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 Apache UIMA OSGI компонента, распознающего позы и аннотирующего данные 3D камеры.

Студент группы ИУ3-73 28.12.2017 В.Э Большаков

Руководитель курсовой работы 28.12.2017 А.М Иванов

Москва, 2017

Содержание

[**Техническое задание**](#_q2lsdjl8evcp)4

[**1 Теоретическая часть**](#_481i776yolpm)4

[1.1 Apache UIMA](#_bdgp30j8ucnh) 4

[**2 Конструкторская часть**](#_yal4irqalpeg)6

[2.1 Структура проекта](#_t2kx9ztzscz7) 6

[2.1.1](#_yrs6vvyczrnd) Плагин com.bmstu.poses.capture 6

[2.1.2](#_pcm249ljgo66) Плагин com.bmstu.poses.capture.annotator 7

[2.1.3](#_f6aqntbykxa5) Плагин com.bmstu.poses.capture.generator 8

[2.1.4](#_f6aqntbykxa5) Плагин com.bmstu.poses.capture.serialization 8

[2.2 Диаграмма компонентов](#_xk9j266ue8xm) 9

[2.3 Диаграмма классов](#_gleda2ia4sai) 10

[**3 Технологическая часть**](#_utn3w6a1a9nz)11

[3.1 Запуск анализа данных](#_1kn55wx41bmi) 11

[3.2 Анализ исходного кода с помощью метрик качества](#_wtah32ql1q16) 11

[3.3 Анализ зависимостей в коде системы](#_malk4ghs54so) 14

[3.4 Тестирование на корректность работы](#_gfuhygqb5z97) 15

**Заключение** 17

[**Список литературы**](#_ozwq05s0bred)18

# 

# Техническое задание

Разработка Apache UIMA OSGi компонента, распознающего позы и аннотирующего данные 3D камеры:

* 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 Apache UIMA

Основным компонентом технологии Apache UIMA является AnalisysEngine. Это программа, анализирующая артефакты (например, текстовые документы) и выделяющая информацию из них. AnalisysEngine состоит из блоков, называемых аннотаторами. Аннотатор это такой компонент, который содержит логику, по который происходит анализ артифакта. Целью технологии Apache UIMA является ограничить каждый аннотатор только своей логикой, его не должны касаться детали развертывания или взаимодействия с другими аннотаторами. [1]

Когда аннотатор проходит по артефакту, он создает дополнительные данные (метаданные) об этом артефакте. Аннотаторы возвращают результаты анализа в виде типизированных структур объектов (Feature Structure), которые представляют собой просто структуры данных, имеющие тип и набор пар (атрибут, значение). Аннотация - это и есть определенный тип структуры объектов, прикрепленный к области анализируемого артефакта (например, к диапазону текста в документе). [1]

Интерфейс CAS предоставляет доступ к структурам объектов для разных аннотаторов, для этого он использует UIMA Type, который создается на основе используемой системы типов (Type System). Этот интерфейс поддерживает написание общих аннотаторов, которые могут работать на всех системах типов. Интерфейс JCas имеет преимущество, когда заранее используемые типы известны. Классы JCas соответствуют определенному типу UIMA, и такой класс включает специальные сеттеры и геттеры, имена которых соответствуют объектам. [1]

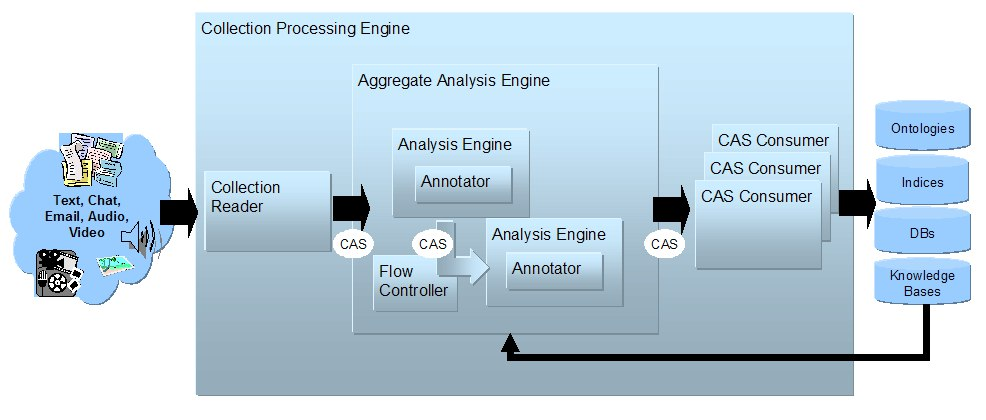


Рисунок 1 - Процесс анализа с помощью технологии UIMA

Рисунок 1 подробно иллюстрирует описанный выше принцип технологии UIMA.

