МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет **ФИЗИЧЕСКИЙ**

Кафедра **ФИЗИКО-ТЕХНИЧЕСКОЙ ИНФОРМАТИКИ**

Направление подготовки **03.04.02 ФИЗИКА**

Образовательная программа: **МАГИСТРАТУРА**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Майборода Вера Андреевна

(Фамилия, Имя, Отчество автора)

Тема работы Пакет программ для отбора событий в эксперименте на Супер С-Тау фабрике

«К защите допущена»
Заведующий кафедрой
Кроковный Павел Петрович
ученая степень, звание
к. ф.-м. н.
должность, место работы
с. н. с. ИЯФ СО РАН
……/……..
(фамилия И., О.) / (подпись, МП)

«……»……20…г.

Научный руководитель
Воробьев Виталий Сергеевич
ученая степень, звание
к. ф.-м. н.
должность, место работы
с. н. с. ИЯФ СО РАН
……/……..
(фамилия И., О.) / (подпись, МП)

«……»……20…г.

Дата защиты: «……»……20…г.

Новосибирск, 2021

Содержание

1. Введение

В настоящее время одним из главных методов экспериментальной проверки теорий физики элементарных частиц являются ускорительные эксперименты по столкновению частиц. Проводя статистический анализ данных, полученных с помощью коллайдеров, можно получать информацию о свойствах частиц и особенностях их взаимодействия.

Супер С-Тау фабрика — проект электрон-позитронного коллайдера ИЯФ СО РАН для изучения частиц, содержащих c-кварк, и τ-лептонов. Место взаимодействия пучков будет оснащено универсальным детектором частиц. Высокая светимость коллайдера, 10^{35} см$^{-2}$ с$^{-1}$, в области высоких энергий подразумевает большое количество столкновений частиц: согласно [1], за один экспериментальный сезон, длящийся 10^7 с, в столкновениях будет рождено около 10^9 τ-лептонов, столько же D-мезонов, и примерно $5 \cdot 10^{11}$ J/ψ-мезонов. Размер массивов собранных данных о произошедших распадах будет составлять около 30 ПБ за экспериментальный сезон. Для проведения экспериментов подобных масштабов требуется программное обеспечение (ПО), регулирующее работу считывающей электроники детектора, сбор и обработку данных.

Сложность анализа данных с детектора приводит к необходимости создания программного обеспечения, автоматизирующего стандартные процедуры первичного отбора событий и предоставляющего высокоуровневый интерфейс для решения подобных задач. Создание таких инструментов является целью представленной работы.

В Главе 2, основываясь на особенностях задач экспериментальной физики элементарных частиц и конкретно Супер С-Тау фабрики, сформулированы задачи, которые должно решать описываемое программное обеспечение, и формализованы предъявляемые к нему требования. Глава 3 рассказывает об уже существующих решениях, предложенных другими коллаборациями.

Результаты данной работы — описание решений поставленных задач и подробности реализации — представлены в Главе 4. Результаты проделанной работы обсуждены в Главе 5.

2. Описание предметной области и формализация задачи

Средства обработки данных, создание которых является целью данной работы, представляют собой инструменты, призванные упростить техническую сторону решения одной из задач экспериментальной физики — анализа данных о распадах частиц, зарегистрированных детектором. В данной главе будут рассмотрены особенности эксперимента SCT на Супер С-Тау фабрике, объяснены основные этапы анализа данных и сформулированы задачи и конкретные требования к разрабатываемым инструментам.

2.1. Эксперимент SCT

SCT (Super charm-tau) — универсальный детектор в месте встречи e^-e^+ пучков Супер С-Тау фабрики. Задача детектора — зарегистрировать частицы, рождающиеся при столкновении пучков. Чтобы восстановить картину произошедшего столкновения, требуется измерить характеристики родившихся частиц: их траектории, импульсы, энергии; и по этим данным идентифицировать частицы. Измерением перечисленных параметров занимаются различные подсистемы детектора (см. рисунок 1). Трековая система, состоящая из дрейфовой камеры и внутреннего трекера, необходима для измерения импульса частицы. Траектория заряженной частицы, движущейся в магнитном поле, искривляется, и величина этого искривления связана с импульсом частицы. Отследить траекторию частицы можно по оставляемому ею ионизационному следу — электронам, которые частицы выбивают из атомов по

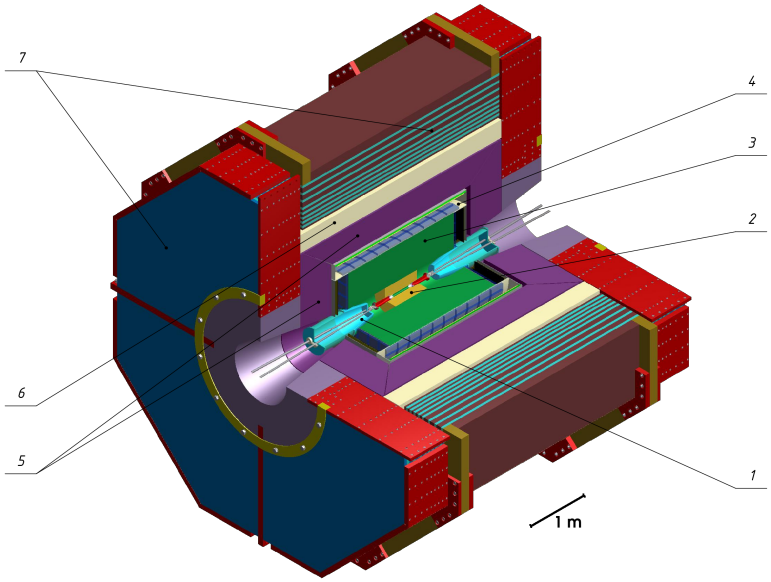

Рис. 1. Схема детектора SCT: 1 – вакуумная камера, 2 – внутренний трекер, 3 – дрейфовая камера, 4 – система идентификации, 5 – калориметр, 6 – сверхпроводящая катушка, 7 – мюонная система и ярмо магнита.

ходу своего движения. Такие свободные электроны собираются электроникой детектора, и по считанным сигналам восстанавливается траектория движения частицы и высчитывается её импульс. Мюонная система позволяет с высокой точностью отличить мюоны от других заряженных частиц, основываясь на особенностях взаимодействия мюонов с веществом. Калориметр необходим для измерения энергии частиц. Проходя через слой плотного вещества, частица сталкивается с его атомами и порождает ливень — поток вторичных частиц. Определенный процент энергии этого ливня выходит из вещества в виде света, который собирается фотоумножителями в виде электрических импульсов. Система идентификации частиц позволяет, с определенной погрешностью, сделать вывод типе зарегистрированной заряженной частицы, например, отличить каон от пиона.

По данным с электроники детектора: координатам в дрейфовой камере, номерам активированных кристаллов калориметра и другим сигналам регистрирующих систем происходит реконструкция частиц. Реконструкция ча-

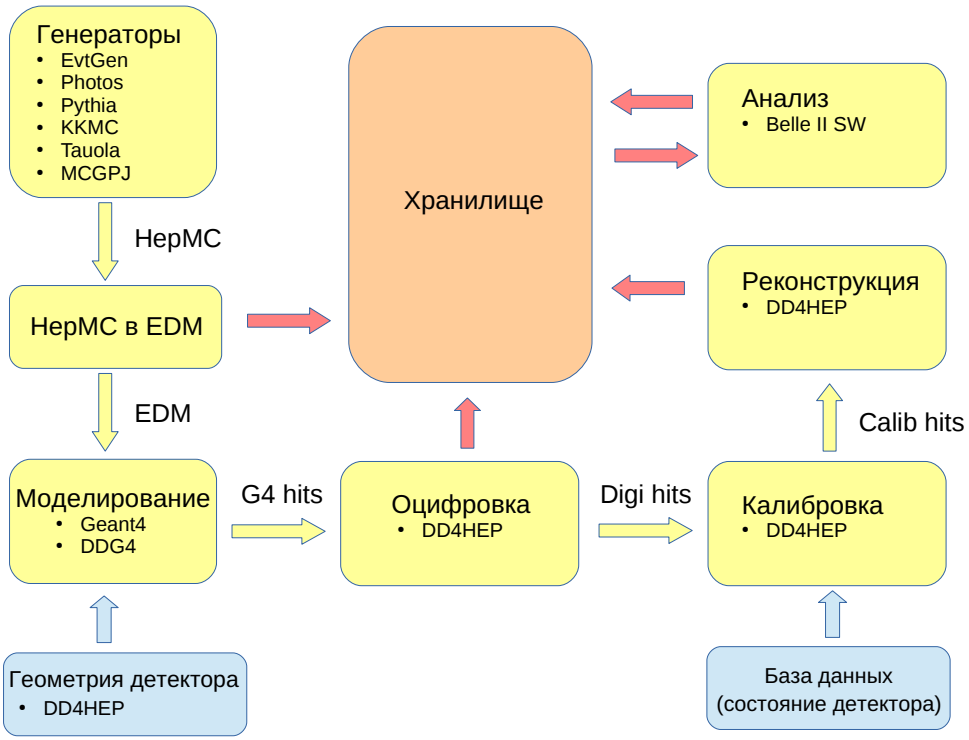

Рис. 2. Схема программного обеспечения детектора SCT.

стиц, родившихся при столкновении пучков, и конечных частиц произошедших распадов подразумевает определение типа частицы и основных её характеристик. Информация о реконструированных частицах записывается в общее хранилище данных и является отправной точкой дальнейшего анализа.

Важным техническим элементом постановки эксперимента и обработки его результатов является программное обеспечение. Программное обеспечение SCT включает в себя все стандартные компоненты, свойственные ускорительным экспериментам: фреймворк, модель данных события (`EDM` — event data model), генератор событий, модель детектора, алгоритмы анализа данных и так далее. Основные блоки ПО детектора и их связи представлены на рисунке 2.1.

Фреймворк ПО SCT носит название `Aurora` и основан на архитектуре `Gaudi` (подробнее про эту архитектуру в разделе 3.1.1). Модель данных SCT `EDM` реализована на языке `C++` в виде объектов `POD` (plain-old-data) при по-

мощи библиотеки PODIO [2]. Данные с детектора структурированы в заходы (англ. runs), представляющие собой наборы независимых событий (англ. events). В событиях записаны измерения конечных продуктов электрон-позитронного столкновения.

Помимо данных, полученных непосредственно с детектора, обрабатываются также данные моделирования: распады частиц генерируются на основе теоретической модели (Стандартной модели или какой-либо теории Новой физики). Моделирование распадов происходит с помощью набора различных генераторов событий: EvtGen [3], KKMC [4], Tauola [5], Pythia [6], MCGPJ [7] и пакета PHOTOS [8]. Данные генераторов имеют формат HepMC3 [9] и конвертируются в формат SCT EDM для записи результатов моделирования в хранилище данных. Описание геометрии детектора выполнено с помощью инструмента DD4Hep [10], для параметрического моделирования используется пакет SctParSim, а для моделирования взаимодействия частиц в веществом детектора — Geant4 [11]. Сгенерированные частицы проходят через модель детектора, предоставляя данные в том же формате, что и данные о реальных зарегистрированных детектором частицах.

