МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пензенский государственный университет»

Кафедра «Математическое обеспечение и применение ЭВМ»

ОТЧЕТ

по учебной (ознакомительной) практике

Выполнила: Лексина А. О

Направление: 09.03.02 «Информационные системы и технологии»

Группа: 21ВИ1

Руководитель практики: Михалев А. Г.

Работа защищена

Оценка

Пенза, 2022

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Математическое обеспечение и применение ЭВМ»

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на учебную (ознакомительную) практику

Задание 1

1. Исследовать современные технологии объектно-ориентированного анализа и проектирования информационных систем. Изучить нотацию и семантику языка UML. Сделать обзор особенностей применения данного языка в качестве средства объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

2. Провести анализ предметной области **«Продажа авиационных билетов»**. Для заданной предметной области построить диаграмму вариантов использования и диаграмму последовательности.

Задание 2

Изучить информационную технологию обработки текстовой информации и технической документации. Сделать обзор ключевых возможностей и особенностей применения данной технологии в различных программных средах. Изучить стандарт ГОСТ 2.105-95 «Общие требования к текстовым документам». Оформить отчет по практике в соответствии с ГОСТ 2.105-95 «Общие требования к текстовым документам».

Задание 3. Получить сертификат открытого университета ИНТУИТ (Pascal, UML).

Срок выполнения индивидуального задания 11 июля 2022 г.

Дата выдачи задания «28» июня 2022 г.

Дата защиты отчета «____» _____

Руководитель _____ А.Г. Михалев

Задание получил «28» июня 2022 г.

Студент _____ Лексина А.О.

Содержание

Введение

На современном этапе развития общества совершенствование многих видов деятельности неразрывно связано с формализацией знаний, одним из ключевых моментов которой является моделирование изучаемых явлений и объектов. Применение метода моделирования позволяет показать универсальность математических уравнений и алгоритмов, дает возможность унифицировать описания разнообразных по своей природе процессов.

При разработке сложных систем, мы сталкиваемся с проблемой представления данной системы как единого целого. Эту проблему способно решить моделирование. В процессе моделирования мы разбиваем сложную задачу на более простые, с акцентом на каком-либо одном аспекте системы, тем самым, упрощая процесс восприятия информации человеком. От простоты поставленной задачи зависит, будет ли использоваться формальное или неформальное моделирование при разработке. При рассмотрении простой задачи используется неформальное моделирование, т.е. модели создаются для однократного применения, визуализируя лишь часть системы.

Моделирование позволяет сократить время выполнения проекта, с полным соответствием первоначальным требованиям заказчика, а также позволяет разработчикам в полной мере представить план системы. При использовании системы в течение продолжительного времени, может возникнуть необходимость в дальнейшей ее разработке и усложнении, но при отсутствии модели могут возникнуть проблемы с усовершенствованием данной системы.

В связи с этим значение языка UML существенно возрастает, поскольку он все более приобретает черты языка представления знаний. При этом наличие в языке UML изобразительных средств для представления структуры и поведения модели позволяет достичь адекватного представления декларативных и процедурных знаний и, что не менее важно, установить между этими формами знаний семантическое соответствие. Все эти

особенности языка UML позволяют сделать вывод о том, что он имеет самые серьезные перспективы уже в ближайшем будущем.

В данной работе мне нужно исследовать современные технологии объектно-ориентированного анализа и проектирования информационных систем. Изучить нотацию и семантику языка UML. Сделать обзор особенностей применения данного языка в качестве средства объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур. Далее мне нужно провести анализ предметной области **«Продажа авиационных билетов»**. Для заданной предметной области построить диаграмму вариантов использования и диаграмму последовательности. Изучить информационную технологию обработки текстовой информации и технической документации. Сделать обзор ключевых возможностей и особенностей применения данной технологии в различных программных средах. Изучить стандарт ГОСТ 2.105-95 «Общие требования к текстовым документам». Оформить отчет по практике в соответствии с ГОСТ 2.105-95 «Общие требования к текстовым документам». Получить сертификат открытого университета ИНТУИТ.

1. Объектно-ориентированный анализ и проектирование информационных систем.

Компьютерные и информационные технологии без преувеличения можно назвать наиболее динамичной областью современных знаний, которые концентрируют в себе самые последние достижения в сфере науки и техники. Появление новых моделей процессоров и комплектующих, версий операционных систем и программного обеспечения происходит на фоне постоянного усложнения не только отдельных физических и программных компонентов, но и лежащих в их основе концепций. Разработка и совершенствование информационных систем приводит к необходимости поддержания единого стиля для различных версий программ при их постоянной доработке и модификации.

Трудоемкость создания современных приложений на начальных этапах проекта, как правило, оценивается значительно ниже реально затрачиваемых усилий, что служит причиной незапланированных расходов и затягивания окончательных сроков готовности программ. В процессе разработки приложений изменяются функциональные требования заказчика, что еще более отдаляет момент окончания работы программистов. Увеличение размеров программ вынуждает привлекать сверхштатных программистов, что, в свою очередь, требует дополнительных ресурсов для организации их согласованной работы. В разработке и внедрении современных корпоративных информационных систем принимает участие множество специалистов различной квалификации, для которых единообразное понимание архитектуры и функциональности является серьезной проблемой.

Таким образом, все эти особенности приводят к настоятельной необходимости моделирования структуры и процесса функционирования программных систем до начала написания соответствующего кода. При этом непременным условием успешного завершения проекта становится построение предварительной модели программной системы.

Модель (model) - абстракция физической системы, рассматриваемая с определенной точки зрения и представленная на некотором языке или в графической форме.

С точки зрения общих принципов системного анализа одна и та же физическая система может быть представлена несколькими моделями. При этом назначение отдельной модели системы определяется характером решаемой проблемы. Основное требование к модели программной системы - она должна быть понятна заказчику и всем специалистам проектной группы, включая бизнес-аналитиков и программистов. Именно для разработки такой нотации потребовались усилия группы специалистов ведущих фирм производителей программного и аппаратного обеспечения, которые привели к появлению языка UML.

Разработка и использование моделей языка UML осуществляется в рамках общей концепции объектно-ориентированного анализа и проектирования, которая, в свою очередь, является обобщением методологии объектно-ориентированного программирования.

1.1. Нотации и семантика UML

Семантика – раздел языкознания, изучающий значение единиц языка, прежде всего его слов и словосочетаний.

Синтаксис – способы соединения слов и их форм в словосочетания и предложения, соединения предложений в сложные предложения, способы создания высказываний как части текста.

Таким образом, применительно к UML, семантика и синтаксис определяют стиль изложения (построения моделей), который объединяет естественный и формальный языки для представления базовых понятий (элементов модели) и механизмов их расширения.

Нотация представляет собой графическую интерпретацию семантики для ее визуального представления.

В UML определено три **типа сущностей**:

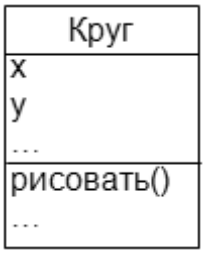
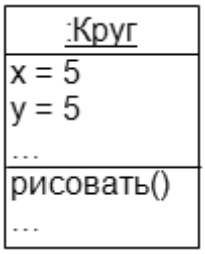
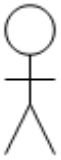
- структурная – абстракция, являющаяся отражением концептуального или физического объекта;

- группирующая – элемент, используемый для некоторого смыслового объединения элементов диаграммы;

- поясняющая (аннотационная) – комментарий к элементу диаграммы.

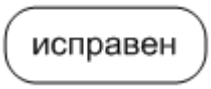
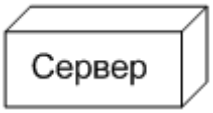
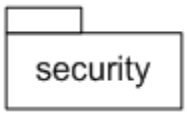
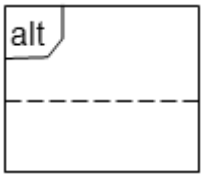
В таблице 1.1 приведено краткое описание основных сущностей, используемых в графической нотации, и основные способы их отображения.

