Санкт-Петербургский Государственный Университет
Математическое обеспечение и администрирование информационных систем

Системное программирование

Ломакин Александр Владимирович

Библиотека MIRF. Применение для анализа рассеянного склероза

Бакалаврская работа

Научный руководитель:
к.т.н., доцент Литвинов Ю.В.

Рецензент:
Технический директор ООО «Системы компьютерного моделирования» Петров А.Г.

Санкт-Петербург
2019

SAINT-PETERSBURG STATE UNIVERSITY
Software and Administration of Information Systems

Software Engineering

Lomakin Alexander

MIRF library and its application in multiple sclerosis analysis

Bachelor's Thesis

Scientific supervisor:
C.Sc., assistant prof Yurii Litvinov

Reviewer:
CTO at Computer Simulation Systems Petrov A.G.

Saint-Petersburg
2019

Оглавление

Введение

В ходе разработки приложения для анализа объема орбиты MISO [13] стало понятно, что для создания работающего программного продукта из нового метода обработки изображений требуется проделать большой объем сопутствующей работы, как например, преобразование форматов, подготовка изображений к анализу, визуализация результатов, генерация отчетов по результатам работы. Многие из этих задач являются общими для класса задач анализа медицинских изображений, и наличие системы, которая бы брала на себя реализацию общих компонент и предоставляла механизмы легкой интеграции пользовательского кода, могло бы ускорить внедрение концептов и методов в различные медицинские учреждения. Поэтому было принято решение разработать подобную систему.

Для того, чтобы показать возможности системы, было решено создать инструмент для генерации отчетов по рассеянному склерозу – хроническому заболеванию, при котором поражается оболочка нервных волокон головного и спинного мозга. Склероз считается достаточно распространенной болезнью, в среднем диагностируют 20-30 случаев на 100 тысяч населения. Основным способом слежения за развитием болезни является МРТ сканирование мозга раз в 6-12 месяцев. На МРТ склеры – участки поражения – выглядят как области светлее нормы. Новое сканирование сравнивается со старыми снимками и на основе сопоставления специалистом составляется отчет – объем новых участков поражения, изменение объема существующих. Так как очагов может быть несколько десятков, а снимков для сравнения несколько сотен, то составление такого отчета является весьма трудоемкой задачей, и хотя бы частичная автоматизация данного процесса может существенно ускорить работу медицинских специалистов.

1. Постановка задачи

Целью данной работы является разработка библиотеки для создания медицинских приложений и подготовка инструмента на базе библиотеки, которые позволят реализовать генерацию отчетов по рассеянному склерозу. Для достижения цели были сформулированы следующие задачи.

1. Произвести обзор предметной области.
2. Выбрать язык программирования на основе желаемых особенностей системы.
3. Определить основные паттерны разработки.
4. Разработать библиотеку.
5. Разработать инструменты, которые позволят реализовать генерацию отчетов по рассеянному склерозу.

2. Обзор существующих решений

Инструменты для анализа медицинских изображений можно поделить на несколько типов: специфические инструменты, специализированные для медицинской сферы, инструменты общего назначения.

Инструменты общего назначения К данной категории в основном относятся программные пакеты для анализа изображений, поддерживающие медицинские форматы. Некоторые из них представляют собой наборы инструментов, решающие типичные задачи, такие как обработка и анализ изображений (ITK) [9], визуализация (VTK) [21], обработка изображений и видео в реальном времени (OpenCV) [3]. Такие решения оперируют уровнем ниже предполагаемой библиотеки, так что не являются её аналогом, но могут быть использованы, например, при создании модулей системы.

Специфические инструменты К данной категории относятся приложения, которые фокусируются на решении задач, связанных с определенными органами или заболеваниями. Например, анализ изображений мозга (продукты компании icometrix [26], такие как icobrain-dm [24], icobrain-tbi [25]). Но узконаправленность данных решений, а также тот факт, что большинство из них имеют закрытый исходный код, исключают радикальное расширение функциональности сообществом, так что использование их как основы для предполагаемой системы труднореализуемо.

Инструменты для медицинской сферы К данной категории относятся системы, которые специализируются на работе с медицинскими снимками. Примеры: Ginkgo CAD [6] и ClearCanvas [4]. Также в данную категорию можно отнести расширяемые медицинские приложения, например Slicer [16], Weasis [22] и OsiriX [19]. Они предоставляют все необходимые базовые методы в интегрированном пользовательском интерфейсе. Такие приложения возможно расширять за счет специально

написанных плагинов под эти платформы. Тем не менее, такой способ не подходит для предполагаемой системы, так как ограничивает спектр применения только десктопными приложениями. Интересным продуктом для данной работы является MITK [14] – система с открытым исходным кодом, комбинирующая алгоритмы, разработанные в ITK [9], с алгоритмами визуализации из библиотеки VTK [21] и дополняющая их, тем самым позволяя создавать разнообразные медицинские системы. Недостатком данной системы является ее устаревание и слабая интеграция с новыми подходами к анализу изображений, в первую очередь средствами машинного обучения.

3. Medical Images Research Framework

Библиотека получила название **Medical Images Research Framework** [12] (в дальнейшем MIRF). Стоит отметить, что данный проект задумывался как проект кафедры, и в этом году, помимо реализованного в данной работе, в рамках другой дипломной работы велась работа над поддержкой мобильных платформ, а также интеграцией с tensorflow.

В качестве платформы для реализации рассматривались .NET Core и Java, так как требуется кроссплатформенное решение. .NET Core, несмотря на ряд преимуществ, пока не обладает надежными инструментами для реализации UI, а также такой же базой open-source решений в области работы с медицинскими форматами, поэтому предпочтение было отдано в пользу Java 8. Для обеспечения необходимой гибкости основным архитектурным стилем, заложенным в основу библиотеки, был выбран Pipes & Filters [17].

