Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой

_____ О. В. Непомнящий
подпись инициалы, фамилия
«____» _____ 2023 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Структуры данных учебной среды
объектно-ориентированного моделирования
тема

09.04.01 Информатика и вычислительная техника
код и наименование направления

09.04.01.04 Технология разработки программного обеспечения
код и наименование магистерской программы

Руководитель	_____	доцент, канд. техн. наук.	Н. Ю. Сиротинина
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	_____		Л. С. Артемьев
	подпись, дата		инициалы, фамилия
Рецензент	_____	доцент, канд. техн. наук.	А. А. Латынцев
	подпись, дата	должность, ученая степень	инициалы, фамилия
Консультант	_____	старший преподаватель	В. С. Васильев
	подпись, дата	должность, ученая степень	инициалы, фамилия
Нормоконтролер	_____		Н. Ю. Сиротинина
	подпись, дата		инициалы, фамилия

Красноярск 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В жизненном цикле программных продуктов с малой длительностью эксплуатации проектирование занимает до 50%, а с большой длительностью – до 20% времени разработки [1]. В ходе проектирования программного обеспечения определяются программные объекты, модули и их взаимосвязь в соответствии с выданными заказчиком требованиями (при выполнении учебных работ в роли заказчика выступает преподаватель). Проектирование является творческим процессом. Впрочем, существуют методологии структурирующие этот процесс, например RUP [2] и ICONIX [3].

Наиболее легковесным и подходящим для обучения является процесс проектирования ICONIX [4], поддерживающий широко применяемую нотацию UML и использующий сокращенный, но достаточный в подавляющем большинстве случаев набор диаграмм. Тем не менее, он обладает рядом недочетов, отмеченных в настоящей работе.

Для поддержки процесса ICONIX был разработан прототип учебной среды объектно-ориентированного моделирования [5], которая предоставляет пользователю возможность строить четыре вида диаграмм (прецедентов, пригодности, последовательности, классов) и выполнять ряд проверок корректности созданных моделей. Для разработки диаграмм используется инструмент PlantUML [6], позволяющий вводить описания диаграмм в текстовом виде с использованием определенной нотации и формирующий на выходе графические представления этих диаграмм. Однако используемые в этой системе текстовые или графические формы представления диаграмм не обеспечивают высокое быстродействие алгоритмов анализа.

Целью магистерской диссертации является разработка структур данных для моделей программ учебной среды объектно-ориентированного моделирования, обеспечивающих высокое быстродействие алгоритмов поиска ошибок проектирования.

Структура работы отражает решаемые **задачи**. Пояснительная записка состоит из 3 глав, содержит 3 таблицы, 16 рисунков, 5 приложений.

В первой главе описывается модернизированный процесс ICONIX, приводятся методы проверки корректности диаграмм, а также входные и выходные данные разрабатываемого модуля.

Во второй главе предлагаются способы представления диаграмм в памяти ЭВМ, выполняется анализ наиболее подходящих структур данных и приводится обоснование для выбранных.

В третьей главе выполняется реализация структур для представления диаграмм и модуля перевода документов PlantUML в эти структуры. Кроме того, код тестируется, определяется процент покрытия тестами, а также выполняется ряд проверок на качество кода.

1 Постановка задачи

В связи с тем, что создаваемые структуры данных должны обеспечить эффективность реализации алгоритмов анализа корректности диаграмм, то в ходе работы необходимо провести анализ их наиболее трудоемких операций. Задача осложняется тем, что в настоящее время известен лишь ряд методов проверки корректности моделей программ, часть из которых описана Дугом Розенбергом [3], а другие связаны с ограничениями, накладываемыми стандартом UML на правила исполнения диаграмм.

По результатам применения прототипа учебной среды объектно-ориентированного моделирования в учебном процессе стали очевидны недостатки канонического процесса ICONIX. Для их исправления предложен модернизированный процесс, который предполагает создание ряда дополнительных видов диаграмм, позволяющих применять к созданным моделям программного обеспечения новые методы поиска ошибок проектирования [7].

1.1 Модернизированный процесс ICONIX

Первым этапом канонического процесса проектирования ICONIX является – формализация требований к программному обеспечению в виде прецедентов [3], однако:

- для клиентских приложений текстовое описание прецедентов отражает работу пользователя в программе, то есть его взаимодействие с интерфейсом. На практике, представления об интерфейсе зачастую формируются параллельно с описанием прецедентов, поэтому целесообразно дополнить процесс проектирования этапом формирования макетов интерфейса. Макеты предложено загружать в систему в виде изображений, но помимо этого реализовать возможность создания карты диалоговых окон с использованием нотации диаграммы состояний UML [8];

- для серверных приложений спецификация требований, по сути, сводится к описанию реакции системы на входящие запросы, которые можно оформить в виде прецедентов. Вместо описания пользовательского интерфейса в этом случае имеет смысл говорить о формате данных запроса и ответа. Предлагается реализовать возможность ввода соответствующей информации в среду моделирования с использованием формата OpenAPI [9];

- форматы данных важны не только в серверных приложениях. Если приложение хранит информацию в файлах, то описание формата файлов должно являться элементом спецификации требований, решено реализовать поддержку описания формата в виде расширенной формы Бэкуса-Наура (РБНФ), а также схем документов (XML/JSON) [10]. Для визуализации отношений между сущностями предложено добавить возможность построения ER-диаграммы в нотации Мартина [11];

- параллельно с выявлением требований удобно формировать модель предметной области, при этом удобно использовать нотацию Мартина. Эта модель может использоваться при построении ER-модели базы данных и диаграммы классов.

На последующих этапах процесса проектирования, макеты интерфейса соответствуют граничным объектам, а форматы сообщений и файлов закрепляется за сущностями.

Последовательность разработки диаграмм, для модернизированного процессу моделирования приведена на рисунке 1.1.

Канонический процесс ICONIX предусматривает построение 4 видов диаграмм (прецедентов, пригодности, последовательности и классов), а его модернизированная версия дополнена:

- макетами интерфейса (изображениями) для каждого граничного объекта;

- диаграммой потока экранов (нотация диаграммы состояний UML), связывающей граничные объекты между собой;

- описанием формата данных или ER-диаграммы для каждой сущности.

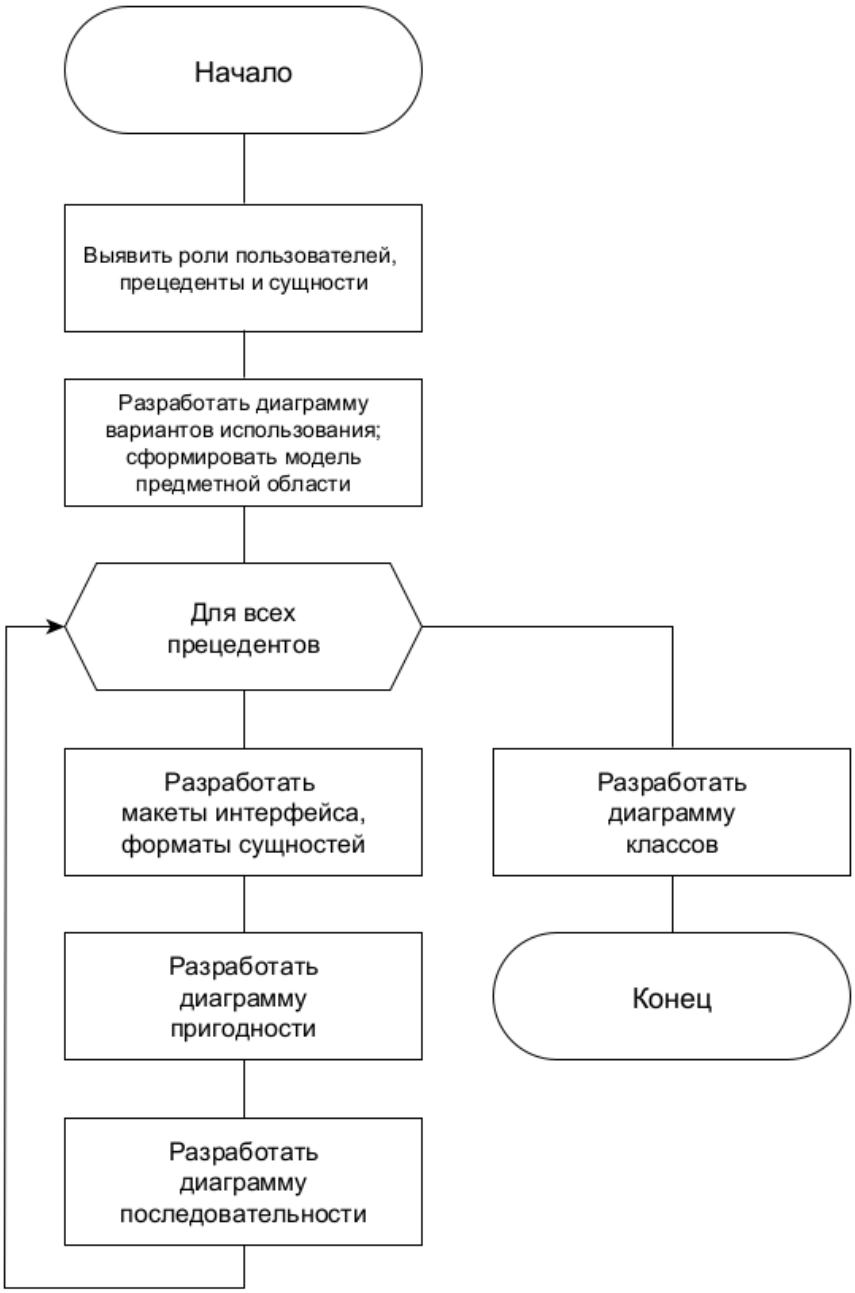


Рисунок 1.1 – Блок-схема алгоритма работы студента в соответствии с модернизированным процессом проектирования

1.2 Подмножество и ограничения UML в рамках процесса ICONIX

Каждый вид диаграммы имеет свои правила исполнения, определенные стандартом UML [12]. В работе [13] приведен обзор некоторых правил их проверки, однако такая комплексная проверка крайне трудна. Стандарт UML

очень сложен, в работе [14] отмечается, что для моделирования 80% случаев достаточно лишь 20% его возможностей. Особенно ярко это проявляется в небольших проектах, к которым относятся учебные работы студентов. В связи с этим, решено реализовать в учебной среде моделирования поддержку наиболее востребованного подмножества стандарта UML.

Обычно **диаграмма вариантов использования** содержит прецеденты и акторы (действующие лица, роли). Между прецедентами возможны четыре вида отношений: ассоциация, включение, расширение и обобщение. Акторы и прецеденты могут быть связаны только с помощью отношения ассоциации. Пример диаграммы вариантов использования представлен на рисунке 1.2.

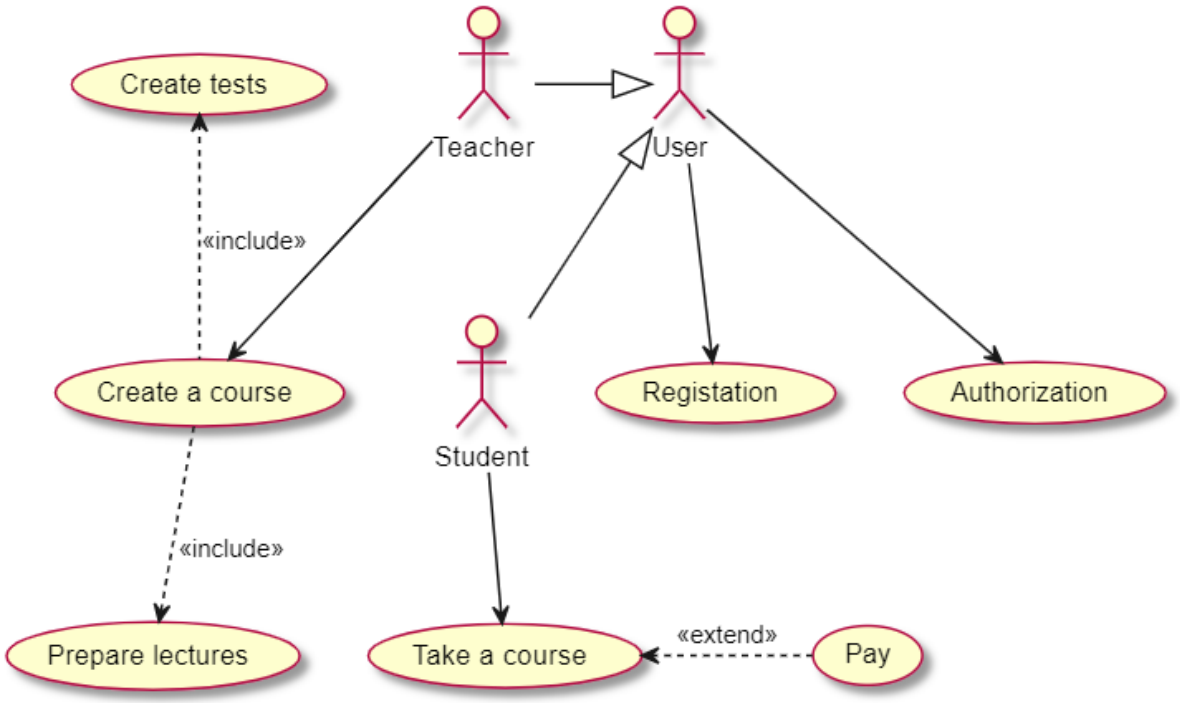


Рисунок 1.2 – Пример диаграммы вариантов использования

На **диаграмме пригодности** размещаются 4 типа узлов - акторы, граничные объекты, сущностные объекты и контроллеры. Узлы связываются лишь одним видом отношения – ассоциацией. Нередко дуги на диаграмме могут дополняться подписями и являться ориентированными. На рисунке 1.3 представлен пример диаграммы пригодности для прецедента «Подготовить лекционный материал».