Таблица 1.1. Сущности

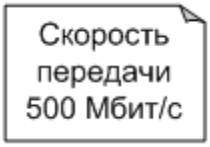
Тип	Наименование	Обозначение	Определение (семантика)
1	2	3	4
Структурная	Класс (class)		Множество объектов, имеющих общую структуру и поведение
	Объект (object)		Абстракция реальной или воображаемой сущности с четко выраженными концептуальными границами, индивидуальностью (идентичностью), состоянием и поведением. С точки зрения UML объекты являются экземплярами класса (экземплярами сущности)
	Актер (actor)		Внешняя по отношению к системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или

		службы пути	решения частных задач. Таким образом актер – это внешний источник или приемник информации

Продолжение таблицы 2

1	2	3	4
Группирующая	Вариант использования (use case)		Описание выполняемых системой действий, которая приводит к значимому для актера результату
	Состояние (state)		Описание момента в ходе жизни сущности, когда она удовлетворяет некоторому условию, выполняет некоторую деятельность или ждет наступления некоторого события
	Кооперация (collaboration)		Описание совокупности экземпляров актеров, объектов и их взаимодействия в процессе решения некоторой задачи
	Компонент (component)		Физическая часть системы (файл), в том числе модули системы, обеспечивающие реализацию согласованного набора интерфейсов
	Интерфейс (interface)		Совокупность операций, определяющая сервис (набор услуг), предоставляемый классом или компонентом
	Узел (node)		Физическая часть системы (компьютер, принтер и т. д.), предоставляющая ресурсы для решения задачи
	Пакет (package)		Общий механизм группировки элементов. В отличие от компонента, пакет – чисто концептуальное (абстрактное) понятие. Частными случаями пакета являются система и модель
	Фрагмент (fragment)		Область специфического взаимодействия экземпляров актеров и объектов. Любая диаграмма в UML также является фрагментом – фрагментом (частью) проекта.
	Раздел деятельности (activity partition)		Группа операций (зона ответственности), выполняемых одной сущностью (актером, объектом, компонентом, узлом и т.д.)

Окончание таблицы

1	2	3	4
Поясняющая	Прерываемый регион (interruptible activity region)		Группа операций, обычная последовательность выполнения которых может прервана в результате наступления нестандартной ситуации
	Примечание (comment)		Комментарий к элементу. Присоединяется к комментируемому элементу штриховой линией

В некоторых источниках, в частности, выделяют также поведенческие сущности взаимодействия и конечные автоматы, но с логической точки зрения их следует отнести к диаграммам.

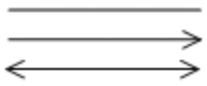
Некоторые из приведенных выше сущностей в соответствии с прициpами иерархического упорядочивания и декомпозиции подразумевают их подробное описание на диаграммах декомпозиции. На диаграмме верхнего уровня они помечаются особым значком или меткой.

Таблица 1.2. Декомпозируемые сущности

Наименование	Обозначение	Диаграмма
Скрытое составное состояние		Диаграмма автоматов
Скрытый фрагмент взаимодействия		Диаграмма последовательности
Деятельность		Диаграмма деятельности

В следующей таблице приведено описание всех видов **отношений** UML, используемых на диаграммах для указания связей между сущностями.

Таблица 1.3. Отношения

Наименование	Обозначение	Определение (семантика)
Ассоциация (association)		Отношение, описывающее значимую связь между двумя и более сущностями. Наиболее общий вид отношения
Агрегация (aggregation)		Подвид ассоциации, описывающей связь "часть"–"целое", в котором "часть" может существовать отдельно от "целого". Ромб указывается со стороны "целого". Отношение указывается только между сущностями одного типа
Композиция (composition)		Подвид агрегации, в которой "части" не могут существовать отдельно от "целого". Как правило, "части" создаются и уничтожаются одновременно с "целым"
Зависимость (dependency)		Отношение между двумя сущностями, в котором изменение в одной сущности (независимой) может влиять на состояние или поведение другой сущности (зависимой). Со стороны стрелки указывается независимая сущность
Обобщение (generalization)		Отношение между обобщенной сущностью (предком, родителем) и специализированной сущностью (потомком, дочкой). Треугольник указывается со стороны родителя. Отношение указывается только между сущностями одного типа
Реализация (realization)		Отношение между сущностями, где одна сущность определяет действие, которое другая сущность обязуется выполнить. Отношения используются в двух случаях: между интерфейсами и классами (или компонентами), между вариантами использования и кооперациями. Со стороны стрелки указывается сущность, определяющее действие (интерфейс или вариант использования)

Для ассоциации, агрегации и композиции может указываться кратность, характеризующая общее количество экземпляров сущностей, участвующих в отношении. Она, как правило, указывается с

каждой стороны отношения около соответствующей сущности. Кратность может указываться следующими способами:

- любое количество экземпляров, в том числе и ни одного

- целое неотрицательное число – кратность строго фиксирована и равна указанному числу (например: 1, 2 или 5);

- диапазон целых неотрицательных чисел "первое число .. второе число" (например: 1..5, 2..10 или 0..5);

- диапазон чисел от конкретного начального значения до произвольного конечного "первое число .. *" (например: 1..*, 5..* или 0..*);

- перечисление целых неотрицательных чисел и диапазонов через запятую (например: 1, 3..5, 10, 15..*).

Если кратность не указана, то принимается ее значение, равное 1. Кратность экземпляров сущностей, участвующих в зависимости, обобщении и реализации, всегда принимается равной 1.

В следующей таблице приведено описание механизмов расширения, применяемых для уточнения семантики сущностей и отношений. В общем случае, механизм расширения представляет собой строку текста, заключенную в скобки или кавычки.

Таблица 1.4. Механизмы расширения

Наименование	Обозначение	Определение (семантика)
Стереотип (stereotype)	« »	Обозначение, уточняющее семантику элемента нотации (например: зависимость со стереотипом «include» рассматривается, как отношение включения, а класс со стереотипом «boundary» – граничный класс)

Сторожевое условие (guard condition)	[ ]	Логическое условие (например: [A > B] или [идентификация выполнена])
Ограничение (constraint)	{ }	Правило, ограничивающее семантику элемента модели (например, {время выполнения менее 10 мс})
Помеченное значение (tagged value)	{ }	Новое или уточняющее свойство элемента нотации (например: {version = 3.2})

Помимо стереотипов, указываемых в виде строки текста в кавычках, на диаграммах могут использоваться графические стереотипы. На следующем рисунке приведены примеры стандартного и стереотипного отображения класса-сущности (см рис. 1.1).

а) стандартное обозначение

б) стандартное обозначение с текстовым стереотипом

в) графический стереотип

Рис. 1.1. - Примеры стандартного и стереотипного отображения класса

Диаграмма представляет собой группировку элементов нотации для отображения некоторого аспекта разрабатываемой информационной системы. Диаграммы представляют собой, как правило, связный граф, в котором сущности являются вершинами, а отношения – дугами. В следующей таблице дана краткая характеристика диаграмм UML.

Таблица 1.5 Диаграммы

Диаграмма			Назначение	Тип диаграммы (модели ПС)		
				по степени физической реализации	по отображению динамики	по отображаемому аспекту
1			2	3	4	5
Вариантов использования			Отображает функции системы, взаимодействие между актерами и функциями	Логическая	Статическая	Статическая
Классов			Отображает набор классов, интерфейсов и отношений между ними	Логическая или физическая	Статическая	Функционально-информационная
Поведения	Состояний		Отображает состояния сущности и переходы между ними в процессе ее жизненного цикла	Логическая	Динамическая	Поведенческая
	Деятельности		Отображает бизнес-процессы в системе (описание алгоритмов поведения)			
	Взаимодействия	Последовательности	Отображает последовательность передачи сообщений между объектами и актерами			
		Последова-	Аналогична диаграмме			

		тельности	последовательности, но основной акцент делается на структуру взаимодействия между объектами			
Реализации	Компонентов		Отображает компоненты системы (программы, библиотеки, таблицы и т.д.) и связи между ними	Физическая	Статическая	Компонентная
	Развертывания		Отображает размещение компонентов по узлам сети, а также ее конфигурацию			

Стандарт UML 2.x определяет также дополнительные, узкоспециализированные диаграммы:

- диаграмму объектов (object diagram) - аналогична диаграмме классов, но вместо классов отображаются объекты;

- диаграмму синхронизации (timing diagram) - описывает состояния объекта с течением времени;

- композитную структурную диаграмму (composite structure diagram) - описывает порты (включая интерфейсы) класса для взаимодействия с другими классами;

- профильную диаграмму (profile diagram) - аналогична диаграмме пакетов с описанием классов, входящих в них;

- обзорную диаграмму взаимодействия (interaction overview diagram) - аналогична диаграмме последовательности, но со скрытыми фрагментами взаимодействия (фрагментами с меткой ref). Представляет собой контекстную (концептуальную) диаграмму последовательности, элементы которой будут конкретизированы на отдельных диаграммах декомпозиции [].

В целях укрупненного концептуального представления внутренней архитектуры системы большинство Case-средств при построении диаграммы классов допускает использование устоявшихся графических стереотипов для так называемых классов анализа. Такая диаграмма называется диаграммой классов анализа, но не относится к перечню диаграмм, определенных стандартом UML.

1.2. История создания UML

Объектно-ориентированные языки моделирования появились в период с середины 70-х до конца 80-х годов, когда исследователи, поставленные перед необходимостью учитывать новые возможности объектно-ориентированных языков программирования и требования, предъявляемые все более сложными приложениями, вынуждены были начать разработку различных альтернативных подходов к анализу и проектированию.

