programmatic Access to Network Parameters : [Электронный ресурс]. Режим доступа: https://docs.oracle.com/javase/tutorial/networking/nifs/index.html

[2] OSGI modulation – Tutorial : [Электронный ресурс]. Режим доступа: http://www.vogella.com/tutorials/OSGi/article.html

[3] OSGI in Action: Creating modular applications in JAVA : Richard Hall 2011. – 416 с.

[4] OSGI and Equinox : J. McAffer, 2010. – 328 с.

[5] OSGI services – Tutorial : [Электронный ресурс]. Режим доступа: http://www.vogella.com/tutorials/OSGiServices/article.html