3.1. Kotlin

После нескольких месяцев разработки было решено перейти на язык программирования Kotlin. Это решение обусловлено несколькими причинами.

1. Язык полностью совместим с JVM, так что возможность использовать библиотеку в Java коде остается.

2. Особенности языка позволяют отказаться от большого количества boilerplate-кода, так что скорость разработки возрастает.

3. Почти нулевое время на переход из-за возможности автоматического перевода Java кода в Kotlin код.

4. Методы расширения позволяют организовать более гибкое распределение функциональности по модулям.

Таким образом, данный переход не ограничивает область применения, упрощая и ускоряя разработку. На данный момент весь ранее написанный на Java код переведен на Kotlin, вся разработка MIRF ведется на Kotlin, исключения составляют примеры использования, некоторые из которых написаны на Java, чтобы показать способы интеграции MIRF в Java код.

4. Архитектура и особенности MIRF

4.1. Core & Features

MIRF делится на 2 глобальных пакета – core и features.

– В core содержатся базовые типы библиотеки: абстрактные классы и интерфейсы, представляющие блоки, фильтры и обрабатываемые данные. Так как предполагаемый контекст использования MIRF – анализ медицинских изображений, в core также присутствуют базовые классы, представляющие медицинское изображение и серию изображений.

– В features содержится основная функциональность MIRF, которая необходима для облегчения разработки: механизмы доступа к хранилищам данных, инструменты для создания отчетов, адаптеры для различных медицинских форматов данных, различные фильтры изображений и инструменты визуализации сегментирования.

4.2. Данные

Большинство медицинских форматов снимков представляют из себя массив пикселей плюс некоторый набор метаданных, хранящих информацию о пациенте, дате исследования, физической интерпретации каждого пикселя. Данные в MIRF представляются в виде класса-наследника абстрактного класса Data. Основная задача класса – взять на себя управление метаданными, а именно списком атрибутов (который в MIRF хранится в классе AttributeCollection), а также обеспечить понятность предназначения сущности – по пайплайнам передаются только классы-наследники Data. MIRF в рамках пакета core предоставляет некоторые стандартные классы для данных, такие как:

– FileData – класс для передачи файлов по пайплайну;

– CollectionData – класс для передачи коллекций элементов;

– классы пакета medimage, такие как MedImage и ImageSeries, – используются для работы с медицинскими изображениями;

– ParametrizedData – класс для передачи любого типа, использующий средства обобщенного программирования в своей реализации.

Диаграмма классов пакета core.data представлена на Рис.1.

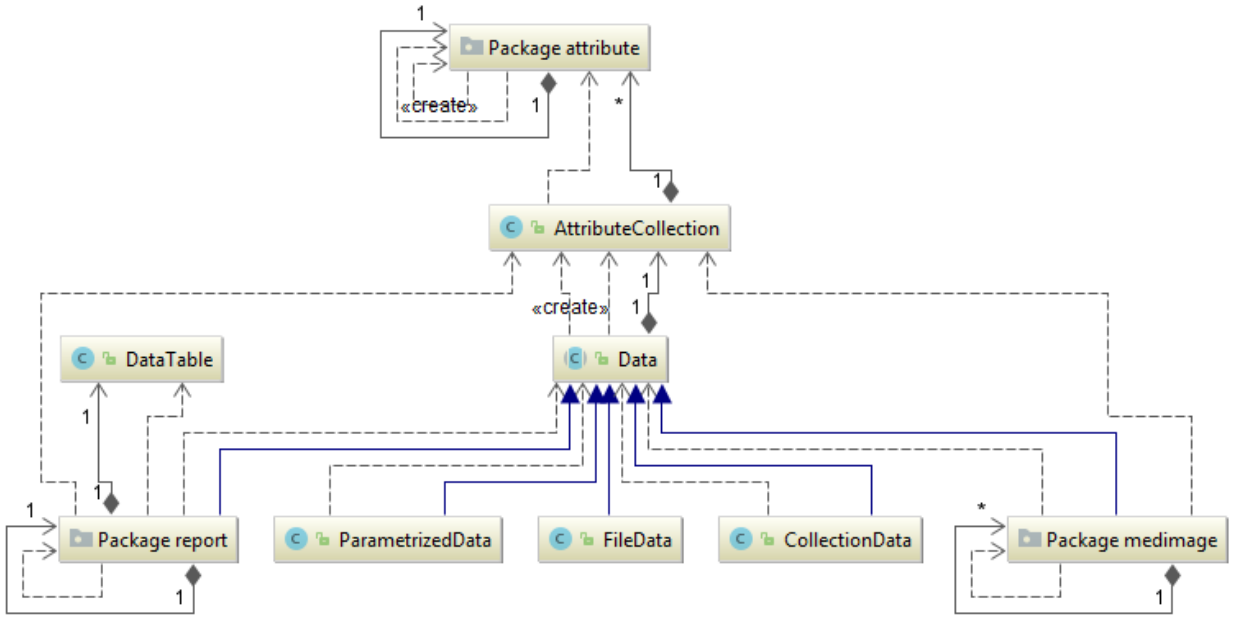

Рис. 1: Диаграмма классов core.data

4.3. Алгоритмы

Основными вычислительными элементами являются реализации интерфейса Algorithm<I, O>, содержащего один метод – O execute(I input);. Идейно Algorithm – это класс-обработчик, который не вызывает побочных действий. Предполагается, что в результате вызова execute измениться может только input, нет вызовов стороннего кода, не относящегося непосредственно к обработке данных и не происходит кеширование результата исполнения. Такое редуцирование ответственности вместе с тем, что алгоритм может быть легко создан из одного метода (с помощью класса SimpleAlg) дает возможности для гибкого

создания алгоритмов и организации иерархий. Типичный пример – статичные классы, содержащие методы-обработчики для данных одного типа, каждый из которых может быть преобразован в алгоритм вместо заведения класса под каждый метод.