С 1989 по 1994 год число различных объектно-ориентированных методов возросло с десяти более чем до пятидесяти. Тем не менее, многие пользователи испытывали затруднения при выборе языка моделирования, который бы полностью соответствовал их потребностям, что послужило причиной так называемой «войны методов». В результате этих войн появилось новое поколение методов, среди которых особое значение приобрели языки **Booch**, созданный ***Грейди Бучем*** (Grady Booch), **OOSE** (Object-Oriented Software Engineering),

разработанный ***Айваром Джекобсоном*** (Ivar Jacobson) и **OMT** (Object Modeling Technique), автором которого является ***Джеймс Рамбо*** (James Rumbaugh). Кроме того, следует упомянуть языки **Fusion, Шлаера-Меллора** (Shlaer-Mellor) и **Коада-Йордона** (Coad-Yourdon). Каждый из этих методов можно считать вполне целостным и законченным, хотя любой из них имеет не только сильные, но и слабые стороны.

Выразительные возможности метода Буча особенно важны на этапах проектирования и конструирования модели. OOSE великолепно приспособлен для анализа и формулирования требований, а также для высокоуровневого проектирования. OMT-2 оказался особенно полезным для анализа и разработки информационных систем, ориентированных на обработку больших объемов данных.

Критическая масса новых идей начала формироваться к середине 90-х годов, когда **Грейди Буч** (компания Rational Software Corporation), **Айвар Джекобсон** (Objectory) и **Джеймс Рамбо** (General Electric) предприняли попытку объединить свои методы, уже получившие мировое признание как наиболее перспективные в данной области. Являясь основными авторами языков **Booch, OOSE и OMT**, партнеры попытались создать новый, унифицированный язык моделирования и руководствовались при этом тремя соображениями.

Во-первых, все три метода, независимо от желания разработчиков, уже развивались во встречном направлении. Разумно было продолжать эту эволюцию вместе, а не по отдельности, что помогло бы в будущем устранить нежелательные различия и, как следствие, неудобства для пользователей.

Во-вторых, унифицировав методы, проще было привнести стабильность на рынок инструментов объектно-ориентированного моделирования, что дало бы возможность положить в основу всех проектов

единый зрелый язык, а создателям инструментальных средств позволило бы сосредоточиться на более продуктивной деятельности.

Наконец, следовало полагать, что подобное сотрудничество приведет к усовершенствованию всех трех методов и обеспечит решение задач, для которых любой из них, взятый в отдельности, был не слишком пригоден.

Начав унификацию, авторы поставили перед собой три главные цели:

– моделировать системы целиком, от концепции до исполняемого артефакта, с помощью объектно-ориентированных методов;

– решить проблему масштабируемости, которая присуща сложным системам, предназначенным для выполнения ответственных задач;

– создать такой язык моделирования, который может использоваться не только людьми, но и компьютерами.

Изобретение языка для объектно-ориентированного анализа и проектирования не слишком отличается от разработки языка программирования. *Во-первых*, требовалось ограничить задачу. Следует ли включать в язык возможность спецификации требований? Должен ли язык позволять визуальное программирование? *Во-вторых*, было необходимо найти точку равновесия между выразительной мощью и простотой. Слишком простой язык ограничил бы круг решаемых с его помощью задач, а слишком сложный мог ошеломить неискушенного разработчика. Кроме того, при объединении существующих методов приходилось учитывать наличие уже разработанных с их помощью продуктов. Внесение слишком большого числа изменений могло бы оттолкнуть уже имевшихся пользователей, а сопротивляясь развитию языка, авторы потеряли бы возможность привлекать новых пользователей и делать язык более простым и удобным для

применения. Создавая UML, разработчики старались найти оптимальное решение этих проблем.

Официально создание UML началось в октябре 1994 года, когда Рамбо перешел в компанию Rational Software, где работал Буч. Первоначальной целью было объединение методов Буча и OMT. Первая пробная **версия 0.8 Унифицированного Метода** (Unified Method), как его тогда называли, появилась в ***октябре 1995 года***. Приблизительно в это же время в компанию Rational перешел Джекобсон, и проект UML был расширен с целью включить в него язык OOSE. В результате совместных усилий в ***июне 1996 года*** вышла **версия 0.9 языка UML**. На протяжении всего года создатели занимались сбором отзывов от основных компаний, работающих в области конструирования программного обеспечения. За это время стало ясно, что большинство таких компаний сочло UML языком, имеющим стратегическое значение для их бизнеса. В результате был основан консорциум UML, в который вошли организации, изъявившие желание предоставить ресурсы для работы, направленной на создание полного определения UML.

Версия 1.0 языка появилась в результате совместных усилий компаний Digital Equipment Corporation, Hewlett Packard, I-Logix, Intellicprp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational, Texas Instruments и Unisys. **UML 1.0** оказался хорошо определенным, выразительным, мощным языком, применимым для решения большого количества разнообразных задач. В январе 1997 года он был представлен Группе по управлению объектами (Object Management Group, OMG) на конкурс по созданию стандартного языка моделирования.

Между январем и июнем 1997 года консорциум UML расширился, в него вошли практически все компании, откликнувшиеся на призыв OMG, а именно: Andersen Consulting, Ericsson, ObjecTime Limited, Platinum

Technology, Ptech, Reich Technologies, Softeam, Sterling Software и Taskon. Чтобы формализовать спецификации UML и координировать работу с другими группами, занимающимися стандартизацией, под руководством Криса Кобрина (Cris Kobryn) из компании MCI Systemhouse и Эда Эйкхолта (Ed Eykholt) из Rational была организована семантическая группа. Пересмотренная версия UML была снова представлена на рассмотрение OMG в июле 1997 года. В сентябре версия была утверждена на заседаниях Группы по анализу и проектированию и Комитета по архитектуре OMG, а 14 ноября 1997 года принята в качестве стандарта на общем собрании всех членов OMG.

Дальнейшая работа по развитию UML проводилась Группой по усовершенствованию (Revision Task Force, RTF) OMG под руководством **Криса Кобрина**. В ***июне 1998 года*** вышла **версия UML 1.2**, а ***осенью 1998*** – **UML 1.3** [].

1.3. Канонические диаграммы языка UML

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций, получивших название диаграмм.

Диаграмма (diagram) — графическое представление совокупности элементов модели. Нотация канонических диаграмм - основное средство разработки моделей на языке UML.

В нотации языка UML определены следующие виды канонических диаграмм:

- вариантов использования (use case diagram);
- классов (class diagram);
- кооперации (collaboration diagram);
- последовательности (sequence diagram);
- состояний (statechart diagram);

- деятельности (activity diagram);

- компонентов (component diagram);

- развертывания (deployment diagram).

Каждая из этих диаграмм детализирует и конкретизирует различные представления о модели сложной системы в терминах языка UML. При этом диаграмма вариантов использования представляет собой наиболее общую концептуальную модель сложной системы, которая является исходной для построения всех остальных диаграмм. Диаграмма классов - логическая модель, отражающая статические аспекты структурного построения сложной системы. Диаграммы кооперации и последовательностей представляют собой разновидности логической модели, которые отражают динамические аспекты функционирования сложной системы. Диаграммы состояний и деятельности предназначены для моделирования поведения системы. И, наконец, диаграммы компонентов и развертывания служат для представления физических компонентов сложной системы и поэтому относятся к ее физической модели.

Кроме графических элементов, которые определены для каждой канонической диаграммы, на них может быть изображена текстовая информация, которая расширяет семантику базовых элементов. В языке UML предусмотрены три специальных механизма расширения, которые включают в себя следующие конструкции:

- стереотип (stereotype) — новый тип элемента модели, который расширяет семантику модели. Стереотипы должны основываться на уже существующих и описанных в модели языка UML типах или классах;

Стереотипы предназначены для расширения семантики, но не структуры уже описанных типов или классов. Некоторые стереотипы предопределены в языке UML, другие могут быть указаны разработчиком. На диаграммах изображаются в форме текста, заключенного в угловые кавычки. Предопределенные стереотипы являются ключевыми словами

языка UML, которые используются на канонических диаграммах на языке оригинала без их перевода.

- помеченное значение (tagged value) — явное определение свойства как пары "имя – значение". В помеченном значении само имя называют тегом (tag);

Помеченные значения на диаграммах изображаются в форме строки текста специального формата, заключенного в фигурные скобки. При этом используется следующий формат записи: {тег = значение}. Теги встречаются в нотации языка UML, но их определение не является строгим, поэтому теги могут быть указаны самим разработчиком.

- ограничение (constraint) — некоторое логическое условие, ограничивающее семантику выбранного элемента модели;

Как правило, все ограничения специфицируются разработчиком. Ограничения на диаграммах изображаются в форме строки текста, заключенного в фигурные скобки. Для формальной записи ограничений предназначен специальный язык объектных ограничений (Object Constraint Language, OCL), который является составной частью языка UML [].