Настройка компонентов UIMA, как правило, достигается путем создания файлов XML-дескрипторов, которые говорят платформе во время выполнения, как компоненты должны быть созданы и развернуты. Эти XML-файлы дескриптора очень тесно связаны с реализацией Java компонентов, которые они описывают.

Analysis Engine Descriptor содержит информацию об аннотаторе, который доступен без доступа к исходному коду. Он содержит информацию, такую как параметры конфигурации, структуры данных, входные и выходные данные аннотаторов и ресурсы, которые использует аннотатор. Дескриптор также используется средой UIMA для загрузки аннотатора.

Все данные, которые создаются аннотаторами или обмениваются между компонентами аннотатора, определяются в системе типов UIMA (UIMA type system). UIMA type system является частью файла дескриптора Analysis Engine, и каждый пользователь или приложение знает типы, которыми оперирует аннотатор. Это одно из основных преимуществ UIMA - структуры данных декларативно определены и хранятся внутри среды UIMA. Это увеличивает совместимость между компонентами и позволяет включать компоненты, разработанные при использовании разных языков программирования.

# 2 Конструкторская часть

2.1 Структура проекта

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

Таблица 1 – Разработанные плагины и их описание

|  |  |
| --- | --- |
| **Плагин** | **Содержимое** |
| com.bmstu.poses.capture | содержит описание модели скелета для распознавания |
| com.bmstu.poses.capture.annotator | содержит непосредственно аннотатор данных |
| com.bmstu.poses.capture.generator | содержит класс, который генерирует произвольные скелеты |
| com.bmstu.poses.capture.serialization | содержит классы, которые реализуют загрузку и выгрузку данных |

2.1.1 Плагин com.bmstu.poses.capture

Содержит 2 пакета:

* com.bmstu.poses.capture
* com.bmstu.poses.capture.model

Пакет com.bmstu.poses.capture содержит интерфейс ISkeletonStateIdentifier и включает в себя метод:

SkeletonState identify(Skeleton skeleton, Collection<SkeletonState> states), который определяет заданное состояние скелета.

Так же в пакете находится класс DefaultSkeletonStateIdentifier, реализующий этот интерфейс и содержащий помимо реализации метода identify ещё 4 метода, которые определяют, лежит ли точка в пределах погрешности.:

**private** **boolean** inBounds(Skeleton skeleton, SkeletonState state)

**private** **boolean** inBounds(SkeletonJoint skeletonJoint, SkeletonJoint stateJoint)

**private** **boolean** inBounds(Point3D position, Point3D statePosition)

**private** **boolean** inBounds(Quaternion orientation, Quaternion stateOrientation)

Пакет com.bmstu.poses.capture.model содержит 6 классов:

* JointType описывает типы связок (соединений) скелета. В общей сложности скелет строится по 15-ти точкам
* Point3D содержит координаты точки (связки) в пространстве, а также методы для получения этих координат
* Quaternion содержит кватернион для каждой точки (связки), необходимый для описания движения в пространстве, и методы для его получения
* Skeleton экземпляр этого класса будет нашим скелетом, в конструкторе класса убеждаемся, что скелет имеет все связки
* SkeletonJoint содержит информацию о типе, позиции и ориентации в пространстве для связки
* SkeletonState связывает имя позы и состояние скелета

2.1.2 Плагин com.bmstu.poses.capture.annotator

Содержит 2 пакета:

* com.bmstu.poses.capture
* com.bmstu.poses.capture.annotator

Пакет com.bmstu.poses.capture содержит 2 класса, созданные автоматически дескриптором:

* SkeletonState
* SkeletonState\_Type

Пакет com.bmstu.poses.capture.annotator содержит 2 класса:

* Annotator содержит метод annotate(Skeleton skeleton), который возвращает имя позы (состояния) скелета, метод loadSkeletonStates(), который возвращает состояние скелетов, метод createSkeleton(**int** index) который создает экземпляр класса Skeleton
* PosesAnnotator является непосредственно самим аннотатором данных, он наследует класс JCasAnnotator\_ImplBase и переопределяет единственный метод process(JCas aJCas).