Создание компонент ПО для этапа анализа накопленных данных является целью данной работы.

2.2. Процесс анализа данных

Конечная цель анализа данных — получение новых знаний о мире частиц и проверка теорий, основанные на статистической обработке записанных в хранилище данных о зарегистрированных детектором частицах.

Рассмотрим схематично сам процесс анализа. Из собранного массива данных выбираются зарегистрированные частицы, участвующие в распаде, являющемся объектом конкретного анализа, и удовлетворяющие определенным критериям. После отбора таких конечных частиц происходит реконструк-

ция цепочки распада, и реконструированные частицы также подвергаются отбору по заданным критериям. Кинематические параметры начальных и восстановленных частиц сохраняются в удобном для дальнейшего изучения виде `n-tuple`, представляющем собой кортеж чисел. Описанные этапы представляют собой первичный отбор событий. Отобранные данные подвергаются детальному исследованию: классификации событий, изучению распределений параметров и так далее. Данные моделирования подвергаются тем же процедурам отбора, что и данные эксперимента. Сравнивая результаты моделирования и результаты анализа реальных данных, делаются выводы о состоятельности модели, лежащей в основе моделирования.

В данной работе уделено внимание вопросу создания инструментов для первичного отбора событий. Существуют стандартные операции, используемые в данной задаче:

- Геометрический анализ события, определяющий форму произошедшего распада: струйный, сферический и так далее.
- Определение типа частицы (англ. flavour tagging). Например, в распаде $\psi(3770)$-мезона на два D-мезона, реконструировав распад одного из них, мы можем автоматически зафиксировать тип второго мезона.
- Сопоставление реконструированных частиц с генераторной информацией для Монте-Карло моделирования.
- Кинематический фит
- Отбор всех частиц, не вошедших в сигнал

2.3. Постановка задач работы

Основная функция инструмента для анализа данных — это автоматизация обработки и обеспечение высокоуровневого интерфейса к большинству

стандартных операций первичного отбора событий. Этот инструмент должен обеспечить возможность быстро приступить к анализу данных, избежать ошибок при выполнении стандартных действий, а также сделать процедуру отбора событий ясной и воспроизводимой. Учитывая поставленные цели и требования, можно сформулировать следующие задачи:

- разработка средств для описания свойств элементарных частиц;
- разработка средств для реконструкции распадов частиц;
- разработка средств для вычисления различных кинематических параметров;
- создание инструмента для отбора событий и сохранения необходимой информации в виде плоских `n-tuples`.

3. Обзор современного программного обеспечения для отбора событий

Разработка программного обеспечения для коллайдерного эксперимента занимает годы и требует вложения большого количества человеко-часов, вследствие чего различные коллаборации ведут разработку ПО на основе опыта своих коллег. Эксперимент SCT не исключение: многие его технические решения в области ПО реализованы по примеру экспериментов LHCb и Belle II, имеющих физические задачи схожие с задачами SCT. Эксперименты LHCb и Belle II выделяются также использованием самых современных подходов и техник при разработке своего программного обеспечения. Разработанные ими инструменты удовлетворяют основным принципам: обеспечивают воспроизводимость результатов, стандартизируют процедуры и имеют необходимую техническую документацию. В этой главе рассмотрены некоторые технические решения, реализованные в рамках ПО LHCb и Belle II, их

связь с экспериментом SCT и конкретно с модулем первичного отбора событий, рассматриваемом в данной работе.

3.1. Фреймворк `Gaudi`

Самым крупным на сегодняшний день центром изучения элементарных частиц является CERN (англ. European Organization for Nuclear Research) — Европейская организация ядерных исследований. LHCb (англ. Large Hadron Collider beauty) — один из экспериментов на LHC (англ. Large Hadron Collider), Большом адронном коллайдере, построенном в CERN'е. Эксперимент LHCb направлен на изучение асимметрии материи и антиматерии посредством изучения взаимодействий c и b-кварков.

Программное обеспечение эксперимента LHCb основано на разработанном участниками эксперимента фреймворке `Gaudi` [12]. Далее будут рассмотрены основные характеристики данного фреймворка и обозначена его связь с данной работой.

3.1.1. Техническая реализация фреймворка `Gaudi`

Фреймворк `Gaudi` определяет базовые компоненты, их функциональность и взаимодействие на всех этапах обработки событий: от триггеров до моделирования событий и физического анализа собранных данных. Такой подход позволяет поддерживать целостность структуры и поощряет переиспользование ранее написанного кода. Реализация данного фреймворка основана на объектно-ориентированном подходе, осуществленного с помощью языка `C++`.

Основные компоненты фреймворка `Gaudi` и их связи отображены на диаграмме объектов (см. рисунок 3). Среди основных компонент выделяются объекты `Algorithms`, `Services`, и хранилища данных. Связь компонент друг с другом реализована посредством интерфейсов, представляющих собой на-

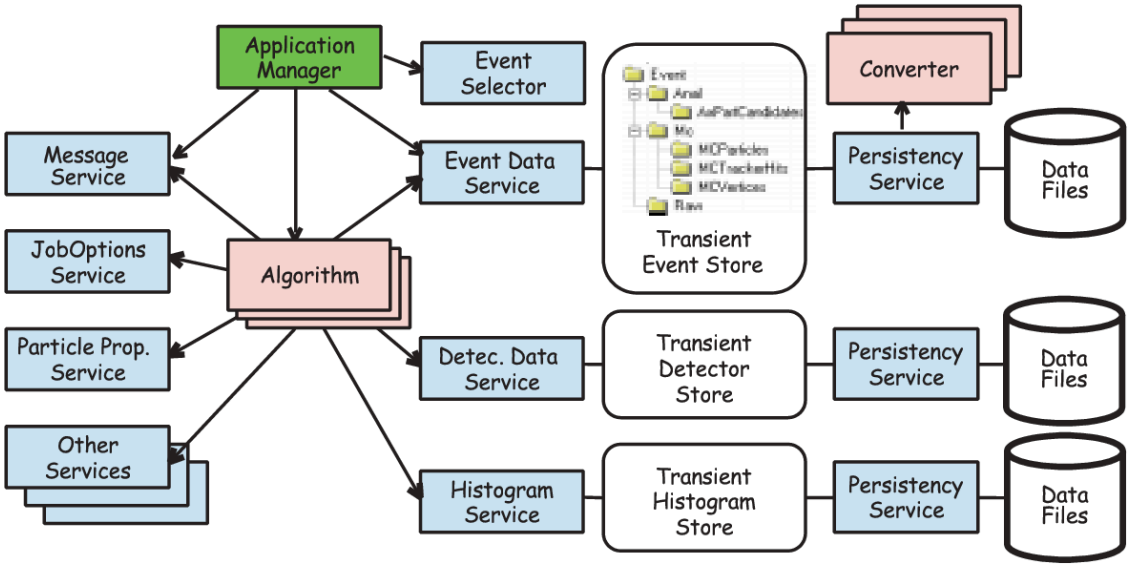

Рис. 3. Диаграмма объектов в `Gaudi` [13].

бор функций, каждая из которых отвечает за определенный тип взаимодействия.

Одна из особенностей фреймворка `Gaudi`– это возможность его адаптации под различные эксперименты и нужды пользователей, где под адаптацией подразумевается добавление функциональности к существующим компонентам без изменения их интерфейсов. Возможность добавления пользовательского кода предусмотрена для ограниченного числа компонент: объектов данных, `Algorithms` и `Converters`.

Объекты данных: Объекты данных описываются посредством классов, хранящих информацию о различных физических объектах: частицах, реальных и смоделированных, детекторе и его частях, и так далее. Выделяется три типа объектов данных:

- Данные о событии: данные непосредственно с детектора, реконструированные события и так далее.
- Данные о детекторе, его геометрия, структура, данные о материалах и так далее.

- Статистические данные, полученные в результате физического анализа данных о событиях. Хранение статистических данных в виде n-tuples реализовано с помощью библиотеки ROOT в формате TTree.

Методы классов объектов данных ограничены обработкой полей этих классов.

Хранение данных организовано посредством хранилищ: постоянных (англ. Persistent Data Store) и временных (англ. Transient Data Store). Постоянные данные записываются на диск и не имеют выделенных ограничений по времени хранения. К хранилищу таких данных доступ имеют только компоненты Services и Converters, реализующие запись или чтение данных. Период существования временных данных ограничен временем выполнения программы. Доступ к временному хранилищу помимо Services и Converters имеют также компоненты Algorithms. Временные данные могут быть записаны в постоянное хранилище посредством специального объекта Service, и, при необходимости, их формат может быть изменен при помощи выделенного объекта Converter.

Объекты Algorithms: Назначение компонент Algorithms — манипуляция данными, их обработка. Имея доступ к хранилищу временных данных, Algorithms производят поэтапную обработку, где каждое событие обрабатывается поочередно несколькими алгоритмами. Это взаимодействие схематично представлено на рисунке 4. Доступ Algorithms к данным и другим переменным окружения реализован через вызов компонент Services. Управлением вызовом методов объектов типа Algorithm и необходимых ему Services занимается компонента Application Manager, связанная с Algorithms через интерфейс IAlgorithm. Благодаря интерфейсу ISvcLocator, определенному для каждого объекта Algorithm, Application Manager может вызвать запрашиваемый объектом Algorithm объект Service.

В архитектуре Gaudi реализован механизм, позволяющий настроить объ-

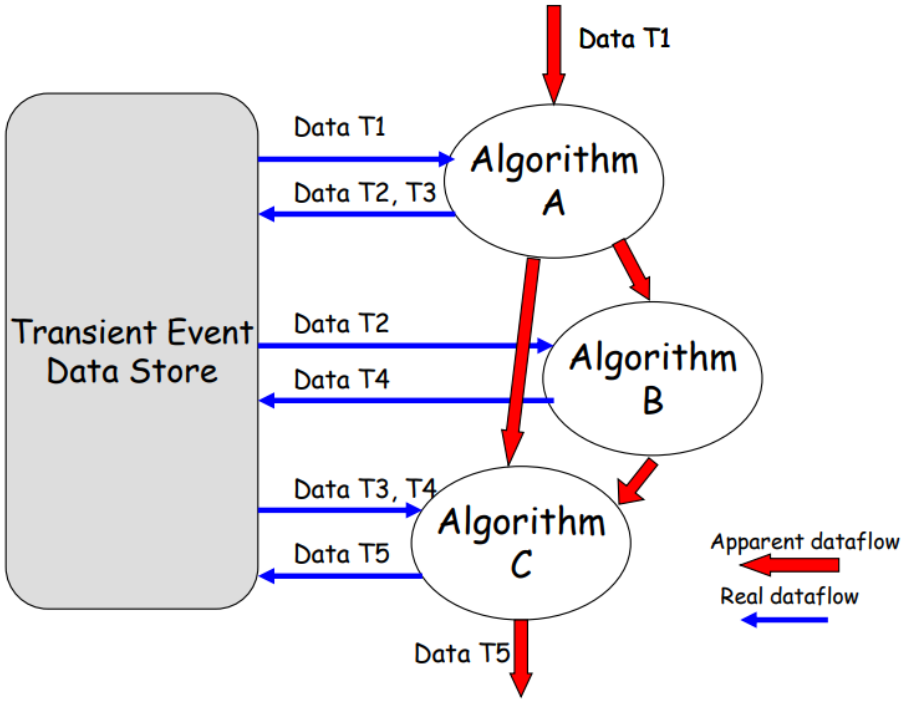

Рис. 4. Схема взаимодействия объектов `Algorithms` с временным хранилищем данных [12].

