МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
(НИЯУ МИФИ)

ИНСТИТУТ ЯДЕРНОЙ ФИЗИКИ И ТЕХНОЛОГИЙ
КАФЕДРА №40 «ФИЗИКА ЭЛЕМЕНТАРНЫХ ЧАСТИЦ»

УДК ???

На правах рукописи

СИМБИРЯТИН ЛЕВ ЛЕОНИДОВИЧ

ПРОГРАММНАЯ ПЛАТФОРМА ДЛЯ МНОГОЭТАПНОЙ ОБРАБОТКИ ДАННЫХ ФИЗИЧЕСКОГО ЭКСПЕРИМЕНТА НА ОСНОВЕ ФРЕЙМВОРКА GAUDI

Направление подготовки 14.04.02 «Ядерная физика и технологии»
Диссертация на соискание степени магистра

Научный руководитель,
к.ф-м.н., доц. _____ Е. Ю. Солдатов

Научный консультант,
к.ф-м.н. _____ А. С. Жемчугов

Москва 2025

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

ПРОГРАММНАЯ ПЛАТФОРМА ДЛЯ МНОГОЭТАПНОЙ ОБРАБОТКИ ДАННЫХ ФИЗИЧЕСКОГО ЭКСПЕРИМЕНТА НА ОСНОВЕ ФРЕЙМВОРКА GAUDI

Студент	_____	Л. Л. Симбирятин
Научный руководитель, к.ф-м.н., доц.	_____	Е. Ю. Солдатов
Научный консультант, к.ф-м.н.	_____	А. С. Жемчугов
Рецензент, к.ф-м.н.	_____	И. О. Фамилия
Секретарь ГЭК, к.ф-м.н.	_____	А. А. Кириллов
Зав. каф. №40, д.ф.-м.н., проф.	_____	М. Д. Скорохватов
Рук. учеб. прог., к.ф-м.н., доц.	_____	Е. Ю. Солдатов

ОГЛАВЛЕНИЕ

Введение **5**

1 Эксперимент SPD **7**

1.1 Основная цель эксперимента 7

1.2 Детектор SPD 8

2 Программное обеспечение в физике высоких энергий **10**

2.1 Geant4 12

2.1.1 Описание геометрии 12

2.1.2 Пользовательские классы 13

2.2 Pythia8 15

2.3 HepMC3 16

2.4 GeoModel 17

3 Фреймворк Gaudi **18**

3.1 Архитектура Gaudi 18

3.1.1 Алгоритмы 19

3.1.2 Сервисы 20

3.1.3 Инструменты (Tools) 21

3.2 Конфигурирование приложения. Job opotions 23

3.3 Контроль и планирование 24

3.3.1 Работа Gaudi-приложения 25

3.4 Многопоточность в Gaudi: GaudiHive 26

3.5 Применение паттернов ООП в Gaudi 29

3.5.1 Паттерн Factory Method 29

3.5.2 Паттерн Bridge 31

3.5.3 Паттерн Adapter 32

4 Gaudi в SPD **34**
4.1 Интеграция Pythia8 34
4.2 Интеграция HepMC3 36
4.3 Интеграция GeoModel 37
4.4 Магнитное поле 38
4.5 Интеграция Geant4 39

Заключение **44**

Список использованных источников **45**

ВВЕДЕНИЕ

Ускорительный комплекс NICA [1] (Nuclotron based Ion Collider fAcility) является проектом масштаба мегасайенс, реализуемым на базе ОИЯИ (Дубна, Россия). На коллайдере предусмотрены две точки пересечения пучков заряженных частиц, в одной из которых предполагается установить детектор SPD [2] (Spin Physics Detector) с целью изучения спиновой структуры протона и дейтрона. Коллайдер предоставляет уникальную возможность для изучения поляризованных pp и dd столкновений с $\sqrt{s} = 27$ ГэВ и светимостью порядка 10^{32} см$^{-2}$с$^{-1}$.

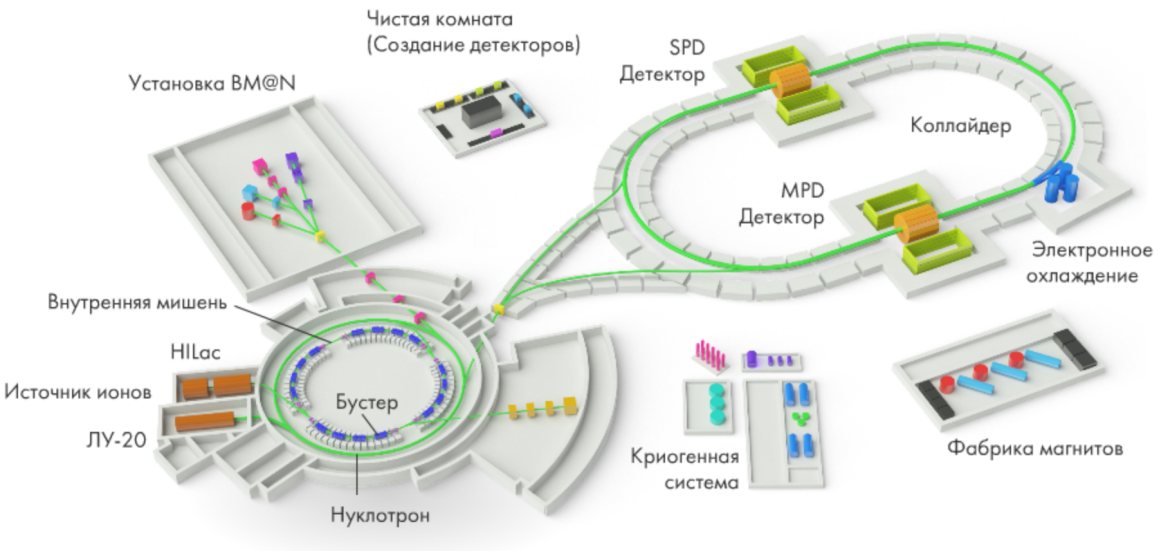


Рисунок 1 — Ускорительный комплекс NICA

Как и любому другому крупному эксперименту в физике высоких энергий, SPD необходимо программное обеспечение для решения задач, связанных с Монте-Карло генерацией и моделированием физических событий, а также с обработкой данных, как сгенерированных, так и реальных, поступивших с детектора. Текущим вариантом физического ПО эксперимента SPD является пакет SpdRoot [3], написанный на основе фреймворка FairRoot [4]. Решая вышеперечисленные задачи, SpdRoot все же обладает

рядом существенных недостатков. В связи с этим, руководством эксперимента было принято решение о разработке замены на базе фреймворка Gaudi [5]. Выбор Gaudi обусловлен его доказанной применением в экспериментах в области физики высоких энергий (см. [6], [7] и [8]) надежностью, а также поддержкой многопоточности.

Разработка физического ПО является комплексной задачей. Ввиду отсутствия реальных данных с детектора, первостепенным является обеспечение средств генерации и моделирования, откуда и вытекают цели и задачи данной работы.

Целью работы является разработка и реализация архитектуры программной платформы, унифицирующей прикладное программное обеспечение для многоэтапной обработки данных эксперимента SPD на базе фреймворка Gaudi. В рамках поставленной цели были решены следующие задачи:

- обзор архитектуры фреймворка Gaudi;
- разработка и реализация архитектуры интеграции библиотеки Pythia8 для генерации первичных вершин;
- разработка средств описания магнитного поля в детекторе;
- интеграция библиотеки GeoModel как средства описания геометрических параметров детектора;
- разработка и реализация архитектуры интеграции библиотеки Geant4 для проведения моделирования;

1 ЭКСПЕРИМЕНТ SPD

Несмотря на важность поиска проявлений частиц, выходящих за рамки Стандартной модели, в рамках барионной материи по-прежнему остается множество открытых вопросов. Даже протон не может считаться в полной мере изученной частицей. В наивной кварковой модели протон представляет собой комбинацию двух u и одного d кварка. Эта простейшая кварковая модель позволяет предсказать такие свойства как электрический заряд, изоспин, четность, магнитный момент. Этот результат является действительно удивительным, ведь такая модель не учитывает угловые моменты кварков, морские кварки, а также глюоны. КХД является современным инструментом описания сильного взаимодействия, она с успехом применяется для описания множества процессов. Основным нюансом КХД является ее непертурбативность на низких энергиях, в частности, одной из нерешенных проблем остается описание свойств адронов (в том числе и протона) напрямую из динамики составляющих их кварков и глюонов.

Детектор SPD (Spin Physics Detecror) будет размещен в одной из двух точек столкновения пучков коллайдера NICA (Дубна, Россия). Целью построения SPD является изучение спиновой структуры нуклонов в поляризованных pp ($\sqrt{s} < 27$ ГэВ) и dd ($\sqrt{s} < 13.5$ ГэВ) столкновениях.

1.1 ОСНОВНАЯ ЦЕЛЬ ЭКСПЕРИМЕНТА

Одним из способов описания внутренней партонной структуры нуклона является использование функций партонных распределений PDF (Parton Distribution Function). В неполяризованном простейшем случае эта функция описывает вероятность найти внутри нуклона партон, несущий определенную долю общего импульса. В общем же случае необходимо также учитывать не только продольную компоненту, но и поперечную (напри-

мер, эффект Сиверса [9]), а также поляризацию как самого нуклона, так и партонов внутри него.

В то время как вклад кварков в общий спин нуклона был довольно точно измерен коллаборациями EMC, HERMES и COMPASS, измерения по глюонной компоненте либо являются менее точными, либо отсутствуют вовсе.

Основная цель эксперимента SPD - извлечь информацию о глюонных функциях распределения, зависящих от поперечного импульса (TMD PDFs), для протона и дейтрона, через измерение одинарных и двойных спиновых асимметрий в процессах рождения чармониев, очарованных частиц, а также прямых фотонов [10].

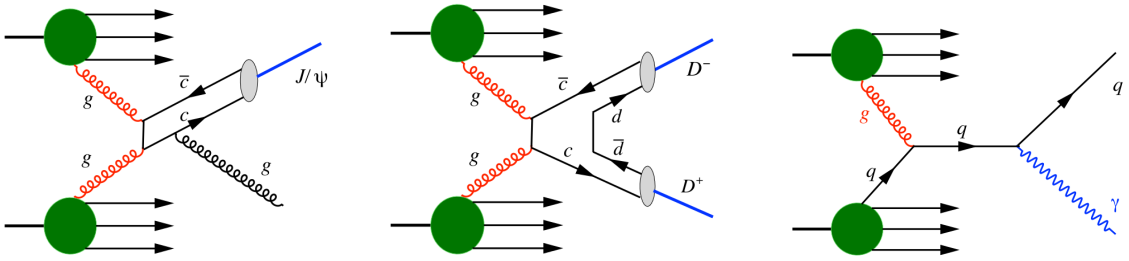


Рисунок 1.1 — Фейнмановские диаграммы процессов-пробников: рождение (а) чармониев, (б) очарованных частиц, (в) прямых фотонов

1.2 ДЕТЕКТОР SPD

Основной целью эксперимента является извлечение TMD PDF для глюонов через измерение спиновых асимметрий в процессах рождения чармониев, очарованных частиц, а также прямых фотонов. Поставленная цель, а также обозначенные процессы-пробники определяют вид и необходимые характеристики детектора SPD.

SPD представляет собой универсальный 4π детектор с характерной для коллайдерных экспериментов цилиндрической формой. Его компонентами являются:

- кремниевый вершинный детектор (VD) с разрешением выше 100 мкм для реконструкции вторичных вершин распадов D мезонов;
- трековая система (TS) $\sigma_{p_T}/p_T \approx 2\%$;

- время-пролетная система (TOF) с разрешением порядка 60 пс для разделения π/K и K/p;
- детектор FARICH для улучшения разделения π/K и K/p;
- электромагнитный калориметр (ECal) с энергетическим разрешением $\sim 5\%/\sqrt{E}$ для регистрации фотонов;
- мюонная система (RS);
- пара счетчиков столкновений (BBC) и калориметров нулевых углов (ZDC) для контроля поляризации и светимости;