2.1.3 Плагин com.bmstu.poses.capture.generator содержит 1 класс:

* SkeletonGenerator генерирует случайные скелеты, в частности генерирует связки, их позицию и ориентацию, содержит метод main, generateRandomSkeleton() и методы:
  + - * **private** SkeletonJoint generateSkeletonJoint(**int** jointType)
      * **private** Point3D generatePosition()
      * **private** Quaternion generateOrientation()

2.1.4 Плагин com.bmstu.poses.capture.serialization содержит 2 пакета:

* com.bmstu.poses.capture.serialization.import
* com.bmstu.poses.capture.serialization.export

Пакет com.bmstu.poses.capture.serialization.import содержит интерфейс ISkeletonReader и 2 класса:

* StringSkeletonReader извлекает параметры скелета из входного string со значениями, разделенными запятыми
* CsvSkeletonReader наследует указанный выше класс и реализует интерфейс, непосредственно считывает данные

Пакет com.bmstu.poses.capture.serialization.export содержит интерфейс ISkeletonWriter и реализующий его класс:

* CsvSkeletonWriter экспортирует данные в файл, приводя их при этом к нужному формату (разделитель - запятая)

2.2 Диаграмма компонентов

Предполагается, что данный проект будет использован в рамках более крупного проекта по распознаванию видео с 3D камеры. Изначально данные будут поступать из камеры и записываться в файл, после чего они будут анализироваться модулем StringSkeletonReader, который считает данные, выделив из них параметры скелета. Далее эти данные поступят на вход разрабатываемого в рамках курсового проекта модуля распознавания поз.

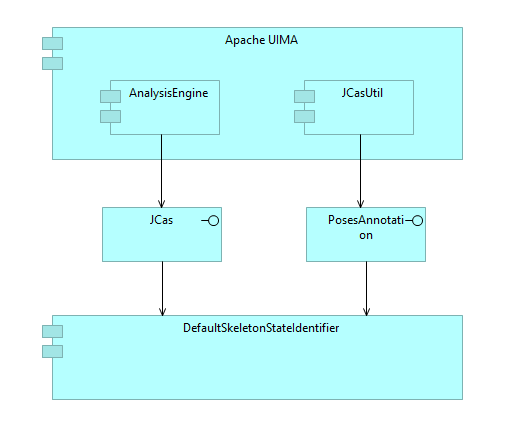


Рисунок 2 – Диаграмма компонентов

2.3 Диаграмма классов

Для наглядности, с помошью редактора UML Papirus была создана диаграмма классов.

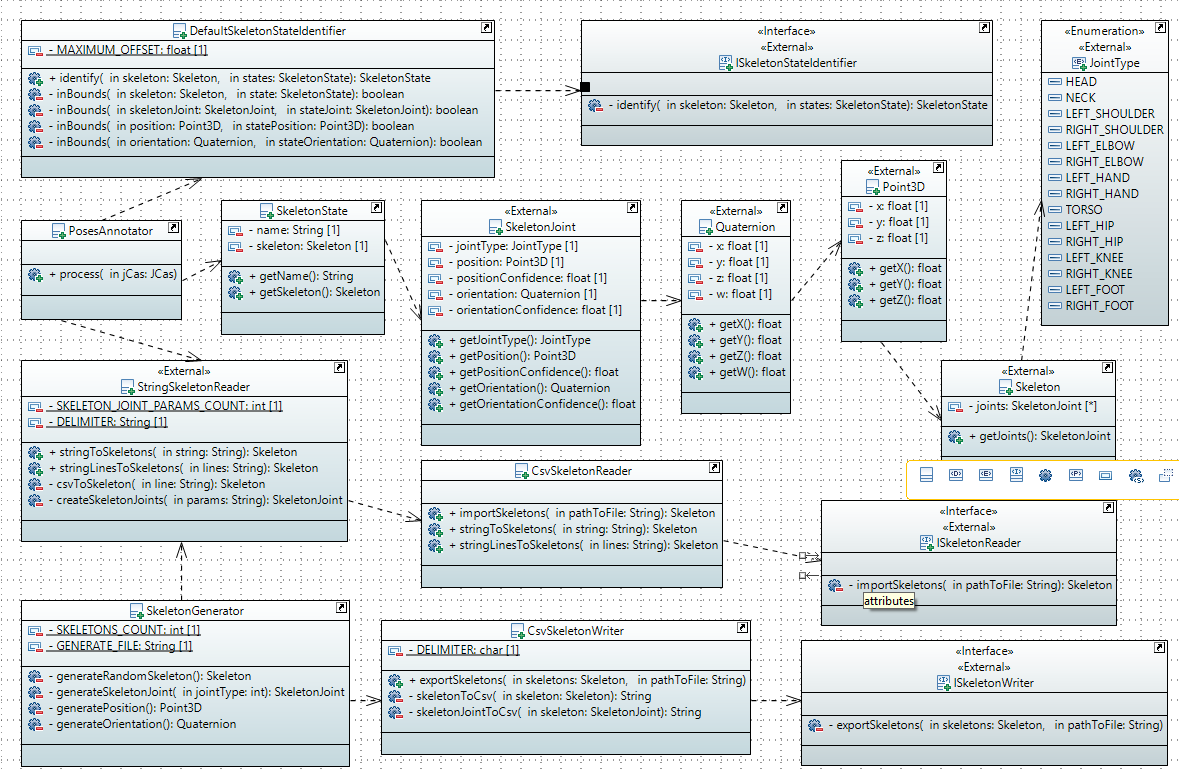


Рисунок 3 – Диаграмма классов

# 3 Технологическая часть

3.1 Запуск приложения

Исходный код доступен по ссылке:

<https://github.com/BolshakovUltraGod/PosesCapture>

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 4 показано соотношение классов проекта по их размеру. Видно, что самый большой размер имеет пакет model.

