|  |
| --- |
| Министерство образования и науки Российской Федерации  Федеральное государственное бюджетное образовательное учреждение  высшего профессионального образования  «Московский государственный технический университет имени Н.Э. Баумана»  (МГТУ им. Н. Э. Баумана)  ФАКУЛЬТЕТ «Информатика и системы управления»  КАФЕДРА «Информационные системы и телекоммуникации»  РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ  НА ТЕМУ:  «Агент мониторинга ресурсов для OSGi контейнера мультиагентной платформы»  Студент ИУ3-82 В.С. Иванов  (Подпись, дата)  Руководитель ВКР А.М. Иванов  (Подпись, дата)  Нормоконтролер С.А. Тренин  (Подпись, дата) |
| 2016 г. |

**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э.Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-3

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Девятков

(И.О.Фамилия)

«21 » января 2016 г.

**ЗАДАНИЕ**

**на выполнение выпускной квалификационной работы бакалавра**

|  |  |  |
| --- | --- | --- |
| Студент группы | ИУ3-82 |  |
| Иванов Владимир Сергеевич | | |

(Фамилия, имя, отчество)

|  |  |
| --- | --- |
| Тема квалификационной работы | Агент мониторинга ресурсов для |
| OSGi контейнера мультиагентной платформы | |
|  | |
|  | |

Источник тематики (НИР кафедры, заказ организаций и т.п.)

|  |
| --- |
| кафедра |
|  |
|  |

Тема квалификационной работы утверждена распоряжением по факультету \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ № \_\_\_\_\_\_\_\_\_\_\_от « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

***Часть 1. \_Научно-исследовательская часть\_***

|  |
| --- |
| Выявление заинтересованных сторон; |
| Выявление основных требований к системе; |
| Исследование существующих систем и стандартов мониторинга ресурсов; |
| Постановка задачи реализации сервиса мониторинга ресурсов; |
| Сравнительный анализ методов мониторинга ресурсов; |
| Разработка архитектуры сервиса мониторинга ресурсов. |
|  |

***Часть 2. \_Проектно-конструкторская часть\_***

|  |
| --- |
| Разработка высокоуровневой архитектуры (использование операторами компонентов |
| системы); |
| Разработка детальной архитектуры (детализация компонентного описания системы;  выбор и анализ необходимых библиотек и подготовка модульного описания, |
| описание файловой структуры исходного кода и инструментов разработки, описание |
| развертывания системы пользователя). |
|  |
|  |
|  |
|  |
|  |

***Часть 3. \_ Технологическая часть\_***

|  |
| --- |
| Методика проверки соответствия архитектуры системы требованиям. |
| Методика тестирования сервисов; |
| Тестирование системы. |
| Оценка соответствия разработанной системы требованиям; |
|  |
|  |
|  |
|  |
|  |

***Оформление квалификационной работы:***

Расчетно-пояснительная записка на 60-70 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

|  |
| --- |
| Модель сервиса мониторинга ресурсов; диаграмма требований к системе; диаграмма |
| постановки задачи; таблица сравнительного анализа систем и стандартов мониторинга |
| ресурсов; диаграмма компонентов системы; диаграмма файловой структуры; диаграмма |
| развертывания системы пользователя; описание разработанных программных модулей; |
| методика и результаты тестирования. |
|  |

Дата выдачи задания «21» января 2016 г.

В соответствии с учебным планом выпускную квалификационную работу выполнить в полном объеме в срок до «01» июня 2016 г.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Руководитель квалификационной работы** |  |  |  | А.М. Иванов |

(Подпись, дата) (И.О.Фамилия)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Студент** |  |  |  | В.С. Иванов |

(Подпись, дата) (И.О.Фамилия)

**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э.Баумана)**

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** **ИУ**  **КАФЕДРА** **ИУ3**  **ГРУППА** **ИУ3-82** | УТВЕРЖДАЮ  Заведующий кафедрой ИУ-3  (Индекс)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Девятков  (И.О.Фамилия)  « 21 » января 2016 г. |

**КАЛЕНДАРНЫЙ ПЛАН**

**выполнения выпускной квалификационной работы**

|  |  |
| --- | --- |
| студента: | Иванов Владимир Сергеевич |

(фамилия, имя, отчество)

|  |  |
| --- | --- |
| Тема квалификационной работы: | Агент мониторинга ресурсов для |
| OSGi контейнера мультиагентной платформы | |
|  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Наименование этапов выпускной квалификационной работы** | **Сроки выполнения этапов** | | **Отметка о выполнении** | |
| **план** | **факт** | **Должность** | **ФИО, подпись** |
|  | Задание на выполнение работы. Формулирование проблемы, цели и задач работы | *\_15.02.16\_\_\_*  *Планируемая дата* |  | Руководитель ВКР |  |
|  | Научно-исследовательская часть | *05.03.16*  *Планируемая дата* |  | Руководитель ВКР |  |
|  | Утверждение окончательных формулировок решаемой проблемы, цели работы и перечня задач | *10.03.16*  *Планируемая дата* |  | Заведующий кафедрой |  |
|  | Проектно-конструкторская часть | *01.04.16*  *Планируемая дата* |  | Руководитель ВКР |  |
|  | Технологическая часть | *20.04.16*  *Планируемая дата* |  | Руководитель ВКР |  |
|  | 1-я редакция работы | *05.05.16*  *Планируемая дата* |  | Руководитель ВКР |  |
|  | Подготовка доклада и презентации | *10.05.16*  *Планируемая дата* |  |  |  |
|  | Заключение руководителя | *15.05.16*  *Планируемая дата* |  | Руководитель ВКР |  |
|  | Допуск работы к защите на ГЭК (нормоконтроль) | *25.05.16*  *Планируемая дата* |  | Нормоконтролер |  |
|  | Внешняя рецензия | *01.06.16*  *Планируемая дата* |  |  |  |
|  | Защита работы на ГЭК | *\_\_\_\_\_\_\_*  *Планируемая дата* |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| *Студент* |  | *Руководитель работы* |  |

# 

# АННОТАЦИЯ

Расчетно-пояснительная записка 70 с., 19 рис., 9 таблиц, 12 источников.

АРХИТЕКТУРА ОБЛАЧНЫХ СИСТЕМ, АВТОМАТИЗАЦИЯ РАЗВЕРТЫВАНИЯ СЕРВИСОВ, МОДУЛЬНОЕ ПО.

Данная работа посвящена решению проблем, связанных с построением сервисов в облаках, а именно проблемам автоматизации развертывания, тестирования, обновления, удаления, балансировки нагрузки и отказоустойчивости.

В исследовательской части представленной работы проводился анализ заинтересованных сторон, были сформулированы необходимые компоненты и функции системы. Также были установлены условия по обеспечению отказоустойчивости, масштабируемости.

В конструкторской части представлен анализ сервиса с открытым исходным кодом Eclipse User Storage Service. Был проведен анализ взаимодействия с этим сервисом, его масштабируемости и отказоустойчивости. Также предложен вариант по его улучшению.

В ходе работы на сервисом настроек был получен инструмент, позволяющий автоматизировать доступ по REST-API к унаследованным OSGi-сервисам, которые изначально не были ориентированы для использования в WEB.

В технологической части дан обзор на использованные технологии, средства работы, описан способ получения исполняемого файла сервиса настроек Eclipse IDE из исходных кодов программы, способ масштабирования сервиса настроек в облачной инфраструктуре.

В заключении дан анализ полученных результатов и дальнейшие планы по улучшению системы.

Итогом работы является отказоустойчивая система, которая может в автоматическом режиме масштабироваться, работать под большой нагрузкой.

СОДЕРЖАНИЕ

[АННОТАЦИЯ 5](#_Toc453283646)

[ВВЕДЕНИЕ 8](#_Toc453283647)

[1 Научно-исследовательская часть 9](#_Toc453283648)

[1.1 Анализ предметной области 9](#_Toc453283649)

[1.2 Мультиагентные системы и агенты 11](#_Toc453283650)

[1.3 Технология OSGi 11](#_Toc453283651)

[1.3.1 Модульность приложений 13](#_Toc453283652)

[1.3.2 Что же такое OSGi? 13](#_Toc453283653)

[1.3.3 Плагины или бандлы как компоненты приложения 14](#_Toc453283654)

[1.3.4 Сервисы 15](#_Toc453283655)

[1.3.5 Файл MANIFEST.MF 15](#_Toc453283656)

[1.4 Выявление заинтересованных сторон 18](#_Toc453283657)

[1.5 Сценарии использования системы 22](#_Toc453283658)

[1.6 Выявление основных требований к системе 25](#_Toc453283659)

[1.7 Исследование существующих систем и стандартов мониторинга ресурсов 27](#_Toc453283660)

[1.7.1 Монитор ресурсов Windows 27](#_Toc453283661)

[1.7.2 Spotlight 28](#_Toc453283662)

[1.7.3 KDE System Guard 29](#_Toc453283663)

[1.8 Постановка задачи 30](#_Toc453283664)

[2 Проектно-конструкторская часть 31](#_Toc453283665)

[2.1 Описание средств разработки 31](#_Toc453283666)

[2.1.1 Особенности платформы Eclipse 32](#_Toc453283667)

[2.1.2 Архитектура платформы Eclipse 32](#_Toc453283668)

[2.2 Архитектура системы 33](#_Toc453283669)

[2.2.1 Диаграмма компонентов 33](#_Toc453283670)

[2.3 Выбор используемых библиотек 35](#_Toc453283671)

[2.3.1 Библиотека OperatingSystemMXBean 35](#_Toc453283672)

[2.3.2 Библиотека Sigar 38](#_Toc453283673)

[2.3.3 Диаграмма модулей 39](#_Toc453283674)

[2.3.4 Диаграмма классов 40](#_Toc453283675)

[2.3.5 Диаграмма развертывания 40](#_Toc453283676)

[3 Технологическая часть 42](#_Toc453283677)

[3.1 Используемые средства 42](#_Toc453283678)

[3.2 Работа Java – компилятора 42](#_Toc453283679)

[3.3 Мониторы ресурсов 44](#_Toc453283680)

[3.4 Получение бинарной сборки системы 44](#_Toc453283681)

[3.5 Тестирование системы 53](#_Toc453283682)

[ЗАКЛЮЧЕНИЕ 58](#_Toc453283683)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 59](#_Toc453283684)

[ПРИЛОЖЕНИЕ А 61](#_Toc453283685)

[ПРИЛОЖЕНИЕ Б 64](#_Toc453283686)

# ВВЕДЕНИЕ

К современным информационным системам предъявляются серьезные требования по функциональности, времени ответа и другим параметрам. Зачастую, современные системы обрабатывают большие объемы данных, обрабатывают миллионы запросов в секунду и должны давать ответ пользователю незамедлительно. Такие системы обычно не умещаются на одном физическом сервере. Они работают распределено на многих физических серверах.

Обеспечение правильной балансировки распределения нагрузки на систему и ее отказоустойчивость - две фундаментальные задач в облачных системах. В такие системы обычно на стадии проектирования закладывают принципы масштабируемости, отказоустойчивости.

Немаловажную роль в облачных системах играют механизмы развёртывания сервисов. Зачастую в крупных проектах количество сервисов превышает сотню. Даже полсотни сервисов проблематично развёртывать на сервере вручную, т.к. это чревато ошибками при конфигурировании этих сервисов. Еще хуже дела становятся, когда в система разрастается до таких размеров, что не помещается на одном сервере. Правильно развернуть и сконфигурировать сотню сервисов на разных машинах практический невозможно, т.к. этим могут заниматься разные люди, да и один и тот же человек может попросту потерять бдительность. В связи с этим появляется необходимость автоматизировать процесс выкладки сервисов на сервера.

Решению проблем отказоустойчивости и распределения нагрузки системы, а также автоматизации выкладки сервисов и их менеджмента посвящена эта работа.

На примере разработанного сервиса настроек Eclipse IDE будет показана архитектура получившийся системы.

# Научно-исследовательская часть

## Анализ предметной области

На данный момент, существует стандартная реализация сервиса настроек Eclipse IDE, которая выполнена в виде обычного Java приложения. Этот сервис не может работать в среде OSGi, а следовательно не может в процессе работы принимать команды и не имеет динамических точек расширения.

Чтобы понять предназначение сервиса настроек Eclipse IDE, обратимся к рисунку 1, который выполнен в терминах ArchiMate[1].

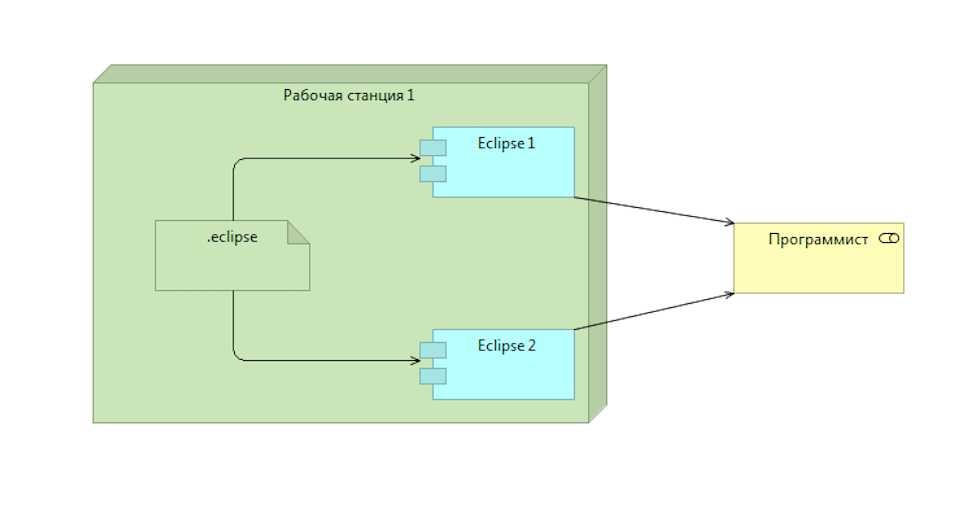


Рисунок 1 – Синхронизация настроек двух Eclipse IDE

Зачастую в процессе разработки программисту приходится работать с несколькими IDE. У программиста бывают свои собственные предпочтения касательно графического оформления IDE, специфических настроек плагинов и т.д. Но если программист будет вручную настраивать каждую свою IDE, то он будет расходовать рабочее время впустую. Для решения этой проблемы в Eclipse IDE уже есть механизм, позволяющий синхронизировать настройки Eclipse IDE в пределах одной рабочей машины через домашнюю папку пользователя, в которой заранее создастся папка .eclipse.