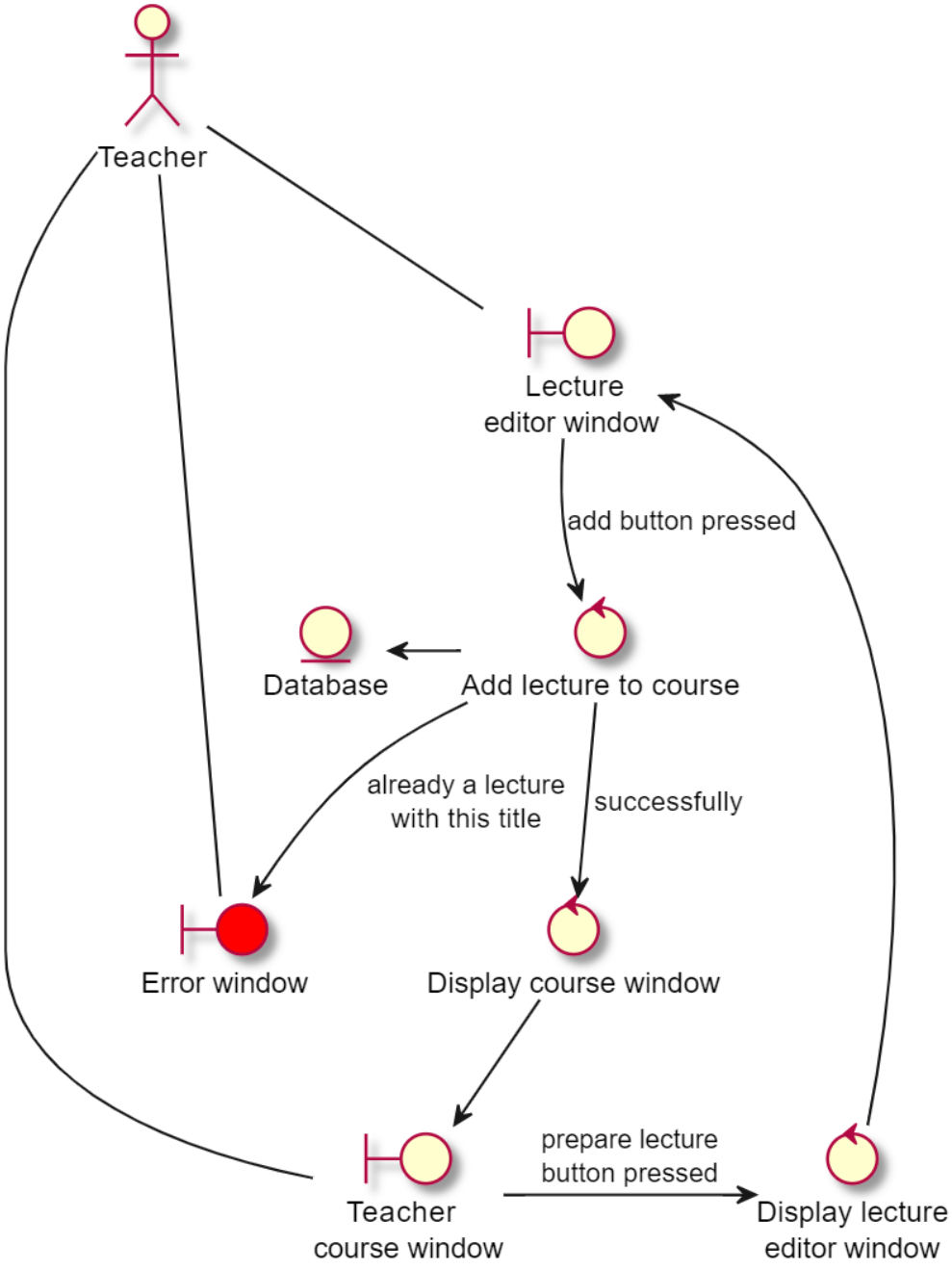


Рисунок 1.3 – Пример диаграммы пригодности

Диаграмма последовательности для каждого прецедента содержит объекты соответствующей диаграммы пригодности, однако контроллеры заменяются на «посылку сообщения». Сообщения упорядочиваются по времени, ось времени на диаграмме направлена сверху вниз. Посылка сообщения изображается сплошной линией, а полученный ответ – штриховой. Поверх оси времени каждого объекта могут отрисовываться вложенные друг в

друга прямоугольники, отражающие факт нахождения функции объекта в стеке. Поверх нескольких линий жизни объектов могут размещаться блоки alt, ref и loop, соответствующие ветвлению, предусловиям или ссылкам на другие прецеденты и циклам. Блок loop содержит описание условия окончания цикла в свободной форме, а блоки alt состоят из секций, каждая из которых дополняется условием входа. На рисунке 1.4 представлен пример диаграммы последовательности для прецедента «Подготовить лекционный материал».

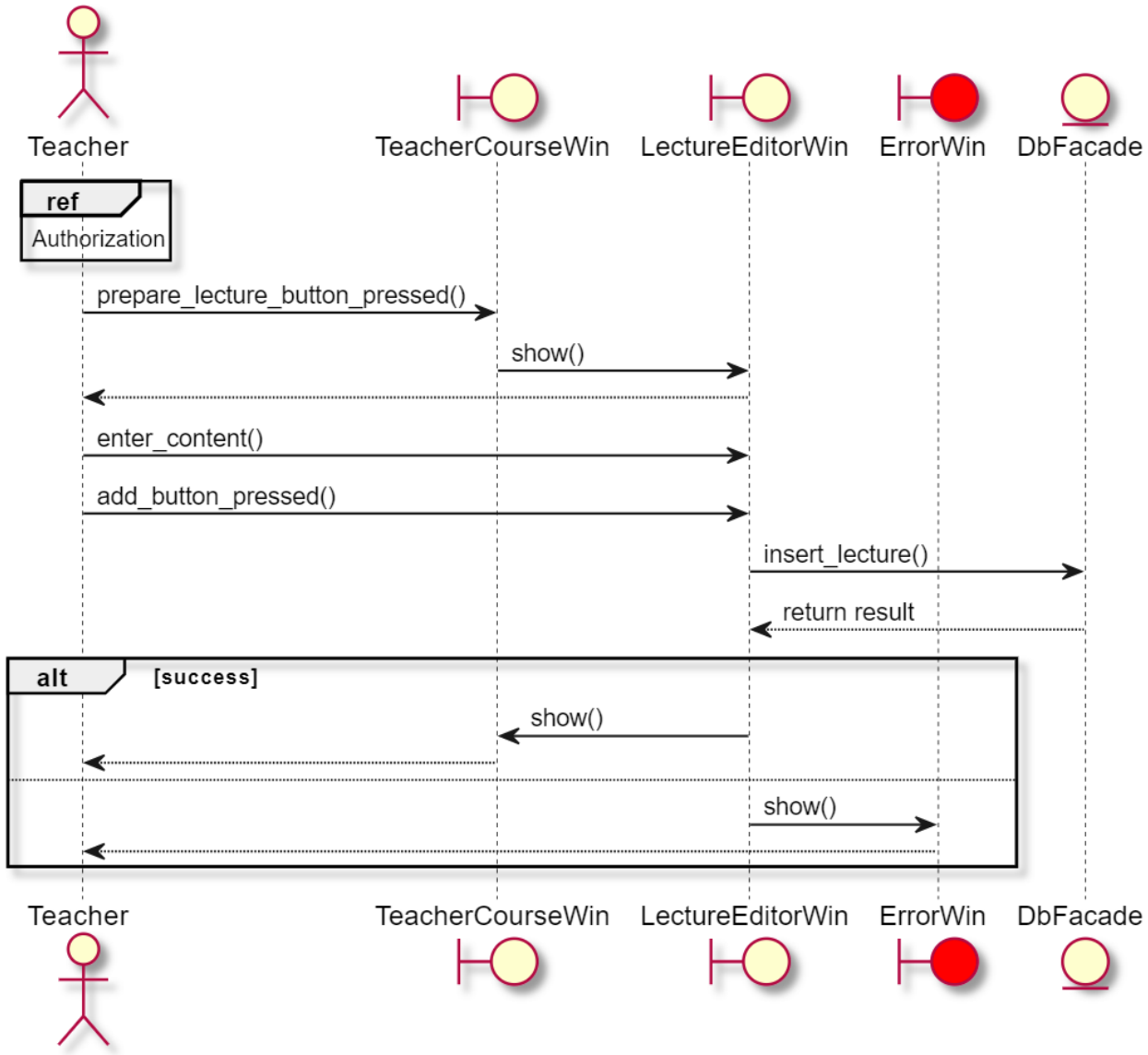


Рисунок 1.4 – Пример диаграммы последовательности

Статическая модель системы в процессе ICONIX отражается в виде **диаграммы классов**. Между классами допустимо использовать шесть видов

отношений: зависимость, ассоциация, агрегация, композиция (агрегация по значению), наследование и реализация (наследование интерфейсного класса). На этой диаграмме должны присутствовать только сами классы (и их наполнение: поля и методы), отношения между ними и перечисления. Члены класса имеют спецификаторы доступа. Пример диаграммы классов представлен на рисунке 1.5.

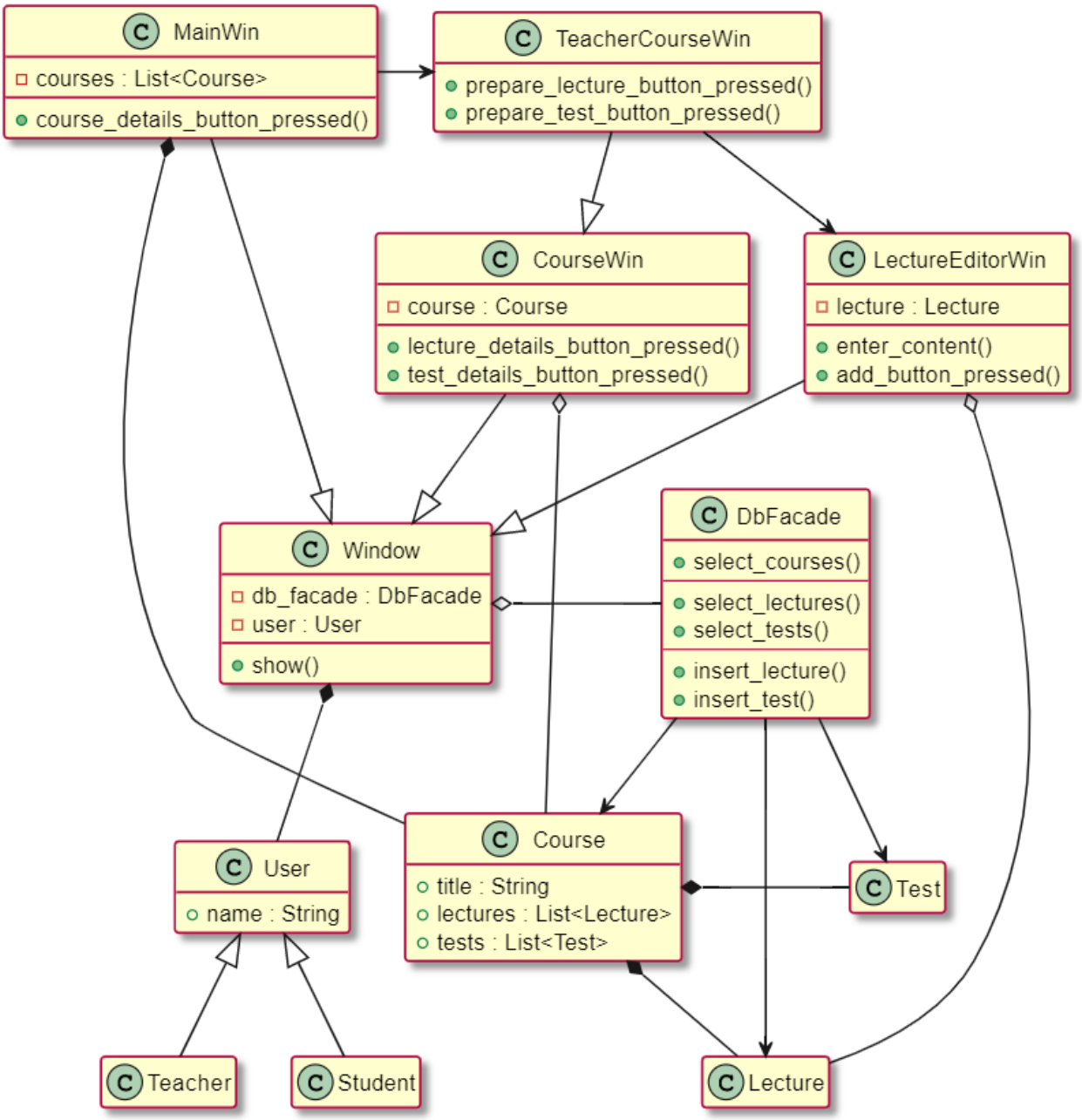


Рисунок 1.5 – Пример диаграммы классов

Описанные выше возможности задают подмножество языка UML, поддерживаемое средой моделирования. В частности, среда не поддерживает возможность вложения классов друг в друга, задания стереотипов классов, изображение шаблонов классов и так далее. Эти возможности сильно осложняют язык UML, за это он критикуется [14], поэтому нецелесообразно реализовывать их поддержку в учебной среде моделирования, нацеленной на проекты с невысокой сложностью.

1.3 Методы выявления ошибок проектирования

В рамках магистерской работы выполнен анализ возможности автоматизации методов проверки моделей канонического процесса ICONIX, предложенных в работах [3, 7]. В результате анализа выделены методы, которые возможно реализовать программно:

а) Проверка наличия для каждого прецедента ассоциированных граничных объектов и сущностей. При задании им характерных несклоняемых имен, например: User, StudentsDatabase, AuthForm, проверка соблюдения правила сводится к поиску в тексте прецедента соответствующих слов.

б) Проверка наличия альтернативных последовательностей для каждого прецедента. При их отсутствии предлагается выводить предупреждения, которые могут быть отключены в среде моделирования. Возможность явного задания и хранения основной и альтернативных последовательностей была реализована уже в прототипе учебной среды моделирования.

в) Проверка длины прецедентов (количество шагов). «Главная последовательность прецедента не должна быть длиннее одного-двух абзацев. Каждая альтернативная последовательность должна состоять из одного-двух предложений», «Прецедент, содержащий единственный контроллер скорее всего также не имеет смысла» [3]. Предлагается выводить предупреждение с предложением *разбить сложный прецедент на несколько более простых если число контроллеров оказалось более 7*. Связаны такие ограничения с

предельным объемом информации, которая может быть быстро усвоена человеком («число Миллера») [15].

г) Проверка соблюдения на диаграмме пригодности паттерна «граничный объект-контроллер-сущностный объект» сводится к проверке запрещенных связей на диаграмме: «граничный объект-сущность», «актор-контроллер», «актор-сущность». Также, стоит проверять что все граничные объекты диаграммы связаны с актором.

д) Предлагается выделять красным цветом элементы диаграмм пригодности и последовательности, имеющие отношение к альтернативным вариантам. Возможно проверять их наличие для прецедентов, имеющих хотя бы одну альтернативную последовательность и при отсутствии выдавать предупреждение.