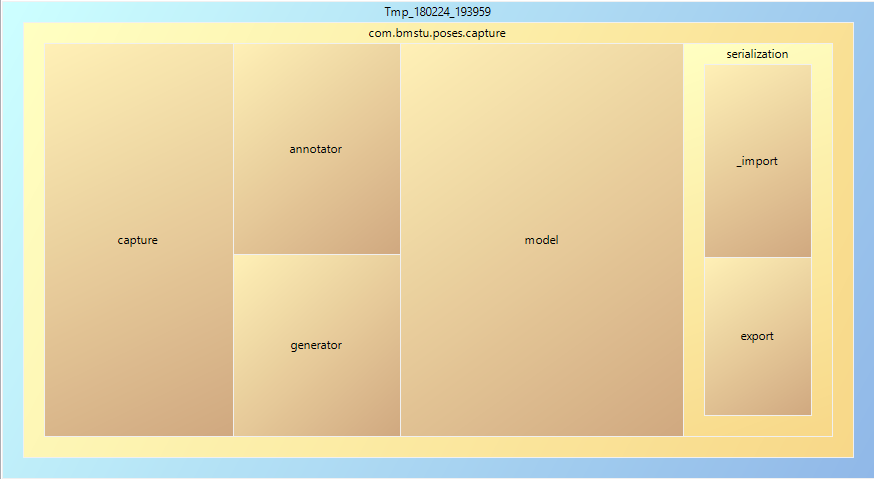


Рисунок 4 – Соотношение классов по размеру

На рисунке 5 отображен список всех метрик по разделам. Всего 4 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).

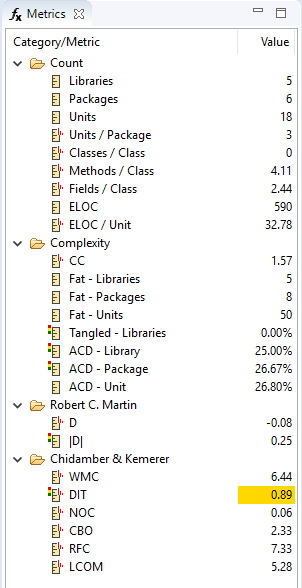


Рисунок 5 – Значения метрик

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).

3.3 Анализ зависимостей в коде системы

Рисунок 6 описывает зависимости между пакетами разработанного ПО:

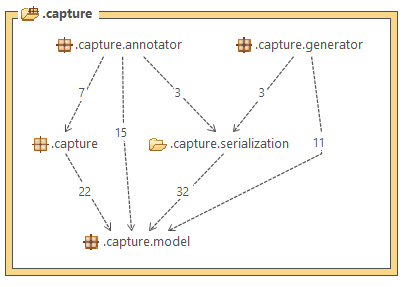


Рисунок 6 – Зависимости между пакетами

3.4 Тестирование ан корректность работы

Для теста будет использоваться один из инструментов, предоставляемых фреймворком UIMA SDK – Cav Visual Debugger (CVD).

Для теста использовались три позиции, как видно на рисунке 7: сидя, стоя и в прыжке.

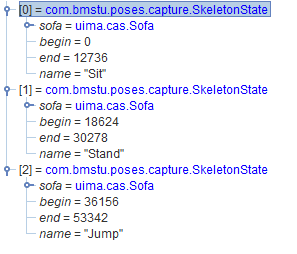


Рисунок 7 – позы для теста

Как видно на рисунке 8, из общего массива данных выделяются значения параметров скелета. Аннотатор работает корректно.



Рисунок 8 – позиция аннотируется

Заключение

В ходе курсовой работы была изучена технология Apache UIMA. Были выявлены преимущества и недостатки.

В результате выполнения проекта были решены задачи, поставленные в начале работы. Также в ходе курсовой работы был спроектирован и протестирован аннотатор с использованием технологии Apache Uima. Аннотатор позволяет распознавать в любом входном тексте данные о позиции человека или человекоподобного существа на видео.

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] Документация по технологии Apache UIMA 3.0.0-beta: [Электронный ресурс]. Режим доступа: <https://uima.apache.org/d/uimaj-3.0.0-beta/index.html>.

[2] Документация по UIMA Annotator: [Электронный ресурс]. Режим доступа: <https://uima.apache.org/doc-uima-annotator.html>