1.4. Особенности графического изображения диаграмм языка UML

Для диаграмм языка UML существуют три типа визуальных графических обозначений, которые важны с точки зрения заключенной в них информации:

Геометрические фигуры на плоскости, играющие роль вершин графов соответствующих диаграмм. При этом сами геометрические фигуры выступают в роли графических примитивов языка UML, а форма этих фигур (прямоугольник, эллипс) должна строго соответствовать изображению отдельных элементов языка UML (класс, вариант использования, состояние, деятельность). Графические примитивы языка UML имеют фиксированную семантику, переопределять которую пользователям не допускается. Графические примитивы должны иметь собственные имена, а, возможно, и

другой текст, который содержится внутри границ соответствующих геометрических фигур или, как исключение, вблизи этих фигур.

Графические взаимосвязи, которые представляются различными линиями на плоскости. Взаимосвязи в языке UML обобщают понятие дуг и ребер из теории графов, но имеют менее формальный характер и более развитую семантику.

Специальные графические символы, изображаемые вблизи от тех или иных визуальных элементов диаграмм и имеющие характер дополнительной спецификации (украшений).

Все диаграммы в языке UML изображаются с использованием фигур на плоскости. Отдельные элементы - с помощью геометрических фигур, которые могут иметь различную высоту и ширину с целью размещения внутри них других конструкций языка UML. Наиболее часто внутри таких символов помещаются строки текста, которые уточняют семантику или фиксируют отдельные свойства соответствующих элементов языка UML. Информация, содержащаяся внутри фигур, имеет значение для конкретной модели проектируемой системы, поскольку регламентирует реализацию соответствующих элементов в программном коде.

Пути представляют собой последовательности из отрезков линий, соединяющих отдельные графические символы. При этом концевые точки отрезков линий должны обязательно соприкасаться с геометрическими фигурами, служащими для обозначения вершин диаграмм, как принято в теории графов. С концептуальной точки зрения путям в языке UML придается особое значение, поскольку это простые топологические сущности. Отдельные части пути или сегменты могут не существовать вне содержащего их пути. Пути всегда соприкасаются с другими графическими символами на обеих границах соответствующих отрезков линий, т.е. пути не могут обрываться на диаграмме линией, которая не соприкасается ни с одним графическим символом. Как отмечалось выше, пути могут иметь в качестве

окончания или терминатора специальную графическую фигуру – значок, который изображается на одном из концов линий.

Дополнительные значки или украшения представляют собой графические фигуры фиксированного размера и формы. Они не могут увеличивать свои размеры, чтобы разместить внутри себя дополнительные символы. Значки размещаются как внутри других графических конструкций, так и вне их. Примерами значков могут служить окончания связей элементов диаграмм или графические обозначения кванторов видимости атрибутов и операций классов.

Строки текста служат для представления различных видов информации в грамматической форме. Предполагается, что каждое использование строки текста должно соответствовать синтаксису в нотации языка UML. В отдельных случаях может быть реализован грамматический разбор этой строки, который необходим для получения дополнительной информации о модели. Например, строки текста в различных секциях обозначения класса могут соответствовать атрибутам этого класса или его операциям. На использование строк накладывается условие: требуется, чтобы семантика всех допустимых символов была заранее определена в языке UML или служила предметом его расширения в конкретной модели [].

1.5. Диаграмма вариантов использования (теория)

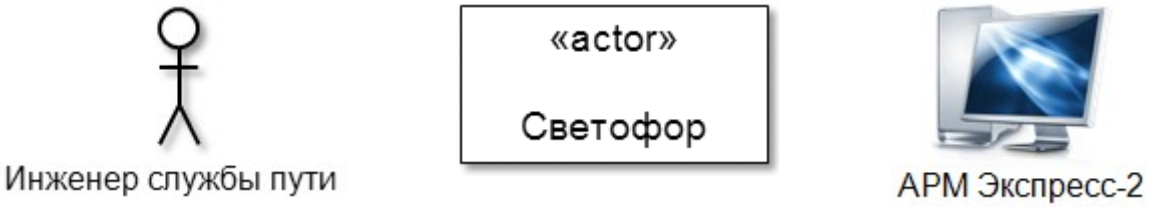
Диаграмма вариантов использования (сценариев поведения, прецедентов) является исходным концептуальным представлением системы в процессе ее проектирования и разработки. Данная диаграмма состоит из актеров, вариантов использования и отношений между ними. При построении диаграммы могут использоваться также общие элементы нотации: примечания и механизмы расширения.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (действующим лицом, актантом, актором) называется любой объект, субъект

или система, взаимодействующая с моделируемой системой извне. В свою очередь вариант использования – это спецификация сервисов (функций), которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемых системой при взаимодействии с актером. При этом в модели никак не отражается то, каким образом будет реализован этот набор действий.

В структурном подходе аналогом диаграммы вариантов использования являются диаграммы IDEF0 и DFD, вариантов использования – работы (IDEF0) и процессы (DFD), актеров – внешние сущности (DFD).

Согласно UML актера графически можно отобразить тремя способами (см. рис. 1.2.).



а) «проволочный человечек»

б) класс с текстовым стереотипом «actor»

в) произвольная иконка

Рис. 1.2. - Примеры отображения актеров

Первый способ отображения в виде «проволочного человечка» является самым распространенным.

Вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его описание, обозначающее выполнение какой-либо операции или действия (см. рис. 1.3.).

Рис. 1.3. - Примеры вариантов использования

Вариант использования, который инициализируется по запросу пользователя, представляет собой законченную последовательность

действий. Это означает, что после того, как система закончит обработку запроса актера, она должна возвратиться в состояние, в котором готова к выполнению следующих запросов.

Варианты использования могут включать в себя описание особенностей способов реализации сервиса и различных исключительных ситуаций, таких как корректная обработка ошибок системы.

Примечания предназначены для включения в диаграмму произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемой системы. В качестве такой информации могут быть комментарии разработчика и ограничения. Графически примечания отображаются прямоугольником с загнутым верхним правым уголком, внутри которого содержится текст примечания. Линия, соединяющая примечание и элемент диаграммы, называется якорем (фиксацией) (см. рис. 1.4.).

Рис. 1.4. - Пример примечания

Связи между актерами и вариантами отображаются с использованием отношений четырех видов:

- ассоциаций;
- обобщения;
- включения (зависимость со стереотипом «include»);
- расширения (зависимость со стереотипом «extend»).

Применительно к рассматриваемой диаграмме отношение ассоциации служит для обозначения взаимодействия актера с вариантом использования (см. рис. 1.5.).

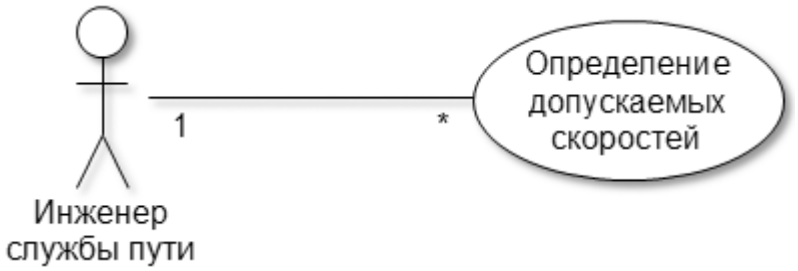


Рис. 1.5. - Пример ассоциации

Ассоциация может отображаться в виде однонаправленной или двунаправленной стрелки, показывающей направление потоков информации или управляющих сигналов.

Отношение обобщения служит для указания того факта, что некоторая сущность A может быть обобщена до сущности B. В этом случае сущность A будет являться специализацией сущности B. На диаграмме данный вид отношения можно отображать только между однотипными сущностями (между двумя вариантами использования или двумя актерами).

Графически данное отношение обозначается сплошной линией со стрелкой, в виде незакрашенного треугольника, от потомка к родителю (см. рис. 1.6.).