екты `Algorithms` во время выполнения, без необходимости перекомпиляции. Настройка заключается в инициализации полей классов `Algorithms` до вызова его методов. Например, через пользовательский интерфейс перед запуском алгоритма могут быть заданы определенные параметры его исполнения: количество итераций, критерии остановки и т.д. Такая возможность реализована с помощью объекта `JobOptionService`, который имеет связь со всеми `Algorithms` через интерфейсы `Property`. Объекты `Property` определяются непосредственно как поля классов `Algorithms`. Синтаксис объявления объектов `Property` показан в листинге 1. Объявление `Property` шаблонное, что позволяет с его помощью задавать значения переменным любого типа. Первый аргумент, `this`, — это указатель на рассматриваемый объект класса. Второй — имя объявляемого объекта `Property` в формате строки. Третий — число типа `int` (где тип соответствует `Gaudi::Property<int>`), значение определяемой переменной по умолчанию. Четвертый — описание объекта в формате строки.

```
1 Gaudi::Property<int> some_int{this, "Name of
    Property object", 0, "Description of some int"};
```

Листинг 1. Определение объекта Property как поля класса Algorithm.

Взаимодействие объектов Algorithm с временными хранилищами данных реализован через интерфейс DataHandle. Механизм использования объектов DataHandle аналогичен использованию Property: такие объекты должны быть объявлены как поля класса Algorithm. В примере, представленном в листинге 2, ObjectIn и ObjectOut — объекты для хранения входных и выходных данных, соответственно. Строки "objectToRead" и "objectToWrite" обозначают имена объектов во временных хранилищах.

```
1 DataHandle<ObjectIn> my_read_object{"objectToRead",
     Gaudi::DataHandle::Reader, this};
2 DataHandle<ObjectOut> my_write_object{"
    objectToWrite", Gaudi::DataHandle::Writer, this};
```

Листинг 2. Определение объекта DataHandle для чтения и записи в Transient store.

Добавление пользовательских компонент типа Algorithms является основным средством разработки инструментов для конкретных экспериментов с помощью фреймворка Gaudi. Пользовательские объекты Algorithms реализуются в виде классов-наследников от базового класса Algorithm, определенного в Gaudi. Базовый класс Algorithm отвечает за инициализацию внутренних указателей и управление информацией, полученной объектами-наследниками от других компонент фреймворка. Для определения класса-наследника Algorithm пользователю необходимо задать 4 метода: конструктор, и методы initialize(), execute(), finalize(), представленные в базовом классе в виде публичных абстрактных методов. Набор параметров конструктора зафиксирован: уникальное имя алгоритма в строковом формате, необходимое для идентификации определяемого алгоритма во фрейморвке, и

указатель на объект ISvcLocator. В конструкторах классов типа Algorithm происходит вызов определенной в Gaudi функции declareProperty для декларированных объектов DataHandle. По заданным именам объектов, которые DataHandle передает JobService, последний ищет соответствующее значение и присваивает эти значения полям класса типа DataHandle. Использование объектов Property обеспечивает доступность стандартных Services и инициализацию нужных переменных перед вызовом метода initialize(). Для поочередной обработки каждого события Application Manager вызывает метод execute(), в котором и определены основные операции. По завершению работы вызывается метод finalize().

3.1.2. Использование фреймворка Gaudi в программном обеспечении SCT

Фреймворк Gaudi был использован в качестве основы для фреймкорка ПО эксперимента SCT, Aurora. Выбор обоснован легкостью адаптации Gaudi и большим сообществом пользователей и разработчиков. В эксперименте LHCb изучаются частицы, рождающиеся при столкновении высокоэнергичных протонов и поток данных о зарегистрированных детектором частицах составляет 1 ТБ в секунду. Так как сохранение таких объемов данных не представляется возможным, с помощью множества триггеров данные фильтруются в режиме онлайн и в постоянное хранилище записываются только представляющие интерес события. Анализ данных LHCb ведется с учетом специфики эксперимента: после первичного анализа становится невозможен анализ формы и отбор по кинематическим ограничениям. Инструменты анализа данного эксперимента сильно отличаются от нужных для эксперимента SCT.

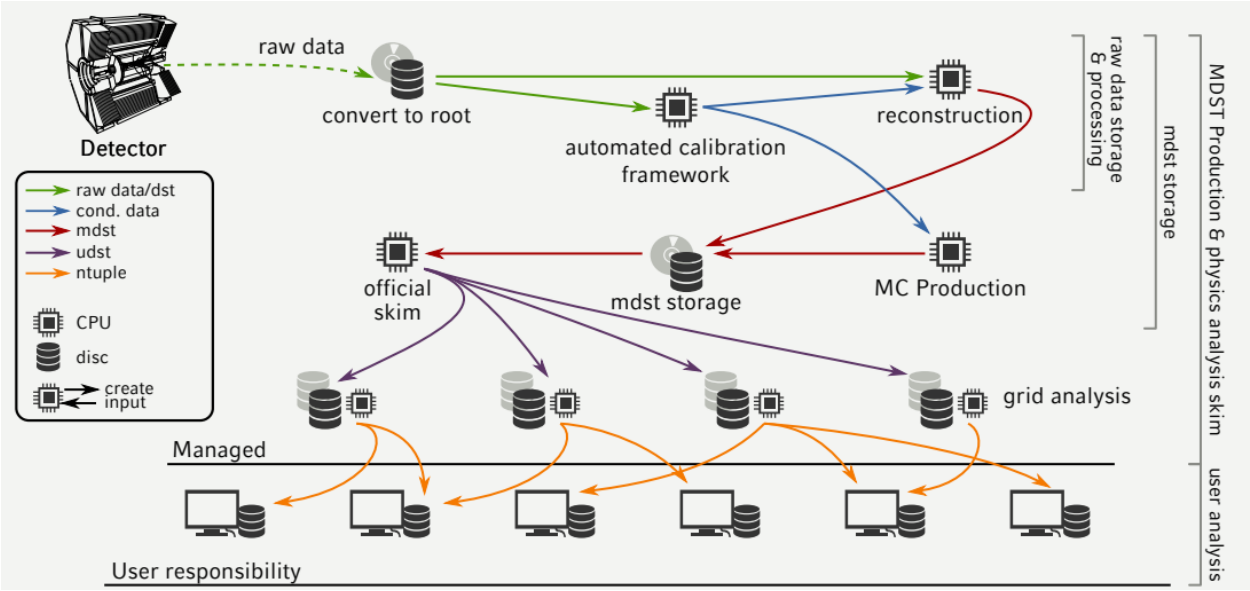

Рис. 5. Схема потоков данных в программном обеспечении эксперимента Belle II [15].

3.2. Фреймворк `basf2`

Эксперимент Belle II проводится на коллайдере SuperKEKB в Исследовательской организации ускорителей высоких энергий (KEK). Эксперимент является преемником эксперимента Belle, завершенного в 2010 году, и предназначен для поиска отклонений от Стандартной Модели путём изучения свойств B-мезонов. Программное обеспечение эксперимента базируется на фреймворке `basf2`, Belle AnalysiS Framework 2. В данном разделе будет представлен обзор на фреймворк `basf2` и, в частности, его модуль анализа.

3.2.1. Техническая реализация фреймворка `basf2`

В программном обеспечении эксперимента Belle II можно выделить три основных части: фреймворк `basf2`, часть `externals`, содержащая сторонний код, используемый `basf2`, и часть `tools`, содержащая скрипты для установки и настройки окружения [14]. Схема потоков данных от детектора до пользователей отображена на рисунке 5.

Фреймворк `basf2` содержит около 40 пакетов для различных этапов об-

работки данных и систему, отвечающую за коммуникацию пакетов и их интеграцию. Содержимое пакетов организовано согласно определенной структуре, имея разделы `modules` (далее модули), `tools`, `data`, `scripts` и так далее. Код, написанный на `C++`, компилируется в динамические библиотеки, устанавливаемые в раздел модулей, и доступен для динамической загрузки фреймворком. Обработка событий происходит путем последовательного исполнения фрейморком `basf2` набора динамически загружаемых модулей. Набор исполняемых модулей, их последовательность и конфигурация определяются через пользовательский интерфейс, реализованный на языке `python`. Каждый такой модуль является классом-наследником базового класса `Module`, в котором определены следующие методы: `initialize()` для инициализации модуля; `beginRun()`, `event()` и `endRun()` для последовательной обработки событий; `terminate()`, вызываемый по завершению обработки всех событий. Параметры модулей являются свойствами, чьи значения могут задаваться пользователем во время исполнения через пользовательский интерфейс. Модули обмениваются данными через хранилища данных, доступ к объектам которых осуществляется по их имени. Данные в хранилищах делятся на два типа по времени хранения: постоянные и временные, перезаписываемые после обработки каждого события. Формат данных определен с помощью библиотеки `ROOT`: каждый тип объекта в хранилище имеет свой словарь `ROOT`.

К описанному функционалу предоставляется пользовательский интерфейс, реализованный на языке `python`. Пользователь управляет обработкой событий через запуск скриптов, пример такого скрипта приведен в листинге 3. Несмотря на то, что модули, предоставляемые фреймворком `basf2`, написаны на `C++`, поддерживается возможность реализации модулей на языке `python` для упрощения процесса создания собственных модулей пользователем.

```
1 fillParticleList('K_S0:pipi', '0.4 < M < 0.6')
2 reconstructDecay('J/psi:mumu -> mu+:loose mu-:loose
    ', '3.0 < M < 3.2')
3 reconstructDecay('B0:jspiks -> J/psi:mumu K_S0:pipi
    ', '5.2 < M < 5.4')
4 vertexRave('B0:jspiks', 0.0, 'B0 -> [J/psi -> ^mu+
    ^mu-] K_S0')
5 matchMCTruth('B0:jspiks')
6 TagV('B0:jspiks', 'breco')
```

Листинг 3. Пример пользовательского скрипта фреймворка `basf2` на `python`, выполняющего реконструкцию распада $B^0 \to J/\psi(\to \mu^+\mu^-)K^0_S(\to \pi^+\pi^-)$.