Современные процессы разработки предполагают наличие у программиста нескольких рабочих станций (рисунок 2). В этом случае Eclispe IDE также предоставляет способ решения проблемы синхронизации рабочих станций, разделенных сетью.



Рисунок 2 – Синхронизация Eclipse IDE на различных рабочих станциях

Но система, показанная на рисунке 2, имеет несколько существенных недостатков:

* Данные сохраняются на удаленном сервере, что неприемлемо для использования в корпоративной среде.
* Нельзя задать какие-либо неизменяемые настройки, например, флаги компиляции.

Это лишь основные недостатки данной системы. Более подробное описание будет представлено в последующих разделах.

Таким образом, становится очевидна потребность в более гибкой системе синхронизации настроек Eclipse IDE. Система такого типа будет разрабатываться в среде OSGi, речь о которой пойдет ниже. Для виртуализации приложения и автоматизации управлением жизненным циклом всей системы будет использована технология Docker. Обоснование выбора данных технологий и способа работы конечного продукта последует в нижележащих разделах.

## Технология виртуализации Docker

Docker – система виртуализации приложений. С помощью Docker можно запускать процессы в минимально рабочем и изолированном окружении. Процессу, запущенному в Docker, кажется, что он работает в минимальном окружении, где есть только он, его потомки и минимальный набор сервисов ОС. Но при этом процесс работает в той же самой операционной системе, как и другие, обычные, процессы, но он попросту не видит. Также процессу, запущенному в Docker-контейнере не доступны, если не разрешить явно, доступ к файловой системе хостовой операционной системы. Присутствует возможность организовать уникальную сеть для группы контейнеров. При этом эта сеть не будет видна другой группе контейнеров. Также можно ограничить число потребляемых ресурсов хостовой ОС вплоть до конкретного контейнера.

По описанным выше причинам в данной работе система будет работать в среде виртуализации Docker.

## Технология запуска контейнеров Docker-compose

Docker-compose это простой инструмент, который позволяет настроить и запустить несколько контейнеров одной командой. До появления этой технологии для запуска Docker-контейнеров приходилось писать свой shell-скрипты, где указывался порядок запуска, число запускаемых контейнеров, проставлялись все необходимые флаги каждому контейнеру, происходила настройка сети и доступ к файловой системе хостовой операционной системы.

С появлением технологии docker-compose появилась возможность автоматизировать процесс развертывания контейнеров. Все параметры необходимые для запуска контейнеров следует прописать в файле docker-compose.yml.

Рассмотри пример docker-compose.yml:

version: '2'

services:

web:

build:

context: ./web

dockerfile: Dockerfile.dev

volumes:

- .web/src:/web/src

ports:

- 8080:8080

mongo:

command: mongod

image: mongo:3.2.0

ports:

- 27100:278080

volumes:

- /var/run/docker.sock:/var/run/docker.sock

Нижеприведенная таблица 1, которая поясняет значение атрибутов в файле docker-compose.yml

Таблица 1. Атрибуты docker-compose.yml

|  |  |
| --- | --- |
| Ключ | Значение |
| version | Версия файла docker-compose.yml |
| services | Сервисы, которые должны быть обработаны |
| web | Имя контейнера |
| build | Команда для сборки Docker-контейнера |
| context | Задает директорию, где необходимо произвести сборку Docker-контейнера с именем web |
| dockerfile | Имя файла, который должен находится в директории, описанной в ключе context. В этом файле дан список команд, необходимый для создания Docker-контейнера. |
| volumes |  |
| ports | Соответствие портов в формате «порт хостовой ОС»:«порт контейнера» |
| mongo | Имя контейнера |
| command | Команда для запуска контейнера |
| image | Изображение(образ) контейнера, которое запустится под именем mongo |

## Технология управлением Docker swarm

Одиночный Docker Engine запускает контейнеры на локальной машине из образов, собранных на той же локальной машине или выкаченных с Docker Hub. Docker Hub это аналог GitHub для готовых образов.

Docker Swarm это кластер Docker’ов. Множество хостов, на каждом из которых запущен отдельный Docker Engine, объединены под общим управлением и в глобальном плане выглядят, как один Docker. Хосты могут динамический добавляться в кластер, и вычислительная емкость Docker будет расти. Это свойство особенно ценно, когда необходимо добавить бесшовно в работающую инфраструктуру новые сервера. Например, данную технологию можно использовать для динамического масштабирования балансировщика нагрузки или кластера баз данных.

Управление Docker Engine может осуществляться через TCP соединение рисунок 3. Демон Docker нужно запустить с некоторыми параметрами, чтобы разрешить ему слушать определенный порт для приема команд.



Рисунок 3 – Управление удаленным Docker.

Управлять Swarm будет Swarm Manager. Это тоже контейнер, запущенный из управляющего образа, swarm, с параметром manage.

Также Docker Swarm поддерживает отказоустойчивое развертывание управляющей ноды рисунок 4. Можно запустить несколько менеджеров, а между собой в автоматическом режиме договорятся между собой, кто будет главной нодой, а кто станет резервной. Это достигается благодаря поддержки Docker алгоритма Raft[!!!!!].



Рисунок 4 – отказоустойчивое развертывание Docker Manager

Реплика периодический проверяет управляющий хост. Если управляющий хост перестает отвечать реплике, то реплика становится master. Когда управляющий хост восстановится и сообщит об этом реплике, которая находится в состоянии master, в системе сбрасываются все master ноды. Реплика и управляющий хост начинают переговоры о том, кто станет новым master.

## Спецификация OSGi

OSGi (Open Services Gateway Initiative) – набор спецификаций, которые описывают создание динамической модульной инфраструктуры для создания Java-приложений. Спецификация OSGi разрабатывается консорциумом OSGi Alliance [!!!]. Данная спецификация имеет широкий спектр применения. Изначально разрабатывалась для создания встроенных систем, однако сейчас, пользуясь спецификации OSGi, создают высокомодульные приложения для ПК. Например, корпоративные системы или системы на базе Eclipse.

Основная идея реализующего спецификации OSGi фреймфорка – все в системе есть плагины или, по-другому бандлы. Основной способ взаимодействия между плагинами – сервисы. Сервисы – объекты, зарегистрированные в системе с заявленными реализованными интерфейсами. Плагины регистрируют сервисы для предоставления некоторой функциональности, реализованной ими, другим плагинам. Помимо этого OSGi предоставляет другие богатые возможности, которые могу быть полезны при разработке высокомодульной системы.

Плагин платформы OSGi содержит Java-классы и другие ресурсы, которые могут реализовывать некоторые функции, а также предоставлять сервисы и пакеты другим плагинам. Конструктивно плагин может либо каталогом в файловой системе, либо jar-архивом. Плагин, который содержит в себе фреймворк и управляет жизненным циклом других плагинов называется системным.

Так как изначально спецификация OSGi разрабатывалась для встроенных систем, то возникла потребность в экономии ресурсов. Для ее решения вводится понятие жизненного цикла плагина. Если система спроектирована правильно, то можно периодический выгружать из памяти неиспользуемые плагины и снова загружать, когда в этом возникнет необходимость. На рисунке 5 показан жизненный цикл плагинов.

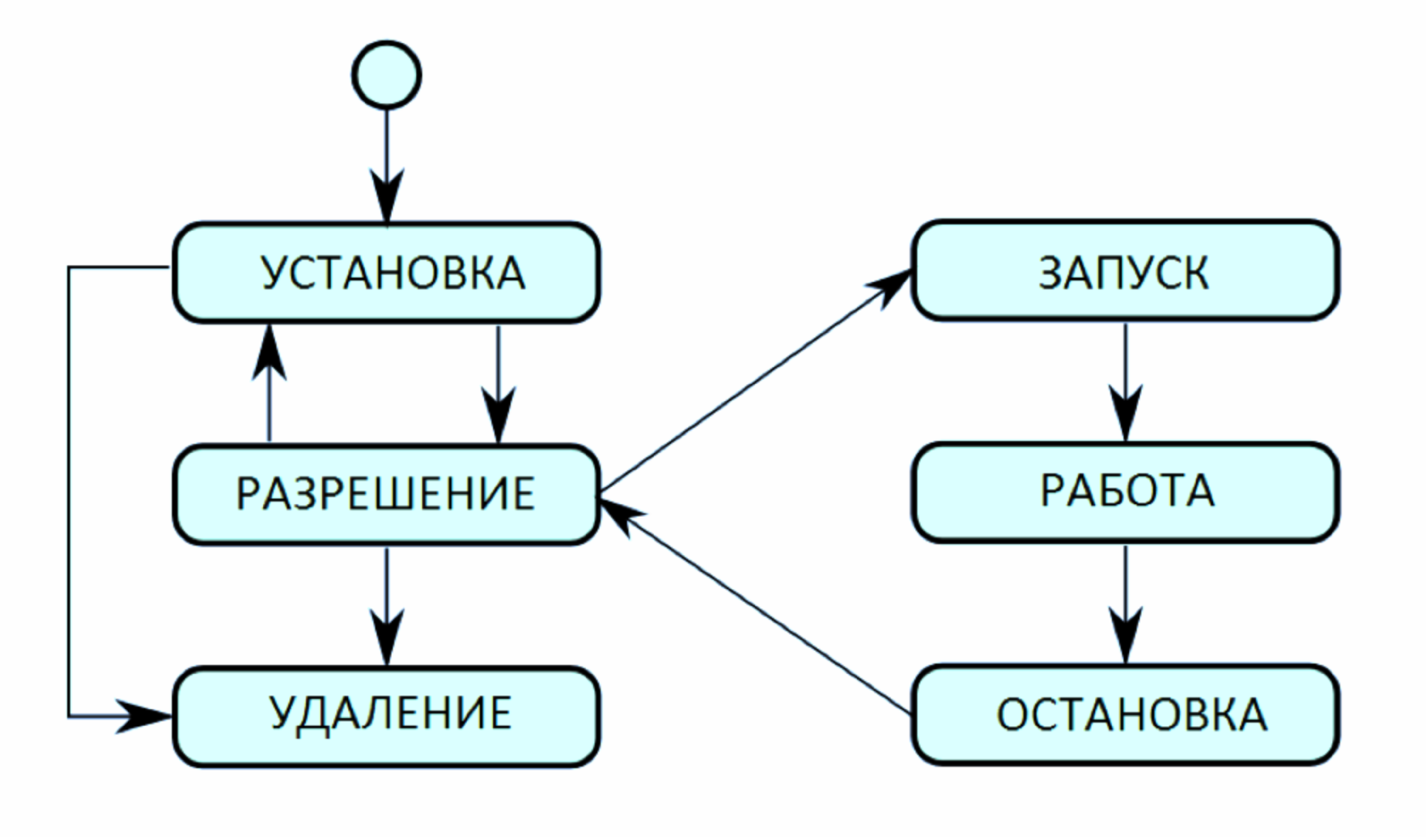


Рисунок 5 – Жизненный цикл плагинов

Жизненный цикл плагина – набор состояний, в которые он может принимать:

* Установка. Значит, что плагин успешно установлен.
* Разрешение. Разрешены все зависимости. Плагину доступны все Java-классы и плагины, от которых он зависит. Данное состояние сигнализирует о том, что плагин готов стартовать.
* Запуск. Плагин стартует. Метод BundleActivator.start() выполняется и пока не вернул значения.
* Работа. Плагин успешно запустился. Метод BundleActivator.start(вернул значение.
* Остановка. Плагин останавливается, но метод BundleActivator.stop() пока не вернул значение.
* Удаление. Плагин не установлен либо удалено, поэтому он не может переходить в другие состояния. Жизненный цикл плагина завершен.

### Модульные приложения

Зачастую современные программные продукты состоят из различных независимых частей. Их называют компонентами или модулями приложения.

Такие компоненты или модули взаимодействуют между собой через фиксированное на стадии проектирования API (Application Programming Interface). API – определяется набором классов и методов, определённых в этих этих классах, которые могут быть использованы компонентами приложения. В языке программирования Java API называется интерфейсом.

Если один компонент использует API второго компонента, то говорят, что первый компонент зависит от второго, а второй компонент выставляет наружу точки взаимодействия с ним.

Язык программирования Java, в текущей версии (Java 8), не предоставляет структурированный способ описания зависимостей компонентов. Java поддерживает использование модификаторов доступа и интерфейсов, но каждый публичный интерфейс может быть вызван из другого компонента программного обеспечения. Возникают ситуации, когда необходимо ограничить доступ к публичным классам или интерфейсам для определенного множества компонентов по какому-либо правилу. Спецификация OSGi предоставляет такую возможность.