е) Проверка соответствия диаграммы классов и диаграмм последовательности: «*Каждый объект диаграммы последовательности относится к какому либо классу, поэтому он должен быть представлен на соответствующей диаграмме*». Подписи на сообщениях диаграммы последовательности должны соответствовать функциям класса, в который входит стрелка.

ж) Проверка наличия диаграмм последовательностей для каждого прецедента и *альтернативных блоков в нем для каждого альтернативного варианта.* При обнаружении полей классов, не представленных в соответствующих описаниях форматов предлагается выдавать предупреждение, так как в ряде случаев в сообщениях передается служебная (вычисляемая) информация, не имеющая прямого отношения к сущности.

з) Значительная часть атрибутов классов должна использоваться при описании форматов данных/сообщений.

и) Проверка правил выполнения отдельных вида диаграмм:

1) Проверка отсутствия на диаграмме классов циклического наследования.

2) Проверка отсутствия на диаграммах недопустимых типов связей и объектов. Например: на диаграмме классов не могут размещаться прецеденты, а на диаграмме прецедентов - классы; между ролью и прецедентом допустимо только отношение ассоциации; между прецедентами – отношения включения, расширения и обобщения.
3) Инициатором действий в каждой диаграмме последовательности должен быть актор или объект, получивший «найденное сообщение». *Перед отправкой своего первого сообщения каждый объект на диаграмме последовательности должен получить хотя бы одно сообщение.*

к) Проверка взаимного соответствия диаграмм последовательности, пригодности, модели предметной области:

1) На диаграмме пригодности, построенной для некоторого прецедента должны использоваться только граничные объекты, соответствующие макетам интерфейса, связанных с этим прецедентом. А также только сущности, связанные с этим прецедентом. С другой стороны, все привязанные к прецеденту сущности должны быть обязательно использованы на диаграмме пригодности.
2) На диаграммах последовательности, построенных для прецедента должны быть использованы сущности и граничные объекты с диаграмм пригодности.
3) Если на диаграмме пригодности граничный объект и сущность взаимодействуют через процесс, *значит на диаграмме последовательности должен быть набор связывающих их сообщений*, однако они не обязательно должны связываться сообщением напрямую, *так как процесс может быть заменен новым классом.*

Таким образом, наиболее востребованными операциями над моделями являются: поиск элементов диаграммы с определенным свойством; поиск фрагмента диаграммы определенного вида.

1.4 Входные и выходные данные модуля формирования внутреннего представления

В разрабатываемую среду проектирования диаграммы вводятся в текстовом виде с использованием PlantUML [6]. Формат описания сущностей и текстовое описание прецедентов вводятся с использованием специальных элементов интерфейса, как показано на рисунке 1.6, и также сохраняются в текстовом виде. Указанная информация хранится в системе в исходном (текстовом) виде, так как именно она будет использоваться при необходимости изменения данных пользователем. Макеты интерфейса добавляются в систему в виде изображений.

На основе текстовых форм представления диаграмм существующий модуль PlantUML формирует соответствующие им изображения. Эти изображения выводятся в интерфейсе пользователя, сохраняются в системе и обновляются при изменении текстов диаграмм.

На вход системы формирования внутреннего представления подается:

- исходные тексты диаграмм (прецедентов, пригодности, последовательности, классов, ER);
- информация о наличии файлов макетов интерфейса для каждого граничного объекта;
- текстовое описание форматов для каждой сущности;
- текстовое описание последовательностей (основной и альтернативных) для каждого прецедента.

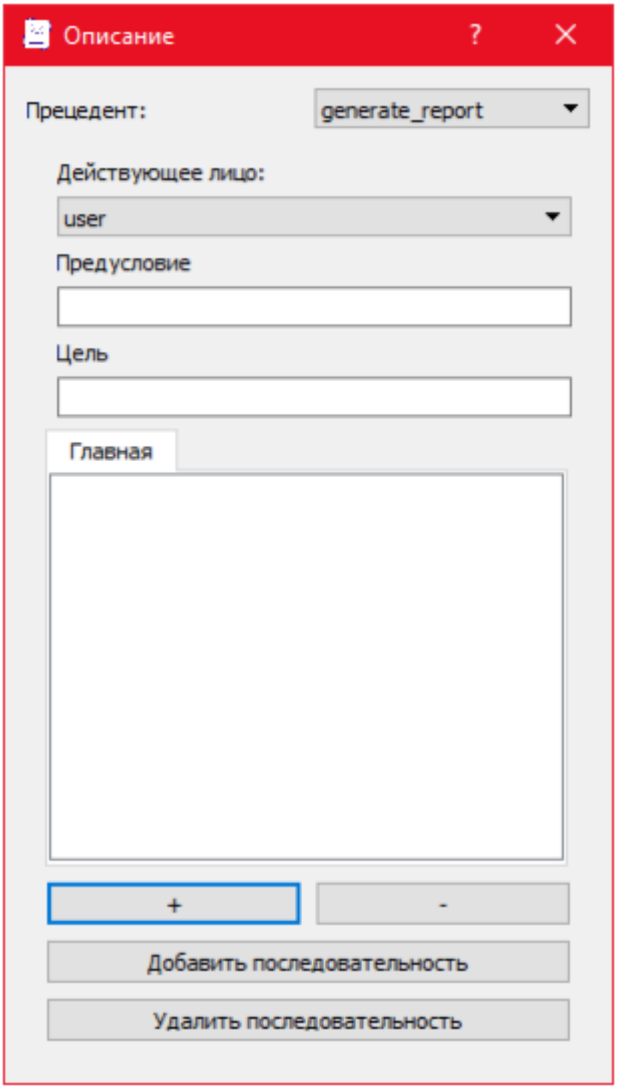


Рисунок 1.6 – Виджет ввода последовательностей прецедента

1.5 Выводы по главе

В результате работы модуля в памяти ЭВМ должны формироваться структуры данных, предоставляющие существенные для анализа моделей аспекты проекта в форме, обеспечивающей высокую эффективность обработки.

1. Рассмотрен модернизированный процесс проектирования ICONIX, выделено подмножество возможностей языка UML, поддержку которого целесообразно реализовать в учебной среде объектно-ориентированного моделирования.

2. Приведены методы проверки корректности проекта, выделены ключевые операции, высокую эффективность которых должны обеспечивать разрабатываемые структуры данных.

2 Проектирование модуля формирования внутреннего представления

Для эффективной реализации алгоритмов анализа, данные преобразуются в более удобный для обработки формат, содержащий в явном виде *все виды необходимых для анализа зависимостей*. Для хранения диаграмм хорошо подойдет графовая форма, так как диаграмма представляет собой набор вершин и дуг. Однако, графы среды моделирования связаны между собой, например, каждая диаграмма пригодности (задаваемая графом) связана с каким-либо прецедентом (соответствующего узлу другого графа).

2.1 Графовая форма представления диаграмм

При хранении в памяти ЭВМ используются несколько способов задания графов, отражающих различные типы зависимостей [16].

Отношение смежности задает связь между двумя вершинами графа, а отношение инцидентности – между ребром и вершиной. В связи с этим, задавать граф с помощью отношений смежности целесообразно, когда связи между вершинами достаточно простые, нет смысла в явном выделении дуг. Почти все дуги в диаграммах учебной среды моделирования являются ориентированными, кроме того:

- на диаграмме прецедентов должно поддерживаться три типа дуг, дуга типа <<*extend*>> может быть связана с точками расширения прецедента;
- на диаграмме классов стоит различать отношения наследования, композиции, агрегации и ассоциации, в ряде случаев имеет смысл хранить текст (комментарий) дуги;
- на диаграмме последовательности поддерживается 2 типа дуги (запрос/ответ), нумерация дуг задает порядок передачи сообщений, сообщения могут вкладываться в блоки альтернативных действий и циклов, анализу будет подвергаться текст дуги;
- дуги диаграммы пригодности нередко дополняются текстом;

- карта диалоговых окон исполняется в нотации диаграммы состояний, где на дугах подписываются условия смены состояний.

Поэтому, наиболее рациональным решением является использование *отношения инцидентности* для хранения всех типов диаграмм среды моделирования.

Хранить информацию о дугах и вершинах можно с использованием различных структур данных, при этом выбор структур влияет на вычислительную сложность исполняемых над графами алгоритмов. Каждое ребро графа инцидентно двум вершинам, поэтому в соответствующей строке/столбце *матрицы инцидентности* этого ребра будет лишь две связи, остальные ячейки матрицы всегда будут заполнены значениями, соответствующими отсутствию связи ребра с вершиной. В связи с этим, наиболее подходящей структурой данных для представления графов среды моделирования являются *списки инцидентности*.

При выполнении различных алгоритмов анализа зачастую необходимо получать информацию как о дугах, входящих в вершину, так и об исходящих дугах и смежных вершинах. Поэтому каждая вершина должна хранить список инцидентных ребер, а каждое ребро – ссылки на узлы, соответствующие началу и концу.

Для эффективной реализации перебора *всех* вершин/дуг графа, их необходимо явно хранить в списках. При этом поддержка упорядоченности списков по каким-либо признакам может снизить время поиска узлов по этим признакам за счет применения алгоритма двоичного поиска. Однако, алгоритмы анализа требуют поиска данных по различным критериям, что не позволяет выделить какой-либо один из них, поэтому хранить дуги и узлы целесообразно в неупорядоченных списках.

В таблице 2.1 рассмотрен ряд способов хранения графа в виде списков инцидентности:

а) граф задается списком вершин, каждая из которых хранит список исходящих дуг;

б) граф задается списком вершин, каждая из которых хранит список исходящих дуг и список входящих дуг;

в) граф задается списком вершин и списком дуг. Каждая вершина хранит список исходящих дуг и список входящих дуг.

На этих способах, была произведена оценка вычислительной сложности с позиции нотации «O» большого [17], для ранее выделенных ключевых операций. При этом используются следующие обозначения: N – количество узлов; M – количество дуг; K – количество дуг инцидентных вершине. Обычно K много меньше M.

Таблица 2.1 – Вычислительная сложность базовых алгоритмов используемых при анализе диаграмм

Операция	**Оценка вычислительной сложности (число операций)**		
	а)	**б)**	**в)**
Поиск дуги с определенным свойством	O(N*K)	O(N*K)	O(M)
Формирование списка смежных вершин	O(N*K)	O(K)	O(K)
Поиск узла с определенным свойством	O(N)	O(N)	O(N)
Формирование списка смежных ребер	O(N*K)	O(K)	O(K)
Поиск подграфа некоторого вида из L вершин	$O((N*K)^L)$	$O((N*K)^L)$	$O((N+M)^L)$

Из приведенной таблицы видно, что наибольшую эффективность алгоритмов анализа корректности моделей учебной среды объектно-ориентированного моделирования обеспечит хранение диаграмм с помощью графа в).

На рисунке 2.1 приведена диаграмма классов, иллюстрирующая хранение графа таким способом.

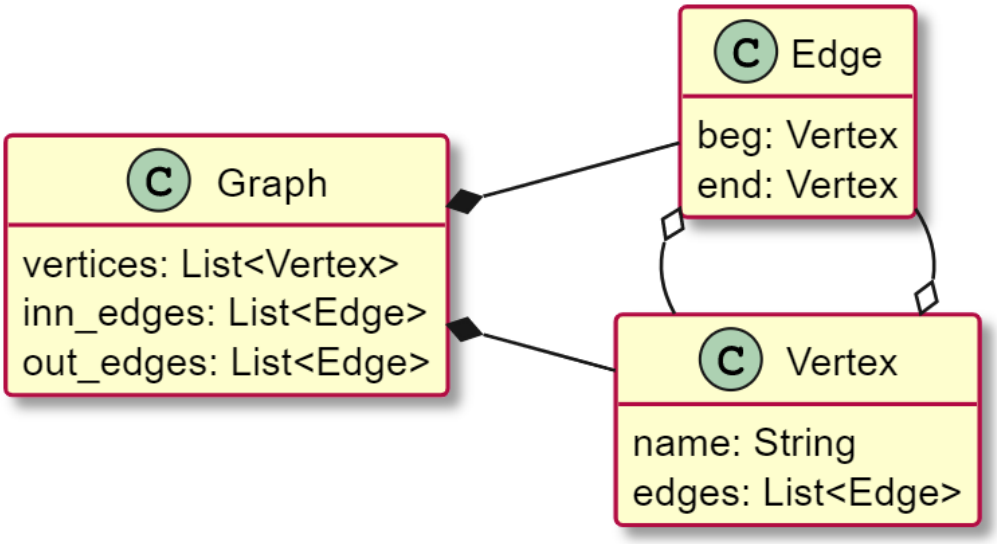


Рисунок 2.1 – Диаграмма классов для представления графа списком инцидентности

2.2 Прецеденты, текстовые описания, макеты и форматы

С учетом описанных в разделе 1 требований к диаграмме вариантов использования, была получена структура данных для ее внутреннего представления (рисунок 2.2).

Диаграмма вариантов использования описывается тремя классами, для графа есть UseCaseGraph, который содержит два списка узлов и дуг. Каждая дуга UcEdge содержит поле с типом, которое может принимать одно значение из четырех. При этом *тип напрямую зависит от стрелки в тексте диаграммы*, а в некоторых случаях и от заметки (таблица 2.2). Кроме того, дуга содержит информацию о начальном и конечном узлах.

Класс узла UcNode содержит тип, а также поля, хранящие информацию о входящих и исходящих дугах, inn_edges и out_edges соответственно. Тип узла явно указывается в тексте диаграммы при объявлении.

Дуги и узлы не могут существовать отдельно от конкретного экземпляра класса UseCaseGraph, именно поэтому на диаграмме отношение узел-граф и дуга-граф показано композицией.