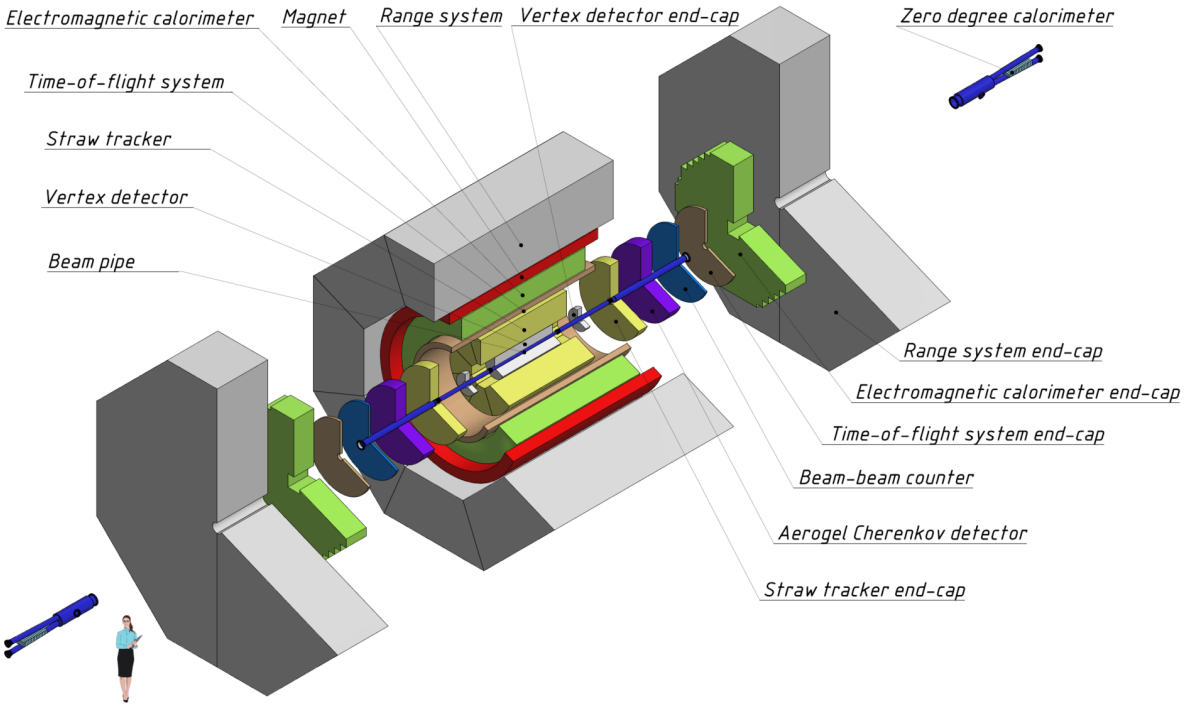


Рисунок 1.2 — Макет детектора SPD в полной сборке

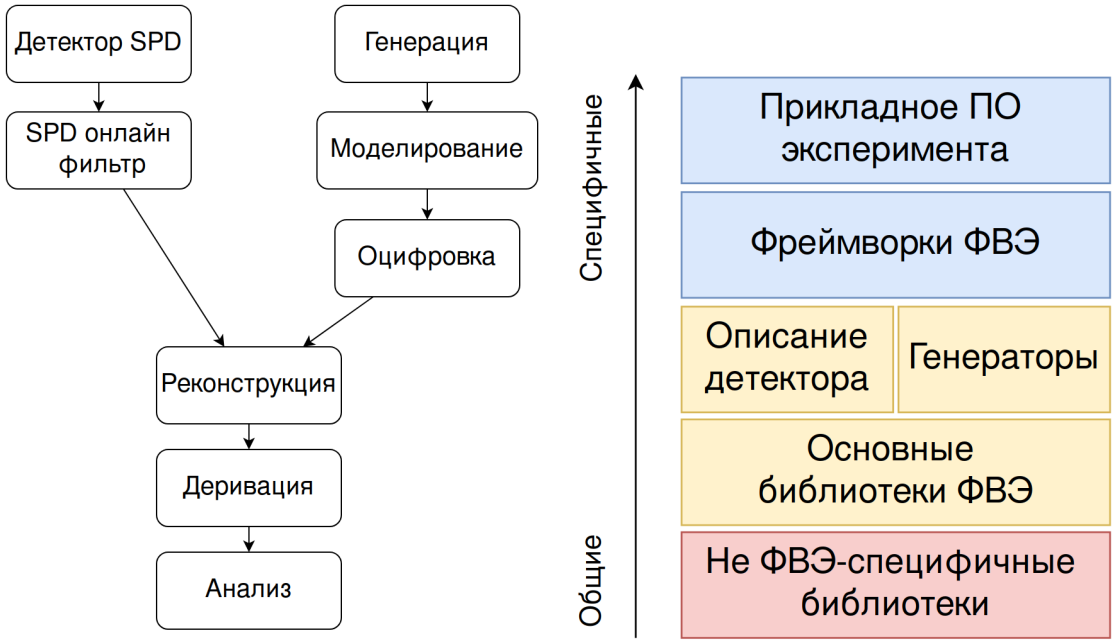
В силу трудностей, возникающих при построении аппаратного триггера, для SPD предполагается безтриггерная система сбора данных. В совокупности с высокой частотой столкновений (до 4 МГц) и сотнями тысяч каналов детектора это представляет собой сложную задачу по разработке эффективной системы сбора и обработки данных.

2 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ В ФИЗИКЕ ВЫСОКИХ ЭНЕРГИЙ

Обработка данных в экспериментах в физике высоких энергий производится в несколько этапов. Этапы обработки данных в эксперименте SPD представлены на рисунке 2.1а. Данные либо поступают с детектора и проходят через отбор в SPD онлайн фильтре, либо же моделируются. Моделирование включает генерацию первичных вершин, моделирование прохождения частиц через вещество детектора, а также процедуру оцифровки, в ходе которой смоделированные в детекторе энергопотери частиц превращаются в сигналы с электроники детектора. Сигналы с детектора, как смоделированные, так и реальные, в дальнейшем подвергаются процедуре реконструкции, в ходе которой восстанавливается картина события в детекторе (родившиеся частицы и их траектории). Проведение моделирования является необходимым, так как оно позволяет определить, каким образом детектор влияет на выходные данные. Это влияние в дальнейшем должно быть учтено при обработке реальных данных.

Разбиение на указанные этапы, за исключением онлайн фильтра, не является уникальным для эксперимента SPD. Каждый из таких этапов представляет собой определенную задачу, для решения которой в физике высоких энергий существуют устоявшиеся наборы инструментов, реализованные в виде соответствующего программного обеспечения. Данное программное обеспечение можно организовать в стек, представленный на рисунке 2.1б. В основе стека лежат библиотеки, не специфичные для физики высоких энергий. Они предназначены для решения технических задач, возникающих в процессе создания более высокоуровневых компонентов. Эти задачи могут быть связаны с использованием эффективных структур данных, внедрением многопоточности, обеспечением взаимодействия с операционной системой и т.п. Основным языком программирования, ис-

пользующимся в написании программ в области физики высоких энергий является C++, соответственно в качестве примеров используемых на этом уровне библиотек можно привести Boost, Intel TBB, Range-v3.



а б

Рисунок 2.1 — (а) Этапы обработки данных в ФВЭ, (б) Стек ПО в ФВЭ

Далее по стеку следуют основные для физики высоких энергий библиотеки. Именно с их помощью решаются задачи генерации, моделирования, реконструкции и анализа. К таким библиотекам относятся пакет ROOT [11], являющийся фреймворком для анализа физических данных, пакет Geant4 [12], предназначенный для моделирования взаимодействия частиц с веществом, HepMC3 [13], представляющий собой универсальный формат для работы с физическими событиями. Отдельно стоит сказать о генераторах, примером которого может служить Pythia8 [14], и об описании детектора, для которого могут использоваться такие библиотеки как GeoModel [15] или DD4hep [16].

Для унификации процесса обработки событий используются физические фреймворки. Здесь примером служит Gaudi [5], являющийся основой для данной работы. Именно через физический фреймворк идет обращение к нижележащим компонентам и на его базе создаются конкретные процедуры обработки данных.

Далее будут кратко рассмотрены библиотеки, использованные в дан-

ной работе.

2.1 GEANT4

Geant4 [12] — это программный пакет, предназначенный для моделирования прохождения частиц или излучения через вещество, написанный на C++, разработанный и поддерживаемый CERN. С помощью Geant4 можно моделировать детекторы в физике высоких энергий, а также записывать результаты моделирования для проведения дальнейшего анализа. Пакет включает в себя полный набор физических моделей для описания электромагнитных, сильных и слабых взаимодействий частиц в веществе в широком диапазоне энергий. Объектно-ориентированный дизайн пакета позволяет ему быть гибко адаптированным под любые нужды, пакет может быть использован как для написания отдельных приложений, так и интегрирован в фреймворк.

2.1.1 ОПИСАНИЕ ГЕОМЕТРИИ

Геометрия детектора в Geant4 строится из цепочки вложенных объемов. Самый большой объем, содержащий все остальные, называется мировым объемом. Каждый объем создается путем описания его формы и физических характеристик, а затем помещается в родительский объем. Когда объем помещается внутрь другого объема, первый называется дочерним, а второй — родительским.

Для описания формы объема используется понятие твердого тела. Твердое тело - это геометрический объект, имеющий определенную форму и определенные значения для каждого из параметров этой формы.

Для описания остальных свойств объема используется понятие логического объема. Он включает геометрические свойства твердого тела и добавляет физические характеристики, такие как материал, наличие чувствительных элементов, электромагнитные магнитное поля и т.д.

Для описания положения дочернего объема в родительском используется понятие физического объема.

2.1.2 ПОЛЬЗОВАТЕЛЬСКИЕ КЛАССЫ

Класс G4RunManager содержит в себе логику выполнения программы и управляет процессом моделирования. При создании G4RunManager также создаются другие основные классы менеджеров. G4RunManager также отвечает за управление процедурами инициализации, включая вызовы методов пользовательских классов инициализации, с помощью которых должна быть предоставлена вся информация, необходимая для создания и запуска моделирования, включая:

- конструкцию детектора,
- частицы и ассоциированные с ними процессы,
- информацию о первичных частицах,
- дополнительный инструкции пользователя.

Настройка моделирования в Geant4 осуществляется с помощью так называемых пользовательских классов. С точки зрения написания программы подразумевается наследование от этих классов и имплементация определенных методов. В Geant4 предусмотрено два типа пользовательских классов: классы инициализации и классы действий.

К пользовательским классам инициализации относятся:

- G4VUserDetectorConstruction
- G4VUserPhysicsList
- G4VUserActionInitialization

Класс G4VUserDetectorConstruction используется для описания детектора. В методе G4VUserDetectorConstruction::Construct() пользователь должен описать материалы и геометрию установки, а в методе G4VUserDetectorConstruction::ConstructSDandField() — чувствительные элементы установки и электромагнитные поля. Описание чувствительных элементов осуществляется с помощью класса G4VSensitiveDetector, а описание магнитного поля — с помощью класса G4FieldManager.

Класс G4VUserPhysicsList используется для описания частиц и соответствующих им процессов. В пакете предусмотрен набор готовых описаний физики, так что пользователь может воспользоваться ими.

Последний класс G4VUserActionInitialization используется для задания пользовательских классов действий. К таковым относятся:

- G4VUserPrimaryGeneratorAction
- G4UserRunAction
- G4UserEventAction
- G4UserStackingAction
- G4UserTrackingAction
- G4UserSteppingAction

Пользовательские классы действий вызываются в строго определенные моменты процесса моделирования и позволяют таким образом пользователю влиять на его ход. В то время как все пользовательские классы инициализации являются обязательными для определения пользователем, единственным обязательным классом действия является класс G4VUserPrimaryGeneratorAction. С его помощью пользователь должен описать первичные частицы для запуска процесса моделирования.

Общая схема отношений пользовательских классов представлена на рисунке 2.2.

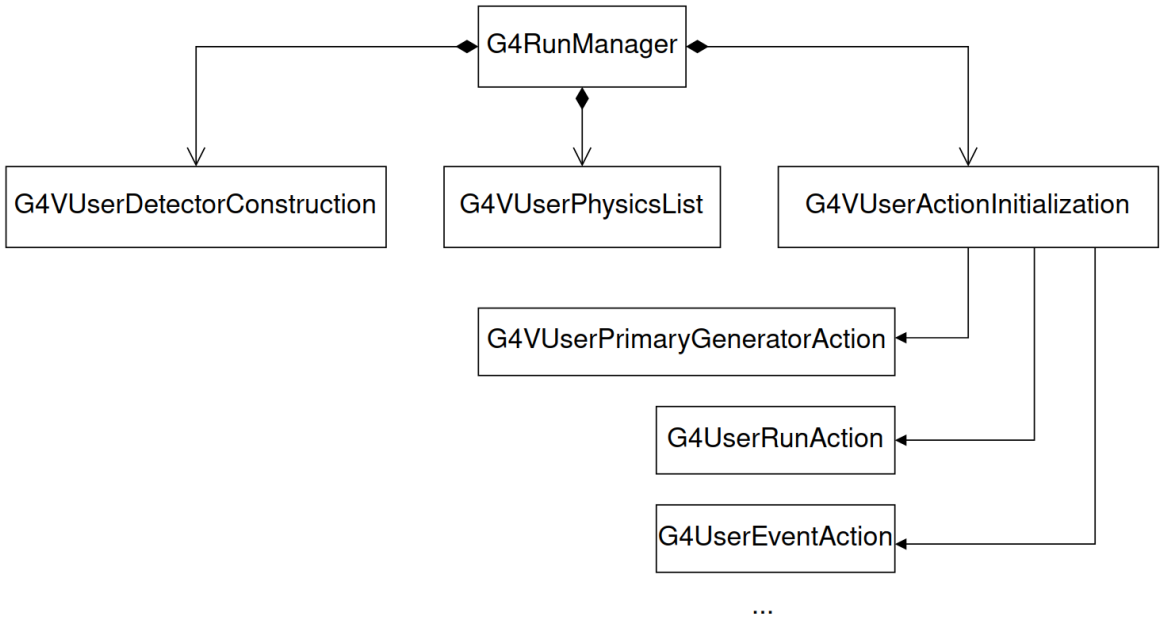


Рисунок 2.2 — Пользовательские классы в Geant4

2.2 PYTHIA8

Как было упомянуто выше, для запуска моделирования в Geant4 необходимо, в том числе, чтобы пользователь указал информацию о первичных частицах. Geant4 не предназначен для моделирования столкновения частиц, для этого в физике высоких энергий существуют генераторы первичных вершин. Произведенные ими частицы затем переносятся в Geant4 для осуществления дальнейшего моделирования.

Pythia8 [14] - это библиотека для генерации событий в физике высоких энергий, то есть для описания столкновений при высоких энергиях между электронами, протонами, фотонами и тяжелыми ядрами. В библиотеке используются как теоретические описания так и эмпирические модели для ряда физических процессов, включая жесткие и мягкие взаимодействия, распределение партонов, партонные потоки в начальном и конечном состояниях, многопартонные взаимодействия, фрагментацию и распад. Pythia8, таким образом, является универсальным Монте-Карло генератором событий.

Основным элементом библиотеки является класс Pythia. Настройка объектов этого класса осуществляется с помощью вызовов
метода Pythia::readString(), также настройки можно задать из файла с помощью Pythia::readFile(). Структура библиотеки приведена на рисунке 2.3.

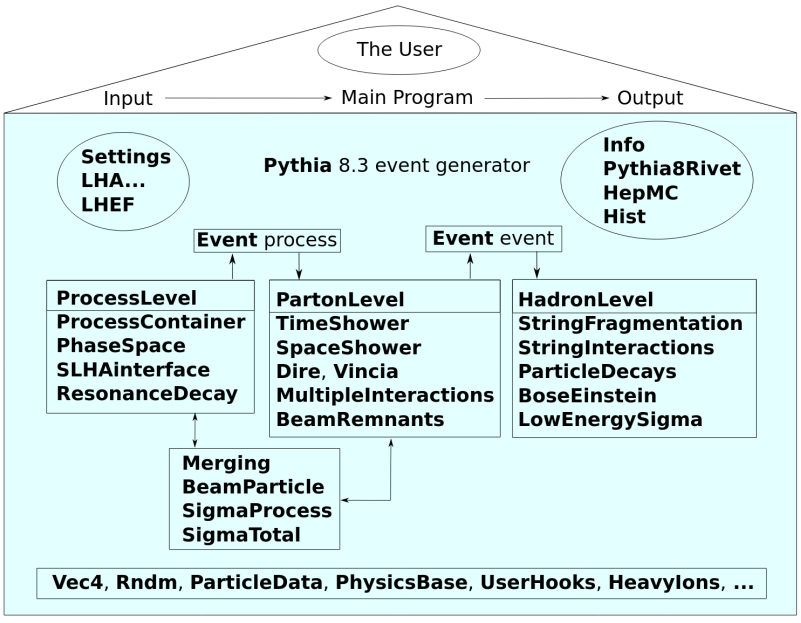


Рисунок 2.3 — Упрощенная структура библиотеки Pythia8

2.3 HEPMC3

Произведенные генератором события иногда необходимо модифицировать, фильтровать, и в любом случае сохранять на диск. Большое разнообразие генераторов, а также, соответственно, и форматов производимых ими событий событий, представляет собой существенное неудобство для компонентов физического ПО, предназначенных для работы с этими событиями. HepMC3 [13] решает эту проблему путем введения единого формата события. Современные генераторы в физике высоких энергий, в частности Pythia8, оснащены конвертерами, переводящими события из внутреннего, специфичного для каждого генератора, формата в формат HepMC3. В противном случае, такие конвертеры должны быть разработаны самостоятельно.