## Микросервисная архитектура приложений

Микросервисная архитектура – это подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и коммуницирует с остальными, используя легковесные механизмы, например, REST API []. Эти сервисы построены вокруг бизнес-потребностей и развертываются независимо с использованием полностью автоматизированной среды. Существует абсолютный минимум централизованного управления этими сервисами. Сами по себе эти сервисы могут быть написаны на разных языках программирования и использовать разные технологии хранения данных.

Сравним микросервисный подход с монолитным стилем написания программы.

Enterprise приложения часто включают три основные части: пользовательский интерфейс (состоящий как правило из HTML страниц и javascript-а), база данных (как правило реляционной, со множеством таблиц) и сервер. Серверная часть обрабатывает HTTP запросы, выполняет доменную логику, запрашивает и обновляет данные в БД, заполняет HTML страницы, которые затем отправляются браузеру клиента. Любое изменение в системе приводит к пересборке и развертыванию новой версии серверной части приложения.

Монолитный сервер – довольно очевидный способ построения подобных систем. Вся логика по обработке запросов выполняется в единственном процессе, при этом вы можете использовать возможности вашего языка программирования для разделения приложения на классы, функции и namespace-ы. Вы можете запускать и тестировать приложение на машине разработчика и использовать стандартный процесс развертывания для проверки изменений перед выкладыванием их в рабочее окружение. Вы можете масштабировать монолитное приложения горизонтально путем запуска нескольких физических серверов за балансировщиком нагрузки.

Монолитные приложения могут быть успешными, но все больше людей разочаровываются в них, особенно в свете того, что все больше приложений развертываются в облаке. Любые изменения, даже самые небольшие, требуют пересборки и развертывания всего монолита. С течением времени, становится труднее сохранять хорошую модульную структуру, изменения логики одного модуля имеют тенденцию влиять на код других модулей. Масштабировать приходится все приложение целиком, даже если это требуется только для одного модуля этого приложения.

Эти неудобства привели к архитектурному стилю микросервисов: построению приложений в виде набора сервисов. В дополнение к возможности независимого развертывания и масштабирования каждый сервис также получает четкую физическую границу, которая позволяет разным сервисам быть написанными на разных языках программирования. Они также могут разрабатываться разными командами.

## Выявление заинтересованных сторон

Теперь, имея представление о технологиях, которые будут использоваться в работе, перейдем к описанию взаимодействия проектируемой системы и окружающего мира.

Перед началом работы над системой, необходимо определить границы системы и понять какие сценарии система должна выполнять. Границы системы изображены на рисунке 3.

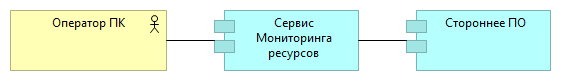


Рисунок 3. Диаграмма границ системы

Для разрабатываемой системы был проведен анализ заинтересованных сторон.

В таблице 2 отобразим заинтересованные лиц и опишем их интересы.

Таблица 2. Заинтересованные стороны и их интересы

|  |  |  |
| --- | --- | --- |
| **Роль** | **Заинтересованная сторона** | **Интересы** |
| Пользователи | Оператор ПК | Работа с системой, возможность отслеживать показатели системы, удобство работы |
| Разработчики | Разработчик | Обеспечение модифицируемости и сопровождаемости системы, быстрое устранение ошибок |
| Тестировщик | Проверка системы, контролепригодность |
| Разработчик стороннего ПО | Легкая дополняемость, адаптируемость кода |
| Вредители | Конкурент | Копирование технологий |
| Злоумышленник | Получение контроля над системой, изменение параметров системы |
| Заказчики | Заказчик | Работоспособность системы, соответствие системы требованиям |

Все выявленные заинтересованные стороны изображены на диаграмме заинтересованных сторон (рисунок 4).

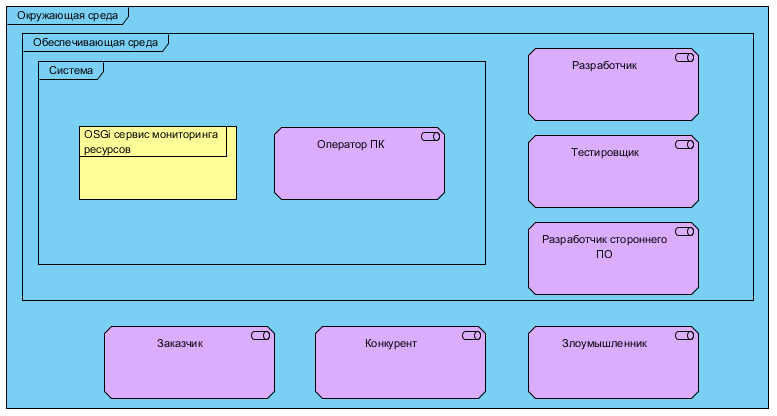


Рисунок 4. Диаграмма заинтересованных сторон

В первом блоке (блок «Система») отображены лица, имеющие непосредственное отношение к системе и тесно связанные с ней.

Во втором блоке (блок «Обеспечивающая среда») изображены стороны, поддерживающие работоспособность системы.

В третьем блоке (блок «Окружающая среда») отображены стороны, которые могут повлиять на систему.

Для выявленных заинтересованных сторон были подобраны их представители, и была составлена диаграмма «Заинтересованных стороны и их представители», изображенная на рисунке 5.

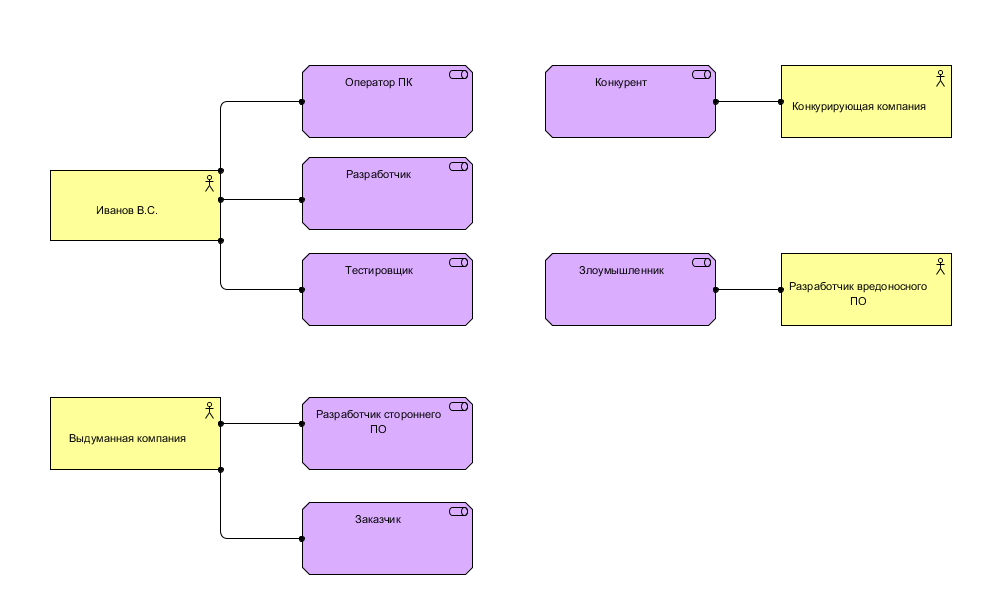


Рисунок 5. Диаграмма заинтересованных сторон и их представителей

## Сценарии использования системы

У каждой из заинтересованных сторон есть свои интересы, которых они хотят достичь, используя разрабатываемую систему. Все цели и их связь с заинтересованными лицами изображены на диаграмме заинтересованных лиц и их целей (рисунок 6).

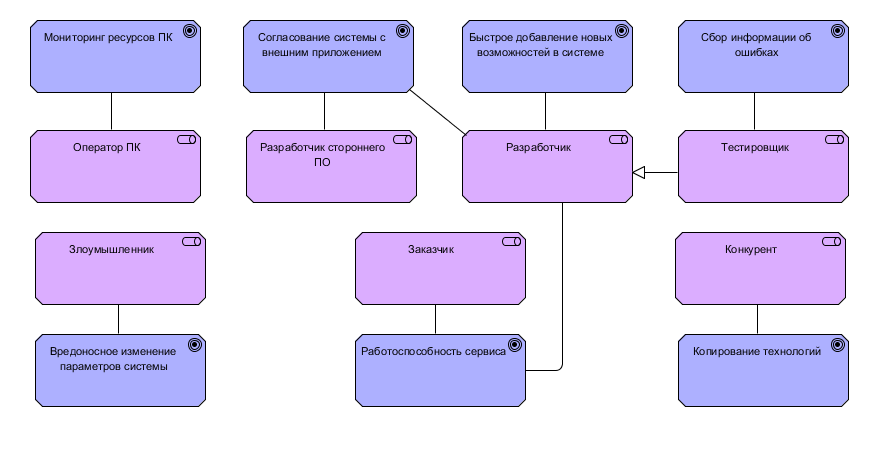


Рисунок 6. Диаграмма заинтересованных лиц и их целей

Основной сценарий использования системы планируется таким:

**Сценарий:** Пользователь проверяет ресурсы системы.

**Действующее лицо:** пользователь системы.

**Основной сценарий:**

Шаг 1: Пользователь запускает программу.

Шаг 2: Программа получает текущее состояние компонентов системы.

Шаг 3: Программа анализирует полученные данные.

Шаг 4: Вывод данных на экран.

**Альтернативные сценарии:**

1. Компонент не обнаружен в системе:

Шаг 1. Принудительный запуск процесса обнаружения компонента.

Шаг 2. Повторная попытка выбора компонента.

2.Передаваемый запрос не был доставлен до узла

Шаг 1. Повторная отправка запроса.

Шаг 2. Обращение к разработчику.

Далее рассмотрим диаграмму сценариев, представленную на рисунке 7.

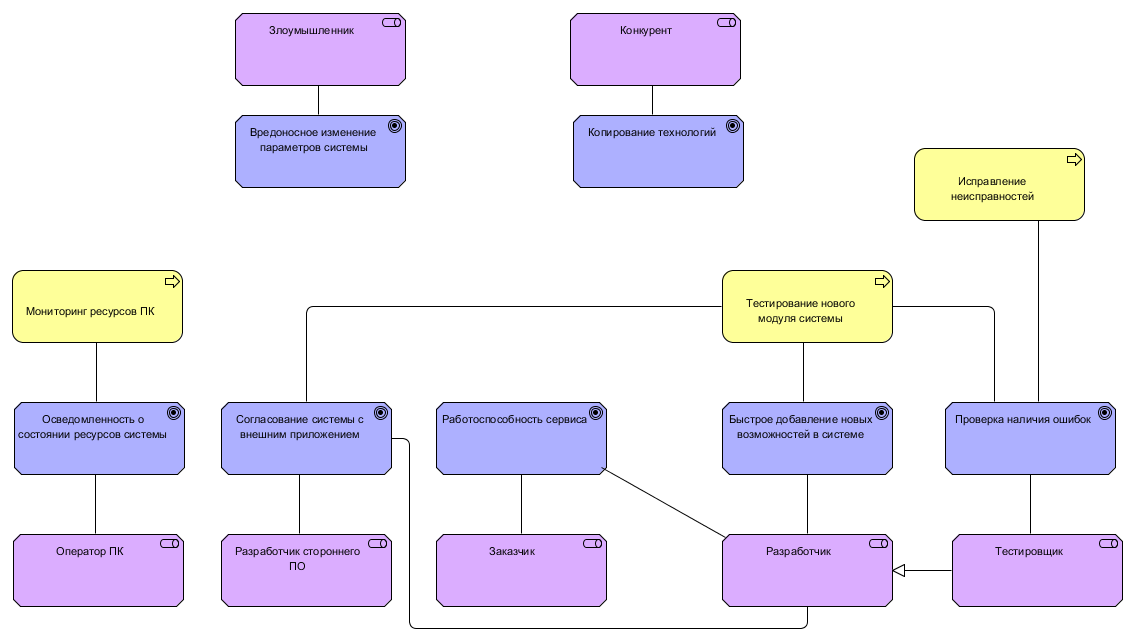


Рисунок 7. Диаграмма сценариев

## Выявление основных требований к системе

Функциональные требования:

* Мониторинг ресурсов системы
* Добавление новых типов ресурсов
* Вывод значений на экран

Нефункциональные требования [4] рассмотрены в таблице 3.

Таблица 3 Таблица показателей качества системы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Показатель качества** | **Актор** | **Воздействие** | **Артефакт** | **Условие работы** | **Реакция** | **Численная мера** |
| Эффективность | Оператор ПК | Проверка распределения ресурсов | Сервис мониторинга ресурсов | Нормальный режим работы | Успешная проверка | Время проверки меньше 5 секунд |
| Модифицируемость | Разработчик | Добавление новых типов мониторов | Сервис мониторинга ресурсов | Нормальный режим работы | Добавление новых мониторов | Возможность  добавления мониторов при нулевом времени простоя |
| Кроссплатформенность | Оператор ПК | Возможность проверки распределения ресурсов на ПК с различными ОС | Сервис мониторинга ресурсов | Нормальный режим работы | Успешная проверка | Не менее двух типов поддерживаемых ОС |

## **Исследование существующих систем и стандартов мониторинга** ресурсов

На данный момент существует множество систем мониторинга ресурсов. Рассмотрим некоторые из них.