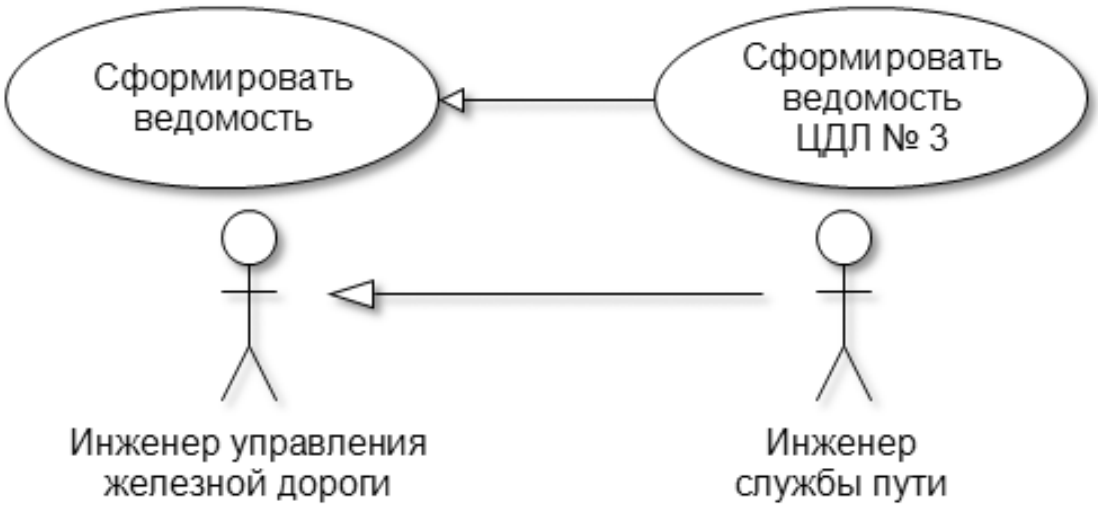


Рис. 1.6. - Примеры обобщения

Отношения включения и расширения являются частным случаем отношения зависимости и могут иметь место только между двумя вариантами использования. Они отображаются штриховой стрелкой с указанием стереотипа.

Отношение включения указывает, что некоторое заданное поведение одного варианта использования обязательно включается в качестве составного компонента в последовательность поведения другого варианта использования (см. рис. 1.7.).

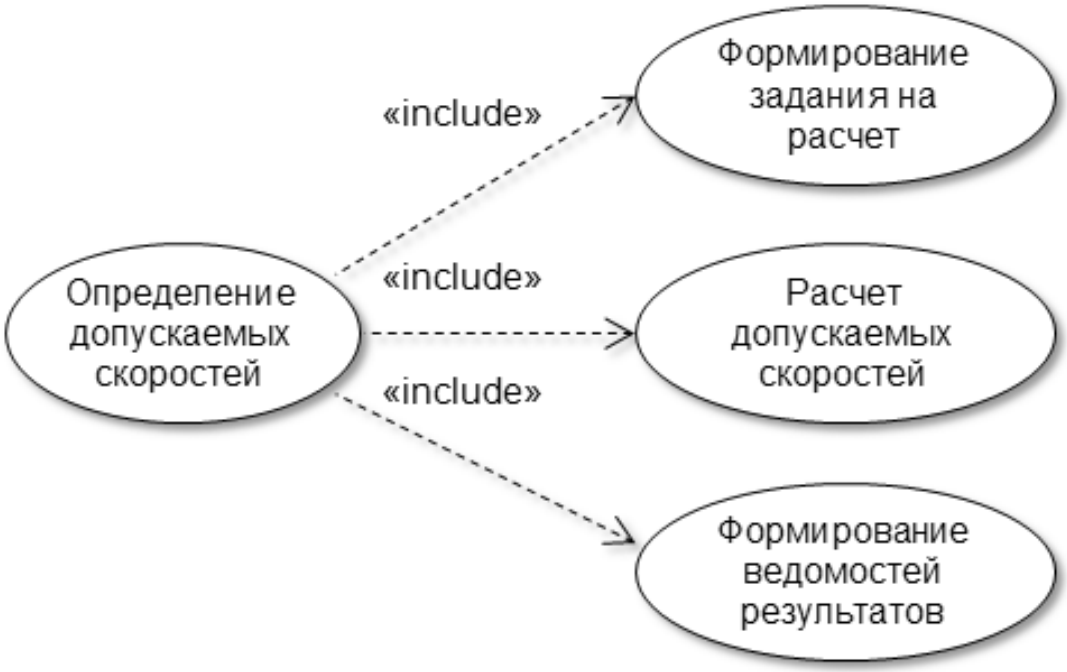


Рис. 1.7. - Пример включения

Стрелка включения должна быть направлена от базового (составного) варианта к включаемому и помечена стереотипом «include» (англ. включает) или «uses» (англ. использует).

В отличие от отношения включения, отношение расширения определяет потенциальную возможность включения поведения одного варианта использования в состав другого. Т. е. дочерний вариант использования может как вызываться, так и не вызываться родительским.

Стрелка расширения должна быть направлена от включаемого варианта к базовому и помечена стереотипом «extend» (англ. расширяет) (см. рис. 1.8.).

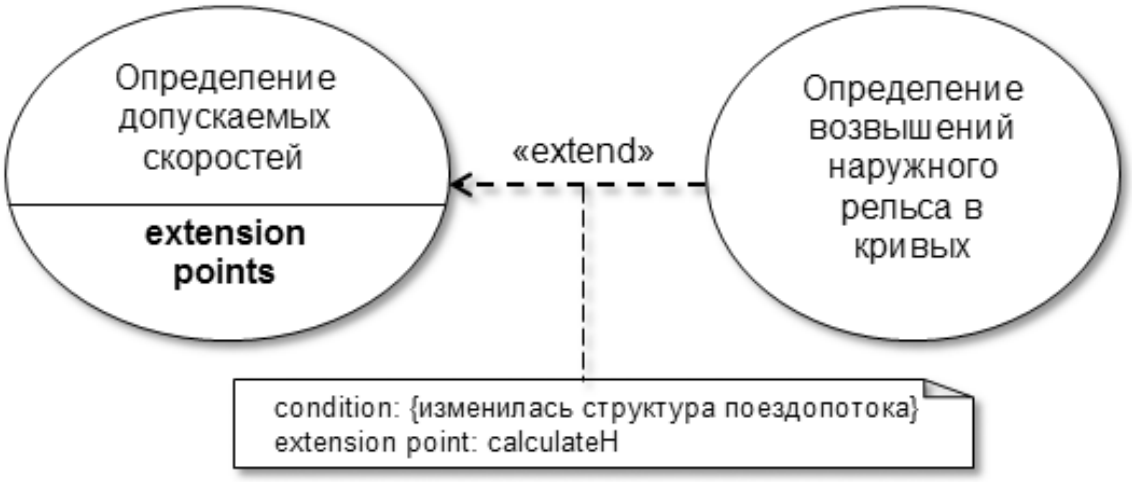


Рис. 1.8. - Пример расширения

Ввиду того, что допускаемая скорость в кривых участках пути зависит в том числе и от возвышения наружного рельса, перед определением допускаемых скоростей может потребоваться определение и установление новых возвышений, которые в свою очередь зависят от структуры пропускаемого поездопотока.

Варианты использования, которые расширяют базовый, подключаются к нему (активируются при его выполнении) через так называемые точки расширения (англ. extension points). Каждая точка расширения маркируется меткой (calculateH) и условием (англ. condition) активации. Обычно перечень точек расширения указывается в базовом варианте использования ниже горизонтальной линии [].

1.6. Диаграмма последовательности (теория)

Диаграммы последовательностей, обычно используемые разработчиками, моделируют взаимодействия между объектами в едином сценарии использования. Они иллюстрируют, как различные части системы взаимодействуют друг с другом для выполнения функции, а также порядок, в

котором происходит взаимодействие при выполнении конкретного случая использования.

Проще говоря, диаграмма последовательности показывает различные части работы системы в “последовательности”, чтобы что-то сделать.

Диаграммы последовательности следует применять тогда, когда требуется посмотреть на поведение нескольких объектов в рамках одного прецедента. Диаграммы последовательности хороши для представления взаимодействия объектов, но не очень подходят для точного определения поведения.

Если вы хотите посмотреть на поведение одного объекта в нескольких прецедентах, то примените диаграмму состояния. Если же надо изучить поведение нескольких объектов в нескольких прецедентах или потоках, не забудьте о диаграмме деятельности.

Если требуется быстро исследовать несколько вариантов взаимодействия, лучше использовать CRC-карточки, поскольку это позволяет избежать непрерывного рисования и стирания. Часто бывает удобно поработать с CRC-карточками для просмотра вариантов взаимодействия, а затем с помощью диаграмм взаимодействий фиксировать те взаимодействия, которые будут применяться позже.

Другим полезным видом диаграмм взаимодействий являются коммуникационные диаграммы, которые показывают соединения, и временные диаграммы, показывающие временные интервалы.

Схема последовательности построена таким образом, что она представляет собой временную шкалу, которая начинается сверху и постепенно опускается, чтобы отметить последовательность взаимодействий. Каждый объект имеет колонку, а сообщения, которыми обмениваются между собой, представлены стрелками.

Нотация линии жизни

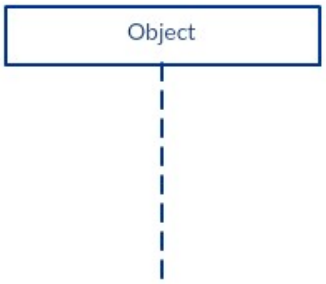


Рис. 2.1. - Нотация линии жизни

Последовательность состоит из нескольких таких обозначений линии жизнеобеспечения, которые должны быть расположены горизонтально в верхней части диаграммы. Никакие две нотации страховочной линии не должны перекрывать друг друга. Они представляют собой различные объекты или части, которые взаимодействуют друг с другом в системе во время последовательности (см рис. 2.1.).