Запись в формате HepMC3 содержит информацию как о настройках генератора (класс HepMC3::GenRunInfo), использовавшихся в момент производства событий, так и о самом событии (класс HepMC3::GenEvent), а именно о его вершинах и частицах, с которыми также может быть ассоциирована произвольная дополнительная информация в формате атрибутов (класс HepMC3::Attribute).

Основным компонентом библиотеки HepMC3 является сериализация. Для записи в различные типы файлов используются абстрактные классы HepMC3::Reader и HepMC3::Writer. Доступными форматами записи являются:

- HEPEVT
- Asciiv3
- ROOT
- ROOTTree
- LHEF

HepMC3 поставляется с дополнительным модулем search для поиска и отбора частиц в событии. С его помощью можно создавать фильтры, отбирающие только те частицы, свойства которых удовлетворяют указанному критерию.

2.4 GEOMODEL

Geant4, как было указано выше, предоставляет собственные средства для описания геометрии детектора. Их можно использовать в случае написания отдельной программы, однако в случае, когда Geant4 используется как часть стека моделирования, ими пользоваться не стоит. Это связано с тем, что описание геометрии таким способом подразумевает написание кода и каждое дальнейшее её изменение будет требовать пересборки проекта. Но что более важно, указанное представление геометрии будет существовать только внутри Geant4, следовательно другие компоненты физического ПО, в частности, отвечающие за реконструкцию, не будут иметь к ней доступа. Исправить данные недостатки позволяет использование стороннего пакета для описания геометрии. В случае данной работы это пакет GeoModel [15]. Компоненты GeoModel представлены на рисунке 2.4.

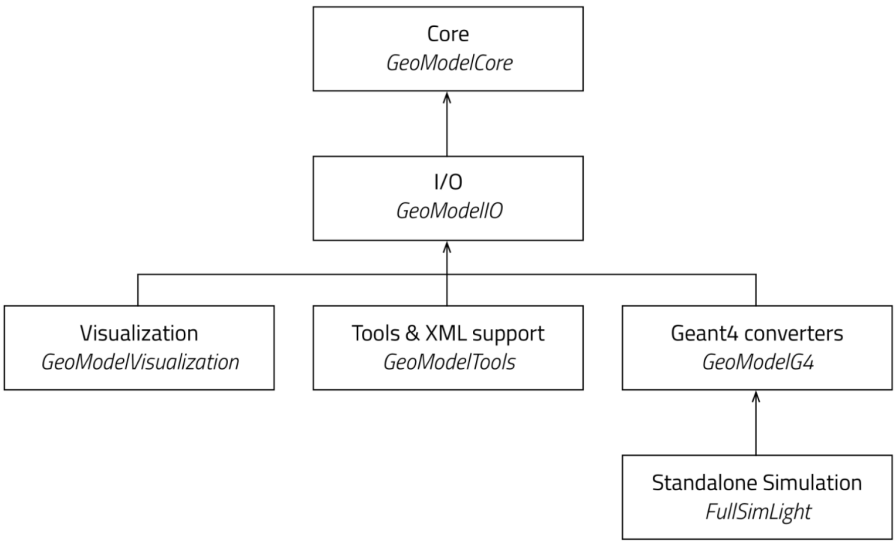


Рисунок 2.4 — Архитектура пакета GeoModel

В частности, модуль GeoModelIO позволяет хранить описание геометрии в виде SQLite базы данных, что позволяет избежать пересборок проектов, а модуль GeoModelG4 предоставляет средства для конверсии геометрии из GeoModel в Geant4.

3 ФРЕЙМВОРК GAUDI

Gaudi [5] представляет собой программный пакет, содержащий все необходимые интерфейсы и компоненты для написания на его основе фреймворков для экспериментов в области физики высоких энергий. Изначально Gaudi разрабатывался по внутренним нуждам коллаборации LHCb, однако вскоре после подключения к разработке коллаборации ATLAS стало ясно, что пакет может быть легко трансформирован и под любой другой эксперимент. Надежность пакета подтверждается его использованием в многочисленных коллаборациях по всему миру.

3.1 АРХИТЕКТУРА GAUDI

Одним из принципиальных решений при создании Gaudi стала изоляция пользователя от деталей внутреннего устройства фреймворка. Достигается такая изоляция за счет построения архитектуры, представляющей собой набор компонентов и правил их взаимодействия. У каждого компонента есть свой интерфейс и функционал. Задача же пользователя сводится к доопределению функционала конкретного компонента с сохранением его интерфейса. Программно это осуществляется путем наследования от одного из базовых классов.

Другим принципиальным решением стало явное разделение между данными и алгоритмами, оперирующими этими данными. Такое разделение обусловлено естественным подходом: данные - это набор чисел, который не стоит перегружать каким-либо дополнительным функционалом, а алгоритмы - это математические процедуры, проводимые с этими числами. Также для хранения данных на диске необходимо предоставить соответствующие конвертеры. Таким образом, в Gaudi представлены следующие базовые классы, предназначенные для пользователя:

- DataObject
- Algorithm
- Converter

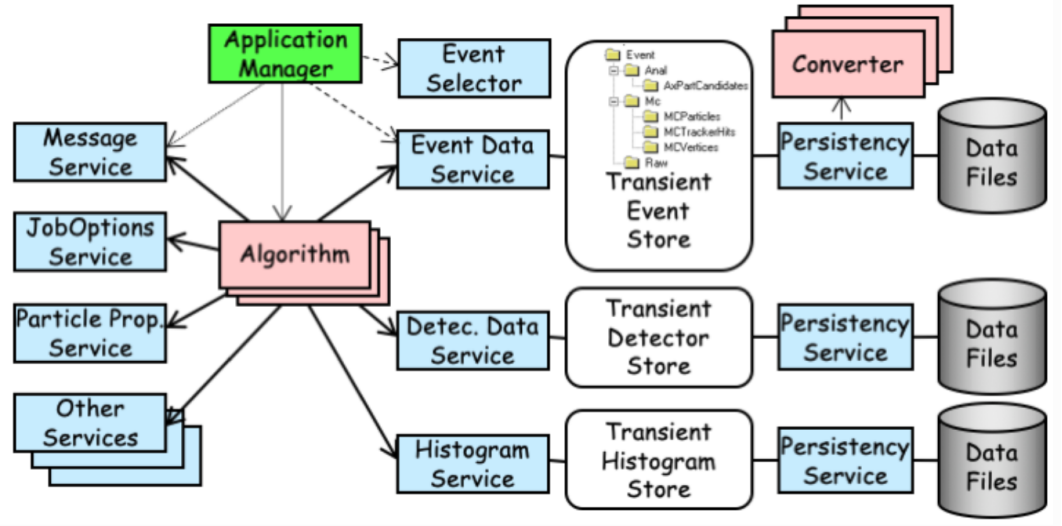


Рисунок 3.1 — Основные компоненты архитектуры Gaudi

3.1.1 АЛГОРИТМЫ

Алгоритмы главным образом производят определенные действия с данными (генерация, реконструкция и т. п.), основная часть модификации фреймворка под нужды конкретного эксперимента заключается в написании алгоритмов.

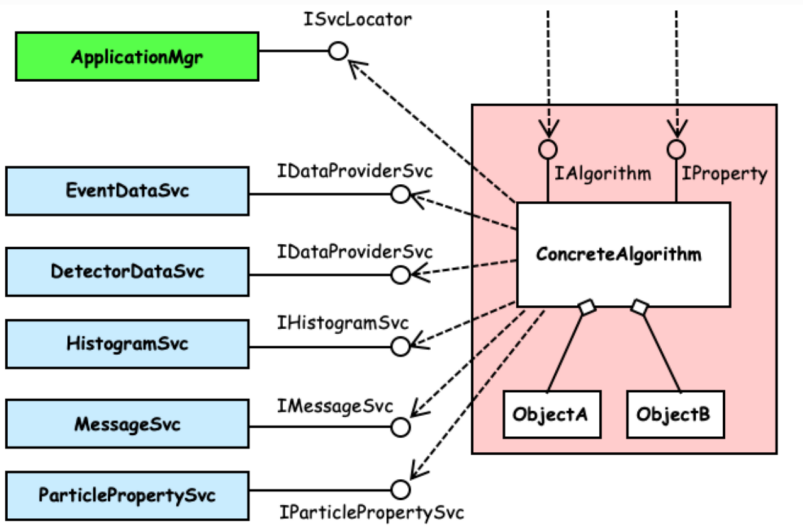


Рисунок 3.2 — Взаимодействие алгоритма с фреймворком в процессе работы

Алгоритм взаимодействует с элементами фреймворка посредством

их интерфейсов. Например, доступ к данным и их запись во временное хранилище осуществляется посредством интерфейса IDataProviderSvc, а в случае необходимости вывода алгоритмом какой-либо информации можно воспользоваться MessageSvc, обращение к которому реализуется через интерфейс IMessageSvc.

Алгоритм является конфигурируемым. Так, перед запуском можно установить значения для внутренних переменных (например, пороговые значения для отборов событий). Это возможно благодаря тому, что базовый класс Algorithm реализует сразу два интерфейса, в том числе IProperty, что дает возможность сервису JobOptionSvc в момент конфигурирования обращаться к полям алгоритма и задавать их значения. Вторым интерфейсом, который реализуется базовым классом Algorithm, является IAlgorithm. IAlgorithm используется для управления алгоритмом в процессе работы фреймворка. Также этот интерфейс содержит три чисто виртуальных метода, реализация которых целиком ложится на конечного пользователя:

- **Initialize**, который может быть использован для создания выходных гистограмм, конфигурирования побочных алгоритмов и т.п.
- **Execute**, который вызывается единожды на событие и совершает соответствующие какой-либо физической задаче преобразования над данными, относящимися к этому событию. Для побочных алгоритмов execute можно вызывать более одного раза.
- **Finalize**, который вызывается в конце работы программы и может быть использован для подведения итоговой статистики, фитирования итоговых гистограмм и т.п.

3.1.2 СЕРВИСЫ

Сервисы предназначены для решения общих задач, возникающих в ходе работы приложения. К таковым можно отнести чтение и запись данных в Transient Data Store, вывод сообщений, генерацию случайных чисел, получение свойств частиц и т. д. Сервисы не относятся к какому-то конкретному алгоритму, они наравне с алгоритмами являются самостоятельными компонентами фреймворка. Так, например, за создание конкретного набора алгоритмов на основе конфигурационного файла отвечает сервис

JobOptionSvc.

Сервисы создаются единожды в начале работы приложения и затем вызываются другими компонентами фреймворка. При этом при создании используется ленивая инициализация. По умолчанию Application Manager создает только JobOptionsSvc и MessageSvc.

Обращение к сервису должно осуществляться через его интерфейс. Для того чтобы какой-либо компонент имел доступ к определенному сервису, компонент нужно снабдить ссылкой или указателем на этот сервис. Для этого внутри фреймворка существует функция serviceLocator.

Помимо создания алгоритмов, Gaudi также позволяет пользователю создавать собственные сервисы. Для этого необходимо предоставить интерфейс нового сервиса, также новый сервис должен быть наследником базового класса Sevice.

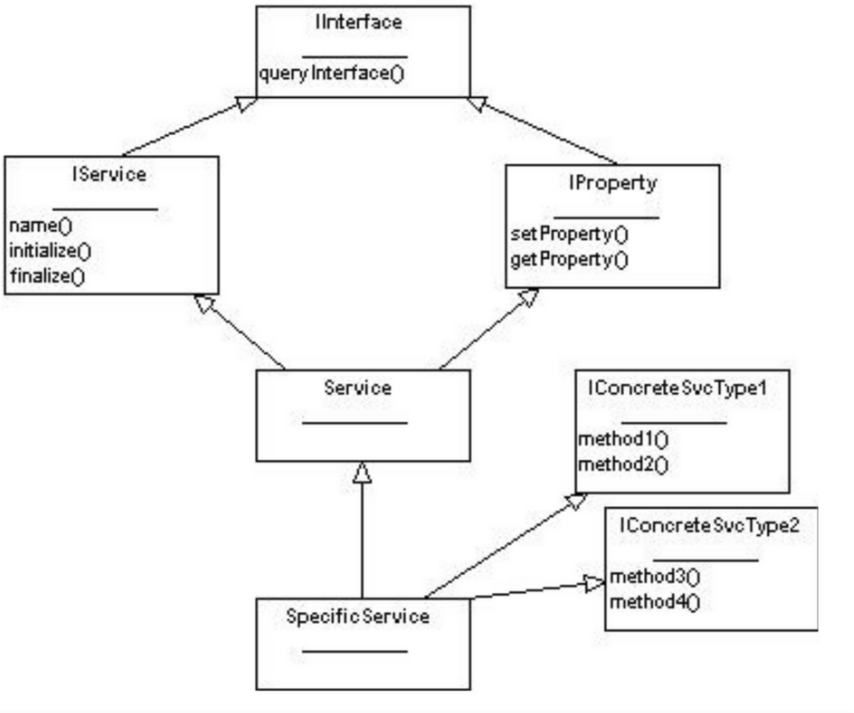


Рисунок 3.3 — Схема создания пользовательского сервиса.

3.1.3 ИНСТРУМЕНТЫ (TOOLS)

Алгоритмы могут использовать сервисы в ходе своей работы. При этом сервис создается в единичном экземпляре и может использоваться сразу несколькими алгоритмами или другими сервисами. Это неизбежно ведет к отсутствию возможности приватной настройки под каждого кон-

кретного клиента. Также иногда хочется иметь возможность конфигурировать код, исполняемый алгоритмом, динамически. Логично выделить такой конфигурируемый код в отдельную сущность. Тогда эту сущность также можно будет сделать динамически конфигурируемой и снабжать каждого клиента своим собственным экземпляром. Такой сущностью в Gaudi являются *инструменты (Tools)*.

Если приватное конфигурирование инструмента не является необходимым и одним и тем же экземпляром могут пользоваться несколько алгоритмов или сервисов, то такой инструмент является *разделяемым*. В противном случае он является *приватным*. Различить, является инструмент приватным или разделяемым, можно по типу класса-родителя для данного инструмента: у разделяемых родителем является сервис ToolSvc, у приватных - конкретные алгоритмы либо же другие сервисы.

Сервис ToolSvc отвечает за создание инструментов и управление ими. Алгоритм запрашивает необходимые ему инструмент у ToolSvc, указывая, запрашивает ли он приватный экземпляр, путем объявления себя родителем для запрашиваемого инструмента. Алгоритмы, сервисы и другие инструменты могут объявлять себя родителями.

Базовым классом для создания собственных инструментов является AlgTool. При этом у разрабатываемого инструмента должен быть также и пользовательский интерфейс с предоставляемым им функционалом. Иерархия классов для разработки инструментов представлена на рисунке 3.4.