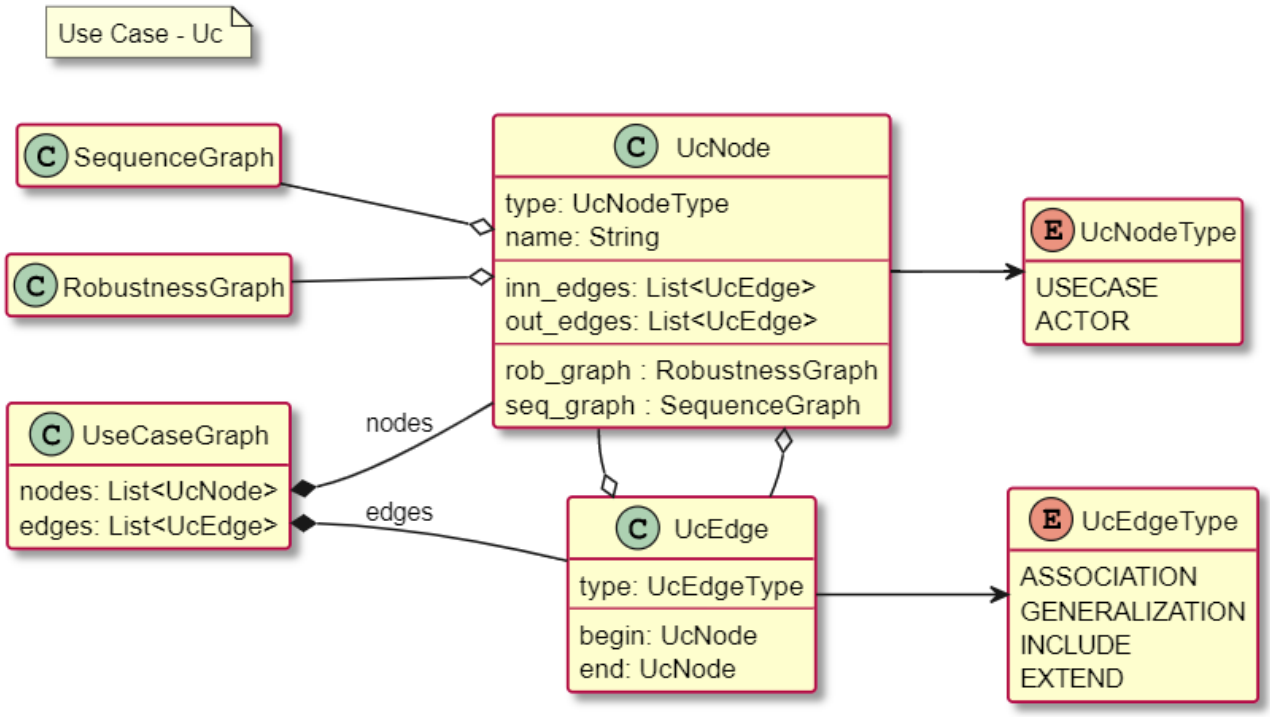


Рисунок 2.2 – Диаграмма классов, отражающая структуру данных диаграммы вариантов использования

Таблица 2.2 – Типы отношений на диаграмме прецедентов

Тип	**Тестовая форма (PlantUML)**		**Отображение**
	Формат описания дуги	**Заметка**	
Ассоциация (ASSOCIATION)	<--		
Обобщение (GENERALIZATION)	<|--		
Включение (INCLUDE)	<..	<<include>>	«include»
Расширение (EXTEND)	<..	<<extend>>	«extend»

Каждый узел диаграммы с типом USECASE должен быть связан с диаграммой пригодности и робастности (для этого в классе UcNode поля rob_graph и seq_graph), так как эти диаграммы необходимо построить для каждого прецедента.

Кроме того, каждый прецедент должен быть *связан по названию* с текстовым описанием, которое в свою очередь может содержать ссылки на макеты интерфейсов построенные пользователем (студентом).

Обычно в тексте прецедента может встречаться информация о различных сущностях (базы данных, взаимодействие по сети, чтение с диска), для которых будут подготавливаться описания форматов.

На рисунке 2.3 показана диаграмма классов, которая отражает как относятся между собой: прецедент, текстовое описание прецедента и дополнительные требования.

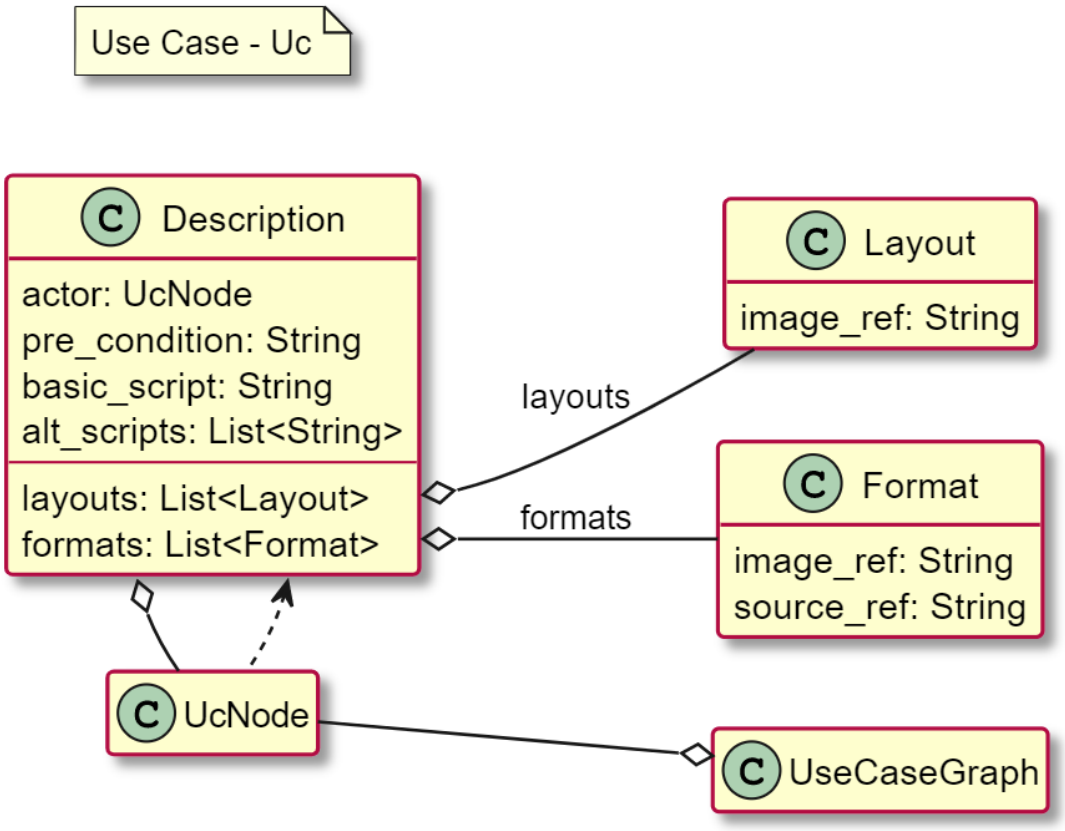


Рисунок 2.3 – Диаграмма классов, отражающая взаимосвязи для текстового описания прецедентов, макетов интерфейсов, форматов сущностей

Для текстового описания достаточно обозначить действующее лицо (актор из диаграммы прецедентов), задать предусловие, текст для главной последовательности, а также несколько альтернативных. С этой целью в классе Description есть соответствующие поля: actor, pre_condition, basic_script, alt_scripts. Ссылки на макеты интерфейсов будут укладываться в список layouts, а ссылки на форматы сущностей в список formats.

В связи с тем что узел диаграммы вариантов использования может быть как актором, так и прецедентом, между узлом и текстовым описанием возникает два типа отношений.

В качестве макета интерфейса пользователь устанавливает изображение image_ref, а для формата сущности – в зависимости от типа, может устанавливаться текст описания сущности source_ref, изображение image_ref (например для ER диаграммы).

2.3 Структура данных диаграммы пригодности

Для каждого прецедента необходимо построить диаграмму пригодности. В связи с этим на рисунке 2.4 между классами UcNode и RobustnessGraph присутствует взаимная агрегация. Кроме этого, структура для графа также содержит список для узлов и список для дуг.

Дуги RobEdge на этой диаграмме разрешены только одного типа. Часть дуг дополняется подписями, для этого в классе существует поле name. Вместе с тем дуга содержит ссылки на узлы: начало и конец, beg и end соответственно.

В свою очередь, узлы RobNode могут иметь разный тип (перечислены в RobNodeType), а также принадлежать либо основному, либо одному из альтернативных сценариев (для этого используется поле-флаг is_error). При этом, если поле type имеет значение BOUNDARY, то узел может быть связан по названию с макетом интерфейса Layout, а если поле type имеет значение ENTITY – то с форматом сущности Format.

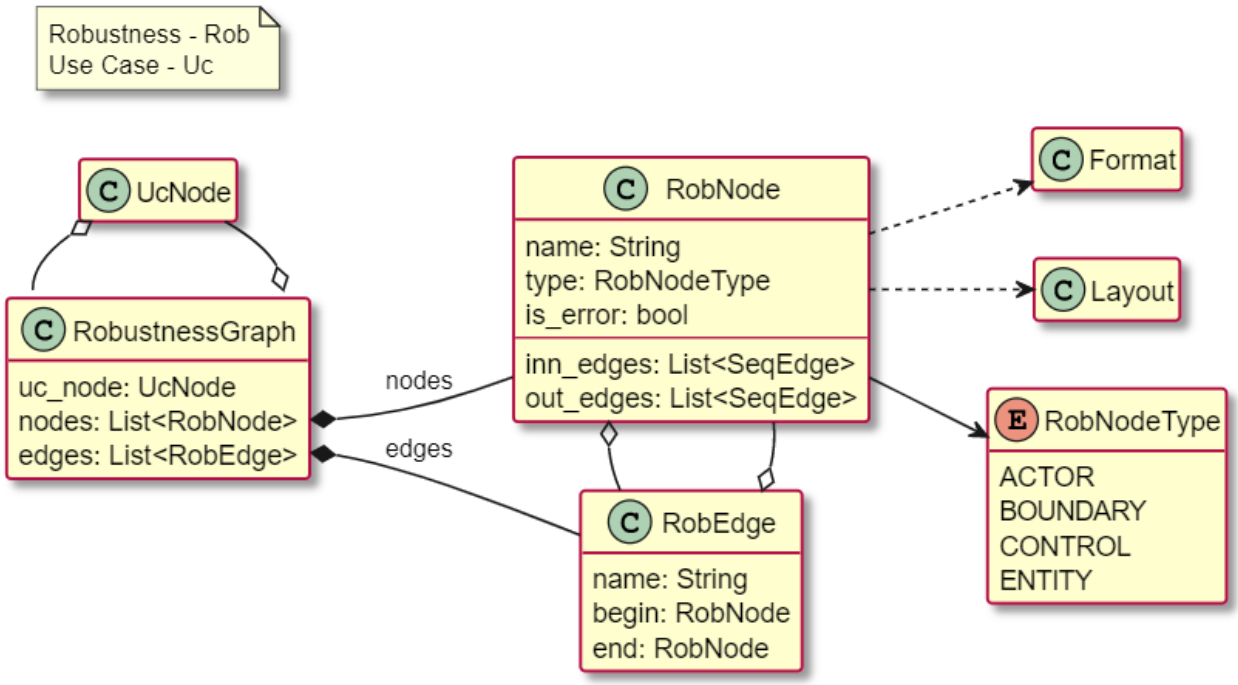


Рисунок 2.4 – Диаграмма классов, отражающая структуру данных диаграммы пригодности

2.4 Структура данных диаграммы последовательности

На рисунке 2.5 представлена структура данных для диаграммы последовательности. Узлы SeqNode на этой диаграмме частично повторяют реализацию узлов диаграммы пригодности RobNode. В тоже время дуги SeqEdge, могут иметь разный тип (зависит от сообщения, допустимо только два типа SYNC и REPLY). Сплошная стрелка в тексте диаграммы на языке PlantUML соответствует дуге с типом SYNC, пунктирная – REPLY. Кроме того, дуги могут группироваться во фрагменты – и для того чтобы сохранить об этом информацию и в дальнейшем использовать, были выделены следующие структуры: SeqGroup, SeqRef, SeqFrag, SeqOpd и Stamp.

Обычно на этой диаграмме с помощью фрагментов описываются альтернативные последовательности. В свою очередь, каждый такой фрагмент состоит из нескольких операндов взаимодействия SeqOpd. Как правило opt и loop состоят только из одного операнда, а alt из нескольких. Каждый операнд имеет условие срабатывания – поле condition.

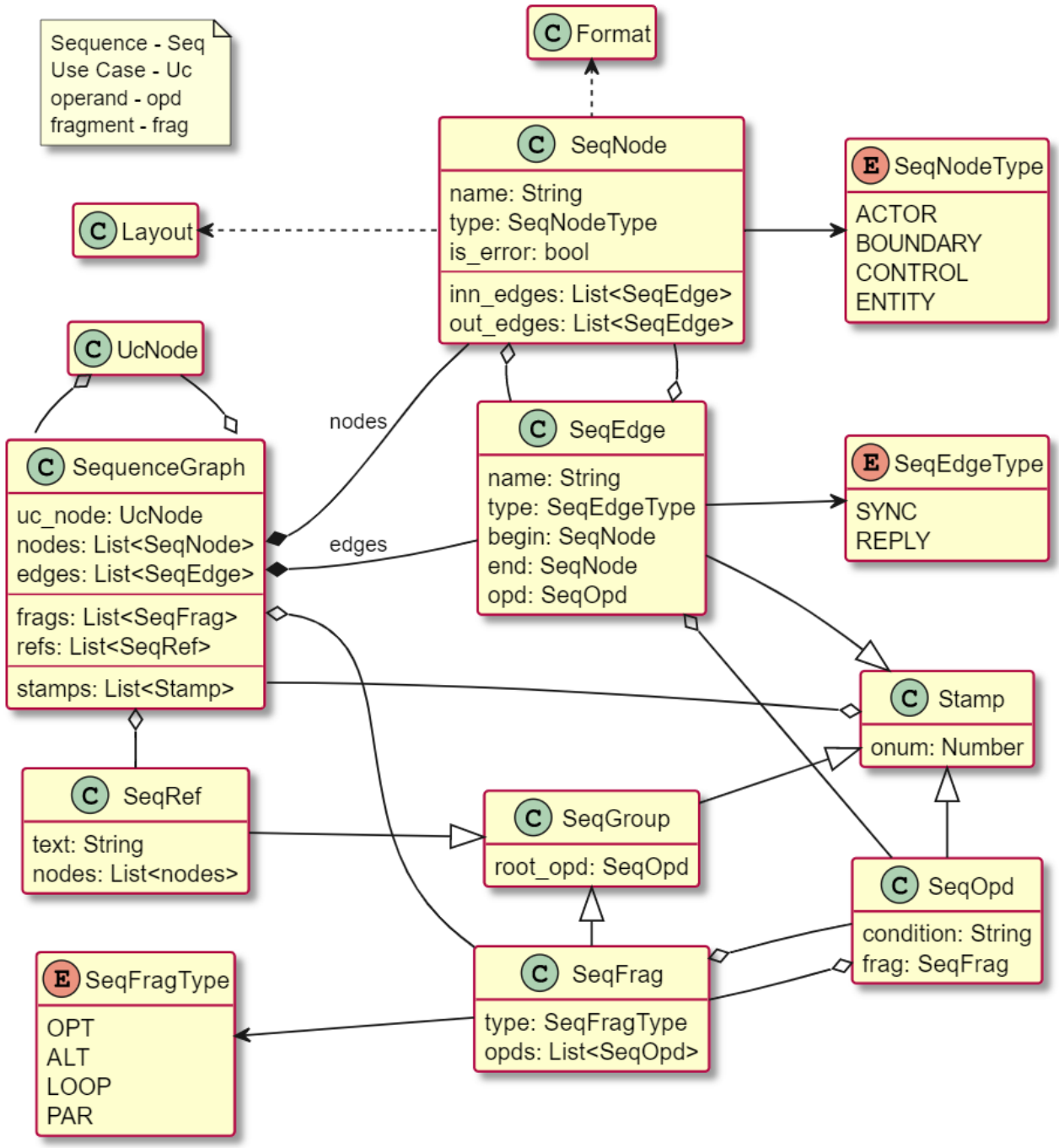


Рисунок 2.5 – Диаграмма классов, отражающая структуру данных диаграммы последовательности

Одной из особенностей диаграммы последовательности является ось времени, именно поэтому важно для каждого элемента на этой оси дополнительно сохранить метку времени (номер появления на оси). Для этого все элементы диаграммы, которые «привязаны» ко времени, наследуются от

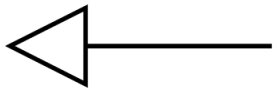
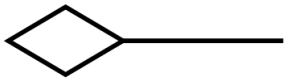
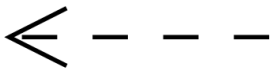
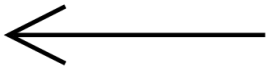
класса Stamp. В свою очередь этот класс содержит поле onum – порядковый номер.