Нотация жизненной линии с символом элемента агента используется в том случае, если конкретная диаграмма последовательности принадлежит случаю использования (см рис. 2.2.).

Рис. 2.2. - Нотация жизненной линии с символом элемента агента

Линия жизни с элементом-субъектом представляет системные данные (см рис. 2.3.). Например, в приложении “Обслуживание клиентов” организация-заказчик будет управлять всеми данными, относящимися к клиенту.Линия жизни сущности (Entity Lifeline)

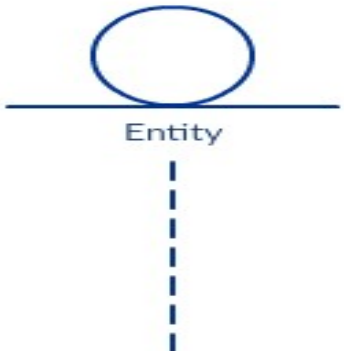


Рис. 2.3. - Линия жизни с элементом-субъектом

Линия жизни с пограничным элементом обозначает системную границу/программный элемент в системе; например, экраны пользовательского интерфейса, шлюзы базы данных или меню, с которыми взаимодействуют пользователи, являются границами (см рис. 2.4.).

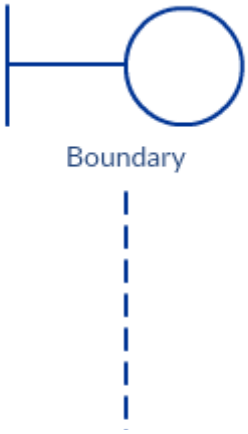


Рис. 2.4. - Линия жизни с пограничным элементом

И линия жизни с элементом контроля указывает на контролирующую организацию или менеджера. Он организует и составляет график взаимодействия между границами и субъектами и выступает в качестве посредника между ними (см рис. 2.5.).

Рис. 2.5. - Линия жизни с элементом контроля

Активационная планка – это коробка, расположенная на страховочной линии. Используется для указания на то, что объект активен (или интонирован) во время взаимодействия между двумя объектами. Длина прямоугольника указывает на продолжительность пребывания объектов в активном состоянии.

На диаграмме последовательности взаимодействие между двумя объектами происходит, когда один объект посылает сообщение другому. Использование строки активации на спасательных линиях вызывающего сообщения (объекта, отправляющего сообщение) и получателя сообщения (объекта, принимающего сообщение) указывает на то, что оба они активны/осуществляются во время обмена сообщением (см рис. 2.6.).

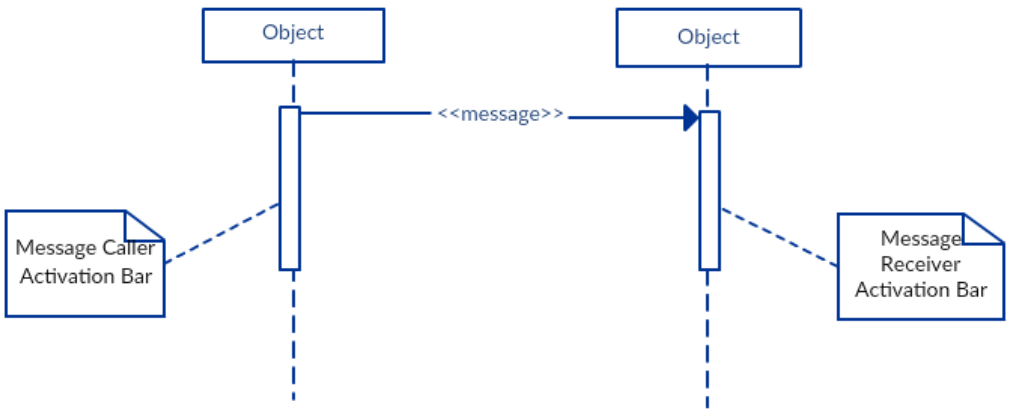


Рис. 2.6. - Диаграмма последовательности взаимодействия между двумя объектами

Стрелка от Звонящего до Получателя сообщения указывает сообщение на схеме последовательности. Сообщение может идти в любом направлении: слева направо, справа налево или обратно к самому вызывающему сообщение абоненту. В то время как вы можете описать сообщение, отправляемое с одного объекта на другой, на стрелке, с разными заголовками стрелок, вы можете указать тип отправляемого или получаемого сообщения.

Стрелка сообщения содержит описание, известное как подпись сообщения. Формат подписи этого сообщения приведен ниже. Все части, кроме имени_сообщения, являются необязательными.

Атрибут = имя_сообщения (аргументы): return_type

Синхронное сообщение

Как показано в примере полосок активации, синхронное сообщение используется, когда отправитель ждет, пока приемник обработает сообщение и вернется, прежде чем продолжить с другим сообщением. Заголовок стрелки, используемый для обозначения этого типа сообщения, является сплошным, как показано ниже (см рис. 2.7.).

Рис. 2.7. - Обозначение синхронного сообщения

Асинхронное сообщение

Асинхронное сообщение используется, когда вызывающая сторона не ждет, пока приемник обработает сообщение и вернется, прежде чем отправить другие сообщения другим объектам в системе. Заголовок стрелки, используемый для отображения этого типа сообщения, представляет собой стрелку в виде строки, как показано в рисунке 2.8. ниже.

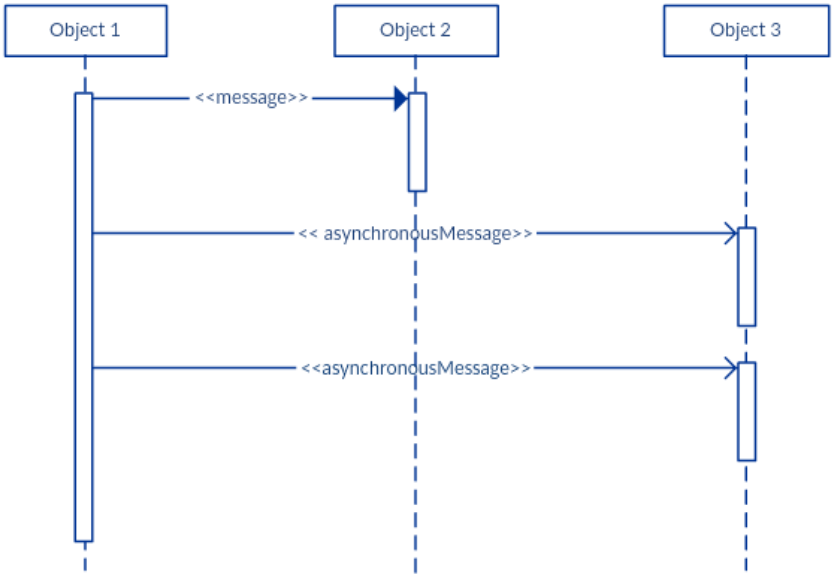


Рис. 2.8. - Пример асинхронного сообщения

Возвратное сообщение

Возвращаемое сообщение используется для указания на то, что приемник сообщения закончил обработку сообщения и возвращает управление вызывающему абоненту. Возвращаемые сообщения являются необязательными элементами нотации, так как строка активации, вызываемая синхронным сообщением, всегда подразумевает возвратное сообщение (см рис. 2.9.).

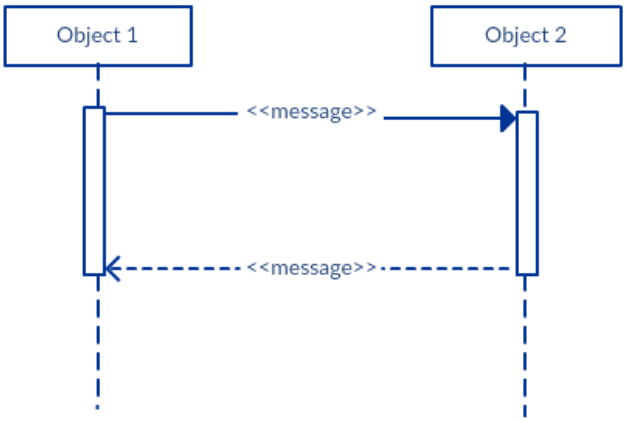


Рис. 2.9. - Пример возвратного сообщения

Сообщение о создании участника

Объекты не обязательно живут в течение всей последовательности событий. Объекты или участники могут быть созданы в соответствии с отправляемым сообщением.

Удаленную нотацию ящика участника можно использовать, когда нужно показать, что конкретного участника не существовало до тех пор, пока не был отправлен вызов на создание. Если созданный участник делает что-то сразу после создания, необходимо добавить окно активации прямо под окном участника (см рис. 2.10.).