4.4. Блоки

Как отмечалось ранее, в качестве основного сценария использования предусматривается создание конечным пользователем пайплайнов – серий обработчиков исходных данных. В качестве фильтров выступают экземпляры Algorithm, инкапсулированные в PipelineBlock – класс, предназначенный для соединения классов, реализующих интерфейс Algorithm, между собой. Сообщение между блоками построено на событийно-ориентированной модели, а сами PipelineBlock'и реализуют паттерн Observer. Некоторые блоки служат только для связи алгоритмов (как, например, класс AlgorithmHostBlock), другие же могут заниматься также агрегацией данных или иметь специфическое предназначение. Так, например, класс AccumulatorBlock может подписаться на несколько блоков и сигнализировать о результате только если результаты всех входных блоков готовы, а ConsumerBlock не позволяет подписаться на себя и служит в качестве терминатора пайплайна. Диаграмма классов для PipelineBlock представлена на Рис.2

4.5. Pipeline. PipelineKeeper

Части блоков может понадобиться информация о среде, в которой они сейчас запускаются. Также, некоторые блоки могут выполнять достаточно длительную работу, вследствие чего желательно иметь средства для отслеживания состояния блока.

Для этих целей вводится тип PipelineKeeper, представленный в виде интерфейса. Код представлен в листинге 1.

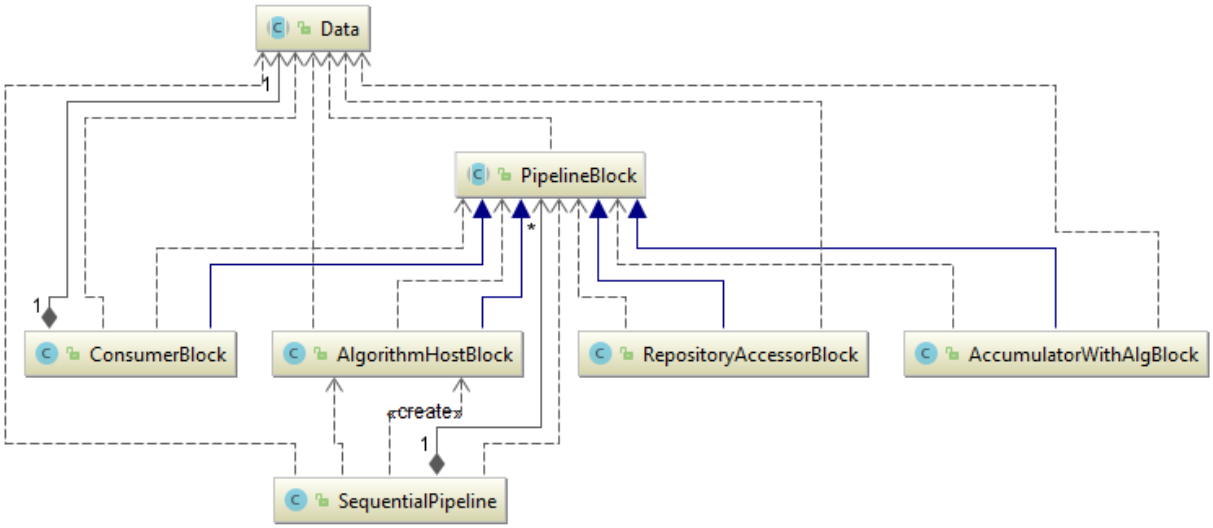

Рис. 2: Диаграмма классов PipelineBlock

```
interface PipelineKeeper {
    fun getCachedDataFor(sender, request): Any
    fun getRepositoryCommander(block): RepositoryCommander
    val session: IPipelineSession
}
```

Листинг 1: Интерфейс PipelineKeeper

Зачастую блокам требуется сохранить промежуточные данные на файловой системе. Типичные сценарии – отладка и ситуации, когда существует инструмент с закрытым кодом, взаимодействие с которым возможно только с помощью передачи ему в качестве аргументов пути к файлам. Поэтому PipelineKeeper предоставляет возможность блокам запросить репозиторий для хранения. Класс, реализующий PipelineKeeper, может возвращать не только локальный репозиторий, и в реализации пользователя репозитории могут быть любые (например, один блок хранит информацию локально, другой отправляет на удаленный сервер).

Таким образом, PipelineKeeper служит для нескольких вещей.

1. Управление репозиториями. Гибкая система позволяет назначать

каждому блоку свой репозиторий для хранения временных файлов, не опасаясь конфликта имен файлов, а единый менеджер позволяет централизованно управлять временем жизни репозиториев. Пример использования механизма выдачи репозитория показан на листинге 2.

2. Управление данными. Класс, реализующий PipelineKeeper, может кэшировать некоторые объекты (такие как подключения к удаленным ресурсам, предзгруженные данные). Таким образом, блок имеет возможность проверить наличие интересующего его ресурса у PipelineKeeper, а не, например, обращаться каждый раз к удаленному серверу, а также появляется возможность кэширования одного объекта для нескольких блоков одного пайплайна.

3. Отслеживание прогресса. IPipelineSession позволяет создавать новые записи в специальном логе пайплайна, указывая статус текущей активности, время создания, описание активности.