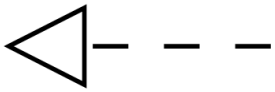
Класс для графа SequenceGraph кроме списков для узлов nodes и дуг edges, содержит еще списки для фрагментов frags, так и для предусловий refs. Отдельно, чтобы расширить возможности по анализу этой диаграммы был выделен список stamps, в котором находятся элементы уложенные по номеру появления на оси времени.

2.5 Структура данных диаграммы классов

Как и диаграмма прецедентов – диаграмма классов одна на весь проект. На этой диаграмме допустимо четыре типа узлов и шесть типов дуг. Для узла тип определяется его названием и явно указывается в тексте диаграммы. В то же время тип дуги определяется стрелкой и зависит не только от тела, но и наконечника. В таблице 2.3 представлены символы из которых состоят стрелки и соответствующий тип отношения.

Таблица 2.3 – Типы отношений на диаграмме классов

Тип	**Тестовая форма (PlantUML)**	**Отображение**
	Формат описания дуги	
Наследование (GENERALIZATION)	<|--	
Композиция (COMPOSITION)	*--	
Агрегация (AGGREGATION)	o--	
Зависимость (DEPENDENCY)	<..	
Ассоциация (ASSOCIATION)	<--	

Тип	Тестовая форма (PlantUML)	Отображение
	Формат описания дуги	
Реализация (IMPLEMENTATION)	<|..	

Структура представлена на рисунке 2.6. Класс графа ClassGraph для этой диаграммы содержит только список дуг nodes и узлов edges. Для узла и дуги используются классы: ClassNode и ClassEdge соответственно. Допустимые значения типов для узлов перечислены в ClassNodeType, а для дуг в ClassEdgeType.

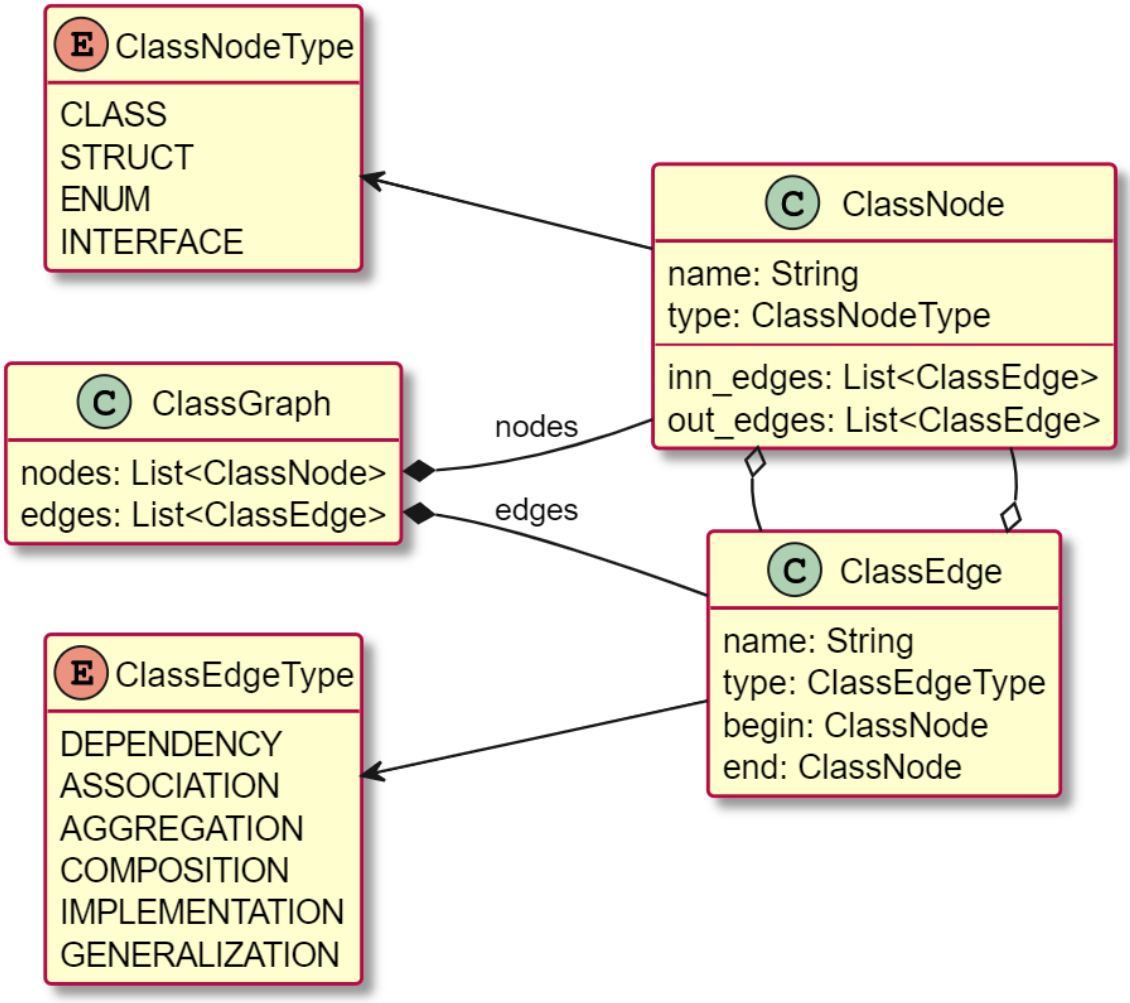


Рисунок 2.6 – Диаграмма классов, отражающая структуру данных диаграммы классов

2.6 Структура данных диаграммы потока экранов

Дополнительным шагом при проектировании является построение пользователем диаграммы макетов интерфейсов, которая исполняется в нотации диаграммы состояний. Для того чтобы в дальнейшем работать с этой диаграммой, ее текст также будет переводиться в структуру, которая представлена на рисунке 2.7.

Как правило на этой диаграмме три типа узлов: обычное состояние, два псевдо-состояния для начала и конца. Допустимые значения, которые может принимать тип узла перечислены в LwNodeType. Списки узлов и дуг диаграммы находятся в классе LayoutFlowGraph. Структуры LwNode и LwEdge используются для представления узла и дуги соответственно. При этом узел связан по названию с макетом интерфейса Layout.

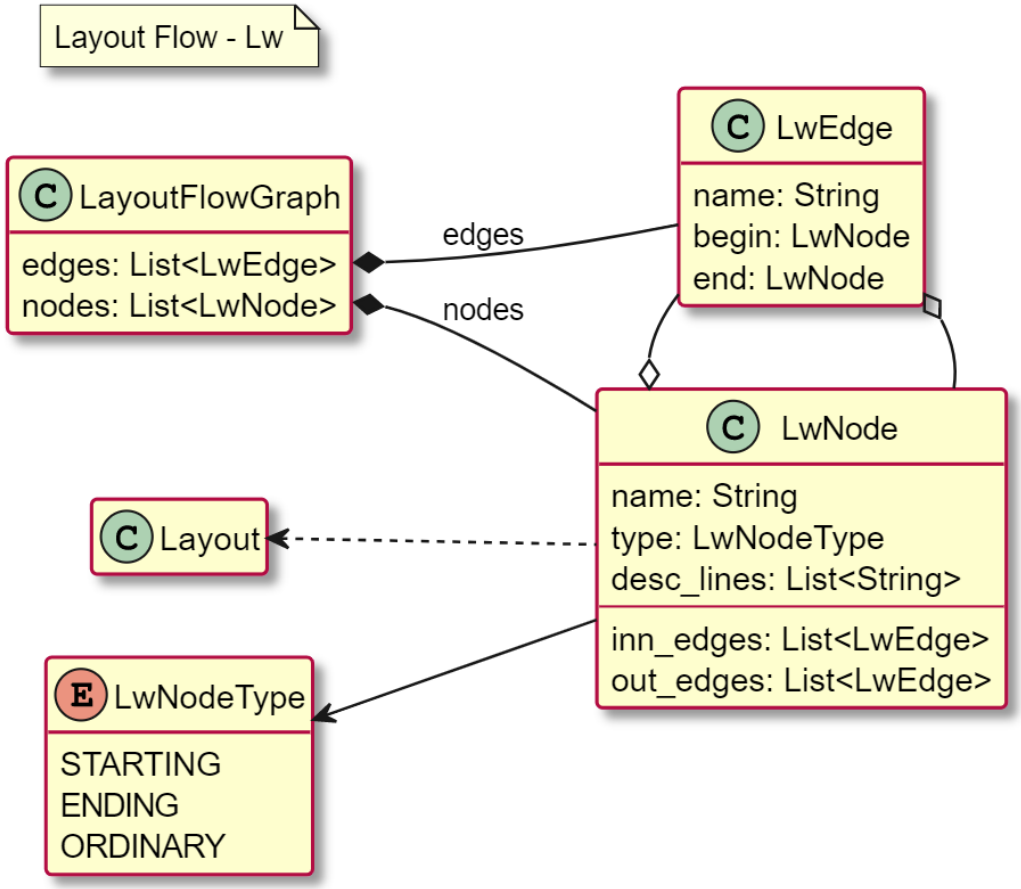


Рисунок 2.7 – Диаграмма классов, отражающая структуру данных диаграммы потока экранов

2.7 Выводы по главе

1. На основе анализа известных способов представления графа в приложении к решаемой задаче, показано, что наиболее эффективным является хранение моделей среды проектирования в виде неупорядоченных списков инцидентности.

2. С учетом специфики каждого вида диаграмм, поддерживаемых в создаваемой среде проектирования, разработаны структуры данных.

3 Реализация и тестирование

Модуль PlantUML, используемый для преобразования текстовой формы диаграмм в изображение реализован на языке Java и поставляется в виде исполняемого файла. В рамках выполняемой работы модифицировать этот модуль не требуется, поэтому целесообразно использовать этот модуль, запуская его при необходимости в виде отдельного процесса.

Прототип учебной среды объектно-ориентированного моделирования (далее учебная среда) разработан на языке программирования C++. Для организации модульного тестирования и создания графического пользовательского интерфейса использовалась библиотека Qt. Разработка новой версии учебной среды продолжена с этим же набором инструментов.

Для тестирования приложения применен подход, заключающийся в сохранении состояния структур данных в памяти во внешние файлы в формате JSON. Для этого применена библиотека nlohmann [18].

3.1 Особенности реализации

На этапе проектирования были получены модели структур данных, с помощью которых, диаграммы, построенные в рамках модернизированного процесса ICONIX, будут храниться в памяти ЭВМ. При реализации этих структур, были выделены базовые классы Graph, Edge, Node и Unit, определение которых приведено в приложении А. Иерархия этих классов представлена на рисунке 3.1.

Для ссылок используются умные указатели из стандартной библиотеки, благодаря этому нет необходимости в ручном управлении выделенными ресурсами. Кроме того, для того чтобы избежать ситуаций с циклической зависимостью [19] используется связка shared_ptr и weak_ptr.

Всем методам, выполняющим ввод данных с файла на вход передается ссылка на класс istream, являющийся базовым для всех классов, задающих

потоки ввода. Поэтому на вход этих методов, помимо файловых потоков, могут быть переданы строковые потоки, содержимое которых формируется программистом явно в исходном коде программы и хранится в оперативной памяти. Такой подход не только упрощает разработку модульных тестов, но и обеспечивает более высокую эффективность их выполнения за счет исключения трудоемких операций доступа к файлам на диске.

Объект потока содержит текст диаграммы. Однако, такой текст без дополнительных ограничений может содержать сложные синтаксические конструкции, так как инструмент PlantUML охватывает большое количество возможностей языка UML. Поэтому, с учетом описанного выше модернизированного процесса ICONIX для каждой диаграммы был подготовлен документ, содержащий описание доступного синтаксиса подмножества языка PlantUML.