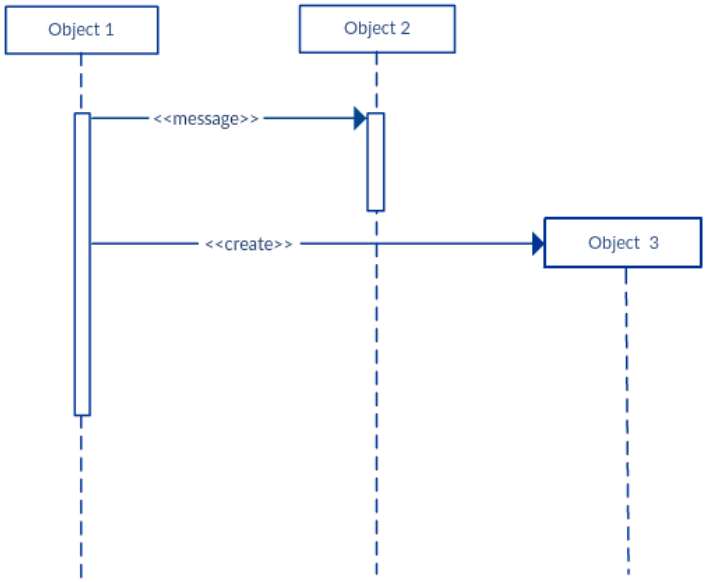


Рис. 2.10. - Пример сообщения о создании участника

Сообщение об уничтожении участника

Аналогичным образом, участники, которые больше не нужны, могут быть удалены из схемы последовательности. Это делается путем добавления буквы “X” в конце спасательной линии указанного участника (см рис. 2.11.) [].

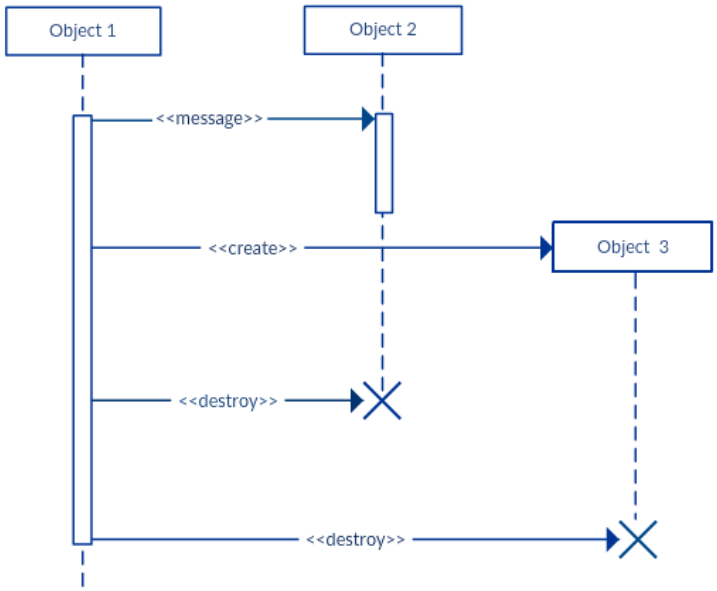


Рис. 2.11. - Пример сообщения об уничтожении участника

Рефлексивное сообщение

Когда объект посылает сообщение самому себе, он называется рефлексивным сообщением. Она обозначается стрелкой сообщения, которая начинается и заканчивается на одной и той же спасательной линии, как показано в рисунке 2.12. ниже.

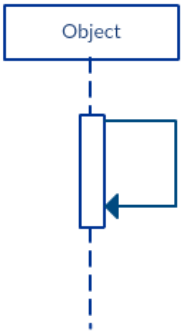


Рис. 2.12. - Пример рефлексивного сообщения

2. Анализ предметной области «Продажа авиационных билетов»

Процесс продажи авиабилетов – самая важная и трудоёмкая для ручной работы задача.

В лице посредника между авиакомпанией и клиентом может выступать частное агентство по продаже авиабилетов. Клиент либо сам на сайте ищет необходимый рейс и осуществляет покупку билета, либо обращается непосредственно в агентство, где ему подбирают рейс. К посредникам можно отнести агентства по продаже перевозок и туристические фирмы.

Авиакомпания предоставляет актуальную информацию о рейсах и ценах.

2.1. Диаграмма вариантов использования предметной области «Продажа авиационных билетов»

В результате анализа предметной области «Продажа авиационных билетов» была построена следующая диаграмма вариантов использования (рисунок 3).

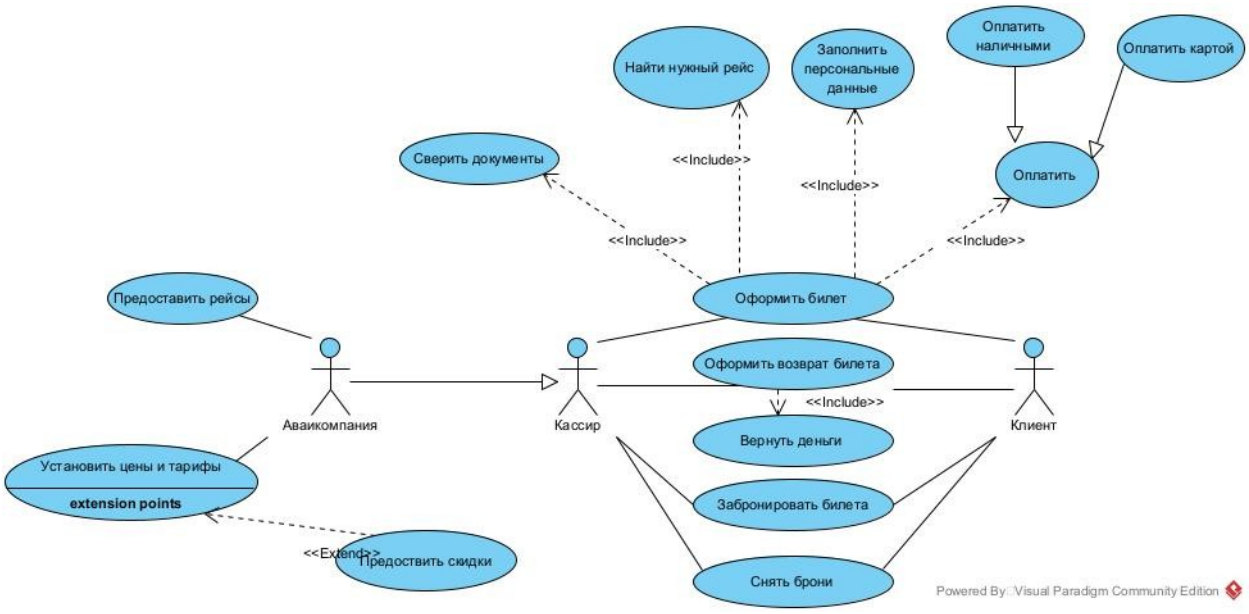


Рис. 3. - Диаграмма вариантов использования «Продажа авиационных билетов»

В предметной области «Продажа авиационных билетов» были выделены актеры «Кассир», «Авиакомпания» и «Клиент», а также варианты

использования «Предоставить рейсы», «Установить цены и тарифы», «Предоставить скидки», «Оформить билет», «Сверить документы», «Найти нужный рейс», «Заполнить персональные данные», «Оплатить», «Оформить возврат билета», «Вернуть деньги», «Забронировать билет», «Снять бронь». Были использованы следующие виды связи: ассоциация, включение, расширение и обобщение.

2.2. Диаграмма последовательности предметной области «Продажа авиационных билетов»

В результате анализа предметной области «Продажа авиационных билетов» была построена следующая диаграмма последовательности (рисунок 4).