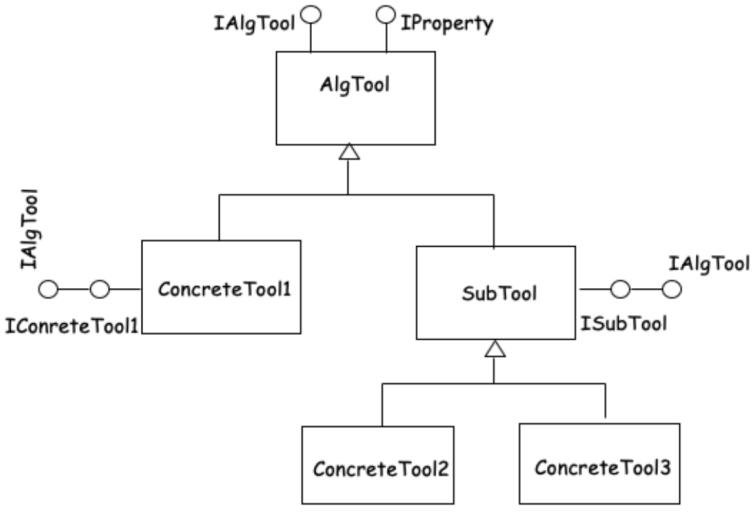


Рисунок 3.4 — Иерархия классов для разработки инструмента.

ToolSvc управляет инструментами. Он проверяет, доступен ли данный инструмент, и создает необходимый экземпляр после проверки, если он еще не существует. Если экземпляр инструмента существует, ToolSvc не будет создавать новый идентичный, а передаст алгоритму существующий экземпляр. Инструменты создаются по принципу “первого запроса”: первый алгоритм, запрашивающий инструмент, инициирует его создание. Взаимосвязь между алгоритмом, ToolSvc и инструментами показана на рисунке 3.5.

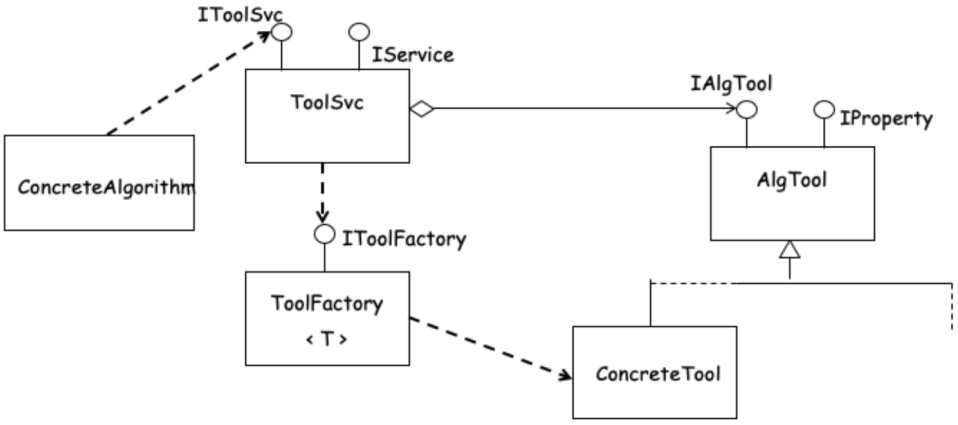


Рисунок 3.5 — Взаимодействие с ToolSvc.

3.2 КОНФИГУРИРОВАНИЕ ПРИЛОЖЕНИЯ. JOB OPOTIONS

Под понятием *Job* имеется в виду запуск программы в определенной конфигурации на определенных входных данных. Для того чтобы сконфигурировать *Job*, в Gaudi предусмотрен механизм *JobOptions* файлов, представляющих собой набор команд, интерпретируемых Gaudi. На языке этих команд описывается последовательность алгоритмов, их параметры, используемые сервисы, входные данные и многое другое.

Однако большинство современных экспериментов использует другой подход. Он подразумевает конфигурирование задач с помощью скриптов на языке *Python*.

Ранее было сказано, что компоненты в Gaudi являются динамически конфигурируемыми. Для этого они должны быть отнаследованы от интер-

фейса IProperty и базового класса PropertyHolder. Именно через задание Properties производится конфигурирование приложения.

3.3 КОНТРОЛЬ И ПЛАНИРОВАНИЕ

Работа приложения начинается с создания экземпляра ApplicationMgr. ApplicationMgr отвечает за создание и инициализацию минимального набора необходимых сервисов, прежде чем управление будет передано другим компонентам (см. рисунок 3.6). EventLoopMgr отвечает за управление циклом обработки событий и планирование алгоритмов. Существует также возможность предоставить полный контроль над приложением компоненту, реализующему интерфейс IRunable. Это необходимо для интерактивных приложений, таких как отображение событий, интерактивный анализ и т.д.

Основными сервисами, которые ApplicationMgr необходимо создать, являются MessageSvc и JobOptionsSvc.

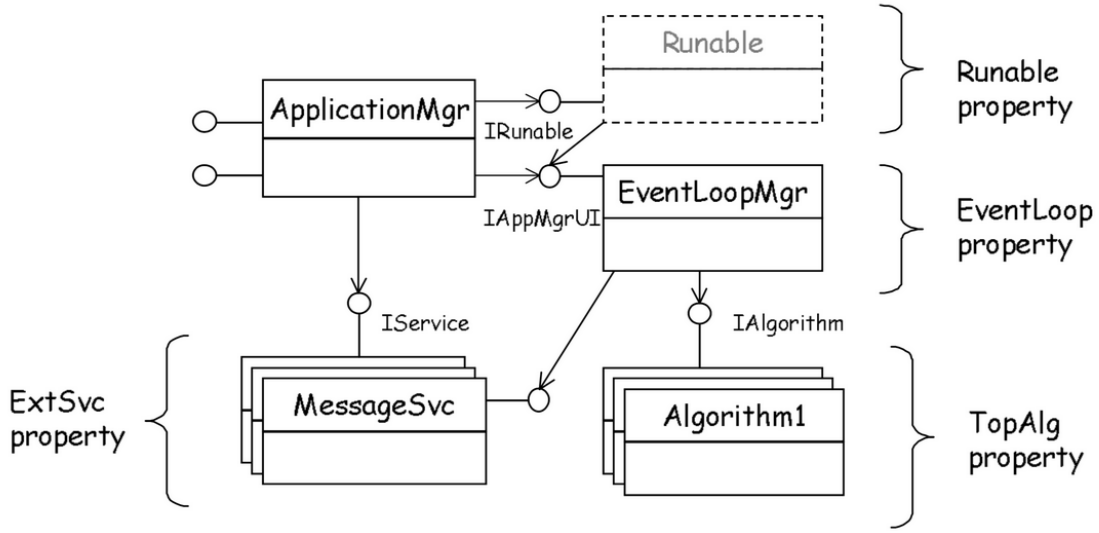


Рисунок 3.6 — Контроль и планирование в Gaudi.

3.3.1 РАБОТА GAUDI-ПРИЛОЖЕНИЯ

Общая схема работы приложения, написанного на базе Gaudi, представлена на рисунке 3.7:

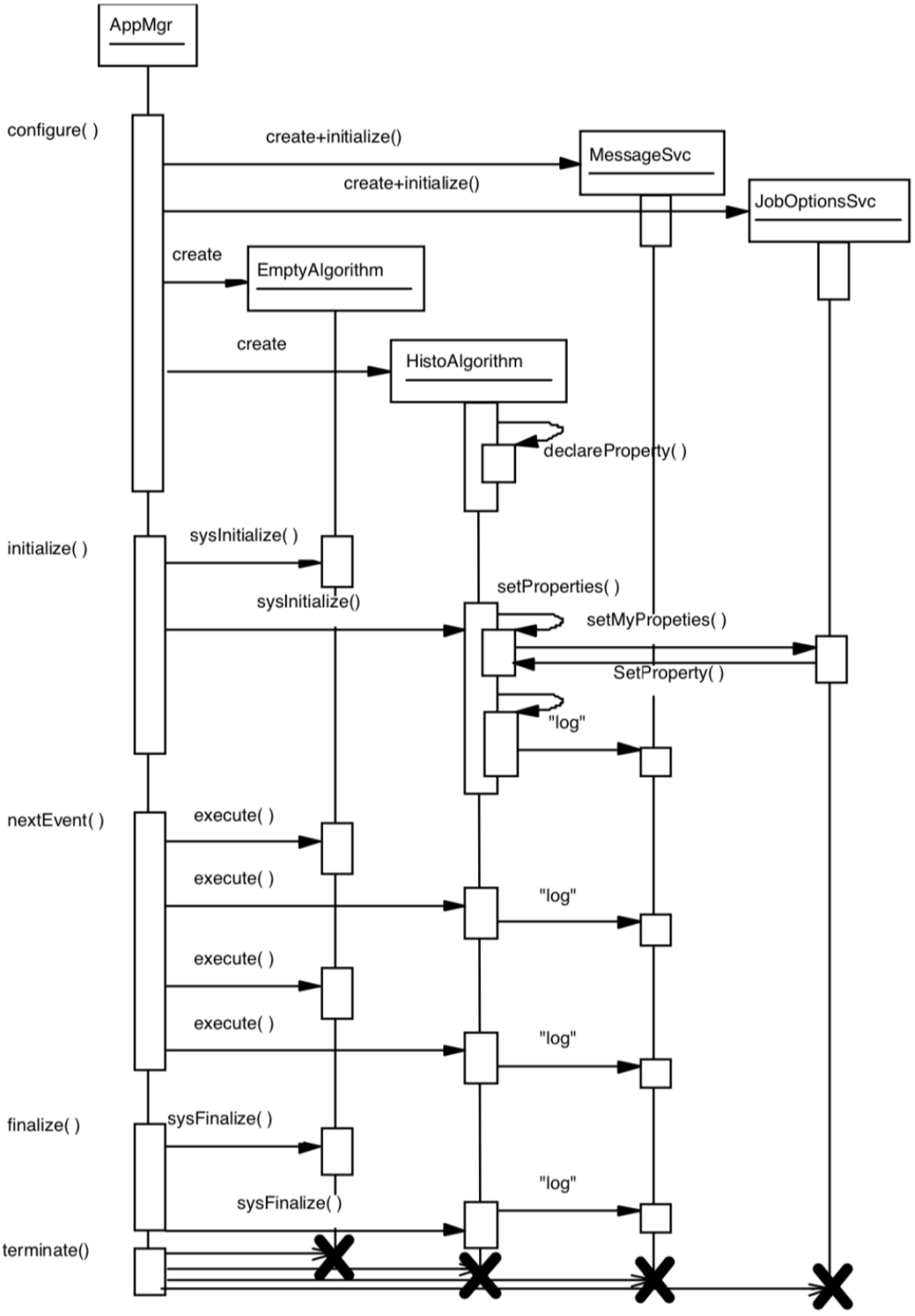


Рисунок 3.7 — Общая схема работы Gaudi-приложения

Порядок работы следующий:

- Application manager создает и инициализирует необходимые сервисы;

- создаются алгоритмы, указанные в JobOptions;
- устанавливаются свойства алгоритмов;
- Application manager начинает цикл обработки событий. Для каждого события вызываются алгоритмы в установленном порядке;
- по завершении цикла обработки событий алгоритмы завершаются;
- сервисы завершаются;
- освобождаются все ресурсы, программа завершается;

3.4 МНОГОПОТОЧНОСТЬ В GAUDI: GAUDIHIVE

Описанная выше архитектура и порядок работы приложения - однопоточные. GaudiHive [17],[18] является расширением Gaudi, предназначенным для повышения производительности путем использования многопоточности.

В своей основе GaudiHive использует концепцию параллельной обработки задач (task parallelism, не путать task и job). Здесь задачей является выполнение алгоритма в контексте обработки какого-либо события. Зависимости этих задач друг от друга могут быть выражены в виде направленного ациклического графа (DAG), сформированного исходя из ввода-вывода алгоритмов. Пример такой зависимости представлен на рисунке 3.8.

GaudiHive планирует запуск задач исходя из доступности данных и ресурсов (т.е. свободных экземпляров алгоритмов для обработки). На практике это достигается следующим образом. Центральные элементы показаны на рисунке 3.9 и включают в себя специальный параллельный планировщик (Scheduler) и потокобезопасное хранилище (Whiteboard). Все алгоритмы должны объявлять свои входные и выходные данные на этапе инициализации. Если нет входных или выходных данных, то ничего не нужно делать.

Whiteboard - это менеджер хранилищ, каждое из которых является аналогом TES из однопоточной версии и ассоциировано с конкретным обрабатываемым событием. Как только в Whiteboard появляются новые данные, планировщик проверяет, есть ли в пуле алгоритмов экземпляры, зависимые от появившихся данных. В пуле может быть более одного экзем-

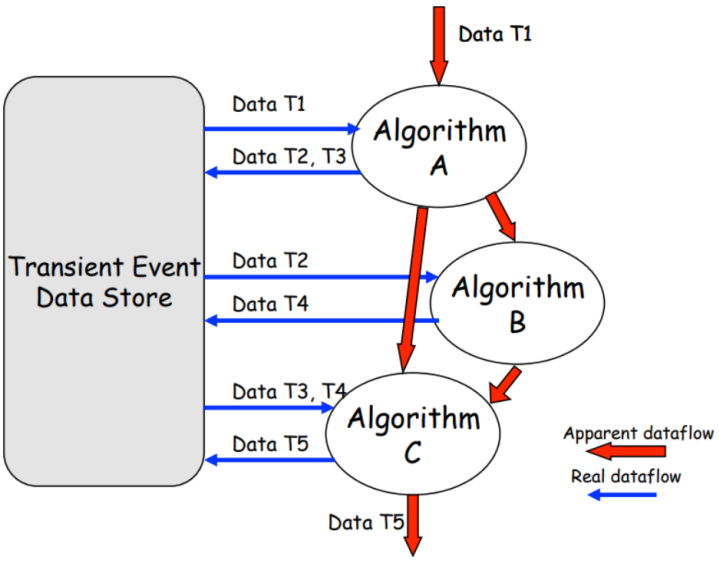


Рисунок 3.8 — Поток данных между алгоритмами, определяющий порядок выполнения задач.

пляра данного типа алгоритма. Затем из пула алгоритмов запрашиваются конкретный свободный экземпляр алгоритма, все входные зависимости которого имеются в наличии. Полученный экземпляр оформляется в задание и отправляется в пул потоков. Как только выполнение задания завершается, экземпляр снова передается в пул алгоритмов.

Таким образом, GaudiHive позволяет распараллеливать обработку событий, а также запускать несколько независимых алгоритмов параллельно в рамках обработки одного события.