### Монитор ресурсов Windows

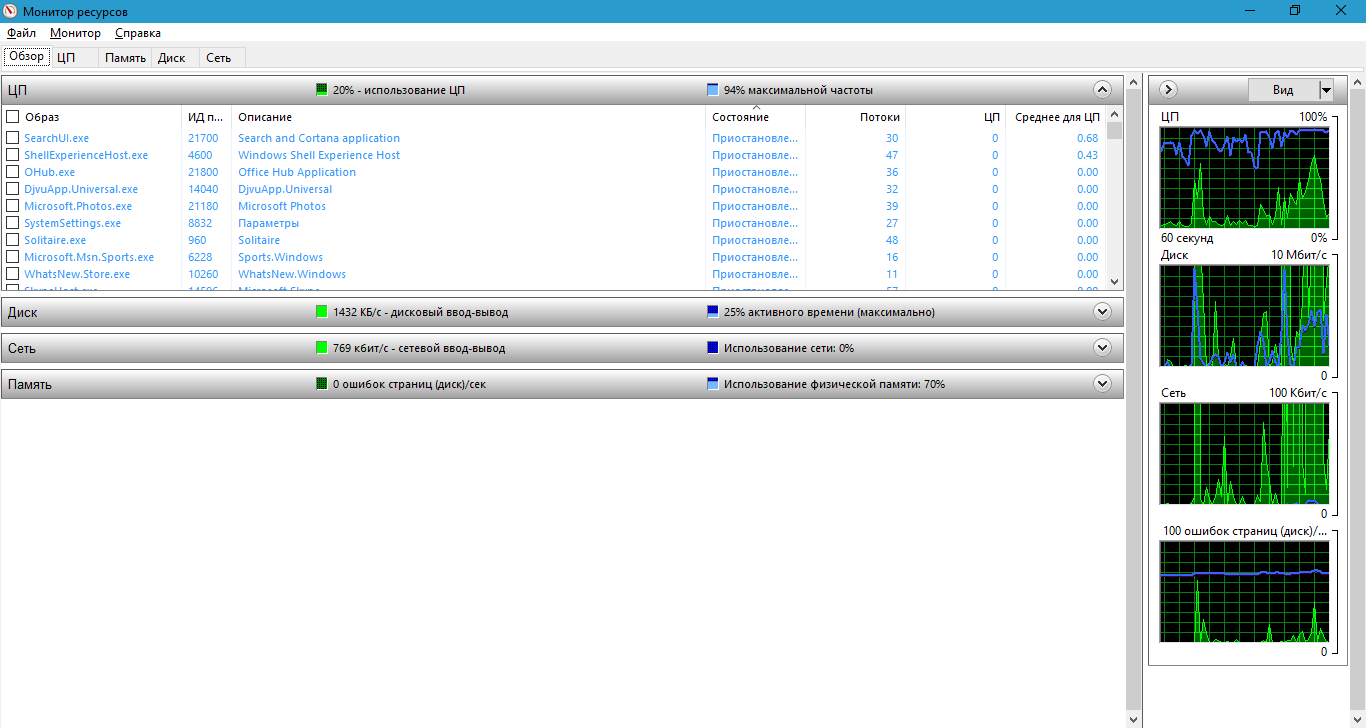


Рисунок 8. Интерфейс монитора ресурсов Windows

Монитор ресурсов (рисунок 8) – стандартное приложение ОС Windows. Оно предоставляет пользователю информацию об использовании ресурсов ПК. Предоставляется информация о 4 параметрах: CPU, RAM, жестком диске и использовании сети.

### Spotlight



Рисунок 9. Интерфейс программы Spotlight

Spotlight (рисунок. 9) – система мониторинга ресурсов, разработанная Quest Sowftare, которой сейчас владеет Dell. В отличии от многих других сторонних систем мониторинга, Spotlight представляет собой отдельное приложение. Это позволяет дать пользователю намного больше информации о ресурсах системы и их использовании.

Вышеприведенные программы имеют богатый функционал, но от реализуемой программы в данной работе их отличает отсутствие модульности и ориентированность лишь а одну платформу.

### KDE System Guard

KSysguard – сетевое приложение для KDE, осуществляющее мониторинг системы и задач. Это средство может работать в ssh сессии. В нем имеется масса возможностей, например, такие, как клиент/серверные технологии, что позволяет осуществлять мониторинг локальных и удаленных хостов. В графической оболочке используются так называемые сенсоры, которые собирают информацию, выдаваемую приложением. Сенсор может возвращать либо простое значение, либо более сложно сформированные данные, например, таблицы. Для каждого вида данных предлагается один или несколько вариантов выдачи данных. Изображаемые данные сгруппированы в виде отдельных листов, которые можно сохранять и загружать независимо друг от друга. Таким образом, KSysguard не только менеджер простых задач, но также и мощное средство для управления большим серверным хозяйством.

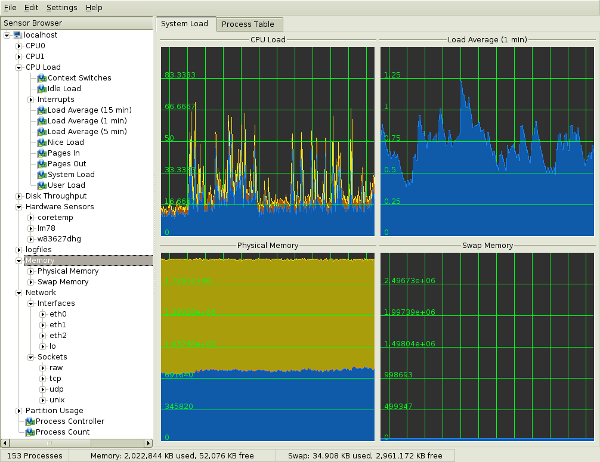


Рисунок 10. Графический интерфейс KDE SystemGuard

## Постановка задачи

Рассмотрев систему в окружении, изучив основную технологию для ее реализации, выяснив требования к системе и изучив заинтересованные стороны, представленные в [5] перейдем к постановке задачи.

Для этого построим диаграмму постановки задачи.

На рисунке 11 показаны основные задачи данной работы.

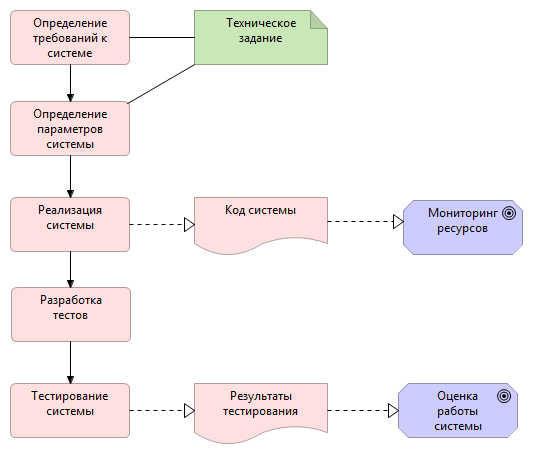


Рисунок 11. Постановка задачи

Целью данной работы является разработка системы мониторинга ресурсов, способная предоставлять информацию об использовании различных компонентов рабочей системы. Разработанную систему мониторинга должно быть легко модернизировать и дополнять новыми видами мониторов ресурсов.

# Проектно-конструкторская часть

Теперь, изучив технологии, которые будут применены при создании системы, рассмотрев взаимодействие между системой и окружающим миром и имея представление о уже разработанных аналогах системы, перейдем к детальному описанию архитектуры и внутреннего устройства разрабатываемой программы.

## Описание средств разработки

Основной средой разработки сервиса мониторинга ресурсов была выбрана платформа Eclipse.

Eclipse является бесплатной программной платформой с открытым исходным кодом, контролируется организацией Eclipse Foundation. Написана на языке программирования Java и основной целью её создания является повышение продуктивности процесса разработки программного обеспечения.

Претендует на статус наиболее популярной Java IDE и является единственным конкурентом такой мощной платформы как NetBeans.

Но в отличие от NetBeans который для создания элементов пользовательского интерфейса использует платформо-независимую библиотеку Swing, в Eclipse используется платформо-зависимая библиотека SWT - Standard Widget Toolkit.

IDE разработанные на базе платформы Eclipse применяются для создания программного обеспечения на различных языках программирования, так как Eclipse является платформой для разработки любых интегрированных сред программирования и расширений для себя же, по принципу "Плагины для Eclipse разрабатываются в самой Eclipse".

### Особенности платформы Eclipse

* Кроссплатформенность - работает под операционными системами Windows, Linux, Solaris и Mac OS X.
* Используя Eclipse можно программировать на множестве языков, таких как Java, C и C++, PHP, Perl, Python, Cobol и других.
* Является фреймворком для разработки других инструментов и предлагает обширный набор API для создания модулей.
* Используя подход RCP (Rich Client Platform) Eclipse является инструментом для создания практически любого клиентского программного обеспечения.
* Гибкость и расширяемость достигается благодаря модульности платформы.

### Архитектура платформы Eclipse

Основным элементом является исполняющая среда - Eclipse Runtime, в которой выполняются коды расширений и модулей. Она обеспечивает всю базовую функциональность платформы - управление расширениями и обновлениями, взаимодействие с операционной системой, обеспечение работы системы помощи.

Следующим элементом является собственно IDE - она отвечает за управление основными элементами программы, их расположением и настройками, управление проектами, отладку и сборку проектов, поиск по файлам и командную разработку.

В стандартную поставку Eclipse SDK включены два плагина - Java Development Tools или JDT, и Plug-in Developer Environment или PDE, таким образом мы получаем полностью готовую IDE для Java программирования и для разработки расширений для Eclipse.

Eclipse SDK - это минимальная версия, идеальная для знакомства с платформой и обучения. В дальнейшем, определившись с целями, вы можете скачать и использовать любую подходящую для ваших задач сборку, уже укомплектованную необходимыми расширениями.

## Архитектура системы

### Диаграмма компонентов

Диаграмма компонентов описывает особенности физического представления системы [6]. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ.

Рассмотрим диаграмму компонентов (рисунок 12). Сервис запускается в реализации OSGi для Eclipse Equinox.

Различные мониторы, которые получают информацию о этих устройствах. Поток данных идет через мониторы к обработчику данных. Там информация обрабатывается и передается на компонент взаимодействия.

Также с помощью компонента взаимодействия пользователь может управлять мониторами. Это происходит через bundle mananger, с помощью которого используется менеджер мониторов, в функции которого входит добавление/удаление и управление мониторами.

Зеленым цветом отмечены реализованные компоненты, красным цветом – компоненты, реализованные с помощью сторонних библиотек. Желтым цветом отмечен сервис мониторинга ресурсов.

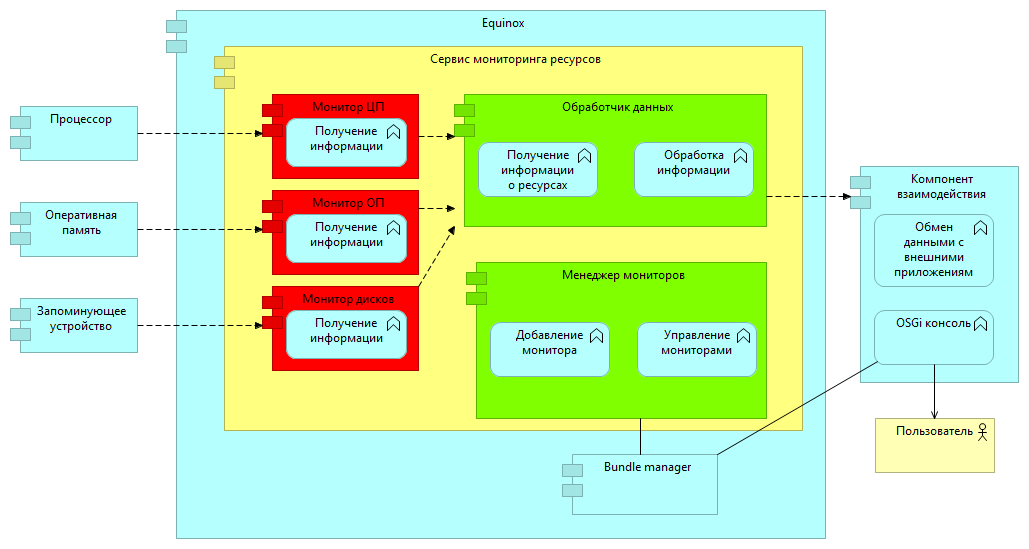


Рисунок 12. Диаграмма компонентов

Диаграмма модулей отображает распределение объектов по модулям в физическом проектировании системы.

На модульной диаграмме (рисунок 13) показаны основные модули системы и потоки данных между ними.

На этой диаграмме раскрыта проблема модульности системы:

Универсальный интерфейс передачи данных, который может передавать информацию от любых мониторов в обработчик данных.

Менеджер мониторов, способный управлять и добавлять новые мониторы в сервис.

## Выбор используемых библиотек

### Библиотека OperatingSystemMXBean

При проектировании системы, встает проблема получения информации о устройствах. Эта проблема решается с помощью сторонних библиотек.

Библиотека OperatingSystemMXBean разработана компанией Sun [7].

Данная библиотека предоставляет набор методов для получения информации о состоянии процессора и оперативной памяти системы. Она является кроссплатформенной, что обеспечивает ее стабильную работу при использовании различных операционных систем, таких как Windows, OS X, Linux.