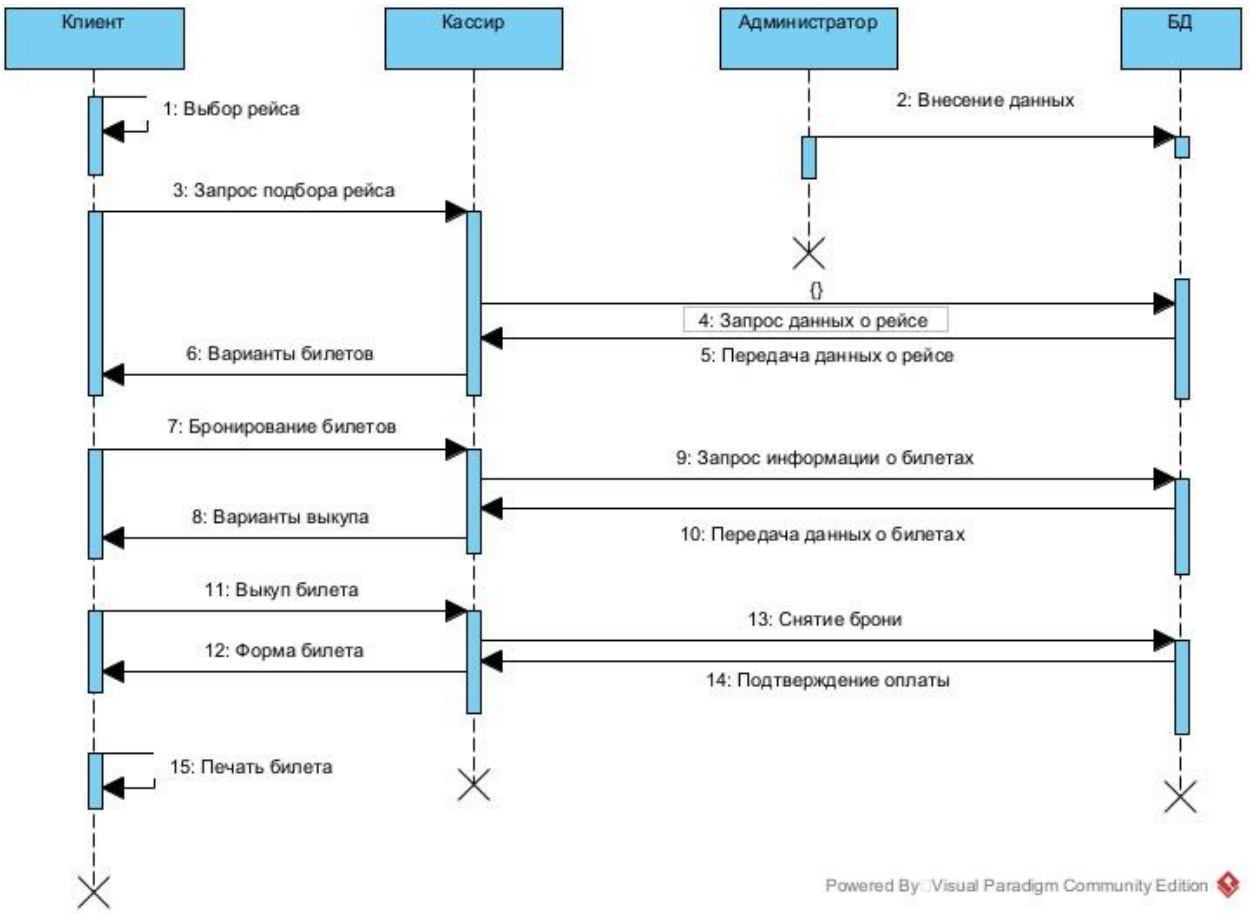


Рис. 4. - Диаграмма последовательностей «Продажа авиационных билетов»

В диаграмме были выделены следующие объекты: «Клиент», «Кассир», «Администратор», «БД» (База Данных). Были использованы стрелки синхронного сообщения и возврата.

3. Технология обработки текстовой информации

В процессе работы над оформлением отчета был использован текстовый редактор Microsoft Word. В нем были применены следующие стандарты.

При оформлении основного текста работы следует соблюдать следующие параметры: шрифт «Times New Roman», размер шрифта – 14, межстрочный интервал 1,5, в параметрах абзаца отступ слева – 0 см, отступ справа – 0 см, интервал перед – 0 пт, интервал после 0 пт.

Каждая глава должна начинаться с новой страницы. Названия глав, разделов и подразделов начинаются с абзацного отступа (красной строки), выравниваются по левому краю и оформляются полужирным шрифтом «Times New Roman» размер шрифта – 14, межстрочный интервал 1,5, в параметрах абзаца отступ слева – 0 см, отступ справа – 0 см, интервал перед – 10 пт, интервал после 10 пт. Точка после номера пункта и в конце заголовка не ставится.

Все ненумерованные заголовки («Введение», «Заключение», «Список использованных источников») не имеют абзацного отступа (красной строки), выравниваются по центру и оформляются полужирным шрифтом «Times New Roman» размер шрифта – 14, межстрочный интервал 1,5, в параметрах абзаца отступ слева – 0 см, отступ справа – 0 см, интервал перед – 10 пт, интервал после 10 пт.

Нумерация страниц должна размещаться в верхнем правом углу, за исключением страницы с рефератом, где номер располагается в основной рамке.

Перечисления в тексте оформляются с абзаца, причем в пределах всей работы необходимо придерживаться одного и того же маркера. Каждый

пункт перечисления заканчивается точкой с запятой, за исключением последнего пункта, который заканчивается точкой. Все пункты начинаются со строчных букв, т.к. является продолжением основного предложения. Не допускается

использование в перечислении сложных пунктов, состоящих из нескольких отдельных предложений.

Таблицы следует размещать сразу после ссылки на них в тексте. Таблицы последовательно нумеруются арабскими цифрами в пределах всей работы. Название таблиц размещается над левым верхним углом соответствующей таблицы, выравниваются по левому краю и оформляются шрифтом «Times New Roman» размер шрифта – 14, межстрочный интервал 1, в параметрах абзаца интервал перед – 15 пт, интервал после 10 пт.

Номер и наименование рисунка записываются через тире в строчку под его изображением без закрывающей точки. Подрисуночная надпись начинается без абзацного отступа (красной строки), выравнивается по центру и оформляется обычным шрифтом «Times New Roman» размер шрифта – 14, межстрочный интервал 1, в параметрах абзаца отступ слева – 0 см, отступ справа – 0 см, интервал перед – 10 пт, интервал после 15 пт.

Список литературы составляется в порядке упоминания литературных источников в работе. Для многотиражной литературы при составлении списка указываются: полное название источника, фамилия и инициалы автора, издательство и год выпуска (для статьи – название издания и его номер).

Приложения оформляются следующим образом:

- нумерация приложений – А, Б, В…;

- номер и название приложения на отдельном листе по центру как по вертикали, так и по горизонтали;

- на приложение в тексте должна быть ссылка.

Заключение

В данной работе мной были исследованы современные технологии объектно-ориентированного анализа и проектирования информационных систем. Изучены нотация и семантика языка UML. Сделан обзор особенностей применения данного языка в качестве средства объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

Также мной был проведен анализ предметной области **«Продажа авиационных билетов»**. Для заданной предметной области были построены диаграмма вариантов использования и диаграмма последовательности. Изучены информационные технологии обработки текстовой информации и технической документации. Сделан обзор ключевых возможностей и особенностей применения данной технологии в различных программных средах.

Получен сертификат открытого университета ИНТУИТ.

Список использованных источников

1. Основы унифицированного языка моделирования: [Электронный ресурс] // SITES GOOGLE. URL: https://www.sites.google.com/site/anisimovkhv/learning/pris/lecture/tema11. (Дата обращения 08.07.2021).

2. Диаграммы вариантов использования: [Электронный ресурс] // SITES GOOGLE. URL: https://www.sites.google.com/site/anisimovkhv/learning/pris/lecture/tema12/tema12_2. (Дата обращения 08.07.2021).

3. Учебное пособие по диаграммам последовательностей: полное руководство с примерами: [Электронный ресурс] // CREATELY. URL: https://creately.com/blog/ru/диаграмма/учебное-пособие-по-последовательной/#Что. (Дата обращения 08.07.2021).

4. История создания UML: [Электронный ресурс] // MAKSAKOV-SA. URL: http://www.maksakov-sa.ru/ModelUML/IstorUML/index.html. (Дата обращения 08.07.2021).

5. Особенности графического изображения диаграмм языка UML: [Электронный ресурс] // НОУ ИНТУИТ. URL: https://intuit.ru/studies/courses/32/32/lecture/1002?page=4. (Дата обращения 08.07.2021).

6. Канонические диаграммы языка UML: [Электронный ресурс] // LEKTSII. URL: https://lektsii.net/4-123079.html. (Дата обращения 08.07.2021).

Приложение А. Результаты прохождения обучающего курса


ИНТУИТ
НАЦИОНАЛЬНЫЙ ОТКРЫТЫЙ УНИВЕРСИТЕТ

Сертификат

Серия О

Регистрационный № 101542740

ВЫДАН

ЛЕКСИНОЙ АНАСТАСИИ ОЛЕГОВНЕ

В ТОМ, ЧТО ОНА С 17 ИЮНЯ 2022 ПО 1 ИЮЛЯ 2022 ПРОШЛА ОБУЧЕНИЕ
В НАЦИОНАЛЬНОМ ОТКРЫТОМ УНИВЕРСИТЕТЕ «ИНТУИТ» ПО КУРСУ
«ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА PYTHON»
В ОБЪЕМЕ 72 ЧАСОВ

ЕРАСИНА

ПО ЛЯ 2022 ГОДА

Рис. 5. - Сертификат № 101542740 по курсу «Введение в программирование на Python»