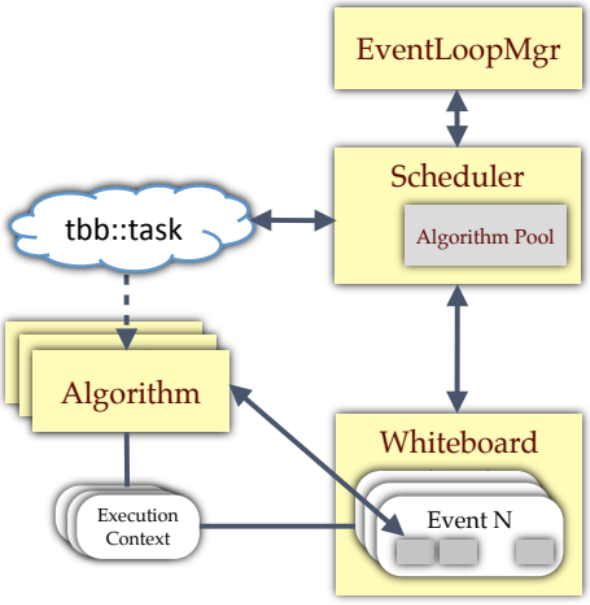


Рисунок 3.9 — Основные компоненты GaudiHive.

Событие считается обработанным, как только выполняются все вершины DAG (все алгоритмы). Но алгоритмы хранятся в пуле алгоритмов, они не связаны ни с каким конкретным событием. Для хранения информации о статусе обработки конкретного события используется концепция *Event Slots*.

Класс EventSlot можно рассматривать как словарь, ключами в котором являются алгоритмы, а значениями - их статусы выполнения. В разных слотах один и тот же алгоритм может быть помечен как имеющий разные состояния. Каждый слот ассоциирован с собственным хранилищем TES в Whiteboard. На рисунке 3.10 показан пример моментального снимка работы GaudiHive. Каждый цвет соответствует различному состоянию выполнения алгоритма.

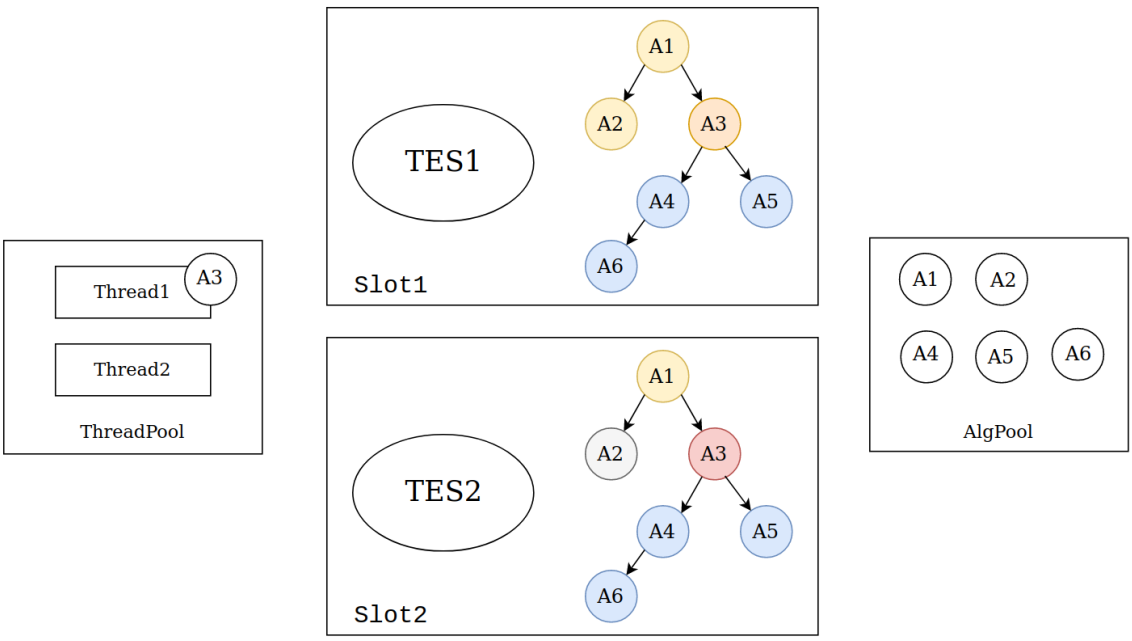


Рисунок 3.10 — Моментальный снимок GaudiHive в процессе работы.

3.5 ПРИМЕНЕНИЕ ПАТТЕРНОВ ООП В GAUDI

Объектно-ориентированный дизайн фреймворка Gaudi позволяет использовать паттерны проектирования для решения тех или иных технических задач. Паттерны проектирования - это типовые решения часто возникающих проблем при разработке программного обеспечения. Они похожи на готовые шаблоны, которые можно использовать для решения повторяющихся проблем проектирования. Выделяют три типа паттернов:

- **Порождающие паттерны** предоставляют механизмы для создания объектов, повышая тем самым гибкость и создавая возможность повторного использования существующего кода;
- **Структурные паттерны** используются для объединения нескольких классов в более крупные структуры, сохраняя при этом гибкость и эффективность;
- **Паттерны поведения** используются для обеспечения эффективной коммуникации и распределения обязанностей между объектами;

Важно понимать, что паттерн не является конкретным фрагментом кода, он представляет из себя общую концепцию для решения конкретной задачи. Фреймворки сильно влияют на выбор используемых паттернов проектирования. Понимая принципы устройства и соглашения внутри фреймворка, можно предвидеть, какие паттерны проектирования будут наиболее полезными и как они обычно применяются в рамках этого фреймворка. Далее приведены примеры паттернов, которые будут в дальнейшем часто использоваться для написания кода на базе фреймворка Gaudi.

3.5.1 ПАТТЕРН FACTORY METHOD

Factory Method — это паттерн проектирования, который предоставляет интерфейс для создания объектов в базовом классе, позволяя наследникам изменять тип создаваемых объектов. Схема паттерна приведена на рисунке 3.11.

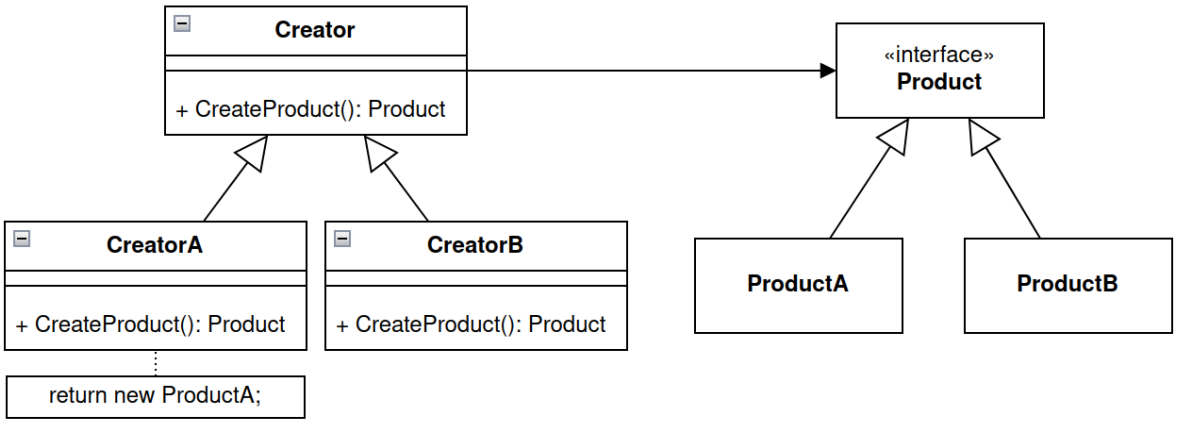


Рисунок 3.11 — Схема паттерна Factory Method

Factory Method отделяет код создания продукта от кода, который фактически использует продукт. Это полезно в случае, когда заранее не известно, с каким типом продукта в дальнейшем предстоит работать.

Пример использования Factory Method в Gaudi приведен на рисунке 3.12.

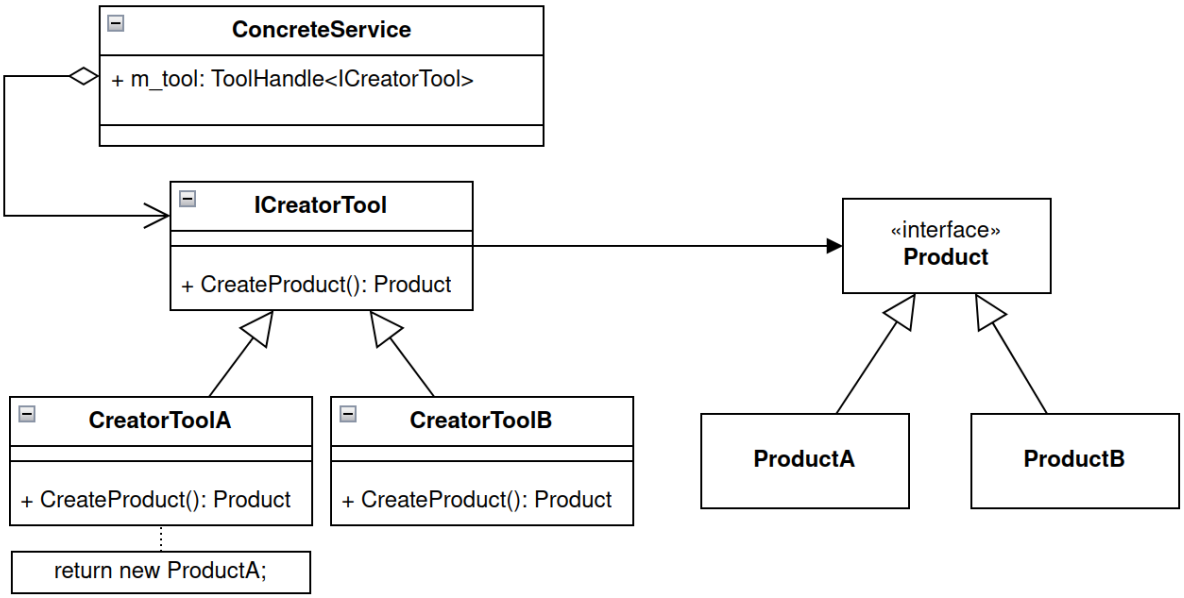


Рисунок 3.12 — Factory Method в Gaudi

Здесь подразумевается, что для работы ConcreteService нужна одна из имплементаций Product. Для создания Product используется ICreatorTool. Выбор конкретной реализации ICreatorTool, умеющей создавать конкретные типы Product, осуществляется в JobOptions.

3.5.2 ПАТТЕРН BRIDGE

Bridge — это структурный паттерн проектирования, который позволяет разделить большой класс или набор тесно связанных классов на две отдельные иерархии — абстракцию и имплементацию, которые могут разрабатываться независимо друг от друга. В этой терминологии абстракция — это высокоуровневый компонент управления некоторой сущностью. Предполагается, что этот уровень не выполняет никакой реальной работы сам по себе. Он должен делегировать работу уровню имплементации. Структура паттерна Bridge представлена на рисунке 3.13.

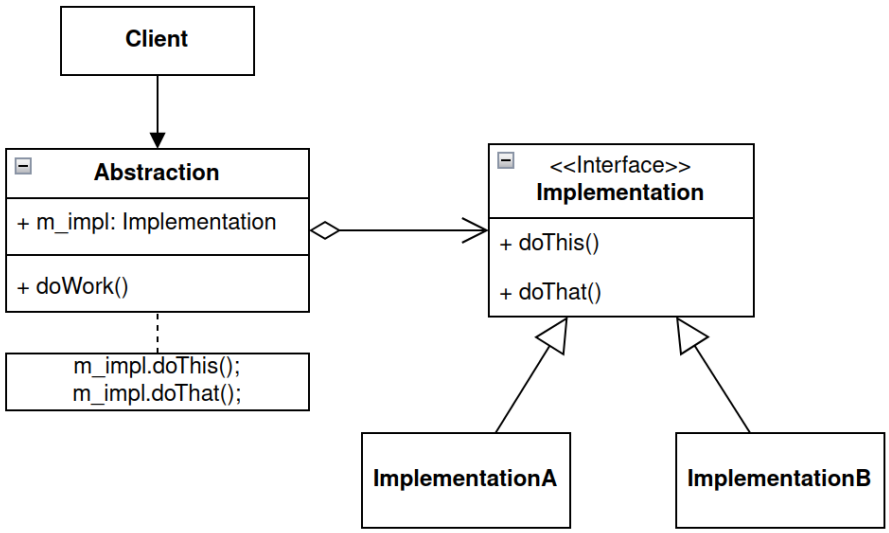


Рисунок 3.13 — Схема паттерна Bridge

Паттерн Bridge позволяет разделить монолитный класс на несколько (в идеале — ортогональных) иерархий классов. После этого появляется возможность изменять классы в каждой иерархии независимо, что позволяет легко масштабироваться и изменять поведение на уровне абстракции без риска возникновения ошибок, свойственных монолитным классам.

Пример использования Bridge в Gaudi представлен на рисунке 3.14. Алгоритм ConcreteAlgorithm в ходе своей работы обращается к серивису ConcreteService. ConcreteService, являясь высокоуровневым компонентом, делегирует выполнение конкретных действий IEasyWorkerTool и IHardworkerTool. Конкретные ипмлементации IEasyWorkerTool и IHardworkerTool задаются пользователем в JobOptoins. Таким образом, работа ConcreteService полностью конфигурируется через JobOptoins.

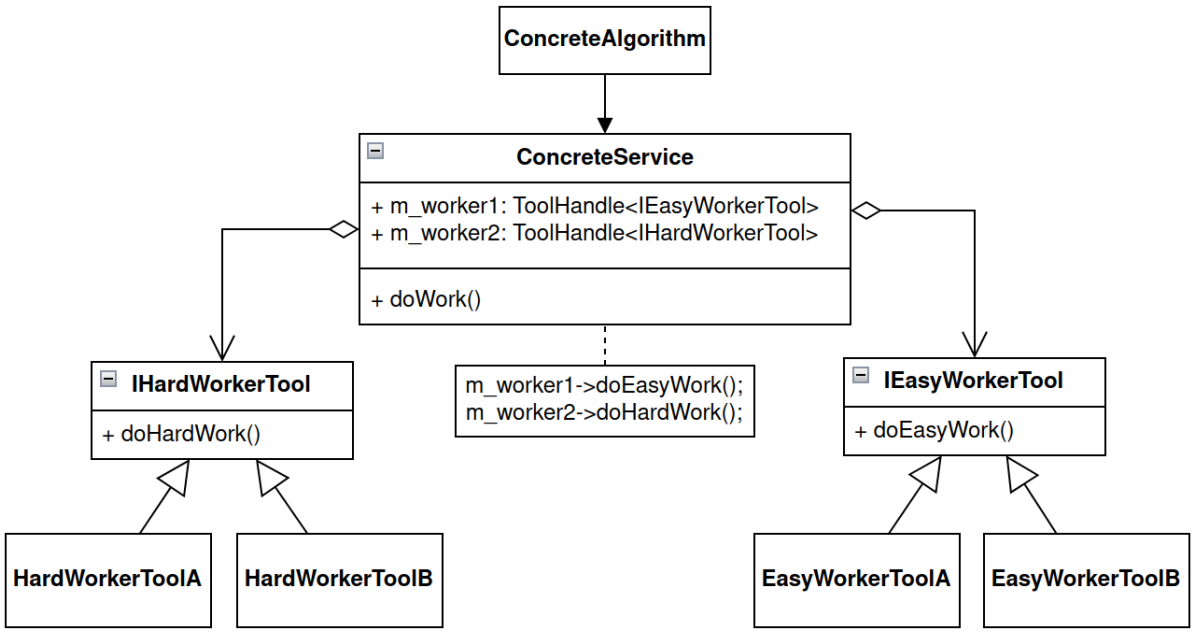


Рисунок 3.14 — Bridge в Gaudi

3.5.3 ПАТТЕРН ADAPTER

Adapter - это структурный паттерн проектирования, который позволяет взаимодействовать объектам с несовместимыми интерфейсами. Схема паттерна приведена на рисунке 3.15.