Эта библиотека полезна для мониторинга основных параметров системы.

Рассмотрим методы библиотеки OperatingSystemMXBean подробнее в таблице 4.

Таблица 4- Описание методов библиотеки OperatingSystemMXBean

|  |  |  |
| --- | --- | --- |
| Метод | Тип возвращаемого объекта | Описание |
| getCommittedVirtualMemorySize() | long | Возвращает объем виртуальной памяти, которая гарантированно будет доступна для запущенного процесса в байтах, или -1, если эта операция не поддерживается. |
| getFreePhysicalMemorySize() | long | Возвращает количество свободной физической памяти в байтах. |
| getFreeSwapSpaceSize() | long | Возвращает количество свободного пространства файла подкачки в байтах. |
| getProcessCpuLoad() | double | Возвращает "недавнее использование центрального процессора" для процесса виртуальной машины Java. Это значение является двойным в интервале [0.0,1.0]. Значение 0.0 означает, что ни один из процессоров не обрабатывал потоки виртуальной машины Java за последний период наблюдения, в то время как значение 1.0 означает, что все процессоры активно обрабатывали потоки из виртуальной машины Java 100% от времени, в течение последнего периода наблюдения. Все значения между 0,0 и 1,0 возможны в зависимости от операций, происходящих в системе и виртуальной машине Java. Если платформа не поддерживает данную операцию, метод возвращает отрицательное значение. |
|  |  |  |
| Продолжение таблицы 4 |  |  |
| Метод | Тип возвращаемого объекта | Описание |
| getProcessCpuTime() | long | Возвращает процессорное время, используемое процессом, на котором выполняется виртуальная машина Java в наносекундах. Возвращенное значение имеет точность до наносекунд. Этот метод возвращает -1, если платформа не поддерживает эту операцию. |
| getSystemCpuLoad() | double | Возвращает "недавнее использование центрального процессора" для всей системы. Это значение является двойным в интервале [0.0,1.0]. Значение 0.0 означает, что все процессоры простаивали в течение последнего наблюдаемого периода времени, в то время как значение 1,0 означает, что все процессоры активно работали 100% времени, в течение последнего периода наблюдения. Все значения между 0,0 и 1,0 возможны в зависимости от операций, происходящих в системе. Если платформа не поддерживает данную операцию, метод возвращает отрицательное значение. |
| getTotalPhysicalMemorySize() | long | Возвращает общее количество физической памяти в байтах. |
| getTotalSwapSpaceSize() | long | Возвращает общий размер файла подкачки в байтах. |

### Библиотека Sigar

Sigar – библиотека, используемая для мониторинга системных ресурсов. Она поддерживает большое количество платформ и предоставляет простой и удобный API для работы [8].

Предоставляемый данной библиотекой интерфейс способен предоставлять информацию о:

* системной памяти
* размер swap-файла
* загруженности центрального процессора
* продолжительности работы
* потреблении памяти и ЦП
* состоянии, параметрах, используемых файлах по отдельным процессам
* данных по сетевому интерфейсу
* таблице tcp/udp соединений
* таблице сетевых маршрутов и т.д.

Эта информация может быть доступна на большинстве операционных систем, но своим специфичным способом.

Также кроме языка Java имеется поддержка таких языков программирования как python, perl, php, ruby и .net.

Поддерживаемые операционные системы: Linux, Windows, Solaris, AIX, HP-UX, Free-BSD, OS X.

Из приведенной выше информации видно, что эта библиотека может быть полезна при расширении функционала системы мониторинга ресурсов, добавлении новых параметров для мониторинга и использовании системы на различных устройствах.

### Диаграмма модулей

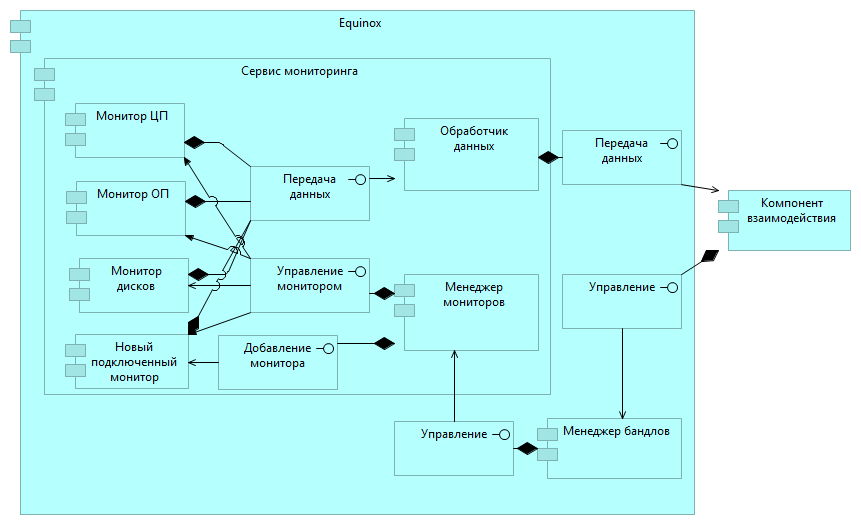


Рисунок 13. Диаграмма модулей

На модульной диаграмме (рисунок 13) показаны основные модули системы и интерфейсы между ними.

На этой диаграмме раскрыта проблема модульности системы:

* Универсальный интерфейс передачи данных позволяет передавать информацию от любых мониторов в обработчик данных.
* Менеджер мониторов, способный управлять и добавлять новые мониторы в сервис.

### Диаграмма классов

Диаграмма классов (рисунок 14) демонстрирует классы системы, их атрибуты и методы, а также их взаимодействие.

Диаграмма классов один из ключевых компонентов в объектно-ориентированном моделировании. Классы представлены в виде диаграммы, состоящей из трех компонентов: имени класса, атрибутов класса и методов класса.



Рисунок 14. Диаграмма классов

### Диаграмма развертывания

Диаграмма развертывания отображает физическое развертывание компонентов на оборудовании. Сервис развертывается на ПК. Вывод информации может происходить через графический интерфейс, стороннее приложение или с помощью OSGi Remote services [9].

Развертывание системы разрабатываемой системы будет происходить по способу, представленному на рисунке 15.

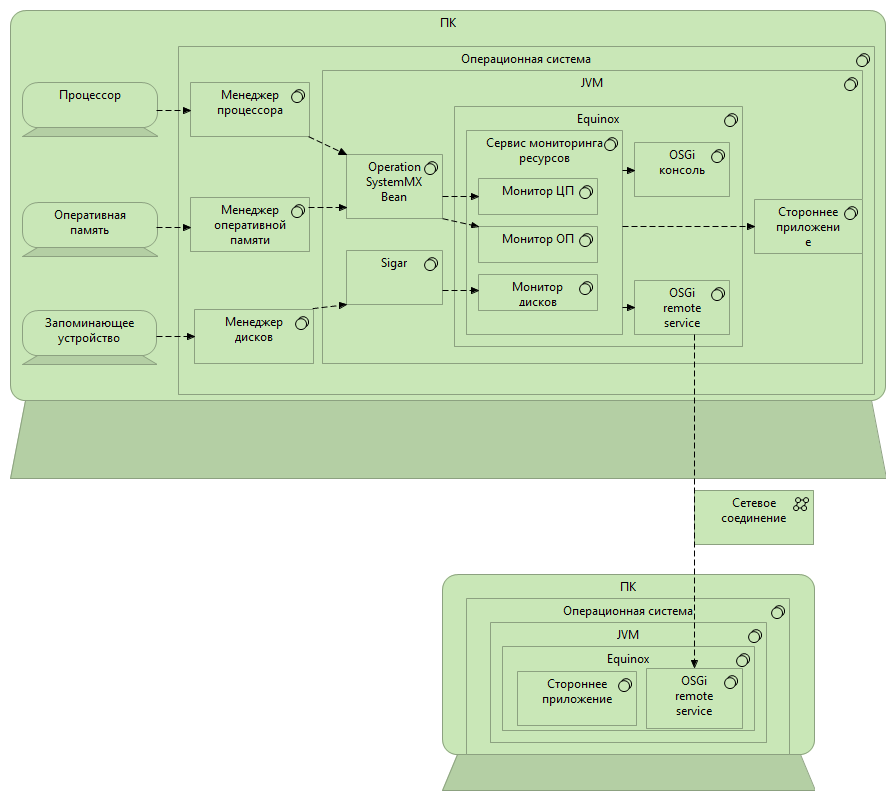


Рисунок 15. Диаграмма развертывания

# Технологическая часть

В технологической части рассмотрены используемые технологии и показаны результаты тестирования системы.

## Используемые средства

В данной работе использовались следующие средства:

* среда разработки Eclipse. Eclipse — свободная интегрированная среда разработки модульных кроссплатформенных приложений. Развивается и поддерживается Eclipse Foundation [10];
* язык программирования Java;

## Работа Java – компилятора

Сборка (англ. assembly) — двоичный файл, содержащий исполняемый код программы или (реже) другой подготовленный для использования информационный продукт.

Автоматизация сборки — этап написания скриптов или автоматизация широкого спектра задач применительно к ПО, применяемому разработчиками в их повседневной деятельности, включая такие действия, как:

* компиляция исходного кода в бинарный код;
* сборка бинарного кода;
* выполнение тестов;
* разворачивание программы на производственной платформе;
* написание сопроводительной документации или описание изменений новой версии;

Для автоматизации сборки проектов традиционно используют системы сборки, такие как make на Unix подобных системах и nmake для компилятора Microsoft. Также традиционно написание файлов для сборки проекта под эти системы является задачей нетривиальной. Конечно, пользуясь только Mictosoft Visual Studio можно даже не подозревать о существовании этих файлов, так как интегрированная среда разработки достаточно удобно скрывает всю схему работы, оставляя снаружи несколько диалоговых окон и кнопку Build. Но для сложных проектов, использующих массу сторонних библиотек и кроссплатформенных проектов такой подход часто оказывается неприемлемым.

В рамках технологии .NET — двоичный файл, содержащий управляемый код. Когда компилятор платформы .NET создает EXE или DLL модуль, содержимое этого модуля называется сборкой. Сборка содержит в себе: номер версии, метаданные и инструкции.

Опишем выполнение java – программы.

Текст программы — это исходный код программы на языке java.

Дополнения — это классы, которые необходимо учитывать во время сборки (библиотеки).

В итоге мы получаем набор файлов с расширением class. То есть, если мы используем сторонние библиотеки – мы должны указать их при сборке. Это могут быть скомпилированные классы или собранные подсистемы.

## Мониторы ресурсов

Важным компонентом системы являются мониторы ресурсов. Они предоставляют информацию о использовании вычислительных ресурсов системы.

В данной работе были реализованы основные мониторы для тестирования работоспособности сервиса, а конкретно монитор процессора, оперативной памяти и запоминающего устройства.

Их реализация не является обязательной частью работы, так как система имеет открытый API и, следовательно, доступна для разработчиков, которые могут реализовывать различные мониторы ресурсов.

Код одного из тестовых мониторов приведен в Приложении А.

## Получение бинарной сборки системы

Для получения бинарной сборки системы был использован Tycho [].

Tycho представляет собой набор Maven-плагинов для построения компонентов Eclipse через систему Maven. Tycho поддерживает процесс сборки Eclipse плагинов и OSGi бандлов. Tycho использует компоненты метаданных Eclipse. Например, Tycho определяет зависимости плагина через файл MANIFEST.MF.

Основная функциональность Tycho обеспечивается Tycho-Maven-плагином. Это плагин для Maven, поддерживающий создание компонентов в Eclipse. Он также позволяет Maven понять какие типы пакетов используются.

Если добавить этот плагин в файл сборки Maven, то Maven загрузит Tycho и использует его для построения в компонентов в Eclipse.

Построение с помощью Tycho конфигурируется с помощью стандартных конфигурационных файлов Maven, обычно называемых pom.xml или POM файлы.

Для построения с помощью Tycho, как правило, создается один файл POM для основной конфигурации (конфигурационный POM) и еще один, который включает в себя компоненты Eclipse (модули) предназначенный для сборки (главный POM).

Каждый компонент Eclipse, может иметь свой оригинальный файл POM, но Tycho позволяет создавать POM файлы по умолчанию, на основе существующих метаданных. Это основано на функционале Maven под названием “polyglot”, введенном в версии Maven 3.3.1. Если сгенерированные по умолчанию файлы не являются достаточными для компонента Eclipse, всегда можно создать специализированный POM файл для компонента.

Чтобы включить автоматическое определение информации о сборке для компонентов Eclipse, нужно добваить дескриптор .mvn / extensions.xml в корень каталога. Этого требует Apache Maven 3.3 и Maven Tycho 0.24.0.

Содержание .mvn / extensions.xml должно выглядеть следующим образом:

<?xml version="1.0" encoding="UTF-8"?>

<extensions>

<extension>

<groupId>org.eclipse.tycho.extras</groupId>

<artifactId>tycho-pomless</artifactId>

<version>0.25.0</version>

</extension>

</extensions>

Следующие правила используются для автоматической генерации POM файлов. В первую очередь .qualifier из компонентов Eclipse автоматически отображаются в -snapshot для сборки Maven. Рассмотрим подробнее в таблице 5.