3.2.2. Использование фреймворка `BASF2` в программном обеспечении SCT

Фреймворк `basf2` не уступает фреймворку `Gaudi` по функциональности, однако структура данной системы не рассчитана на адаптацию для других экспериментов. Специфика эксперимента Belle II предполагает изучение e^+e^- столкновений на относительно невысоких энергиях, что обеспечивает величину потока данных, возможную для полного сохранения в хранилищах для последующей обработки, как и в эксперименте SCT. Имея схожие задачи и условия, Belle II и SCT имеют также схожие нужды в инструментах анализа данных. Разработанные для эксперимента Belle II инструменты анализа позволяют удовлетворить всем требованиям, сформулированным в предыдущей главе. Однако прямое их использование не представляется возможным в виду глубокой интеграции данного модуля с фреймворком `basf2`. Для создания инструментов анализа для SCT было решено реализовать модуль анализа в программной среде `Aurora`, основываясь на модуле анализа Belle II.

4. Средства для отбора событий в эксперименте SCT

Задачи, описанные в главе 2, в данной работе предлагается решать путём адаптации алгоритмов анализа Belle II:

- С программной точки зрения: базируясь на опыте Belle II создать инструменты в рамках программной среды, разработанной для эксперимента SCT на C-Tau фабрике.

- С физической: создать среду, ориентированную под решение задач, свойственных SCT, упростить пользовательский интерфейс, обеспечить расчет необходимых кинематических параметров.

Разработка велась на языке `C++` в рамках программной среды `Aurora`. Общее пространство имён называется `sct`. Объекты и алгоритмы, непосредственно связанные с модулем анализа данных, лежат в одном из внутренних пространств имён `sct::ana`. Пользовательский интерфейс к разработанным средствам обеспечен на языке `python`.

Набор пакетов, составляющих модуль анализа данных, осуществляет последовательную обработку событий, записанных в хранилище в объектах типа `POD`, формируя объект `n-tuple`, хранящий кинематические параметры отобранных частиц. Три основных этапа преобразования данных схематично отображены на рисунке 6. Первый пакет реализует класс-наследник компоненты `Algorithm`, `EventLoader`, который производит чтение заданных пользователем частиц из хранилища, преобразование этих данных из формата `POD` во внутренний формат модуля анализа и запись полученных объектов во временное хранилище. Второй алгоритм, `ParticleCombinerAlg`, отвечает за реконструкцию распадов по набору их дочерних частиц. Третий алгоритм, `VarsToNtupleAlg`, производит расчет указанных пользователем кинематических параметров частиц и сохраняет результат в виде объектов `n-tuples`.

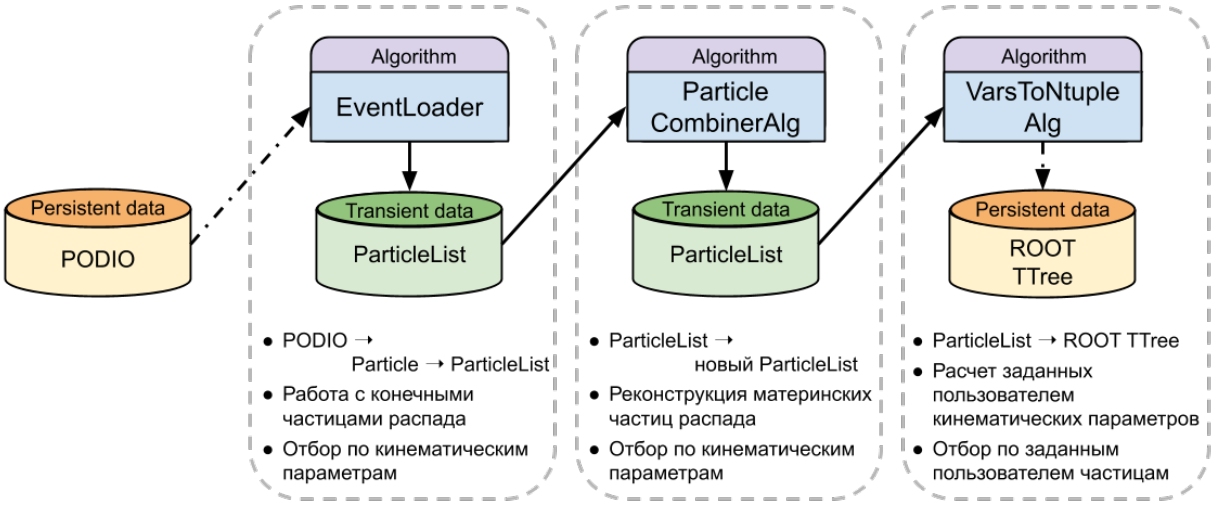

Рис. 6. Общая схема преобразований данных в модуле анализа данных эксперимента SCT.

Помимо трех представленных в смехе объектов `Algorithm`, модуль анализа данных включает в себя и другие пакеты.

В данной главе будет представлен пользовательский интерфейс к реализованному модулю анализа данных и его возможности (раздел 4.1), и описана реализация представленных инструментов на `C++` (раздел 4.2).

4.1. Обзор пользовательского интерфейса

Главной задачей блока анализа данных является обеспечение пользователя удобным инструментом для первичного отбора частиц. Следуя опыту экспериментов LHCb и Belle II, выбор языка реализации пользовательского интерфейса для модуля анализа данных эксперимента SCT был сделан в пользу `python`. Благодаря простому для понимания синтаксису `python` пользователь может быстро понять принцип использования предоставленных инструментов. Фреймворк `Gaudi` предоставляет возможность автоматически генерировать библиотеки на языке `python` для использования объектов `Algorithms`.

В секции 4.1.1 рассмотрены возможности строкового описания частиц и

распадов, которое составляет основу взаимодействия пользователя с представленными инструментами. Конкретный пример использования пользователем разработанных инструментов продемонстрирован в секции 4.1.2.

4.1.1. Правила использования строковых описаний

Строковое описание частиц, распадов и кинематических ограничений является оптимальным для пользователя форматом взаимодействия с инструментами анализа.

Строковое описание частиц: возможность указания типа частиц через объекты-строки основывается на сопоставлении каждого типа частицы с её именем, записанным в виде строки на каком-либо языке программирования. Например, заряженные пионы π^+ и π^- могут быть записаны в виде строк `"pi+"` и `"pi-"`, нейтральный D^0-мезон и его античастица $\bar{D}^0$ — в виде `"D0"` и `"anti-D0"`. Используемые правила записи названий частиц в строковом виде зафиксированы в генераторе `EvtGen` [3]. Помимо строкового обозначения, каждому виду частиц сопоставлен уникальный номер (или id) согласно схеме нумерации частиц Монте-Карло. Так, π^+ и π^- имеют коды 211 и -211 соответственно, а D^0 и его античастица $\bar{D}^0$ коды 421 и -421. Основная информация о каждой частице: имя, код, масса, заряд и так далее, хранится в текстовом файле `evt.pdl`. Кроме стандартных правил записи имени частицы, в предложенном инструменте обозначены дополнительные договорённости, расширяющие потенциал строкового описания частиц:

- Реализована возможность привязки метки к частице. Метки представляются в виде строк, добавленных к имени частицы через символ `":"`. Метки выполняют функцию пользовательского комментария: сообщения пользователем дополнительной информации о частице, обозначения отбора по какому-либо кинематическому параметру, или обозначения, ка-

кому каналу распада принадлежит рассматриваемая частица. Например, строка `"pi-:cut_E"`, где подстрока `"cut_E"` является меткой, несет информацию о том, что осуществлен отбор по энергии данной частицы. Строка `"pi-:D0"` обозначает, что рассматриваемый пион из распада D^0-мезона. Метки не могут содержать пробелов и других символов помимо латинских букв и символа `"_"`.

- Символ `"^"` перед именем частицы (например, `"^pi-"`) ставится пользователем для обозначения необходимости дальнейшего рассмотрения кинетических параметров этой частицы.

Строковое описание распадов: Распады представляются в виде строк, таких как

```
"D0 -> K- pi+",
```
(1)

где слева от стрелки указана материнская частица, а справа — её дочерние частицы, продукты распада. Данный пример соответствует распаду нейтрального D^0-мезона на заряженные K^- и π^+. Запись каскадных распадов, то есть распадов, в которых дочерние частицы не являются конечными, выглядит следующим образом:

```
"B+ -> [D0 -> K- pi+] K+".
```
(2)

Данная строка описывает распад B^+-мезона на D^0 и K^+, где, в свою очередь, D^0 распадается на K^- и π^+. Помимо непосредственной записи распада, разработанный инструмент имеет ряд дополнительных возможностей:

- Некоторые разновидности анализа подразумевают особое рассмотрение промежуточных частиц и фотонов.
  - В распаде (2) D^0-мезон является промежуточной частицей: он не входит в набор конечных частиц распада (здесь это K^+, K^- и π^+), и не

является верхней частицей в иерархии распада. В случае, если пользователь проводит анализ, в котором идёт рассмотрение лишь начальных и конечных частиц, но игнорируются промежуточные, он имеет возможность указать этот факт при написании строки распада.

– Часто фотоны, особенно ”мягкие”, то есть с низкой энергией, остаются незамеченными детектором. Вследствие этого реконструкция распадов, среди конечных частиц которых присутствуют фотоны, усложняется: цепочка распада не может быть полностью восстановлена из-за отсутствия информации о незарегистрированных фотонах. В некоторых анализах таких распадов оправдано проигнорировать факт наличия фотонов среди конечных частиц распада чтобы увеличить количество кандидатов среди реконструированных частиц. Разработанный инструмент предоставляет возможность обозначить это на этапе отбора событий.

Модуль анализа позволяет обозначить необходимость учёта фотонов и промежуточных частиц посредством использования различных типов стрелок при строковом описании распада. Список опций представлен в таблице 1. К примеру, строка `"B+ --> [D0 -> K- pi+] K+"` обозначает, что пользователя не интересует информация о D^0-мезоне.

Тип стрелки	Учёт фотонов	Учёт промежуточных частиц
`->`	`+`	`+`
`=>`	`-`	`+`
`-->`	`+`	`-`
`==>`	`-`	`-`

Таблица 1. Соответствие типов стрелок опциям учёта промежуточных частиц и фотонов в строковых описаниях распадов.

• Отбор распадов может быть произведен эксклюзивным или инклюзив-

ным путём. При эксклюзивном отборе происходит реконструкция всей цепочки распада и рассматриваются те события, в которых все конечные частицы зарегистрированы. Инклюзивный же отбор априори подразумевает, что какие-то конечные частицы остались неизмеренными (например, нейтрино). Для обозначения инклюзивности проводимого отбора в строку распада вместо имени нерегистрируемой частицы включается подстрока `"..."` при перечислении конечных частиц распада. Например, строка `"D0 -> K- pi+ ..."` обозначает, что помимо K^- и π^+ среди конечных частиц есть и другие, которые не были измерены.