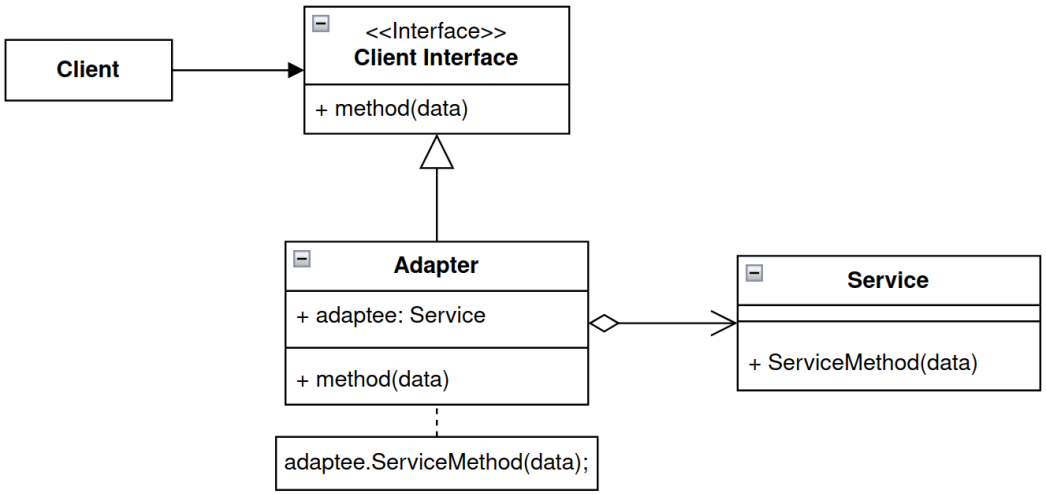


Рисунок 3.15 — Схема паттерна Adapter

Паттерн Adapter позволяет взаимодействовать с несовместным по интерфейсу объектом путем создания промежуточного слоя. Объект этого промежуточного слоя является совместным по интерфейсу с клиентом,

однако все клиентские вызовы он перенаправляет в инкапсулированный внутри себя несовместный по интерфейсу объект. Пример использования Adapter в Gaudi представлен на рисунке 3.16.

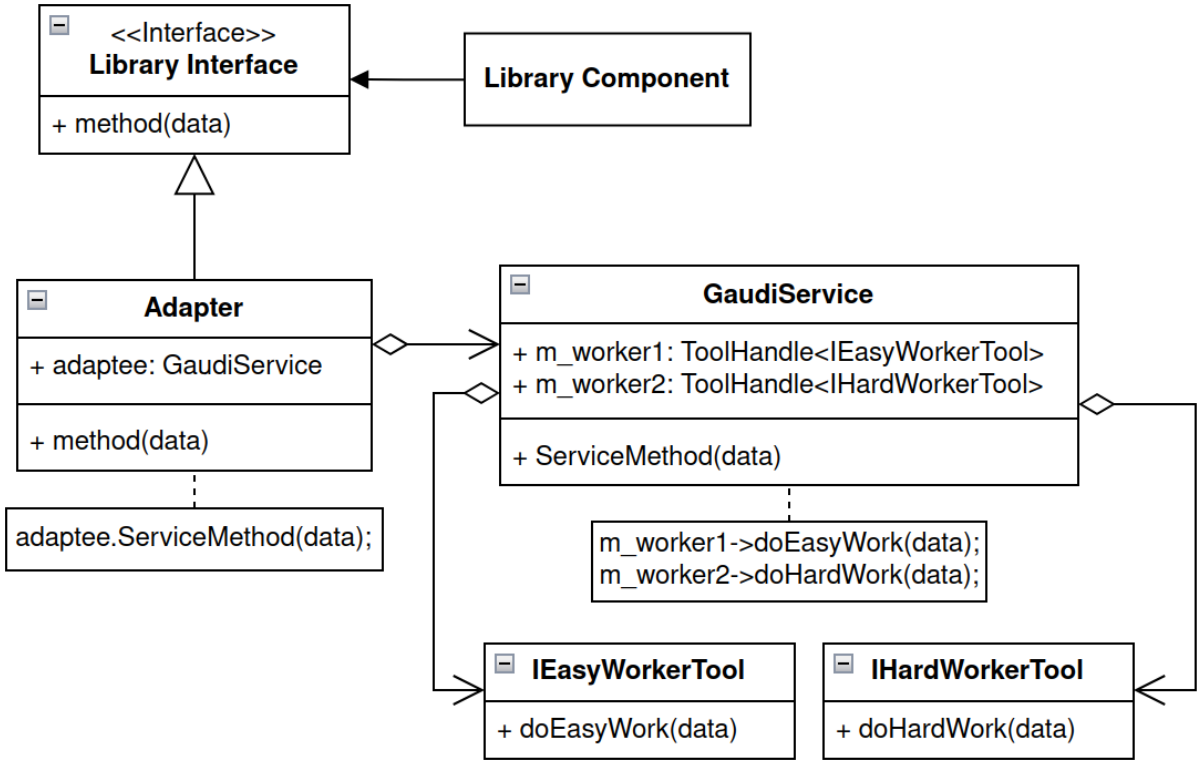


Рисунок 3.16 — Adapter в Gaudi

При использовании сторонних библиотек часто для их работы требуется задать некоторое определенное пользователем поведение. Часто это делается путем наследования от предоставляемого библиотекой Library Interface и имплементацией соответствующих методов. При этом, задаваемое поведение должно быть динамически конфигурируемым. Как было показано выше, очень удобным способом для динамического конфигурирования поведения в Gaudi является использование паттерна Bridge и механизма JobOptions. Используя паттерн Adapter, можно обернуть динамически конфигурируемый сервис из Gaudi в интерфейс, поставляемый библиотекой. Таким образом, все вызовы из библиотеки будут перенаправляться в сервис, функционал которого полностью контролируется пользователем.

4 GAUDI В SPD

В данной главе будет рассмотрена интеграция библиотек стека ПО в физике высоких энергий, описанных в главе 2, в инфраструктуру фреймворка Gaudi. Для выполнения этой интеграции активно использовались принципы, описанные в секции 3.5. Также будет рассмотрен механизм хранения и расчета значений магнитного поля в детекторе.

4.1 ИНТЕГРАЦИЯ PYTHIA8

Генераторы событий, в соответствии с ходом их работы, в контексте Gaudi логично представить в виде алгоритмов. В таком случае генерация одного события будет соответствовать вызову метода *execute()* соответствующего генератору алгоритма. Общая схема организации классов для интеграции Pythia8 представлена на рисунке 4.1.

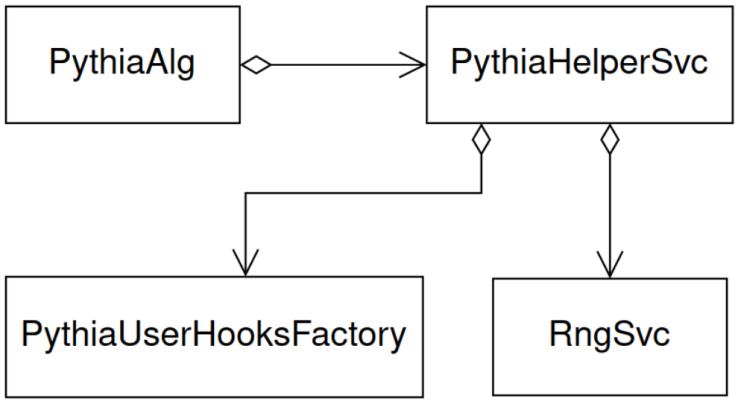


Рисунок 4.1 — Схема интеграции Pythia8.

Такой дизайн позволяет использовать многопоточность по аналогии с тем, как это было сделано в классе *PythiaParallel*, поставляемым разработчиками библиотеки Pythia8. В *PythiaParallel* создается пул потоков, каждому из которых соответствует собственный объект класса Pythia8. Эти объекты создаются путем клонирования вспомогательного объекта

pythiaHelper класса Pythia8, который сам в генерации участия не принимает, а служит лишь для сбора статистики с прочих экземпляров, для которых он выступает в роли прототипа.

В нашем случае такой прототип хранится в сервисе PythiaHelperSvc. Настройка этого прототипа производится через задание свойств самого сервиса в JobOptions. PythiaHelperSvc выступает в роли фабрики для создания экземпляров Pythia8 на основе хранимого им прототипа. Указатели на все созданные экземпляры сохраняются в сервисе для сбора статистики, сервис же и отвечает за их уничтожение. Алгоритмы класса PythiaAlg являются клиентами PythiaHelperSvc. В методе *initialize()* они запрашивают экземпляр Pythia8.

Если Pythia8 используется совместно с другими библиотеками, то необходимо, чтобы во всех объектах, так или иначе ответственных за генерацию чего-либо, использовались одни и те же генераторы случайных чисел. Для этого вводится сервис RngSvc, выступающий в роли фабрики генераторов случайных чисел. Создаваемые в PythiaHelperSvc объекты Pythia8 снабжаются такими генераторами.

Также библиотека Pythia8 предоставляет пользователю возможность изменять стандартное поведение Pythia8 через механизм хуков. Созданный хук должен быть наследником класса Pythia8::UserHooks. Для динамического создания хуков используется класс PythiaUserHooksFactory, в котором зарегистрированы все возможные для создания виды хуков. Набор желаемых к использованию хуков передается в PythiaHelperSvc через JobOptions.

В разработанном дизайне есть одно ключевое отличие от *PythiaParallel*: в случае GaudiHive невозможно заранее знать, в какой поток попадет данный генератор, следовательно, невозможно назначить ему строго определенное число событий, которое он должен сгенерировать. Это ведет к потере детерминированности результата, что, скорее всего, не является желательным сторонним эффектом. Упрощенный пример потери детерминированности для случая двух потоков приведен на рисунке 4.2.

Устранить возникшую неопределенность возможно, если для каждого генерируемого события будет строго определенный seed для генератора. Рассчитывать этот seed можно на основе номера события и общего для

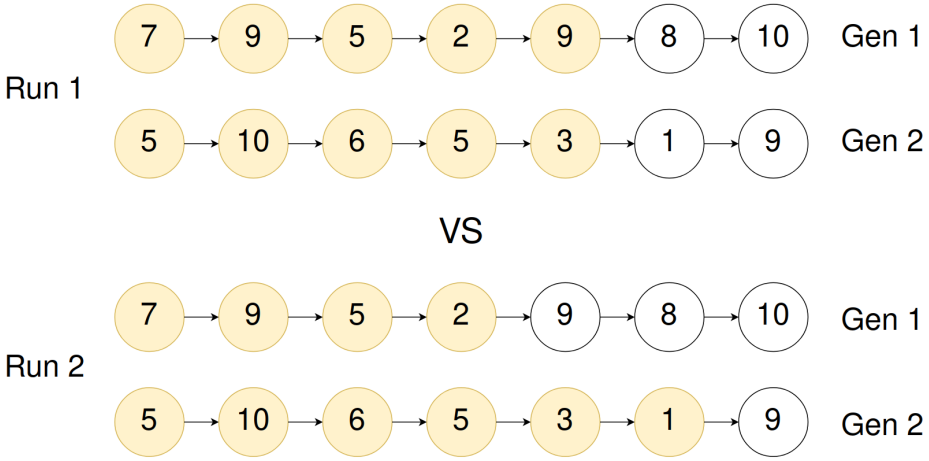


Рисунок 4.2 — Потеря детерминированности при многопоточной генерации за счет того, что заранее неизвестно, сколько раз каждый генератор будет вызван. В Run1 и Run2 получаются разные наборы чисел.

данного запуска seed, передаваемого через JobOptions.

4.2 ИНТЕГРАЦИЯ HEPMC3

Библиотека HepMC3 поставляется с удобными интерфейсами HepMC3::Reader и HepMC3::Writer, конкретные имплементации которых предназначенных для чтения и записи в различные форматы файлов. В рамках Gaudi на базе этих интерфейсов были разработаны сервисы HepMCReaderSvc и HepMCWriterSvc, схема приведена на рисунке 4.3.

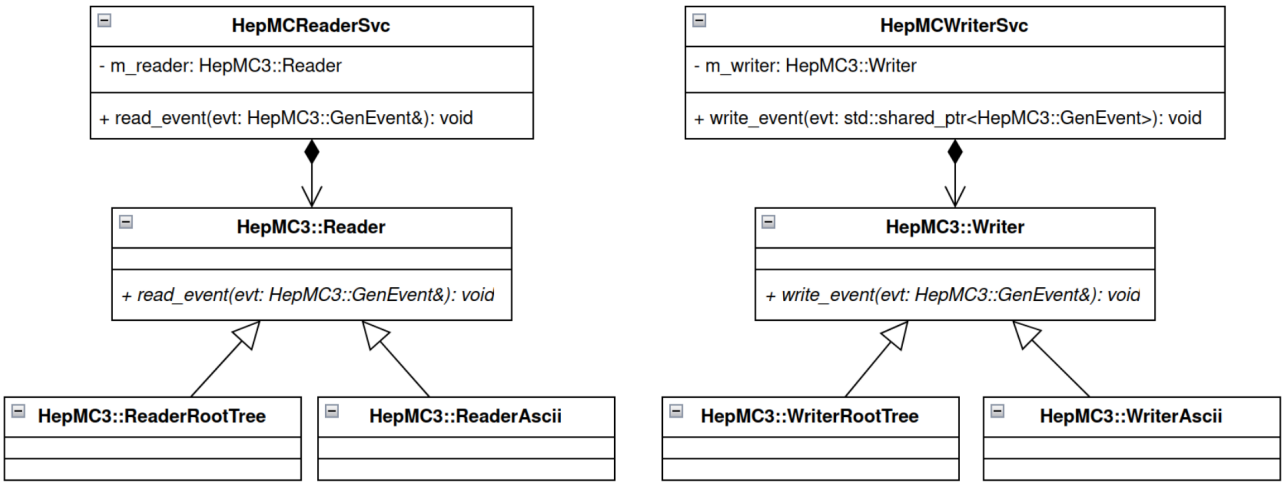


Рисунок 4.3 — Схема интеграции HepMC3

Конкретный тип HepMC3::Reader и HepMC3::Writer задается в JobOptions.