Реализацией по умолчанию интерфейса PipelineKeeper является класс Pipeline. Он предоставляет блокам стандартную реализацию IPipelineSession с возможностью подписаться на новые записи в логе. Таким образом, например, графический интерфейс может оперативно обновлять информацию о прогрессе пайплайна. Также, Pipeline при создании выбирает рабочую папку – ту, в которой он будет хранить временные файлы, а выдача репозитория блоку – это создание подпапки в рабочей папке пайплайна. Также Pipeline управляет временем жизни файлов на репозитории, позволяя очищать их по завершению жизни Pipeline, если такая опция задана при конфигурации.

```
// используется возможность получить
// временный репозиторий, чтобы сохранить
// content - данные от сервиса
val commander = pipelineKeeper.getRepositoryCommanderFor(this)
val resultLink = commander.saveFile(
                    content.uncompressGzipArray(),
                    ''result.nii'')
```

Листинг 2: Пример использования механизма выдачи репозитория

4.6. Форматы

Работа с разными форматами изображений построена на основе единого подхода – медицинский снимок представляется в виде изображения и набора метаданных. Такое представление позволяет унифицировать способ взаимодействия с разными форматами, абстрагировавшись от их внутреннего представления. Так, например, формат DICOM имеет собственные типы представления метаданных. Класс DicomReader, ответственный за считывание серий данном формате, преобразовывает DICOM типы в Java примитивы, которые сохраняются как значения некоторых MIRF-аналогов специфических для формата атрибутов, что значительно упрощает написание алгоритмов и создает дополнительный уровень абстракции, так как некоторые алгоритмы могут не знать, с каким именно форматом изображения они работают, важно лишь чтобы в метаданных были необходимые атрибуты.

Из медицинских форматов на данный момент поддерживаются DICOM и NIFTI, а также формат MHD, использующийся в библиотеке ITK. Для взаимодействия с DICOM используется библиотека pixelmed [18], интеграция NIFTI и MHD реализована с помощью расширений для ImageJ [10].

Форматы NIFTI и MHD содержат в себе информацию, относящуюся либо непосредственно к изображению, либо к его физической интерпретации (площадь в $см^2$, расстояние между снимками), так что

набор атрибутов весьма ограничен. Поэтому при загрузке этих форматов MIRF старается вычитать все атрибуты, для которых существуют mirf-аналоги. С другой стороны, формат DICOM является весьма сложным из-за обилия диалектов, наличия большого количества метаданных (имя пациента, название клиники, имя врача и т.д.), многие из которых могут понадобиться, но преобразовывать все из них нецелесообразно. Поэтому для работы с DICOM используется механизм отложенного преобразования атрибутов. Его блок-схема представлена на Рис. 3.

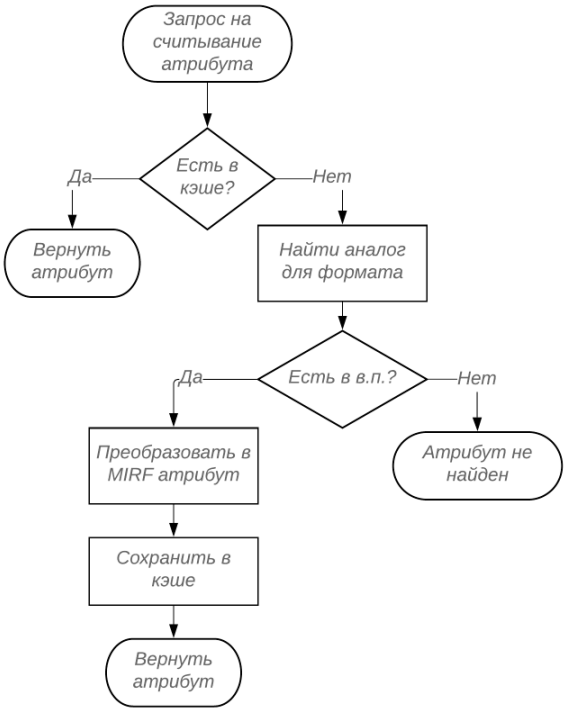

Рис. 3: Блок-схема механизма отложенного преобразования атрибутов. В.п. – внутреннее представление

Основная идея состоит в том, что при чтении файла его данные сохраняются в классе DicomImage (реализации интерфейса
MedImage для формата DICOM) в виде pixelmed (внутреннего) представления, а затем при запросе на чтение атрибута сперва проверяется кэш MIRF атрибутов, и если там нет запрашиваемого атрибута, то будет произведен поиск DICOM аналогов в pixelmed представлении. Если аналоги найдены, то произойдет генерация MIRF атрибута, который сохранится в кэш и вернется как результат считывания.

4.7. Инициализация пайплайна

Процесс инициализации пайплайна можно разбить на 2 части: создание пайплана и блоков и установление связей.

4.7.1. Особенности создания пайплайна и блоков

Как видно из кода, приведенного в листинге 3, существуют разные подходы к созданию блоков:

- простые действия, не требующие дополнительной логики, могут быть инкапсулированы в блок с помощью класса AlgorithmHostBlock;
- более сложные активности могут быть созданы с помощью наследования от класса PipelineBlock;

```
// 1 часть - вызов конструкторов пайплайна и блоков
// Изначально создается пайплайн
pipe = Pipeline(...)
// Простые действия можно инкапсулировать в готовый MIRF блок
val mySimpleBlock = AlgorithmHostBlock<Data, ImageSeries>(
        { mySimpleAction() }, ...)
// Возможно написание собственных блоков со сложной конфигурацией
val myCustomBlock = MyCustomBlock(...)
```

Листинг 3: Пример инициализации

4.7.2. Пример установлений связей между блоками

```
// Пример простой связи
simpleBlock.dataReady += anotherBlock::inputReady
// пример более сложных связей
myCustomBlock.setFirstSender(anotherBlock)
myCustomBlock.setSecondSender(simpleBlock)
```

Листинг 4: Пример установления связей между блоками

dataReady – это событие, которое вызывается блоком по готовности данных. Некоторым блокам для корректного получения входных данных достаточно только подписаться на событие, другим же требуется выполнить дополнительные действия, поэтому такие блоки могут иметь собственные методы задания входных данных, инкапсулирующие внутри подпись на dataReady. Таким образом, можно относительно несложно создавать пайплайны практически любой конфигурации, включая ветвление, обратные связи, ожидание данных от нескольких блоков и другие. Пример установления связей показан в листинге 4.