Строковое описание параметров: Ограничения на кинематические параметры частицы обозначаются строками по типу `"0.1 < E < 0.5"`. Каждому параметру, определенному в модуле анализа данных, сопоставлено строковое имя, например, `"E"` для энергии частицы, или `"charge"` для её заряда. Полный список доступных переменных определен в файле `Variables.h` модуля анализа данных.

Числовые значения задаются в ГэВ для энергии, ГэВ$/c$ для импульса и ГэВ$/c^2$ для массы соответственно. Единицы измерения параметров задаются в отдельном файле, общем для модулей программной среды `Aurora`.

Строки с кинематическими ограничениями поддерживают обработку следующих операций:

- строгое неравенство: `"<"` и `">"`;
- нестрогое неравенство: `"<="` и `">="`;
- равенство и неравенство: `"=="` и `"!="`;
- конъюнкция и дизъюнкция: `"and"` и `"or"`.

Структура строки может быть:

- простой: `"E < 0.5"`;

- с двумя неравенствами: `"0.1 < E < 0.5"`;
- составной: `"E < 0.5 and pt > 0.1"`.

4.1.2. Пример использования интерфейса

Отбор событий происходит через запуск пользовательских скриптов на `python`. Структура скриптов основана на последовательном вызове алгоритмов, представленных на схеме 6, через пользовательский интерфейс.

В данном разделе будет рассмотрен пример эксклюзивного отбора распадов D^0 на K^- и π^+, и зарядово-сопряженного распада: $\bar{D}^0$ на K^+ и π^-, через пользовательский интерфейс с помощью реализованных в данной работе инструментов. К рассматриваемым распадам также будет применен отбор по кинетическим параметрам: импульс каонов не должен превышать 0.6 ГэВ$/c$, энергия D^0-мезонов должна быть больше 0.5 ГэВ. В качестве выходного файла будет создан объект `ROOT TTree`, содержащий информацию о кандидатах: их массе, энергии, импульсе и проекции импульса материнской частицы D^0, а также заряде, энергии и проекции импульсов для дочерних частиц, пионов и каонов. Полная версия кода скрипта может быть найдена в Приложении.

Обработка конечных частиц: Первый смысловой блок — чтение и отбор конечных частиц из хранилища и их представление в нужном формате, реализован через алгоритм фрейморвка `Gaudi EventLoader` (реализация на `C++` подробно рассмотрена в секции 4.2.3).

```
1 evlo = EventLoader('EvtLoader')
2 evlo.pcl.Path = 'Particles'
3 evlo.pListMap.Path = 'FSPLists1'
```

```
4 evlo.plists = [['pi+ cc'], ['K+:cut_p cc', 'p < 0.6
    ']]
```

Листинг 4. Использование алгоритма `EventLoader` через пользовательский интерфейс на `python`.

В листинге 4 представлен пример вызова алгоритма `EventLoader` с помощью интерфейса языке `python`. В первой строке показан вызов конструктора объекта класса `EventLoader` с заданным именем. Во второй указывается имя коллекции `PODIO`, необходимой для чтения: `Particles` для реконструированных частиц, и `MCParticles` для данных моделирования: частиц, сгенерированных методом Монте-Карло. В третьей строке обозначено имя списка частиц, под которым отобранные частицы частицы в будут записаны во временное хранилище. В последней строке задается список конечных частиц рассматриваемого распада и критерии их отбора. В данном примере помимо положительно заряженных частиц из хранилища также считываются и отрицательно заряженные, что указано в подстроке `"cc"`, добавленной к именам конечных частиц. Из прочитанных частиц создаются списки пионов и каонов. Параметры каонов, входящих в сформированный `ParticleList`, проверяются на соответствие кинематическому ограничению на импульс: его значение не должно превосходить 0.6 ГэВ$/c$. Помимо этого, список отобранных каонов помечается меткой `"cut_p"`, указывающим на их отбор по значению импульса.

Обработка распадов: Второй блок — реконструкция распадов на основе отобранных конечных частиц и для отбор материнских частиц по кинематическим критериям, реализован в виде алгоритма `ParticleCombiner`.

```
1 cmbr = ParticleCombiner('Cmbr',
2     decStr = 'D0 -> pi+ K-',
3     cutStr = 'E < 0.5 and p < 0.4',
```

```
4     selfConj = False
5     )
6 cmbr.pListMapI.Path = evlo.pListMap.Path
7 cmbr.pListMapO.Path = 'FSPLists2'
```

Листинг 5. Пример использования алгоритма `ParticleCombiner` через пользовательский интерфейс на `python`.

Пример вызова алгоритма `ParticleCombiner` представлен в листинге 5. В конструктор объекта класса `ParticleCombiner` принимаются имя объекта, строковое описание распада, который пользователь хочет реконструировать, ограничения на кинематические параметры реконструированной частицы, и отметка о том, является ли материнская частица самосопряженной. В строке 6 происходит указание имени, по которому алгоритм может найти во временном хранилище созданные `EventLoader` списки отобранных конечных частиц. Последняя строка отвечает за назначение имени выходным данным, состоящим из списков отобранных частиц распада, которые будут также записаны в временное хранилище.

Обработка кинематических параметров: Третий блок — расчёт заданных кинематических параметров и их сохранение в виде `n-tuple`, представлен в виде алгоритма `VarsToNtuple`.

```
1 tupl = VarsToNtupleAlg('D0Tuple')
2 tupl.listName = 'D0'
3 tupl.pListMapI.Path = cmbr.pListMapO.Path
4 tupl.fileName = 'd0_tuple/tup'
5 tupl.vars = [
6  ['px', 'py', 'pz', 'p', 'E', 'pt', 'M', ''],
```

```
7  ['charge', 'px', 'py', 'pz', 'E', 'D0 -> ^pi+ ^K-'
     ]]
```

Листинг 6. Пример использования алгоритма `VarsToNtuple` через пользовательский интерфейс на `python`

В листинге 6 показан пример вызова данного алгоритма через пользовательский интерфейс. В первой строке вызывается конструктор объекта класса `VarsToNtupleAlg` по заданному имени. Во второй указывается имя списка материнских частиц. В третьей — имя входного объекта в хранилище. Четвертая строка нужна для указания имени выходного файла. В строке 5 указываются наборы параметров и частиц: каждый набор состоит из списка кинематических параметров, записанных в строковом виде, и строки, обозначающей частицы, параметры которых нужно рассчитать. В данном примере в строке 6 обозначен набор параметров: значение импульса и его проекций, энергия и масса, которые должны быть рассчитаны для материнской частицы распада, D^0-мезона, на что указывает пустая строка в конце набора. Строка 7 обозначает, что проекции импульсов, энергия и заряд должны быть записаны для дочерних частиц рассматриваемого распада, выделенных символом `"^"`: каонов и пионов.

В выходном файле, записанном в формате `ROOT TTree`, для каждого события записан набор указанных пользователем кинематических параметров частиц. На рисунке 7 представлен пример события, параметры частиц которого записаны в формате `ROOT TTree`. Названия параметров дочерних частиц помимо имени параметра имеют подстроку с именем частицы и материнской частицы (для избежания конфликта имён в случае наличия одинаковых типов частиц из разных каналов распада).

```
======> EVENT:0
 px                = -0.0245595
 py                = -0.00658065
 pz                = 0.029596
 p                 = 0.398039
 E                 = 0.396122
 pt                = 0.0254258
 M                 = 0.394196
 D0_pi_charge      = 1
 D0_K_charge       = -1
 D0_pi_px          = 0.0426676
 D0_K_px           = -0.0672271
 D0_pi_py          = -0.083965
 D0_K_py           = 0.0773843
 D0_pi_pz          = 0.11414
 D0_K_pz           = -0.084544
 D0_pi_E           = 0.203417
 D0_K_E            = 0.192705
```

Рис. 7. Параметры частиц события, записанные в формате `ROOT TTree`.

4.2. Реализация инструментов на `C++`

Модуль анализа данных состоит из нескольких пакетов, содержащих определения и реализации различных классов. Главными блоками инструментов отбора событий, чей интерфейс был описан в разделе 4.1, являются объекты `Algorithms`, компоненты фреймворка `Gaudi`, на основе которого создана программная среда `Aurora`. Помимо них в модуле анализа данных заданы объекты данных и другие вспомогательные классы. В данном разделе будут представлены подробности реализации основных пакетов модуля анализа данных.

4.2.1. Пакет `AnaDataObjects`

Пакет `AnaDataObjects` определяет объекты данных, необходимые для работы модуля анализа данных. Данные хранятся посредством объектов, определенных через классы `Particle` и `ParticleList`.

Объекты `Particle` содержат информацию о реконструированных частицах. Трёх- и четырёхимпульсы частицы и матрица ошибок хранятся в объ-

ектах классов ThreeVector, FourVector и ErrMatrix соответственно. Перечисленные классы определены с помощью библиотеки Eigen3 [16] в пакете SctKine.

В хранилище данных каждое событие представляет собой фиксированный набор реконструированных по сигналам детектора частиц, чьи типы не определены. На основе одной реконструированной частицы может быть создано несколько различных объектов класса Particle, представляющих частицы разного типа. Поле m_mdstAssociated класса Particle, представляющее собой множество целочисленных значений, предназначено для хранения индекса реконструированной частицы, на основе которой был создан объект данного класса. В наборе m_mdstAssociated конечные частицы имеют один индекс, в то время как реконструированные частицы распадов содержат несколько индексов, запоминая индексы дочерних частиц. При восстановлении распадов важно сделать проверку, что все дочерние частицы, представленные в виде Particle, были созданы на основе разных реконструированных частиц. Такая проверка выполняется посредством сравнения полей m_mdstAssociated дочерних частиц: если множества индексов пересекаются, реконструкция материнской частицы невозможна. Проверка реализуется методом overlapsWith, принимающего как аргумент частицу Particle и возвращающего булеву переменную, равную 1 при наличии пересечения. В алгоритмах анализа Belle II подобный метод реализован другим путём: только конечные частицы распада хранят индексы реконструированных частиц и проверка на пересечение происходит путём обхода дерева распада, задавая алгоритму сложность $O(n^2)$. Благодаря тому факту, что сложность поиска элемента в неупорядоченном массиве является константной, реализованный в данной работе алгоритм имеет сложность $O(n)$. Таким образом реализованный алгоритм выигрывает в производительности по времени, но проигрывает в объеме занимаемой памяти, так как предполагает хранение индексов конечных частиц в полях их материнских частиц.