4.3 ИНТЕГРАЦИЯ GEOMODEL

Так как описание геометрии может потребоваться многим компонентам фреймворка, то внутри Gaudi GeoModel реализуется в виде сервиса. В рамках проделанной работы был разработан соответствующий сервис GeoModelSvc, основными задачами которого являются:

- Обеспечение подключения к базе данных с описанием геометрии, проверка корректности подключения;
- Выгрузка элементов геометрии из базы данных (по первому запросу), создание дерева объектов;
- Предоставление мирового объема по запросу;

GeoModelSvc представляет собой фасад, изолирующий классы GeoModel от пользователя. На данном этапе также можно предвидеть, что в дальнейшем (например, на этапе реконструкции) может потребоваться не все дерево объемов, а лишь какая-то конкретная подсистема детектора. С целью предоставления такого функционала был разработан соответствующий дизайн, представленный на рисунке 4.4. Инструменты, реализующие интерфейс IDetGeoTool, являются клиентами GeoModelSvc и на его основе добывают параметры подсистемы, к которой они относятся.

Также возможно на определенном этапе придется создать более абстрактную сущность для получения геометрии, не привязанную к конкретной библиотеке.

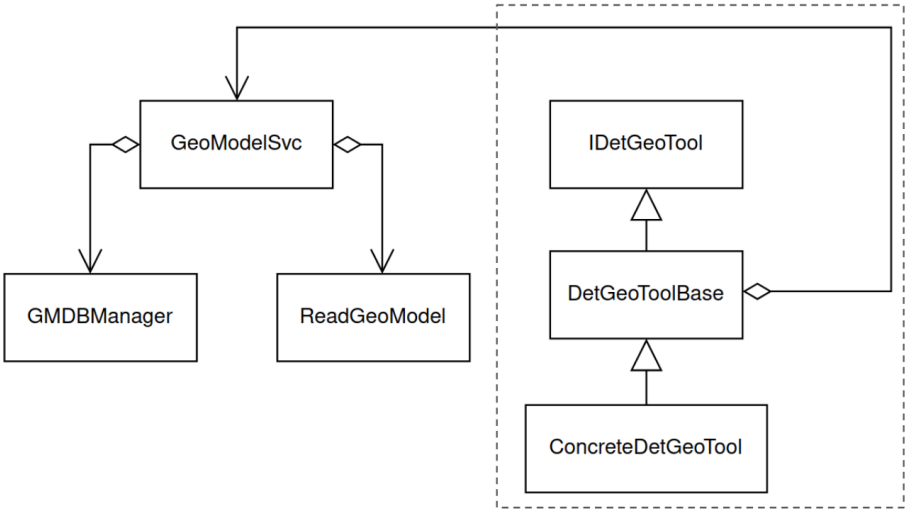


Рисунок 4.4 — Схема интеграции GeoModel.

4.4 МАГНИТНОЕ ПОЛЕ

Магнитное поле является неотъемлемым компонентом любого эксперимента в физике высоких энергий. Соответственно, как при моделировании, так и при реконструкции необходимо иметь возможность получить значение магнитного поля в заданной точке детектора. С этой целью был разработан сервис MagFieldSvc, схема которого представлена на рисунке 4.5.

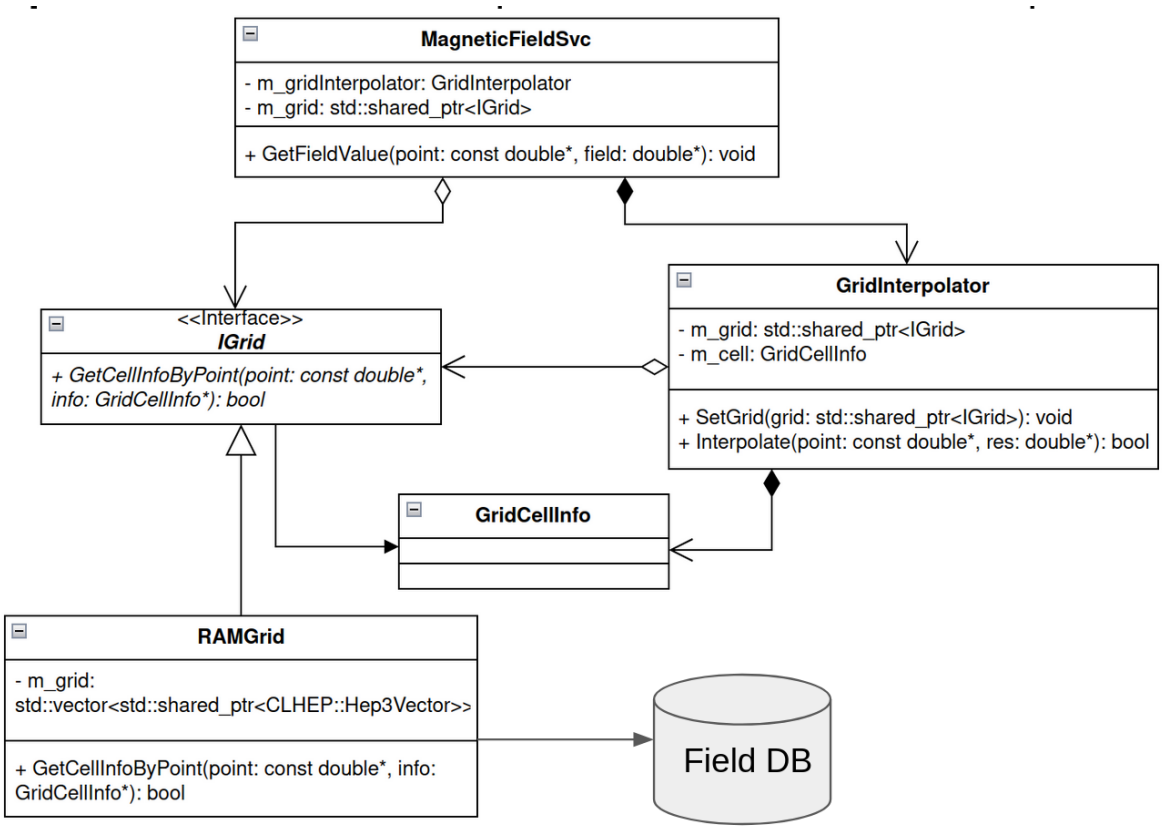


Рисунок 4.5 — Реализация сервиса магнитного поля в Gaudi

Так как поле в детекторе невозможно описать аналитической функцией, в эксперименте SPD для его описания используется так называемая карта поля, представляющая собой равномерную трехмерную сетку, в узлах которой промеряются реальные (или моделируются) значения магнитного поля. Для расчета значений поля не в узлах сетки используется трилинейная интерполяция. Карта хранится в виде SQLite базы данных.

Для проведения трилинейной интерполяции используется класс GridInterpolator, который необходимо снабдить сеткой, представленной классом, реализующим интерфейс IGrid. Возможны различные имплементации

IGrid, например, можно хранить соединение с базой данных с полем открытым и при необходимости оценки поля каждый раз делать запрос в базу для поиска соседних точек, используемых в интерполяции. Такая реализация занимает мало места в оперативной памяти, но тратит много процессорного времени на выполнение запросов, так как оценка магнитного поля производится многократно по ходу выполнения моделирования. Другая реализация, представленная классом RAMGrid, предполагает полную загрузку карты поля в оперативную память с последующим закрытием соединения с базой данных. Такая реализация работает существенно быстрее, но потребляет существенно больше оперативной памяти.

GridInterpolator и конкретный тип IGrid обмениваются GridCellInfo — классом, объекты которого хранят информацию о значениях поля в вершинах ячейки сетки, в которую попала точка, значение поля в которой необходимо рассчитать.

4.5 ИНТЕГРАЦИЯ GEANT4

Краткое описание работы пользователя с Geant4 приведено в секции 2.1. Для запуска моделирования в Geant4 пользователь должен предоставить собственные имплементации классов G4VUserDetectorConstruction, G4VUserActionInitializaiton и G4VUserPhysicsList. Для последнего в Geant4 предусмотрены дефолтные имплементации. В рамках Gaudi создание и конфигурирование этих объектов логично производить соответствующими сервисами, сам же запуск моделирования одного события осуществляется вызовом метода *execute()* соответствующего алгоритма.

Для создания G4VUserPhysicsList используется G4PhysListSvc, схема которого представлена на рисунке 4.6. Работу по созданию конкретного G4VUserPhysicsList сервис делегирует IG4PhysListTool, в то время как IG4PhysOptionTool используется для внесения дополнительных процессов или замены существующих на альтернативные в созданном G4VUserPhysicsList. Для генерации G4VUserPhysicsList, поставляемых самим пакетом Geant4, используется имплементация G4DefaultPhysListTool интерфейса IG4PhysListTool.

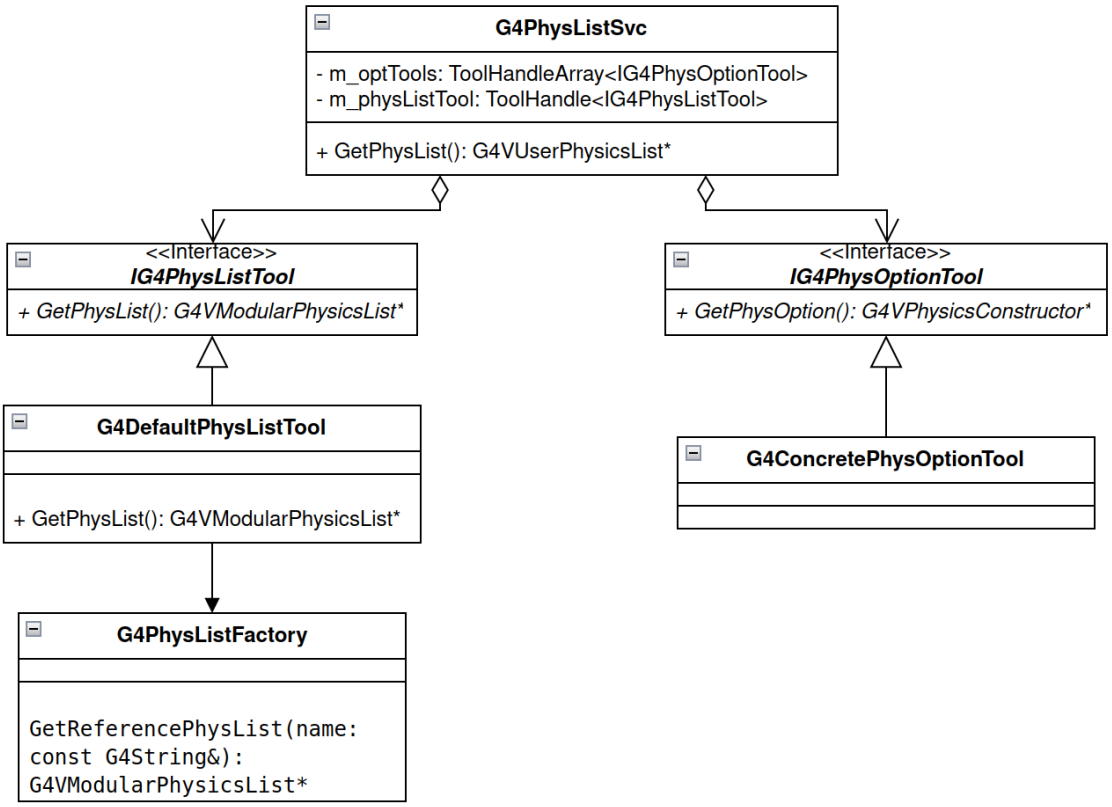


Рисунок 4.6 — Схема G4PhysListSvc

Для создания G4VUserDetectorConstruction используется G4DetectorConstructionSvc, схема которого представлена на рисунке 4.7.

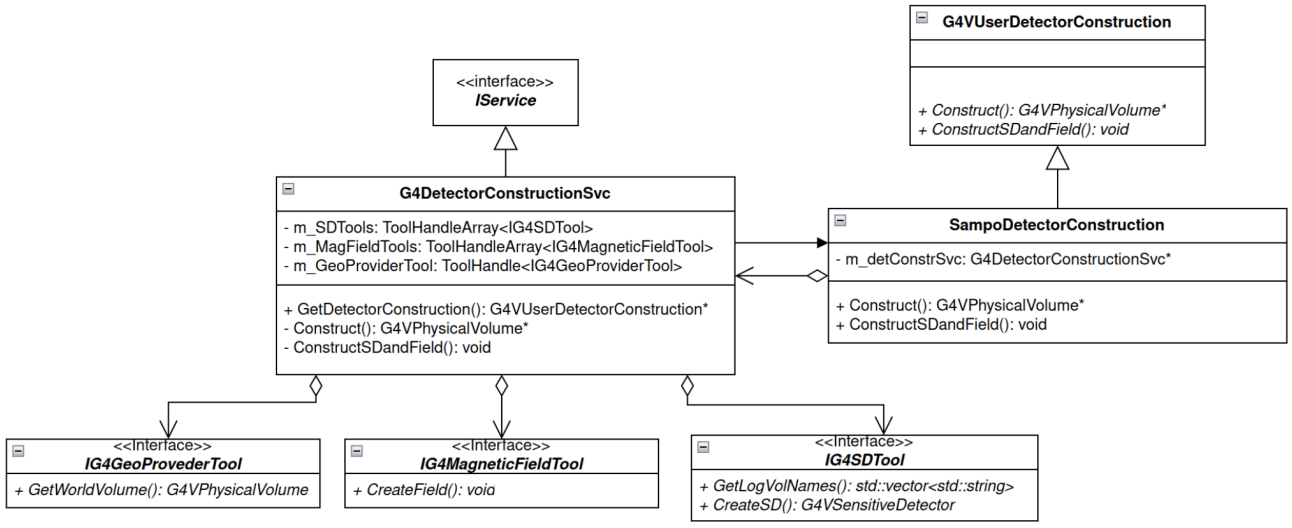


Рисунок 4.7 — Схема G4DetectorConstructionSvc

При запросе *G4DetectorConstructionSvc::GetDetectorConstruction()*, сервис заворачивает себя в класс-адаптер SampoDetectorConstruction, который все вызовы со стороны Geant4 перенаправляет в хранящийся внутри него сервис.

Кроме создания класса-адаптера, в задачи G4DetectorConstructionSvc входит также создание непосредственно геометрии установки, определение ее чувствительных элементов, а также создание магнитных полей. Вся эта работа делегируется IG4GeoProviderTool, IG4SDTool и IG4MagneticFieldTool соответственно.