4.8. Генерация PDF отчетов

Алгоритм непосредственной генерации отчетов для медицинских снимков по проанализированным данным зачастую представляет собой последовательное отображение параграфов, содержащих текстовую, табличную информацию или некоторый набор изображений.

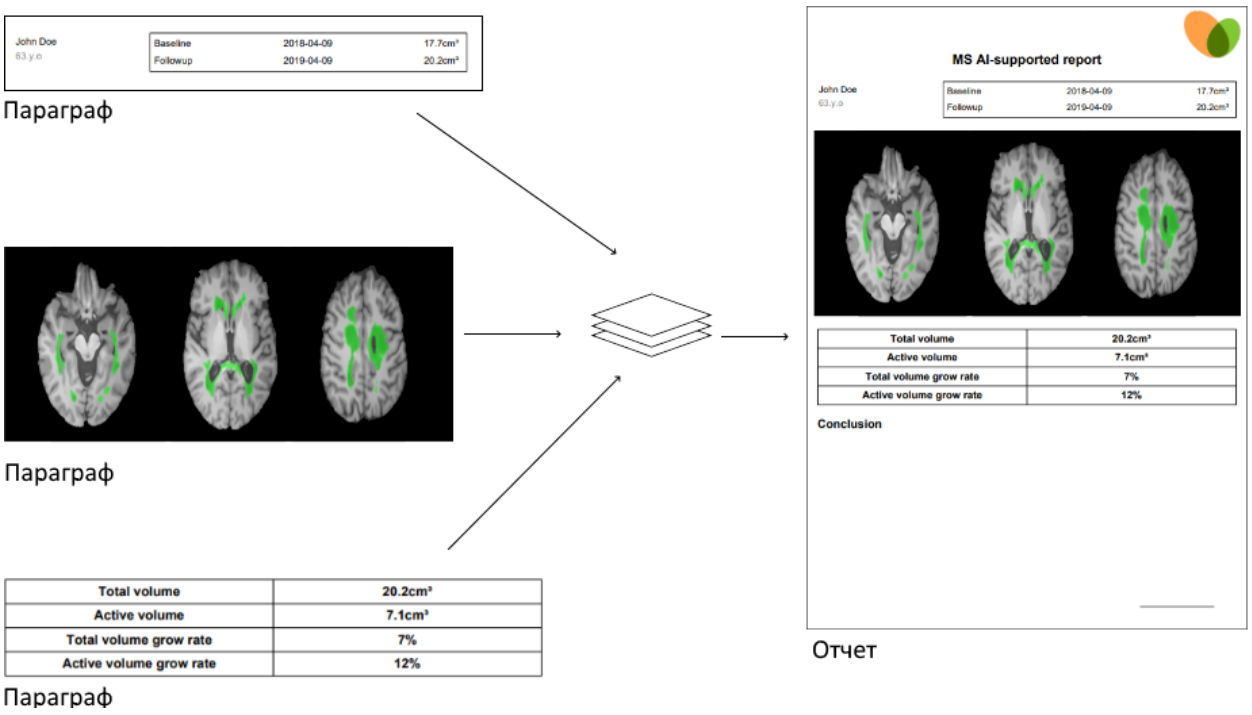

Рис. 4: Пример генерации отчета по рассеянному склерозу. Параграфы генерируется отдельно, после чего собираются и отображаются в PDF документе.

MIRF предоставляет инструменты, предназначенные упростить этот процесс. В качестве основной единицы передачи PDF данных внутри пайплайна служит PdfElementData – класс-наследник Data, инкапсулирующий параграф PDF документа. Стоит отметить, что параграфы могут содержать другие параграфы внутри себя, образуя вложенную структуру. Пакет feaures.pdf содержит методы расширений для различных типов MIRF, которые создают PdfElementData из указанного типа. Большинство из них имеет вид *TypeName*.asPdfElementData(...), где *TypeName* – имя типа:

```
fun ImageSeries.asPdfElementData(...): PdfElementData
```

Далее, как только необходимый набор параграфов сгенерирован, блок-аккумулятор (на данный момент MIRF предоставляет класс PdfElementsAccumulator для последовательного отображения списка параграфов) может создать PDF документ. Пользователь библиотеки может комбинировать готовые инструменты. Для создания более сложных компоновок есть возможность создать собственные блоки, которые

генерируют PdfElementData, и использовать существующие инструменты для их агрегации или, если же ничего из представленной функциональности не подходит, написать собственный блок-создатель готового документа. В качестве основной библиотеки для отображения результатов рассматривались Apache PDFBox [1] и IText [23]. Второй вариант лучше документирован, а также обладает бо̀льшим сообществом, так что выбор был сделан в пользу IText 7.1.2.