Таблица 5. Отображение свойств POM файла

|  |  |
| --- | --- |
| Свойство | Отображение свойства |
| packaging | eclipse-plugin если MANIFEST.MF обнаружен  eclipse-feature если feature.xml обнаружен  eclipse-test-plugin если Bundle-SymbolicName заканчивается на .tests |
| groupId | Такой же, как и в родительском POM файле |
| artifactId | eclipse-plugin: Bundle-SymbolicName из файла MANIFEST.MF  eclipse-feature: featureid из файла feature.xml |
| version | Bundle-Version из файла MANIFEST.MF  или  Feature version из файла feature.xml |

Если нужно определить файл POM для компонента Eclipse, необходимо указать атрибут упаковки в файле POM. Этот атрибут определяет, какой компонент Eclipse создается.

Таблица 6. Атрибуты POM файла

|  |  |
| --- | --- |
| Атрибут | Описание |
| eclipse-plugin | Для плагинов |
| eclipse-test-plugin | Для тестовых плгинов |
| eclipse-feature | Для свойств |
| eclipse-repository | Для p2 обновлений и продуктов Eclipse |
| eclipse-target-definition | Объявление целевой папки для сборки |

В дополнение к атрибутам упаковки, POM из компонента Eclipse также должен указывать имя и версию компонента. Artefact ID и version в файле POM должны соответствовать Bundle-Symbolic-Name и Bundle-Version из файла MANIFEST.MF.

Каждый модуль имеет конфигурационный файл pom.xml, который определяет атрибуты соответствующие Eclipse компоненту. Он также должен содержать ссылку на основной POM или POM файл конфигурации чтобы при сборке была определена правильная конфигурация.

Нижеприведенный листинг содержит пример POM файла для плагина.

<?xml version="1.0" encoding="UTF-8"?>

<project>

<modelVersion>4.0.0</modelVersion>

*<!-- Link to the parent pom -->*

<parent>

<artifactId>com.example.todo.build.parent</artifactId>

<groupId>com.example.e4.rcp</groupId>

<version>0.1.0-SNAPSHOT</version>

<relativePath>../com.example.todo.build.parent</relativePath>

</parent>

<groupId>com.example.e4.rcp</groupId>

<artifactId>com.example.e4.bundleresourceloader</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>eclipse-plugin</packaging>

</project>

Для того, чтобы иметь некоторое разделение плагинов, принято иметь отдельные папки для них в файловой системе как на рисунке 16.

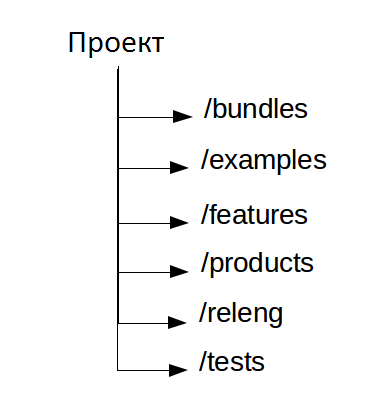


Рисунок 16. Расположение папок при сборке

Теперь, рассмотрев основной алгоритм и правила сборки проекта с помощью Tycho Maven, покажем сборку разработанного проекта.

Для сборки проекта в ОС Windows проделаем следующее:

Так как стандартные настройки Maven не включают в себя индексную систему, то будет невозможно искать и добавлять зависимости в POM файл. Поэтому для скачивания перейдем в настройки Maven и поставим галочку напротив опции “Download repository index from startup”.

Далее расположим файлы проекта в соответствии с вышеприведенным изображением. Проект будет выглядеть как на рисунке 17.

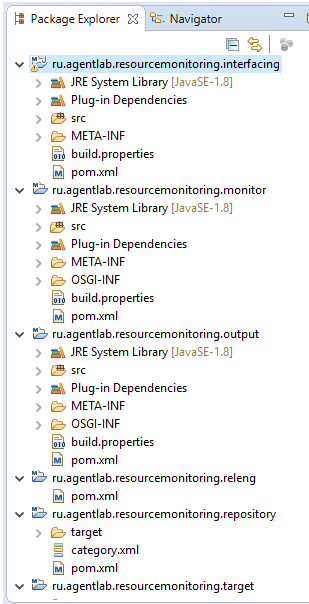


Рисунок 17. Расположение файлов и папок в проекте

А POM файл для целевой папки будет иметь следующий вид:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<artifactId>ru.agentlab.resourcemonitoring.target</artifactId>

<packaging>eclipse-target-definition</packaging>

<parent>

<groupId>ru.agentlab.resourcemonitoring</groupId>

<artifactId>ru.agentlab.resourcemonitoring.releng</artifactId>

<version>4.12.0-SNAPSHOT</version>

<relativePath>../ru.agentlab.resourcemonitoring.releng/pom.xml</relativePath>

</parent>

</project>

Для инициализирования сборки в контекстном меню [11] проекта перейдем в раздел Run As и выберем Maven Build… (рисунок 18)

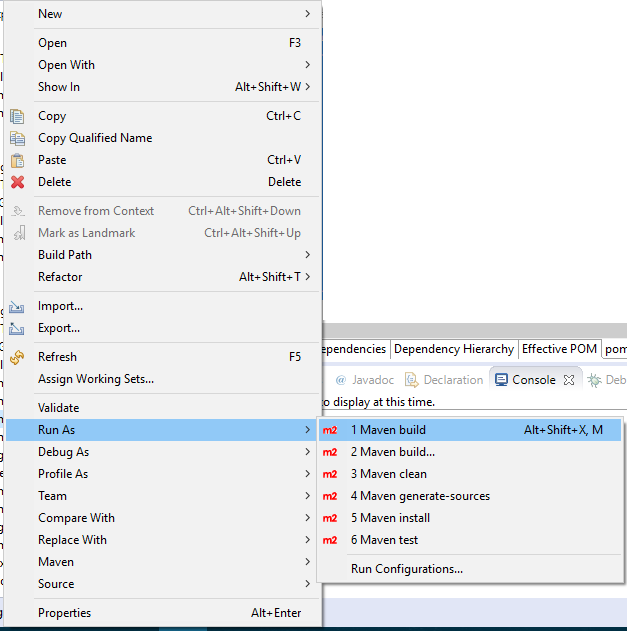


Рисунок 18. Запуск сборки проекта

Откроется окно с настройками конфигурации запуска. В поле Goal введем clean install, как показано на рисунке 19.

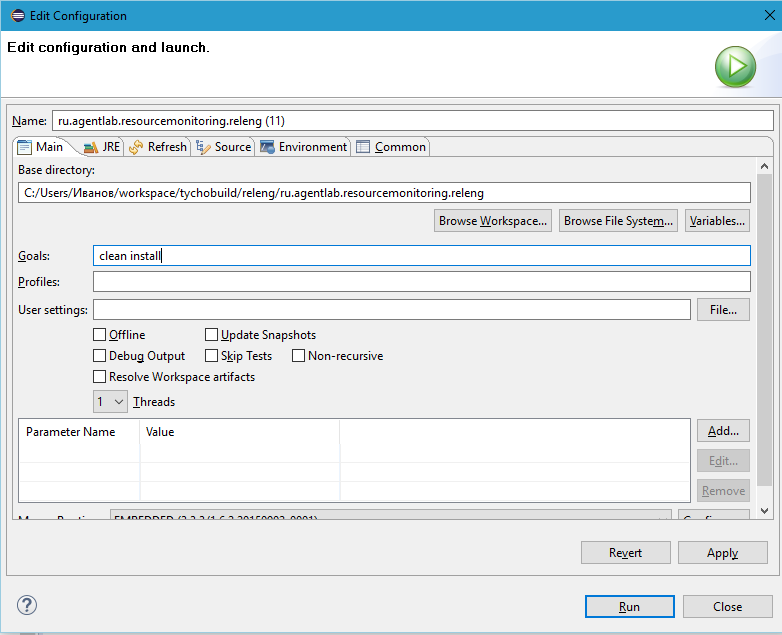


Рисунок 19. Окно конфигурации запуска.

После окончания сборки необходимо обновить проект. В папке target появятся артефакты сборки, в том числе и .jar файл.

## Тестирование системы

Перед началом тестирования был разработан план тестирования [12], представленный в таблице 7 и тестовые случаи, представленные в таблице 8.

После этого было проведено тестирование, и полученные результаты были представлены в таблице 9.

Таблица 7- План тестирования

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Объект тестирования | Уровень тестирования | Тестируемая ошибка | Виды деятельности | Частота тестирования | Ответственный исполнитель |
| Система мониторинга ресурсов | Системный | Неработоспособность системы | Выбор тестовых случаев | До сдачи варианта  системы заказчику | Тестировщик­ |

Таблица 8 – Тестовые случаи

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестируемые части системы | Входные данные | Ожидаемые результаты |
| 1 | Вся система | Запуск системы в обычном режиме на длительное время (несколько часов) | Вывод сообщения о старте системы, вывод значений на экран. Стабильная работа системы. |
| 2 | Вся система | Запуск системы в обычном режиме на длительное время (несколько часов) | Вывод сообщения о старте системы, вывод значений на экран. Незначительное увеличение ресурсов, потребляемых сервисом. |
| 3 | Вся система | Запуск системы в обычном режиме | Вывод сообщения о старте системы, вывод значений на экран. Незначительное искажение данных о ресурсах системы. |

Таблица 9 – Результаты тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тестируемые части системы | Входные данные | Полученные данные | Заключение |
| 1 | Вся система | Запуск системы в обычном режиме на длительное время (несколько часов) | Вывод сообщения о старте системы, вывод значений на экран. Система работает стабильно. | Результаты соответствуют ожидаемым |
| 2 | Вся система | Запуск системы в обычном режиме на длительное время (несколько часов) | Вывод сообщения о старте системы, вывод значений на экран. Количество ресурсов, потребляемых сервисом не увеличилось. | Результаты соответствуют ожидаемым |
| 3 | Вся система | Запуск системы в обычном режиме | Вывод сообщения о старте системы, вывод значений на экран. Сервис незначительно искажает данные о ресурсах системы. | Результаты соответствуют ожидаемым |

Как видно из результатов тестирования, система работает соответственно ожиданиям. Она работает стабильно в течение длительного времени, в ней отсутствуют утечки памяти. Также разработанная система способна справляться с ошибками в подключаемых мониторах.

# ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы была обозначена проблематика, выявлены требования к системе на основе анализа заинтересованных сторон. Проведено исследование существующих аналогов, что показало отсутствие реализации агента мониторинга ресурсов для OSGi контейнера. Была разработана архитектура системы, построены соответствующие диаграммы, показывающие принцип работы системы взаимодействие модулей системы между собой, и выбраны библиотеки, позволяющие получить данные о расходовании ресурсов системы, описаны используемые для работы средства и проведено тестирование по составленному плану.

На основе проделанной работы был реализован агент мониторинга ресурсов, а конкретно его компоненты мониторинга, обработки данных и менеджер мониторов, а также компонент взаимодействия с другими агентами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Спецификация ArchiMate: [Электронный ресурс] – Режим доступа: [http://pubs.opengroup.org/architecture/archimate2-doc/toc.html - Дата обращения 15.12.2015](http://pubs.opengroup.org/architecture/archimate2-doc/toc.html%20-%20Дата%20обращения%2015.12.2015)
2. Спецификация OSGi: [Электронный ресурс] – Режим доступа: <https://www.osgi.org/developer/specifications/> - Дата обращения: 25.05.2016
3. Java Classloader: JAR hell: [Электронный ресурс] – Режим доступа: https://en.wikipedia.org/wiki/Java\_Classloader - Дата обращения: 25.05.2016
4. Alexander, Ian (Ian F.), 1954- Discovering requirements : how to specify products and services / Ian Alexander & Ljerka Beus-Dukic. p. cm. Includes bibliographical references and index. ISBN 978-0-470-71240-5 (pbk. : alk. paper)
5. Иванов В.С., 2015 – Курсовая работа по РПО, 25 с. / В.С. Иванов
6. Басс, Л., Клементс, П., и др. Архитектура программного обеспечения на практике. – СПб.: Питер, 2006. – 576 с. – (Классика Computer Science) – ISBN 5-469-00494-5.
7. Спецификация библиотеки OperatingSystemMXBean: [Электронный ресурс] – Режим доступа: http://docs.oracle.com/javase/7/docs/jre/api/management/extension/com/sun/management/OperatingSystemMXBean.html- Дата обращения: 25.05.2016
8. Спецификация библиотеки Sigar: [Электронный ресурс] – Режим доступа: https://support.hyperic.com/display/SIGAR/Home - Дата обращения: 25.05.2016
9. OSGi remote services and ECF: [Электронный ресурс] – Режим доступа: https://wiki.eclipse.org/OSGi\_Remote\_Services\_and\_ECF- Дата обращения: 25.05.2016
10. OSGi remote services and ECF: [Электронный ресурс] – Режим доступа: <https://wiki.eclipse.org/Main_Page> - Дата обращения: 25.05.2016
11. Сборка и выполнение Java программ [Статья]. URL: http://www.quizful.net/post/sborka\_i\_vipolnenie\_Java (дата обращения: 20.04.2016).
12. Д. Макгрегор, Д. Сайкс: Тестирование объектно-ориентированного программного обеспечения – ДиаСофт, 2007.