Для описания геометрии установки используется рассмотренный ранее GeoModelSvc. Соответственно, чтобы перенаправить вызов на создание геометрии в GeoModelSvc, используется G4GeoModelTool, имплементирующий интерфейс IG4GeoProviderTool. Схема G4GeoModelTool представлена на рисунке 4.8.

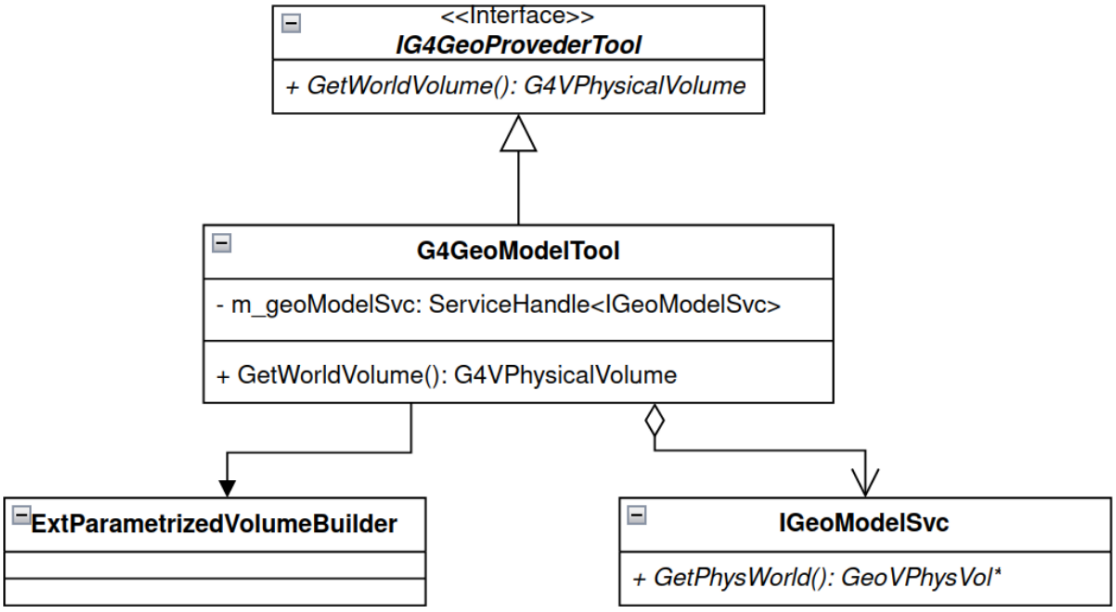


Рисунок 4.8 — Схема G4GeoModelTool

Здесь GeoModelSvc используется для создания геометрии в представлении GeoModel, а ExtParametrizedVolumeBuilder для ее конверсии в Geant4.

Для работы с магнитными полями в Geant4 используются объекты класса G4FieldManager. G4FieldManager назначается логическому объему, внутри которого нужно задать поле. Самому объекту G4FieldManager задаются драйвер для решения дифференциального уравнения движения и имплементация G4MagneticField, умеющая определять значение поля по заданной координате. В то время как назначение G4FieldManager в определенные логические объемы, а также выбор драйверов решения уравнения движения осуществляется в G4MagneticFieldToolBase, создание G4MagneticField должно быть переопределено в его наследнике. В нашем случае для работы с полями используется MagneticFieldSvc. Класс G4SampoMagneticFieldTool, являясь наследником G4MagneticFieldToolBase, создает G4MagneticField путем помещения MagneticFieldSvc в класс-адаптер SampoMagneticField. Схема отношений между вышеперечисленными классами представлена на рисунке 4.9.

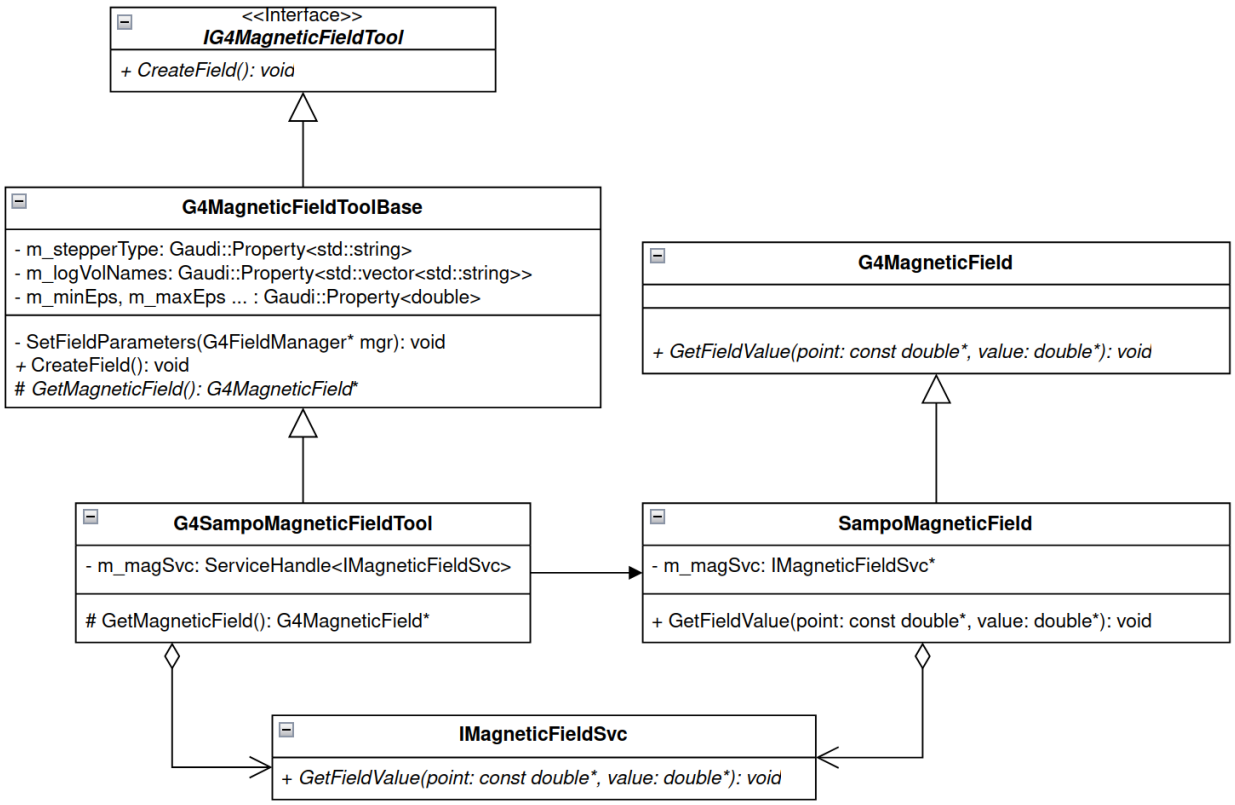


Рисунок 4.9 — Схема G4SampoMagneticFieldTool

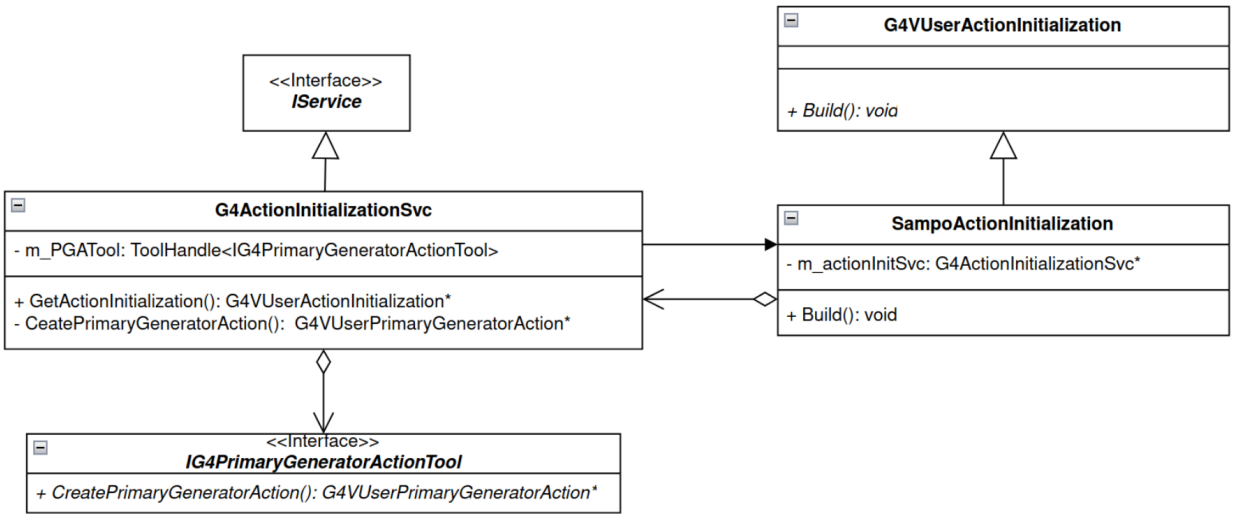


Рисунок 4.10 — Схема G4ActionInitializationSvc

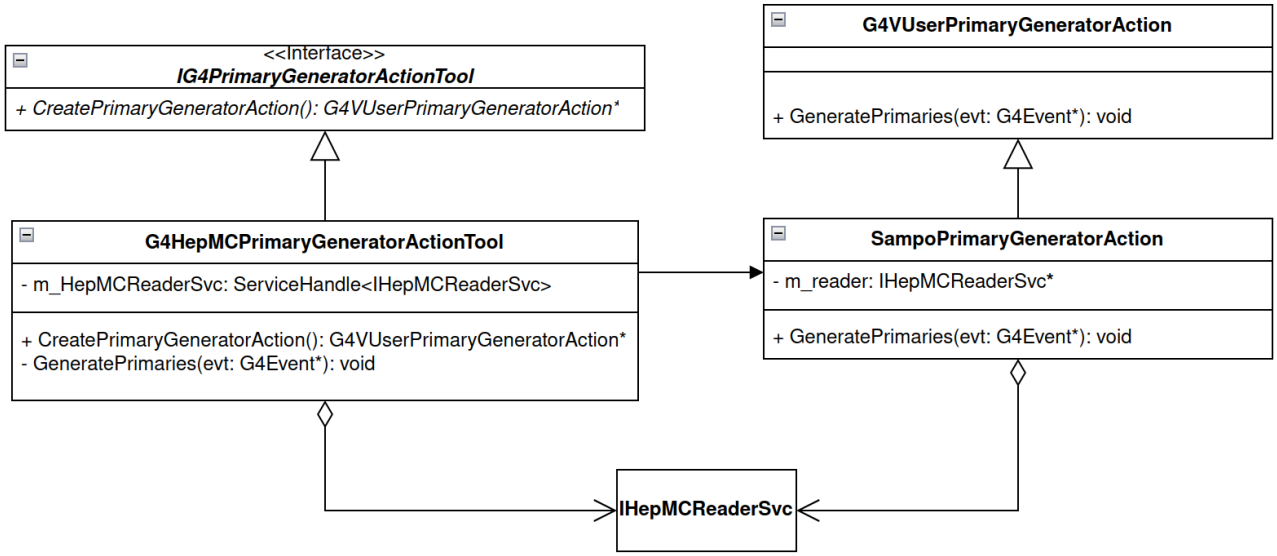


Рисунок 4.11 — Схема G4HepMCPrimaryGeneratorActionTool

ЗАКЛЮЧЕНИЕ

Данная работа посвящена разработке программной платформы для прикладного программного обеспечения эксперимента SPD на базе фреймворка Gaudi. Были рассмотрены основные компоненты и архитектура Gaudi, многопоточная архитектура GaudiHive, а также основные этапы жизненного цикла данных в физическом эксперименте. Каждый из этапов этого цикла требует использования специализированных физических библиотек, устройство которых также было рассмотрено, а сами библиотеки были интегрированы в общий фреймворк.

Результатами данной работы стали:

- интеграция генератора Pythia8 в инфраструктуру Gaudi в виде соответствующего набора из алгоритма, сервиса и фабрики пользовательских расширений. В дальнейшем, если потребуются другие генераторы, они могут быть интегрированы аналогичным образом;
- разработка сервисов чтения и записи на диск сгенерированных событий в формате HepMC3. Формат HepMC3 выбран как универсальный формат представления события;
- интеграция библиотеки GeoModel как универсального средства описания геометрических параметров установки в виде соответствующего сервиса;
- разработка сервиса для расчетазначения магнитного поля;
- интеграция пакета Geant4 как средства моделирования взаимодействия частиц с веществом детектора;

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Мегапроект NICA. — дата обр.: 04.05.2025. https://nica.jinr.ru/ru.
2. Technical Design Report of the Spin Physics Detector at NICA / V. Abazov [и др.] // Natural Sci. Rev. — 2024.
3. SPDRoot software. — дата обр.: 04.05.2025. https://git.jinr.ru/nica/spdroot.
4. The FairRoot framework / M. Al-Turany [и др.] // J. Phys. Conf. Ser. — 2012.
5. GAUDI — A software architecture and framework for building HEP data processing applications / G. Barrand [и др.] // Computer Physics Communication 2001.
6. Software and computing for Run 3 of the ATLAS experiment at the LHC / G. Aad [и др.] // Eur. Phys. J. C. — 2025.
7. Offline data processing system of the BESIII experiment / J. Zou [и др.] // The European Physical Journal C. — 2024.
8. LHCb Brunel software. — дата обр.: 10.05.2025. https://gitlab.cern.ch/lhcb/Brunel/.
9. *Sivers D. W.* Single Spin Production Asymmetries from the Hard Scattering of Point-Like Constituents // Phys. Rev. D. — 1990.
10. On the physics potential to study the gluon content of proton and deuteron at NICA SPD / A. Arbuzov [и др.] // Prog. Part. Nucl. Phys. — 2021.
11. ROOT package. — дата обр.: 10.05.2025. https://root.cern.ch/.
12. Geant4: Book for application Developers. — 2017. — дата обр.: 10.05.2025. https://indico.cern.ch/event/647154/contributions/2714212/attachments/1529029/2397032/BookForApplicationDevelopers.pdf.

13. The HepMC3 event record library for Monte Carlo event generators / A. Buckley [и др.] // Comput. Phys. Commun. — 2021.

14. A comprehensive guide to the physics and usage of PYTHIA 8.3 / C. Bierlich [и др.] // SciPost Phys. Codeb. — 2022.

15. The new GeoModel suite, a lightweight detector description and visualization toolkit for HEP / M. Bandieramonte [и др.] // J. Phys. Conf. Ser. — 2023.

16. AIDASoft/DD4hep / M. Frank [и др.]. — 2018. — дата обр.: 10.05.2025. `http://dd4hep.cern.ch/`.

17. *Hegner B.*, *Mato P.*, *Piparo D.* Evolving LHC data processing frameworks for efficient exploitation of new CPU architectures. — 2012.

18. Introducing concurrency in the Gaudi data processing framework / M. Clemencic [и др.] // Journal of Physics: Conference Series. — 2014.