4.8.1. Изменение внешнего вида параграфов

Библиотека MIRF предоставляет возможность для кастомизации внешнего вида создаваемых параграфов (далее, декорирование). Кастомизация происходит за счет передачи в метод *.asPdfElementData(...) декорирующей лямбда-функции. Выбранная библиотека IText позволяет изменять внешний вид элемента после его создания, что облегчает декорирование. Каждый метод создания параграфа сначала создает «каркас» будущего параграфа, затем производит декорирование переданной лямбдой. Помимо косметических улучшений, таких как размер шрифта, цвет, возможны изменения и в генерируемом «каркасе»: так, например, в методе для генерации параграфа из таблицы можно указать, нужно ли отображать имена колонок и т.д. На Рис 5 показан пример декорированного параграфа.

До декорирования

name	value
Total volume	46.172 cm³
Active volume	12.0 cm³
Total volume grow rate	144.2875%
Active volume grow rate	120.0%

После декорирования

Total volume	**46.172 cm³**
Active volume	**12.0 cm³**
Total volume grow rate	**144.2875%**
Active volume grow rate	**120.0%**

Рис. 5: Пример применения лямбды-декоратора к табличному параграфу. Изменены шрифт, отображение имен колонок, ширина колонок, выравнивание текста по центру.

5. Применение

В данном разделе будут описаны некоторые способы применения MIRF, а также приведен код, показывающий основные возможности. Как писалось ранее, для демонстрации возможностей библиотеки было решено создать инструмент генерации отчетов по рассеянному склерозу. За основу взят открытый код nicMs [15] – академический проект по сегментации рассеянного склероза, в основе которого лежит применение каскадных нейронных сетей. Цель описываемого проекта, получившего название mirfMs, – показать, как легко с помощью MIRF превратить наработки в приложение, создавая из скрипта на языке Python инструмент с распределенным выполнением и автогенерацией отчетов. Полный код доступен на странице проекта MIRF на Github [12]. В данном разделе код будет приводиться с некоторыми упущениями для того, чтобы не перегружать описание. Графический интерфейс пользователя, а также пример получаемого отчета приведены на рис 6.

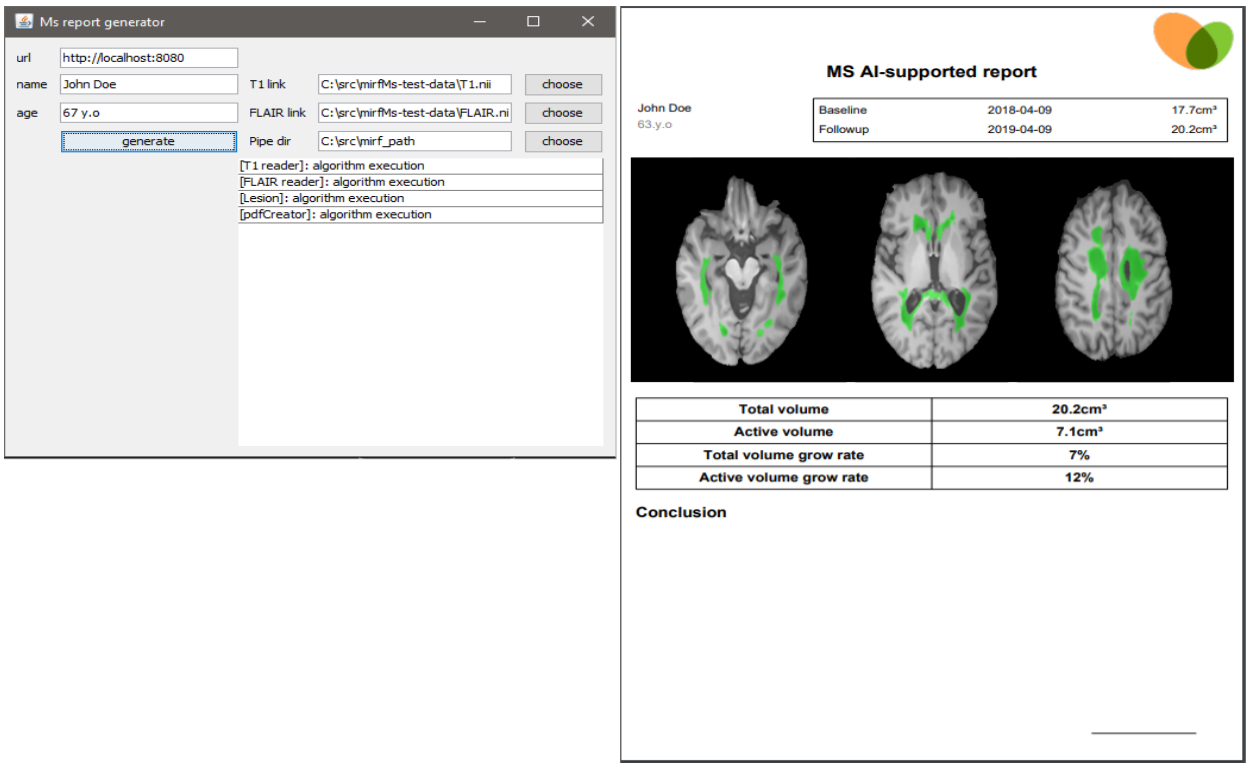

Рис. 6: Интерфейс программы-примера mirfMs и пример получаемого отчета

5.1. mirfMs

Перед тем, как начать рассмотрение, хотелось бы прояснить основные моменты, связанные с предметной областью, и показать общую структуру mirfMs. Для получения отчета необходима информация о:

– пациенте – имя и возраст;

– предыдущем исследовании – 4 вещественных числа (обозначающие некоторый объем);

– текущем исследовании – 2 серии изображений (называемые T1 и FLAIR);

На этапе конфигурации пайплайна задаются данные о пациенте и предыдущем исследовании, а также пути до серий T1 и FLAIR. Непосредственно пайплайн состоит из следующих этапов.