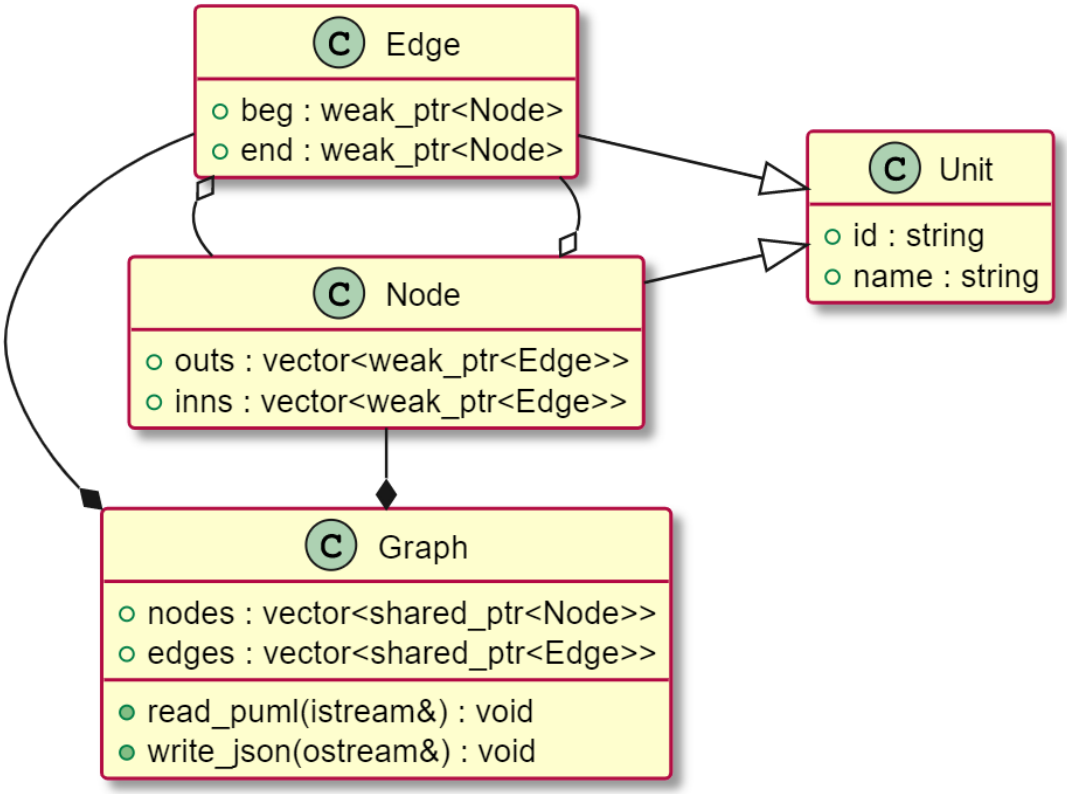


Рисунок 3.1 – Диаграмма базовых классов

Для формализации поддерживаемого набора инструкций использовалась нотация РБНФ [20]. Описание доступного для использования в учебной среде подмножества языка PlantUML представлено в приложении Б.

При написании текста диаграммы на языке PlantUML необходимо соблюдать определенные правила форматирования [21]. Одно из таких требований – каждая «синтаксическая конструкция» должна начинаться с новой строки. Это означает, что каждый элемент диаграммы, такой, как узел, дуга или комментарий, должен быть записан в отдельной строке, чтобы соответствовать стандартам PlantUML. Такие особенности языка позволяют читать документ с текстом диаграммы построчно, при этом накладывать дополнительные ограничения, которые описаны нотацией РБНФ.

При этом некоторые допустимые «синтаксические конструкции» могут приводить к добавлению в граф нескольких элементов. Например A --o B описывает дугу графа между узлами A и B. Однако, если узлы не были описаны ранее – то она приводит к добавлению в граф не только дуги, но и узлов с типами, которые задаются по умолчанию инструментом PlantUML. Для диаграммы вариантов использования будет актор, а для диаграммы классов – класс.

Чтобы реализовать такие ограничения в программном коде, а также получить информацию, которая необходима для формирования структур использовались регулярные выражения [22].

3.2 Документация

Так как РБНФ достаточно сложна для восприятия – чтобы познакомиться с синтаксисом PlantUML можно воспользоваться официальным справочным руководством [21]. Однако в этом руководстве есть множество сложных текстов диаграмм, которые нельзя преобразовать в структуры, так как модуль поддерживает лишь ограниченный набор инструкций. Приложение В настоящей диссертации содержит примеры допустимых текстов диаграмм,

охватывающих почти весь набор «синтаксических конструкций» с учетом ограничений.

Кроме того, описание доступных инструкций *в свободной форме* можно найти в репозитории разработанного модуля и структур данных [23]. Такая возможность позволит пользователю более подробно изучить доступный синтаксис и писать код диаграмм с учетом ограничений модуля.

3.3 Организация модульного тестирования

Разработанный модуль формирует в памяти ЭВМ графовые структуры данных. Одна из проблем тестирования модуля заключается в том, что один и тот же граф может быть представлен в памяти различными способами, отличающимися, например, порядком перечисления узлов или дуг. Проверка эквивалентности двух графов, в общем случае, является NP-сложной задачей. В связи с этим, при тестировании точно задается порядок узлов и дуг в памяти ЭВМ.

Задача тестирования осложнялась также необходимостью сравнения графовых структур, узлы которых связаны по указателям. Эта проблема была решена с помощью написанного модуля, выполняющего экспорт графого представления модели в текстовую форму (в формате JSON). При тестировании выполняется сравнение текстовых форм для эталонного графа и сформированного созданным в рамках работы модулем. Пример перевода текста диаграммы классов на языке PlantUML в набор объектов JSON представлен в приложении Г.

В ходе тестирования модуля разработано более ста тестовых случаев, оценка тестового покрытия получена с помощью OpenCppCoverage [24] и составляет 93%

3.4 Инструментирование кода

С помощью инструмента Cppcheck [25] и PVS Studio [26] были обнаружены проблемы связанные с производительностью, а также получены рекомендации по улучшению качества кода. Некоторые их них:

- передача аргумента по значению, когда можно осуществить передачу по ссылке;
- использование обычного цикла, когда можно взять алгоритм из стандартной библиотеке;
- перекрытие аргумента функции локальной переменной.

Исходный код был проанализирован на возможные ошибки и утечки памяти. Для этого использовался инструмент Valgrind [27]. При этом запуск выполнялся на программе, которая тестирует модуль. Утечки обнаружены не были (результат представлен на рисунке 3.2).

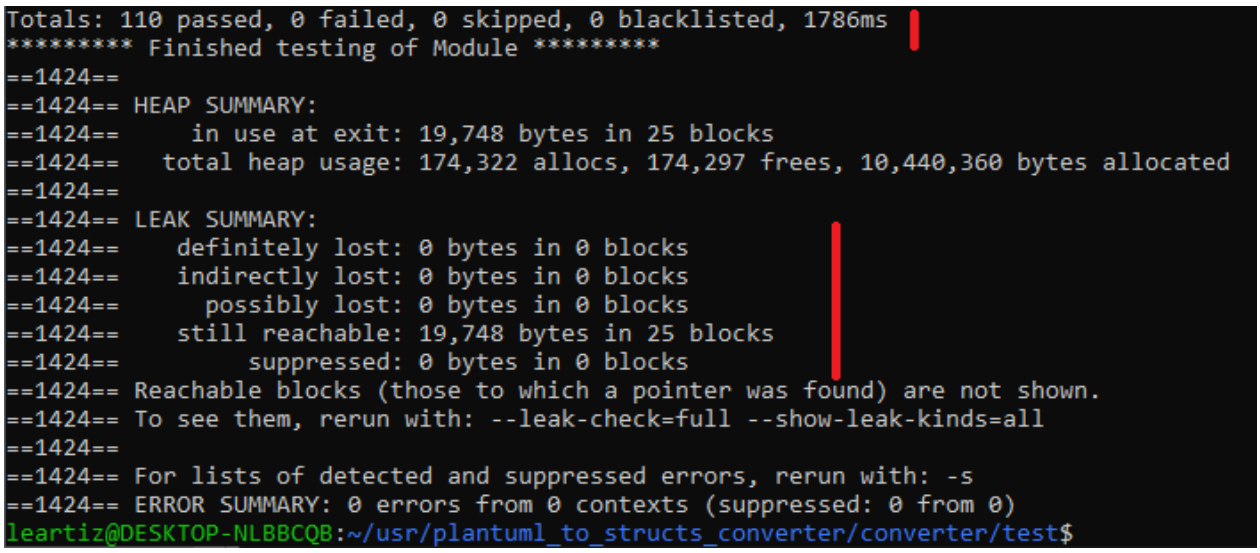


Рисунок 3.2 – Отчет полученный в результате выполнения Valgrind

3.5 Интеграция в среду объектно-ориентированного моделирования

Общая схема связи разработанного модуля формирования внутреннего представления учебной среды с другими элементам, а также входными и выходными данными представлена на рисунке 3.3.

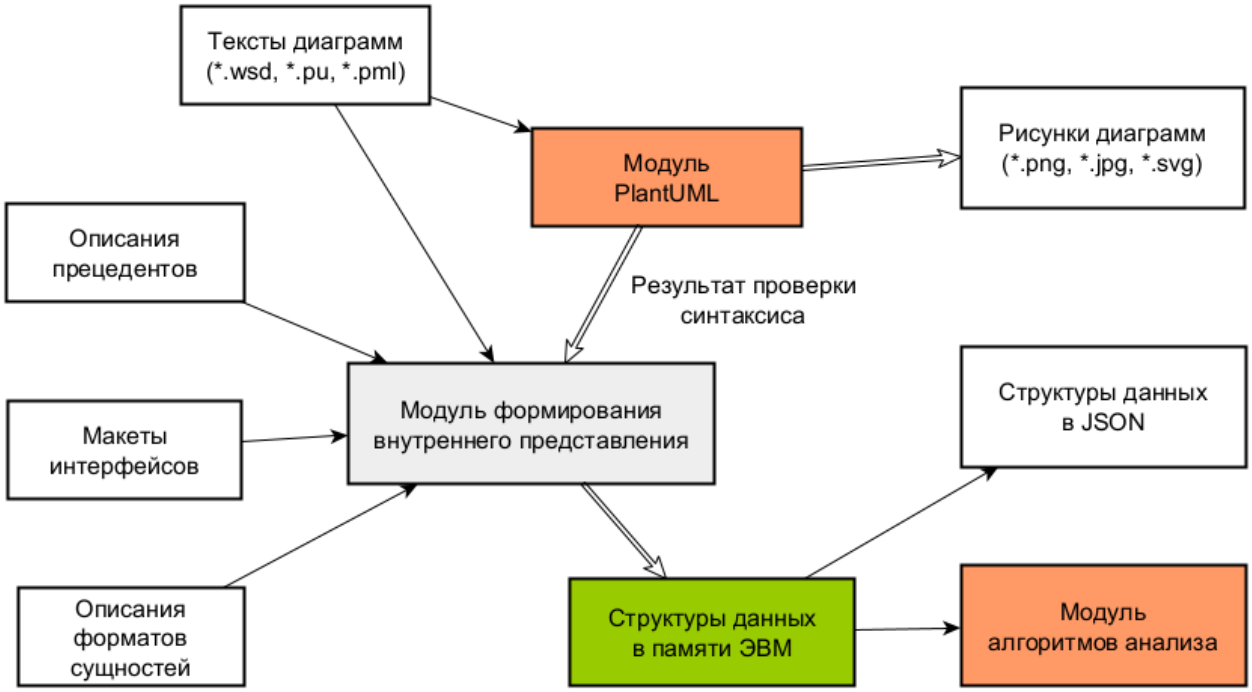


Рисунок 3.3 – Эскизная схема «расположения» других модулей и данных по отношению к разработанному модулю

В дальнейшем разработанный модуль можно без трудностей добавить в учебную среду, для этого достаточно подключить файлы с исходным кодом, которые доступны в GitHub репозитории [23]. Благодаря тому, что при разработке использовались кросс-платформенные инструменты, сборка проекта с добавленным модулем не должна вызвать проблем как на Linux, так и на Windows.

3.6 Выводы по главе

1. Описаны допустимые синтаксические конструкции при написании текстов диаграмм для подмножества языка PlantUML в нотации РБНФ.

2. Разработаны структуры для внутреннего представления диаграмм и модуль перевода документов PlantUML в эти структуры – подготовлены для внедрения в учебную среду.

3. Выполнено тестирование кода; с помощью открытых инструментов получена информации о том насколько код покрыт тестами.

4. С помощью статических анализаторов кода были найдены и устранены недочеты, улучшено качество программного кода.

5. Подготовлена документация для пользователя. Приведены примеры текстов диаграмм, написанные с учетом ограничений модуля.

ЗАКЛЮЧЕНИЕ

В рамках работы разработаны структуры данных, обеспечивающие высокое быстродействие алгоритмов анализа моделей программ, созданных в учебной среде объектно-ориентированного проектирования. Реализован программный модуль, обеспечивающий формирование разработанных структур в памяти ЭВМ на основе введенных пользователем данных, а также модуль выполняющий сериализацию структур для проверки корректности формирования. Для программных модулей разработаны наборы модульных тестов, выполнена проверка кода различными инструментами анализа.

Работа апробирована на трех конференция разного уровня:

- 65-я Всероссийская научная конференция МФТИ (Москва) [28, 29], получен диплом за лучший устный студенческий доклад в секции алгоритмов и технологий программирования;
- «Проспект Свободный — 2023» (Красноярск) [30];
- «Современные цифровые технологии» (Алтай) [31].

СПИСОК СОКРАЩЕНИЙ

ЭВМ – Электронная вычислительная машина

UML (англ. Unified Modeling Language) – унифицированный язык моделирования

JSON (англ. JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript

РБНФ – Расширенная форма Бэкуса – Наура

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Frank, U. Ein Bezugsrahmen zur Beurteilung objektorientierter Modellierungssprachen-veranschaulicht am Beispiel von OML und UML / U. Frank, M. Prasse // Arbeitsberichte des Instituts für Wirtschaftsinformatik. – 1997. – №. 6.

2. Разработка программных проектов на основе Rational Unified Process (RUP) / Г. Поллис, Л. Огастин, К. Лоу, Д. Мадхар. – Москва : Бином, 2005. – 256 с. – ISBN: 5-9518-0096-X, 0-321-19950-2.

3. Розенберг, Д. Применение объектного моделирования с использованием UML и анализ прецедентов на примере разработки книжного Internet-магазина / Д. Розенберг, К. Скотт. – Москва : ДМК Пресс, 2002. – 160 с. – ISBN 5-94074-050-2.

4. Васильев, В. С. Процесс разработки программного обеспечения ICONIX / В. С. Васильев // Блог программиста. – URL: https://pro-prof.com/archives/4126 (дата обращения: 13.02.2023).

5. Исайкин, А. А. Средство объектно-ориентированного моделирования для учебного процесса : специальность 09.03.01 «Информатика и вычислительная техника» : выпускная квалификационная работа бакалавра / А. А. Исайкин ; Сибирский федеральный университет. – Красноярск, 2020.

6. Инструмент для создания диаграмм PlantUML. – URL: https://plantuml.com/ru (дата обращения: 13.02.2023).

7. Шишкина, И. С. Методы поиска ошибок проектирования для учебной среды объектно-ориентированного моделирования / И. С. Шишкина, А. А. Исайкин, А. Г. Хантимиров // Наука в современном мире: результаты исследований и открытий : Сборник научных трудов по материалам IV Международной научно-практической конференции, Анапа, 08 июня 2022 года. – Анапа : Общество с ограниченной ответственностью «Научно-исследовательский центр экономических и социальных процессов» в Южном Федеральном округе, 2022. – С. 110-116. – EDN YLXBKJ.

8. Чилясов, В. Создание проекта форм интерфейса и карты диалоговых окон в PLANTUML / В. Чилясов // Хабр. – URL: https://habr.com/ru/articles/279373 (дата обращения: 13.02.2023).