Конструкторы класса Particle можно разделить на два вида:

- Частицы могут создаваться на основе заданных кинематических параметров, кода PDG или набору дочерних частиц типа Particle (см. листинг 7). В таких случаях часть полей инициализируется заданными значениями, а остальным полям присваиваются значения по умолчанию: 0 для кинематических параметров, пустой вектор для списка дочерних частиц и так далее.

```
1 Particle();
2 Particle(int pdgCode);
3 Particle(const ThreeVector& momentum, ftype mass,
    int pdgCode, EParticleType particleType=
    c_Undefined);
4 Particle(FourVector momentum, int pdgCode,
    EParticleType particleType=c_Undefined);
5 Particle(const std::vector<ParticlePtr>& daughters,
     int pdgCode);
```

Листинг 7. Конструкторы класса Particle пакета AnaDataObjects, создающие объекты класса по известным параметрам.

- Частицы могут создаваться на основе объектов типа POD: sct::Particle и sct::MCParticle (см. листинг 8).

```
1 Particle(const sct::MCParticle &mcparticle);
2 Particle(const sct::Particle &particle, int pdg,
    size_t partIndex);
```

Листинг 8. Конструкторы класса Particle пакета AnaDataObjects, создающие объекты класса на основе объектов типа POD.

Заполнение полей происходит на основе информации, содержащейся в объектах типа POD: трёх- и четырёхимпульсам, массе (некоторые поля

sct::Particle представлены в листинге 9). При создании объекта класса sct::ana::Particle (или Particle для краткости) на основе объекта sct::Particle также происходит заполнение поля m_mdstAssociated.

```
1  sct::Particle:
2   Members:
3    - sct::BareParticle core
4    - float dedx
5   OneToOneRelations:
6    - sct::MCParticle mcPcl
7    - sct::GenVertex mcVtx
8   OneToManyRelations:
9    - sct::Track tracks
10   - sct::CaloCluster clusters
```

Листинг 9. Некоторые параметры класса sct::Particle, объекта типа POD, определенного в файле edm.yaml.

Помимо вышеперечисленных методов, класс Particle содержит методы, позволяющие доступаться до значений полей и изменять их, а также метод, возвращающий список конечных частиц.

Объекты ParticleList — контейнеры объектов Particle, хранят наборы частиц, объединенных по типу.

Помимо вектора самих частиц, в полях класса ParticleList хранятся имя частиц и код PDG, а также указатель на список античастиц в виде объекта ParticleList. Для избежания цикличности ссылок связь со списком античастиц реализована при помощи умного указателя, содержащего ”слабую” ссылку на объект, std::weak_ptr.

Конструктор ParticleList является приватным методом, создание объектов класса происходит через вызов методов create и createUnique, воз-

вращающих указатели на созданные объекты: `shared_ptr` или `unique_ptr` соответственно. Также реализованы методы для доступа к полям класса, проверки, входит ли заданная частица в контейнер, доступа к частицам по их индексам в векторе.

4.2.2. Пакет `DecayDescriptor`

Пакет `DecayDescriptor` реализует язык описания распадов — инструмент, позволяющий по заданной строке распада восстанавливать его иерархию, и по имени частицы определять её тип. Данный инструмент реализован через класс `DecayDescriptor` и вспомогательные классы `DecayString`, `DecayDescriptorParticle` и `Parser`.

Класс `DecayDescriptorParticle` предназначен для работы с частицами. Его функция — представление параметров частиц в удобном для дальнейшей обработки формате.

Класс хранит в своих полях код частицы согласно PDG, метку и информацию о том, была ли частица отмечена знаком `"^"`.

Конструктор класса в качестве аргументов принимает значения всех полей. Метод `nameSimple()` по коду PDG ищет в файле `evt.pdl` имя частицы, и возвращает его в упрощенном виде: без знака заряда, без приставки `"anti-"`, и с символом `"*"` замененным на строку `"ST"`. Также реализованы методы, возвращающие имя частицы и имя её античастицы, и значения полей класса.

Класс `DecayString` предназначен для описания распадов частиц. Объекты класса хранят материнские частицы в виде `DecayDescriptorParticle` и списки их дочерних частиц. Дерево распада через `DecayString` описывается рекурсивным методом: дочерние частицы представлены через объекты класса `DecayString`.

Поля класса `DecayString` перечислены в листинге 10. Дочерние частицы

хранятся в поле в формате внутреннего вспомогательного класса Daughters (строки 4-14). Поля Daughters включают в себя указанный тип стрелки распада (строка 9), флаг-пометка о том, является ли отбор инклюзивным (строка 13), и список дочерних частиц, описанных через объекты класса DecayString (строка 11).

```
1  class DecayString {
2    DecayDescriptorParticle m_mother;
3    class Daughters {
4      public:
5       Daughters(ArrowType type, std::vector<
          DecayString> da, bool incl) :
6       m_strArrow(type), m_daughters(std::move(da)),
          m_strInclusive(incl) {}
7       ArrowType m_strArrow;
8       std::vector<DecayString> m_daughters;
9       bool m_strInclusive;
10   };
11   std::unique_ptr<Daughters> m_daug;
12 };
```

Листинг 10. Поля класса DecayString, определенного на C++.

Конструкторы класса делятся на два вида: конструктор, принимающий на вход значения полей создаваемого объекта, и конструкторы от других объектов типа DecayString. Остальные методы обеспечивают доступ к значениям полей класса.

Класс Parser включает в себя методы создания вспомогательных классов DecayDescriptorParticle и DecayString по заданной строке. Он содержит два метода, которые принимают строку в качестве аргумента: ме-

тод `parse_particle`, создающий объект `DecayDescriptorParticle`, и метод `parse_decay`, создающий объект `DecayString`.

Метод `parse_particle` выделяет из данной строки информацию, необходимую для инициализации полей объекта класса `DecayDescriptorParticle`: переменную булевого типа, обозначающую, отмечена ли частица знаком `"^"`, номер частицы согласно PDG, и её метку. Благодаря правилам строковой записи частиц, определенных в разделе 4.1.1, необходимая информация извлекается из строки проверкой условий:

- Если частица отмечена символом `"^"`, то этот символ всегда является первым символом строки.
- Если в строке присутствует символ `":"`, то подстрока справа от этого символа является меткой частицы, в противном случае метка считается равной пустой строке.
- Оставшаяся нерассмотренной подстрока является именем частицы и с помощью файла `evt.pdl` этому имени сопоставляется код PDG.

Метод `parse_decay` конструирует объект класса `DecayString`, рекурсивно восстанавливая дерево распада, в котором листья являются объектами класса `DecayDescriptorParticle`, а промежуточные узлы — объектами класса `DecayString`. Основываясь на правилах строковой записи распадов, строка поэтапно анализируется. По наличию подстроки, обозначающей стрелку распада, делается вывод о том, какой объект описывает данная строка: частицу или распад. Для частиц вызывается метод `parse_particle`. Строка, описывающая распад, обрабатывается следующим образом:

- Создается объект класса `DecayString`;
- Внутренний метод определяет тип указанной в распаде стрелки и инициализирует соответствующее поле объекта класса;

- Объект `DecayDescriptorParticle` заданный от подстроки, стоящей слева от стрелки, назначается материнской частицей;
- Подстрока, стоящая справа от стрелки, рассматривается поэтапно:
  - Если в подстроке объявлен распад дочерней частицы, заключенный в квадратные скобки, то такая подстрока подается как аргумент методу `parse_decay`, а материнская частица данного подраспада включается в список дочерних частиц рассматриваемого объекта класса `DecayString`.
  - Остальные подстроки, разделенные пробелами и обозначающие частицы, служат аргументами функции `parse_particle`, и включаются в список дочерних частиц.

Класс `DecayDescriptor` является основным классом одноименного пакета, и предназначен для хранения информации о дереве распада или его частях. Концептуальное его отличие от класса `DecayString` заключается в том, что класс `DecayDescriptor` связывает абстрактное дерево распада с конкретными частицами `Particle`.

Поля этого класса определены в листинге 11 и включают в себя материнскую частицу в формате `DecayDescriptorParticle` (строка 2), вектор прямых дочерних частиц в формате `DecayString` (строка 3) и информацию об отборе (строки 5-8).

```
1 class DecayDescriptor {
2   DecayDescriptorParticlePtr m_mother;
3   std::vector<DecayDescriptor> m_daughters;
4   std::optional<size_t> m_iDaughter_p;
5   bool m_isIgnorePhotons;
6   bool m_isIgnoreIntermediate;
7   bool m_isInclusive;
```

```
8     bool m_isNULL;
9 };
```

Листинг 11. Поля класса DecayDescriptor, определенного на C++.

Конструкторы объектов класса DecayDescriptor принимают как аргумент либо строку распада, либо объект DecayString. В первом случае по заданной строке распада конструируется объект DecayString при помощи метода Parser::parse_decay, и вызывается конструктор второго вида.

Методы класса позволяют осуществлять проверку, подходит ли частица Particle под описание данного DecayDescriptor. Проверка осуществляется путем сравнения количества дочерних частиц и их кодов PDG.

Класс DecayDescriptor имеет метод selectionNames(), позволяющий генерировать имена выбранных частиц распада в форме, удобной для использования при записи параметров этих частиц в n-tuple. Например, в распаде, описанном строкой (2), если каоны были бы отмечены знаком "^", рассматриваемый метод вернул бы их имена в виде строк "K" для каонов из распада B-мезона и "D0_K" для каонов из распада D^0-мезона. Данный метод осуществляет спуск по дереву распада, запоминая имена всех частиц, возвращаемые методом DecayDescriptorParticle::nameSimple(), и добавляя к ним подстроку с именем материнской частицы (за исключением случая, когда материнской частицей является верхняя в иерархии распада частица). Частицы, для которых были сгенерированы одинаковые имена, нумеруются: например, "pi1" и "pi2".

Остальные методы класса реализуют доступ к значениям полей.

4.2.3. Пакет EventLoader

Пакет EventLoader состоит из одноименного класса EventLoader, определенного как один из алгоритмов архитектуры Gaudi. EventLoader служит для считывания данных, записанных в хранилище в объектах типа POD, и для

формирования списков частиц ParticleList из прочитанных данных, где каждая частица представлена в виде объектов класса Particle, определенного в разделе 4.2.1.

Поля класса EventLoader включают в себя локальную структуру PList, хранящую информацию о свойствах считываемых частиц: номер кода согласно PDG, название частицы, булеву переменную, отражающую факт наличия античастиц и строку с их названием, а также кинематические ограничения в виде объекта Cut (подробнее о нем в разделе 4.2.5); а также вектор из таких объектов. Среди полей класса также декларирован словарь m_pListMap, сопоставляющий объекты класса ParticleList с именами частиц, хранящихся в них. В листинге 12 приведены некоторые поля класса EventLoader: объект Property, необходимый для получения заданного пользователем через интерфейс python списка частиц для отбора (строка 1); и два объекта DataHandle: для считывания и записи данных (строки 2-3). Подробная информация о таких объектах была приведена в разделе 3.1.

```
1 Gaudi::Property<std::vector<std::vector<std::string
    >>> m_PListStr{this, "plists", {}, "Particle
    lists to be constructed"};
2 DataHandle<sct::ParticleCollection> m_pcl{"pcl",
    Gaudi::DataHandle::Reader, this};
3 DataHandle<PListMap> m_pListMap{"pListMap", Gaudi::
    DataHandle::Writer, this};
```

Листинг 12. Некоторые поля класса EventLoader, определенного на C++.