1. Считывание серий с режимами T1, FLAIR.

2. Предобработка данных. В данный этап входят следующие шаги:

   (a) вначале проводится так называемая регистрация серии FLAIR на T1. В данном контексте под регистрацией понимается приведение серии к единой координатной системе, а также устранение деформаций объекта отображения (деформация может быть вызвана движением человека в аппарате МРТ, дыханием и т.д.). «FLAIR на T1» означает, что серия под названием T1 берется как «фиксированная» (не меняющаяся в процессе регистрации);

   (b) удаление черепа со снимков.

3. Сегментация склер. Для данного этапа используется nicMs.

4. Генерация отчета. Отчет состоит из следующих параграфов:

   (a) заголовок, общая информация об отчете;

   (b) изображения некоторых срезов;

   (c) блок, описывающий объем;

   (d) место для заключения врача.

5. Сохранение отчета на файловой системе.

Этапы 2,3,4 выполняются на удаленном сервере (проект mirf-ms-service).

5.2. Удаленное выполнение

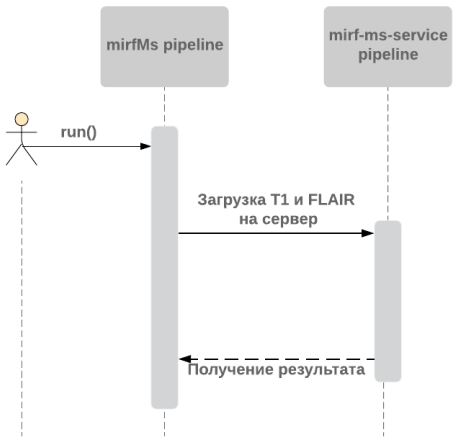

Рис. 7: Взаимодействие

Основные инструменты для считывания изображений в MIRF (NiftiReader, DicomReader, MhdReader) позволяют считывать из потока данных (массива байт), в том числе сжатого с помощью gzip. Эта особенность позволяет считывать данные, например, из сокетов или

содержимого http запросов, упрощая создание распределенных пайплайнов, где часть функциональности выполняется на удаленном сервере. В описываемом примере проект mirf-ms-service запускается на вычислительном сервере, а mirfMs с помощью специального блока посылает запрос на сервер, ожидает выполнения обработки и считывает полученный результат. Mirf-ms-service в свою очередь при получении запросов от клиентов запускает свою часть пайплайна, в которой выполняются основные вычисления, и отсылает результат вычислений обратно клиенту.

5.3. Пайплайн

MirfMs активно использует возможности MIRF – вся активность происходит в рамках пайплайна, инициализация которого занимает меньше 20 строк кода, однако позволяет добиться следующих преимуществ:

- особенности построения пайплайна обеспечивают по умолчанию параллельное считывание двух серий (T1 и FLAIR);
- отслеживания прогресса с помощью лога сессии, который отображается в графическом интерфейсе mirfMs;
- сохранение промежуточных результатов с помощью механизма репозиториев.

5.4. Генерация отчета

В mirfMs для генерации отчета используется 2 класса:

1. MsPdfReportParagraphsBuilder – класс с основным методом build(), отвечающий за создание из данных предыдущих блоков PDF элементов, отвечающих за визуализацию сегментации, а также таблицы с объемами. Этот класс хранит необходимые декора-

торы (см. раздел 4.8.1) и создает параграфы средствами MIRF. Код для генерации таблицы с объемами представлен в листинге 5.

2. MsPdfReportCreator – класс, создающий из результата работы класса MsPdfReportParagraphsBuilder PDF документ. Необходимость создания отдельного класса вызвана тем, что в итоговом отчете содержится нижний колонтитул (линия для подписи врача), которую трудно отобразить методами MIRF. Код создания документа приведен в листинге 6.

```
private fun getVolumeTable(table: DataTable): PdfTable {

    val decorator = { x: PdfTable ->
        x.setTextAlignment(TextAlignment.CENTER)
                .setFontSize(14f)
                .setFont(PdfFontFactory
                        .createFont(StandardFonts.TIMES_BOLD))
                .setWidth(UnitValue.createPercentValue(100f))
                .setMargins(0f, 5f, 0f, 5f)
    }

    return table.asPdfElement(decorator = decorator,
            displayHeaders = false)
}
```

Листинг 5: Создание параграфа с объемом. Как можно видеть, mirfMs определяет только косметические особенности, полностью отдавая генерацию каркаса MIRF.

```
fun createReport(): Document {
    // инициализация PDF документа
    val pdf = PdfDocument(...)
    val document = Document(pdf)

    document.add(createHeader())
    document.add(seriesVisualization)
    document.add(volumeTable)
    document.add(createConclusion())
    document.add(createFooter())

    document.close()
    return pdf
}
```

Листинг 6: Метод класса MsPdfReportParagraphsBuilder для генерации документа из параграфов. MirfMs занимается только специфическими вещами – генерация нижнего колонтитула (часть “Conclusion” и место для подписи).seriesVisualization и volumeTable – параграфы, созданные классом MsPdfReportParagraphsBuilder.

Как можно видеть из листингов 5 и 6, MIRF позволяет сократить количество кода, необходимое для генерации отчета, при этом оставляя возможность для кастомизации внешнего вида.

6. Особенности процесса разработки

В качестве инструмента для сборки исходного кода MIRF, mirfMs и mirf-ms-service используется Gradle [8] из-за его скорости работы и компактности скриптов. Git [7] используется как система контроля версий. Выбор в пользу git обусловлен скоростью создания коммитов и возможностью работать оффлайн. Код хранится на публичном репозитории на сайте github.com [12] так как этот сервис для хостинга является самым популярным в сфере проектов с открытым исходным кодом. Travis CI [20] и Appveyor [2] из-за их удобной интеграции с github.com используются для проверки собираемости продукта на Linux/OS X и Windows, а также прохождения тестов. В качестве библиотеки для юнит-тестирования используется JUnit 4 [11]. Код анализируется на качество инструментом Codacy [5].