9. Schwichtenberg, S. From open API to semantic specifications and code adapters / S. Schwichtenberg, C. Gerth, G. Engels // Proceedings of the 24th IEEE International Conference on Web Services (ICWS). – IEEE, 2017. – P. 484-491.

10. Barbaglia, G. Definition of REST web services with JSON schema / G. Barbaglia, S. Murzilli, S. Cudini // Software: Practice and Experience. – 2017. – Т. 47. – №. 6. – P. 907-920.

11. Васильев, В. С. Нотации модели сущность-связь (ER диаграммы) / В. С. Васильев // Блог программиста. – URL: https://pro-prof.com/archives/8126 (дата обращения: 13.02.2023).

12. ISO/IEC 19505-1:2012 Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 1: Infrastructure // ISO – International Organization for Standartization. – URL: https://www.iso.org/standard/32624.html (дата обращения 03.10.2022).

13. Дерюгина, О. А. Программно-математические средства рефакторинга UML-диаграмм классов с учетом заданных критериев качества / О. А. Дерюгина // Cloud of science. – 2018. – Т.5. – №1. URL: https://cyberleninka.ru/article/n/programmno-matematicheskie-sredstva-refaktoringa-uml-diagramm-klassov-s-uchetom-zadannyh-kriteriev-kachestva (дата обращения: 17.10.2022).

14. Фаулер, М. UML. Основы / М. Фаулер, К. Скотт. – 2-е изд., пер. с англ. – СПб. : Символ-Плюс, 2002. – 192 с.

15. Miller, G. A. The magical number seven, plus or minus two: Some limits on our capacity for processing information / G. A. Miller // The Psychological review. – 1956. – Т. 63. – №. 2. – P. 81-97.

16. Васильев, В. С. Оптимизация и трансформация функционально-потоковых параллельных программ : специальность 2.3.5 «Математическое и

программное обеспечение вычислительных систем, комплексов и компьютерных сетей» : диссертация на соискание ученой степени кандидата технических наук / Васильев Владимир Сергеевич ; Сибирский федеральный университет. – Красноярск, 2023. – 166 с.

17. Васильев, В. С. Анализ сложности алгоритмов. Примеры / В. С. Васильев // Блог программиста. – URL: https://pro-prof.com/archives/1660 (дата обращения: 13.02.2023).

18. JSON for Modern C++. – URL: https://json.nlohmann.me (дата обращения: 20.04.2023).

19. Проблемы с круговой зависимостью с std::shared_ptr и умный указатель std::weak_ptr. – URL: https://radioprog.ru/post/1312 (дата обращения: 20.04.2023).

20. Легалов, А. И. Основы разработки трансляторов / А. И. Легалов // SoftCraft. – URL: http://www.softcraft.ru/translat/lect/ (дата обращения: 20.04.2023).

21. Справочное руководство по языку PlantUML. Построение диаграмм UML с использованием PlantUML. – URL: https://pdf.plantuml.net/PlantUML_Language_Reference_Guide_ru.pdf (дата обращения: 20.04.2023).

22. Козлов, С. В. Применение регулярных выражений для обработки текстовых данных / С. В. Козлов, А. В. Светлаков // International Journal of Open Information Technologies. – 2022. – Т. 10, № 9. – URL: https://cyberleninka.ru/article/n/primenenie-regulyarnyh-vyrazheniy-dlya-obrabotki-tekstovyh-dannyh/viewer (дата обращения: 20.04.2023).

23. Модуль и структуры данных для внутреннего представления диаграмм учебной среды объектно-ориентированного моделирования. – URL: https://github.com/Leartiz/plantuml_to_structs_converter.

24. OpenCppCoverage – инструмент покрытия кода с открытым исходным кодом для C++ под Windows. – URL:

https://github.com/OpenCppCoverage/OpenCppCoverage (дата обращения: 20.04.2023).

25. Cppcheck is a static analysis tool for C code. – URL: http://cppcheck.net/ (дата обращения: 20.04.2023).

26. Статический анализатор кода для программ, написанных на C, C++, C++/CLI, C++/CX, C# и на Java. – URL: https://pvs-studio.ru/ru/ (дата обращения: 20.04.2023).

27. Valgrind – программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования. – URL: https://valgrind.org/ (дата обращения: 20.04.2023).

28. 65-я Всероссийская научная конференция / МФТИ. – URL: https://conf.mipt.ru/ (дата обращения: 14.05.2023).

29. Материалы 65-й Всероссийской научной конференции МФТИ в честь 115-летия Л. Д. Ландау // Прикладная математика и информатика / МФТИ. – URL: https://mipt.ru/upload/medialibrary/094/prikladnaya-matematika-i-informatika.pdf.

30. XIX Международная научная конференция студентов, аспирантов и молодых ученых «Проспект Свободный – 2023» / Сибирский федеральный университет. – URL: https://conf.sfu-kras.ru/prospect-svobodniy-2023 (дата обращения: 14.05.2023).

31. II Всероссийская научно-практическая конференция «Современные цифровые технологии» / Алтайский государственный технический университет им. И.И. Ползунова. – URL: https://www.altstu.ru/structure/unit/noo/scienceevent/4874/ (дата обращения: 14.05.2023).

ПРИЛОЖЕНИЕ А

Фрагменты исходного кода

Листинг А.1 – Определение базовых классов Graph, Edge и Node

```
1.   struct Graph {
2.       struct Unit {
3.           std::string id, name;
4.           virtual ~Unit() = default;
5.       };
6.
7.       struct Edge;
8.       struct Node : Unit {
9.           std::vector<std::weak_ptr<Edge>> outs, inns;
10.      };
11.
12.      struct Edge : Unit {
13.          std::weak_ptr<Node> beg, end;
14.      };
15.
16.  public:
17.      Graph();
18.      virtual ~Graph() = default;
19.
20.      virtual void read_puml(std::istream&) = 0;
21.      virtual void write_json(std::ostream&) = 0;
22.
23.  public:
24.      std::vector<std::shared_ptr<Node>> nodes;
25.      std::vector<std::shared_ptr<Edge>> edges;
26.  }
```

ПРИЛОЖЕНИЕ Б

Описание доступного PlantUML

Листинг Б.1 – Описание PlantUML для диаграммы прецедентов

```
<nln> ::= "\n"
<ids_char> ::= ([A-Z] | [a-z] | [0-9] | "_")+

<string>  ::= <any_char>+
<node_id> ::= <ids_char>+

<end_curly_brace> ::= "}"

/* ---------------------------------------------------------------------- */

/* стрелка */
<left_arrow_head> ::= "<" | "<|"
<rght_arrow_head> ::= ">" | "|>"
<arrow_direction> ::= ("l" | "r" | "d" | "u") |
                      ("left" | "right" | "down" | "up")
<arrow_body>      ::= ("." | "-")+ <arrow_direction> ("." | "-")+
<arrow>           ::= <left_arrow_head>? <arrow_body> <rght_arrow_head>?

/* составные конструкции (не используются напрямую) */

<part_whole_node> ::= "\"" <string> "\"" "as" <node_id>
<usecase_id>      ::= "(" <string> ")"
<actor_id>        ::= ":" <string> ":"
<node>            ::= (<usecase_id> | <actor_id> | <node_id>)

/* составные конструкции (построчно вводит пользователь) */

<enduml>    ::= "@enduml"
<startuml>  ::= "@startuml"
<directive> ::= <startuml> | <enduml>

/* --- */

<actor_short_node> ::= <actor_id> <reqwhs> "as" <node_id>
<actor_whole_node> ::= "actor" (<actor_short_node> | <part_whole_node>)
<actor_node>       ::= (<actor_short_node> | <actor_whole_node>)

<usecase_short_node> ::= <usecase_id> "as" <node_id>
<usecase_whole_node> ::= "usecase" \
                         (<usecase_short_node> | <part_whole_node>)
<usecase_node>       ::= (<usecase_short_node> | <usecase_whole_node>)

/* --- */

<connection> ::= <node> <arrow> <node>

/* --- */

<beg_grouping> ::= ("rectangle" | "package") "{"
<grouping>     ::= <beg_grouping>
```

```
                    <nln>
                    ((<actor_node> | <usecase_node> | <connection>) <nln>)*
                    <nln>
                    <end_curly_brace>

/* --- */

<diagram> ::= <startuml> <nln>
              ((<actor_node> | <usecase_node> |
              <connection> | <grouping>) <nln>)*
              <enduml>

```

Листинг Б.2 – Описание PlantUML для диаграммы пригодности

```
<nln> ::= "\n"
<ids_char> ::= ([A-Z] | [a-z] | [0-9] | "_")+

<string>  ::= <any_char>+
<node_id> ::= <ids_char>+

<end_curly_brace> ::= "}"
/* ------------------------------------------------------------------ */
/* стрелка */
<left_arrow_head> ::= "<"
<rght_arrow_head> ::= ">"
<arrow_direction> ::= ("l" | "r" | "d" | "u") |
                      ("left" | "right" | "down" | "up")
<arrow_body>      ::= "-"+ <arrow_direction> "-"+
<arrow>           ::= <left_arrow_head>? <arrow_body> <rght_arrow_head>?

/* составные конструкции (не используются напрямую) */

<part_whole_node> ::= "\"" <string> "\"" "as" <node_id>
<node>            ::= <node_id>

/* составные конструкции (построчно вводит пользователь) */

<enduml>    ::= "@enduml"
<startuml>  ::= "@startuml"
<directive> ::= <startuml> | <enduml>

/* --- */

<whole_node> ::= ("actor" | "boundary" |
                  "entity" | "control") <part_whole_node>
<short_node> ::= ("actor" | "boundary" |
                  "entity" | "control") <node_id>

/* --- */

<connection> ::= <node> <arrow> <node>

/* --- */

<diagram> ::= <startuml> <nln>
              ((<whole_node> | <short_node> |
              <connection>) <nln>)*
              <enduml>

```

Листинг Б.3 – Описание PlantUML для диаграммы последовательности

```
1.   <nln> ::= "\n"
2.   <ids_char> ::= ([A-Z] | [a-z] | [0-9] | "_")+
3.
4.   <string>  ::= <any_char>+
5.   <node_id> ::= <ids_char>+
6.
7.   <end_curly_brace> ::= "}"
8.
9.   /* ------------------------------------------------------------------ */
10.
11.  /* стрелка */
12.  <left_arrow_head> ::= "<"
13.  <rght_arrow_head> ::= ">"
14.  <arrow_direction> ::= ("l" | "r" | "d" | "u") |
15.                       ("left" | "right" | "down" | "up")
16.  <arrow_body>      ::= "-"+ <arrow_direction> "-"+
17.  <arrow>           ::= <left_arrow_head>? <arrow_body> <rght_arrow_head>?
18.
19.  /* составные конструкции (не используются напрямую) */
20.
21.  <part_whole_node> ::=   "\"" <string> "\"" "as" <node_id>
22.  <node>            ::= <node_id>
23.
24.  /* составные конструкции (построчно вводит пользователь) */
25.
26.  <enduml>    ::= "@enduml"
27.  <startuml>  ::= "@startuml"
28.  <directive> ::= <startuml> | <enduml>
29.
30.  /* --- */
31.
32.  /* сторожевое условие */
33.  <condition> ::= <any_char>+
34.  /* перечисление узлов */
35.  <nodes>     ::= <any_char>+
36.
37.  <whole_node> ::= ("actor" | "boundary" |
38.                    "entity" | "control" |
39.                    "participant") <part_whole_node>
40.  <short_node> ::= ("actor" | "boundary" |
41.                    "entity" | "control" |
42.                    "participant") <node_id>
43.
44.  /* --- */
45.
46.  <connection> ::= <node> <arrow> <node>
47.
48.  /* --- */
49.
50.  <beg_fragment>  ::= ("alt" | "opt" | "loop" | "par") <condition>
51.  <else_opd>      ::= "else" <any_char>+
52.  <end_opd>       ::= "end"
53.  <fragment_body> ::= (<connection> <nln>)+ (<else_opd> <nln>)?
54.  <fragment>      ::= <beg_fragment> <nln> <fragment_body> <end_opd>
55.
56.  <beg_ref_over>  ::= "ref" "over" <nodes>
57.  <end_ref>       ::= <end_opd> "ref"
58.  <ref_over_body> ::= (<string> <nln>)+
59.  <ref_over>      ::= <beg_ref_over> <nln> <ref_over_body> <end_ref>
```

```
60.
61.  <grouping> ::= (<fragment> | <ref_over>)
62.
63.  /* --- */
64.
65.  <diagram> ::= <startuml> <nln>
66.                ((<whole_node> | <short_node> |
67.                <connection> | <grouping>) <nln>)* <enduml>
68.
```

Листинг Б.4 – Описание PlantUML для диаграммы классов

```
1.   <nln> ::= "\n"
2.   <ids_char> ::= ([A-Z] | [a-z] | [0-9] | "_")+
3.
4.   /* --- */
5.
6.   <string>  ::= <any_char>+
7.   <node_id> ::= <ids_char>+
8.
9.   <end_curly_brace> ::= "}"
10.
11.  /* Class */
12.  /* ---------------------------------------------------------------- */
13.
14.  /* стрелка */
15.  <left_arrow_head> ::= "<" | "<|" | "*" | "o"
16.  <rght_arrow_head> ::= ">" | "|>" | "*" | "o"
17.  <arrow_direction> ::= ("l" | "r" | "d" | "u") |
18.                       ("left" | "right" | "down" | "up")
19.  <arrow_body>      ::= ("." | "-")+ <arrow_direction> ("." | "-")+
20.  <arrow>           ::= <left_arrow_head>? <arrow_body> <rght_arrow_head>?
21.
22.  /* составные конструкции (не используются напрямую) */
23.
24.  <node>        ::= <node_id>
25.  <member_mark> ::= "+" | "#" | "-"
26.
27.  /* составные конструкции (построчно вводит пользователь) */
28.
29.  <enduml>    ::= "@enduml"
30.  <startuml>  ::= "@startuml"
31.  <directive> ::= <startuml> | <enduml>
32.
33.  /* --- */
34.
35.  <beg_interface>         ::= "interface" <node_id> "{"
36.  <interface_member_func> ::= "+"? <ids_char>+ "(" (<ids_char>+ ",")+ ")"
37.  <interface_body>        ::= (<interface_member_func> <nln>)+
38.  <interface>             ::= <beg_interface> <nln>
39.                             <interface_body>
40.                             <end_curly_brace>
41.
42.  <beg_class>         ::= "class" <node_id> "{"
43.  <class_member_data> ::= <member_mark> <ids_char>+ ":" <ids_char>+
44.  <class_member_func> ::= <member_mark> <ids_char>+
45.                         "(" (<ids_char>+ ",")+ ")"
46.                         ":" <ids_char>+
47.  <class_member>      ::= <class_member_data> | <class_member_func>
48.  <class_body>        ::= (<class_member> <nln>)+
```

```
49.  <class>            ::= <beg_class> <nln> <class_body> <end_curly_brace>
50.
51.  <beg_enum>   ::= "enum" <node_id> "{"
52.  <enum_value> ::= <ids_char>+
53.  <enum_body>  ::= (<enum_value> <nln>)+
54.  <enum>       ::= <beg_enum> <nln> (<enum_value> <nln>)+ <end_curly_brace>
55.
56.  <short_node> ::= ("class" | "enum" | "interface") <node_id>
57.  <whole_node> ::= <class> | <enum> | <interface>
58.
59.  /* --- */
60.
61.  <connection> ::= <node> <arrow> <node>
62.
63.  /* --- */
64.
65.  <diagram> ::= <startuml> <nln>
66.                ((<whole_node> | <short_node> |
67.                <connection>) <nln>)*
68.                <enduml>
69.
```