Являясь классом-наследником компоненты Algorithm, класс EventLoader имеет 4 метода: конструктор, initialize(), execute() и finalize().

Конструктор класса EventLoader вызывает определенную во фреймворке Gaudi функцию declareProperty для декларированных объектов типа DataHandle. Через пользовательский интерфейс задается имя объекта для

считывания из временного хранилища, список частиц для считывания и ограничения на их кинематические параметры, а также имя списка частиц, сформированного на выходе. По именам объектов данных, которые `DataHandle` передает `JobService`, последний ищет соответствующие данные и считывает их в поля класса `EventLoader`, объекты `DataHandle`.

Метод `initialize()` к моменту своего выполнения имеет поля `m_pcl` и `m_pListMap` уже связанные с входными и выходными объектами из временного хранилища, и поле `m_PListStr`, инициализированное парами строк, содержащими имена частиц и кинематические условия их отбора в строковом виде. Строка с именем частицы, указываемая через пользовательский интерфейс, может содержать подстроку `"cc"`, например, `"pi+ cc"`. Аббревиатура *cc* (от англ. charge conjugated — зарядово-сопряженные) обозначает необходимость рассмотрения также и античастиц. В приведенном примере добавление `"cc"` означает, что пользователя интересуют как π^+, так и π^-. Метод `initialize()` на основе заполненной пользовательскими данными переменной `m_PListStr` заполняет объект `m_PclLists` объектами `PList`, соответствующими указанным частицам.

Метод `execute()` осуществляет обработку события. На основе списка реконструированных частиц события, прочитанного из хранилища в формате `sct::Particle`, и списка частиц, заданного пользователем, создаются объекты типа `Particle`. На основе инициализированного поля `m_pcl` происходит считывание переменных типа `ParticleCollection` из хранилища. Для каждой пары строк, записанной в поле `m_PclLists`, создается соответствующий объект класса `ParticleList` и, при необходимости, `ParticleList` для античастиц. На основе каждой прочитанной частицы события, хранящейся в формате `ParticleCollection`, создается объект класса `Particle` и его параметры проверяются на предмет соответствия заданному кинематическому ограничению посредством объекта `Cut` (описан в разделе 4.2.5). На основе отобранных частиц заполняются соответствующие их типам контейнеры

ParticleList.

Рассмотрим пример. Пусть в событии было зарегистрировано три частицы: положительно заряженная, отрицательно заряженная, и нейтральная. Такие частицы прочитаны из хранилища в формате sct::Particle. Пусть пользователь производит отбор частиц π^+ и K^- и их зарядово-сопряженных. В таком случае по завершению метода execute() в соответсвующие списки ParticleList в объекте m_pListMap будут добавлены следующие объекты: Particle, определенный как положительно заряженный пион, созданный на основе положительно заряженной реконструированной частицы, и его анти-частица. Аналогично для каонов. Таким образом каждая заряженная реконструированная частица послужила основой для создания двух объектов Particle, соответствующих заряженным частицам разных типов.

4.2.4. Пакет AnaParticleCombiner

Пакет AnaParticleCombiner предназначен для реконструкции частиц, осуществляемой через перебор всевозможных комбинаций дочерних частиц и конструировании из них материнской частицы, удовлетворяющей заданным кинематическим ограничениям. Данный инструмент реализован через класс ParticleCombinerAlg, являющийся алгоритмом Gaudi, и два вспомогательных класса: ParticleIndexGenerator и ParticleListGenerator.

Класс ParticleIndexGenerator по заданным длинам списков частиц генерирует всевозможные комбинации индексов частиц. В сгенерированных комбинациях присутствует по одному индексу из каждого списка, таким образом количество элементов в комбинации соответствует количеству списков. Порядок элементов в комбинации не важен.

Конструктор класса принимает на вход вектор размеров списков. С помощью алгоритма STL std::accumulate определяется количество доступных комбинаций. Метод класса loadNext() реализует механизм итерации по сге-

нерированным наборам индексов.

Класс `ParticleListGenerator` реализует алгоритм реконструкции частиц, создавая материнские частицы типа `Particle`.

В качестве аргументов конструктор класса принимает строку распада, условия отбора на материнскую частицу, и объекты `ParticleList`, содержащие конечные частицы рассматриваемого распада.

Обработка осуществляется через метод `process`:

- По строке распада инициализируется объект класса `DecayDescriptor`.
- Среди данных объектов `ParticleList` выбираются соответствующие конечным частицам указанного распада.
- С помощью объекта класса `ParticleIndexGenerator` для выбранных `ParticleList` происходит перебор всевозможных комбинаций конечных неповторяющихся частиц. При этом происходит проверка, что все дочерние частицы кандидата были реконструированы на основе разных сигналов систем детектора.
- На основе каждой комбинации создается объект `Particle`, соответствующий кандидату в материнские частицы: его PDG код известен из объекта `DecayDescriptor`, дочерние частицы соответствуют частицам рассматриваемой комбинации. Кинематические параметры, такие как импульс, энергия и масса, вычисляются на основе закона сохранения импульса по параметрам дочерних частиц.
- Реконструированная материнская частица также рассматривается на предмет соответствия кинематическим ограничениям и, при успешном прохождении этого отбора, добавляется в финальный список соответствующего объекта класса `ParticleList`.

Класс `ParticleCombinerAlg` реализован как один из алгоритмов `Gaudi`. Используя функционал вспомогательного класса `ParticleListGenerator`, `ParticleCombinerAlg` осуществляет реконструкцию распада, заданного пользователем через интерфейс.

Определены следующие поля класса: словарь с именами конечных частиц и соответствующих `ParticleList`, объекты `Property`, позволяющие пользователю задать строку распада, кинематические ограничения и обозначить необходимость рассмотрения материнских античастиц; объекты `DataHandle` для входных и выходных данных, и объект `ParticleListGenerator`.

В конструкторе алгоритма `ParticleCombinerAlg` вызываются функции `declareProperty` для инициализации объектов `DataHandle`.

В методе `initialize()` поле класса типа `ParticleListGenerator` инициализируется по заданной строке распада и строке условий отбора.

При вызове метода **`execute()`** читаются объекты `ParticleList`, соответствующие спискам конечных частиц каждого события, и с помощью объявленного `ParticleListGenerator` происходит реконструкция кандидатов в материнские частицы и формирование из них объекта `ParticleList`, который записывается во временное хранилище.

4.2.5. Пакет `AnaVarManager`

Пакет `AnaVarManager` отвечает за вычисление кинематических параметров частиц и их запись в конечное дерево в формате `n-tuple`. Пакет включает в себя основной класс `VarsToNtupleAlg`, являющийся алгоритмом `Gaudi`, и три вспомогательных класса: `Manager`, `Variable` и `Cut`.

Класс `Manager`, а также внутренние классы `Proxy` и `GroupProxy`, являются техническими инструментами, с помощью которых реализовано сопоставление кинематических параметров и их строковой записи.

Среди полей класса `Manager` определены три вида функций:

- `Simple`-функции принимают объекты `Particle` в качестве аргумента и возвращают число `double`.
- `Parameter`-функции при таком же выходном значении помимо объекта `Particle` принимают вектор параметров типа `double`.
- `Meta`-функции по вектору строк возвращают указатель на функцию.

Основная информация о кинематических параметрах хранится в структуре `VarBase` (строки 1-5 листинга 13): имя вычисляемой переменной (например, `"E"` для энергии частицы), строка с комментарием для переменной (например, `"Particle energy"`) и строка, обозначающая, к какой группе переменных принадлежит данный параметр (например, `"Kinematics"`). Имена переменных могут содержать только цифры, буквы латинского алфавита и знак `"_"`). Связь информации о переменной в формате `VarBase` с указателем на функцию для расчета значения этой переменной для конкретной частицы реализована через шаблон `Var` (строки 6-9). Среди полей класса также есть контейнер для всех заданных переменных (строка 10) и словарь их имён. При написании класса `Manager` был использован паттерн Singleton, подразумевающий, что может быть определен лишь один экземпляр данного класса. Этот выбор обусловлен тем, что в объекте данного класса хранятся все объявленные кинематические переменные и дублирование не имеет смысла.

```
1 struct VarBase {
2 std::string name;
3 std::string description;
4 std::string group;
5 };
6 template<typename F> struct Var : public VarBase {
7 F function;
8 Var(std::string n, F f, std::string d, std::string
```

```
    g = "") : VarBase(std::move(n), std::move(d), std
    ::move(g)), function(f) {}
};
std::vector<VarBaseConstPtr>
    m_variablesInRegistrationOrder;
```

Листинг 13. Некоторые поля класса `Manager`, определенного на `C++`.

Метод `registerVariable` по заданным имени, функции и строке с комментарием создает объект `Var` и добавляет его в список переменных. Метод `createVariable` принимает на вход имя переменной, заданной пользователем, и сопоставляет это имя функциям, уже зарегистрированным в данном модуле. Метод `evaluate` по заданному имени функции и заданной частицы вычисляет значение функции для данной частицы.

Класс `Variables` предназначен для задания стандартных переменных. В нем определяются функции для вычисления различных кинематических параметров частиц. С помощью макроса `REGISTER_VARIABLE` переменные, определенные в данном классе, добавляются в поле класса `Manager`, хранящее все зарегистрированные функции и их имена.

```
double particleP(const Particle& part) {return part
    .p(); }
REGISTER_VARIABLE("p", particleP, "momentum
    magnitude")
```

Листинг 14. Пример регистрации переменной через класс `Variables`, определенный на C++.

Класс `Cut` отвечает за обработку строк с кинематическими ограничениями, такими как `"E > 0.3 and p <= 0.2"`. Операции представлены через вспомогательные классы `OPE` и `Operation`.

Класс `OPE` хранит нумерованные обозначения доступных операций: `EMPTY` для отсутствия ограничений, `AND` и `OR` для логических ”и” и ”или” соответственно, `LT` и `GT` для строгих неравенств, `LE` и `GE` для нестрогих неравенств, `EQ` и `NE` для проверки равенства и неравенства соответственно, и `NONE` для операций, не являющихся ни одной из вышеперечисленных.

Класс `Operation` выполняет вспомогательную функцию, связывая объект типа `Cut` с операцией. Класс `Cut` является дружественным классом к классу `Operation`.

Объекты этого класса имеют единственное поле `m_cut`, хранящее объекты класса `Cut`. Соответственно, конструктор класса `Operation` принимает объект `Cut` в качестве аргумента и инициализирует им поле. Методы данного класса, используя поля и методы поля `m_cut`, по заданной частице могут получать значения параметров, а также проверять их значение на соответствие заданному критерию.