7. Результаты

В ходе данной работы была разработана библиотека MIRF, которая содержит функциональность для создания пайплайнов, инструменты работы с различными медицинскими форматами, генерации отчетов в форме PDF файлов, а также создан пример использования, показывающий основные возможности. Для этого были проделаны следующие шаги.

1. Проведен обзор предметной области.
2. Выбран стек технологий, включая язык программирования и библиотеки для работы с форматами, генерации отчетов.
3. Настроен процесс разработки, включающий системы контроля версий и непрерывной интеграции.
4. Разработана функциональность MIRF.
5. Создан инструмент mirfMs, создающий отчеты по снимкам пациентов с рассеянным склерозом.
6. По результатам данной работы была написана статья «Medical Images Research Framework», принятая к публикации на CEUR-ws.org.
7. Результаты данной работы были представлены на конференции «СПИСОК-2019» в рамках инновационной секции.

Реализованная в рамках двух дипломных работ функциональность библиотеки может использоваться в качестве основы для создания медицинских инструментов и может облегчить реализацию следующих подзадач.

1. Чтение и запись файлов в формате DICOM и NIFTI.
2. Обработка медицинских изображений.
3. Создание отчетов в формате PDF.
4. Интеграция tensorflow модулей.

5. Распределение вычислений.

В дальнейшем планируется развитие MIRF в сторону увеличения доступных методов обработки изображений, а также добавление возможности интерактивного создания моделей.

Список литературы

[1] Apache PDFBox main page. — URL: https://pdfbox.apache.org/ (online; accessed: 06/05/2019).

[2] Appveyor main page. — URL: https://www.appveyor.com/ (online; accessed: 03/05/2019).

[3] Bradski G. The OpenCV Library // Dr. Dobb's Journal of Software Tools. — 2000.

[4] ClearCanvas. — URL: https://www.clearcanvas.ca/ (online; accessed: 20/12/2018).

[5] Codacy main page. — URL: https://www.codacy.com/ (online; accessed: 03/05/2019).

[6] Ginkgo CAD. — URL: https://github.com/gerddie/ginkgocadx (online; accessed: 20/12/2018).

[7] Git main page. — URL: https://git-scm.com/ (online; accessed: 06/05/2019).

[8] Gralde main page. — URL: https://gradle.org/ (online; accessed: 03/05/2019).

[9] ITK main page. — URL: http://www.itk.org (online; accessed: 20/12/2018).

[10] ImageJ main page. — URL: https://imagej.net/Welcome (online; accessed: 20/12/2018).

[11] JUnit-4 main page. — URL: https://junit.org/junit4/ (online; accessed: 03/05/2019).

[12] MIRF main page. — URL: https://github.com/MathAndMedLab/Medical-images-research-framework (online; accessed: 20/12/2018).

[13] Medical Images Segmentation Operations / Sabrina Musatian, Alexander Lomakin, Stanislav Sartasov et al. // Proceedings of the Institute for System Programming of the RAS. — 2018. — 01. — Vol. 30. — P. 183–194.

[14] The Medical Imaging Interaction Toolkit: challenges and advances / Marco Nolden, Sascha Zelzer, Alexander Seitel et al. // International journal of computer assisted radiology and surgery. — 2013. — 07. — Vol. 8, no. 4. — P. 607–620.

[15] One-shot domain adaptation in multiple sclerosis lesion segmentation using convolutional neural networks / Sergi Valverde, Mostafa Salem, Mariano Cabezas et al. // NeuroImage: Clinical. — 2018. — P. 101638. — URL: http://www.sciencedirect.com/science/article/pii/S2213158218303863 (online; accessed: 20/12/2018).

[16] Pieper S. Halle M. Kikinis. Development of an open source software module for enhanced visualization during MR-guided interstitial gynecologic brachytherapy // International Symposium on Biomedical Imaging. — 2004. — P. 632–635. — URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4004789/pdf/40064_2013_Article_912.pdf (online; accessed: 20/12/2018).

[17] Pipes & Filters pattern. — URL: http://lya.fciencias.unam.mx/jloa/patrones/PF.html (online; accessed: 20/12/2018).

[18] Pixelmed DICOM toolkit main page. — URL: http://www.pixelmed.com/dicomtoolkit.html (online; accessed: 20/12/2018).

[19] Rosset Antoine, Spadola Luca, Ratib Osman. OsiriX: An Open-Source Software for Navigating in Multidimensional DICOM Images // Journal of digital imaging : the official journal of the Society for Computer Applications in Radiology. — 2004. — 10. — Vol. 17. — P. 205–216.

[20] Travis-CI main page. — URL: https://travis-ci.org/ (online; accessed: 03/05/2019).

[21] VTK main page. — URL: http://www.vtk.org (online; accessed: 20/12/2018).

[22] Weasis main page. — URL: http://nroduit.github.io/en/ (online; accessed: 20/12/2018).

[23] iText pdf main page. — URL: https://itextpdf.com/ (online; accessed: 20/12/2018).

[24] icobrain-dm main page. — URL: http://icometrix.com/products/icobrain-dm (online; accessed: 06/05/2019).

[25] icobrain-tbi main page. — URL: http://icometrix.com/products/icobrain-tbi (online; accessed: 06/05/2019).

[26] icometrix main page. — URL: http://icometrix.com/ (online; accessed: 06/05/2019).