# ПРИЛОЖЕНИЕ А

**Листинг программы**

**Класс Service.java**

**package** ru.agentlab.resourcemonitoring.output;

**import** ru.agentlab.resourcemonitoring.interfacing.IService;

**public** **class** Consumer {

**private** IService service;

**public** **void** quote() **throws** InterruptedException {

System.***out***.println(service.getQuote());

}

**public** **synchronized** **void** setQuote(IService service) **throws** InterruptedException {

System.***out***.println("Service set.");

**this**.service = service;

System.***out***.println(service.getQuote());

}

**public** **synchronized** **void** undsetQuote(IService service) {

System.***out***.println("Service unset.");

**if** (**this**.service == service) {

**this**.service = **null**;

}

}

}

**Component.xml для класса Service.java**

<?xml version="1.0" encoding="UTF-8"?>

<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="ru.agentlab.resourcemonitoring.output">

<implementation class="ru.agentlab.resourcemonitoring.output. Consumer"/>

<reference bind="setQuote" cardinality="1..1" interface="ru.agentlab.resourcemonitoring.interfacing.IService" name="IService" policy="static" unbind="unsetQuote"/>

</scr:component>

**MANIFEST.MF для Service.java**

Manifest-Version: 1.0

**Bundle-ManifestVersion**: 2

**Bundle-Name**: Output

**Bundle-SymbolicName**: ru.agentlab.resourcemonitoring.output

**Bundle-Version**: 1.0.0.qualifier

**Bundle-RequiredExecutionEnvironment**: JavaSE-1.8

Service-Component: OSGI-INF/component.xml

**Import-Package**: ru.agentlab.resourcemonitoring.interfacing

**Класс IService.java**

**package** ru.agentlab.resourcemonitoring.interfacing;

**public** **interface** IService {

String getQuote() **throws** InterruptedException;

}

**Класс RAMMonitor.java**

**package** ru.agentlab.resourcemonitoring.monitor;

**import** java.lang.management.ManagementFactory;

**import** java.lang.management.OperatingSystemMXBean;

**import** ru.agentlab.resourcemonitoring.interfacing.IService;

**public** **class** Service **implements** IService {

@Override

**public** String getQuote() **throws** InterruptedException {

OperatingSystemMXBean bean = (com.sun.management.OperatingSystemMXBean) ManagementFactory.*getOperatingSystemMXBean*();

**long** freeMem = ((com.sun.management.OperatingSystemMXBean) bean).getFreePhysicalMemorySize();

**long** totalMem = ((com.sun.management.OperatingSystemMXBean) bean).getTotalPhysicalMemorySize();

**long** usedMem = totalMem - freeMem;

**float** totalMemFloat = (**float**)totalMem;

**float** usedMemFloat = (**float**)usedMem;

**float** usedMemPercent = usedMemFloat/totalMemFloat;

**int** roundedMemPercent = Math.*round*(usedMemPercent\*100);

**return**("Memory usage: " + roundedMemPercent + "%");

}

}

**Component.xml для класса RAMMonitor.java**

<?xml version="1.0" encoding="UTF-8"?>

<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="ru.agentlab.resourcemonitoring.monitor">

<implementation class="ru.agentlab.resourcemonitoring.monitor.Service"/>

<service>

<provide interface="ru.agentlab.resourcemonitoring.interfacing.IService"/>

</service>

</scr:component>

**MANIFEST.MF для Monitor.java**

Manifest-Version: 1.0

**Bundle-ManifestVersion**: 2

**Bundle-Name**: Monitor

**Bundle-SymbolicName**: ru.agentlab.resourcemonitoring.monitor

**Bundle-Version**: 1.0.0.qualifier

**Bundle-RequiredExecutionEnvironment**: JavaSE-1.8

Service-Component: OSGI-INF/component.xml

**Import-Package**: ru.agentlab.resourcemonitoring.interfacing

# ПРИЛОЖЕНИЕ Б

**Графическая часть квалификационной работы**

1. Диаграмма постановки задачи (рисунок Б.1);
2. Диаграмма проблематики (рисунок Б.2);
3. Диаграмма компонентов (рисунок Б.3);
4. Диаграмма модулей (рисунок Б.4);
5. Диаграмма классов (рисунок Б.5);
6. Диаграмма развертывания системы (рисунок Б.6);