Класс `Cut`: так как существует порядок выполнения операций, ограничения можно представить в виде дерева, сохраняющего иерархию операций, соответственно класс `Cut` имеет рекуррентную структуру: среди полей класса есть объекты типа `Cut`: `m_left` и `m_right`, соответствующие левой и правой частям относительно знака операции. В листьях деревьев, отображающих иерархию ограничения, хранятся либо кинематические параметры, либо числовое значение. Например, в полях класса `Cut`, заданном по строке `"E > 0.1"`, в `m_left` будет лежать указатель на объект типа `Cut`, заданный по строке `"E"`, а в `m_right` – `Cut` от строки `"0.1"`, а в `m_operation` – объект `Operation`, заданный по операции `"GT"`, соответствующей `">"`, и самому объекту `Cut`. В объекте `m_left` поле `m_isNumeric` будет равно false, а `m_var` инициализирован переменной, заданной по строке `"E"`; в `m_right` поле `m_isNumeric` будет равно true, а `m_number` равен 0.1.

```
1  class Cut {
2    friend class Operation;
3    Manager::SimpleVarPtr m_var;
4    double m_number;
5    bool m_isNumeric;
6    CutPtr m_left;                  // Left-side cut
7    CutPtr m_right;                 // Right-side cut
8    std::unique_ptr<Operation> m_operation;
9    ...
10 };
```

Листинг 15. Некоторые поля класса `Cut`, определенного на `C++`.

При инициализации методом `compile()` объекта класса `Cut` по строке происходит проверка строки на соответствию одному из трех типов условий: является ли она условием типа $n_1 < A < n_2$, логическим условием (таким как A *or* B), или условием типа $A < n$. В таких случаях происходит разбиение ограничения на части и инициализация полей `m_left` и `m_right`. Иначе проверяется, является ли данный объект листом в иерархическом дереве, то есть числом или переменной. Имея построенное дерево, то есть инициализированный строкой объект `Cut`, с помощью метода `check()`, принимающего на вход частицы в формате `Particle`, можно проверить, удовлетворяют ли параметры данной частицы заданному ограничению.

Класс `VarsToNtupleAlg`, являющийся алгоритмом `Gaudi`, создает `n-tuple` и записывает в него заданные переменные, рассчитанные для конкретных частиц. Выходной файл имеет формат `ROOT TTree`.

Поля класса представлены в листинге 16. С помощью объектов `Property` фреймворка `Gaudi` через пользовательский интерфейс определены следующие объекты: имя списков в дереве, имя дерева `ROOT TTree` и список пере-

менных для записи. Входные данные, то есть списки частиц, получены через интерфейс DataHandle. Списки переменных для записи хранятся в виде вектора объектов VarSet, в который входят переменные в формате SimpleVars, ветки выходного дерева и частицы, для которых нужно записать параметры, в виде DecayDescriptor.

```
1  class VarsToNtupleAlg : public AuroraAlgorithm {
2    using PListMap = std::unordered_map<std::string,
       ParticleListPtr>;
3    using VarNamesAndDD = std::vector<std::string>;
4    Gaudi::Property<std::string> m_listName{this, "
       listName", "", "List name"};
5    Gaudi::Property<std::string> m_fileName{this, "
       fileName", "", "ROOT file name"};
6    Gaudi::Property<std::vector<VarNamesAndDD>>
       m_vars{this, "vars", {}, "Variable names and
       DecayDescriptor"};
7    DataHandle<PListMap> m_pListMapI{"pListMapI",
       Gaudi::DataHandle::Reader, this};
8    NTuple::Tuple* m_ntprt;
9    struct VarSet {
10     std::vector<Variable::Manager::SimpleVarPtr>
         pars;
11     std::vector<NTuple::Item<ftype>> vals;
12     std::unique_ptr<DecayDescriptor> dd;
13   };
14   std::vector<VarSet> m_varSets;
15 };
```

Листинг 16. Поля класса VarsToNtupleAlg, определенного на C++.

Как у наследника объекта Algorithm, VarsToNtupleAlg имеет четыре метода. В **конструкторе** происходит связывание с входными данными посредством вызова declareProperty.

В **методе initialize()** происходит заполнение основных полей класса:

- С помощью сервиса ntupleSvc происходит инициализация выходного дерева по заданному имени файла.
- По заданным m_vars происходит заполнение переменной m_varSets
- По данным поля m_varSets объявляются ветки выходного дерева, с формированием имен по типу "E" для материнских частиц и по типу "E_K" для отобранных дочерних частиц.

Метод execute() заполняет листья дерева по считанному событию: итерируясь по считанным частицам события и по объектам вектора m_varSets, программа выполняет расчет указанных кинематических параметров для конкретных частиц (с помощью класса Manager), и записывает эти параметры в выходной n-tuple.

5. Заключение

В ходе данной работы исследовалось решение проблемы создания инструментов первичного отбора событий для эксперимента SCT. Были сформулированы требования в разрабатываемым инструментам.

После обзора существующих решений, анализа их преимуществ и недостатков, было принято решение реализовать инструменты первичного отбора на языке C++ в рамках программной среды Aurora с интерфейсом на языке python для удобства пользователей. В качестве технического решения предложено реализовать инструменты, беря за основу алгоритмы анализа эксперимента Belle II. Был реализован набор пакетов, позволяющий поэтапно производить отбор необходимых частиц, реконструировать распады, отбирать

частицы по кинематическим критериям, и использовать строковое описание частиц и распадов.

Основным практическим результатом работы является создание модуля анализа данных, интегрированного в программное обеспечение эксперимента SCT. Создана база, на основе которой возможно расширение функционала.

Список литературы

[1] B.I. Khazin et al., ”Super Charm-Tau Factory”, Conceptual design report, (2018).

[2] F. Gaede, B. Hegner, P. Mato, “PODIO: An Event-Data-Model Toolkit for High Energy Physics Experiments”, J. Phys.Conf.Ser. 898, 7(2017), DOI:10.1088/1742-6596/898/7/072039.

[3] Anders Ryd et al., “EvtGen: A Monte Carlo Generator for B-Physics”, EVTGEN-V00-11-07, (2005).

[4] S. A. Yost et al., ”KKMC-hh: A Precision Event Generator for EW Radiative Corrections in Hadron Scattering”, (2018), arXiv:1801.03560 [hep-ph]

[5] M. Chrzaszcz et al., ”TAUOLA of τ-lepton decays - framework for hadronic currents, matrix elements and anomalous decays”, Computer Physics Communications, 232 (2018).

[6] T. Sjöstrand, S. Mrenna and P. Skands, ”A brief introduction to PYTHIA 8.1”, Computer Physics Communications, 178.11, (2008).

[7] A.B.Arbuzov et al., ”Monte-Carlo generator for e^+e^- annihilation into lepton and hadron pairs with precise radiative corrections”, The European Physical Journal, C46.3 (2006).

[8] P. Golonka and Z. Was, ”PHOTOS Monte Carlo: a precision tool for QED corrections in Z and W decays”, The European Physical Journal, C45.1 (2006).

[9] A. Buckley et al., ”The HepMC3 Event Record Library for Monte Carlo Event Generators”, arXiv 1912.08005, (2019).

[10] M. Frank et al., ”DD4hep: A Detector Description Toolkit for High Energy Physics Experiments”, J. Phys. Conf. Ser., 513 (2014).

[11] S. Agostinelli et al., ”GEANT4: A Simulation toolkit”, Nucl. Instrum.Meth., A506 (2003).

[12] G. Barrand et al., ”GAUDI - A software architecture and framework for building HEP data processing applications”, Comput.Phys.Commun., 140(2001).

[13] Stefan Roiser, ”Event data modelling for the LHCb experiment at CERN”, CERN-THESIS-2003-031, (2003).

[14] T. Kuhr et al., ”The Belle II Core Software”, Computing and Software for Big Science, 3.1 (2018).

[15] M.Ritter, ”Data Analysis in Belle II”, HEP Analysis Eco-System Workshop, (2017).

[16] G. Guennebaud, B. Jacob et al., Eigen v3., (2010), URL: http://eigen.tuxfamily.org.

Приложение: пользовательский скрипт на python, осуществляющий отбор распада $D^0 \to \pi^+ K^-$ с кинематическими ограничениями.

```
1  import os
2  from Configurables import ApplicationMgr
3
4  from Gaudi.Configuration import *
5  from Configurables import ScTauDataSvc
6  from Configurables import PodioInput
7  from Configurables import sct__ana__EventLoader as
     EventLoader
8  from Configurables import
     sct__ana__ParticleCombinerAlg as ParticleCombiner
9  from Configurables import sct__ana__VarsToNtupleAlg
      as VarsToNtupleAlg
10 from Configurables import NTupleSvc
11
12 podioevent = ScTauDataSvc("EventDataSvc", input='./
     psi3770_parsim.root')
13 podioinput = PodioInput("PodioReader", OutputLevel=
     INFO, collections=['Particles'])
14
15 evlo = EventLoader('EvtLoader')
16 evlo.pcl.Path = 'Particles'
17 evlo.pListMap.Path = 'FSPLists1'
18 evlo.plists = [['pi+ cc'], ['K+ cc', '']]
19
```

```
20 cmbr = ParticleCombiner('Cmbr',
21     decStr = 'D0 -> pi+ K-',
22     cutStr = '',
23     selfConj = False
24 )
25
26 cmbr.pListMapI.Path = evlo.pListMap.Path
27 cmbr.pListMapO.Path = 'FSPLists2'
28
29 tupl = VarsToNtupleAlg('D0Tuple')
30 tupl.listName = 'D0'
31 tupl.fileName = 'd0_tuple/tup'
32 tupl.vars = [
33     ['px', 'py', 'pz', 'p', 'E', 'pt', 'M', ''],
34     ['charge', 'px', 'py', 'pz', 'E', 'D0 -> ^pi+ ^
        K-'],
35 ]
36
37 tupl.pListMapI.Path = cmbr.pListMapO.Path
38 ntSvc = NTupleSvc ( Output = ["d0_tuple DATAFILE='
     dtup.root' OPT='NEW' TYP='ROOT'"] )
39
40 options = {
41     'TopAlg' : [podioinput, evlo, cmbr, tupl],
42     'EvtSel' : 'NONE',
43     'ExtSvc' : [podioevent],
44     'EvtMax' : 10000,
45     'StatusCodeCheck' : True,
```

```
46      'AuditAlgorithms' : True,
47      'AuditTools'      : True,
48      'AuditServices'   : True,
49      'OutputLevel'     : INFO,
50      'HistogramPersistency' : 'ROOT',
51  }
52  ApplicationMgr(**options)
```

Листинг 17. Пользовательский скрипт на `python`, осуществляющий отбор распада $D^0 \to \pi^+ K^-$ с кинематическими ограничениями.