Листинг Б.5 – Описание PlantUML для диаграммы потока экранов

```
1.   <nln> ::= "\n"
2.   <ids_char> ::= ([A-Z] | [a-z] | [0-9] | "_")+
3.
4.   <string>  ::= <any_char>+
5.   <node_id> ::= <ids_char>+
6.
7.   <end_curly_brace> ::= "}"
8.
9.   /* ------------------------------------------------------------------ */
10.
11.  /* стрелка */
12.  <left_arrow_head> ::= "<"
13.  <rght_arrow_head> ::= ">"
14.  <arrow_direction> ::= ("l" | "r" | "d" | "u") |
15.                        ("left" | "right" | "down" | "up")
16.  <arrow_body>      ::= "-"+ <arrow_direction> "-"+
17.  <arrow>           ::= <left_arrow_head>? <arrow_body> <rght_arrow_head>?
18.
19.  /* составные конструкции (не используются напрямую) */
20.
21.  <part_whole_node> ::= "\"" <string> "\"" "as" <node_id>
22.  <node>            ::= <node_id> | "[*]"
23.
24.  /* составные конструкции (построчно вводит пользователь) */
25.
26.  <enduml>    ::= "@enduml"
27.  <startuml>  ::= "@startuml"
28.  <directive> ::= <startuml> | <enduml>
29.
30.  /* --- */
31.
32.  <whole_node> ::= ("state") <part_whole_node>
33.  <short_node> ::= ("state") <node_id>
34.
35.  /* --- */
36.
```

```
37.  <connection> ::= <node> <arrow> <node>
38.
39.  /* --- */
40.
41.  <diagram> ::= <startuml> <nln>
42.                ((<whole_node> | <short_node> |
43.                <connection>) <nln>)*
44.                <enduml>
```

ПРИЛОЖЕНИЕ В

Примеры текстов диаграмм

Листинг В.1 – Текст диаграммы вариантов использования

```
1.  @startuml
2.
3.  actor "User" as User
4.  actor "Student" as Student
5.  actor "Teacher" as Teacher
6.
7.  usecase (Registation) as Reg
8.  usecase (Authorization) as Auth
9.
10. usecase (Create a course) as CreateCourse
11. usecase (Prepare lectures) as PrepareLectures
12. usecase (Create tests) as CreateTests
13.
14. usecase (Take a course) as TakeCourse
15. usecase (Take a course) as TakeCourse
16. usecase (Pay) as Pay
17.
18. ' ------------------------------------------
19.
20. Student -u-|> User
21. Teacher -r-|> User
22.
23. User -d-> Reg
24. User -d-> Auth
25. Teacher -d-> CreateCourse
26. Student --> TakeCourse
27. TakeCourse <.r. Pay : <<extend>>
28.
29. CreateCourse .d.> PrepareLectures : <<include>>
30. CreateCourse .u.> CreateTests : <<include>>
31.
32. @enduml
```

Листинг В.2 – Текст диаграммы пригодности

```
1.  @startuml
2.
3.  actor "Teacher" as Teacher
4.
5.  boundary "    Teacher\ncourse window" as TeacherCourseWin
6.  boundary "     Lecture\neditor window"      as LectureEditorWin
7.  boundary "Error window"               as ErrWin #red
8.
9.  control "Display lecture\n editor window" as DisplayLectureEditorWin
10. control "Display course window"          as DisplayTeacherCourseWin
11. control "Add lecture to course"          as AddLectureToCourse
12.
13. entity "Database" as DbFacade
14.
15. ' ------------------------------------------------------------------
```

```
16.
17.  Teacher -- TeacherCourseWin
18.  Teacher -- LectureEditorWin
19.  Teacher -- ErrWin
20.
21.  TeacherCourseWin -r-> DisplayLectureEditorWin : prepare lec\nbutton pressed
22.  DisplayLectureEditorWin -l-> LectureEditorWin
23.
24.  LectureEditorWin --> AddLectureToCourse : add button pressed
25.  AddLectureToCourse -l-> DbFacade
26.
27.  AddLectureToCourse --> ErrWin : already a lecture\nwith this title
28.  AddLectureToCourse --> DisplayTeacherCourseWin : successfully
29.
30.  DisplayTeacherCourseWin -d-> TeacherCourseWin
31.
32.  @enduml
```

Листинг В.3 – Текст диаграммы последовательности

```
1.   @startuml
2.
3.   actor "Teacher"             as Teacher
4.
5.   boundary "TeacherCourseWin" as TeacherCourseWin
6.   boundary "LectureEditorWin" as LectureEditorWin
7.   boundary "ErrorWin"         as ErrWin #red
8.
9.   entity "DbFacade"           as DbFacade
10.
11.  ' ------------------------------------------------------------
12.
13.  ref over Teacher
14.      Authorization
15.  end ref
16.
17.  Teacher -> TeacherCourseWin : prepare_lecture_button_pressed()
18.  TeacherCourseWin -> LectureEditorWin : show()
19.  LectureEditorWin --> Teacher
20.
21.  Teacher -> LectureEditorWin : enter_content()
22.  Teacher -> LectureEditorWin : add_button_pressed()
23.
24.  LectureEditorWin -> DbFacade : insert_lecture()
25.  DbFacade --> LectureEditorWin : return result
26.
27.  alt success
28.      LectureEditorWin -> TeacherCourseWin : show()
29.      TeacherCourseWin --> Teacher
30.  else
31.      LectureEditorWin -> ErrWin : show()
32.      ErrWin --> Teacher
33.  end
34.
35.  @enduml
```

Листинг В.4 – Текст диаграммы классов

```
@startuml

' domain
' ------------------------------------------------

class User {
    +name : String
}

class Teacher {
}

class Lecture {
}
class Test {
}

class Course {
    +title : String
    +lectures : List<Lecture>
    +tests : List<Test>
}

' logic
' ------------------------------------------------

class DbFacade {
    +select_courses() : List<Course>
    +select_lectures() : List<Lecture>
    +select_tests() : List<Test>
    +insert_lecture() : Void
    +insert_test() : Void
}

class Window {
    +show() : Void

    -db_facade : DbFacade
    -user : User
}

class LectureEditorWin {
    +enter_content() : Void
    +add_button_pressed() : Void

    -lecture : Lecture
}

class TeacherCourseWin {
    +prepare_lecture_button_pressed() : Void
    +prepare_test_button_pressed() : Void
}

class CourseWin {
    +lecture_details_button_pressed() : Void
    +test_details_button_pressed() : Void
    -course : Course
}
```

```
59.
60.  class MainWin {
61.      +course_details_button_pressed() : Void
62.      -courses : List<Course>
63.  }
64.
65.  ' conns
66.  ' ------------------------------------------------
67.
68.  MainWin --|> Window
69.  CourseWin --|> Window
70.  LectureEditorWin --|> Window
71.  TeacherCourseWin --|> CourseWin
72.
73.  ' ***
74.
75.  MainWin -r-> TeacherCourseWin
76.  TeacherCourseWin --> LectureEditorWin
77.
78.  LectureEditorWin o-- Lecture
79.
80.  CourseWin o-- Course
81.
82.  User <|-- Student
83.  User <|-- Teacher
84.
85.  Window *-- User
86.  Window o-r- DbFacade
87.
88.  MainWin *-d- Course
89.
90.  Course *-- Lecture
91.  Course *-r- Test
92.
93.  ' ***
94.
95.  DbFacade --> Course
96.  DbFacade --> Lecture
97.  DbFacade --> Test
98.
99.  @enduml
```

Листинг В.5 – Текст диаграммы потока экранов

```
1.   @startuml
2.
3.   state "Auth window\n<img:layouts/3.png>" as AuthWindow
4.   state "Main window\n<img:layouts/0.png>" as MainWindow
5.   state "Reference window\n<img:layouts/2.png>" as ReferenceWindow
6.   state "Transport window\n<img:layouts/1.png>" as TransportWindow
7.
8.   [*] --> AuthWindow
9.   AuthWindow --> [*] : закрыли\nокно
10.  AuthWindow : Пользователь может:
11.  AuthWindow : - ввести пароль
12.  AuthWindow : - закрыть окно
13.
14.  AuthWindow -> MainWindow : выполнен вход в систему
15.  MainWindow : Пользователь может:
16.  MainWindow : - перейти на вкладку "Справочники"
```

```
17.  MainWindow : …
18.
19.  MainWindow --> [*] : закрыли\nокно
20.  MainWindow --> ReferenceWindow : переход на\nвкладку "Справочники"
21.  ReferenceWindow : Пользователь может:
22.  ReferenceWindow : - выбрать "Название справочника"
23.  ReferenceWindow : …
24.
25.  ReferenceWindow --> TransportWindow : выбран\n"Виды транспорта"
26.  TransportWindow : Пользователь может:
27.  TransportWindow : …
28.
29.  @enduml
```

ПРИЛОЖЕНИЕ Г

Перевод структур данных в JSON

Листинг Г.1 – Пример диаграммы классов на PlantUML

```
@startuml

class Client {
    +name : String
    +rating : Number
}

class Order {
    +time : Time
    +client : Client
    +goods : List<Good>
}

class Good {
    +name : String
    +price : Number
}

Order o-- Client
Order *-- Good

@enduml
```

Листинг Г.2 – Результат преобразования структуры в JSON

```
{
  "id": "class_dia",
  "nodes": [
    {
      "datas": [
        {
          "mark": "+",
          "name": "name",
          "type": "String"
        }
      ],
      "funcs": [],
      "id": "Client",
      "inn_edges": [
        {
          "id": "edge_1"
        }
      ],
      "name": "Client",
      "out_edges": [],
      "type": "class"
    },
    {
      "datas": [
        {
```

```
            "mark": "+",
            "name": "name",
            "type": "String"
          },
          {
            "mark": "+",
            "name": "price",
            "type": "Number"
          }
        ],
        "funcs": [],
        "id": "Good",
        "inn_edges": [
          {
            "id": "edge_2"
          }
        ],
        "name": "Good",
        "out_edges": [],
        "type": "class"
      },
      {
        "datas": [
          {
            "mark": "+",
            "name": "time",
            "type": "Time"
          },
          {
            "mark": "+",
            "name": "client",
            "type": "Client"
          },
          {
            "mark": "+",
            "name": "goods",
            "type": "List<Good>"
          }
        ],
        "funcs": [],
        "id": "Order",
        "inn_edges": [],
        "name": "Order",
        "out_edges": [
          {
            "id": "edge_1"
          },
          {
            "id": "edge_2"
          }
        ],
        "type": "class"
      }
    ],
    "edges": [
      {
        "beg": {
          "id": "Order"
        },
        "end": {
          "id": "Client"
```

```
87.         },
88.         "id": "edge_1",
89.         "name": "",
90.         "type": "aggregation"
91.       },
92.       {
93.         "beg": {
94.           "id": "Order"
95.         },
96.         "end": {
97.           "id": "Good"
98.         },
99.         "id": "edge_2",
100.        "name": "",
101.        "type": "composition"
102.      }
103.    ]
104. }
```

ПРИЛОЖЕНИЕ Д

Дипломы, сертификаты

65
ВСЕРОССИЙСКАЯ
НАУЧНАЯ
КОНФЕРЕНЦИЯ
МФТИ

Диплом

награждается

Артемьев Леонид Сергеевич

за лучший устный студенческий доклад в рамках
65-й Всероссийской научной конференции МФТИ
в честь 115-летия Л.Д. Ландау
в секции алгоритмов и технологий программирования

Ректор МФТИ
Д.В. Ливанов

Москва, 2023


65
ВСЕРОССИЙСКАЯ
НАУЧНАЯ
КОНФЕРЕНЦИЯ
МФТИ
Сертификат участника
настоящий сертификат подтверждает, что
Артемьев Л. С.
принял(а) участие в 65-й Всероссийской научной конференции МФТИ в честь 115-летия Л.Д. Ландау в секции алгоритмов и технологий программирования с докладом «Внутреннее представление диаграмм учебной среды объектно-ориентированного моделирования»
Ректор МФТИ
Д.В. Ливанов
Москва, 2023

СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ | SIBERIAN FEDERAL UNIVERSITY

ПРОСПЕКТ СВОБОДНЫЙ

СЕРТИФИКАТ

подтверждает, что

Артемьев Леонид Сергеевич

принял(-а) **заочное участие** в XIX Международной научной конференции студентов, аспирантов и молодых ученых «ПРОСПЕКТ СВОБОДНЫЙ – 2023»

Руководитель направления
по молодежной науке
Офиса развития научной
деятельности СФУ

К. А. Кистерский

№ 27650 -2023

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
«22» 06 2023 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

«Структуры данных учебной среды
объектно-ориентированного моделирования»

09.04.01 «Информатика и вычислительная техника»

09.04.01.04 «Технология разработки программного обеспечения»

Научный руководитель	Н.Сир 22.06.23	доцент, канд.тех.наук	Н.Ю.Сиротинина
	подпись, дата	должность, ученая степень	инициалы,фамилия
Выпускник	_____		Л.С. Артемьев
	подпись, дата		инициалы,фамилия
Рецензент	_____	доцент кафедры СИИ,	А.А.Латынцев
	подпись, дата	канд.тех.наук	инициалы,фамилия
		должность, ученая степень	
Консультант	Васильев 22.06.23	старший преподаватель	В.С. Васильев
	подпись, дата	должность, ученая степень	инициалы,фамилия
Нормоконтролер	Н.Сир 22.06.23		Н.Ю.Сиротинина
	подпись, дата		инициалы,фамилия

Красноярск 2023