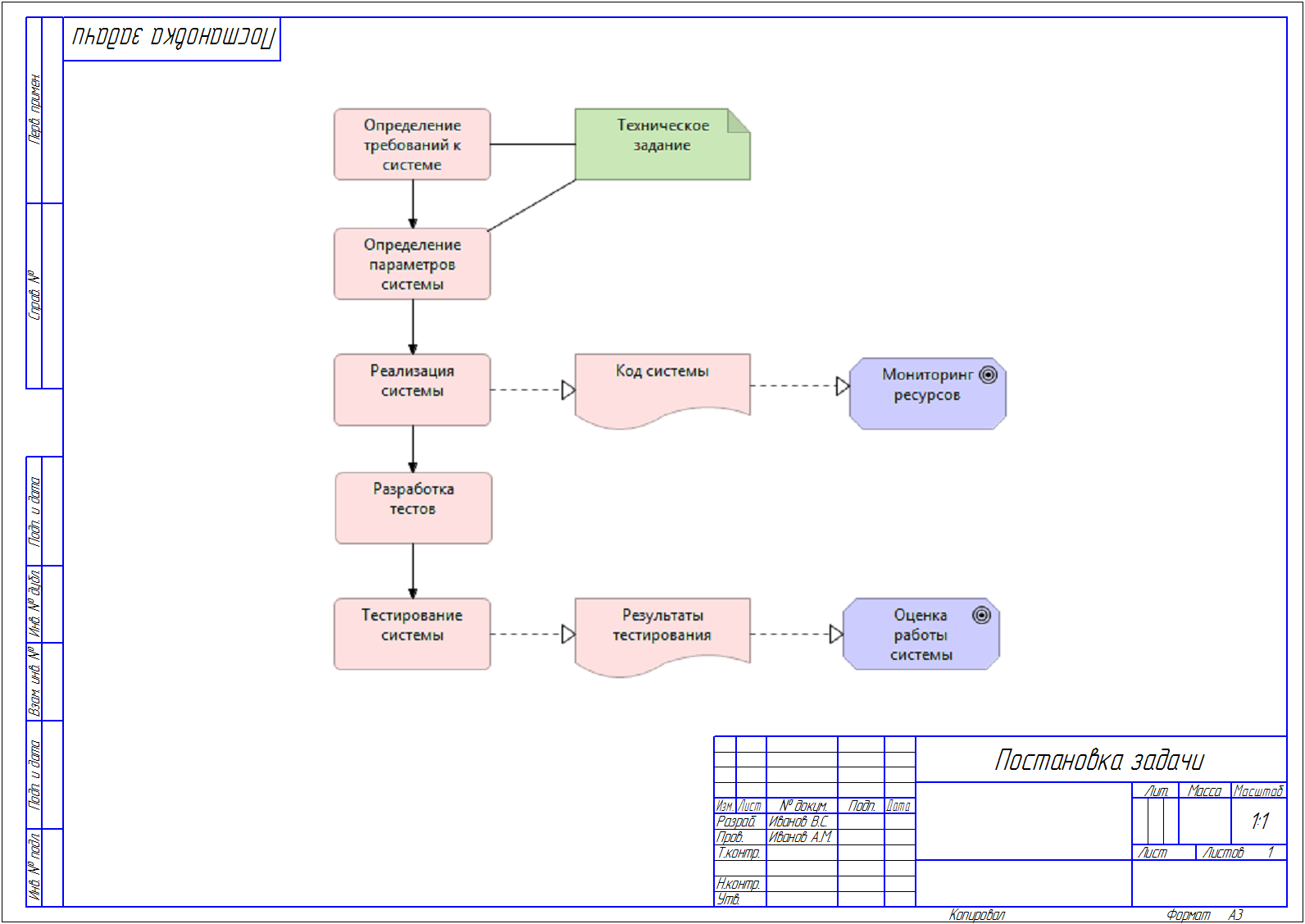


Рисунок Б.1- Постановка задачи

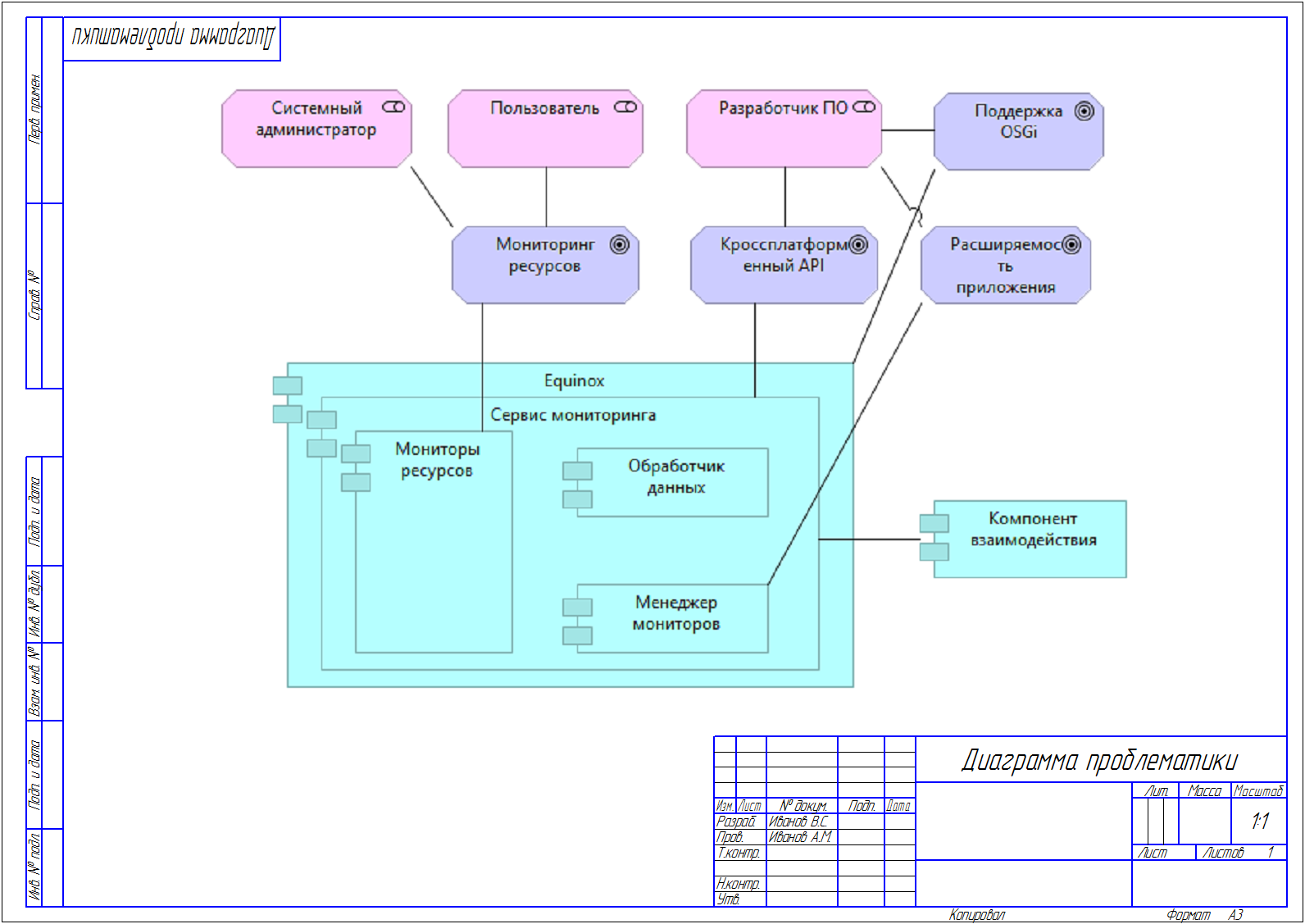


Рисунок Б.2 – Диаграмма проблематики

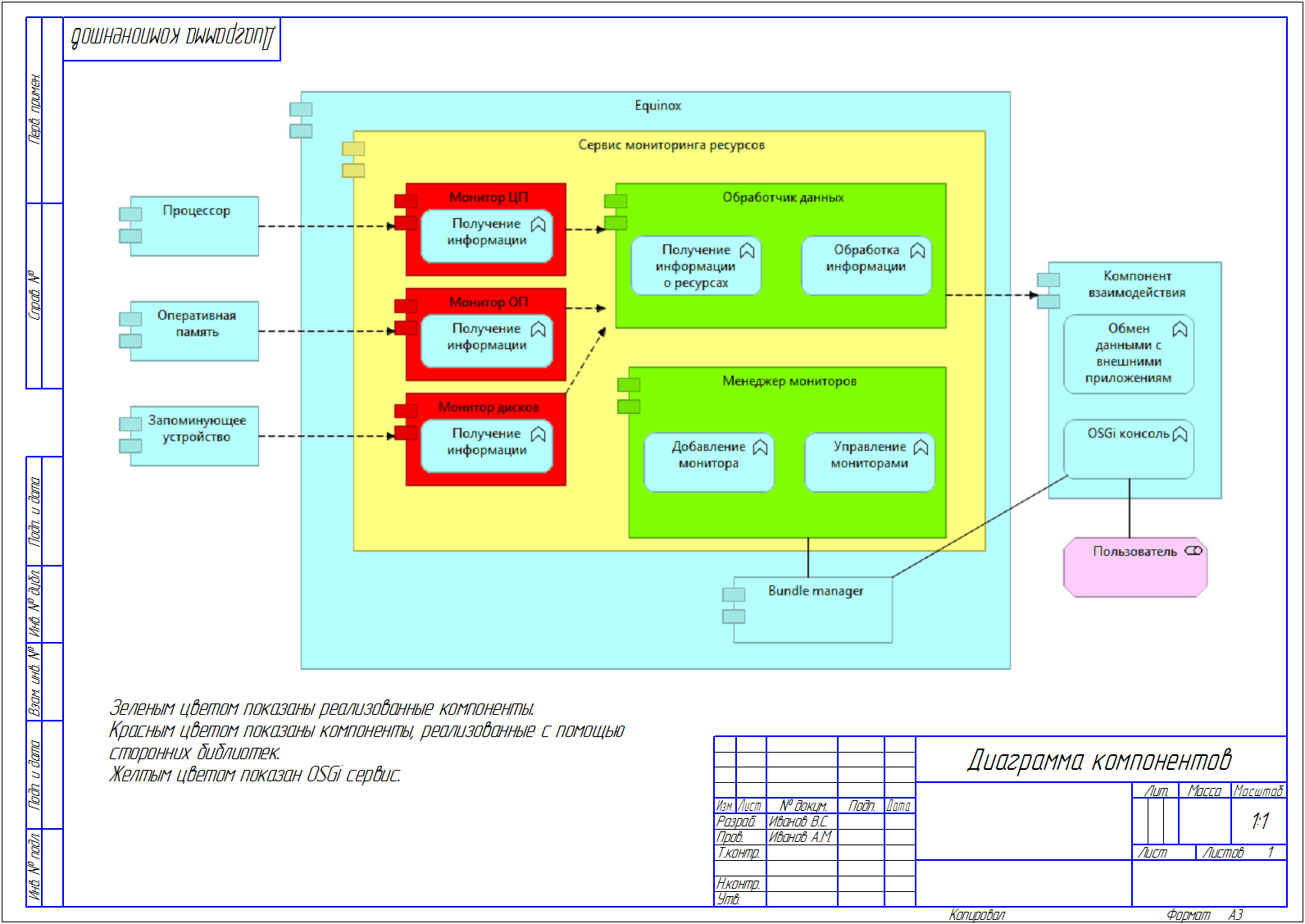


Рисунок Б.3 – Диаграмма компонентов

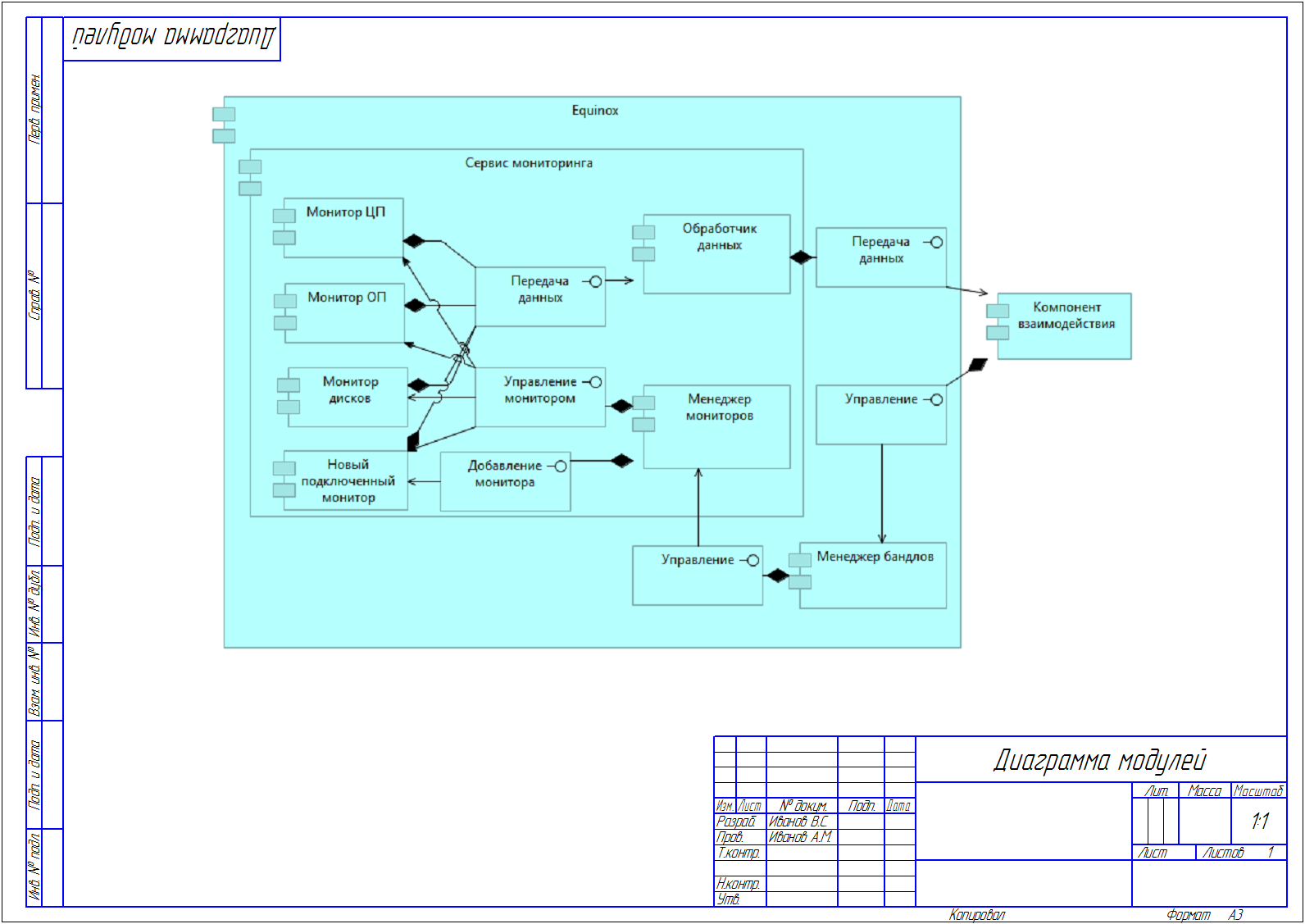


Рисунок Б.4 – Диаграмма модулей

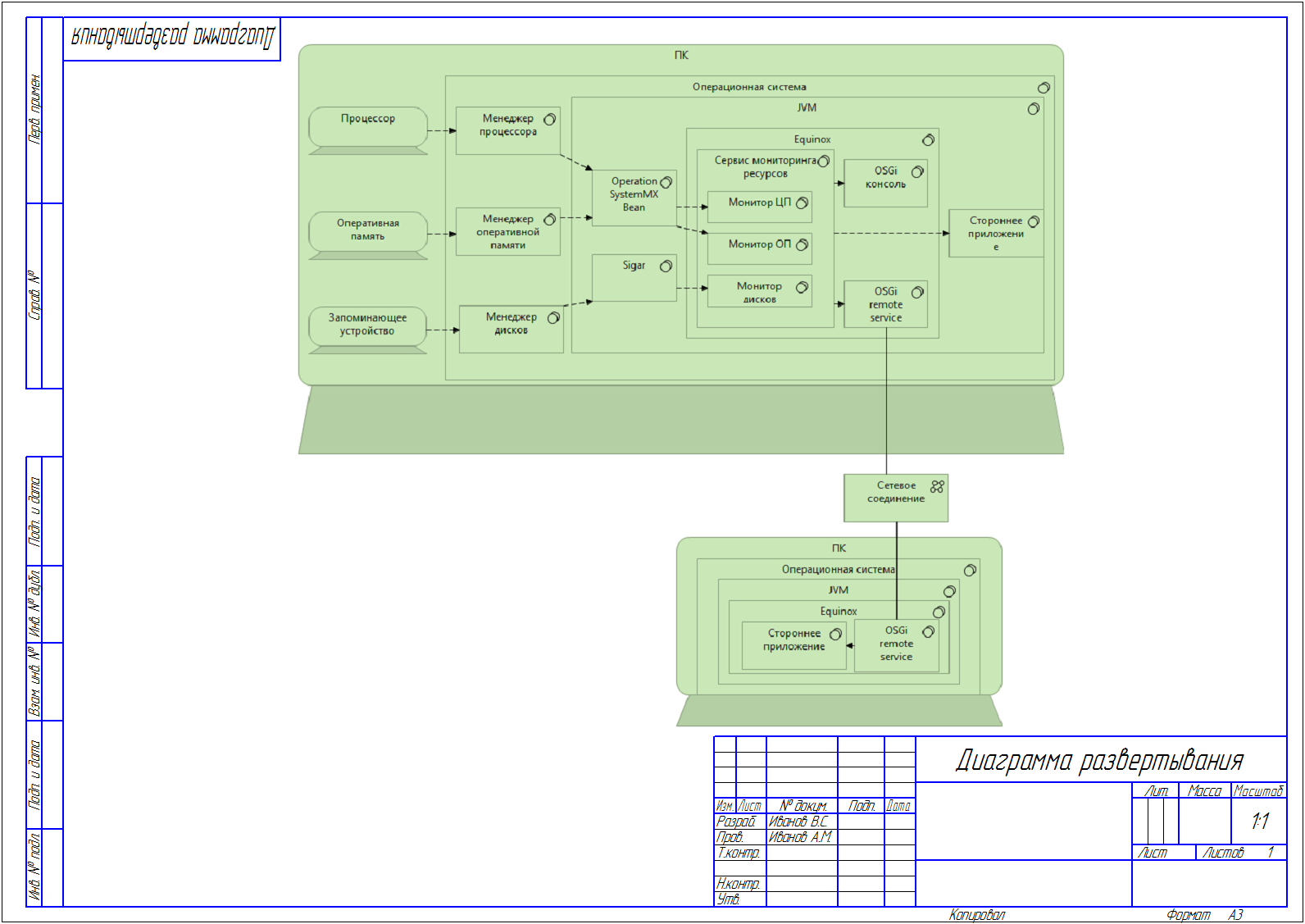


Рисунок Б.5 – Диаграмма развертывания

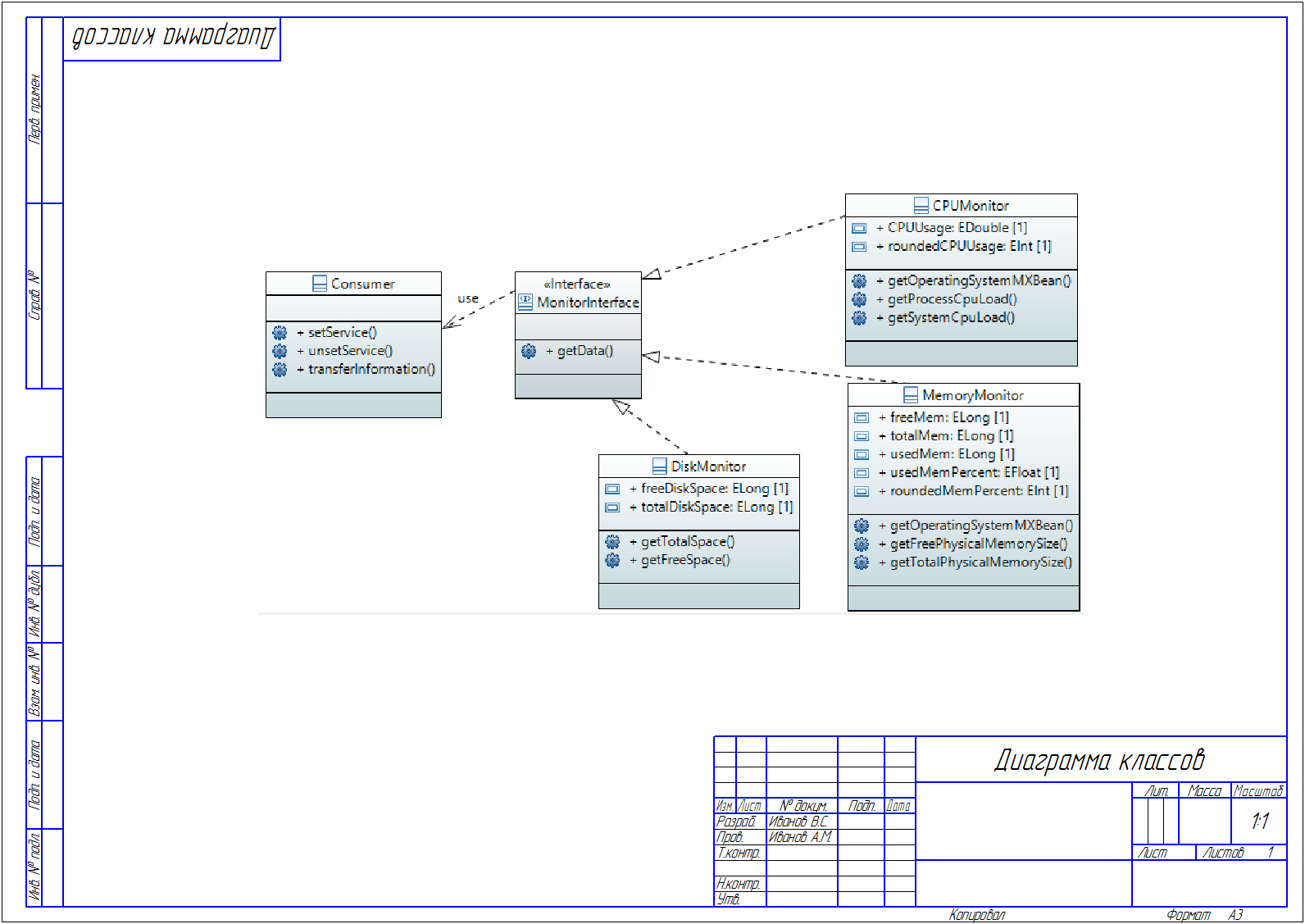


Рисунок Б.6 – Диаграмма классов